VLSI Design

Selected Papers from the Midwest Symposium on Circuits and Systems

Guest Editors: Gregory D. Peterson, Ethan Farquhar, and Benjamin Blalock

Selected Papers from the Midwest Symposium on Circuits and Systems

VLSI Design

Selected Papers from the Midwest Symposium on Circuits and Systems

Guest Editors: Gregory D. Peterson, Ethan Farquhar, and Benjamin Blalock

Editorial Board

Contents

Hindawi Publishing Corporation
VLSI Design
Volume 2010, Article ID 538454, 2 pages
doi:10.1155/2010/538454

Editorial

Selected Papers from the Midwest Symposium on Circuits and Systems

Gregory D. Peterson,[1] Ethan Farquhar,[2] and Benjamin Blalock[1]

[1] *Department of Electrical Engineering and Computer Science, University of Tennessee, Knoxville, TN 37996-2100, USA*
[2] *Oak Ridge National Laboratory, Oak Ridge, TN 37831-6006, USA*

Correspondence should be addressed to Gregory D. Peterson, gdp@utk.edu

Received 7 April 2010; Accepted 7 April 2010

This special issue of the VLSI Design journal is dedicated to the 51st IEEE International Midwest Symposium on Circuits and Systems (MWSCAS), held in August 2008 at Knoxville, Tennessee. The papers at MWSCAS spanned a broad spectrum of topics including theory, CAD, digital systems, and emerging technologies.

A number of quality papers in theory and CAD were submitted to this special issue. For example, nonlinear signal processing enables improved performance and power usage, but requires analog circuitry for practical systems. Odame and Hasler explore techniques for representing nonlinear circuit behavior in their paper *Nonlinear circuit analysis via perturbation methods and hardware prototyping*.

Zhu et al. explore a novel capacitor array structure with theoretic and simulation studies to develop a Successive Approximation Register for use with converters in *Linearity analysis on a series-split capacitor array for high-speed SAR ADCs*.

The paper from Reza Hashemanian presents a new approach for biasing analog circuits. The technique begins with localized optimization of nonlinear components at their operating points before turning to global optimization including the use of fixators and norators.

The paper *Error immune logic for low power probabilistic computing* from Marr et al. presents theorems related to constructing datapath circuits that yield an increased immunity to noise and can be applied to develop ultra-low power datapath circuits.

The paper by Yelamarthi and Chen presents an algorithm for timing optimization of CMOS dynamic logic along with a Path Oriented IN Time (POINT) optimization flow for mixed-static-dynamic CMOS logic.

Achieving timing closure looms as a major challenge in taping out a design. In *Post-CTS delay insertion*, Lu and Taskin explore techniques for improving clock performance by developing an optimization approach for implementing clock skew scheduling on circuits after clock tree synthesis is complete.

A number of digital circuits and systems were presented at MWSCAS. In *A low power digitally controlled oscillator for all digital phase locked loops*, Zhao and Kim present a low power and low jitter 12-bit CMOS digitally controlled oscillator. This DCO is a key component of their all-digital PLL that helps achieve faster acquisition times with low power and jitter.

Sumathi and Janakiraman present *FPGA implementation of an amplitude modulated continuous wave ultrasonic ranger using restructured phase locking scheme* that discusses a design for an accurate ultrasonic range finder with a restructured phase-locked loop employing a Sliding Discrete Fourier Transform.

Industrial applications of Reconfigurable computing to power systems exploit the FPGA's flexibility and performance. Selvajyothi and Janakiraman discuss their design for using an FPGA to control a single phase inverter.

In *Evolvable block-based neural network design for applications in dynamic environments,* Merchant and Peterson explore the use of reconfigurable computing to implement embedded neural network structures that are optimized during execution through the use of genetic algorithms.

Pseudorandom number generation represents one of the more expensive computational tasks commonly required for scientific computing, cryptography, simulation, and even gaming algorithms. Lee et al. present a hardware-accelerated

implementation of a scalable parallel library of pseudo-random number generators.

In *A pipelined and parallel architecture for quantum monte carlo simulations on FPGAs,* Gothandaraman et al. present a parameterized framework for supporting computational chemistry research on high performance reconfigurable computing platforms.

The MWSCAS papers also include emerging technologies and novel applications. Ulaganathan et al. discuss the implementation of an extreme environment circuit for data acquisition in *A SiGe BiCMOS instrumentation channel for extreme environment applications.*

The emerging technology of carbon nanotubes promise useful characteristics for devices or wires. Xu et al. explore these topics in *Emerging carbon nanotube electronic circuits, modeling and performance.*

Acknowledgments

The authors would like to conclude this Editorial by thanking our Editor in Chief, Dr. Saeid Nooshabadi for his support in the publication of this Special Issue. They would also like to express our gratitude for the excellent help and advice of the reviewers. They diligently read the submissions, provided invaluable feedback to the paper authors, and helped us to select the highest quality papers.

Gregory D. Peterson
Ethan Farquhar
Benjamin Blalock

Hindawi Publishing Corporation
VLSI Design
Volume 2010, Article ID 687498, 10 pages
doi:10.1155/2010/687498

Research Article

Nonlinear Circuit Analysis via Perturbation Methods and Hardware Prototyping

K. Odame[1] and P. E. Hasler[2]

[1] *Thayer School of Engineering, Dartmouth College, Hanover, 03755, USA*
[2] *School of Electrical and Computer Engineering, Georgia Institute of Technology, Atlanta, 30332-0250, USA*

Correspondence should be addressed to K. Odame, odame@dartmouth.edu

Received 3 June 2009; Accepted 8 December 2009

Academic Editor: Benjamin J. Blalock

Nonlinear signal processing is necessary in many emerging applications where form factor and power are at a premium. In order to make such complex computation feasible under these constraints, it is necessary to implement the signal processors as analog circuits. Since analog circuit design is largely based on a linear systems perspective, new tools are being introduced to circuit designers that allow them to understand and exploit circuit nonlinearity for useful processing. This paper discusses two such tools, which represent nonlinear circuit behavior in a graphical way, making it easy to develop a qualitative appreciation for the circuits under study.

1. Introduction

Portable and implantable, always-on electronics stand to benefit from analog signal processing, when only low levels of precision are necessary [1, 2]. To achieve sophisticated signal processing with low power and area overhead, an analog processor can exploit the fundamental nonlinear dynamics that are found in devices and simple circuits. So, the circuit designer must depart from the traditional linear systems paradigm, and learn to analyze and understand circuits from a nonlinear dynamical systems theory perspective.

In order to make nonlinear circuit design relevant to an engineer, it must be taught intuitively enough to foster creativity, yet rigorously enough to be of practical benefit. The two most popular tools for studying nonlinear circuits are harmonic balance and Volterra series. Within certain limitations, they are rigorous, but not necessarily intuitive. Since harmonic balance is simulation-based, it can be used to predict nonlinear behavior without ever requiring a deep understanding [3] of the circuit. Volterra series is an analytical tool that quickly leads to high entropy [4] mathematical expressions, from which the causative physical phenomena are hard to discern, much less purposefully manipulate.

To bridge the gap between rigor and intuition, we can use visual representation techniques. If the appropriate visualization is formed from rigorous definitions of a circuit's dynamics, then the human vision system, with its pattern recognition ability, will perceive the circuit's qualitative behavior [5].

We will present a filter block diagram for analyzing harmonic distortion that is derived from perturbation analysis. Unlike Volterra series kernels, our filter block diagram does not include multidimensional Fourier transforms, and so is accessible to an introductory-level engineering audience. In the second part of this paper, we will discuss the creation and use of phase plane plots of nonlinear circuits. We will describe how to rapidly create the phase plane plots with a reconfigurable hardware platform, instead of with a numerical simulator.

This paper is an expansion of the work presented in [6].

2. Regular Perturbation

Whenever designers want to get an analytical handle on the sources and causes of distortion, the most commonly-used tool is Volterra series analysis. If a problem is tractable using

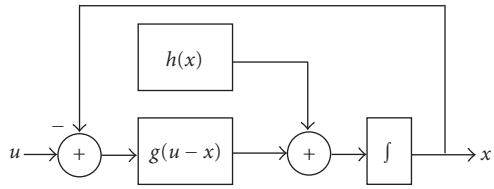


Figure 1: General block diagram form of a first-order circuit. The primary processing block is $g(\cdot)$, which is a nonlinear function of the input u and of x via feedback. The nonlinearity $h(x)$ models such nonidealities as finite output impedance.

Volterra series, then it can also be solved with perturbation theory, which will yield asymptotically-identical results [7].

There are certain problems for which Volterra series are ill-suited—multiple-time-scale behavior and multiple steady states, for instance [8]—that can be solved with perturbation theory. Despite the power of perturbation theory, it is still a relatively obscure concept in discussions about nonlinearity and distortion in analog circuits.

We therefore find it worthwhile to give a basic treatment of regular perturbation—the simplest perturbation method—as applied to distortion analysis of first-order analog circuits. In addition, a filter block diagram representation of the circuit will naturally evolve from our analysis, making it visually clear how the distortion terms are manifested, and how well-known tenets of low-distortion design, such as feedback, come about.

Consider the initial value problem

$$\dot{x} = f(t, x, \epsilon); \qquad x(t_0) = x_0(\epsilon), \tag{1}$$

where ϵ is a small perturbation parameter such that $\epsilon = 0$ yields an analytically-soluble equation. If f is sufficiently smooth (the specific smoothness requirements of f are discussed in [9]), then the problem has a unique solution $x(t, \epsilon)$. As the solution for $\epsilon \neq 0$ may not be analytical, it can be approximated as a power series in ϵ to an accuracy of $O(\epsilon^{n+1})$. That is, we can write the solution as

$$x(t, \epsilon) = \underbrace{\sum_{i=0}^{n} \left(x_i(t)\epsilon^i\right)}_{\hat{x}(t,\epsilon)} + O(\epsilon^{n+1}), \tag{2}$$

where $\hat{x}(t, \epsilon)$ is the approximate solution. To conduct regular perturbation, we apply the substitution $x(t, \epsilon) \approx \hat{x}(t, \epsilon)$ to (1). The resulting system is then solved by equating like powers of ϵ. The following sections will illuminate this idea.

3. The Basic First-Order Circuit

Most common first-order analog ciruits (simple amplifiers, buffers, switches, etc.) are of the form depicted in Figure 1. The governing equation is

$$\dot{x} = g(u - x) + h(x), \tag{3}$$

where u is the a.c. input signal, x is the a.c. output signal and $g(\cdot)$ and $h(\cdot)$ are nonlinear functions. The dependence of the system on the output, other than through feedback to the input, is modeled by $h(x)$. In practice, $h(x)$ is typically some nonideality such as finite output resistance.

In order to apply perturbation analysis to (3), we begin by assuming that the input signal has a small amplitude. This is expressed as $u = \epsilon v$, where ϵ is a small perturbation parameter and v is a suitably-scaled version of the input signal. Note that with the definition of u, (3) is solvable via separation of variables for the special case $\epsilon = 0$.

With the introduction of the perturbation parameter ϵ, we can approximate the solution to (3) with the power series

$$x(t) \approx \sum_{i=1}^{n} \epsilon^i x_i(t). \tag{4}$$

Note the ϵ^0 term of (4) is set to 0. This corresponds to analyzing a circuit about its d.c. bias point, where the d.c. bias point is shifted to the origin. For ease of notation, define $z = u - x$. The approximation of z is defined similarly to (4), with $z_1 = v - x_1$ and $z_i = -x_i$, for all $i > 1$.

If ϵ is sufficiently small, then the functions $g(z)$ and $h(x)$ can be approximated by their truncated Taylor series as

$$\begin{aligned} g(z) &\approx g_1 z + g_{n-1} z^{n-1} + g_n z^n, \\ h(x) &\approx h_1 x + h_{n-1} x^{n-1} + h_n x^n. \end{aligned} \tag{5}$$

Functions g and h are assumed to be dominantly $(n-1)$th-order nonlinearities, with $g_i = g^{(i)}(0)/i!$ and $h_i = h^{(i)}(0)/i!$. Equation (5) assumes $g(0) = h(0) = 0$, which, again, corresponds to analyzing a circuit about its d.c. bias point.

Substituting (4) and (5) into (3) and collecting powers of ϵ, we get the following set of first-order *linear* equations

$$\begin{aligned} &\dot{x}_1 + (g_1 - h_1)x_1 = g_1 v, \\ &\qquad\vdots \\ &\dot{x}_k + (g_1 - h_1)x_k = 0 \quad \forall k < n-1 \\ &\qquad\vdots \\ &\dot{x}_{n-1} + (g_1 - h_1)x_{n-1} = g_{n-1} z_1^{n-1} + h_{n-1} x_1^{n-1} \\ &\dot{x}_n + (g_1 - h_1)x_n = g_n z_1^n - n g_{n-1} z_1^{n-1} x_2 \\ &\qquad\qquad + h_n x_1^n - n h_{n-1} x_1^{n-1} x_2. \end{aligned} \tag{6}$$

The ϵ^1 equation is the linearized portion of (3) with input v. Taking the Laplace transform of this equation, we write

$$X_1(s) = g_1 H(s) V(s), \tag{7}$$

where $H(s) = 1/(s + g_1 - h_1)$.

The ϵ^k equations $(k < (n-1))$ are filters with 0 input. As such, the steady state solutions of these equations is 0.

4. Harmonic Distortion Terms

The inputs of the ϵ^{n-1} equation are terms of z_1^{n-1} and x_1^{n-1}. To understand the implications of these terms to harmonic

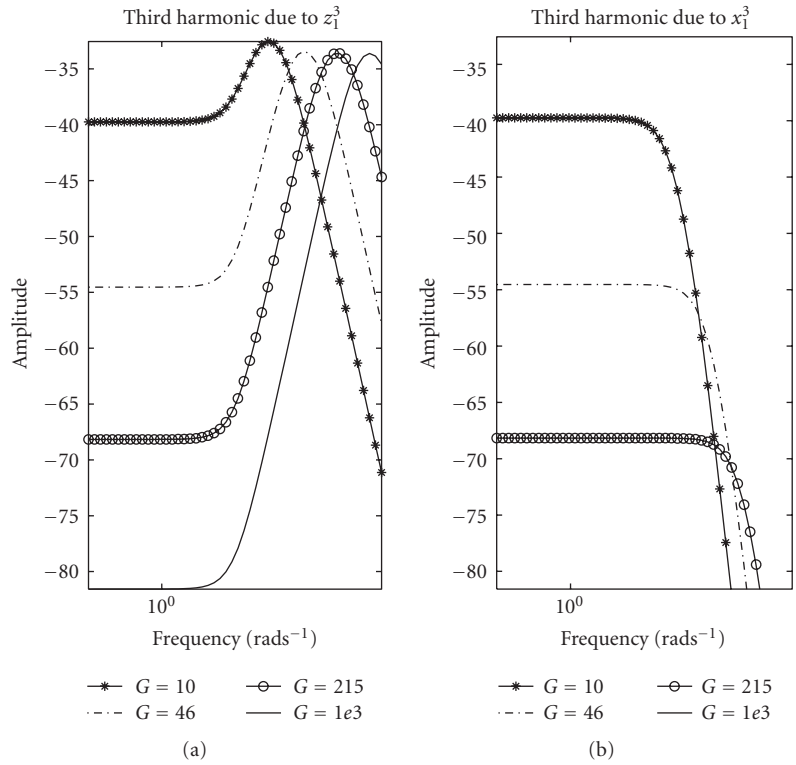


Figure 2: Magnitude-frequency plots of the third harmonic. The "gain", G of the $g(z)$ function is varied from 10 to 1000. This causes the band-pass shape of the z_1^3-contributed harmonic to shift to the right, while that contributed by x_1^3 falls in magnitude.

distortion, assume a single-tone input, $v = \cos(\omega t)$. This elicits the signals

$$\begin{aligned} x_1 &= g_1 \left| H(j\omega) \right| \cos(\omega t + \phi(j\omega)), \\ z_1 &= \left| 1 - g_1 H(j\omega) \right| \cos(\omega t + \phi_{z1}(jw)), \\ &= \left| \underbrace{(s - h_1) H(j\omega)}_{H_{z1}(j\omega)} \right| \cos(\omega t + \phi_{z1}(jw)), \end{aligned} \tag{8}$$

Here we have defined $H_{z1}(s) = (1 - g_1 H(s))$. The phases $\phi(s)$ and $\phi_{z1}(s)$ are the arguments of $H(s)$ and $H_{z1}(s)$, respectively. The signals x_1 and z_1 are single tones of frequency ω as well, since they are merely linearly-filtered versions of v.

Raising z_1 and x_1 each to the $(n-1)$th power produces harmonics as follows. If $(n-1)$ is odd(even), then odd(even) harmonics up to the $(n-1)$th harmonic are generated. The amplitude of the $m\omega$ frequency term in x_1^{n-1} is

$$\frac{(n-1)!g_1}{((n+m-1)/2)!((n-m-1)/2)!2^{n-2}} \left| H(j\omega) \right|, \tag{9}$$

while that of the $m\omega$ frequency term in z_1^{n-1} is

$$\frac{(n-1)!}{((n+m-1)/2)!((n-m-1)/2)!2^{n-2}} \left| H_{z1}(j\omega) \right|. \tag{10}$$

After filtering in the ϵ^{n-1} equation, the amplitudes of these terms will be, respectively,

$$\frac{(n-1)!h_{n-1}g_1}{((n+m-1)/2)!((n-m-1)/2)!2^{n-2}} \left| H(j\omega) \right| \left| H(jm\omega) \right|, \tag{11}$$

$$\frac{(n-1)!g_{n-1}}{((n+m-1)/2)!((n-m-1)/2)!2^{n-2}} \left| H_{z1}(j\omega) \right| \left| H(jm\omega) \right|. \tag{12}$$

Analogous to that of the ϵ^{n-1} equation, the input to the ϵ^n equation has terms in z_1^n and x_1^n. In general, the x_2 terms are identically zero, except for the special case $n = 3$.

5. Feedback and Distortion

We now make some observations about the harmonic distortion results that were discussed in the previous section.

In the ϵ^{n-1} equation, the amplitude of the mth harmonic that the z_1^{n-1} term contributes is given by (12). We plot this amplitude expression, along with that of (11), as a function of frequency in Figure 2 for the third-order harmonic generated by a dominantly-third order nonlinearity. That is, $n = 4$ and $m = 3$. Also, we chose $h_1 = 1$, $h_3 = 1/3$, $g_1 = G$, $g_3 = G/3$, where G was varied from 10 to 1000.

Notice from the figure that if $g_1 \gg h_1$, then, for a given frequency, the amplitude of the z_1^{n-1}-contributed harmonic is greatly reduced. In fact, if we ensure $g_i \gg h_i$ for all i, then the

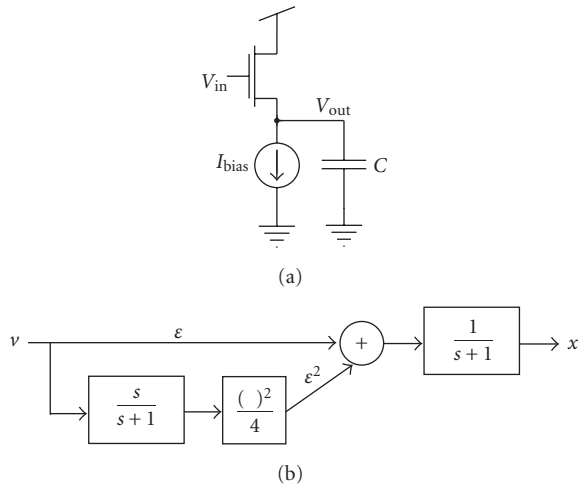


Figure 3: Source follower amplifier. (a) Circuit schematic. (b) Block diagram representation of source follower output. The fundamental harmonic is a low-pass filtered version of the input. The second order terms are generated by high-pass filtering the input, squaring and then low pass filtering. The total output is a power series of ϵ terms.

harmonic contribution of the x_1^{n-1} terms is negligible. This would mean that the distortion is effectively due only to z_1, whose associated harmonics are band-pass filtered. This in turn means that the distortion can be kept small if the circuit is operated well below the corner frequency.

These two notions—that frequency and feedback gain can be sacrificed for higher linearity—conform with the traditional rules-of-thumb for low-distortion design.

6. Illustrative Examples

6.1. Source Follower Amplifier. According to KCL, the circuit equation of the source follower amplifier in Figure 3(a) is

$$C\frac{dV_{\text{out}}(t)}{dt} = F(V_{\text{in}}, V_{\text{out}}) - I_{\text{bias}}, \tag{13}$$

where the function F is defined as

$$F(V_{\text{in}}, V_{\text{out}}) = \frac{K}{2}(\kappa V_{\text{in}}(t) - V_{\text{out}}(t) - V_{\text{th}})^2, \tag{14}$$

if M_1 is in above-threshold saturation, and

$$F(V_{\text{in}}, V_{\text{out}}) = I_{\text{o}} e^{(\kappa V_{\text{in}}(t) - V_{\text{out}}(t))/U_{\text{T}}}, \tag{15}$$

if it is in subthreshold saturation. The parameter K depends on transistor dimensions and doping and V_{th} is the threshold voltage. Also, κ, I_{o}, and U_{T} have their usual meanings from the EKV MOSFET model [10].

Note that $I_{\text{bias}} = F(V_{\text{g}}, V_{\text{S}})$, where V_{g} and V_{S} are the d.c. bias-points of the gate and source of M_1, respectively. Let us define a *characteristic voltage*, V_{c}, as

$$V_{\text{c}} = \begin{cases} \dfrac{\left(\kappa V_{\text{g}} - V_{\text{S}} - V_{\text{th}}\right)}{2}, & \text{above threshold} \\ U_{\text{T}}, & \text{subthreshold.} \end{cases} \tag{16}$$

Now, (13) can be nondimensionalized [9] by making the substitutions

$$\tau = \frac{I_{\text{bias}}}{(CV_{\text{c}})} \cdot t; \quad u = \frac{\kappa v_{\text{in}}}{V_{\text{c}}}; \quad x = \frac{v_{\text{out}}}{V_{\text{c}}}, \tag{17}$$

where v_{in} and v_{out} are the a.c. portions of V_{in} and V_{out}. This gives the state-space equation of the source follower as

$$\frac{dx}{d\tau} = u - x + \frac{(u-x)^2}{4}, \tag{18}$$

for above threshold, and

$$\frac{dx}{d\tau} = u - x + \frac{(u-x)^2}{2}, \tag{19}$$

for the truncated Taylor expansion in subthreshold. The point is that, regardless of region of operation of M_1, the nonlinear equation that describes the source follower has the same functional form. Relating the source follower equations to (3), we have $g(z) \sim z + z^2$ and $h(x) = 0$. As such, we expect the harmonic distortion terms to have a band-pass-like dependence on frequency. To show this, we will apply regular perturbation to (18).

First, define $u = \epsilon v$, where the small parameter ϵ is a scaled version of the input amplitude. That is, $\epsilon = A_{\text{in}}/V_{\text{c}}$. Also, taking $x = \epsilon x_1 + \epsilon^2 x_2$ and $z = u - x$ and equating like powers of ϵ up to ϵ^2, we have

$$\epsilon^1 : \dot{x}_1 = v - x_1, \tag{20}$$

$$\epsilon^2 : \dot{x}_2 = \frac{z_1^2}{4} - x_2, \tag{21}$$

as depicted in Figure 3(b). Assume a pure-tone input, $v = \cos(\omega t)$. Equation (20) is the linear portion of the amplifier. Equation (21) is a linear filter with input $z_1^2/4$. The squaring produces a second-harmonic term as well as a d.c. offset. In addition, since $z_1 = v - x_1$, the second harmonic generated by the squaring is high-pass filtered. The overall effect is that x_2 is a band-pass filtered version of a second harmonic of v. Figure 4 is a plot of experimental data that corroborates our analysis. There is error in the second harmonic measurement due to the small amplitudes involved.

6.2. Unity Gain Buffer. Consider the unity-gain buffer depicted in Figure 5(a). It is formed by placing an operational transconductance amplifier (OTA) in negative feedback. If we operate the OTA above threshold, the describing equation is

$$C\frac{dV_{\text{out}}}{dt} = \sqrt{\kappa\beta I_{\text{bias}}}(V_{\text{in}} - V_{\text{out}})\sqrt{1 - \frac{\kappa\beta(V_{\text{in}} - V_{\text{out}})^2}{4I_{\text{bias}}}}, \tag{22}$$

while it is

$$C\frac{dV_{\text{out}}}{dt} = I_{\text{bias}}\tanh\left(\frac{\kappa(V_{\text{in}} - V_{\text{out}})}{2U_{\text{T}}}\right), \tag{23}$$

for subthreshold operation. Notice that we have ignored the output conductance term, which is considered very small for OTAs.

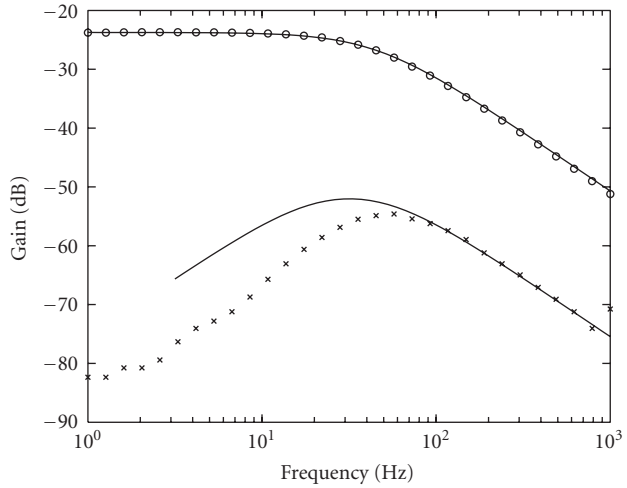


Figure 4: Magnitude-frequency response of source follower. Analytical prediction is in bold, and experimental data is plotted as "x"s and "o"s. The fundamental harmonic is a low-pass filtered version of the input. The second harmonic has a bandpass shape, as predicted by perturbation analysis.

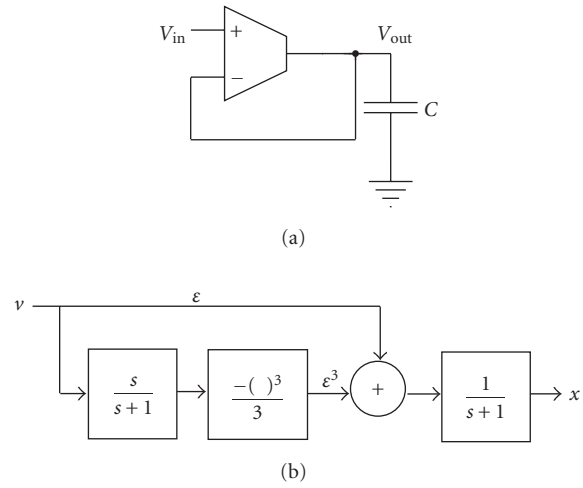


Figure 5: Unity gain buffer. (a) Circuit schematic. (b) Block diagram representation of output. The fundamental harmonic is a low-pass filtered version of the input. The third-order terms are generated by high-pass filtering the input, cubing and then low pass filtering. The total output is a power series of ϵ terms.

We can define a characteristic voltage, V_c, as

$$V_c = \begin{cases} \dfrac{2U_T}{\kappa}, & \text{subthreshold} \\ \sqrt{\dfrac{I_{bias}}{\kappa\beta}}, & \text{above threshold.} \end{cases} \tag{24}$$

Then, with the following definitions

$$\tau = \frac{I_{bias}}{(CV_c)} \cdot t; \quad u = \frac{v_{in}}{V_c}; \quad x = \frac{v_{out}}{V_c}, \tag{25}$$

the nondimensional form of the unity-gain buffer's describing equations (taken to the first few Taylor series terms) is

$$\frac{dx}{d\tau} = \begin{cases} (u-x) - \dfrac{(u-x)^3}{4}, & \text{above threshold} \\ (u-x) - \dfrac{(u-x)^3}{3}, & \text{subthreshold.} \end{cases} \tag{26}$$

Again, the functional form of the equations is identical, regardless of region of operation.

To calculate distortion terms, assume $u = \epsilon v$ is a pure-tone signal and proceed as usual. For subthreshold, the separated equations of ϵ are

$$\epsilon^1 : \dot{x}_1 = v - x_1, \tag{27}$$

$$\epsilon^2 : \dot{x}_2 = 0 - x_2, \tag{28}$$

$$\epsilon^3 : \dot{x}_3 = \frac{z_1^3}{3} - x_3. \tag{29}$$

These equations are depicted in the block diagram of Figure 5(b). Equation (27) is the linear portion of the amplifier. Equation (28) is a linear filter with 0 input; it contributes no harmonics at steady state. Equation (29) is a linear filter with input $z_1^3/3$. The cubing produces a third-harmonic term as well as a fundamental-frequency term (this fundamental-frequency term will cause gain compression, which is not discussed in this paper). Since $z_1 = v - x_1$, the overall effect is that x_3 is a band-pass filtered version of a third harmonic of v, as shown in Figure 6.

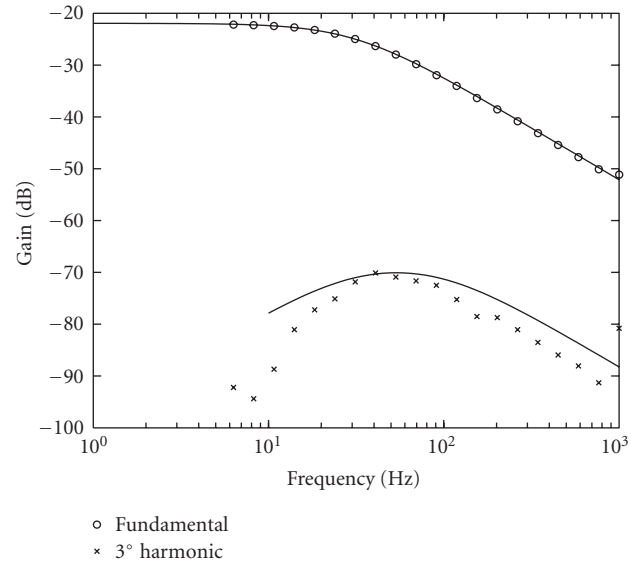


Figure 6: Magnitude-frequency response of unity-gain buffer. Analytical prediction is in bold, and experimental data is plotted as "x"s and "o"s. The fundamental harmonic is a low-pass filtered version of the input. The third harmonic has a bandpass shape, as predicted by perturbation analysis.

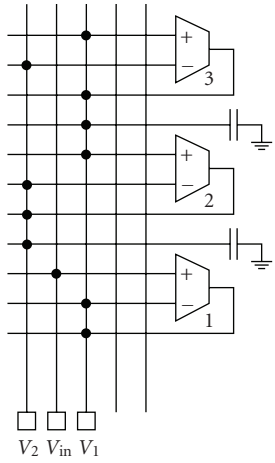


Figure 7: Compiled Circuit on an FPAA. Component terminals can be connected or disconnected using the switch matrix. In this illustration, connections are indicated by solder dots. The gain of each amplifier is programable over a continuous range of values over a few decades.

6.3. Note on Above-Threshold versus Subthreshold Operation. The harmonic behavior of a circuit is similar for above- and subthreshold operation. In absolute numbers, however, above threshold operation yields less distortion. This is because the parameter $\epsilon = A_{\text{in}}/V_{\text{c}}$ is much smaller for above threshold than for subthreshold. Since the harmonics are multiplied by ϵ^i, the smaller ϵ seen in above threshold operation translates to lower distortion.

7. Field Programmable Analog Array

We have developed a field programmable analog array (FPAA) that can be configured to synthesis and analyze a vast variety of circuits [11]. Figure 7 depicts a simple, second-order filter compiled on our FPAA. More complex circuit configurations are possible, and would involve a larger number of the over 400 components in the FPAA. In this part of the paper we demonstrate the utility and versatility of the FPAA in analyzing the dynamics of a number of fundamental circuit blocks.

8. One-Dimensional Systems

8.1. Simple Current Mirror. Consider the simple current mirror depicted in Figure 8(a). From Kirchhoff's Current Law (KCL), it obeys the following differential equation:

$$C\frac{dV_{\text{g}}}{dt} = I_{\text{b}} - f\left(V_{\text{g}}\right), \tag{30}$$

where $f(V_{\text{g}})$ is the drain current of transistor M1. Assuming M1 and M2 are identical and are both saturated, we have $f(V_{\text{g}}) = I_{\text{out}}$, which gives

$$C\frac{dV_{\text{g}}}{dt} = I_{\text{b}} - I_{\text{out}}. \tag{31}$$

For subthreshold operation in saturation, the current through transistors M1 and M2 is [10]

$$f\left(V_{\text{g}}\right) = I_{\text{out}} = I_{\text{o}}e^{(\kappa V_{\text{g}} - V_{\text{S}})/U_{\text{T}}}, \tag{32}$$

where I_{o} is a pre-exponential constant dependent on the transistor's size and on doping concentrations. Also, κ is the body-effect coefficient and U_{T} is the thermal voltage. V_{S} is the source voltage, which, for this case, is zero. Setting $V_{\text{S}} = 0$ and taking the derivative of (32) with respect to time, we get

$$\frac{dI_{\text{out}}}{dt} = \frac{\partial}{\partial V_{\text{g}}}\left(I_{\text{o}}e^{\kappa V_{\text{g}}/U_{\text{T}}}\right)\frac{dV_{\text{g}}}{dt}, \tag{33}$$

$$= \frac{\kappa}{U_{\text{T}}}I_{\text{out}}\frac{dV_{\text{g}}}{dt}, \tag{34}$$

which allows us to rewrite (31) as

$$\begin{aligned}\frac{CU_{\text{T}}}{\kappa I_{\text{b}}}\frac{dI_{\text{out}}}{dt} &= I_{\text{out}}\left(1 - \frac{I_{\text{out}}}{I_{\text{b}}}\right),\\ \tau\frac{dI_{\text{out}}}{dt} &= I_{\text{out}}\left(1 - \frac{I_{\text{out}}}{I_{\text{b}}}\right).\end{aligned} \tag{35}$$

The time constant is identified as $\tau = (CU_{\text{T}})/(\kappa I_{\text{b}})$.

Equation (35) happens to be the *logistic equation*, a simple model of population dynamics. It can be solved exactly either by separation of variables followed by partial fractions, or by solving it as Bernoulli's equation. The solution is

$$I_{\text{out}}(t) = \frac{I_{\text{b}}e^{t/\tau}}{e^{t/\tau} - 1 + I_{\text{b}}/I_{\text{out}_0}}, \tag{36}$$

where I_{out_0} is the initial value of I_{out}. We are lucky to have an exact solution to (35), given that it is a nonlinear differential equation. Even so, it is difficult to discern much useful information about I_{out}'s qualitative behavior from (36). For instance, it is not clear how the behavior of I_{out} might change with different initial conditions. To answer questions of this sort, it is helpful to do geometric analysis on the system's corresponding vector field.

Since the simple current mirror is a one-dimensional system, its vector field is represented as a flow on a line. The direction and speed of the flow are dictated by the right hand side (RHS) of (35). It is a quadratic, as shown in Figure 10. The I_{out}-intercepts are 0 and I_{b}. There is a maximum at $I_{\text{out}} = I_{\text{b}}/2$. The vector field is depicted as the arrows on the I_{out} axis. For positive values of dI_{out}/dt, I_{out} is increasing, meaning the arrows point to the right. For negative values of dI_{out}/dt, I_{out} is decreasing, meaning the arrows point to the left. When $dI_{\text{out}}/dt = 0$, there is no change in I_{out} and the circuit is said to be at equilibrium.

The vector field provides clear, qualitative information about the behavior of I_{out}. There are two equilibrium points, namely $I_{\text{out}} = 0$ and $I_{\text{out}} = I_{\text{b}}$. Note that the vector field flows away from $I_{\text{out}} = 0$. This equilibrium point is unstable, since the system will not recover from slight disturbances away from it. The vector field flows towards $I_{\text{out}} = I_{\text{b}}$, implying

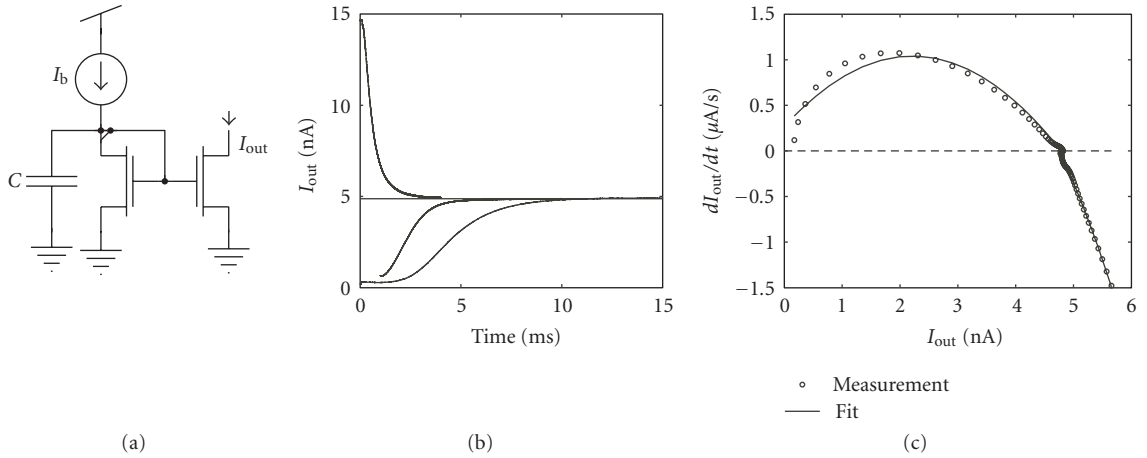


Figure 8: Simple current mirror. (a) Circuit that was compiled onto the FPAA. (b) Measured trajectories for different initial conditions. (c) Vector field derived from trajectory measurements. The origin is an unstable equilibrium point, while 5 is stable.

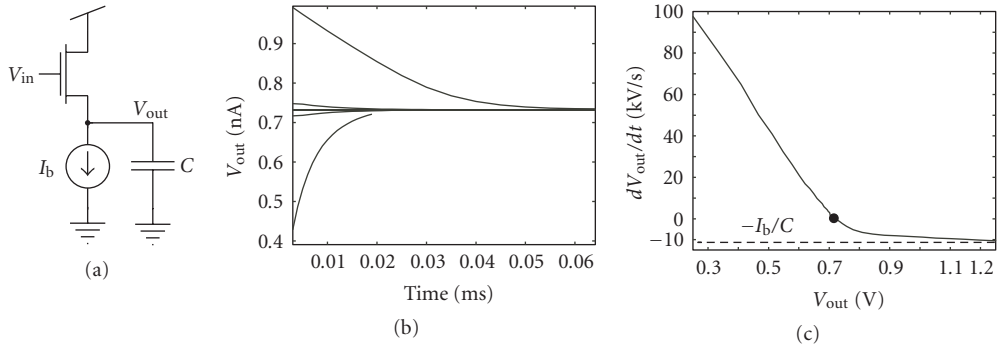


Figure 9: Source Follower Amplifier Acting as a Simple Peak Detector. (a) Circuit that was compiled onto the FPAA. (b) Measured step responses. (c) Vector field derived from step response measurements. The point $(0, 0.7)$ is a stable fixed point.

that this is a stable equilibrium point. If the system is initially at $I_{out} = I_b$ and then experiences a small disturbance, it will tend back to the $I_{out} = I_b$ point.

The vector field in Figure 10 also gives information about the acceleration of I_{out} as it approaches the $I_{out} = I_b$ equilibrium point. For $0 < I_{out} < I_b/2$, the rate of change of I_{out} increases until it reaches a peak at $I_{out} = I_b/2$. Between $I_b/2$ and I_b, the system decelerates until the rate of change of I_{out} eventually becomes zero. For $I_{out} > I_b$, the rate of change of I_{out} steadily decreases until $I_{out} = I_b$. It is interesting to note that, for $I_{out} < I_b$, the rate of change of I_{out} is limited to a maximum of $I_b/(4\tau)$.

The geometric analysis predictions can be checked against experimental measurements of a current mirror that was compiled onto an FPAA. Figure 8(b) depicts various trajectories, or solutions, of the system of (35) for different initial conditions. Notice that trajectories that start at values lower than $I_{out} = I_b/2$ have a sigmoidal shape, with the point of inflection corresponding to the maximum rate of change of current $dI_{out}/dt = I_b/(4\tau)$. The parabolic shape of dI_{out}/dt can be extracted from these trajectories, and it is shown in Figure 8(c).

8.2. Simple Peak Detector. Assuming subthreshold operation, the KCL equation for the source follower amplifier of Figure 9(a) is the following.

$$C\frac{dV_{out}}{dt} = I_o e^{(\kappa V_{in} - V_{out})/U_T} - I_b. \tag{37}$$

Note that

$$\frac{d}{dt}e^{V_{out}/U_T} = \frac{e^{V_{out}/U_T}}{U_T}\frac{dV_{out}}{dt}, \tag{38}$$

in which case, the solution to (37) is

$$V_{out} = \kappa V_{in} + U_T \log\left(\frac{I_o}{I_b} - \left(\frac{I_o}{I_b} - e^{(V_{out_0} - \kappa V_{in})/U_T}\right)e^{-t/\tau}\right), \tag{39}$$

where $\tau = CU_T/I_b$ and V_{out_0} is the initial value of V_{out}.

The time that it takes for V_{out} to be within 10% of its final value is

$$t_{10} = \tau \log\left|\frac{I_o/I_b - e^{(V_{out_0} - \kappa V_{in})/U_T}}{I_o/I_b - e^{0.1\kappa V_{in}/U_T}}\right|. \tag{40}$$

For a large positive step input, $e^{(V_{\text{out}_0}-\kappa V_{\text{in}})} \approx 0$, and (40) is approximately

$$t_{10} = t_{10+} \approx \tau \log\left|\frac{I_o}{I_o - I_b e^{0.1\kappa V_{\text{in}}/U_T}}\right|. \tag{41}$$

For a large negative step input, $e^{(V_{\text{out}_0}-\kappa V_{\text{in}})} \gg I_o/I_b$, and (40) becomes

$$\begin{aligned} t_{10} = t_{10-} &\approx \tau \log\left|\frac{I_b e^{(V_{\text{out}_0}-\kappa V_{\text{in}})}}{I_o - I_b e^{0.1\kappa V_{\text{in}}/U_T}}\right| \\ &= t_{10+} + \tau\left(\frac{V_{\text{out}_0}}{U_T} - \kappa\frac{V_{\text{in}}}{U_T}\right)\log\left(\frac{I_b}{I_o}\right). \end{aligned} \tag{42}$$

Equations (41) and (42) indicate that the response of the peak detector is slower for a negative input step than it is for a positive input step. We surmise that if the input is continuously varying at a rate faster than $1/(t_{10-})$, then the output will be a reasonable representation of the input's peak values. Explaining the peak detector's behavior with (41) and (42) is rigorous, but depends on having to manipulate the expression of (39).

One way of avoiding the math is to employ intuitive descriptions of the charging action of the active device (i.e., the transistor) versus the discharging action of the current source [12]. A more rigorous approach is to apply nonlinear geometric analysis to the problem. Consider the plot of dV_{out}/dt versus V_{out} shown in Figure 9(c). We constructed it from a number of step response measurements (Figure 9(b)) that we took after compiling the source-follower amplifier onto the FPAA. A large negative input step corresponds to an initial value of $V_{\text{out}_0} \gg V_{\text{in}}$. The rate of growth of V_{out} is bounded by I_b/C. For a large positive input step, however, $V_{\text{out}_0} \ll V_{\text{in}}$, and the maximum rate at which V_{out} approaches V_{in} can be much greater than I_b/C. The maximum rate of approach in this case is limited only by the initial value, V_{out_0}. As such, there is an asymmetry in the speed of the circuit's response to up-going versus down-going movements on the input. The effect of this asymmetry is that V_{out} tracks increasing V_{in} and not decreasing V_{in}, which is the behavior of a peak detector.

9. A Two-Dimensional System

Figure 11 depicts Lyon and Mead's classic second-order section [13]. It is a Gm-C filter with two poles that can be placed anywhere on the real/imaginary plane. We begin our analysis by writing down the governing equations for the circuit, assuming that the OTAs are based on subthreshold MOS transistor differential pairs:

$$\begin{aligned} C_1\frac{dV_2}{dt} &= \frac{I_2}{k}\tanh\left(\frac{\kappa(V_1 - V_2)}{2U_T}\right), \\ C_1\frac{dV_1}{dt} &= I_1\tanh\left(\frac{\kappa(V_{\text{in}} - V_1)}{2U_T}\right) - I_3\tanh\left(\frac{\kappa(V_2 - V_1)}{2U_T}\right), \end{aligned} \tag{43}$$

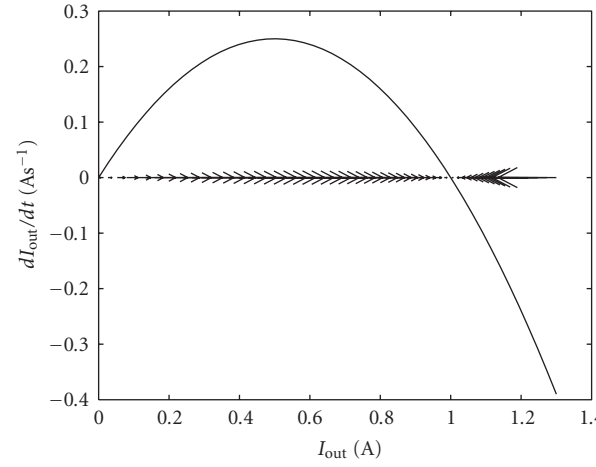


Figure 10: The vector field of the logistic equation is represented as a flow on the I_{out} axis. For positive values of dI_{out}/dt, I_{out} is increasing and the flow is to the right. For negative values of dI_{out}/dt, I_{out} is decreasing and the flow is to the left.

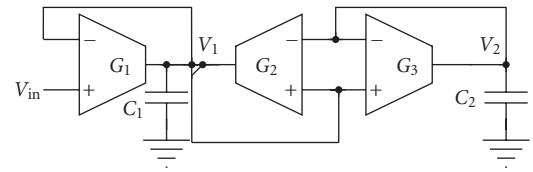


Figure 11: Second-Order Section. Varying the bias currents of the various amplifiers leads to interesting dynamics.

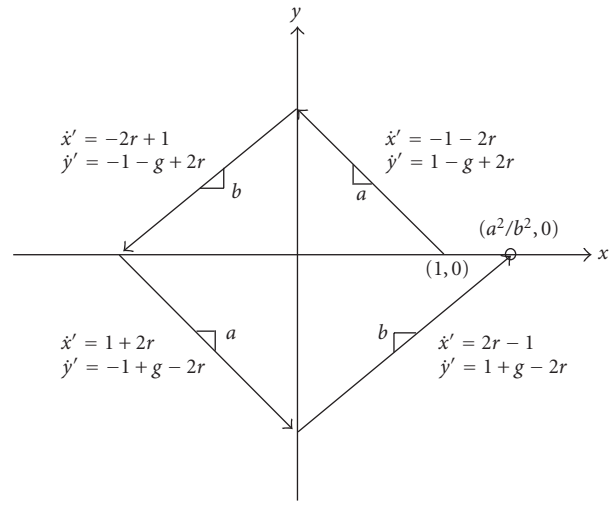


Figure 12: Sketch of SOS Phase Plane for Large Signals.

where $I_{1,2,3}$ are the bias currents of the OTAs. Also, k is the ratio of the C_2 to C_1.

If we define

$$x = \frac{\kappa(V_1 - V_{\text{in}})}{2U_T}, \qquad y = \frac{\kappa(V_2 - V_1)}{2U_T}, \tag{44}$$

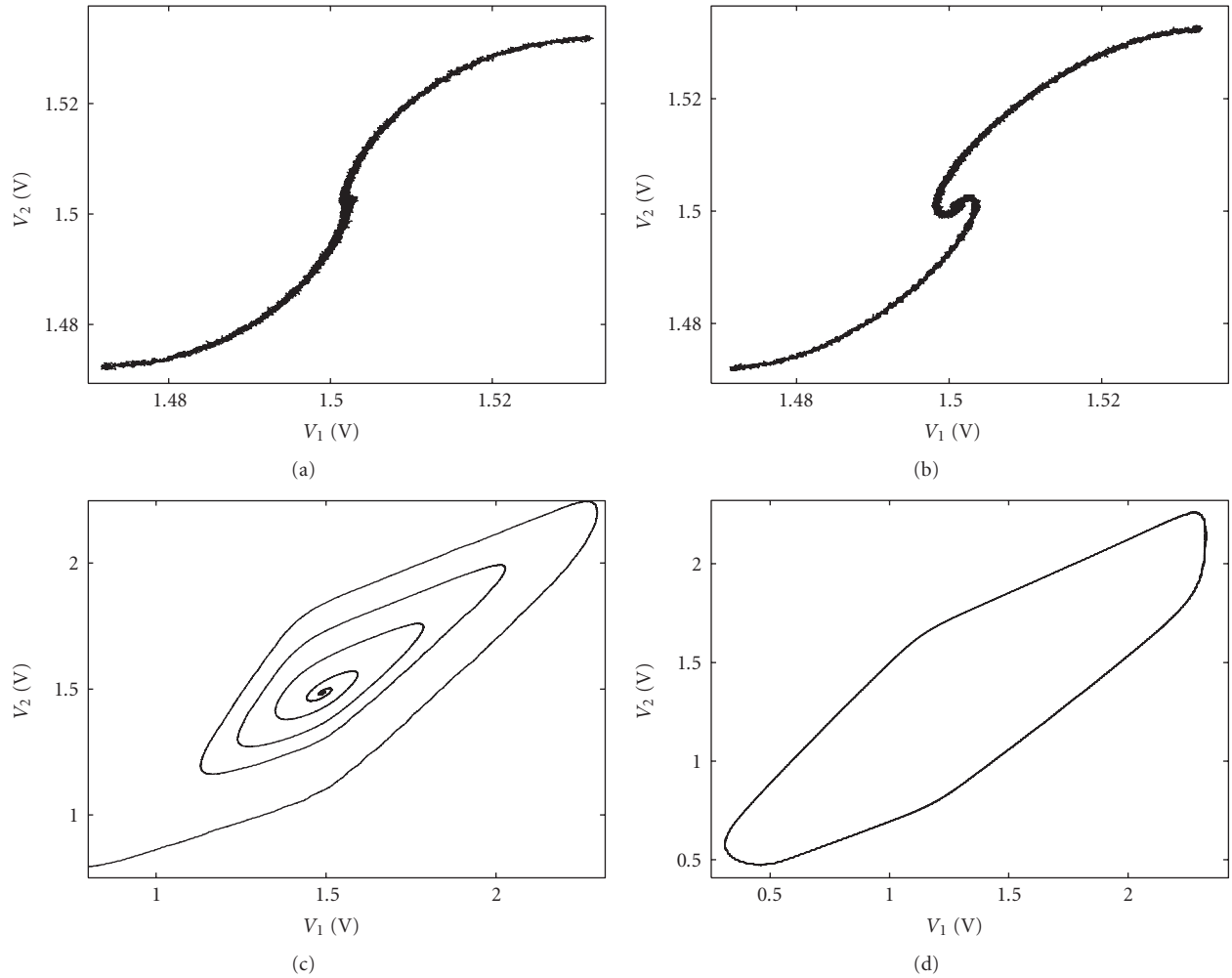


Figure 13: SOS Experimental Phase Plane Results for Various Values of r.

then (43) become

$$\frac{2U_T C_1}{\kappa}\frac{dx}{dt} = -I_1\tanh(x) - I_3\tanh(y),$$
$$\frac{2U_T C_1}{\kappa}\frac{dy}{dt} = I_1\tanh(x) + \left(I_3 - \frac{I_2}{k}\right)\tanh(y). \tag{45}$$

Further defining

$$I_1 = I_{\text{bias}},$$
$$I_2 = gkI_{\text{bias}},$$
$$I_3 = 2rI_{\text{bias}}, \tag{46}$$
$$t = \tau\cdot\frac{2U_T C_1}{\kappa I_{\text{bias}}},$$

where $g \geq 0$, we get the following dimensionless equation

$$\frac{dx}{d\tau} = -\tanh(x) - 2r\tanh(y),$$
$$\frac{dy}{d\tau} = \tanh(x) + (2r-g)\tanh(y). \tag{47}$$

9.1. Small Signal Analysis. We can linearize (47) by replacing the RHS with its Jacobian, giving

$$\begin{bmatrix}\dot{x}\\ \dot{y}\end{bmatrix} \approx \begin{bmatrix}-1 & -2r\\ 1 & 2r-g\end{bmatrix}\begin{bmatrix}x\\ y\end{bmatrix}. \tag{48}$$

The origin is a fixed point. In fact, it is a unique fixed point, since (from (47)) $\tanh(x) = -2r\tanh(y) \Longrightarrow x = y = 0$. The origin is stable for

$$r < \frac{1+g}{2}, \tag{49}$$

and unstable otherwise. It is a spiral for

$$\frac{1+g}{2} - \sqrt{g} < r < \frac{1+g}{2} + \sqrt{g}, \tag{50}$$

and a node otherwise.

9.2. Large Signal Analysis. For certain values of r and g, the nonlinearities of the second-order section causes it to suffer instability. In this region of parameter space, linear

analysis accurately predicts that the circuit is *small signal stable*, but completely fails to recognize that instability would occur for *large signals*. Mead addresses this issue in [14], but we will present a somewhat more thorough treatment of the problem, using phase-plane analysis and experimental verification with the FPAA.

For very large values of x and y, the tanh functions get saturated, and can each be approximated with a signum function. Equation (47) becomes

$$\begin{aligned} \frac{dx}{d\tau} &= -\operatorname{sgn}(x) - 2r\operatorname{sgn}(y), \\ \frac{dy}{d\tau} &= \operatorname{sgn}(x) + (2r - g)\operatorname{sgn}(y). \end{aligned} \tag{51}$$

Figure 12 shows the phase plane (x versus y) that corresponds to (51). The depicted motion is valid only if $1/2 < r < 1$. The gradient in the first and third quadrants is

$$a = \frac{2r - 1 + g}{1 + 2r}, \tag{52}$$

and that in the second and fourth quadrants is

$$b = \frac{2r - 1 - g}{1 - 2r}. \tag{53}$$

Observe that, with an initial condition of $(1, 0)$, (51) predicts that the positive x-axis will again be intercepted at $(a^2/b^2, 0)$. If $a^2/b^2 > 1$, then x and y will grow without bound. Stated in terms of the r and g variables, there is large signal instability if

$$r > \frac{g + \sqrt{g^2 + 4}}{4}. \tag{54}$$

Our analysis of the second-order section can readily be verified experimentally. We compiled the filter on the FPAA, as shown in Figure 7. The bias currents of all three OTAs are user-programmable, and varying them corresponds to varying the values of r and g. The FPAA thus allows us to explore the parameter space of the filter, and to observe changes in its qualitative behavior. It can effectively be used for bifurcation analysis.

Figure 13 shows the filter's phase plane plots for various values of r, with g kept fixed. Just as we predicted, there is a unique fixed point, which is initially stable, and gradually changes from a node to a spiral (Figures 13(a) and 13(c)). While linear analysis would predict these three responses as damped, slightly underdamped, and very underdamped, it fails to recognize the possibility of the fourth response, which is large-signal instability. In the fourth panel, r meets the criterion derived from nonlinear analysis, (54), and we observe oscillation. Further analysis and exploration of parameter space reveals that this second-order section is capable of low-distortion sinusoidal oscillation [15]. Such functionality is valuable in communication systems.

10. Conclusion

In this paper, we have introduced visual and graphical techniques for analyzing nonlinear circuit dynamics. Our approach to studying harmonic distortion yields information about the various processing flows that are responsible for each harmonic term. The FPAA was used to rapidly create phase plane plots, which concisely encapsulate the nonlinear dynamics of the circuit under study. We have provided various examples of our techniques and have compared our predictions to experimentally-measured data.

References

[1] R. Sarpeshkar, M. W. Baker, C. D. Salthouse, J.-J. Sit, L. Turicchia, and S. M. Zhak, "An analog bionic ear processor with zero-crossing detection," in *Proceedings of IEEE International Solid-State Circuits Conference (ISSCC '05)*, vol. 48, pp. 78–79, San Francisco, Calif, USA, February 2005.

[2] F. Serra-Graells, L. Gomez, and J. L. Huertas, "A true-l-V 300-W CMOS-subthreshold log-domain hearing-aid-on-chip," *IEEE Journal of Solid-State Circuits*, vol. 39, no. 8, pp. 1271–1281, 2004.

[3] F. Marton and R. Säljö, "On qualitative differences in learning 1: outcome and process," *British Journal of Educational Psychology*, vol. 46, pp. 4–11, 1976.

[4] R. D. Middlebrook, "Low-entropy expressions: the key to design-oriented analysis," in *Proceedings of 21st Frontiers in Education Conference*, pp. 399–403, West Lafayette, Ind, USA, September 1991.

[5] T. A. DeFanti, M. D. Brown, and B. H. McCormick, "Visualization: expanding scientific and engineering research opportunities," *Computer*, vol. 22, no. 8, pp. 12–16, 1989.

[6] K. Odame and P. E. Hasler, "Harmonic distortion analysis via perturbation methods," in *Proceedings of the 51st Midwest Symposium on Circuits and Systems*, pp. 554–557, Knoxville, Tenn, USA, August 2008.

[7] A. Buonomo and A. L. Schiavo, "Perturbation analysis of nonlinear distortion in analog integrated circuits," *IEEE Transactions on Circuits and Systems I*, vol. 52, no. 8, pp. 1620–1631, 2005.

[8] R. Rand, *Lecture Notes on Nonlinear Vibrations*, The Internet-First University Press, Ithaca, NY, USA, 2003.

[9] S. H. Strogatz, *Nonlinear Dynamics and Chaos*, Westview, Cambridge, Mass, USA, 1994.

[10] C. C. Enz, F. Krummenacher, and E. A. Vittoz, "An analytical MOS transistor model valid in all regions of operation and dedicated to low-voltage and low-current applications," *Analog Integrated Circuits and Signal Processing*, vol. 8, no. 1, pp. 83–114, 1995.

[11] C. M. Twigg and P. Haslet, "A large-scale reconfigurable analog signal processor (RASP) IC," in *Proceedings of the IEEE Custom Integrated Circuits Conference (CICC '06)*, pp. 5–8, September 2006.

[12] B. P. Lathi, *Modern Digital and Analog Communication Systems*, Oxford University Press, New York, NY, USA, 1998.

[13] R. F. Lyon and C. Mead, "An analog electronic cochlea," *IEEE Transactions on Acoustics, Speech, and Signal Processing*, vol. 36, no. 7, pp. 1119–1134, 1988.

[14] C. Mead, *Analog VLSI and Neural Systems*, Addison-Wesley, Reading, Mass, USA, 1989.

[15] K. Odame and P. Hasler, "Exploiting ota nonlinearity in the design of a second order section oscillator," in *Proceedings of the RISP International Workshop on Nonlinear Circuits and Signal Processing*, pp. 5–8, March 2006.

Hindawi Publishing Corporation
VLSI Design
Volume 2010, Article ID 706548, 8 pages
doi:10.1155/2010/706548

Research Article

Linearity Analysis on a Series-Split Capacitor Array for High-Speed SAR ADCs

Yan Zhu,[1] U-Fat Chio,[1] He-Gong Wei,[1] Sai-Weng Sin,[1] Seng-Pan U,[1] and R. P. Martins[1,2]

[1] *Analog and Mixed Signal VLSI Laboratory, Faculty of Science and Technology, University of Macau, Macao, China*
[2] *Instituto Superior Técnico, TU of Lisbon, 1049-001 Lisbon, Portugal*

Correspondence should be addressed to Yan Zhu, ya87403@umac.mo

Received 31 May 2009; Accepted 8 December 2009

Academic Editor: Benjamin J. Blalock

A novel Capacitor array structure for Successive Approximation Register (SAR) ADC is proposed. This circuit efficiently utilizes charge recycling to achieve high-speed of operation and it can be applied to high-speed and low-to-medium-resolution SAR ADC. The parasitic effects and the static linearity performance, namely, the INL and DNL, of the proposed structure are theoretically analyzed and behavioral simulations are performed to demonstrate its effectiveness under those nonidealities. Simulation results show that to achieve the same conversion performance the proposed capacitor array structure can reduce the average power consumed from the reference ladder by 90% when compared to the binary-weighted splitting capacitor array structure.

1. Introduction

The SAR ADC is widely used in many communication systems, such as ultra-wideband (UWB) and wireless sensor networks which require low-to-medium-resolution converters with low power consumption. Traditional SAR ADCs are difficult to be applied in high-speed design; however the improvement of technologies and design methods have allowed the implementation of high-speed, low-power SAR ADCs that become consequently more attractive for a wide variety of applications [1, 2].

The power dissipation in an SAR converter is dominated by the reference ladder of the DAC capacitor array. Recently, a capacitor splitting technique has been presented, which was proven to use 31% less power from the reference voltage and achieve better DNL than the binary-weighted capacitor (BWC) array. The total power consumption of a 5 b binary-weighted split capacitor (BWSC) array is 6 mW which does not take into account the power from the reference ladder [3]. However, as the resolution increases, the total number of input capacitance in the binary-scaled capacitive DAC will cause an exponential increase in power dissipation as well as a limitation with reduction of speed due to a large charging time-constant. Therefore, small capacitance spread for DAC capacitor arrays is highly desirable in high-speed SAR ADCs [4].

This paper presents a novel structure of a split capacitor array for optimization of the power efficiency and the speed of SAR ADCs. Due to the series combination of the split capacitor array both small capacitor ratios and power-efficient charge recycling in the DAC capacitor array can be achieved, leading to fast DAC settling time and low power dissipation in the SAR ADC. The parasitic effects, the position of the attenuation capacitor, as well as the linearity performance (INL and DNL) of the proposed structure will be theoretically discussed and behavioral simulations will be performed. Different from the BWSC array, which only achieves better DNL (but not INL) than the BWC array, the proposed capacitor array structure can have both better INL and DNL than the series capacitor (SC) array. The design and simulations of an 8 b 180-MS/s SAR ADC in 1.2-V supply voltage are presented in 90 nm CMOS exhibiting a Signal-to-Noise-and-Distortion Ratio (SNDR) of 48 dB, with a total power consumption of 14 mW which demonstrates the feasibility of the proposed structure.

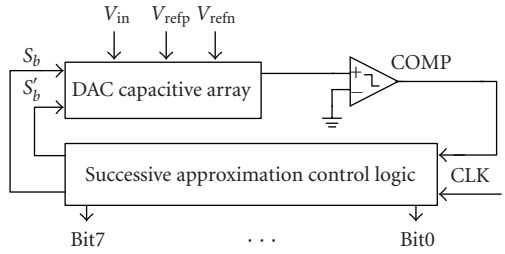


Figure 1: Simplified block diagram of an SAR ADC architecture.

2. The Overall SAR ADC Operation

The architecture of an SAR ADC is shown in Figure 1, consisting of a series structure of a capacitive DAC, a comparator, and successive approximation (SA) control logic. The SA control logic includes shift registers and switch drivers which control the DAC operation by performing the binary-scaled feedback during the successive approximation. The DAC capacitor array is the basic structure of the SAR ADC and it serves both to sample the input signal and as a DAC for creating and subtracting the reference voltage.

3. Capacitor Array Structure

3.1. Capacitor Structure Design. The major limitation on the speed of the SA converter is often related with the RC time constants of the capacitor array, reference ladder, and switches. For a BWC array the size of capacitors rises exponentially with the resolution in number of bits, which causes large power and RC settling time, thus limiting the speed of the overall SAR ADC. To solve this problem, Figure 2(a) shows an SC array [5], which utilizes attenuation capacitors C_{atten} to separate the capacitive DAC into b_M bits MSB and b_L bits LSB arrays. Thus, smaller capacitor ratios can be achieved when compared to the BWC array. However, charge-redistribution switching method for the SC array has been proven to be inefficient when discharging the MSB capacitor and charging the MSB/2 capacitor, which consumes 5 times more power than the charge-recycling switching method. Thus, a series-split capacitor (SSC) array is proposed, as shown in Figure 2(b), which can both alleviate the speed limitation and implement a charge-recycling switching approach.

The solution to perform charge-recycling for SC array is different from the BWC array, which just splits the MSB capacitor C_{MSB} into $n-1$ subarrays. As illustrated in Figure 2(b), the C_{MSB} of the SC array is split into $b_M - 1$ subarrays in the MSB array, where the total capacitance of the b_M-1 subarrays is $C_{MSB} - C_0$ and as a result the capacitors in LSB array and C_{atten} should be doubled; thus the C_{eq} can be calculated as

$$C_{eq} = 2C_{atten} // C_{totalLSB} = 2C_0,$$
$$C_{totalLSB} = 2^{b_L+1}C_0,$$
$$C_{totalMSB} = \sum_{n=1}^{b_M-1} 2^n C_0, \quad (1)$$

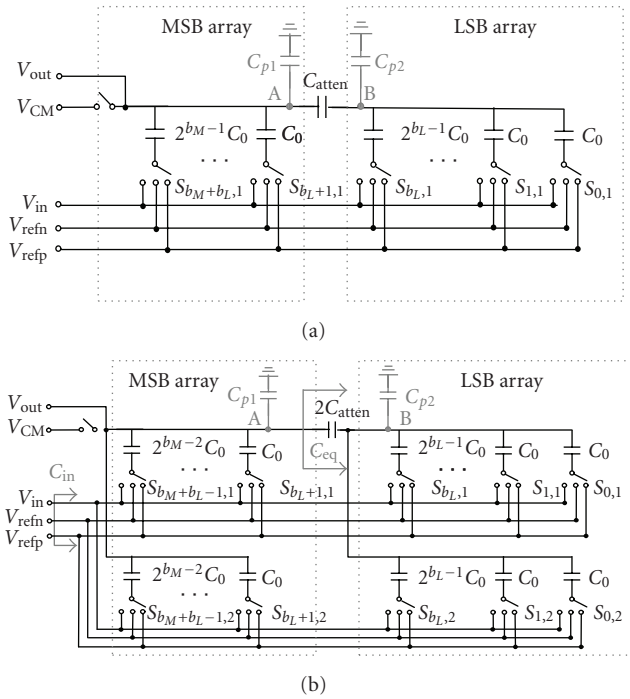


Figure 2: (a) $(b_M + b_L)$-bit SC array, (b) $(b_M + b_L)$-bit SSC array.

where $C_{totalLSB}$ and $C_{totalMSB}$ are the sum of LSB and MSB array capacitors, respectively. The C_{eq} can then be seen as two split unit capacitors C_0 attached to the right side of MSB array to maintain the capacitive ratio as $2^{bM-2} : \cdots : 2 : 1 : 1$. Therefore, the charge-recycling methodology in each section can perform binary-scaled feedback during the successive approximation.

3.2. Charge Recycling Implementation. In the proposed implementation the series-split capacitor array is designed to achieve charge recycling for the n $(n = b_M + b_L + 1)$ bit capacitive DAC, as shown in Figure 2(b). During the global sampling phase, the voltage V_{in} is stored in the entire capacitor array. Then, the algorithmic conversion begins by switching all upper capacitor arrays to V_{ref} and the lower to $-V_{ref}$, respectively, instead of switching only the MSB capacitor to V_{ref} and others to $-V_{ref}$. This implies that in the conversion phase 1 (corresponding to MSB capacitor conversion) V_{out} settles to (considering only differential node voltage)

$$V_{out}[1] = -V_{in} + \frac{V_{ref}}{2} - \frac{V_{ref}}{2} = -V_{in}, \quad (2)$$

and the comparator output will be

$$D_1 = \begin{cases} -1, & V_{in} > 0, \\ 1, & V_{in} < 0. \end{cases} \quad (3)$$

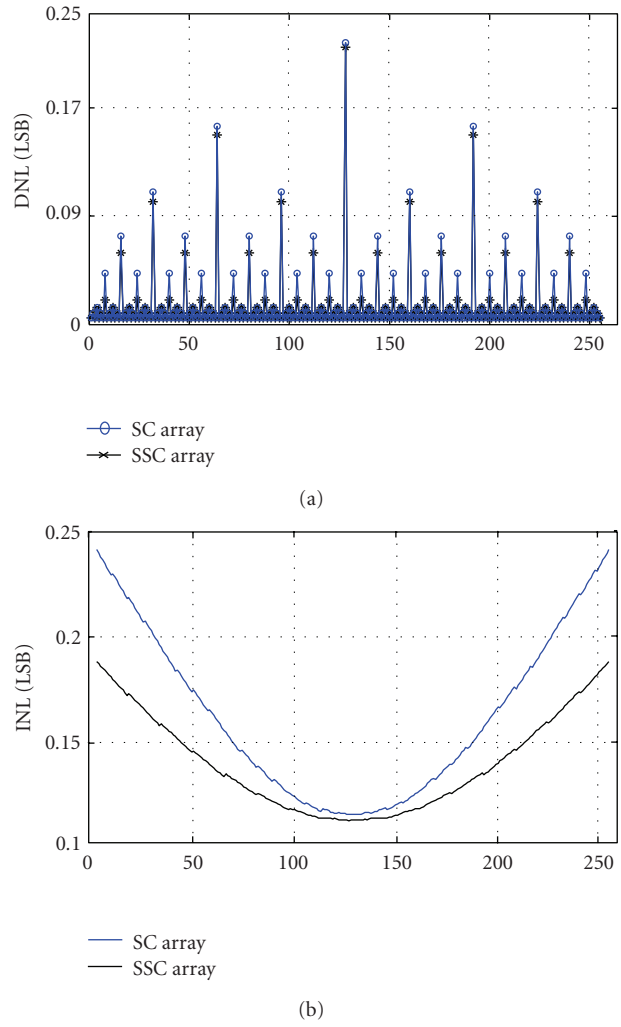


Figure 3: Behavioral simulations comparing the linearity of the SSC and the SC array.

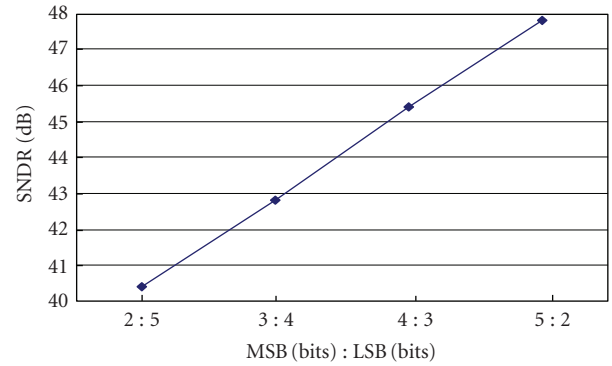


Figure 4: Behavioral simulation of 1000 Monte Carlo SNDR versus the different bits distribution of MSB and LSB arrays.

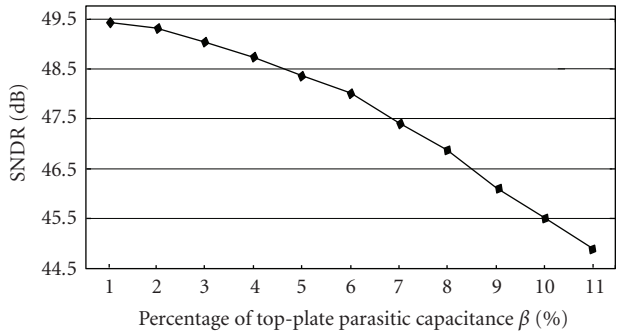


Figure 5: Behavioral simulation of 1000 Monte Carlo SNDR versus the percentage of the top-plate parasitic capacitance β for the SSC array at 8 bit level.

The comparator output will decide the switching logic of $S_{bM+bL-1,1}$ and $S_{bM+bL-1,2}$. If D_1 is low, $S_{bM+bL-1,1}$ is switched to $-V_{\text{ref}}$, dropping the voltage at $V_{\text{out}}[2]$ to $-V_{\text{in}} - V_{\text{ref}}/2$. If D_1 is high, $S_{bM+bL-1,2}$ is switched to V_{ref}, raising the voltage at $V_{\text{out}}[2]$ to $-V_{\text{in}} + V_{\text{ref}}/2$. The above process is repeated for $n-1$ cycles. As $S_{bM+bL-1,1}$ is switched from V_{ref} to $-V_{\text{ref}}$ (bit decision back from "1" to "0") the switches, from $S_{0,1}$ to $S_{bM+bL-2,1}$, are kept connected to V_{ref} and drive $V_{\text{out}}[1]$ to $V_{\text{out}}[2]$. The initial charge, supplied by V_{ref} in phase 1, is kept stored in the capacitors which will connect to V_{ref} at phase 1, instead of being redistributed; so the charge formed at phase 1 can be recycled in the next $n-1$ phases. However, the conventional switching method that discharges MSB capacitor and charges the MSB/2 capacitor will cause charge redistribution in the capacitor array and thus consuming more power.

3.3. Linearity Performance. To analyze the linearity of the SSC and SC arrays, each of the capacitors is modeled as the sum of the nominal capacitance value and the error term, as follows:

$$C_{n,1} = 2^{n-1}C_0 + \delta_{n,1},$$
$$C_{n,2} = 2^{n-1}C_0 + \delta_{n,2}. \tag{4}$$

Consider the case where all the errors are in the unit capacitors whose values are independent-identically-distributed Gaussian random variables with a variance of

$$E\left[\delta_{n,1}^2\right] = E\left[\delta_{n,2}^2\right] = 2^{n-1}\sigma_0^2, \tag{5}$$

and where σ_0 is the standard deviation of the unit capacitor.

The accuracy of an SAR ADC is dependent on the DAC outputs which are calculated here in the case of no initial charge on the array ($V_{\text{in}} = 0$). For a given DAC digital input X, with $D_{n,m}$ equals 1 or 0 representing the ADC decision

for bit n, the analog output $V_{\text{out}}(X)$ of the SSC array can be calculated as

$$\begin{aligned} &V_{\text{out}}(X) \\ &= \frac{2C_{\text{atten}}\left(\sum_{n=1}^{b_L}\sum_{m=1}^{2} D_{n,m}C_{n,m} + \sum_{n=1}^{b_M-1}\sum_{m=1}^{2} D_{n,m}C_{n,m}\right)}{2C_{\text{atten}}(C_{\text{totalLSB}} + C_{\text{totalMSB}}) + C_{\text{totalLSB}}C_{\text{totalMSB}} + \Delta C} V_{\text{ref}} \\ &\quad + \frac{\left(C_{\text{totalLSB}} + \sum_{n=1}^{b_L}\sum_{m=1}^{2} \delta_{n,m}\right)\sum_{n=1}^{b_M-1}\sum_{m=1}^{2} D_{n,m}C_{n,m}}{2C_{\text{atten}}(C_{\text{totalLSB}} + C_{\text{totalMSB}}) + C_{\text{totalLSB}}C_{\text{totalMSB}} + \Delta C} V_{\text{ref}} \end{aligned} \tag{6}$$

where

$$\begin{aligned} C_{n,m} &= 2^{n-1}C_0 + \delta_{n,m}, \\ \Delta C &= \sum_{n=1}^{b_{M-1}}\sum_{m=1}^{2} \delta_{n,m}(2C_{\text{atten}} + C_{\text{totalLSB}}) \\ &\quad + \sum_{n=1}^{b_L}\sum_{m=1}^{2} \delta_{n,m}\left(2C_{\text{atten}} + C_{\text{totalMSB}} + \sum_{n=1}^{b_{M-1}}\sum_{m=1}^{2} \delta_{n,m}\right). \end{aligned} \tag{7}$$

Subtracting the nominal value (i.e., $\delta_{n,m} = 0$ in (6)) from (6) the INL can be calculated as

$$\begin{aligned} &\text{INL}_{\text{ssc}} \\ &= \frac{2C_{\text{atten}}(\delta_X + \delta_Y) + \delta_Y\delta_X + C_{\text{totalLSB}}\delta_Y}{2C_{\text{atten}}(C_{\text{totalLSB}} + C_{\text{totalMSB}}) + C_{\text{totalLSB}}C_{\text{totalMSB}} + \Delta C} V_{\text{ref}} \\ &\quad + \frac{\sum_{n=1}^{b_M-1}\sum_{m=1}^{2} D_{n,m}C_{n,m}\delta_X}{2C_{\text{atten}}(C_{\text{totalLSB}} + C_{\text{totalMSB}}) + C_{\text{totalLSB}}C_{\text{totalMSB}} + \Delta C} V_{\text{ref}}, \\ &\delta_X = \sum_{n=1}^{b_L}\sum_{m=1}^{2} D_{n,m}\delta_{n,m}, \qquad \delta_Y = \sum_{n=1}^{b_{M-1}}\sum_{m=1}^{2} D_{n,m}\delta_{n,m}, \end{aligned} \tag{8}$$

The first and second terms are quite small when compared with the third and fourth terms in the numerator, and the third term ΔC in the denominator does not depend on the bit decision $D_{n,m}$, which only causes a gain error; then they will be neglected. Thus, (8) can be simplified as

$$\text{INL}_{\text{ssc}} \approx \frac{C_{\text{totalLSB}}\delta_Y + \sum_{n=1}^{b_M-1}\sum_{m=1}^{2} D_{n,m}C_{n,m}\delta_X}{2C_{\text{atten}}(C_{\text{totalLSB}} + C_{\text{totalMSB}}) + C_{\text{totalLSB}}C_{\text{totalMSB}}} V_{\text{ref}}, \tag{9}$$

and the variance can be expressed as

$$\begin{aligned} &E\left[\text{INL}_{\text{ssc}}^2\right] \\ &= \frac{C_{\text{totalLSB}}^2(E_{b_{M-1},1} + E_{b_{M-1},2})}{\left[2C_{\text{atten}}(C_{\text{totalLSB}} + C_{\text{totalMSB}}) + C_{\text{totalLSB}}C_{\text{totalMSB}}\right]^2} V_{\text{ref}}^2 \\ &\quad + \frac{\left(\sum_{n=1}^{b_M-1}\sum_{m=1}^{2} D_{n,m}C_{n,m}\right)^2 (E_{b_L,1} + E_{b_L,2})}{\left[2C_{\text{atten}}(C_{\text{totalLSB}} + C_{\text{totalMSB}}) + C_{\text{totalLSB}}C_{\text{totalMSB}}\right]^2} V_{\text{ref}}^2, \end{aligned} \tag{10}$$

$$\begin{aligned} E_{b_{M-1},1} &= E\left[\left(\sum_{n=1}^{b_{M-1}} D_{n,1}\delta_{n,1}\right)^2\right], \\ E_{b_{M-1},2} &= E\left[\left(\sum_{n=1}^{b_{M-1}} D_{n,2}\delta_{n,2}\right)^2\right], \\ E_{b_L,1} &= E\left[\left(\sum_{n=0}^{b_L} D_{n,1}\delta_{n,1}\right)^2\right], \\ E_{b_L,2} &= E\left[\left(\sum_{n=0}^{b_L} D_{n,2}\delta_{n,2}\right)^2\right]. \end{aligned} \tag{11}$$

To simplify the analysis only the worse INL is considered that combines all the errors together (i.e., $D_{n,m} = 1$). For (5) it can be concluded that $E_{bM-1,1} = E_{bM-1,2}$ and $E_{bL,1} = E_{bL,2}$. Thus (10) can be simplified as

$$\begin{aligned} &E\left[\text{INL}_{\text{ssc}}^2\right] \\ &= \frac{\left(2^{b_L}C_0\right)^2\sum_{n=1}^{b_M-1} 2^n\sigma_0^2 + \left(\sum_{n=1}^{b_M-1} 2^{n-1}C_0\right)^2\sum_{n=1}^{b_L} 2^n\sigma_0^2}{\left[C_{\text{atten}}(2^{b_L+1}C_0 + \sum_{n=1}^{b_M-1} 2^nC_0) + 2^{b_L}C_0\sum_{n=1}^{b_M-1} 2^nC_0\right]^2} V_{\text{ref}}^2. \end{aligned} \tag{12}$$

While for the SC array, the $E[\text{INL}_{\text{sc}}^2]$ can be calculated similarly as

$$\begin{aligned} &E\left[\text{INL}_{\text{sc}}^2\right] \\ &= \frac{\left(2^{b_L}C_0\right)^2\sum_{n=1}^{b_M} 2^{n-1}\sigma_0^2}{\left[C_{\text{atten}}(2^{b_L}C_0 + \sum_{n=1}^{b_M} 2^{n-1}C_0) + 2^{b_L}C_0\sum_{n=1}^{b_M} 2^{n-1}C_0\right]^2} V_{\text{ref}}^2 \\ &\quad + \frac{\left(\sum_{n=1}^{b_M} 2^{n-1}C_0\right)^2\sum_{n=1}^{b_L} 2^{n-1}\sigma_0^2}{\left[C_{\text{atten}}(2^{b_L}C_0 + \sum_{n=1}^{b_M} 2^{n-1}C_0) + 2^{b_L}C_0\sum_{n=1}^{b_M} 2^{n-1}C_0\right]^2} V_{\text{ref}}^2. \end{aligned} \tag{13}$$

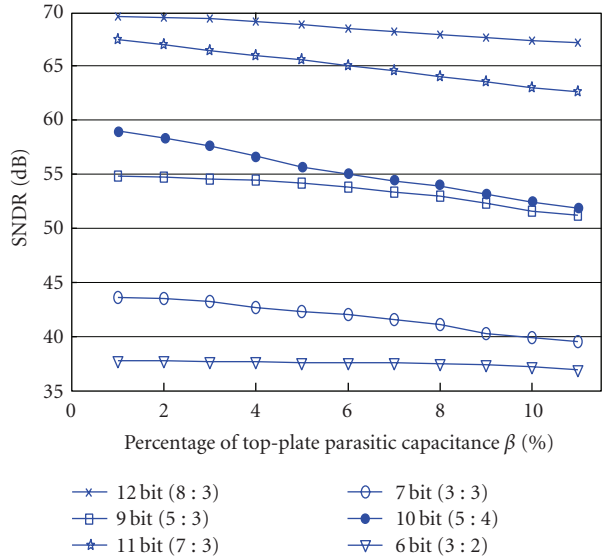


Figure 6: Behavioral simulation of 1000 Monte Carlo SNDR versus the percentage of the top-plate parasitic capacitance β of series-split capacitor array at 6- to 12-bit level.

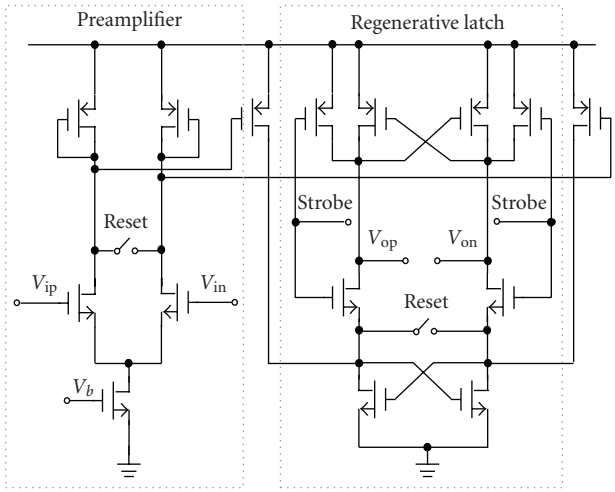


Figure 7: Circuit schematic of the dynamic comparator.

Then, subtracting (12) from (13), its value will become

$$E\left[\mathrm{INL}_{\mathrm{sc}}^2\right] - E\left[\mathrm{INL}_{\mathrm{ssc}}^2\right] \approx \frac{\left(2^{b_L}C_0\right)^2\sigma_0^2 + \left(\sum_{n=1}^{b_{M-1}} 2^n C_0\right)^2 \sum_{n=1}^{b_L} 2^{n-2}\sigma_0^2}{\left[C_{\mathrm{atten}}\left(2^{b_L}C_0 + \sum_{n=1}^{b_M} 2^{n-1}C_0\right) + 2^{b_L}C_0 \sum_{n=1}^{b_M} 2^{n-1}C_0\right]^2} > 0. \tag{14}$$

As a result of (14), the INL of the SSC array should be lower than the SC array which is different from the BWC and BWSC arrays that were already proven to have no difference between the INLs [1].

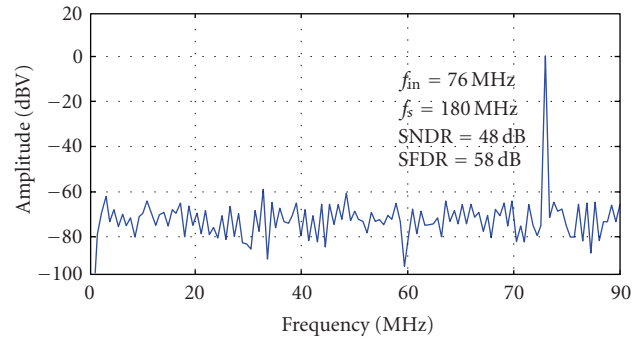


Figure 8: Simulated FFT spectrum of the ADC.

The maximum DNL for the SSC array is expected to occur at the step below the MSB transition [1], and the two output voltages can be calculated as

$$V_{\mathrm{err}}(X) \approx \frac{C_{\mathrm{totalLSB}} \sum_{n=1}^{b_{M-1}} \delta_n + \sum_{n=1}^{b_{M-1}} 2^{n-1}C_0 \sum_{n=1}^{b_L} \sum_{m=1}^{2} \delta_{n,m}}{2C_{\mathrm{atten}}\left(C_{\mathrm{totalLSB}} + C_{\mathrm{totalMSB}}\right) + C_{\mathrm{totalLSB}}C_{\mathrm{totalMSB}}} V_{\mathrm{ref}}, \tag{15}$$

$$V_{\mathrm{err}}(X-1) \approx \frac{C_{\mathrm{totalLSB}} \sum_{n=1}^{b_M-2} \sum_{m=1}^{2} \delta_{n,m}}{2C_{\mathrm{atten}}\left(C_{\mathrm{totalLSB}} + C_{\mathrm{totalMSB}}\right) + C_{\mathrm{totalLSB}}C_{\mathrm{totalMSB}}} V_{\mathrm{ref}} + \frac{\sum_{n=1}^{b_M-2} 2^n C_0 \sum_{n=1}^{b_L} \sum_{m=1}^{2} \delta_{n,m}}{2C_{\mathrm{atten}}\left(C_{\mathrm{totalLSB}} + C_{\mathrm{totalMSB}}\right) + C_{\mathrm{totalLSB}}C_{\mathrm{totalMSB}}} V_{\mathrm{ref}}, \tag{16}$$

subtracting (16) from (15), they will yield

$$\mathrm{DNL}_{\mathrm{ssc}} = \frac{2^{b_L+1}C_0\left(\delta_{b_{M-1}} - \sum_{n=1}^{b_M-2} \delta_{n,2}\right) + C_0 \sum_{n=1}^{b_L} \sum_{m=1}^{2} \delta_{n,m}}{2C_{\mathrm{atten}}\left(C_{\mathrm{totalLSB}} + C_{\mathrm{totalMSB}}\right) + C_{\mathrm{totalLSB}}C_{\mathrm{totalMSB}}} V_{\mathrm{ref}}^2 \tag{17}$$

with variance

$$E\left[\mathrm{DNL}_{\mathrm{ssc}}^2\right] = \frac{\left(2^{b_L}C_0\right)^2\left(2^{b_M-2}\sigma_0^2 - \sum_{n=1}^{b_M-2} 2^{n-1}\sigma_0^2\right)}{\left[C_{\mathrm{atten}}\left(2^{b_L+1}C_0 + \sum_{n=1}^{b_M-1} 2^n C_0\right) + 2^{b_L}C_0 \sum_{n=1}^{b_M-1} 2^n C_0\right]^2} V_{\mathrm{ref}}^2 + \frac{C_0 \sum_{n=1}^{b_L} 2^{n-1}\sigma_0^2}{\left[C_{\mathrm{atten}}\left(2^{b_L+1}C_0 + \sum_{n=1}^{b_M-1} 2^n C_0\right) + 2^{b_L}C_0 \sum_{n=1}^{b_M-1} 2^n C_0\right]^2} V_{\mathrm{ref}}^2 = \frac{\left[\left(2^{b_L}C_0\right)^2 + 2C_0 \sum_{n=1}^{b_L} 2^{n-1}\right]\sigma_0^2}{2\left[C_{\mathrm{atten}}\left(2^{b_L+1}C_0 + \sum_{n=1}^{b_M-1} 2^n C_0\right) + 2^{b_L}C_0 \sum_{n=1}^{b_M-1} 2^n C_0\right]^2} V_{\mathrm{ref}}^2. \tag{18}$$

For SC array the $E[\mathrm{DNL}_{\mathrm{sc}}^2]$ can be calculated similarly as

$$E\left[\mathrm{DNL}_{\mathrm{sc}}^2\right] = \frac{\left[\left(2^{b_L}C_0\right)^2 + C_0\sum_{n=1}^{b_L}2^{n-1}\right]\sigma_0^2}{\left[C_{\mathrm{atten}}\left(2^{b_L}C_0+\sum_{n=1}^{b_M}2^{n-1}C_0\right)+2^{b_L}C_0\sum_{n=1}^{b_M}2^{n-1}C_0\right]^2}V_{\mathrm{ref}}^2; \tag{19}$$

thus, $E[\mathrm{DNL}_{\mathrm{sc}}^2]/\ E[\mathrm{DNL}_{\mathrm{ssc}}^2]$ can be expressed as

$$\frac{E\left[\mathrm{DNL}_{\mathrm{ssc}}^2\right]}{E\left[\mathrm{DNL}_{\mathrm{sc}}^2\right]} \approx \frac{\left(2^{b_L}C_0\right)^2 + 2C_0\sum_{n=1}^{b_L}2^{n-1}}{2\left[\left(2^{b_L}C_0\right)^2 + C_0\sum_{n=1}^{b_L}2^{n-1}\right]} < 1. \tag{20}$$

Thus, from (20) it can be concluded that the maximum DNL of the SSC is also lower than that of the SC array.

3.4. Parasitic Nonlinearity Effect. One potential issue with these two series capacitor array structures (SSC and SC) is the parasitic capacitances C_{p1} and C_{p2} on the nodes A and B, which will deteriorate the desired voltage division ratio and result in poor linearity. The parasitic effect is caused by the bottom- and top-plate parasitic capacitance of C_{atten} as well as the top-plate parasitic capacitance of MSB and LSB array capacitors which can be calculated as

$$\begin{aligned} C_{p1} &= \alpha\cdot C_{\mathrm{atten}} + \beta\cdot C_{\mathrm{sumMSB}},\\ C_{p2} &= \beta\cdot C_{\mathrm{atten}} + \beta\cdot C_{\mathrm{sumLSB}}, \end{aligned} \tag{21}$$

where α and β represent the percentage of bottom- and top-plate parasitic capacitances of each capacitor, respectively (with metal-isolator-metal (MIM) capacitor option, α = 10%, β = 5%). For the SSC array, the analog output $V_{\mathrm{out}}(X)$ with C_{p1} and C_{p2} taken in to account can be calculated as

$$\begin{aligned} V_{\mathrm{out}}(X) &= \frac{C_{\mathrm{atten}}\left(\sum_{n=1}^{b_L}\sum_{n'=1}^{2}D_{n,n'}2^{n-1}C_0+\sum_{n=1}^{b_M}\sum_{b'=1}^{2}D_{n,n'}2^{n-1}C_0\right)}{C_{\mathrm{atten}}\left(C_{\mathrm{sumLSB}}+C_{\mathrm{sumMSB}}+C_{p1}+C_{p2}\right)+\mathfrak{D}}V_{\mathrm{ref}}\\ &\quad+\frac{\left(C_{\mathrm{sumLSB}}+C_{p2}\right)\sum_{n=1}^{b_M}\sum_{n'=1}^{2}D_{n,n'}2^{n-1}C_0}{C_{\mathrm{atten}}\left(C_{\mathrm{sumLSB}}+C_{\mathrm{sumMSB}}+C_{p1}+C_{p2}\right)+\mathfrak{D}}V_{\mathrm{ref}}, \end{aligned} \tag{22}$$

where $\mathfrak{D}$ denotes $(C_{\mathrm{sumLSB}} + C_{p2})(C_{\mathrm{sumMSB}} + C_{p1})$. This equation shows that the parasitic capacitances C_{p1} and C_{p2} in the denominator are completely uncorrelated in the bit decisions, which can cause only a gain error and have no effect into the linearity performance. However, the parasitic capacitance C_{p2} in the numerator contributes with a code-dependent error, which degrades the linearity of the SAR ADC. Subtracting the nominal value the error term will become

$$\begin{aligned} V_{\mathrm{error}}(X) &= \frac{C_{p2}\sum_{n=1}^{b_M}\sum_{n'=1}^{2}D_{n,n'}2^{n-1}C_0}{C_{\mathrm{atten}}\left(C_{\mathrm{sumLSB}}+C_{\mathrm{sumMSB}}+C_{p1}+C_{p2}\right)+\mathfrak{D}}V_{\mathrm{ref}}. \end{aligned} \tag{23}$$

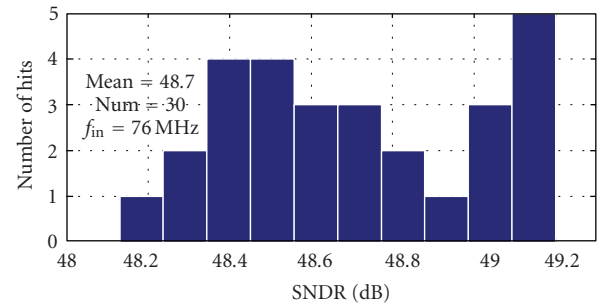


Figure 9: 30-times Monte Carlo simulation of SNDR from the 8 b SAR.

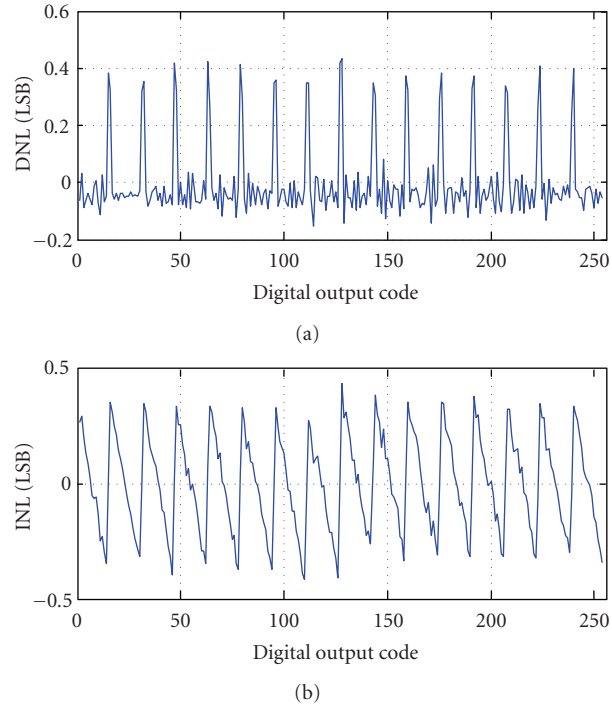


Figure 10: Simulated linearity: (a) DNL and (b) INL.

The parasitic capacitance C_{p2} is composed of the parasitic capacitance of C_{atten} and C_{sumLSB}. By reducing the number of bits in the LSB array, the size of C_{sumLSB} can be minimized; thus the nonlinearity effect can be alleviated. But, this will enlarge the capacitor spread in the MSB array; thus the distribution of bits in both MSB and LSB arrays should consider the trade-off between linearity, tolerance, and capacitance spread limitations.

3.5. Behavioral Simulations. Four behavioral simulations of the SSC and the SC array DAC were performed to verify the previous analysis. The values of the unit and attenuation capacitors used are Gaussian random variables with standard deviation of 1% (σ_0/C_0 = 0.01), and the ADC is otherwise ideal. Figure 3 shows the result of 10000-time Monte Carlo runs, where the standard deviation of DNL and INL is plotted versus output code at the 8-bit level. As expected,

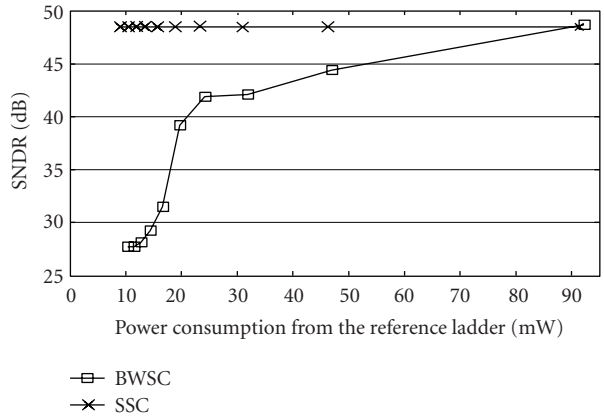


Figure 11: Simulated SNDR versus power consumption from the reference ladder for series-split (SSC) and binary-weighted split-capacitor array (BWSC).

the SSC array has better INL and DNL than its SC array counterpart. Figure 4 shows the result of 1000 Monte Carlo runs with 5% top-plate and 10% bottom-plate parasitic capacitances, where the SNDRs are plotted versus different distribution of bits in the MSB and LSB array at the 8-bit level. Comparing the SNDR shown in Figure 4, and as expected, a larger number of bits in the LSB array will cause poor linearity. Although MSB : LSB = 5 : 2 can achieve the best SNDR, since larger capacitor ratios will both reduce the conversion speed and increase the power dissipation, MSB : LSB = 4 : 3 will be adopted for circuit implementation due to both good linearity performance and smaller capacitor ratios. Figure 5 illustrates the result of 1000-time Monte Carlo runs, where the SNDRs are plotted versus the percentage of the top-plate parasitic capacitance β for the SSC array structure for an 8-bit ADC. With C_{p2} increasing, the parasitic capacitance will decrease the SNDR of the conversion performance. But with approximate ±5% variance of β a good linearity performance of an SAR ADC can still be achieved. Figure 6 illustrates the result of 1000 Monte Carlo runs, where the SNDRs are plotted versus the percentage of the top-plate parasitic capacitance β at the 6- to 12-bit level with proper bits distribution of the LSB and MSB arrays. From it we can find that the parasitic nonlinearity effect is insignificant; thus the series split structure can also be utilized in high-resolution applications.

3.6. Power Consumption Analysis. The power consumption of the SAR converter is dominated by the DAC capacitor array, the comparator, and the switches' drivers. The array's power is proportional to the sum of the array total capacitance C_{total} of which the bottom-plate is connected to the reference voltage supply and can be calculated as

$$P_{\text{array}} = C_{\text{total}} V_{\text{ref}} V_{\text{FS}} \tag{24}$$

where V_{FS} is the full-scale input voltage, assuming that V_{FS} has been fully sampled on to the capacitor array and the charge is all supplied by the reference voltage V_{ref} [1]. In a 8-bit case, the C_{total} of the proposed structure is $46C_0$ (with MSB : LSB = 4 : 3), but for a binary-weighted capacitor array the C_{total} is $256C_0$, which can consume 5 times more power than the proposed structure. The series combination allows a significant reduction of the largest capacitor ratio; in an 8-bit case, the largest capacitor C_{max} of the series split and binary-weighted split capacitor array structure is $8C_0$ and $64C_0$, respectively, which decreases the DAC settling time and speeds up the conversion. The total input capacitance of the proposed structure is not completely dependent on the number of bits of the ADC and can be calculated as

$$C_{\text{in}} = C_{\text{sumMSB}} + 2C_0. \tag{25}$$

The power consumptions of the comparator and switch drivers are also proportional to the equivalent input capacitance C_{in}. Therefore, the smaller C_{total}, C_{max}, and C_{in} it will imply an increase in efficiency of the overall conversion performance.

Table 1: Performance Summary of the SAR ADC.

Technology	90-nm CMOS with MIM
Resolution	8 bit
Sampling Rate	180 MS/s
Supply Voltage	1.2 V
Full Scale Analog Input	1.2 Vpp differential
SNDR ($@f_{\text{in}} = 76$ MHz)	48 dB
SFDR ($@f_{\text{in}} = 76$ MHz)	58 dB
ENOB ($@f_{\text{in}} = 76$ MHz)	7.7 bit
FOM	0.37 pJ/conversion step
DNL	±0.5 LSB
INL	±0.5 LSB
Power Consumption	
Analog	9.4 mW
Digital	2.3 mW
Reference ladder	2.3 mW
Total	14 mW

4. Circuit Implementation Details

A high-speed SAR converter imposes a stringent requirement in the clock generation; for example, an 8 b 180 MS/s SAR ADC requires an internal master clock of over 1.62 GHz. To generate such a high-frequency clock pulse the generator will consume even more power than the ADC itself. Due to the power limitations of the clock generator in a synchronous SA design an asynchronous SAR processing technique [4] will be adopted here, where only a master clock of 180 MHz is required.

The dynamic comparator [6] used in this ADC is shown in Figure 7 and it is composed of a preamplifier and a regenerative latch. The preamplifier can provide sufficient gain to suppress the relatively high input referred offset voltage of the latch. Also, the kickback noise of the latch can be isolated by the current mirror between the two stages. The dynamic operation of this circuit is divided into a reset phase and a regeneration phase. During the reset phase the

two outputs (V_{op} and V_{on}) are pulled up to V_{DD}. After the input stage has settled, the voltage difference is then amplified to a full swing during the regeneration phase. The differential output can generate a data ready signal to indicate the completion of the comparison, which will be used to trigger a sequence of shift registers and the switch drivers to perform asynchronous conversion [4]. Dynamic logic circuits are also utilized instead of traditional static logic to release the limitation of digital feedback propagation delay in the SA loop.

5. Simulation of an 8-Bit 180 MS/s SAR ADC

To verify the proposed capacitor structure of the capacitive DAC, a 1.2 V, 8 b, 180-MS/s SAR ADC was designed using a 90 nm CMOS process with metal-isolator-metal (MIM) capacitor option. The SAR ADC was implemented in a fully differential architecture, with a full scale differential input range of 1.2 V_{PP}. Considering the parasitic capacitance of the attenuation capacitors that will reduce the linearity of the ADC, 5% top-plate and 10% bottom-plate, they were included in the simulations according to the data from the foundry datasheet.

Figure 8 shows a spectrum plot of the SAR ADC after a Monte Carlo simulation with an input signal of 76 MHz leading to an SNDR of 48 dB, which clearly demonstrates the tolerance to the parasitic effect caused by the C_{p2}. Figure 9 also shows the corresponding 30-times Monte Carlo mismatch simulations where the ADC achieves a mean SNDR of 49 dB with an input signal of 76 MHz. The DNL and INL are both within ±0.5 LSB as shown in Figure 10. Figure 11 shows the SNDR versus power consumption from the reference ladder in the proposed architecture, as well as in the BWSC array structure, clearly demonstrating that the BWSC results are poor in terms of SNDR mainly due to the large RC settling time. To reach the same conversion performance the BWSC array consumes 10 times more power than the proposed structure. Table 1 summarizes the overall performance of the SAR ADC with the total power consumption of 14 mW only and an FoM of 0.37 pJ/conversion-step, distinctly proving the low power dissipation feature of the proposed technique.

6. Conclusions

A novel series-split capacitive DAC technique has been proposed which can both implement an efficient charge recycling SAR operation and achieve a small input capacitance. The reduction of the maximum ratio and sum of the total capacitance can lead to area savings and power efficiency, which allow the SAR converter to work at high speed while meeting a low power consumption requirement. Theoretical analysis and behavioral simulations of the linearity performance demonstrate that the proposed SSC structure can have a better INL and DNL than the traditional SC array structure. Simulation results of a 1.2 V, 8 b, 180-MS/s SAR ADC were presented exhibiting an SNDR of 48 dB at a 76 MHz input with the total power consumption of 14 mW that certifies the power efficiency of the novel circuit structure.

Acknowledgments

This work was financially supported by research grants from the University of Macau and FDCT with Ref nos. RG-UL/07-08S/Y1/MR01/FST and FDCT/009/2007/A1.

References

[1] B. P. Ginsburg and A. P. Chandrakasan, "500-MS/s 5-bit ADC in 65-nm CMOS with split capacitor array DAC," *IEEE Journal of Solid-State Circuits*, vol. 42, no. 4, pp. 739–747, 2007.

[2] D. Draxelmayr, "A 6b 600 MHz 10 mW ADC array in digital 90 nm CMOS," in *Proceedings of IEEE International Solid-State Circuits Conference (ISSCC '04)*, vol. 1, pp. 264–265, February 2004.

[3] B. P. Ginsburg and A. P. Chandrakasan, "An energy-efficient charge recycling approach for a SAR converter with capacitive DAC," in *Proceedings of IEEE International Symposium on Circuits and Systems (ISSCAS '05)*, pp. 184–187, 2005.

[4] M. S. W. Chen and R. W. Brodersen, "A 6b 600 MS/s 5.3 mW asynchronous ADC in 0.13 μm CMOS," in *Proceedings of IEEE International Solid-State Circuits Conference (ISSCC '06)*, pp. 574–575, February 2006.

[5] R. J. Baker, H. W. Li, and D. E. Boyce, *CMOS Circuit Design, Layout, and Simulation*, John Wiley & Sons, New York, NY, USA, 2nd edition, 2004.

[6] G. M. Yin, F. Eynde, and W. Sansen, "A high-speed CMOS comparator with 8-b resolution," *IEEE Journal of Solid-State Circuits*, vol. 27, no. 2, pp. 208–211, 1992.

Hindawi Publishing Corporation
VLSI Design
Volume 2010, Article ID 297083, 12 pages
doi:10.1155/2010/297083

Research Article

Local Biasing and the Use of Nullator-Norator Pairs in Analog Circuits Designs

Reza Hashemian

Department of Electrical Engineering, Northern Illinois University, DeKalb, IL 60115, USA

Correspondence should be addressed to Reza Hashemian, reza@ceet.niu.edu

Received 9 April 2009; Accepted 8 December 2009

Academic Editor: Benjamin J. Blalock

A new technique is presented for biasing of analog circuits. The biasing design begins with local biasing of the nonlinear components (transistors), done according to the pre-specified operating points (OPs) and for the best performance of the circuit. Next, the transistors are replaced with their linear models to perform the AC design. Upon finishing with the AC design we need to move from the local biasing to global (normal) biasing while the OPs are kept unchanged. Here fixators—nullators plus sources—are shown to be very instrumental and with norators—as the place holders for the DC supplies in the circuit—they make pairs. The solution of the circuit so prepared provides the DC supplies at the designated locations in the circuit. The rules to engage in circuit analysis with fixator-norator pairs are discussed, and numerous pitfalls in this line are specified. Finally, two design examples are worked out that clearly demonstrate the capability and power of the proposed technique for biasing any analog circuit.

1. Introduction

Biasing of large and complex analog circuits has always been a great challenge for the designers. The challenge is normally in two areas. First, to get the number of iterations minimized and make the convergence possible and quick; second, to move to the right regions of operations for the active components so that the output signals could get far from being distorted or clipped. Both problems become complex as the number of active devices increases, the design requirements become tighter, and more efficient designs are in demand. One difficulty appears to be the lack of separation between the linear and nonlinear components in the circuit during the process. Traditional biasing techniques deal with the circuit as a whole, with no break or circuit partitioning; hence, the complexity quickly increases as the circuit grows [1, 2].

Recently a new biasing technique has been introduced that somewhat breaks the tradition [3–5]. It starts with biasing nonlinear components (say transistors) individually and makes each transistor to become DC isolated and to operate at its own selective operating point (OP). Here each transistor is biased locally without interfering with other components in the circuit. A major advantage in using this technique is to deal with nonlinearity locally and to avoid any nonlinear operation in the original circuit, as a whole. One may even claim zero nonlinearity being involved in this situation. This is because biasing individual transistors to operate at their desired OPs is just a matter of local sourcing.

In the method presented by Verhoeven et al. [3] the design of amplifiers is carried out linearly and in AC domain. Here the circuit biasing—performed at the end of the design—is reduced to just the transistors' biasing, which is again a local sourcing of the transistors, so that they can get to the OPs intended for each without being interfered by other DC sources. This technique makes each transistor DC isolate and it uses controlled sources for local adjustments. Although the controlled sources are later removed but initially they cause timely iterations until they are eliminated. In [4] this author presents a somewhat similar biasing method, called "local biasing." Despite the technique used in [3], here no controlled source is used for the biasing purposes. In addition, it is shown in [3, 4] that in each locally biased port only one DC source delivers power to the device and the other source is sitting idle. As demonstrated, this property helps to cut down the number of biasing sources

in the circuit by half, and the other half can be replaced with coupling capacitors. This is of course for the case that the biasing voltage sources are sitting idle. Similarly, in the case of the current sources delivering zero power inductors can replace them.

With all advantages and tremendous simplifications that the local biasing offers for biasing complex analog circuits [4, 5], one major difficulty still remains to be addressed. The question is how to deal with those "scattered supplies" used in the circuit due to the local biasing? As expected, each bipolar transistor needs four (voltage and current) sources to get locally biased. There are known circuit techniques [3, 6] that are used to deal with the problem. The method proposed by Verhoeven et al. [3] uses shifts and other source transfer techniques to reduce the sources and push them to specific locations [6]. Techniques such as voltage dividing, source shifting, and current sourcing, and mirroring help to reduce the number of DC supplies and push them to the right locations in the circuit. As expected, the method is more gradual and long and tedious procedures used often reduce the attraction and practicality of these methods. In addition, by implementing these procedures, there is no guaranty to ensure an optimal or a desirable solution.

In our methodology we are offering a new source transformation technique that despite the conventional one it does the entire process in a single step. We may begin the design of an analog circuit by choosing a desirable circuit topology, first. In case the design uses discrete components the nonlinear components (transistors) represent the drivers that must be biased to selected OPs. Now, because the regions of operations for the drivers are specified we can simply replace the transistors by their small signal linear models, bypass the entire nonlinearity, and go directly for the design of the (linearized) AC circuit. Note that because no DC analysis is attempted yet then no DC supplies are specified. Indeed in our methodology the circuit biasing is pushed to the end and it begins when the AC design is successfully completed, and the regions of operations (or simply OPs) for the drivers are specified, for maximum output swings and minimum distortions.

On the other hand, if the target circuit is an integrated circuit then we are facing with two types of nonlinear components: the drivers and the supporting components, such as current sources, current mirrors, and active loads. Similar to the previous case here we also start the design for AC signals. We replace the drivers with their small signal linear models at the desired Ops, and the other nonlinear supporting components are also replaced with their linearized equivalent impedances for small signals (such as a dynamic load resistance ro), as specified by the design criteria.

Up to this point, the process of analog circuit design has followed a conventional routine. We still need to know how to design the biasing of the circuit to fulfill the following two conditions: (i) have the DC (voltage) supplies with specified values located at their selective locations in the circuit, and (ii) have the drivers, as well as the supporting components, biased at their selective OPs. In most cases the location of supplies, such as V_{CC} and V_{DD}, and their values are predetermined for the design. In such cases the question is how to fulfill both set of requirements: (i) have the DC power supplies with specified values and specified locations in the circuit, and (ii) achieve the AC design requirements without any nonlinear iterations and with minimum design efforts? As discussed earlier, one way to do this is to design for the AC case first with the load and node impedances required for the design and do the biasing later. Traditional biasing methods start with fixed supplies at fixed locations in the circuit; whereas our method is to start biasing the individual transistors and move the biasing sources to the desired locations later. With the first method nonlinearity is unavoidable and because of the fixed AC models of the transistors fulfilling all design specifications is hard and time consuming. However, the difficulty with the individually biasing the transistors is to end up with too many (voltage and current) sources in the circuit, and unless we move all the sources to one or two designated locations, for the DC supplies, the job is incomplete. Again, what makes the proposed technique more attractive is the fact that in one step this move takes place and those one or two supplies get replaced for all the sources used for the individual transistors biasing. This certainly eliminates all those conventional source reduction and transformation as well.

The tool we are going to use to achieve our goal is fixator-norator pair (*this is similar to nullor pair except that a fixator-norator pair can accept sources*). It is shown that while the use of fixators helps to keep the critical biasing specs unchanged the matching norators actually find the values and the locations of the DC supplies. As it is shown, the use of fixator-norator pairs (or actually nullor pairs) is temporary here. The pair actually works as a catalyst and get removed from the circuit after the DC supplies are allocated. This suggests that, there is no need to replace the norators with actual devices (such as Op-Amps or OTAs). In fact, because of the temporary nature of the nullor pairs ideal controlled sources of type VCVS, VCCS, CCVS, and CCCS with high gains, approaching infinity, will perfectly do the job.

The use of nullor pair in analog designs has been very extensive [7–9]. Telelo-Cuautle also introduces biasing techniques for amplifiers at nullor levels [10, 11]. These techniques use nullor pairs and their governing rules to simplify the biasing. The pairs are then replaced with transistors or Op-Amps for the final design. Haigh, Clarke, and Radmore introduce a new framework for linear active circuits that use a special type of limit variable in the circuit admittance matrix. This variable being initially finite can approach infinity resembling high gain Op-Amps or transistors as nullors [12–14]. Claudio Beccari [15] also uses the nullor concept eloquently to find and allocate the transmission zeros in a circuit.

The method proposed in this paper is using nullor pairs (in form of fixator-norator pairs) quite differently. The pairs are used as tools to reallocate the DC supplies and conduct the DC power to the transistors and then disappear. It is only during the circuit analysis and simulation that, in a fixator-norator pair, the fixator is used to sense certain specified current or voltage (OP) in the circuit. This sensing then tries

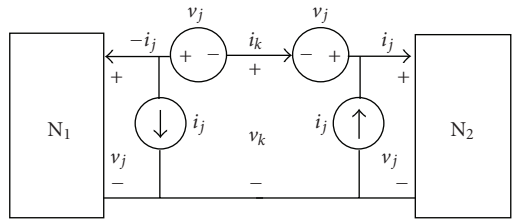


Figure 1: Port nullification procedure.

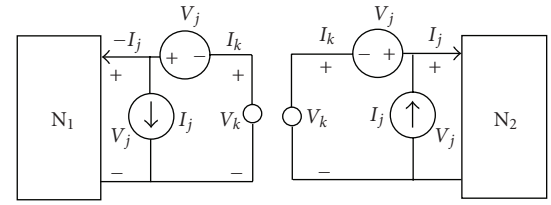


Figure 2: Two networks N_1 and N_2 disjointed at port $k(v_k, i_k)$ and each terminated by a nullator.

to control a voltage across or a current through the associated norator. As a result, the voltage or current found for each norator is, in fact, an indication that indeed a DC (voltage or current) supply exists at the location that has caused the biasing. We repeat this for all critical ports (with biasing specified) until all supplies are allocated.

The rest of the materials in this paper are arranged as follows. Section 2 is on fixator-norator pairs. The behavior and properties of fixators and norators are discussed here. Local biasing is reviewed in Section 3. Models of locally biased MOS and bipolar transistors are given in this section. The use of fixator-norator pairs in global biasing of analog circuits is investigated in Section 4. Rules governing the fixators and norators are also discussed here. Section 5 is on implementation aspects of fixator-norator pairs for biasing. An algorithm, developed in this section, provides a systematic procedure into the design of biasing for analog circuits. Two examples are worked out in Section 6 that use the methodology introduced here as the basis for the biasing design of analog circuits. Finally, Section 7 concludes our discussions on analog circuit design with emphasis on biasing.

2. Nullification and Fixators

First we need to define some terms that are used in this paper [4]. Also, all our discussions here apply to DC signals and biasing, unless stated otherwise.

2.1. Nullification

Null port. consider two powered networks. (*a powered network is a network with at least one power supply*). N_1 and N_2 connected through a port $k(v_k, i_k)$. Port $k(v_k, i_k)$ is said to be null if both voltage v_k and current i_k are zero.

Port nullification. consider two powered networks N_1 and N_2 connected through a port $j(v_j, i_j)$. Port $j(v_j, i_j)$ is nullified if it is augmented, from both sides (N_1 and N_2), by current sources i_j and voltage sources v_j so that a null port $k(v_k, i_k)$ is created as a result, as shown in Figure 1.

Apparently, there will be no change in the currents and voltages within both networks N_1 and N_2 if we disjoint the two networks at port $k(v_k, i_k)$ and connect each to a nullator, as illustrated in Figure 2. This results in a new finding as follows.

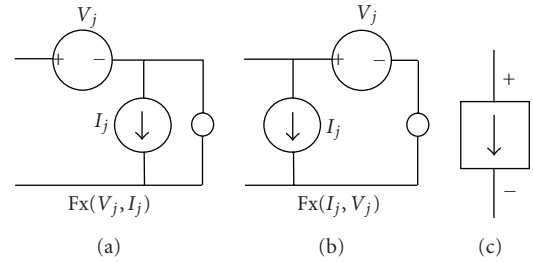


Figure 3: (a) Voltage Fixator; (b) current Fixator; (c) the symbol.

2.2. Fixator and Fixator Modeling.

Fixator. a two-terminal component (*a component here can be of any size two terminal network*). in a circuit is called a Fixator if both voltage across the component terminals and the current through the component represent two independent sources.

One way to create a fixator with voltage V and current I is to have a nullator parallel with a current source I and in series with a voltage source V, as depicted in Figures 3(a) and 3(b). Note the difference between the two fixators Fx(V, I) and Fx(I, V). In Fx(V, I) the voltage source V provides (or consumes) power and the current source I is sitting idle; whereas, in Fx(I, V) the current source I provides (or consumes) power and the voltage source V is sitting idle.

Now, referring to Figure 2, we realize that both ports in networks N_1 and N_2 are terminated with Fixators. This leads to the following property.

Property 1. Consider two powered networks N_1 and N_2 connected through a port j (v_j, i_j), and no other external component is connected to N_1 or N_2. There will be no (current or voltage) change in the network N_2 if N_1 is replaced with a fixator Fx(v_j, $-i_j$), as shown in Figure 4. Similarly, there will be no change in the network N_1 if N_2 is replaced with a fixator Fx(v_j, i_j).

Before we move on, it is important to compare the fixator Fx(v_j, $-i_j$), replacing N_1 in Figure 4, with N_1 Thevenin or Norton equivalent circuits, which also can replace N_1. This topic is fully discussed in [16] and it is briefly given here. In fact, instead of replacing N_1 with its Thevenin or Norton model we can alternatively replace it with a more generalized model known as nH (nullified Hybrid) model [16], where instead of one, voltage or current, source an nH-model has

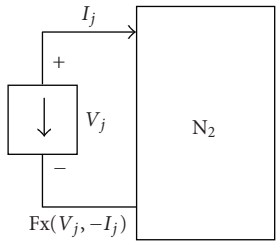


Figure 4: Network N_2 terminated with a Fixator.

one voltage and one current source combined as shown in Figure 5(a). It is easy to show that both Thevenin and Norton equivalent circuits are in fact two special cases of nH-models [16] for a two-terminal network. Note also that an nH-model can model a nonlinear two-terminal network as well, as represented in Figure 5(b). Now, we are closer to the result we aimed at! In comparing the network N_2 in Figure 5(b) with that in Figure 2 we realize that the only difference between the two is that in Figure 2 the network N_1, with no source, is replaced with a nullator. This may look rather strange and seems to violate the circuit (KVL and KCL) laws! However, it makes sense if we consider the situation as a "snap shot" of the circuit. Here is the difference: the circuit in Figure 5 allows us to change the component values inside N_2 and observe the changed made at the port J, where as in Figure 2, we are not allowed to make any changes inside N_2 that causes V_j or I_j to change. By the way, this means that changes inside N_2 that do not affect the port values are allowed. One may then ask, how can we change the component values inside N_2 and still use Figure 2 for modeling with the expectation that V_j and I_j remain unchanged (*we are of course talking about nonseparable networks here*)? The answer to this question is as follows. When we change the component values inside N_2 and expect not to change anything at the port J then we need to have another component, $p(V_p, I_p)$, inside N_2 with unspecified voltage and current so that the changes, expected to appear at the port J, are "transferred" into the changes in V_p and I_p. This component, p, is called norator. Therefore, to have the circuit laws in place the fixators (nullators) and norators must appear in pairs in a circuit.

2.3. Rules Governing Fixators and Norators. Here are some of the useful properties of fixators and norators. First, notice that a nullator is a special case of a fixator represented by Fx(0, 0), where both the device voltage and current are zero. Therefore, like nullators, fixators must pair with norators in order to have computational stability in a circuit. We should also remember that a fixator represents a current source and a voltage source combined. For instance, a current source in series with a fixator may violate the KCL, and a voltage source in parallel with a fixator may violate the KVL. In general, a cutset of fixators with or without current sources may violate the KCL and a loop of fixators with or without voltage sources may violate the KVL. Here are some other properties of the pair, as discussed in [17]

(i) The power consumed in a fixator $Fx(V, I)$ is $P = V*I$.

(ii) A resistance R in series with a $Fx(V, I)$ is absorbed by the fixator and the fixator becomes $Fx(V_1, I)$, where $V_1 = V + R*I$. A resistance R in parallel with a $Fx(V, I)$ is absorbed by the fixator and the fixator becomes $Fx(V, I_1)$, where $I_1 = I + V/R$.

(iii) A current source I_S in parallel with a $Fx(V, I)$ is absorbed by the fixator and the fixator becomes $Fx(V, I_1)$, where $I_1 = I + I_s$.

(iv) A voltage source V_s in series with a $Fx(V, I)$ is absorbed by the fixator and the fixator becomes $Fx(V_1, I)$; where $V1 = V + V_S$.

(v) A current source in series with a norator absorbs the norator; a voltage source in parallel with a norator absorbs the norator. In addition, a current source in parallel with a norator is absorbed by the norator; a voltage source in series with a norator is absorbed by the norator.

(vi) A resistance in series or in parallel with a norator is absorbed by the norator.

(vii) A norator in series with a fixator $Fx(V, I)$ becomes a current source I; a norator in parallel with a fixator $Fx(V, I)$ becomes a voltage source V.

3. Local Biasing

Local biasing (LB). A port is locally biased if it is nullified. Likewise, a network or a component is locally biased if all its ports are nullified.

Apparently, local biasing of a network or a component is not unique. A port can be locally biased for any selection of an OP on its characteristic curve. This is also true for a component with multiple ports. Evidently, connecting a locally biased device to a network with no DC (voltage or current) supply does not change the biasing condition of the device. This simply means that a locally biased device does not need any external DC supply to keep it operating, while still responding normally to any AC signal.

Note that when we locally bias nonlinear components in a circuit, we in fact replace the DC power supplies with local sources that are only responsible to provide biasing/power to individual devices. This certainly gives a choice to a circuit designer to select his/her own operating regions for nonlinear devices in the circuit and bias them on spot without going through timely iterations, typically required for nonlinear designs. In fact when all the transistors. are locally biased there is no need to have DC supply for the rest of the circuit (*diodes and other nonlinear components may also be present in a nonlinear circuit, but for simplicity we refer all to transistors*). Therefore, by replacing the transistors with their linear models the circuit becomes entirely linear suitable for AC design. In addition, as discussed in [5], having local biasing reduces the DC power consumption to its minimum, and just enough to bias the transistors individually. This means that the signals in the rest of the circuit always remain AC and unaffected.

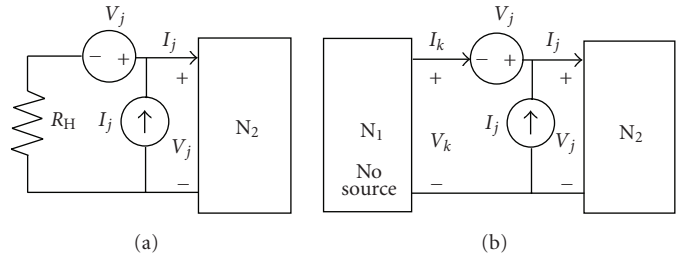


Figure 5: (a) nH-model for linear N_1; (b) nH-model for non-linear N_1.

Another important property of the local biasing is its locality and the possibility of (DC) circuit partitioning (divide and conquer) without affecting the integrity and the operation of the AC circuit. The local biasing allows the designer to locally tune the circuit and make changes in the operating regions of individual devices or a local subcircuit without disturbing the rest of the circuit. There are number of applications using this property of the local biasing, such as replacing some faulty transistors with new ones, and even with different types of transistors, for instance, changing BJTs to MOS transistors or vice versa.

Another application of the local biasing is in local testing and debugging of a complex circuit. For example, consider the circuit in Figure 6(a), where the MOS transistor M is malfunctioning because its output OP, at $Q(V, I)$, is at the wrong place on the characteristic curve (Figure 6(b)). To correct the situation we need to move the OP to the right on the curve, positioning it at $Q_1(V+dV, I+dI)$ as indicated in Figure 6(b). Here the local biasing will help us by augment M with two difference sources dV and dI, as shown in Figure 6(c), which causes the OP of the transistor to move from Q to Q_1 without affecting the rest of the circuit. Later, we may need to get rid of the sources, dV and dI, and move them inside N to get integrated with the rest of the DC supplies in the network, but that is a separate issue.

3.1. Component Biasing. Within the three major semiconductor components p-n junction diodes are one-port devices. Bipolar-junction Transistors are generally considered two-ports, but they can be turned into two one-port devices if Ebers-Moll or the Transport large signal model [1] is used. MOS transistors are considered three-port devices but only four sources are used to locally bias the device. This is because for the drain-source we need both ID and VDS sources to nullify the port; whereas for the gate-source and the substrate-source we only need VGS and VBS to nullify the ports, respectively. Figure 7 illustrates both an nMOS and a pMOS locally biased; however, for simplicity we are going to drop the substrate effect, VBS, later in our discussion. Similarly, Figure 8 shows an npn and a pnp transistors locally biased.

4. Global Biasing

With all advantages of the local biasing, one major difficulty remains unsolved, and that is the "scattered supplies" throughout the circuit. Although through circuit manipulations such as current mirroring, voltage dividing and source transformation the problem may be reduced to certain degrees but overall the problem of the scattered supplies in large circuits makes the design difficult, inefficient, costly, and mostly impractical.

In our approach, the problem is dealt with in a single step without any iteration or source shifting and transforming. In our design procedure we go through two major steps. In the first step we translate the design specs into the transistors' OPs using the local biasing. Then using the transistors' linear models we perform the linearized circuit design for AC signals. In the second step we try to remove the scattered sources caused by the local biasing and replace them with regular DC supplies in the circuit. Property 2 is very instrumental in this approach.

Property 2. A locally biased component within a network N can be removed and be replaced with nullators, one for each port, without affecting the DC currents and voltages within the network N.

Note that locally biasing of a component produces zero voltages and currents at its ports without affecting its internal currents and voltages. This simply means that, in an analog circuit when certain components are locally biased they can be separated and removed from the main circuit without imposing any change on the circuit operation or on the biasing conditions of those components themselves, provided that each disjointed port, from both sides is connected to a nullator (Figure 2). In addition, applying Property 2 to a nonlinear circuit removes the nonlinearities (transistors) from the circuit and makes them perform linearly.

4.1. Port Modeling with Fixators. Now, we are ready to go one step further by introducing Property 3.

Property 3. A two-terminal component in a circuit (linear or nonlinear), such as a p-n junction diode, that is biased by current I_D and exhibits a junction voltage V_D can be replaced with a fixator $\mathrm{Fx}(I_D, V_D)$ without causing any change in the currents and voltages within the rest of the circuit.

Property 3 directly results from Property 1. Note that $\mathrm{Fx}(I_D, V_D)$ models a diode only when the diode is in a

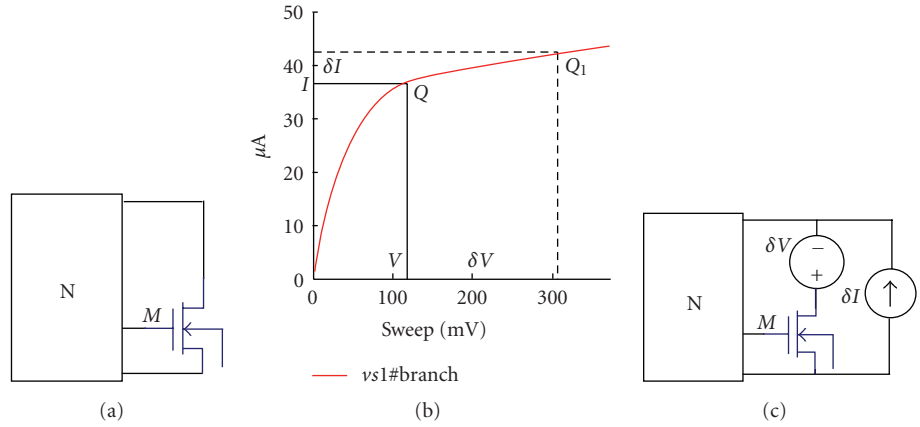


Figure 6: Partially locally biasing an MOS in a circuit.

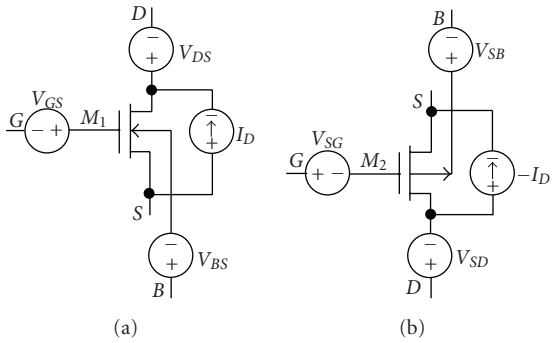


Figure 7: Locally biased (a) NMOS, and (b) PMOS transistors.

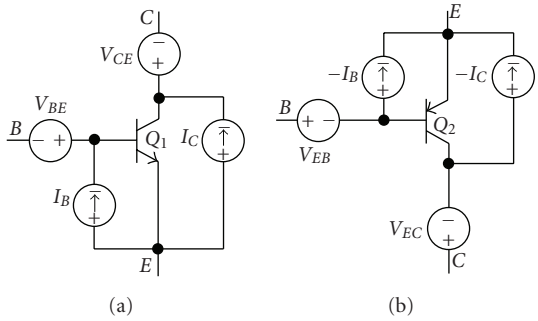


Figure 8: Locally biased (a) NPN, and (b) PNP transistors.

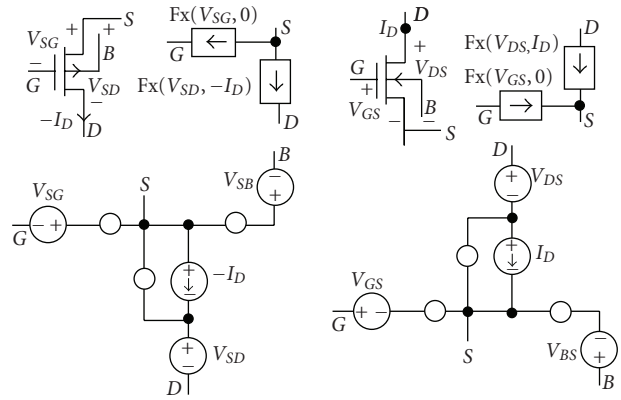


Figure 9: Fixator models of nMOS and pMOS transistors when globally biased for V_{GS} (V_{SG}), V_{DS} (V_{SD}), I_D, and V_{BS} (V_{SB}). Both symbolic and expanded versions are shown.

circuit and globally (not locally) biased with I_D and V_D. Property 3 can also be extended to include the components with multiple ports such as bipolar and MOS transistors. Figure 9 shows the fixator models of nMOS and pMOS transistors when they are globally biased for V_{GS} (V_{SG}), V_{DS} (V_{SD}), I_D, and V_{BS} (V_{SB}). Likewise, Figure 10 depicts the fixator models of npn and pnp transistors when they are globally biased for V_{BE} (V_{EB}), V_{CE} (V_{EC}), and I_C. Note that, similar to Property 2, applying Property 3 to a nonlinear circuit removes the nonlinearities (transistors) and makes the circuit perform linearly.

There is a clear distinction between Property 2 and Property 3 however. Property 2 shows how we can remove a locally biased port (component) and replace it with a nullator; whereas, Property 3 makes us replace a globally biased port (component) with a fixator. For Property 2 there is no DC power external to the locally biased components to worry about; whereas Property 3 indicates that there must be DC sources external to the components that provide the biasing for them. Where are these sources in the circuit!? So we need to concentrate more on Property 3 now. What is missing is how to produce the global biasing, capable of producing fixators—biased components—that follow the design specs (OPs).

Indeed the job turns out to be already resolved. Earlier we discussed the problem in a previous section; that is, for each fixator we need to assign one norator inside the (linear) circuit. We can assume that this norator acts as a placeholder for a DC supply in the circuit. What this basically means is that, when we replace a transistor with its fixator model, in exchange we are getting a ticket to assign a DC source in the circuit wherever we like, provided that this DC source is "reachable" by the fixator. This is indeed how the fixators

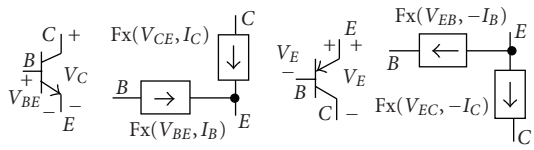


FIGURE 10: Fixator models of npn and pnp transistors when globally biased for V_{BE} (V_{EB}), V_{CE} (V_{EC}), and I_C (the expanded versions not shown).

(globally biased transistors) can manage to have the DC supplies in the circuit in the designated locations. We still need one more step to go! Up to this point the norators—the matching partners of the fixator-norator pairs—can be assigned to the locations that are reserved for the global DC supplies. But we still need to find the values of those supplies. This is done through the circuit analysis created up to this point. The circuit here is a linear circuit with fixator-norator pairs; there is no DC supply. in the circuit except each fixator does represent a fixed voltage and a fixed current source (*in partial DC biasing design cases it is possible to have DC supplies as well as fixator-norator pairs in the circuit*). When the circuit analysis is complete each norator in the circuit can be replaced either (i) with a voltage source equivalent to the voltage across the norator, or (ii) with a current source equivalent to the current through the norator.

This study still needs to address two questions. First, what can we do if the DC supplies (mainly the voltage sources) so obtained are beyond the conventional and standard values—12 V, 5 V, 3.3 V, and so on? In case of smaller voltage values techniques such as voltage dividers can help. For larger values, however, the solution may get more involved. An adjustment in the "power conducting" resistors is one possibility to do the job. Linearity scaling and other linear methods are also useful to adjust the supplies to the conventional values. The second question deals with having the number of fixators and norators unequal. Typically the number of fixators exceeds the number of norators. For example, for a three-stage amplifier with three driver transistors we might have as many as six fixators; whereas one power supply, V_{CC}, can be represented by only one norators. There are different approaches to handle this problem, and some are still under investigation. One approach is to assume multiple virtual supplies that can later be easily combined into one or two actual supplies. Another approach is to reduce the number of fixators by removing "noncritical" ones from the list.

In general there are two ways we can think of a good biasing design for an analog circuit. One way is to take care of each and every individual transistor in a circuit and bias it to operate at its best OP. With this approach we definitely need as many fixators as there are ports for the nonlinear devices. A second approach, however, is to classify the nonlinear ports as "critical" (*a port is critical if the port voltage or its current must meet the design specifications*) and "noncritical" ports, and assign fixators only to those critical ports. We will be more covering this subject in the next section.

5. Implementation

Design of high performance analog circuits can be a complex and often a multistage processes—typically noise, distortion, gain, bandwidth, and biasing. One approach to simplify the design and cut loops and feedbacks between the stages is to use as much orthogonally between stages as possible [3]. In our proposed method, this orthogonally is practiced between the circuit performances and the biasing of the components, or simply between AC and DC circuit designs. The first task is to design for the circuit performances, mainly noise, signal power, and bandwidth [3]. In this study, we only deal with the biasing part of the design. A full discussion on the performance design can be found in [3].

We can in fact start with a certain circuit topology suitable for our design, then select regions of operations for the transistors so that the design requirements can be fulfilled. After designing the circuit for AC signals the time comes to bias the circuit with DC supplies (voltages and currents) so that those selected operating regions, and therefore, the design requirements are best met. Section 5.1 provides a systematic procedure to do the circuit biasing using the proposed new approach.

5.1. Algorithm 1. *Preparation*. given the design specification, we begin with the performance design by selecting a working circuit topology. We then choose the right OPs for the drivers so that we can best meet the design requirements. In the next step, we replace all the transistors with their small signal linear models so that the entire circuit becomes linearly ready for the AC design. Upon the completion of the performance (AC) design, we begin doing the biasing design as follows.

(1) Assign one fixator, carrying the biasing spec, to each "critical" transistor port. Also assign one norator to each to-be-specified DC supply in the circuit. If they do not match, reduce the number of critical ports so that the number of the fixators and norators becomes equal.

(2) Next, pair each norator with a fixator in the circuit. This step is rather critical and needs some care (see "Singularity and Circuit Divergence"). In general, any pair must work (although may not be optimal), except for the cases where a fixator is not sensitive to the changes in the norator.

(3) Assign one controlled source to each pair of fixator-norator. It is permissible to assume an ideal controlled source with very high gain; this is because these controlled sources will disappear later, leaving the DC supplies behind. A controlled source can be of type VCVS, VCCS, CCVS, or CCCS; the choice depends on the sensitivity issue, stated in step 2.

(4) Solve the linear circuit equations so prepared and the DC solution (simulation) provides the currents and voltages for the circuit components including those for the norators, represented by the controlled sources.

(5) Remove all the controlled sources from the circuit and replace each with either a voltage source, V_j, or a current source, I_j, where V_j and I_j are the voltage and current found for the controlled source (norator).

Now that we are done with the biasing design there are still issues that must be dealt with before we leave the subject. First, as mentioned earlier, the equivalency of number of fixators (nullators) and norators is necessary to solve the circuit equations but it is not sufficient. The issue is related to the independency of the circuit (KCL and KVL) equations. As we discussed in Section 2, a fixator carries the properties of a voltage source and a current source combined; therefore, it cannot be in a loop with voltage sources, neither it can be in a cutset of current source.

The second problem is the fact that with each transistor added to the circuit at least two fixators (for input and output ports) are added to the linear circuit; whereas the number of supplies are usually much more limited. One way to deal with this problem is to have a selective number of the (transistor) ports to be critical; only those that directly reflect on the performance and effectiveness of the design. For example, we may consider the output voltage swing of an amplifier as a critical component of the design for reducing the distortion. Similarly, we may assume the input stage of an amplifier as a critical stage for noise reduction. As another example, suppose we are designing a multistage amplifier where the first stage is presumably going to be a preamp and the possibility is that the voltage swing for this stage may stay within, say 100 mV. Therefore, as long as the (driver) transistor operates in the active region the chances are that it may always stay in that region, even during the AC operation; hence, this transistor may be labeled as noncritical. Now, it is a matter of the designer skill to separate critical ports from those non-critical ones and assign fixators only to those critical.

Another solution to the problem is to prioritize. A skilful designer can always prioritize the design specs so that the most critical criteria come first and he/she can stop when the numbers match the number of unknown supplies. There are also ways to increase the number of norators to match the number of fixators. For instance, in designing a multi-stage amplifier with only one DC power supply, say V_{DD}, we can initially start with multiple supplies—one for each stage, for example—and then combining to one by doing certain adjustments. Techniques such as using voltage dividers and current mirrors can also help in increasing the number of norators.

Another issue in the circuits with fixator-norator pairs is, in fact, the lack of sufficient sensitivity between the fixator and the norator in a pair. If a variation in a norator does not reach to the corresponding fixator in a pair the link is lost. For example, if there is no feedback from the output stage to the input stage, in a multi-stage amplifier, then a fixator at the input stage (say, V_{BE} of the input driver) cannot pair with a norator at the output stage. In addition, if a norator and a fixator are placed in a positive feedback divergence might happen. This subject is discussed next.

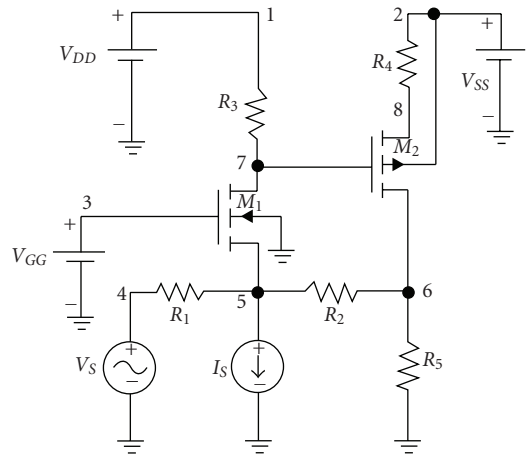


Figure 11: Initial configuration of an MOS feedback amplifier.

5.2. Singularity and Circuit Divergence. There is a word of cushion here; our experiments show that in large circuits with numerous critical specs, and hence numerous fixator-norator pairs, there is a possibility that singularity or circuit divergence might happen. In a way, the problem is related to the independency in the circuit equations (KVL and KCL), and possible inequality that might happen between the number of independent norators and fixators, although they might have originally set equal. The problem is often caused by violating the rules for norators and fixators, explained in Section 2. For example, a fixator should be treated as both a voltage source and a current source; therefore, it cannot be in a loop of all fixators and voltage sources; or a fixator cannot be in a cutset of all fixators and current sources. Other difficulties may be the lack of enough sensitivity between a fixator and a norator in a pair. If a change in a norator does not reach to the corresponding fixator the link is broken. For example, in a design, if there is no feedback from the output stage to the input stage, then a fixator at the input stage (say V_{GS} of the input driver) cannot pair with a norator at the output stage. Finally, if a fixator and a norator are positioned in a positive feedback loop divergence might happen.

6. Examples

The following examples demonstrate the biasing design flow for two cases. Example 1 is a two-stage MOS amplifier with feedback where the biasing of both transistors is specified for the design. Example 2 is also a two-stage BJT differential amplifier with feedback, entirely designed in [3].

Example 1. Consider a two-stage MOS amplifier with feedback [2], shown in Figure 11. The schematic provides the topology of the design but the circuit values are left to be specified. We start the design by selecting the desirable OPs for the MOS transistors to meet the design specs. The selected values for V_{GS}, V_{DS}, and I_D, for both transistors, are provided in Table 1 (for simplicity the body effect is ignored).

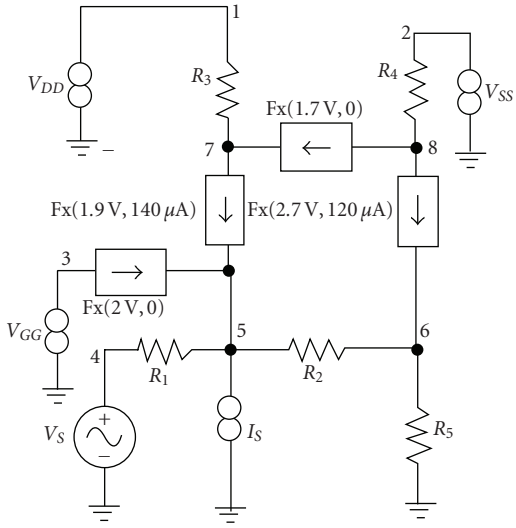


Figure 12: The linear DC modeling of the amplifie, using the fixator-norator modeling concept for the MOS transistors.

Table 1: Specified design values for the MOS transistors.

Devices	V_{GS}	I_D	V_{DS}
M_1	2.0 V	140 μA	1.9 V
M_2	−1.7 V	−120 μA	-2.7 V

Given the selected OPs our next job is to create fixator models that represent fixed OPs for the transistor port (Alg.#1). These models are then substituted for the transistors in the circuit changing it to a linear circuit (Figure 12). The next step is to identify the locations of the DC supplies for the general biasing. These locations could be anywhere in the circuit as long as the sensitivity issue (discussed in Section 5) is taken care of. However, looking at Figure 11 we realize that the supplies' locations, namely, V_{DD}, V_{SS}, V_{GG}, and I_S, are already determined; hence, only their values remain to be evaluated. To do this we assign one norator to each supply location, as indicated in Figure 12. Note that it so happens that the number of fixators and norators match in this case and no other supply is present in the circuit. This is a special case of "total" circuit biasing (Alg.#2).

We are now ready to solve the linear circuit with fixator-norator (or rather nullor) pairs. SPICE cannot directly solve the circuit equations with nullors; but if we carefully replace each pair with an ideal high-gain dependent source the simulator solves the circuit equations and provides the currents and voltages for the elements in the circuit, including those for the norators (Alg.#3). Note that the use of high-gain dependent source for this purpose is just temporary (for the lack of nullor model representation in the simulator) and do not appear in the actual design. After the voltages and currents for the norators are found it is then our choice to replace each with either a voltage source or a current source (Alg.#4).

Up to this point the biasing design is theoretically over; however, at this stage we may need to reassign or adjust other circuit parameters such as the power-conducting resistor values (conducting the DC power to the transistors) in the circuit. This is often required in cases when the DC voltage supplies, resulted from the circuit simulation, are not within the conventional rating.

Another point of caution is that this fixator modeling of transistors is developed only for DC biasing. Thus, for AC analysis and design of analog circuits we still need to replace the transistors with their linear small signal models. The advantage of fixator modeling, however, is that it directly targets specified OPs for the devices; hence, to get the linear models for the transistors all we need is to refer to the device characteristics at those OPs.

Next, for the resistor values R_1 = 10 K, R_2 = 50 K, R_3 = 15 K, R_4 = 3 K, and R_5 = 20 K we simulate the design using WinSPICE. A part of the simulation code is shown below with high gain dependent sources.

```
vdn     7     52     1.9
Vgn     3     32     2.9
vdp     8     62     2.7
vgp     72    7      1.7

in      7     5      140u
ip      8     6      120u
fs      5     53     vnd   1000MEG
hgg     3     0      vng   1000MEG
hss     2     0      vpd   1000MEG
hdd     1     0      vpg   1000MEG
```

Note that eight distinct values for the fixators are provided (two zeros are not shown). For the norators, the first one is a CCCS and the next three are CCVS, all with gain of 10^9. The result of the simulation is also given below.

```
TEMP = 27 deg C
DC analysis... 100%%
v(1) = 5.140025e+00          VDD
v(2) = 5.100133e+00          VSS
v(3) = 4.040102e+00          VGG
vis#branch = 4.399384e-05    IS
WinSpice 1 ->
```

Evidently, with slight adjustments to the resistors that conduct DC current to the transistors (R_1 to R_5) we can get V_{DD} = 5 V, V_{SS} = 5 V, V_{GG} = 4 V and, I_S = 44 μA for this design (Alg.#5). This completes the DC biasing of the feedback amplifier. Finally, the MOS transistors must be sized to deliver the currents required. For fixed L = 2 μm and with a simple calculation we get Wn = 20 μm and Wp = 60 μm for the transistors.

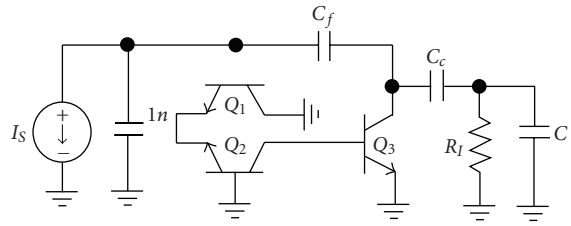


Figure 13: A three-stage amplifier topology after going through the performance AC design [3].

Example 2. This example presents a negative feedback amplifier, fully explained in [3]. Figure 13 shows a schematic of the amplifier after it has gone through the performance design in the three areas: noise reduction, clipping/distortion reduction, and high loop-gain-poles-product (*for details please refer to Chapter 10 in* [3]). To perform the biasing of the circuit we need to first specify the values for the DC supplies and their locations in the circuit. Next, we need to select the OPs for the transistors so that they can fulfill the design specs. For the actual power supplies, we choose two DC sources of $4\,V$ and $-4V$, as selected in [3]. For other components conducting the DC power to the drivers we notice from the amplifier performance design (Figure 13) that the followings must be taken care of.

(i) The emitters of Q_1 and Q_2 must be driven by a high impedance current source, I_e.

(ii) The base of Q_2 must be driven by a low impedance voltage source, V_{b2}.

(iii) The collector of Q_1 can be driven directly by V_{CC}.

(iv) The collector of both Q_2 and Q_3 must be driven by high impedance current sources I_{S2} and I_{S3}, to maximize the gain.

(v) The base current of Q_1 can be provided through a feedback resistor R_f (*the resistance R_f is in the bias loop and part of a required AC filter*).

For this particular design we choose the collector-emitter voltages of the transistors (v_{ce2} and v_{ce3}) as the "critical" design values, except for V_{ce1} of Q_1, which is "non-critical," directly connected to V_{CC}. Also all three collector currents i_{c1}, i_{c2}, and i_{c3} are also considered "critical. Table 2, columns 1 and 2, provides all five critical values for the OPs and also all five fixators that keep the critical values fixed during the design (Alg.#1). Column 3 shows the matching norators replaced with appropriate voltage source, current sources and one with feedback resistor (Alg.#2). Figure 14 is extracted from Figure 13 after the fixator-norator pairs, specified in Table 2, are added to the circuit.

Below is a piece of the WinSPICE program code simulating the DC biasing of the amplifier. Note that each fixator-norator pair is simulated by a very high gain controlled source (namely, VCVS, CCVS, VCCS, CCCS, and VCCS in sequence) (Alg.#3).

Table 2: Bias design specs and fixator-norators.

Critical specs	Fixator representations	Norator representations
$I_{C1} = 0.1$ mA	Fx(0, 0.1 mA)	R_f
$V_{CE2} = 0.67$ V	Fx(0.67 V, 0)	V_{B2}
$I_{C2} = 0.5$ mA	Fx(0, 0.5 mA)	I_E
$V_{CE3} = 2.2$ V	Fx(2.2 V, 0)	I_{S3}
$I_{C3} = 3.6$ mA	Fx(0, 3.6 mA)	I_{S2}

Table 3: Component values for the specified biasing.

$R_f = 1.53$ MEGΩ
$V_{B2} = 0.677$ V
$I_E = 0.607$ mA
$I_{S3} = 3.601$ mA
$I_{S2} = 0.523$ mA

```
ic1   2    a    DC      1.0e-04
e1    4    51   a   2   1000MEG
vce2  c    7    DC      0.67
hb2   Vb2  0    vce2    1000MEG
ic2   3    c    DC      0.5m
ge    7    11   3   c   1000MEG
vce3  e    0    DC      2.2
fc3   21   4    vce3    1000MEG
ic3   4    e    DC      3.6m
gc2   12   3    4   e   1000MEG
```

The results from the WinSPICE simulation are shown and listed in Table 3 (Alg.#4).

```
TEMP = 27 deg C
DC analysis... 100%
(v(4)-v(5))/vf#branch = 1.528640e+06
vb2 = 6.770538e-01
ve#branch = 6.068945e-04
vs3#branch = 3.601024e-03
vs2#branch = 5.229127e-04
WinSpice 6 ->
```

Finally, we remove the controlled sources (representing the fixator-norator pairs) from the circuit and replace each with the computed voltage source, current sources, and one feedback resistance (Alg.#5). The final amplifier so designed is depicted in Figure 15 (*for simplicity the current sources are presented in their ideal form in Figure 12. A detailed current sourcing and mirroring can be found in* [3]). As expected, the resulted DC sourcing matches with those in [3]; here obtained much quicker.

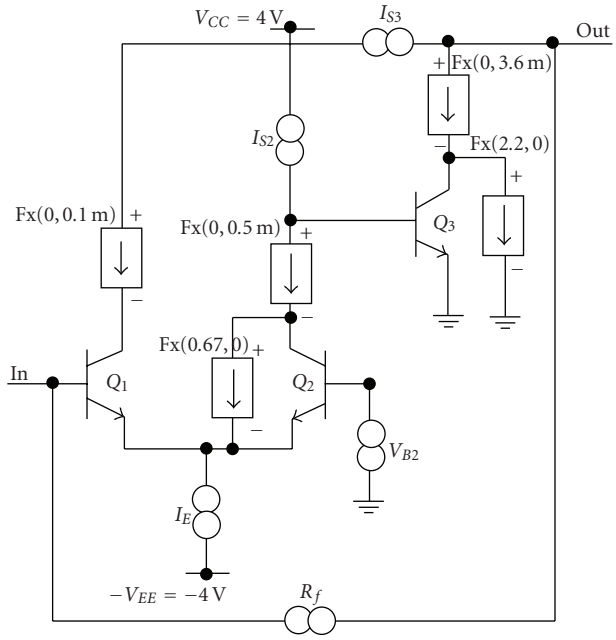


Figure 14: The three stage amplifier with fixator-norator pairs indicating the biasing design specs.

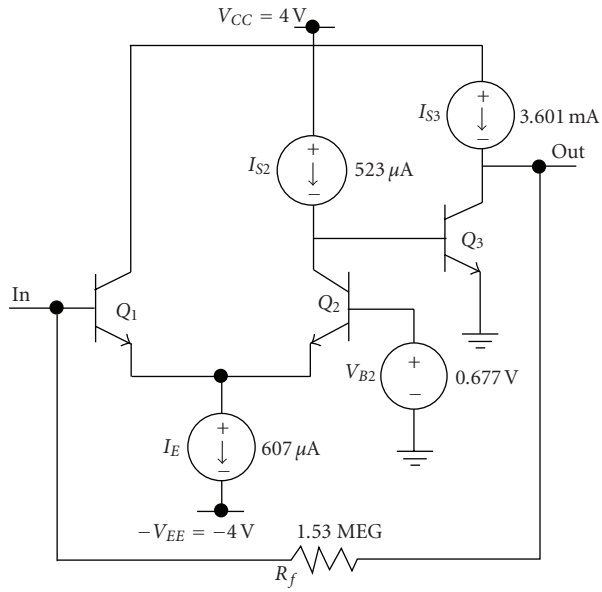


Figure 15: The three stage amplifier with complete biasing.

6.1. Some Potential Impacts of the Proposed Methodology. (this Discussion Was Suggested by One of the Reviewers.) We believe that the proposed methodology will have a tremendous impact to the research, to circuit designers, and to analog circuit design as a whole. The changes and effects are shown to be so vast and extensive that it can be interpreted as the beginning of a new era in analog circuit designs. Here are some of the evidences, discussed earlier.

(i) No matter how complex, the nonlinearity in analog designs is entirely removed and is placed on the linearized equivalent circuit.

(ii) If selected, each transistor (nonlinear component) is individually biased to the selective and desirable OPs.

(iii) It is shown that local biasing minimizes the DC power consumption in the circuit. In general, the methodology can be used to reduce the DC power consumption.

(iv) The proposed method is not a destructive one. A mixture of the traditional and the new method is possible for the design, and in fact is recommended for circuit modification and debugging.

This is just the beginning of the change; because we are only dealing with DC analysis and design of analog circuits. The possibility of applying the methodology, such as fixator-norator pairs, in AC design of analog circuits is quite high, and under investigation.

What it brings for a designer is simplicity, time, and management. It brings simplicity because no matter how complex the circuit might be it can be linearized. The designer can save time because by linearization he/she has entirely removed the nonlinear iterations from the operations, typically needed for nonlinear circuits. The designer is in full control and management because he/she is not facing with a complex network of mixed linear and nonlinear components but individual transistors to assume the right OPs, getting the liner models for the transistors and finally solve for the entire linear circuit for results. Because of the exact and selective environment provided the designer is capable to accurately calculate for possible distortions, noise, bandwidth, power, and other design attributes.

The impact on the research and on analog circuit design as a whole is also very high and promising. As mentioned earlier, work still has to be done on the AC aspects of the design using the new methodology. Getting "scattered supplies" within the circuit to move them to the designated locations with assigned values, in a single step, is by itself a great achievement and worth pursuing further.

Another important feature of the proposed methodology is its unrestricted use of the special biasing for any type of analog circuits. As we have always referred to "analog circuit," the method is applicable to any kind of analog circuit, such as amplifiers, filters, oscillators, and modulators. For example, consider designing a single frequency oscillator for a specified frequency, quality factor, gain, noise control, port impedances, and so on. The task of the designer starts with design of the linear equivalent circuits for the oscillator; including the tank circuit, tuned amplifier, and the buffers. Certainly, the linear equivalent models are substituted for each and every nonlinear component in the circuit until the linear circuit is fully designed, simulated, and verified to meet the specs. Now it is time to do the biasing! As discussed earlier, this phase of the design is in two parts: (i) local biasing of the nonlinear components to meet the "critical" specifications, and (ii) using fixator-norator pairs to move

the scattered sources to those locations designated for the power supplies for the oscillator.

7. Conclusion

A new approach is presented for the biasing design of analog circuits. The main features of this approach are (i) circuit linearization for the design; (ii) individual component biasing; (iii) DC power reduction; (iv) full management of the design to meet the specifications. With specified operating points selected for the driving transistors and fixing them during the design process we can get the supplies needed for the biasing and at the locations designated by the designer. In this approach, the fixators (nullators plus sources) are used to keep the critical biasing values unchanged; whereas the norators are used to allocate the DC supplies, or adjust and modify their values if necessary. It is important to note that the use of fixator-norator pair in the present method is temporary; the pairs disappear after the actual supplies are substituted for the norators. Two design examples are worked out, which clearly demonstrate the power and simplicity of the methodology.

Acknowledgment

The author would like to thank Ms. Golli Hashemian for her valuable editing of the manuscript.

References

[1] R. C. Jaeger and T. N. Blalock, *Microelectronic Circuit Design*, McGraw-Hill, New York, NY, USA, 3rd edition, 2008.

[2] R. J. Baker, *CMOS, Circuit Design, Layout, and Simulation*, IEEE Press, Wiley Interscience, New York, NY, USA, 2nd edition, 2008.

[3] C. J. Verhoeven, A. van Staveren, G. L. E. Monna, M. H. L. Kouwenhoven, and E. Yildiz, *Structured Electronic Design: Negative-Feedback Amplifiers*, Kluwer Academic Publishers, Dordrecht, The Netherlands, 2003.

[4] R. Hashemian, "Analog circuit design with linearized DC biasing," in *Proceedings of IEEE International Conference on Electro Information Technology*, pp. 285–289, East Lansing, Mich, USA, May 2006.

[5] R. Hashemian, "Designing analog circuits with reduced biasing power," in *Proceedings of the 13th IEEE International Conference on Electronics, Circuits, and Systems (ICECS '06)*, pp. 1069–1072, Nice, France, December 2006.

[6] T. L. Pillage, R. A. Rohrer, and C. Visweswariah, *Electronic Circuit & System Simulation Methods*, McGraw-Hill, New York, NY, USA, 1995.

[7] R. Kumar and R. Senani, "Bibliography on Nullor and their applications in circuit analysis, synthesis and design," in *Analog Integrated Circuit and Signal Processing*, Kluwer Academic Publishers, Dordrecht, The Netherlands, 2002.

[8] H. Schmid, "Approximating the universal active element," *IEEE Transactions on Circuits and Systems II*, vol. 47, no. 11, pp. 1160–1169, 2000.

[9] E. Tlelo-Cuautle, M. A. Duarte-Villaseñor, C. A. Reyes-García, et al., "Designing VFs by applying genetic algorithms from nullator-based descriptions," in *Proceedings of the European Conference on Circuit Theory and Design (ECCTD '07)*, pp. 555–558, Seville, Spain, August 2008.

[10] E. Teleo-Cuautle and L. A. Sarmiento-Reyes, "Biasing analog circuits using the nullor concept," in *Proceedings of the Southwest Symposium on Mixed-Signal Design (SSMSD '00)*, San Diego, California, USA, February 2000.

[11] E. Tlelo-Cuautle, "An efficient biasing technique suitable for any kind of the four basic amplifiers designed at nullor level," in *Proceedings of IEEE International Symposium on Circuits and Systems*, vol. 3, pp. 535–538, Phoenix, Ariz, USA, May 2002.

[12] D. G. Haigh, T. J. W. Clarke, and P. M. Radmore, "Symbolic framework for linear active circuits based on port equivalence using limit variables," *IEEE Transactions on Circuits and Systems I*, vol. 53, no. 9, pp. 2011–2024, 2006.

[13] D. G. Haigh and P. M. Radmore, "Admittance matrix models for the nullor using limit variables and their application to circuit design," *IEEE Transactions on Circuits and Systems I*, vol. 53, no. 10, pp. 2214–2223, 2006.

[14] D. G. Haigh, "A method of transformation from symbolic transfer function to active-RC circuit by admittance matrix expansion," *IEEE Transactions on Circuits and Systems I*, vol. 53, no. 12, pp. 2715–2728, 2006.

[15] C. Beccari, *Transmission Zeros*, Departimento di Electronica, Turin Instisute of Technology, Turino, Italy, 2001.

[16] R. Hashemian, "Hybrid equivalent circuit, an alternative to thevenin and norton equivalents, its properties and applications," in *Proceedings of IEEE International Midwest Symposium on Circuits and Systems*, Cancun, Mexico, August 2009.

[17] R. Hashemian, "Local biasing and the use of nullator-norator pairs in analog circuits designs," in *Proceedings of the Midwest Symposium on Circuits and Systems*, pp. 466–469, Knoxville, Tenn, USA, August 2008.

Hindawi Publishing Corporation
VLSI Design
Volume 2010, Article ID 460312, 9 pages
doi:10.1155/2010/460312

Research Article

Error Immune Logic for Low-Power Probabilistic Computing

Bo Marr, Jason George, Brian Degnan, David V. Anderson, and Paul Hasler

School of Electrical and Computer Engineering, Georgia Institute of Technology, Atlanta, GA 30332, USA

Correspondence should be addressed to Bo Marr, marr.bo@gmail.com

Received 27 May 2009; Accepted 19 November 2009

Academic Editor: Gregory D. Peterson

Two novel theorems are developed which prove that certain logic functions are more robust to errors than others. These theorems are used to construct datapath circuits that give an increased immunity to error over other naive implementations. A link between probabilistic operation and ultra-low energy computing has been shown in prior work. These novel theorems and designs will be used to further improve probabilistic design of ultra-low power datapaths. This culminates in an asynchronous design for the maximum amount of energy savings per a given error rate. Spice simulation results using a commercially available and well-tested 0.25 μm technology are given verifying the ultra-low power, probabilistic full-adder designs. Further, close to 6X energy savings is achieved for a probabilistic full-adder over the deterministic case.

1. Introduction

As digital technology marches on, ultra-low voltage operation, atomic device sizes, device mismatch, and thermal noise are becoming commonplace and so are the significant error rates that accompany them. These phenomena are causing ever increasing bit-error rates, and with billion-transistor digital chips being produced today, even a 1-in-100-million bit error rate becomes costly.

This paper will present a novel discovery of boolean logic that certain logic gates are more robust to error than others, and in fact it will be shown that some logic even *improves* the error rate just through natural computation. The paper will show how these principles translate into CMOS and other implementations, but these principles are *independent of technological implementation* since they are properties of boolean logic itself. Thus these design principles will stand the test of time.

The motivation behind studying computing architectures robust to error then becomes clear, especially as technological breakthroughs have recently shown that extreme power savings can be traded for a certain level of error—known as probabilistic computing [1]. Paying more attention to error rates is critical if scaling power consumption and devices is to continue [2].

Recently, ultra-low power computing has been achieved by lowering the supply voltage of digital circuits into near threshold or even the subthreshold region [1, 3]. Indeed a fundamental limit to voltage scaling technology has been proposed: the thermodynamic limit of these devices [4]. When the supply voltage becomes comparable to thermal noise levels in these types of ultra-low power designs, devices start to behave probabilistically giving an incorrect output with some nonzero probability [4, 5]. Kish predicts that the thermal noise phenomenon will result in the "death" of Moore's law [6]. The International Technology Roadmap for Semiconductors predicts that thermal noise will cause devices to fail catastrophically during normal operation—without supply voltage scaling—in the next 5 years [6, 7].

A paradigm-shifting technology has been introduced in part by the authors called probabilistic CMOS or pcmos to combat the failure in voltage scaling due to the kT/q thermal noise limit. Experiments have already been completed that show that this thermal noise barrier can be overcome by computing with deeply scaled *probabilistic* CMOS devices. It was shown, in part by the authors, that probabilistic operation of devices allows for applications in arithmetic and digital signal processing that use a fraction of the power of their deterministic counterparts [1, 8]. This paper builds upon this work, and offers improved solutions for ultra-low power datapath units.

Logic gates are the fundamental building blocks of all digital technology, and it has been discovered that not all logic gates propagate error equally *regardless of technological implementation*. The main contributions of this paper are the following.

(i) A novel property of boolean logic is presented showing that different types of boolean logic propagate errors much differently *regardless of implementation or technology generation*

(ii) Several theorems will be given that offer a guideline for logic that most reduces error rates.

(iii) A case study using full-adders is given using these principles to achieve reduced error rates strictly through intelligent implementation (e.g., without error correction logic).

(iv) An analysis is given for asynchronous logic showing that a lower error rate is achieved when compared to its synchronous counterparts.

(v) It is shown that reducing error rates can be translated into reduced power consumption via probabilistic computing.

This work is an expansion of the work proposed in [9]. The background to this research and related work is given in Section 2. Some definitions and assumptions are given in Section 3. A case study analyzing which implementation is optimal for noisy full-adders is given in Section 5. Theorems illustrating the reliability of logic circuits under a probabilistic fault model are presented in Section 4. Conclusions and future directions are discussed in Section 6.

2. Logic Tables and Device Physics

2.1. All Logic Is Not Created Equal. Boolean logic functions are most simply represented by a *Truth Table* mapping each input combination to an output. When a bit-error is present at the input, or one of the input bits is flipped in other words, if this new input combination is mapped to the same output, then no error results in calculating the given logic function. In this case the logic function did not *propagate* the error. If one assumes a small, given error rate per bit, then a single-bit error is more likely than two simultaneous bit errors which is more likely than three simultaneous errors, and so forth. A logic function that has the least input-output mappings where a single-bit error on an input causes a mapping to a different output will be the least likely to propagate a bit error. This phenomenon is shown in Figures 1(a) and 1(b) using an NAND function and an XOR function as an example.

Figure 1 illustrates the theme that not all logic gates propagate errors equally. If one calculates the average probability of error across all possible input combinations of NAND and XOR logic, an extremely interesting result emerges. Assume that a probability of error of $\epsilon = 0.2$ is present at the inputs of these gates. Further assume that full-adders are built such that a probability of error at the input $\epsilon = 0.2$ is also present and that one of the full-adders is built with an NAND-NAND implementation and the other is built with a standard XOR implementation. Figure 2 shows the drastically differing output probabilities δ that result. Note that in Figure 2, the gate itself has no probability of computing erroneously associated with it. This shows that different gates have differing abilities to mask errors regardless of input combination.

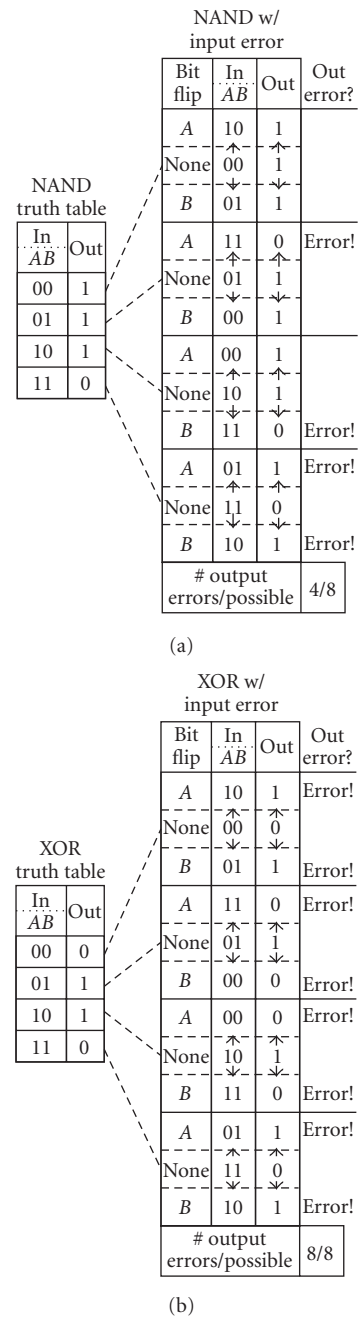


Figure 1: The effect of a single-bit error or bit flip on the inputs of an NAND function and XOR function is shown. Note that if the new input combination resulting from a bit flip maps to the same output as the input combo with no bit flip, no error is seen at the output. A NAND function is much less likely to propagate an input bit flip to the output resulting in an error than an XOR function. (a) NAND function (b) XOR function.

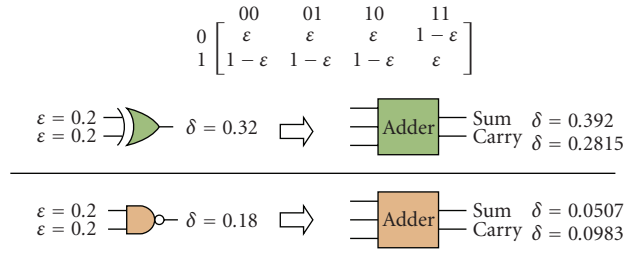


Figure 2: The probability of output error δ, of a NAND versus an XOR when a probability error of $\epsilon = 0.2$ is present at the input. Further, δ is compared for a full-adder implemented with NAND-NAND and with a standard XOR when $\epsilon = 0.2$ is present at the input. Note that not only is the probability of output error δ much lower for a NAND gate than an XOR gate, but that the error propagated from the NAND is *improved* over the input error. In a sense, a NAND "heals" the circuit. Note that the gate itself does not compute erroneously in this example.

As shown in Figure 2, the NAND gate has a much lower output probability of error, δ, than the XOR gate when under the same input error conditions. An interesting property also emerges that a NAND actually propagates an *improved* error probability due to its logic properties through its natural computation. Obviously, a $\epsilon = 0.2$ is orders of magnitude higher than the actual error rates seen in digital logic, but it will later be shown that the output error probability of logic gates monotonically decreases as a function of decreasing error rates and thus this concept holds true at more realistic bit-error rates. Subsequent sections will go into more depth explaining these properties.

2.2. MOSFET Implementation and Device Properties. There are two key issues that cause device failures in digital logic implemented with MOSFET transistors: device mismatch and thermal noise. This work will address the thermal noise problem; device mismatch will be addressed in future works. As for other types of noise, the switching speed of devices is faster than the frequencies typically seen for the $1/f$ type of noise (flicker noise), so there is no need to address that effect. A plot of thermal noise measured from a 0.35 μm chip is shown in Figure 3(a). The derivation of the probability of a digital "1" or "0" is seen in Figure 3(b) using a comparator.

Device mismatch continues to get worse with minimum size devices as process sizes scale down. Mismatch problems can be mitigated with larger device sizes, but this is not always reasonable, increases power dissipation, and might give loss of computing performance. Other techniques are possible for tuning digital gates and is the subject of much recent work [10, 11] that has shown promise for programmatically tuning devices with floating gates. With sufficient calibration using these tuning techniques, the device mismatch effect can be removed. This result is quite significant, as interest in subthreshold digital circuits continues to increase. Mismatch in threshold voltage impacts circuits by $e^{\kappa \Delta V_t / U_t}$, where $U_t = kT/q$ or 25.4 mV. Hence a mismatch of 20 mV, which happens for minimum devices in 0.35 μm CMOS process, could result in a change of current by

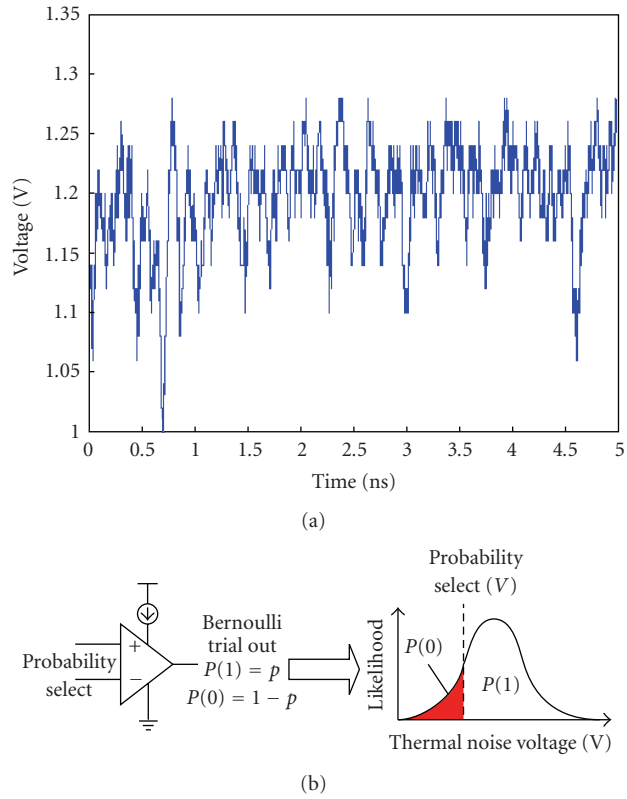


Figure 3: (a) Amplified thermal noise measured from a 0.35 μm chip. (b) Plot showing the derivation of probability of a digital "1" or "0" when thermal noise and a probability select signal are both put through a comparator. Data was collected to verify probability of error when CMOS logic is subject to noise.

a factor of 2 [12]. As we scale down, this level of V_t mismatch continues to increase. The failure in this case is that timing of the gates somehow does not occur when expected on a probabilistic basis.

Even a conservative estimate of the thermal noise problem is disconcerting. The thermal noise magnitude for a digital gate is approximated as kT/C noise. For a small 1 1F capacitor, the rms noise level is roughly 2 mV. However, because of a trend of larger interconnect capacitances, large digital buffers, and higher operating temperatures due to transistor density, thermal noise has been shown to increase to 200 mV in even nanoscale digital circuits [5]. In low-power digital logic, a supply voltage in the hundreds of mV is not unreasonable especially in subthreshold logic and hence the probability of an error becomes exceedingly likely in this case [3]. Of course, one would need to look at the probability over a large number of gates, say 1 billion gates, and never having an error all the sudden gets interesting.

An estimate given by Kish in [6] predicts untenable device failure due to thermal noise at the 22 nm transistor node assuming $V_t = 0.2$ V, creating an argument for a not so distant problem. So far, device mismatch issues are the larger effect. However, thermal noise is the eventual limit that will be reached in low-power circuit design, and thus merits the treatment given here.

2.3. Theoretical Background for Probabilistic Gates. Design with probabilistic gates is an old problem that has been given a new application via ultra-low power computing. In [13], von Neumann showed two important results. Reliable circuits can be built with noisy (probabilistic) gates with an arbitrarily high reliability, but this circuit will compute more slowly than the same circuit built with noiseless gates due to necessary error correction logic. von Neumann refers to triple modular redundancy (TMR) and others refer to N-modular redundancy (NMR) to achieve these results shown in [13]; however, these techniques often result in high circuit overhead of as much as 200% or more. Pippenger showed exactly how much error correction would be needed [14] to achieve arbitrary reliability.

Pippenger improved upon this result by showing that the fraction of layers of a computing element that must be devoted to error correction for reliable computation is $2\epsilon/\ln 3$, but he was unable to show how such a computing element could be built [14].

Others have addressed soft error rate (SER) reduction and error masking techniques through time redundancy which catches transient errors due to particle strikes and delay-based errors [15, 16]. This work differs from these others because here we consider bits that are truly probabilistic due to thermal noise properties and the error rate is independent of time. Therefore, the shadow latching techniques and timing redundancy techniques presented in the aforementioned work are not as effective. Not to mention that these techniques pose the same drawback as TMR for time-independent errors in that there is significant overhead circuit involved and often involve pipeline flushes if an error is detected. Alternatively, in this work we show that logic synthesis can be done such that the fabric of the logic itself reduces error propagation.

The work presented in [17] presents a probabilistic analysis framework similar to the one presented in this paper and addresses adders and logic gates also similarly to this paper. However, they neither come to the conclusion of the relative error propagation characteristics of different logic networks nor are they able to link probability to power the way this paper does. It also presents a framework for analysis and not necessarily a solution to probabilistic errors.

Krishnaswamy et al. in [18] address probabilistic gates by using logic transformations to reduce error rates by considering that observability do not care (ODC) sets. This work differs from the current paper in that probabilistic thermal noise faults are considered herein, not deterministic stuck-at-faults. This work is superior to previous methods in the metric of energy savings in that no error correction logic or overhead is used.

Synthesizing probabilistic logic into CMOS using Markov Random Fields was presented in [8]. However, the CMOS circuits presented are not standard and use far more transistors (20 transistors for an inverter) than the gates proposed here.

Finally, Chakrapani et al. in [19] have done work in synthesizing probabilistic logic for inherently probabilistic applications, but do not address probabilistic design for deterministic applications.

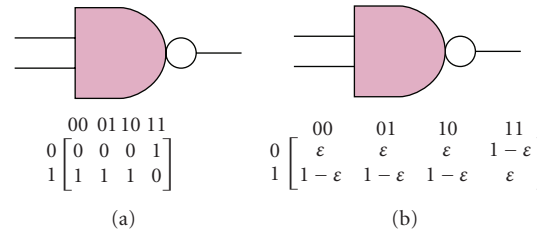


Figure 4: (a) NAND2 ITM (b) NAND2 PTM.

3. Defining Probabilistic Gates via Matrices

A tight analysis of the differing error rates present in boolean logic is given. To give a framework for the discussion two definitions are presented.

Definition 1. ϵ is the probability that a bit flip occurred on a circuit node.

Definition 2. δ is the probability that the output of a network of gates is incorrect as a function of ϵ.

The assumptions used in this work are similar to previous works on the subject [13, 14]: $0 < \delta < 0.5$, $0 < \epsilon < 0.5$, and each node is assumed to fail (flip) independently and with a uniform probability, ϵ. All input combinations are assumed to be equally likely. Stein in [5] and Cheemalavagu et al. in [4] show that thermal noise can be modeled as errors occurring with a precise probability defined by the noise to V_{dd} ratio and that this noise can be modeled at the input or output of each gate, thus ϵ is defined at the *nodes* of the circuit. The probability model used in this paper is the same as in [4], which has been verified by running an HSPICE circuit simulation and measurements from a 0.25 μm chip. "Min", "Maj", and "AOI" will be used throughout this paper which stand for the Minority gate, Majority gate, and And-Or-Inv gates, respectively, which are shown in [20]. To review a minority gate simply outputs the value equal to the value present on the minority of inputs. Thus for a 3-input minority function with the input "001" or "000" the output would be "1".

Transfer matrix methodology is used for calculating probabilities at the outputs of a complex network of gates. This methodology allows the probability to be calculated in a strict, mathematically defined way regardless of input [21]. The ideal transfer matrix (ITM) and PTM for a NAND2 gate can be seen in Figure 4 where each entry in the matrix is the probability of that input-output combination occurring. The top row of numbers in Figure 4(a): "00", "01", "10", "11" are the possible input combinations and the columns on the left side are the possible output values. An ITM simply is a matrix with a "1" as an entry that corresponds to the correct output for a given input combination. In other words, $\epsilon = 0$ for an ITM.

The PTM methodology is used in the definition of δ, illustrated in Figure 5. To calculate δ, all inputs are considered to be equally likely. Once the final probability transfer

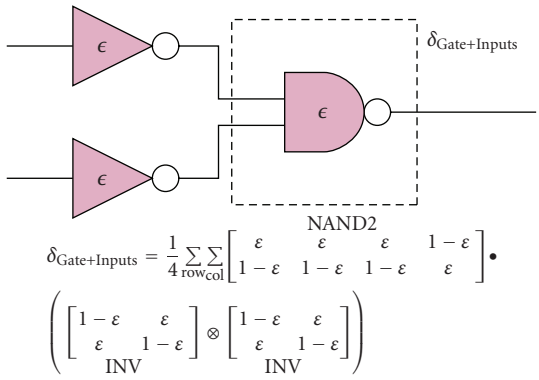


Figure 5: This circuit illustrates the definitions for ϵ and δ. Symbol ϵ is the error probability for a node and δ is the probability that a network of gates is erroneous. For a single gate, δ is the error probability of that gate inclusive of the fact that the inputs may be erroneous. Each gate is represented by a matrix and the matrices from each level of the circuit are multiplied to get the final PTM.

matrix is calculated for the circuit, giving a probability of each output occuring for each input combination, the probability of error is then calculated by summing the probability of error for each input combination and then dividing by the number of input combinations. In Figure 5, there are 2 inputs to the circuit yielding 4 possible input combinations, thus the final sum is divided by 4 to get the average probability of error.

The algorithm described in [22] shows how to calculate an overall circuit PTM from individual gate PTMs. Briefly, one calculates the final PTM from the output. Each level of the circuit is represented by a matrix. Each level is then matrix multiplied with the level before it. If there are more than one gate at a given level, the PTM is calculated using the kronecker product of all gates.

4. Theorems for Noisy Gates

Two theorems proving properties of noisy gates are outlined here, which can be used as a guideline for designing circuits with minimal probability of error. The first theorem proves which gate types have the lowest probability of propagating an error, and the second one proves that the probability of an error at the output of a circuit increases as the depth of that circuit increases.

Theorem 1. *A noisy gate of 2 or 3 inputs will have a minimal probability of error, δ, when the cardinality of its ON-Set or OFF-Set is 1.*

Note that the cardinality of the ON-Set of an AND gate is 1 and the cardinality of the OFF-Set of a NAND gate is 1. In other words, there is 1 input-output mapping that gives a value of ON (1) and OFF (0), respectively.

Proof. The probability of an *correct* value at the output of the gate, $1 - \delta_{\text{gate}}$, is calculated in (1).

$$1 - \delta_{\text{GI}} = \frac{1}{C}\sum_{r=1}^{R}\sum_{c=1}^{C}\text{ITM}_{\text{GI}}(r,c) * \text{PTM}_{\text{GI}}(r,c), \tag{1}$$

where GI = Gates + Inputs, C = number of columns in transfer matrix, R = number of rows in transfer matrix.

The PTM for a gate including the inputs is the dot product of the PTM of the gate itself with the kronecker product of the PTM for inputs.

$$\text{PTM}_{\text{GI}} = \text{PTM}_{\text{G}} \cdot (A_1 \otimes A_2 \otimes \ldots \otimes A_n), \tag{2}$$

where $A_1 \ldots A_n$ are PTMs for inputs A_1 through A_n, G = Gates.

The δ error function is a monotonic function with respect to ϵ and thus so is $1 - \delta$.

The equation for the minimal δ as calculated from (1) is shown for each possible value of $y = \min(|F^{\text{ON}}|, |F^{\text{OFF}}|)$ in (3). For a 3-input function, $y \in \{1, 2, 3, 4\}$.

$$\begin{aligned}
1 - \delta_{y=1} &= \frac{1}{8}\Big[8(1-\epsilon)^4 + 18\epsilon(1-\epsilon)^3 \\
&\qquad \cdots + 24\epsilon^2(1-\epsilon)^2 + 12\epsilon^3(1-\epsilon) + 2\epsilon^4\Big], \\
1 - \delta_{y=2} &= \frac{1}{8}\Big[8(1-\epsilon)^4 + 16\epsilon(1-\epsilon)^3 \\
&\qquad \cdots + 20\epsilon^2(1-\epsilon)^2 + 16\epsilon^3(1-\epsilon) + 4\epsilon^4\Big], \\
1 - \delta_{y=3} &= \frac{1}{8}\Big[8(1-\epsilon)^4 + 14\epsilon(1-\epsilon)^3 \\
&\qquad \cdots + 20\epsilon^2(1-\epsilon)^2 + 16\epsilon^3(1-\epsilon) + 6\epsilon^4\Big], \\
1 - \delta_{y=4} &= \frac{1}{8}\Big[8(1-\epsilon)^4 + 16\epsilon(1-\epsilon)^3 \\
&\qquad \cdots + 16\epsilon^2(1-\epsilon)^2 + 16\epsilon^3(1-\epsilon) + 8\epsilon^4\Big],
\end{aligned} \tag{3}$$

where

$$\begin{aligned}
&\delta_{y=k}\ \delta \text{ for 3-input function} \\
&y\ \min\Big(\big|F^{\text{ON}}\big|, \big|F^{\text{OFF}}\big|\Big), \\
&\epsilon \text{ probability of error at gate node.}
\end{aligned} \tag{4}$$

The equation for δ for a each value of y for 2 input functions is given in (5).

$$\begin{aligned}
&1 - \left(\delta_{y=1}\right) \\
&\quad = \frac{1}{4}\Big[4(1-\epsilon)^3 + 4\epsilon(1-\epsilon)^2 + 6\epsilon^2(1-\epsilon) + 2\epsilon^3\Big], \\
&1 - \left(\delta_{y=2}\right) \\
&\quad = \frac{1}{4}\Big[4(1-\epsilon)^3 + 4\epsilon(1-\epsilon)^2 + 4\epsilon^2(1-\epsilon) + 4\epsilon^3\Big].
\end{aligned} \tag{5}$$

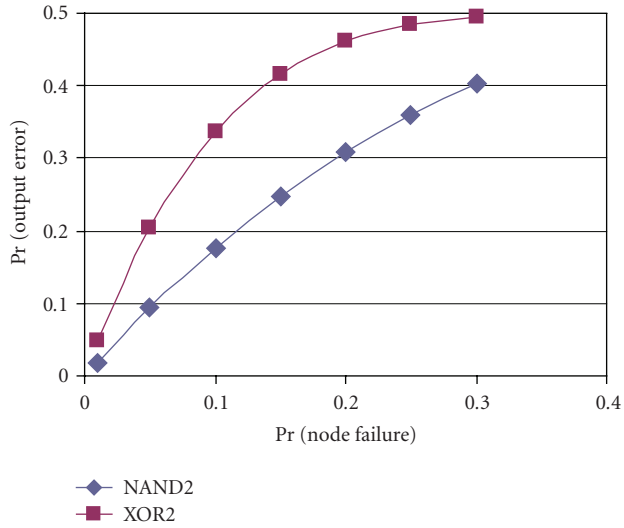


Figure 6: δ versus ϵ for a NAND2 gate and an XOR2 gate.

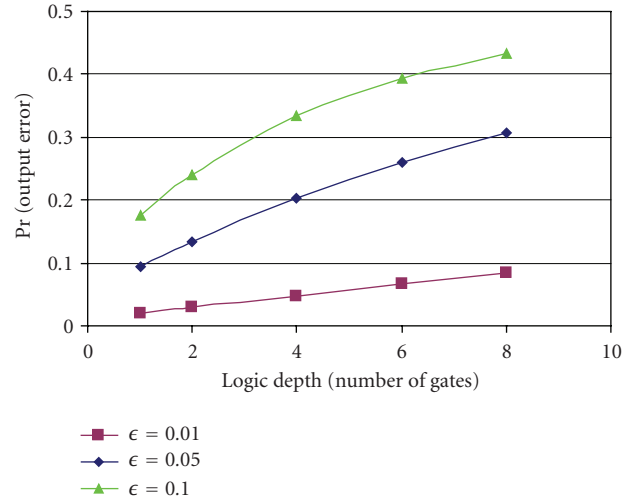


Figure 7: δ versus Logic Depth for an inverter chain for $\epsilon = \{0.01, 0.05, 0.1\}$.

For $0 < \epsilon < 0.5$, the probability of error, δ, for 2 and 3 input gates is minimal for $y = 1$. □

Corollary 1. *NAND and NOR have the minimal probability of error, δ, of any gate type subjected to noise for 2 and 3 inputs. Among the set of 2 and 3 input gates subjected to noise for the set: {NAND, NOR, XOR, AOI, OAI, Maj,* Min*}, NAND and NOR have a lower probability of error than any other gate in this set.*

Proof. NAND and NOR are gates where $y = \min(|F^{\text{ON}}|, |F^{\text{OFF}}|) = 1$, and the other gates in the set are such that $y > 1$. Therefore, from Theorem 1, NAND and NOR have the minimal probability of error, δ, which is less than δ of the XOR, AOI, OAI, Maj, and Min gates. □

It is easy to conclude from Figure 6 that the XOR2 gate has a higher error, δ, than the NAND2 gate for all values $0 < \epsilon < 0.5$. Thus it could be said that XOR2 gates propagate errors in a circuit with a higher probability than a NAND2 gate. As per Theorem 1, NAND2 will in fact have the lowest error rate of any gate for any $0 < \epsilon < 0.5$.

Another implication of Theorem 1 and its corollary is that the results can be extended to circuits of an arbitrary size. That is to say that any logic function will have the least probability of error when implemented with gates where the cardinality of those gates' *ON-Set* or *OFF-Set* is 1. Further, the smaller the minimum cardinality of either the *ON-Set* or *OFF-Set* of these gates, the smaller the error probability of the network of gates.

Secondly, the error rate of a NAND gate decreases as a function of number of inputs, for example going from a 2-input to a higher input version, whereas the opposite is true for an XOR gate.

Theorem 2. *The probability of error at an output of a noisy circuit δ increases as the logic depth of that circuit increases.*

Table 1: δ for 2 and 3-input Gates for $\epsilon = .005$.

Gate	δ	Gate	δ
NAND2	.0099	XOR2	.0149
NAND3	.0087	XOR3	.0197
AOI3	.0112	MIN3	.0124
NOR2	.0099	NOR3	.0087

Proof. Let A be the probability transfer matrix (PTM) for a given noisy circuit of depth d. Depth is defined as the longest path in number of gates from primary input to primary output. To increase the length of this circuit to $d+1$, additional noisy logic is added to compute the inputs of A. Let the kronecker product of the PTMs of the additional input logic to A be B. The resulting PTM of the new circuit of length $d+1$ is $A \cdot B$ according to [21].

Assume $\delta_{A \cdot B} \leq \delta_A$. Then according to (1), the noisy circuit B would have to have $\epsilon \leq 0$, a contradiction. Therefore, $\delta_{A \cdot B} > \delta_A$. □

The results for increasing logic depth of an inverter chain with different values of ϵ can be seen in Figure 7.

From Figure 7, one can see that no matter the value of ϵ, the error of the circuit, δ, increases as the logic depth increases. Another result that is available from the experiment is that the rate at which δ increases with an increase in logic depth is proportional to ϵ. This implies that the importance of logic depth becomes increasingly important with the level of noise present in the circuit. This further implies that as deep submicron transistor generations are explored, logic cones and pipeline stages must become increasingly smaller to sustain the same level of reliability.

Table 1 shows the different probabilities of error δ of gates given $\epsilon = 0.005$.

Several observations of regarding logic synthesis under a probabilistic fault model were observed.

Table 2: Single-Bit Flips (SBF) at the input that could cause a flip in the output of the carryout calculation of a dual-rail asynchronous adder.

A	B	C	A_t	A_f	B_t	B_f	C_t	$\overline{\text{Cout}}_t$	SBF Error
∅	∅	∅	0	0	0	0	0	1	∅
0	0	0	0	1	0	1	0	1	∅
0	0	1	0	1	0	1	1	1	A_t,B_t
0	1	0	0	1	1	0	0	1	C_t
0	1	1	0	1	1	0	1	0	C_t
1	0	0	1	0	0	1	0	1	C_t,B_t
1	0	1	1	0	0	1	1	0	C_t,A_t
1	1	0	1	0	1	0	0	0	B_t,A_t
1	1	1	1	0	1	0	1	0	∅

Table 3: Full-Adder Implementations used in experiment.

FA	C_{out}	Sum	No. Transis.
async	—	—	42
f28	Min3-Inv	Min3-Min4-Inv	28
fmaj	Inv-Min3	XOR3	32
fbase	XOR2-And2-Or2	XOR2-XOR2	42

Table 4: simulation results for 4 different full-adder implementations in TSMC 0.25 μm at $\epsilon = .005$ ($V_{\text{dd}} = 1.2\,V$, $U_n = 0.3\,V$).

FA Type	Energy (pJ)	C_{out} delay (ns)	C_{out} δ	Energy savings versus deterministic
async	0.691	1.07	.0159	5.91X
f28	1.409	1.96	.0173	2.90X
fmaj	1.549	2.22	.0196	2.64X
fbase	4.087	1.19	—	—

(i) $\delta_{\text{Minority-Inv}} = \delta_{\text{Majority}}$: a minority gate followed by an inverter will have the same δ value as a majority gate.

(ii) For $x_1 \oplus x_2 \oplus x_3$: $\delta_{\text{XOR2-XOR2}} > \delta_{\text{NAND2-NAND2}} > \delta_{\text{XOR3}}$: XOR3 has the minimal error rate, δ.

(iii) $\delta_{\text{NAND2-NAND2}} > \delta_{\text{NAND4}}$.

(iv) For $x_1 \cdot x_2 + x_3 \cdot x_4$: δ_{Majority} is minimal.

Table 1 confirms the results predicted by the theory given earlier in this section. Namely, NAND and NOR type gates have the lowest probability of error, δ, of any gate. Further, for a given number of inputs, as $y = \min(|F^{\text{ON}}|, |F^{\text{OFF}}|)$ increases, so does the probability of error *delta*. So for example, a gate that has $y = 1$ such as a NAND3 gate has a minimal error. But for $y = 3$ such as an AOI3 gate δ is increased, and δ further increases for $y = 4$ as in the XOR3 gate.

5. Case Study: The Full-Adder

The full-adder is a primary building block in digital arithmetic computation, present in nearly all digital adders and multipliers, and is one of the most well-studied circuits with a large variety of implementations. Its diverse array of circuit implementations and relative importance in creating low-power datapath units make it a prime candidate for study.

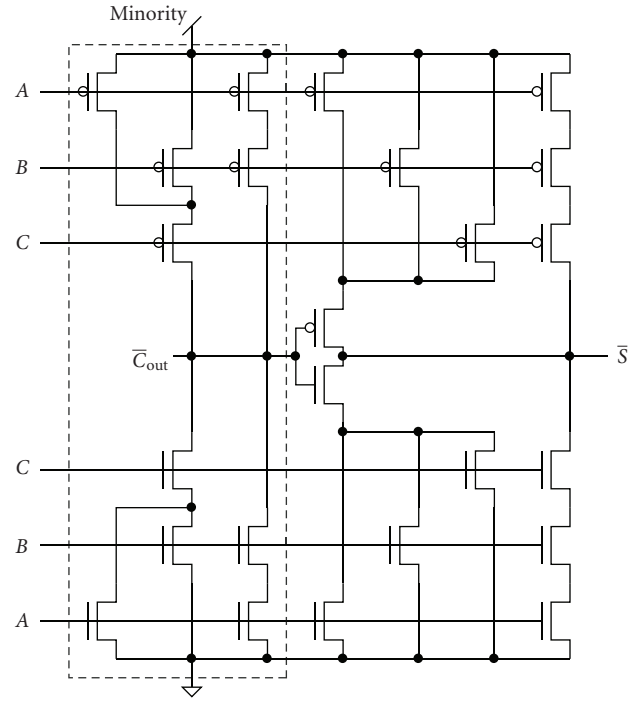


Figure 8: Circuit diagram of "f28" or "mirror" full-adder. Minority gate circuit used for the carry-out function of the 40-transistor adder as well is shown.

5.1. Full-Adder Microarchitecture. In [20, 23], a CMOS full-adder is presented that is optimized for transistor count, which is known as the 28-transistor implementation or "mirror adder"; it will be referred to as "f28". In [23], the authors claim the f28 is not only faster but consumes less power than the low-power carry-pass logic (CPL) implementation. The transistor level implementation of the "f28" full-adder is shown in Figure 8

It should also be noted that a majority gate is nothing more than a minority gate with an inverted output. A majority gate can also be achieved by inverting the inputs to the minority gate, which is useful should the inverted

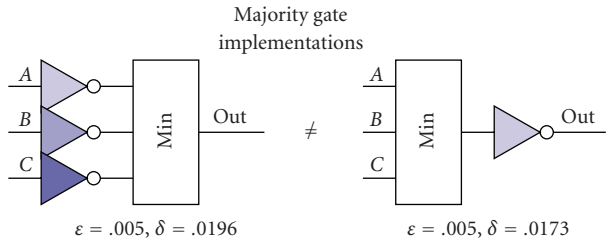


Figure 9: A minority gate with an inverted output is logically equivalent to a majority gate, which is also logically equivalent to a minority gate with inverted input. However, the probability of a bit error at the output is not equivalent based on the properties presented here.

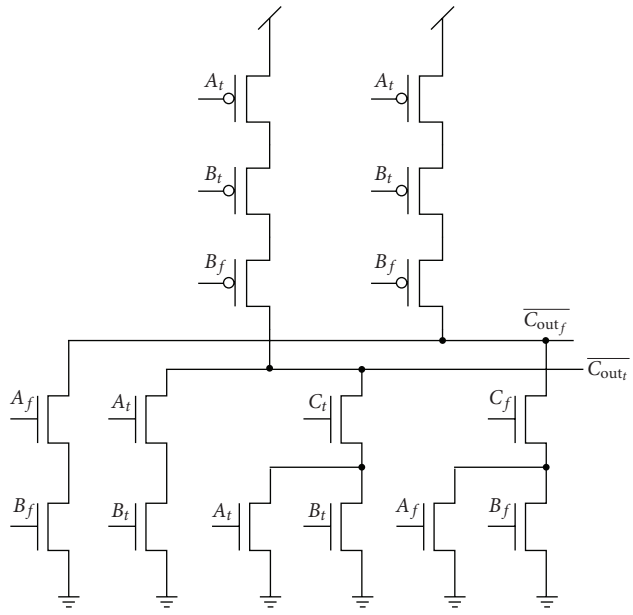


Figure 10: The transistor level schematic of the carryout bit calculation using dual-rail asynchronous logic. Each bit is represented by a true and false line, A_t and A_f, for example, where $A = 1 \rightarrow A_t = 1, A_f = 0$. If $A = 0 \rightarrow A_t = 0, A_f = 1$. The carryout bits are left as inverted because in dual-rail asynchronous adders, every other adder can calculate using inverted carry-in bits.

inputs be needed for some other part of the circuit. This phenomenon is illustrated in Figure 9.

Finally a dual-rail asynchronous adder is also presented for comparison. Asynchronous logic can be visited more in depth in [24]. The circuit for carry-out computation is shown in Figure 10.

In terms of the theorems proven earlier, dual-rail asynchronous logic is quite attractive because of a unique property. This type of logic switches between a *valid* state and an *invalid* state, and when the logic is in an invalid state, between one and many bit-flips on the input can be sustained before an output error occurs. These states are defined such that if bit A is *invalid* then $A_t = 0$, $A_f = 0$. Otherwise, if $A_t = 1$ or $A_f = 1$, then A is in *valid* state. Table 2 summarizes the result of single-bit flips (SBF)on the inputs of dual-rail asynchronous logic for each relevant input/output mapping.

Table 2 shows the single-bit flips at the input that would cause the output to erroneously flip for the carryout calculation of the dual-rail asynchronous adder. For example, no single-bit flip would cause an output error when the async-adder is in *invalid* state because at the very least the output will float. The async-adder is designed to leak back to the previous input in the case of a floating output. As another example if the async-adder is in state $ABC = 001$, a bit flip on A_t or B_t would cause $\overline{\text{Cout}_t}$ to flip. This is because the output is supposed to be pulled up to 1 but with the input $C_t = 1$ a bit flip on either of A_t or B_t will create a DC path to ground, pulling the output down to 0. As it turns out, the async-adder has the least single-bit flip errors on the input that can cause an output bit flip of any adder.

Several other adders were built including a "fmaj" full-adder built with an XOR3 gate for the sum bit and an Maj3 for the carryout. A baseline adder, measured at full voltage so that no probabilistic errors would occur called "fbase", was built for comparison. This adder, "fbase" was sized and built with several stages according to logical effort to maximize speed.

Simulations were run using the PTM algorithm in [22] to assess which full-adder types were most robust to error. These 4 full-adders were built in *Cadence* with TSMC 0.25 μm technology, and voltage scaled (except for the deterministic "fbase" adder) such that probabilistic errors occurred at each node in the circuit due to thermal noise, and then measured with the HSPICE circuit simulator. The results can be seen in Table 4. This particular V_{dd} was chosen because it was shown in [1] that adders and multipliers can be built to compute successfully with probabilities of error up to $\delta = .03$ and still successfully be used for image processing.

By scaling down supply voltages, the circuits will of course run more slowly; however, the energy-performance product metric showed a gain of up to 2.5X in these experiments which is not shown in Table 4 for simplification.

6. Conclusion

It was shown that probabilistic design principles can be used for ultra-low power computing. Probability and energy become closely linked in ultra-low power regimes because supply voltage is lowered so much so that it becomes comparable to the thermal noise level rendering the circuit probabilistic.

As shown in Table 4, error-aware logic design can produce a great reduction in the error rate and thus energy consumption over the baseline case which is optimized for speed. It was shown that in fact an asynchronous adder had the best performance producing both the lowest probability of error δ for a given ϵ and least energy consumption as well.

The principles behind the efficiency improvement were proven to be two important theorems developed under the probabilistic transfer matrix model. All gates do not propagate errors equally and that the depth of a digital circuit has a direct effect on the efficiency in terms of

energy consumed for a given error rate. Future work includes extending these theorems to a general logic synthesis algorithm, and continued work on tuning threshold voltages through floating gate technology to mitigate other device effects such as device mismatch.

Acknowledgments

The authors would like to thank the National Science Foundation for supporting this research and would also like to thank the reviewers for their insightful comments.

References

[1] J. George, B. Marr, B. E. S. Akgul, and K. V. Palem, "Probabilistic arithmetic and energy efficient embedded signal processing," in *Proceedings of the International Conference on Compilers, Architecture and Synthesis for Embedded Systems (CASES '06)*, pp. 158–168, Seoul, Korea, 2006.

[2] T. Sakurai, "Perspectives on power-aware electronics," in *Proceedings of the IEEE International Solid-State Circuits Conference (ISSCC '03)*, pp. 19–29, San Francisco, Calif, USA, February 2003.

[3] R. Hegde and N. R. Shanbhag, "Energy-efficient signal processing via algorithmic noise-tolerance," in *Proceedings of the International Symposium on Low Power Electronics and Design*, pp. 30–35, San Diego, Calif, USA, August 1999.

[4] S. Cheemalavagu, P. Korkmaz, K. V. Palem, B. E. S. Akgul, and L. N. Chakrapani, "A probabilistic cmos switch and its realization by exploiting noise," in *Proceedings of the IFIP International Conference on Very Large Scale Integration of System-On-Chip (VLSI-SoC '05)*, Perth, Australia, October 2005.

[5] K.-U. Stein, "Noise-induced error rate as a limiting factor for energy per operation in digital ICs," *IEEE Journal of Solid-State Circuits*, vol. 12, no. 5, pp. 527–530, 1977.

[6] L. B. Kish, "End of Moore's law: thermal (noise) death of integration in micro and nano electronics," *Physics Letters A*, vol. 305, no. 3-4, pp. 144–149, 2002.

[7] ITRS 2005 edition, http://www.itrs.net/Links/2009ITRS/Home2009.htm.

[8] K. Nepal, R. I. Bahar, J. Mundy, W. R. Patterson, and A. Zaslavsky, "Designing logic circuits for probabilistic computation in the presence of noise," in *Proceedings of the Design Automation Conference (DAC '05)*, pp. 485–490, Anaheim, Calif, USA, June 2005.

[9] B. Marr, J. George, P. Hasler, and D. V. Anderson, "Increased energy efficiency and reliability of ultra-low power arithmetic," in *Proceedings of the Midwest Symposium on Circuits and Systems (MWSCAS '08)*, pp. 366–369, Knoxville, Tenn, USA, August 2008.

[10] A. Basu, C. M. Twigg, S. Brink, et al., "RASP 2.8: a new generation of floating-gate based field programmable analog array," in *Proceedings of the Custom Integrated Circuits Conference (CICC '08)*, pp. 213–216, San Jose, Calif, USA, September 2008.

[11] B. Marr, A. Basu, S. Brink, and P. Hasler, "A learning digital computer," in *Proceedings of the Design Automation Conference (DAC '09)*, pp. 617–618, San Francisco, Calif, USA, July 2009.

[12] C. Mead, *Analog VLSI and Neural Systems*, Addison-Wesley, Reading, Mass, USA, 1989.

[13] J. von Neumann, *Automata Studies*, Princeton University Press, Princeton, NJ, USA, 1956.

[14] N. Pippenger, "Reliable computation by formulas in the presence of noise," *IEEE Transactions on Information Theory*, vol. 34, no. 2, pp. 194–197, 1988.

[15] S. Krishnamohan and N. R. Mahapatra, "A highly-efficient technique for reducing soft errors in static CMOS circuits," in *Proceedings of the IEEE International Conference on Computer Design: VLSI in Computers and Processors (ICCD '04)*, pp. 126–131, San Jose, Calif, USA, October 2004.

[16] S. Das, S. Pant, D. Roberts, et al., "A self-tuning DVS processor using delay-error detection and correction," in *Proceedings of IEEE Symposium on VLSI Circuits*, pp. 258–261, kyoto, Japan, June 2005.

[17] J. P. Hayes, I. Polian, and B. Becker, "An analysis framework for transient-error tolerance," in *Proceedings of the IEEE VLSI Test Symposium (VTS '07)*, pp. 249–255, Berkeley, Calif, USA, May 2007.

[18] S. Krishnaswamy, S. M. Plaza, I. L. Markov, and J. P. Hayes, "Enhancing design robustness with reliability-aware resynthesis and logic simulation," in *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design (ICCAD '07)*, pp. 149–154, San Jose, Calif, USA, November 2007.

[19] L. N. Chakrapani, B. E. S. Akgul, S. Cheemalavagu, P. Korkmaz, K. V. Palem, and B. Seshasayee, "Ultra-efficient (embedded) SOC architectures based on probabilistic CMOS (PCMOS) technology," in *Proceedings of the Conference on Design, Automation and Test in Europe (DATE '06)*, vol. 1, Munich, Germany, March 2006.

[20] D. Harris and N. H. Weste, *CMOS VLSI Design: A Circuits and Systems Perspective*, Pearson Education, Upper Saddle River, NJ, USA, 2005.

[21] K. N. Patel, J. P. Hayes, and I. L. Markov, "Evaluating circuit reliability under probabilistic gate-level fault models," in *Proceedings of the 12th International Workshop on Logic and Synthesis (IWLS '03)*, pp. 59–64, Laguna Beach, Calif, USA, May 2003.

[22] S. Krishnaswamy, G. F. Viamontes, I. L. Markov, and J. P. Hayes, "Accurate reliability evaluation and enhancement via probabilistic transfer matrices," in *Proceedings of the Conference on Design, Automation and Test in Europe (DATE '05)*, vol. 1, pp. 282–287, Munich, Germany, 2005.

[23] R. Zimmermann and W. Fichtner, "Low-power logic styles: CMOS versus pass-transistor logic," *IEEE Journal of Solid-State Circuits*, vol. 32, no. 7, pp. 1079–1090, 1997.

[24] A. J. Martin, "Asynchronous datapaths and the design of an asynchronous adder," *Formal Methods in System Design*, vol. 1, no. 1, pp. 117–137, 1992.

Hindawi Publishing Corporation
VLSI Design
Volume 2010, Article ID 230783, 13 pages
doi:10.1155/2010/230783

Research Article

Dynamic CMOS Load Balancing and Path Oriented in Time Optimization Algorithms to Minimize Delay Uncertainties from Process Variations

Kumar Yelamarthi[1] and Chien-In Henry Chen[2]

[1] *School of Engineering and Technology, Central Michigan University, Mt Pleasant, MI 48859, USA*
[2] *Department of Electrical Engineering, Wright State University, Dayton, OH 45435, USA*

Correspondence should be addressed to Kumar Yelamarthi, kumar.yelamarthi@cmich.edu

Received 2 June 2009; Revised 19 October 2009; Accepted 3 December 2009

Academic Editor: Ethan Farquhar

The complexity of timing optimization of high-performance circuits has been increasing rapidly in proportion to the shrinking CMOS device size and rising magnitude of process variations. Addressing these significant challenges, this paper presents a timing optimization algorithm for CMOS dynamic logic and a Path Oriented IN Time (POINT) optimization flow for mixed-static-dynamic CMOS logic, where a design is partitioned into static and dynamic circuits. Implemented on a 64-b adder and International Symposium on Circuits and Systems (ISCAS) benchmark circuits, the POINT optimization algorithm has shown an average improvement in delay by 38% and delay uncertainty from process variations by 35% in comparison with a state-of-the-art commercial optimization tool.

1. Introduction

The performance improvement of microprocessors has been driven traditionally by dynamic logic and microarchitectural improvements [1] and can be further enhanced through circuit design and topology organization. Dynamic logic is an effective logic style in terms of timing and area when compared to its static counterpart due to (1) the absence of requirement for design implementation in complementary PMOS logic, and (2) the use of a clock signal in its implementation of combinational logic circuits. In general, CMOS dynamic logic uses fast NMOS transistors in its pull-down network. Its delay is dependent on the number and size (width) of transistors in the NMOS critical path. This paper presents an NMOS transistor sizing optimization for a faster operation.

Static logic is slower because it has twice the loading, higher thresholds, and actually uses slow PMOS transistors for computation. Dynamic logic has been predominantly used in microprocessors, and their usage has increased the timing performance significantly over static CMOS circuits [1, 2]. However, timing optimization of dynamic logic is challenging due to several issues such as charge sharing, noise-immunity, leakage, and environmental and semiconductor process variations. Also, with dynamic circuits consuming more power over static CMOS, an optimal balance of delay and power can be achieved at the architectural level through effective partitioning of design into a mixed-static-dynamic circuit style [3].

Process variations introduce design uncertainties at each step of process development, design, manufacturing, and test. The ratio of these process variations to the nominal values has been increasing with the shrinking device size towards 32 nm [4], causing an impending requirement to account for process variations during timing optimization. They need to be taken into account during the design phase to make sure that performance analysis provides an accurate estimation [5].

One of the challenges in timing optimization of CMOS logic is delay uncertainty (Δ) from process variations, $\Delta = T_{\max} - T_{\min}$, where $T_{\max}$ and $T_{\min}$ are the maximum and minimum delays of a timing path. In the 180 nm CMOS

technology, these process variations have caused about 30% variation in chip frequency, along with 20X variation in current leakage [6]. The magnitude of intradie channel length variations has been estimated to increase from 35% of total variations in 130 nm to 60% in 70 nm CMOS process and variation in wire width, height, and thickness is also expected to increase from 25% to 35% [7]. In CMOS 65 nm process, the parameters that affect timing the most are device length, threshold voltage, device width, mobility, and oxide thickness [8]. For process variation sensitive circuits such as SRAM arrays and dynamic logic circuits, these process variations may result in functional failure and yield loss [7].

Addressing the challenges of timing optimization and delay uncertainty from process variations, this paper presents a timing optimization algorithm for dynamic circuits, and a timing optimization flow for mixed-static-dynamic CMOS logic. The proposed algorithm and flow are validated through implementation on several benchmark circuits and a 64-b binary adder in 130 nm CMOS process. This paper is an extension of our previous work [9] and is organized as follows. Section 2 presents previous work on transistor sizing (width) optimization for timing and process variation minimization. Section 3 presents the proposed transistor sizing optimization for CMOS dynamic logic. Section 4 presents implementation and results of several ISCAS benchmark circuits. Based on this timing optimization algorithm for dynamic circuits, Section 5 presents a timing optimization flow for mixed-static-dynamic CMOS logic and its implementation and results of several ISCAS benchmark circuits. Finally, conclusion is presented in Section 6.

2. Previous Work

Methods of automating transistor sizing for timing optimization were proposed in [3, 10–16], but many of them focus on static CMOS circuits and technologies using multiple threshold voltages. TImed LOgic Synthesizer (TILOS) [10] presented an algorithm of iteratively sizing transistors by a certain factor in the critical path. The algorithm is not a deterministic approach, as it does not guarantee convergence in timing optimization. MINFLOTRANSIT [11] is another algorithm proposed for transistor sizing based on iterative relaxation method but requires iterative generation of directed acyclic graphs for every step of timing optimization. Computation of "Logical Effort" is the other method proposed for timing optimization [16]. However, it has two limitations. First, it requires estimation of input capacitance, of which circuits with complex branches or multiple paths have difficulty in accurate estimate. Second, it optimizes timing at the cost of increased area [17].

Methods to mitigate the effect of process variations in CMOS circuits were proposed in [6, 7, 18–23]. These methods deal with statistical variations and are not optimal for designs with large number of parameter variations [24]. A technique called Adaptive Body Biasing (ABB) was presented in [6, 22] to compensate for variation tolerance. The ABB technique is implemented in postsilicon where each die receives a unique bias voltage, reducing the variance of frequency variation. However, this method does not minimize intradie variations, as each block in the design requires a unique bias voltage. Another limitation is the increasing leakage power, caused by the reduction of threshold voltage. Programmable keepers were proposed to compensate for process variations in [23]. This method works for designs with large number of parallel stacks (similar to the NOR gates). However, it requires additional hardware to program the keeper transistors for other designs.

Research has shown that intradie variations primarily impact the mean delay, and interdie variations impact the variance of delay [18]. As timing optimization should consider both interdie and intradie variations, both mean and variance should be accounted for. In addition to optimizing path delay, other parameters affected by process variations that need to be considered and reduced are delay uncertainty ($\Delta = T_{\max} - T_{\min}$) and sensitivity ($\delta = \sigma/\mu$), where $T_{\max}$ and $T_{\max}$ are the maximum and minimum delays, μ is the mean delay, and σ is the standard deviation of delay distribution.

3. Transistor Sizing Optimization of Dynamic Circuits

The delay of dynamic circuit is highly dependent on the number and size (width) of transistors in the critical path. Increasing width of transistors in a path will increase the discharging current and reduce the output pull-down path delay. However, increasing width of transistors to reduce one path delay may increase the capacitive load of channel-connected transistors on other paths and substantially increase their delays. This complexity increases along with the number of paths present in the circuit. A 2-b Weighted Binary-to-Thermometric Converter (WBTC) that is used in high-performance binary adders [25] shown in Figure 1 is used as an example to explain the path delay optimization complexity while considering process variations.

Figure 1 highlights two timing paths: path-A (T_{28}-T_7-T_8-T_{12}-T_{18}-T_{32}) and path-B (T_{28}-T_0-T_4-T_{11}-T_{15}-T_{16}-T_{31}). A test was performed to optimize path-A by gradually increasing widths of T_7, T_8, T_{12}, and T_{18}. It was observed that the delay of path-A reduced by 4%, but delay of path-B increased by 9.3%. This is a result of transistors on path-B being channel-connected to transistors on path-A. For instance, T_4 and T_{11} are channel-connected to T_7 and T_8, and T_{15} and T_{16} are channel-connected to T_{12} and T_{18}. Increasing widths of T_7, T_8, T_{12}, and T_{18} in path-A causes the capacitive load of T_4, T_{11}, T_{15}, and T_{16} to increase and therefore increase delay of path-B. This circuit example illustrates that increasing widths of transistors on one critical path increases capacitive load and delay of the other critical paths.

Conventionally, a path delay is denoted by the mean (μ) of its delay distribution, which accounts only for intradie variations. As interdie variations are equally important, its standard deviation (σ) is as important and should be considered as well. Consider the delay distribution of two paths of WBTC shown in Figure 2. Path-B has a high mean delay, while path-A has a high standard deviation. Typically, path-B would be chosen as the critical path for timing optimization as it has the highest mean delay (μ). Optimizing the design by increasing width of transistors on path-B may

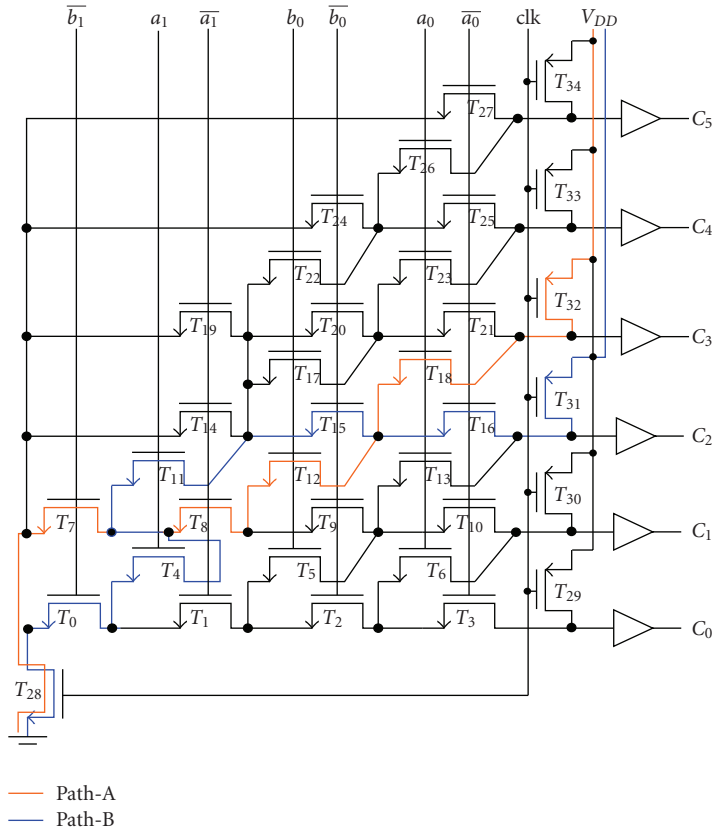


Figure 1: 2-b Weighted Binary to Thermometric Converter.

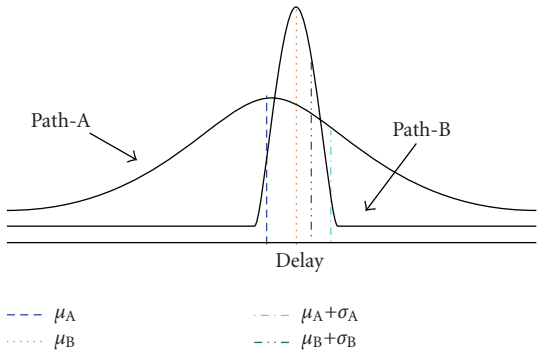


Figure 2: Delay distribution of two paths in 2-b WBTC.

reduce the mean delay (μ) but may not reduce its standard deviation (σ) as well. However, by considering both the mean and the standard deviation (i.e., $\mu + \sigma$), path-A would be chosen as the critical path to be optimized. As both interdie and intradie variations are equally important, the proposed timing optimization algorithm ranks critical paths based on the sum of the path mean delay and its standard deviation, ($\mu + \sigma$). The Load Balance of Multiple Paths (LBMPs) timing optimization algorithm proposed for transistor sizing of dynamic circuits while considering process variations is presented in Figure 3.

Consider the circuit with a series of n channel-connected NMOS transistors, $T_1, T_2, \ldots, T_n$, in Figure 4. While T_n conducts the discharge current of the load capacitance C_L, T_1 conducts the discharge current from a total capacitive load, $C_{\text{total}} = C_L + \cdots + C_3 + C_2 + C_1$, which is substantially large when n increases. So, the discharge time of T_1, the transistor near Gnd, is longer than T_n, the transistor near output. Accordingly, accounting for the discharge times increasing from T_n *to* T_1, the transistor sizes are made progressively larger, starting from a minimum-size transistor at T_n to reduce the total discharge time of the pull-down path (out, $T_n, \ldots, T_2, T_1$). The width of the next to the last transistor is scaled up by a factor. In the proposed LBMP algorithm we assign a *weight* (used for transistor sizing) in the range of 0.05–0.5 to each transistor relative to its distance from the output for this reason. For instance, the 2-b WBTC in Figure 1 is comprised of seven transistor stacks relative to their distance from the output. Stack-1, closest to the output, includes transistors T_3, T_{10}, T_{16}, T_{21}, T_{25}, and T_{27}. Stack-2 includes transistors T_6, T_{13}, T_{18}, T_{23}, and T_{26}. Stack-3 includes transistors T_2, T_9, T_{15}, T_{20}, and T_{24}. Stack-4 includes transistors T_5, T_{12}, T_{17}, and T_{22}. Stack-5 includes transistors T_1, T_8, T_{14}, and T_{19}. Stack-6 includes transistors T_4 and T_{11}. Stack-7 farthest from the output includes transistors T_0 and T_7. Accordingly, transistors in stacks 1–7 are assigned *weights* of 0.05, 0.1, 0.15, 0.2, 0.3, 0.4, and 0.5, respectively. For designs with different number of stacks, *weights* of transistors

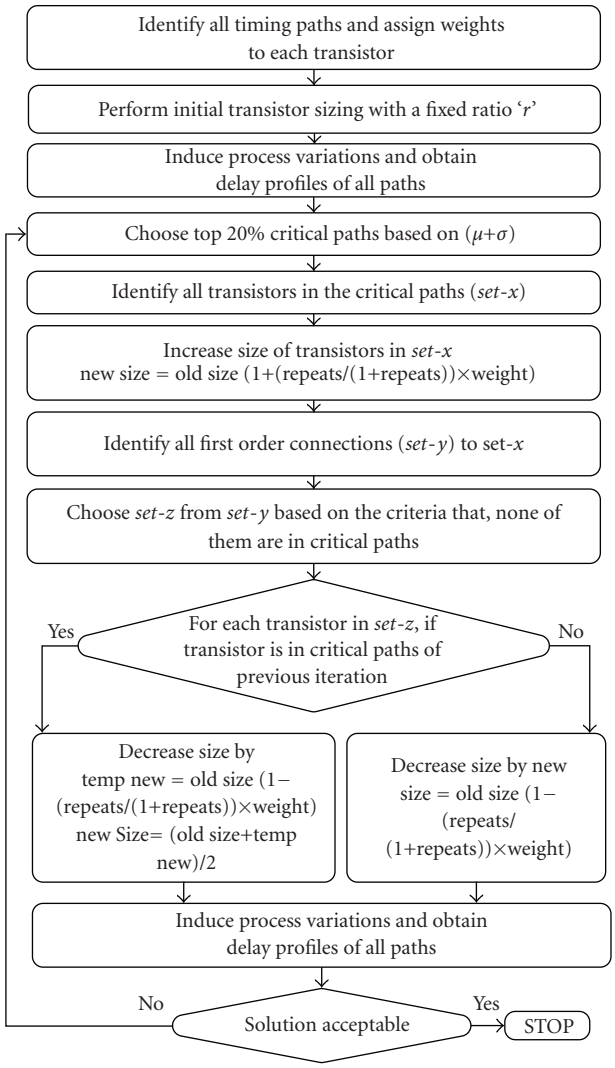


Figure 3: LBMP Transistor sizing algorithm.

in stacks are evenly distributed in the range 0.05–0.5 relative to its distance from the output; *weight* of 0.05 is assigned to stack of transistors closest to the output, and a *weight* of 0.5 is assigned to stack of transistors farthest from the output.

As increasing the width of transistor that appears in the most number of paths reduces overall delay, the number of paths a transistor is present in is computed and denoted by "*repeats*". The initial step in LBMP algorithm is to size transistors on every path with a fixed ratio, for example, 1.1 for faster optimization convergence [26]. After the *repeats* and the *weights* of all transistors are computed, simulations are performed to obtain delay distribution of each path. The transistors on the top 20% critical paths are grouped to *set-x*, and their widths are increased and calculated by (1):

$$\text{New\_Size} = \text{Old\_Size}\left(1 + \left(\frac{\text{Repeats}}{1 + \text{Repeats}}\right) \times \text{Weight}\right). \tag{1}$$

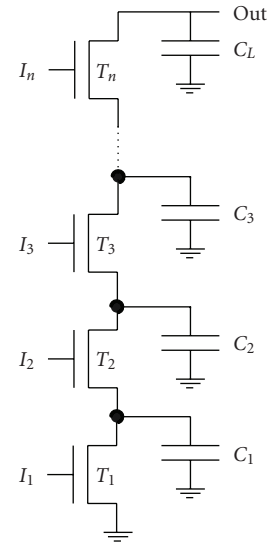


Figure 4: Transistor sizes are made progressively larger.

As the delay of critical path is dependent on the capacitive load of channel-connected transistors, reducing this capacitive load reduces the overall delay. The 1st-order connection transistors in *set-x* are identified and grouped to *set-y*. Then, transistors in *set-x* are excluded from *set-y* to form *set-z*. For each transistor in *set-z*, it is checked if the transistor is present in *set-x* of previous iteration. If so, its width is decreased and calculated by (2) and (3). If not, its width is decreased and calculated by (4). Once new transistor widths are determined, simulations are performed to locate the new critical paths. This algorithm is repeated until a convergence of an optimal solution is obtained:

$$\text{TempNew} = \text{Old\_Size}\left(1 - \left(\frac{\text{Repeats}}{1 + \text{Repeats}}\right) \times \text{Weight}\right), \tag{2}$$

$$\text{New\_Size} = \frac{(\text{Old\_Size} + \text{TempNew})}{2}, \tag{3}$$

$$\text{New\_Size} = \text{Old\_Size}\left(1 - \left(\frac{\text{Repeats}}{1 + \text{Repeats}}\right) \times \text{Weight}\right). \tag{4}$$

4. Optimization of Delay, Uncertainty, and Sensitivity from Process Variations

A 2-b Weighted Binary-to-Thermometric Converter (WBTC) used in high-performance binary adders was shown in Figure 1 [25]. This circuit is used as an example to illustrate the complexity of transistor sizing optimization. With less than 50 transistors, the 2-b WBTC has 34 timing paths, and of which path delays change dramatically with different transistor sizes.

The timing paths of the 2-b WBTC are shown in Table 1, and transistor *repeat* and *weight* profiles are shown in

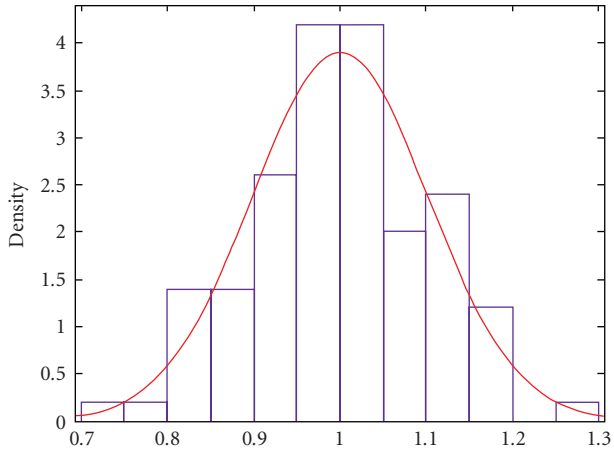


Figure 5: Delay Distribution of 2-b WBTC before Optimization.

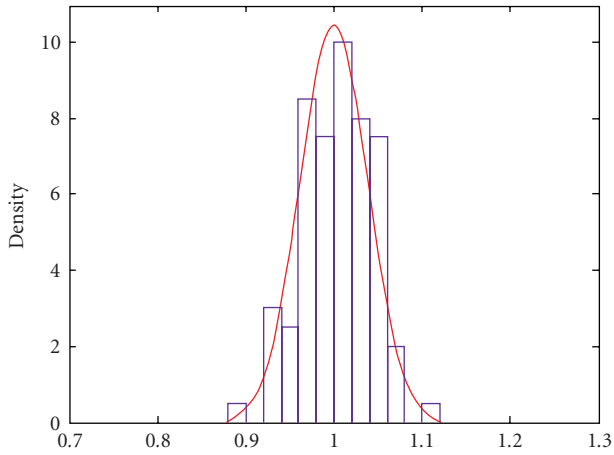


Figure 6: Delay Distribution of 2-b WBTC after Optimization.

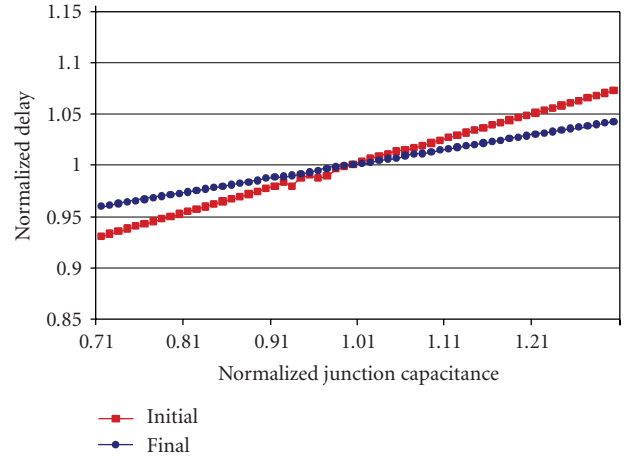


Figure 7: Delay Impact from Variation in Junction Capacitance.

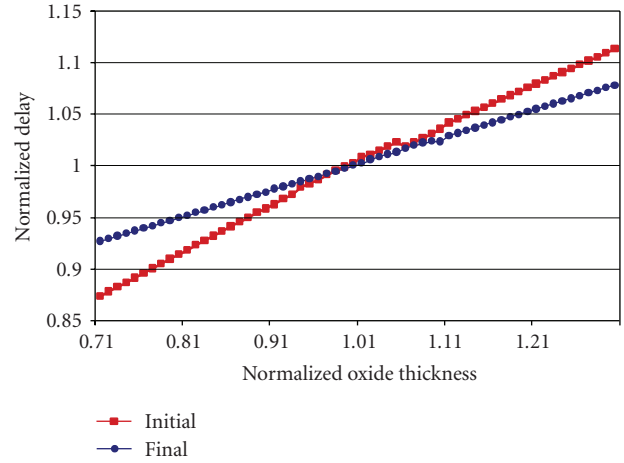


Figure 8: Delay Impact from Variation in Oxide Thickness.

Table 2. Prior to optimization, the worst-case delay of 2-b WBTC was 355 psec from path-1. The top 20% critical paths are path-1, 2, 5, 8, 26, and 29. Widths of all transistors in these critical paths are initially increased by a ratio of 1.1 to their initial values. For example, the sizes of transistors (T_{22}, T_{11}, T_4, and T_0) in path-1 are increased to $160 \times 1.1 = 176$ nm, $160 \times 1.1^2 = 193$ nm, $160 \times 1.1^3 = 213$ nm, $160 \times 1.1^4 = 234$ nm, respectively. After initial transistor sizing, process variations are considered in simulations in which delay distribution of each path is obtained. Then, transistor sizes are updated using (1)–(4), and simulations are performed to obtain a new critical path order. After several iterations of the LBMP timing optimization algorithm, the worst-case delay of 2-b WBTC is reduced and finally converged to 157 psec, accounting for a 55.77% improvement.

Efficiency of the LBMP algorithm is further illustrated through reduction in delay uncertainty (Δ). Figures 5 and 6 show the normalized delay distribution of the 2-b WBTC before and after optimization, respectively. It is clearly evident that delay uncertainty has reduced and distribution has been narrowed significantly in the optimized design. With major contributors towards delay uncertainty being gate length, channel width, capacitance, supply voltage, and threshold voltage [5], timing analysis was performed to categorize the impact of each. Figure 7 shows the reduction in delay uncertainty of 2-b WBTC from 14% to 8% due to variation in zero-bias junction capacitance.

Kinget in his work on device mismatch [27] has shown that variance in delay distribution of ΔV_T is dependent on device area, $W \times L$ as shown in (5), where W is the transistor width, L is the transistor channel length, and A_{VT} is the proportionality constant (a technology dependant value). Experimental results of delay impact from variation in oxide thickness are shown in Figure 8 where increasing device area after transistor sizing reduces the delay uncertainty of the 2-b WBTC from 24% to 15%. The other research [5] shows that a drop in supply voltage degrades cell timing at a quadratic rate; a 5% drop in total rail-to-rail voltage may result in a 15% timing degradation. Figure 9 shows the delay uncertainty of the 2-b WBTC before and after optimization using the LBMP algorithm. It is observed that a 20% drop in total rail-to-rail voltage (from 1.0 V to 0.8 V) results in a 4% variation in timing (much less than 15%), which

Table 1: Timing Paths in 2-b WBTC.

Path No.	Transistors	Path No.	Transistors
1	T_{28}, T_0, T_4, T_{11}, T_{22}, T_{26}	18	T_{28}, T_7, T_8, T_{12}, T_{18}
2	T_{28}, T_7, T_{11}, T_{22}, T_{26}	19	T_{28}, T_7, T_{11}, T_{15}, T_{18}
3	T_{28}, T_{19}, T_{22}, T_{26}	20	T_{28}, T_7, T_{11}, T_{17}, T_{21}
4	T_{28}, T_{24}, T_{26}	21	T_{28}, T_7, T_{11}, T_{20}, T_{21}
5	T_{28}, T_0, T_4, T_{11}, T_{17}, T_{23}	22	T_{28}, T_{14}, T_{15}, T_{18}
6	T_{28}, T_0, T_4, T_{11}, T_{20}, T_{23}	23	T_{28}, T_{14}, T_{17}, T_{21}
7	T_{28}, T_0, T_4, T_{11}, T_{22}, T_{25}	24	T_{28}, T_{19}, T_{20}, T_{21}
8	T_{28}, T_7, T_{11}, T_{17}, T_{23}	25	T_{28}, T_0, T_1, T_5, T_{13}
9	T_{28}, T_7, T_{11}, T_{20}, T_{23}	26	T_{28}, T_0, T_4, T_{11}, T_{15}, T_{16}
10	T_{28}, T_7, T_{11}, T_{22}, T_{25}	27	T_{28}, T_7, T_8, T_9, T_{13}
11	T_{28}, T_{14}, T_{17}, T_{23}	28	T_{28}, T_7, T_8, T_{12}, T_{16}
12	T_{28}, T_{19}, T_{20}, T_{23}	29	T_{28}, T_7, T_{11}, T_{15}, T_{16}
13	T_{28}, T_{19}, T_{22}, T_{25}	30	T_{28}, T_{14}, T_{15}, T_{16}
14	T_{28}, T_{24}, T_{25}	31	T_{28}, T_0, T_1, T_2, T_6
15	T_{28}, T_0, T_4, T_{11}, T_{15}, T_{18}	32	T_{28}, T_0, T_1, T_5, T_{10}
16	T_{28}, T_0, T_4, T_{11}, T_{17}, T_{21}	33	T_{28}, T_7, T_8, T_9, T_{10}
17	T_{28}, T_0, T_4, T_{11}, T_{20}, T_{21}	34	T_{28}, T_0, T_1, T_2, T_3

Table 2: Repeat and Weight Profiles of 2-b WBTC.

Repeats	Near GND						Near *Vdd*
16		T_{11}					
12	T_0, T_7						
8		T_4					
6				T_{17}, T_{22}	T_{15},T_{20}	T_{23}	T_{21}
4			T_1, T_8, T_{14}, T_{19}			T_{18},T_{26}	T_{16}, T_{25}
2				T_5, T_{12}	T_2,T_9,T_{24}	T_{13}	T_{10}
1						T_6	T_3, T_{27}
Weight	0.5	0.4	0.3	0.2	0.15	0.1	0.05

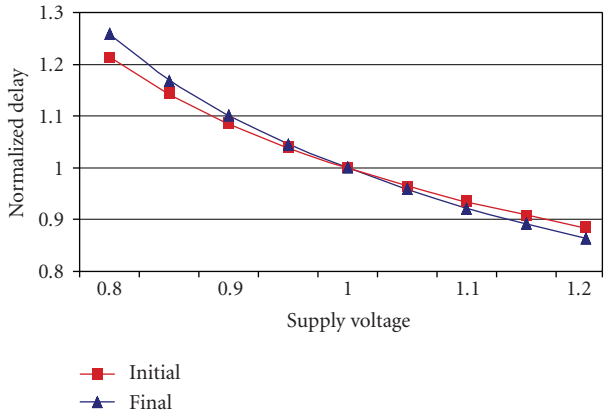


Figure 9: Delay Uncertainty from Variation in Supply Voltage.

further illustrates that the LBMP algorithm is less sensitive to variation in supply voltage:

$$\sigma^2(\Delta V_T) = \frac{{A_{VT}}^2}{W \cdot L}. \quad (5)$$

Another benchmark used to validate the algorithm is a 4-b Unity Weight BTC (UWBTC) that is used in high-performance digital-to-analog converters, as shown in Figure 10. Along with an increase in the number of transistors, the number of timing paths to be considered is also increased to 83. Prior to optimization, the 4-b UWBTC had a worst-case delay of 152 psec. Through iterative optimization using the LBMP algorithm, the worst-case delay of 4-b UWBTC was reduced to 103 psec, an improvement of 33%. Furthermore, the LBMP algorithm was also implemented on several ISCAS benchmark circuits of which the ratio of the number of critical paths to the number of transistors is shown in Table 3. Through implementation and verification in 130 nm CMOS process, in Table 4 the LBMP algorithm has shown an average delay reduction by 47.8%, uncertainty reduction by 48%, power increase by 13%, and an area increase by 39.8%. The delay convergence profiles of these circuits are shown in Figure 11.

As delay in general can be reduced by increasing power consumption [28], power-delay product (PDP) is a key evaluation parameter to compare the design performance among different circuit structures. Table 5 shows the PDP of benchmark circuits before and after optimization. Through

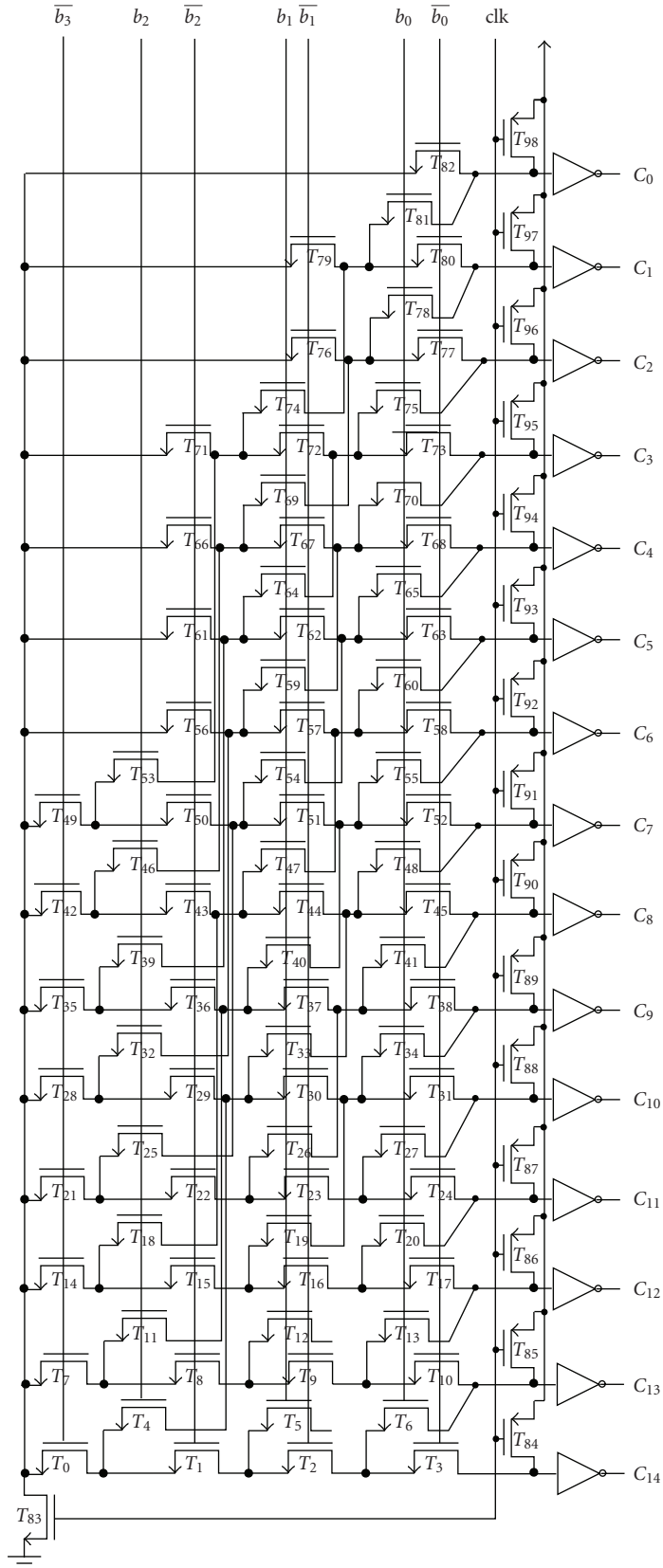


Figure 10: 4-bit Unity Weight BTC.

Table 3: Characteristics of Benchmark Circuits.

Design	# Inputs	# Outputs	# Paths	# Transistors
CCT-2	8	6	6	7
2-b WBTC	4	6	34	28
4-b UWBTC	4	15	83	83
74181-CLA	10	6	18	24
74181-E Mod	8	6	6	7
C2670-CLA	24	1	15	39
C3540-CC5	7	1	4	7
C3540-CC8	7	3	17	35
C3540-CC9	8	3	22	47
C3540-UM12-7	9	1	24	50
C5315-CalP2	6	1	7	14
C5315-GLC4_2	8	1	5	8
C5315-CB4	9	1	5	9
C7552-GLC5_1	8	1	4	9
C7552-CGC34_4	9	4	14	9
C7552-CGC17	6	1	3	5
C7552-CGC20	7	1	4	7

Table 4: Optimization results from the LBMP Algorithm.

Design	Initial delay (psec)	Final delay (psec)	Delay reduction (%)	Uncertainty reduction (%)	Power increase (%)	Area increase (%)
CCT-2	226	109	52	46.8	8.6	53.1
2-b WBTC	355	157	55	48.6	2.3	65.5
4-b UWBTC	152	103	33	63.6	14.0	12.6
74181-CLA	209	103	51	46.8	22.3	53.1
74181-E Mod	225	110	51	47.3	9.4	54.2
C2670-CLA	391	206	47	52.7	14.4	58.5
C3540-CC5	144	77	46	61.6	13.2	43.3
C3540-CC8	427	216	50	53.5	14.2	62.1
C3540-CC9	341	202	41	64.3	18.4	32.1
C3540-UM12-7	485	178	63	43.7	n/a	33.4
C5315-CalP2	230	137	39	40.0	2.7	9.2
C5315-GLC4_2	197	122	38	40.4	18.4	12.2
C5315-CB4	243	134	45	48.3	15.2	32.3
C7552-GLC5_1	196	95	52	34.3	17.5	62.8
C7552-CGC34_4	468	161	65	67.7	9.5	51.8
C7552-CGC17	136	84	39	35.0	14.2	21.3
C7552-CGC20	144	78	46	24.4	13.4	19.3
Average (%)			**47.8**	**48.1**	**13.0**	**39.8**

optimal sizing of transistor widths, the proposed LBMP timing optimization algorithm has reduced the PDP by an average of 40.17%.

The other electronic performance measurement associated with timing optimization is delay sensitivity (∂) due to process variations. Traditionally, CMOS device switching speed improves at a lower temperature due to increase in mobility. However, Negative Bias Temperature Instability (NBTI) effects may degrade the device switching speed over time via threshold voltage shifts in PMOS transistors [29, 30], even at a lower temperature. The delay sensitivities of several ISCAS benchmark circuits, due to process variations at different temperatures are reported in Figure 12. It is observed that all circuits after timing optimization have a very little difference in the delay sensitivity reduction for different temperatures. The LBMP timing optimization provides consistent delay sensitivity at different temperatures.

Table 5: Power Delay Product optimization results from the proposed algorithm.

Design	Initial PDP (fWs)	Final PDP (fWs)	PDP Reduction (%)
CCT-2	3.57	1.87	47.49
2-b WBTC	13.66	6.185	54.74
4-b UWBTC	6.61	5.11	22.57
74181-CLA	5.04	3.04	39.55
74181-E Mod	3.55	1.90	46.46
C2670-CLA	2.97	1.75	41.07
C3540-CC5	1.09	0.66	38.78
C5315-CalP2	7.47	4.57	38.78
C5315-GLC4_2	3.68	2.70	26.48
C5315-CB4	4.61	2.93	36.43
C7552-CGC34_4	16.42	6.19	62.26
C7552-CGC17	1.63	1.15	28.97
C7552-CGC20	1.09	0.67	38.70
	Average (%)		40.17

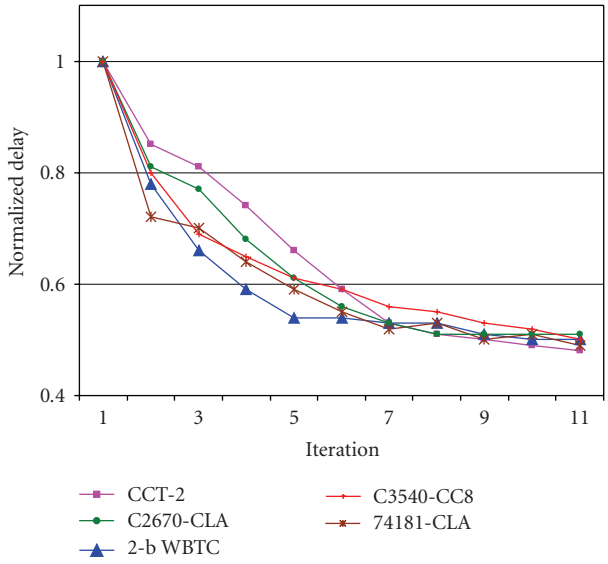


Figure 11: Delay Convergence using LBMP Algorithm.

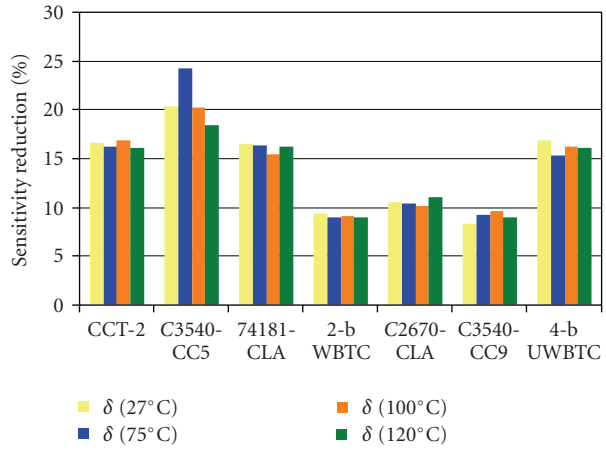


Figure 12: Delay Sensitivity Reduction.

5. Timing Optimization of Mixed-Static-Dynamic Circuits

Conventionally, synthesis tools perform design and optimization using static CMOS logic [31, 32]. It is not uncommon for the synthesis tools to not find an acceptable solution in terms of timing. This challenge can be answered through utilizing the advantage of fast timing in dynamic logic. Dynamic logic has smaller gate capacitances compared to their static CMOS counterparts, which accounts for a significant speedup [3, 33]. With static and dynamic logic having their respective advantages of low power and low delay, an optimal balance can be obtained by partitioning the design to use both static and dynamic logic in an effective manner.

At the architecture level, a common limitation in most design optimization flows is the limited accountability for process variations. Typically after placement and route, if a design fails to meet the timing constraints, optimization flow is reiterated. Even after several iterations, design may still not meet the timing constraint and miss the time-to-market window. The process variation-aware Path Oriented IN Time (POINT) optimization algorithm proposed in Figure 13 answers these challenges of timing optimization and also accounts for process variations. Utilizing the LBMP algorithm proposed in Section 3, the POINT optimization algorithm partitions the design to effectively utilize both dynamic and static CMOS logic to meet the timing constraints.

Initially, a high-level description of a design is input to Synopsys Design Compiler (SDC) [31] for synthesis and optimization. The optimized designs from SDC are considered as the initial case for POINT optimization flow. Following synthesis and optimization, static timing

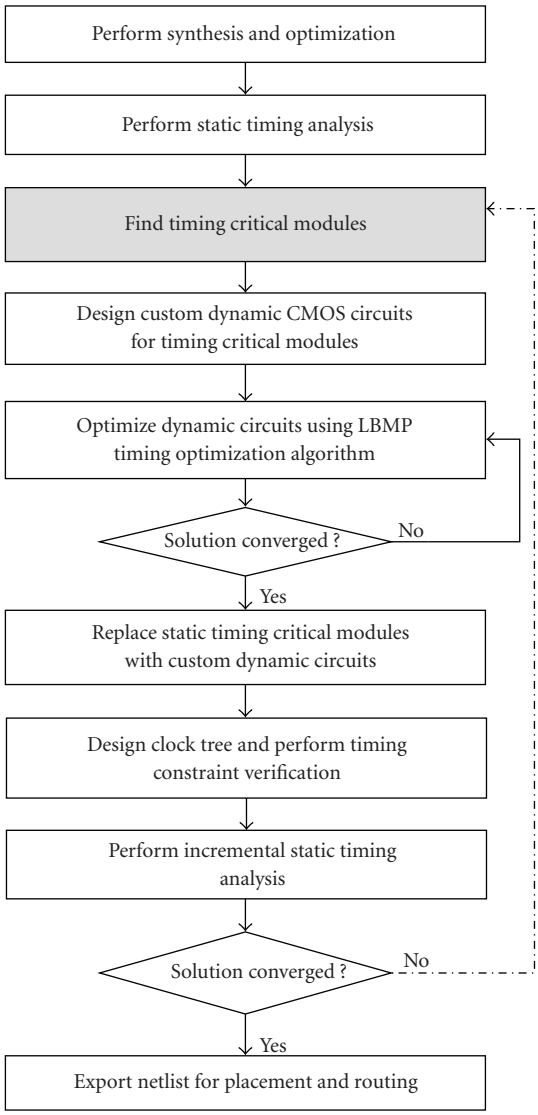


Figure 13: POINT Optimization Algorithm.

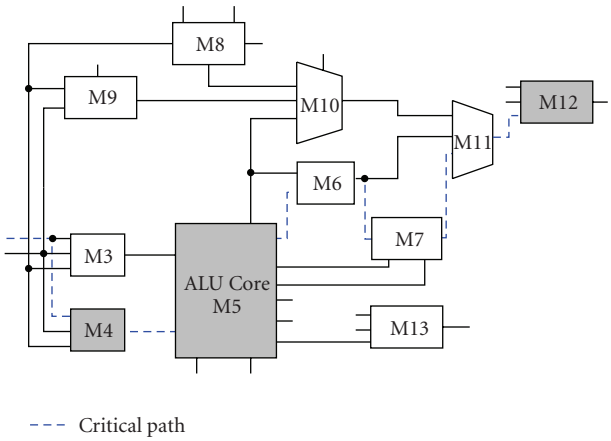


Figure 14: ISCAS benchmark–C3540 (8-b ALU).

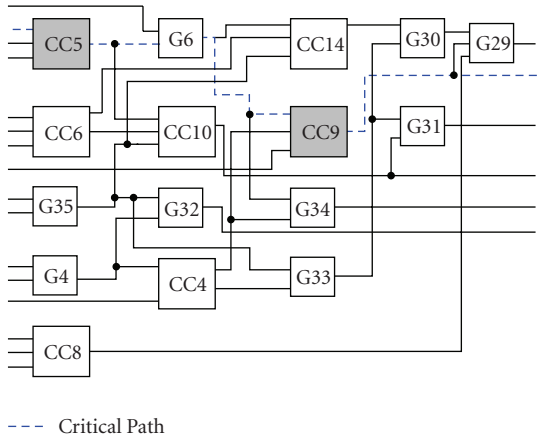


Figure 15: Architecture of M5/UM5_6 in C3540.

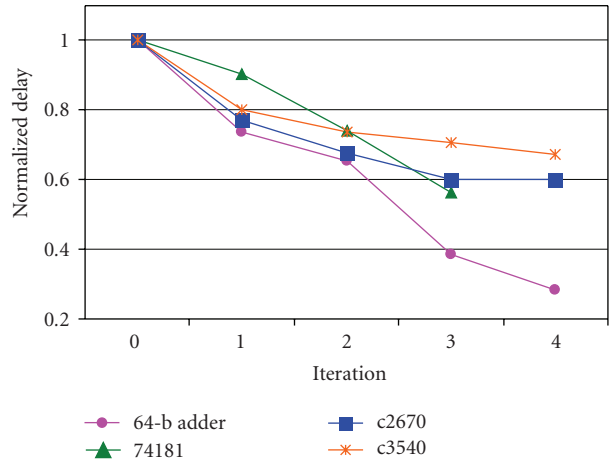


Figure 16: Delay Convergence using POINT Optimization Flow.

analysis (STA) is performed using Synopsys PrimeTime (SPT) to identify the critical paths. Also, the critical timing modules identified by the number of occurrences, and delay significance on the critical paths are reported and dynamic circuits of the same are designed. Using the LBMP algorithm, iterative transistor sizing optimization for timing is performed on these dynamic circuits.

With the updated design comprising of dynamic logic circuits, clock tree design and timing verification is performed. After the design is verified for clock signal timing constraints, incremental STA is performed to verify for timing convergence. The algorithm is iteratively repeated towards convergence of acceptable solution. Following the timing convergence through iterations, the final mixed-static-dynamic circuit design is exported for placement and route.

The POINT optimization algorithm is verified through implementation on several ISCAS benchmark circuits, including C3540, an 8-b ALU as shown in Figure 14 [34]. Initial synthesis and optimization was performed using SDC, and static timing analysis was performed using SPT

Table 6: POINT Optimization Flow results.

Design	# Inputs	# Outputs	# Gates	Delay reduction (%)	Uncertainty reduction (%)
74181	14	8	74	43	14
c2670	233	140	1193	39	32
adder64	130	65	1491	61	63
c3540	50	22	1669	32	40
c5315	178	123	2406	29	32
c7552	207	108	3512	26	31
Average (%)				**38**	**35**

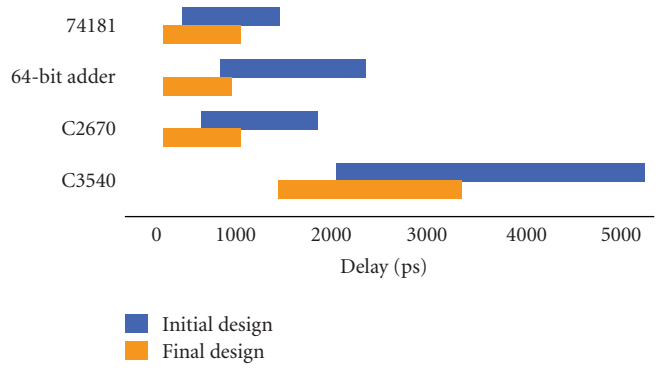


Figure 17: Timing Profiles using POINT Optimization Algorithm.

[35]. For the design in hierarchical format (synthesis and optimization was performed at block level, and design flatten option was disabled), the critical path delay was found to be 3.6 nanoseconds. The critical modules and the critical paths obtained from STA are highlighted in Figure 14. Based on the STA report, it is shown that the ALU Core-M5 with a delay of 1.24 nanoseconds is the timing critical module with the most number of worst-case paths. Figure 15 shows the schematic of UM5_6 from ALU Core-M5 with the critical paths highlighted; the submodules labeled CC5 and CC9 are timing critical with delays of 0.5 nanoseconds and 0.61 nanoseconds respectively.

With timing optimization being the primary goal in this stage, submodules CC5 and CC9 in M5/UM5_6 of C3540 are designed in dynamic logic, and timing optimization is performed using the LBMP algorithm. With dynamic circuits optimized using LBMP algorithm, the delay of CC5 was reduced from 0.5 nanoseconds to 0.07 nanoseconds, and delay of CC9 was reduced from 0.61 nanoseconds to 0.20 nanoseconds, respectively. After the first iteration of the POINT optimization flow, the critical path delay of C3540 was reduced from 3.6 nanoseconds to 2.8 nanoseconds. Further iterations of POINT optimization flow reduced the critical path delay from 3.6 nanoseconds to 2.4 nanoseconds, as shown in Figure 16. In addition to reducing the delay by 33%, the delay uncertainty due to process variations is also reduced by 40% as shown in Figure 17.

Similarly, the process variation-aware POINT optimization was implemented on other benchmark circuits and timing optimization results are presented in Table 6, where both delay and uncertainty from process variations were reduced by an average of 38% and 35%, respectively, over initial designs optimized with Synopsys Design Compiler.

6. Conclusion

In this paper a process variation-aware timing optimization of dynamic logic and a timing optimization flow for mixed-static-dynamic CMOS logic have been presented. Solutions addressing further design challenges are presented by (1) considering delay uncertainties from process variations and (2) developing a process variation-aware Path Oriented IN Time (POINT) optimization algorithm for mixed-static-dynamic logic.

Through implementation and verification of several benchmark circuits in 130 nm CMOS process, the process variation-aware timing optimization algorithm has shown by average a delay reduction by 47.8%, an uncertainty reduction by 48%, a power increase by 13%, and a power-delay-product reduction by 40%. Validated through implementation of mixed-static-dynamic logic on a 64-b adder and several ISCAS benchmark circuits, the POINT optimization algorithm has demonstrated an average improvement in delay reduction by 38% and delay uncertainty reduction from process variation by 35%.

7. Acknowledgment

The authors wish to thank the anonymous reviewers for insightful comments to improve the quality of presentation.

References

[1] D. H. Allen, S. H. Dhong, H. P. Hofstee, et al., "Custom circuit design as a driver of microprocessor performance," *IBM Journal of Research and Development*, vol. 44, no. 6, pp. 799–822, 2000.

[2] P. E. Gronowski, W. J. Bowhill, R. P. Preston, M. K. Gowan, and R. L. Allmon, "High-performance microprocessor design," *IEEE Journal of Solid-State Circuits*, vol. 33, no. 5, pp. 676–685, 1998.

[3] M. Zhao and S. S. Sapatnekar, "Timing-driven partitioning and timing optimization of mixed static-domino implementations," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 19, no. 11, pp. 1322–1336, 2000.

[4] L. Zhang, *Statistical timing analysis for digital circuit design*, Ph.D. dissertation, December 2005.

[5] P. McGuinness, "Variations, margins, and statistics," in *Proceedings of the International Symposium on Physical Design*, pp. 60–67, Portland, Ore, USA, April 2008.

[6] J. Tschanz, K. Bowman, and V. De, "Variation-tolerant circuits: circuit solutions and techniques," in *Proceedings of Design Automation Conference*, pp. 762–763, 2005.

[7] P. S. Zuchowski, P. A. Habitz, J. D. Hayes, and J. H. Oppold, "Process and environmental variation impacts on ASIC timing," in *Proceedings of IEEE/ACM International Conference on Computer Aided Design (ICCAD '04)*, pp. 336–342, November 2004.

[8] S. B. Samaan, "The impact of device parameter variations on the frequency and performance of VLSI chips," in *Proceedings of IEEE/ACM International Conference on Computer Aided Design (ICCAD '04)*, pp. 343–346, 2004.

[9] K. Yelamarthi and C.-I. H. Chen, "A path oriented in time optimization flow for mixed-static-dynamic CMOS logic," in *Proceedings of the 51st Midwest Symposium on Circuits and Systems*, pp. 454–457, Knoxville, Tenn, USA, August 2008.

[10] J. P. Fishburn and A. E. Dunlop, "TILOS: a posynomial programming approach to transistor sizing," in *Proceedings of IEEE International Conference on Computer Aided Design (CCAD '85)*, pp. 326–328, Santa Clara, Calif, USA, 1985.

[11] V. Sundararajan, S. S. Sapatnekar, and K. K. Parhi, "Fast and exact transistor sizing based on iterative relaxation," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 21, no. 5, pp. 568–581, 2002.

[12] M. Borah, R. M. Owens, and M. J. Irwin, "Transistor sizing for minimizing power consumption of CMOS circuits under delay constraint," in *Proceedings of the International Symposium on Low Power Design*, pp. 167–172, Dana Point, Calif, USA, April 1995.

[13] S.-O. Jung, K.-W. Kim, and S.-M. Kang, "Transistor sizing for reliable domino logic design in dual threshold voltage technologies," in *Proceedings of the 11th Great Lakes Symposium on VLSI (GLSVLSI '01)*, pp. 133–138, West Lafayette, Ind, USA, March 2001.

[14] Z. Luo, "General transistor-level methodology on VLSI low-power design," in *Proceedings of the ACM Great Lakes Symposium on VLSI (GLSVLSI '06)*, pp. 115–118, Philadelphia, Pa, USA, April 2006.

[15] A. R. Conn, I. M. Elfadel, W. W. Molzen Jr., et al., "Gradient-based optimization of custom circuits using a static-timing formulation," in *Proceedings of Design Automation Conference*, pp. 452–459, June 1999.

[16] I. Sutherland, B. Sproull, and D. Harris, *Logical Effort: Designing Fast CMOS Circuits*, Morgan Kaufmann, San Francisco, Calif, USA, 1999.

[17] N. Weste and D. Harris, *CMOS VLSI Design: A Circuits and Systems Perspective*, Addison Wesley, Boston, Mass, USA, 3rd edition, 2004.

[18] K. A. Bowman, S. G. Duvall, and J. D. Meindl, "Impact of die-to-die and within-die parameter fluctuations on the maximum clock frequency distribution for gigascale integration," *IEEE Journal of Solid-State Circuits*, vol. 37, no. 2, pp. 183–190, 2002.

[19] M. Orshansky, *Increasing Circuit Performance through Statistical Design Techniques*, Closing the Gap between ASIC & Custom, Kluwer Academic Publishers, Dordrecht, The Netherlands, 2003.

[20] D. Burnett, K. Erington, C. Subramanian, and K. Baker, "Implications of fundamental threshold voltage variations for high-density SRAM and logic circuits," in *Proceedings of the Symposium on VLSI Technology*, pp. 15–16, Honolulu, Hawaii, USA, June 1994.

[21] K. Takeuchi, T. Tatsumi, and A. Furukawa, "Channel engineering for the reduction of random-dopant-placement-induced threshold voltage fluctuation," in *Proceedings of the IEEE Electron Devices Meeting (IDEM '97)*, pp. 841–844, Washington, DC, USA, December 1997.

[22] S. Borkar, T. Karnik, S. Narendra, J. Tschanz, A. Keshavarzi, and V. De, "Parameter variations and impact on circuits and microarchitecture," in *Proceedings of Design Automation Conference*, pp. 338–342, 2003.

[23] C. H. Kim, K. Roy, S. Hsu, R. Krishnamurthy, and S. Borkar, "A process variation compensating technique with an on-die leakage current sensor for nanometer scale dynamic circuits," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 14, no. 6, pp. 646–649, 2006.

[24] L. Scheffer, "The Count of Monte Carlo," in *Proceedings of the ACM/IEEE International Workshop on Timing Issues in the Specification and Synthesis of Digital Systems (TAU '04)*, February 2004.

[25] F. Maloberti and C. Gang, " Performing arithmetic functions with the Chinese abacus approach," *IEEE Transactions on Circuits and Systems II*, vol. 46, no. 12, pp. 1512–1515, 1999.

[26] B. Fu, Q. Yu, and P. Ampadu, "Energy-delay minimization in nanoscale domino logic," in *Proceedings of the 16th ACM Great Lakes Symposium on VLSI (GLSVLSI '06)*, pp. 316–319, Philadelphia, Pa, USA, April 2006.

[27] P. R. Kinget, "Device mismatch and tradeoffs in the design of analog circuits," *IEEE Journal of Solid-State Circuits*, vol. 40, no. 6, pp. 1212–1224, 2005.

[28] W. Wolf, *Modern VLSI Design: IP-Based Design*, Prentice Hall, Upper Saddle River, NJ, USA, 4th edition, 2008.

[29] B. Lasbouygues, R. Wilson, N. Azemard, and P. Maurine, "Timing analysis in presence of supply voltage and temperature variations," in *Proceedings of the International Symposium on Physical Design*, pp. 10–16, 2006.

[30] W. Wang, S. Yang, S. Bhardwaj, et al., "The impact of NBTI on the performance of combinational and sequential circuits," in *Proceedings of the 44th Annual Design Automation Conference*, pp. 364–369, 2007.

[31] Synopsys Design Compiler, http://www.synopsys.com/.

[32] Cadence Encounter, http://www.cadence.com/.

[33] R. Puri, "Design issues in mixed static-dynamic circuit implementation," in *Proceedings of International Conference on Computer Design*, pp. 270–275, 1998.

[34] M. C. Hansen, H. Yalcin, and J. P. Hayes, "Unveiling the ISCAS-85 benchmarks: a case study in reverse engineering," *IEEE Design and Test of Computers*, vol. 16, no. 3, pp. 72–80, 1999.

[35] Synopsys PrimeTime, http://www.synopsys.com/.

Hindawi Publishing Corporation
VLSI Design
Volume 2010, Article ID 451809, 9 pages
doi:10.1155/2010/451809

Research Article

Post-CTS Delay Insertion

Jianchao Lu and Baris Taskin

Department of Electrical and Computer Engineering, Drexel University, Philadelphia, PA 19104, USA

Correspondence should be addressed to Jianchao Lu, jl597@drexel.edu and Baris Taskin, taskin@coe.drexel.edu

Received 29 May 2009; Revised 23 October 2009; Accepted 18 November 2009

Academic Editor: Gregory D. Peterson

A post-clock-tree-synthesis (post-CTS) optimization method is proposed that suggests delay insertion at the leaves of the clock tree in order to implement a limited version of clock skew scheduling. Delay insertion is limited on each clock tree branch simultaneous with a global monitoring of the total amount of delay insertion. The delay insertion for nonzero clock skew operation is performed only at the clock sinks in order to preserve the structure and the optimizations implemented in the clock tree synthesis stage. The methodology is implemented as a linear programming model amenable to two design objectives: fixing timing violations or optimizing the clock period. Experimental results show that the clock networks of the largest ISCAS'89 circuits can be corrected post-CTS to resolve the timing conflicts in approximately 90% of the circuits with minimal delay insertion (0.159 $\times$ clock period per clock path on average). It is also shown that the majority of the clock period improvement achievable through unrestricted clock skew scheduling are obtained through very limited insertion ($\approx$43% average improvement through 10% of max insertion).

1. Introduction

One of the tools at the designers' expense during the design of high performance ASIC circuits is the manipulation of clock delays to compensate for the timing critical paths at the physical design stage. After power and timing aware physical design steps of floorplanning, placement, and clock tree synthesis steps, timing verification can still reveal a number of violated paths, which might need an overall redesign of the system or iterative physical design steps to be resolved. Post-clock-tree-synthesis (post-CTS) optimization can be used to resolve such violated paths or to improve the clock period. The two objectives are considered in this paper.

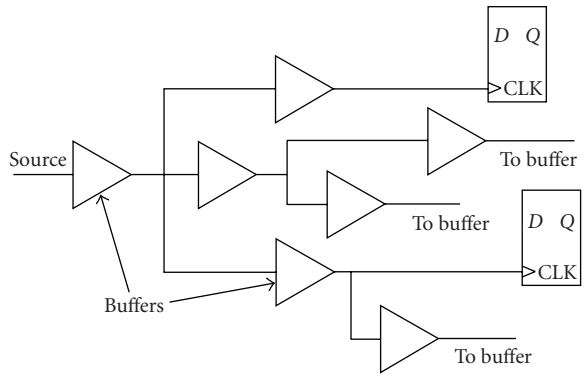
In particular, a practical delay insertion process to be performed on a synthesized clock tree is introduced. This process is devised to work with industry standard automation tools, such that, the clock distribution network (i.e., clock tree) and, the placement results are the inputs to the proposed methodology. The timing verification tools are used to detect the violations on the data paths. These violations are eliminated (i.e., fixed) by inserting small delay elements on the clock branches. For circuits where timing is satisfied (no timing violations), delay mismatch can be used to implement a limited version of clock skew scheduling in order to improve the operating clock frequency [1]. A systematic study of the effectiveness of the delay insertion method in both fixing timing violations and improving circuit frequency is presented. The formulation and mathematical analysis of the post-CTS delay insertion on clock leaves are presented that

(i) preserves the structure of the zero clock skew tree,

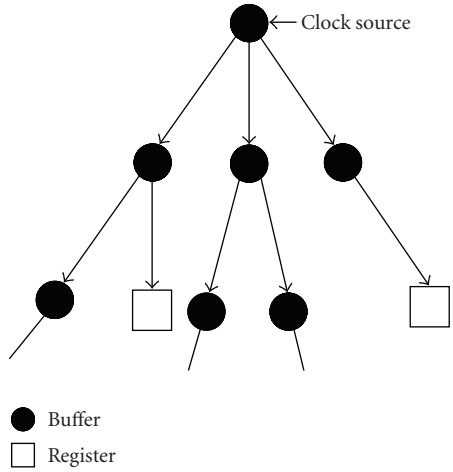
(ii) limits the amount of insertion on each clock branch,

(iii) limits the amount of insertion on the overall clock tree.

Existing delay insertion methods, including [2], only limit the amount of delay insertion on clock branches. Such a limitation per branch is not optimal as there are often paths that do not require any delay insertion. The available space on the chip can be utilized more efficiently by permitting higher levels of delay insertion on each branch while simultaneously monitoring the total amount of delay insertion (such that the available space is not overused). Existing clock skew scheduling methods, including [2], are implemented with continuous delay models and do not limit the delay insertion, which are not practical. Consequently, practical implementation of clock skew scheduling resorts to suboptimal, iteration-based delay insertion procedures which are rarely methodical. The post-CTS delay insertion proposed in this paper constitutes a methodical and practical implementation of clock skew scheduling.



(a) Circuit structure of the clock distribution network



(b) Equivalent graph of the clock tree that corresponds to the circuit in (a)

Figure 1: Tree structure of a clock distribution network.

This paper is organized as follows. In Section 2, the timing constraints are reviewed and a brief description of the clock tree is introduced. In Section 3, the motivation of this paper is explained. In Section 4, the proposed post-CTS optimization methodology is demonstrated. In Section 5, experimental results on a suite of ISCAS'89 benchmark circuits are presented. The paper is finalized in Section 6.

2. Technical Background

The timing constraints of a synchronous local data path are used as a part of the proposed mathematical framework to perform post-CTS delay insertion. In Section 2.1, the clock network design process is outlined as in relevance to this work. In Section 2.2, these timing constraints of a synchronous local data path are reviewed.

2.1. Clock Network Design. Clock network design (also called clock tree synthesis) [3] is an essential step in the physical design flow of integrated circuits. During the clock network design step, the interconnect topology of the clock distribution network is designed based on the placement and routing information. The clock distribution network

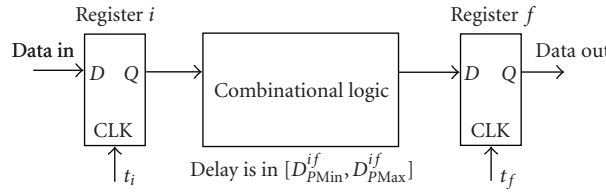


Figure 2: Setup and hold constraints.

is frequently organized as a rooted tree structure [4, 5], as illustrated in Figure 1. A circuit schematic of a clock distribution network is shown in Figure 1(a). An abstract graphical representation of the tree structure is shown in Figure 1(b). The clock signal is distributed from the source to every register in the circuit through a sequence of *buffers and interconnect wires*. Such minimal or zero clock skew can be achieved by different routing strategies [6–9], buffered clock tree synthesis, symmetric n-ary trees [10] (most notably H-trees), using deskew buffers [11] or a distributed series of buffers connected as a mesh [12].

In this work, a generic tree implementation as shown in Figure 1 is considered. The proposed optimization methodology is performed *post-CTS*, thus, the synthesis of the clock tree and sizing of the buffers are considered complete. Consequently, any clock tree synthesis methodology or tool can be used for the clock tree synthesis process.

2.2. Static Timing Constraints. As shown in Figure 2, minimum and maximum propagation delays on the combinational path from register R_i to register R_f are denoted by D^{if}_{PMin} and D^{if}_{PMax}, respectively. The clock arriving time of a register R_i is denoted by t_i; whereas the setup and hold times are denoted by S_i and H_i, respectively. The clock arriving time t_i represents the clock signal delay from the source to register R_i at that branch. The clock period is denoted by T. The clock to output delay of each register is D^i_{CQ}. The timing analysis of a synchronous circuit is performed by satisfying the *setup* timing constraints for each local data path:

$$\text{Setup: } t_i + D^i_{CQ} + D^{if}_{PMax} \le t_f + T - S_f, \tag{1}$$

$$\text{Hold: } t_i + D^i_{CQ} + D^{if}_{PMin} \ge t_f + H_f. \tag{2}$$

For zero clock skew systems, clock delays t_i and t_f are identical

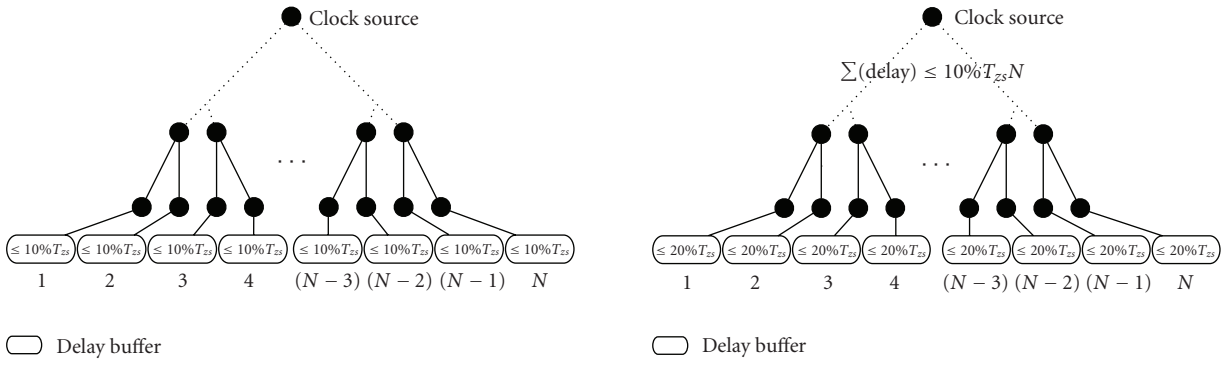
$$t_i = t_f \Longrightarrow t_i - t_f = 0. \tag{3}$$

This equality of clock delays to registers simplifies the timing constraints. Further assuming that the internal register delays can be neglected ($D_{CQ} = S = H = 0$), a limitation on the clock period T is derived from(1)

$$\text{Setup: } D^{if}_{PMax} \le T. \tag{4}$$

The setup constraint must be satisfied on all timing paths, leading to the following inequality:

$$\max_{\forall(i,j)} \left(D^{ij}_{\max}\right) = T_{zs} \le T. \tag{5}$$



(a) Delay insertion in multidomain scheduling [2]

(b) Proposed delay insertion with the additional constraint

Figure 3: Post-CTS delay insertion examples on a sample binary clock tree with N sinks.

Thus, if the circuit operates at any clock period less than the largest maximum data propagation time, a *timing violation* occurs [13]. Finding a clock period T for a zero clock skew circuit is always possible, making it convenient to design zero clock skew systems. Consequently, the application of zero clock skew schemes has been central to the design of fully synchronous digital circuits for decades [4]. The minimum clock period at zero skew T_{zs} is defined at the equality condition for inequality (5) and is used in the formulations as the basis metric to measure the improvement through clock skew scheduling.

3. Motivation

The proposed methodology of delay insertion at the leaves of the clock tree is a limited version of *clock skew scheduling*. Clock skew scheduling permits the modification of the clock delays to be different from each other, leading to a nonzero clock skew system

$$t_i \neq t_f. \tag{6}$$

The clock arrival time t_i might be less or greater than t_f, causing more time to the path between R_i and R_f, or the paths leading to R_i. The advantages of clock skew scheduling are well known and documented extensively in the literature [1]. The minimum clock period of a circuit with zero clock skew is the largest logic path delay in that circuit (5) and with clock skew scheduling, the minimum clock period can be improved on average by 30% [1] through rearranging the *slack time* between the long and short paths. Clock skew scheduling improvements are described for unbounded amount of clock delays. In other words, t_i and t_f are modeled as continuous variables thereby allowing clock tree delays to have different values. In practice, clock delays can be changed by only a limited amount. This limitation is due to the size and discreteness of the delay values available. Consequently, the limited amount of delay insertion problem presented in this work is a practical implementation of clock skew scheduling. The proposed methodology introduces simultaneous limitations on delay insertion per branch and per clock tree. The simultaneous limitations are proposed in order to more accurately reflect the practical limitations of an integrated circuit; that, the integrated circuit has a limited amount of area for delay buffering, which can be unevenly distributed between each clock branch. The limitation per clock tree is representative of the available space. The limitation per branch is to prevent exorbitant delay insertion on one branch. In this paper, the delay insertion method is explored with two purposes: (1) to fix the timing violations and (2) to optimize the circuit frequency with a very limited amount of delay insertion.

3.1. Challenge 1: Timing Violations. As the minimum feature size of VLSI circuits continues to shrink, process variations have become significantly worse [14]. The delay variations on clock network branches, for instance, correspond to 10% of their nominal value for deep sub-micron technologies [15]. This trend for global skew mismatches in recent microprocessors has been well documented [16]. Furthermore, the increasing functionality and speed of operation require a smaller clock period, which further complicate the timing closure of integrated circuits. Physical design tools are optimized to satisfy timing in presence of variations and the increasing clock frequencies. However, in practice, timing violations remain that require engineering change order (ECO) changes, such as the post-CTS methodology described in this paper.

3.2. Challenge 2: Clock Period Optimization. Consider the sample clock tree with N sinks shown in Figure 3. The clock tree is a balanced binary tree synthesized for a zero clock skew operation (without the delay buffers). The multidomain clock skew scheduling methodology [2] suggests the definition of multiple clock domains and the limitation of clock skew on each clock domain to a fixed percentage of the (zero clock skew) clock period T_{zs}. Consider that a single clock domain is selected for simplicity and the clock delay variation limit is set to 10% of the zero clock skew clock period. Such a limitation means a maximum skew of 10% $\times$ T_{zs} to be observed on the clock tree. In the *worst* case, the proposed delay insertion will be performed on $N - 1$ of

the N clock branches. This is such, as maximum insertion on each of the N branches would result in zero clock skew, which can be achieved with zero delay insertion as well. In this worst case, the total amount of insertion corresponds to a total delay insertion of $10\% \times T_{zs} \times (N-1)$. It is more advantageous to use the insertion area corresponding to a total of $10\% \times T_{zs} \times (N-1)$ time units as follows.

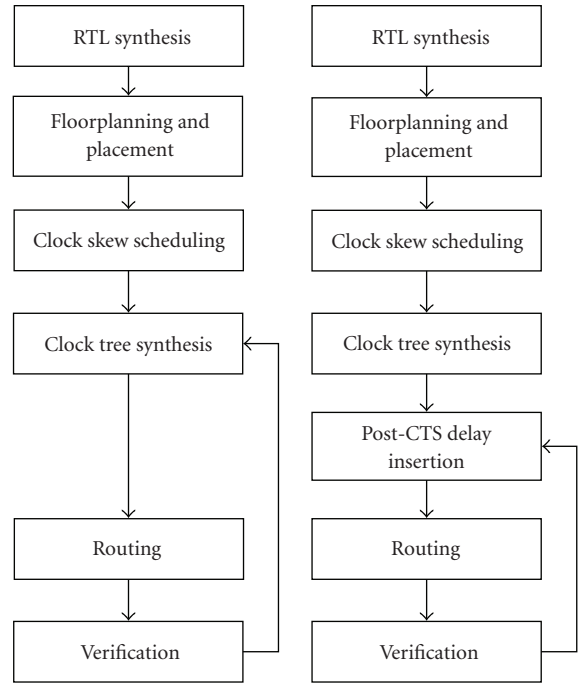
Instead of constraining the amount of insertion on each clock branch to a smaller number (e.g., 10%) that guarantees the overall insertion limitation in the worst case, the limitations on each branch are held more flexible. The adherence to the overall delay insertion budget is maintained with a general constraint that controls all of the branches at the same time. In other words, the sum of all delay insertion on each branch is limited to the same amount of $10\% \times T_{zs} \times (N-1)$; however, the limitation on each branch is raised to a higher amount. Under the proposed scheme, some clock branches can be allocated more than the $10\% \times T_{zs}$ delay insertion whereas only a fraction of the clock branches can have a high (or maximum) delay insertion. Assume that this fraction is selected to be 0.5, thus, in the delay insertion process depicted in Figure 3(b), the maximum delay insertion on each branch is raised to $20\% \times T_{zs}$ (from $10\% \times T_{zs}$) but only half of the clock branches are allowed to accommodate the maximum delay insertion. For a high number of registers N, the overall delay insertion is approximately the same, that is, $10\% \times T_{zs} \times (N-1) \approx 20\% \times T_{zs} \times N/2$.

4. Proposed Methodology

The traditional design flow with clock skew scheduling and the design flow with proposed method are illustrated in Figures 4(a) and 4(b), respectively. The proposed methodology analyzes each branch of a presynthesized clock tree (post-CTS) to explore the possibility of additional delay insertion only on the clock leaves. The proposed additional insertion is performed only to take place at the clock leaves, which are the sinks of the clock tree topology. Such delay insertion is advantageous in preserving most of the automated optimizations during the clock tree synthesis stage. It also requires less effort in order to fix the timing violations after verification since the new CTS step might not be necessary in the flow in Figure 4(b).

In the rest of the discussion and in experimentation, a zero skew clock tree is considered as the output of the clock tree synthesis step, and thus, the input to the proposed post-CTS delay insertion process. This simplification reflects the mainstream practice in clock tree synthesis in minimizing the clock skew subject to the system resource constraints (e.g., power, area, etc.). Nonetheless, the generality of the proposed discussion still holds for an arbitrary clock tree and slight modifications can be performed to handle any arbitrary tree.

In Section 4.1, a linear programming formulation is presented to fix timing violations. In Section 4.2, the mathematical framework proposed for clock period minimization is presented. In Section 4.3, discussions are presented based on presented formulations.



(a) Traditional clock skew scheduling design flow

(b) Design flow with proposed post-CTS stage

Figure 4: Integrated Circuit Design flow.

4.1. Formulation 1: Delay Insertion to Fix Timing Violations. In mainstream IC design, automated placement and CTS tools are used to compute the physical implementation of the circuit. The logic and memory elements are placed with timing, congestion and, power driven objectives. The clock tree is implemented in one of the forms described in Section 2.1 to deliver identical delays to each synchronous component. Despite aggressive optimizations, however, clock network is still subject to random (and systematic) variations. These variations cause small shifts in the clock delays, leading to skew mismatches and potentially timing violations. A delay insertion method is proposed to be performed after the clock tree synthesis step (post-CTS) to fix the timing violations. The problem definition is

> *Given a pre-computed placement, and a synthesized clock tree of an IC (thus, given the clock delay t_i of each branch, clock period T_{zs}, local data path propagation time $[D_{PMin}, D_{PMax}]$ of each local data path, internal register delays S, H and D_{CQ} of each register), compute the minimum amount of delay Δ_i to be inserted on each clock tree branch in order to eliminate timing violations, considering upper bounds for delay insertion per branch and total delay insertion.*

Note that, typically the last stage buffers of a clock tree drive more than one register. The presented formulation can be easily changed to reflect this requirement. For simplicity of presentation, each leaf buffer is selected to drive only one synchronous component.

Table 1: LP model for post-CTS delay insertion method.

Minimize inserted delay	
min	$\sum_{i=0}^{N-1} \Delta_i$
s.t.	$t_i + \Delta_i + D^i_{\mathrm{CQ}} + D^{if}_{\mathrm{PMax}} \le t_f + \Delta_f + T_{zs} - S^f$
	$t_i + \Delta_i + D^i_{\mathrm{CQ}} + D^{if}_{\mathrm{PMin}} \ge t_f + \Delta_f + H^f$
	$\Delta_i \le k_1 T_{zs}$
	$\sum_{i=0}^{N-1} \Delta_i \le k_2 T_{zs} N$

The mathematical formulation for this problem is derived as an Linear Programming (LP) form. After post-CTS delay insertion, (1) and (2) can be written as

$$\text{Setup} : t_i + \Delta_i + D^i_{\mathrm{CQ}} + D^{if}_{\mathrm{PMax}} \le t_f + \Delta_f + T_{zs} - S^f, \quad (7)$$

$$\text{Hold} : t_i + \Delta_i + D^i_{\mathrm{CQ}} + D^{if}_{\mathrm{PMin}} \ge t_f + \Delta_f + H^f, \quad (8)$$

where the added term Δ_i is the delay element on clock tree branch driving R_i. We also assume two practical limitations on the delay insertion process. First, we assume that the amount of delay to be inserted on a clock tree branch has an upper bound proportional to the overall clock period T_{zs}:

$$\Delta_i \le k_1 T_{zs}, \quad (9)$$

where k_1 is a design parameter. Second, we assume that the total amount of delay to be inserted ($\sum_{i=0}^{N} \Delta_i$) has an upper bound proportional to the clock period T_{zs} and the number of registers N in the circuit:

$$\sum_{i=0}^{N-1} \Delta_i \le k_2 T_{zs} N, \quad (10)$$

where k_2 is a design parameter. In a practical implementation, k_1 and k_2 can be determined by evaluating the physical design information such as the area utilization, the number of clock tree levels, and the power dissipation budget.

The LP model is shown in Table 1. The objective is to minimize the total amount delay insertion. The first two set of constraints are the setup and hold time constraints, respectively, defined for each local data path. The third set of constraints is the delay insertion upper bounds given in (9) defined for each clock branch. The fourth constraint is the total delay insertion bound given in (10). In this formulation, the clock arriving time t_i of each clock branch and the clock period are known. The value of delay insertion Δ_i necessary to fix the timing violations is obtained by solving the formulation.

The LP problem formulation guarantees minimum delay insertion. For instance, if no delay insertion is necessary, Δ_i evaluates to zero. For some circuits, the LP might return infeasibility which means either the timing violations cannot be resolved with the proposed delay insertion upper bounds or the circuit has reconvergent paths which are pathological cases [17] that cannot be solved with clock delay manipulation. Otherwise, the minimal amount of delay to be inserted on each branch is returned as a continuous variable. A more detailed integer linear programming problem (ILP) formulation can also be devised to model the discrete values of delay to be inserted for a higher practical purpose.

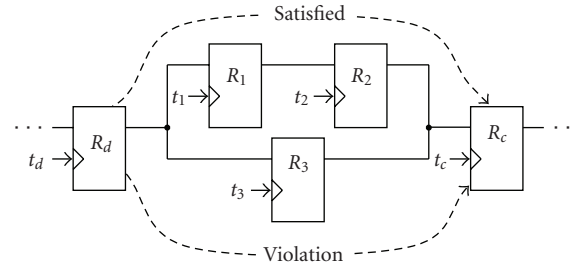


Figure 5: A sample reconvergent path system. Clock delays t_d and t_c satisfy the timing of paths $R_d \to R_1$, $R_1 \to R_2$, $R_2 \to R_c$. However, the timing of one or both of paths $R_d \to R_3$, $R_3 \to R_c$ is violated.

The prescribed topology of a simple reconvergent path is shown in Figure 5. For *some* such systems, timing violations cannot be resolved by manipulating clock delay values as the timing of both branches depends on the clock delays at the divergent R_d and convergent R_c registers. As presented in [17], in such cases, *delay insertion into the logic network* or *reduction of clock frequency* is necessary.

4.2. Formulation 2: Clock Period Optimization. The post-CTS delay insertion methodology proposed for clock period optimization targets the objective of clock period minimization while preserving the original clock tree. The problem definition is

> *Given a pre-computed placement and a synthesized clock tree of an IC (thus, given the clock delay t_i of each branch, propagation time $[D^{ij}_{\mathrm{PMin}}, D^{ij}_{\mathrm{PMax}}]$ of each local data path, internal register delays S, H and D_{CQ} of each register), compute the amount of delay Δ_i to be inserted on each clock tree branch leaf in order to optimize the clock period, considering upper bounds for delay insertion per branch and total delay insertion.*

The mathematical formulation for this problem is derived as a LP problem similar to the formulation in Section 4.1. One difference is that the objective of this LP is clock period minimization so the clock period T is not a known parameter. The resulting LP formulation is presented in Table 3.

The LP formulation guarantees minimum clock period operation with the amount of delay insertion specified by parameters k_1 and k_2. The LP formulation always returns a feasible result, which in worst case is the zero clock skew clock period T_{zs} (i.e., if no improvements are possible through the specified amount of delay insertion). For higher amounts of delay insertion that are allowed, lower minimum clock periods are expected (not guaranteed). In the experiments presented in the next section, the consequences of the level of permitted delay insertion (i.e., k_1 and k_2) on the clock period improvement are analyzed experimentally to observe these trends.

Table 2: Post-CTS Delay insertion results for a suite of ISCAS'89 benchmark circuits.

Circuit info			Clock	Violation data			Insertion data		
Circuit	N	#*paths*	T	$\#p^{\text{vio}}$	$\%p^{\text{vio}}$	$\sum$ vio	$\sum\Delta$	$(\sum\Delta)/N$	Metric
s1196	18	20	20.8	1	5.0%	0.5	10.8	0.6	2.8%
s1423	74	1471	92.2	77	5.2%	1070.7	1531.8	20.7	22.5%
s1488	6	15	32.2	12	80.0%	80.8	61.2	10.2	31.7%
s6669	239	2138	128.6	68	3.2%	1630.2	6811.5	28.5	22.1%
s9234	228	247	75.8	86	34.8%	2204.3	N/A	N/A	N/A
s13207	669	3068	85.6	81	2.6%	1020.5	6279.2	9.4	11.0%
s15850	597	14257	116.0	406	2.8%	5025.4	11581.8	19.4	16.7%
s15850.1	534	10830	81.2	774	7.1%	8580.2	5767.2	10.8	13.3%
s35932	1728	4187	34.2	794	19.0%	6224.5	11232.0	6.5	19.0%
s38417	1636	28082	69.0	2206	7.9%	18811.2	12270.0	7.5	10.9%
s38584	1452	15545	94.2	178	1.1%	1837.1	34412.4	23.7	25.2%
Average					15.3%		8314.6	10.9	15.9%

4.3. Discussion. As discussed earlier, the delay insertion of the proposed work is performed at the post-CTS stage. In order to implement the delay insertion practically, some blocks of reserved white space must be allocated on the chip area. These blocks of white space should be reserved during the floorplanning stage. Depending on the size and the number of cells in the design, designers have to define the utilization area of the chip at the floorplanning stage. If the size requirement is not very strict, a low utilization factor of a chip can be defined in floorplanning so that the delay insertion space at post-CTS stage will be abundant to lead to a better result for fixing the timing violations or optimizing the frequency. If there is not enough space to insert post-CTS delay, a re-design of the layout (floorplanning) might be necessary.

5. Experimental Results

Proposed post-CTS optimization methods are used in experiments on a suite of ISCAS'89 benchmark circuits. A single phase clock signal with a 50% duty cycle is selected for synchronization. The internal register delays (i.e., setup, hold, clock-to-output times) are assumed negligible. The clock network is built experimentally as a zero skew clock tree. The experiments are performed on a 3.2 GHz Intel Xeon processor with a 16 GB RAM. The simplex optimizer of the GNU LP solver GLPK (version 4.31) [18] is used to solve the LP problems. The timing information for ISCAS'89 circuits is generated with a pre-determined algorithm, in which the fanout, size, and type of logic gates are considered. In the floorplanning stage, the utilization factor is chosen to be on the order of 40%.

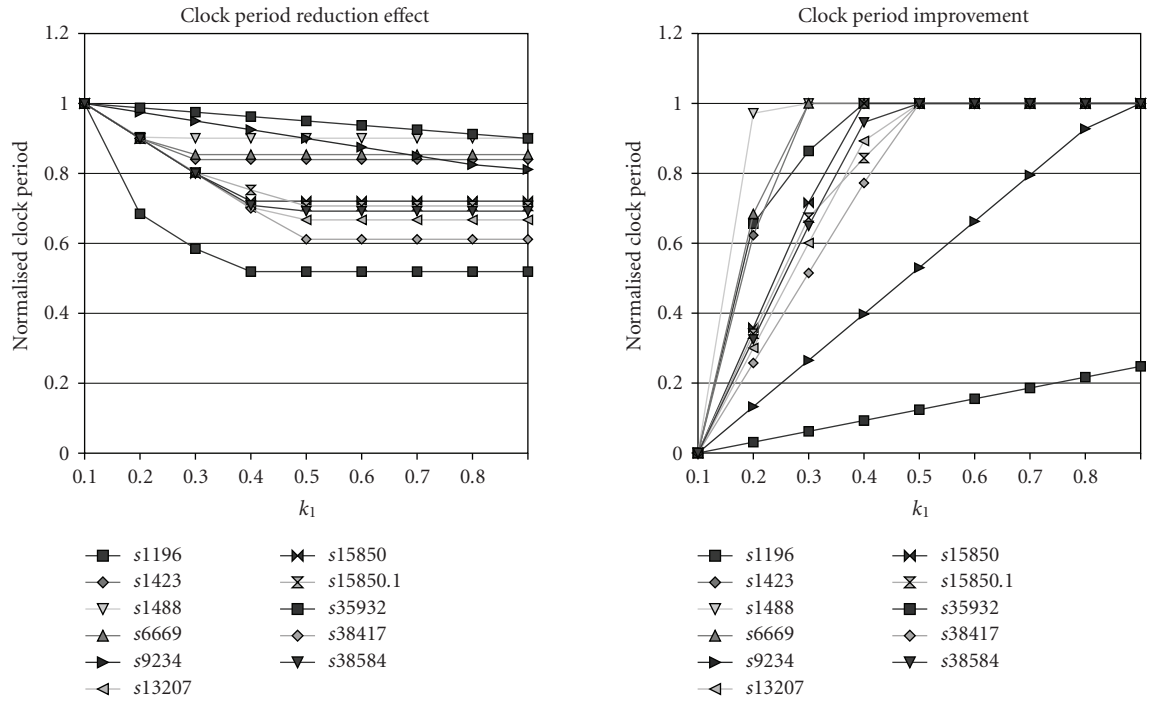
5.1. Experiment 1: Fixing Timing Violations. In order to fix the timing violations with minimum delay insertion, the formulation in Table 1 is applied in the experiments. It is assumed that the clock period is selected as the largest data propagation delay in the circuit (which is typical in an ASIC design, see (5)), the clock delay t_i to each register is arbitrarily selected to be $4T$ with a 10% variation which simulates the variation on the skew. Upper bounds of post-CTS clock delay insertion are set to be $0.8T$ on each branch ($k_1 = 0.8$) with a total delay insertion of $0.4TN$ ($k_2 = 0.4$). In a real application, all experimental assumptions can be easily changed according to automated tool results.

The results are presented on Table 2. In Table 2, circuit information, zero clock skew operation frequency, timing violation data, and post-CTS delay insertion data are presented. The numbers of registers and paths are shown in columns marked N and #*paths*, respectively. The clock period T is selected as the largest data propagation delay in the circuit as derived in (5) to be functional for zero clock skew operation. The number of paths returning timing violation are shown in $\#p^{\text{vio}}$. The percentage of paths with timing violation are shown in $\%p^{\text{vio}}$. The total amount of violation (on all paths) is shown in column $\sum$ vio. Post-CTS inserted delay information is presented in the last three columns, $\sum\Delta$ is the total inserted delay, $(\sum\Delta)/N$ is the average inserted delay per branch (register), and *metric* is a measure of the delay inserted per clock period, that is, $(\sum\Delta)/(NT)$. The metric is used as an arbitrary measure of inserted delay density, as the delay values increase with an increasing clock period regardless of the circuit complexity.

It is observed in Table 2 that post-CTS delay insertion on the clock network is applicable to all circuits except for *s9234* on the selected suite of circuits. Due to the 10% variations in delays—which are randomly generated in experimentation—timing violations occur on 15.3% of the paths but as many as 80% of all the paths (*s1488*) and as low as one (1) path (*s1196*) for a given circuit. The upper bounds of clock delay insertion, set by $k_1 = 0.8$ and $k_2 = 0.4$, enable us to fix the timing violations in most of the circuits by minimal delay insertion. The selected metric for delay insertion density

$$\text{Metric:} \frac{(\sum\Delta)}{(NT)} \tag{11}$$

has an average of 15.9%, which is reasonably small for a practical implementation.



(a) Normalised clock period reduction with increasing k_1 and k_2

(b) Clock period improvement with increasing k_1 and k_2

Figure 6: Clock period optimization results.

Table 3: LP model for post-CTS delay insertion method.

Minimize inserted buffer delay	
min	T
s.t.	$t_i + \Delta_i + D^i_{\text{CQ}} + D^{if}_{\text{PMax}} \le t_f + \Delta_f + T - S^j$
	$t_i + \Delta_i + D^i_{\text{CQ}} + D^{if}_{\text{PMin}} \ge t_f + \Delta_f + H^f$
	$\Delta_i \le k_1 T_{zs}$
	$\sum_{i=0}^{N-1} \Delta_i \le k_2 T_{zs} N$

The timing violations for benchmark circuit *s9234* cannot be resolved with post-CTS delay insertion because of *reconvergent paths* [17]. Although not observed in our experiments, the maximum delay insertion bounds on each clock branch $\Delta_i \le k_1 T$ and the total delay insertion constraint $\sum \Delta < k_2 TN$ can also be limiting. For such circuits, designers can choose to follow typical procedures of performing iterative runs of placement, routing (or synthesis) to satisfy the specified timing budget. When such practices are costly, frequency specification can be relaxed to have the IC operate at a lower speed.

5.2. Experiment 2: Clock Period Optimization. In order to optimize the clock period, the formulation in Table 3 is applied in the experiments. In these experiments, upper bound of delay insertion on each branch of the clock tree is set to $k_1 \times T_{zs}$ and the upper bound of delay insertion on the clock tree is set to $k_2 \times T_{zs} \times N$, where N is number of leaves in the clock tree and T_{zs} is the minimum clock period at zero clock skew. In the experiments, k_2 is set equal to one half of k_1 ($k_2 = 0.5k_1$), which suggests that the amount of delay insertion allowed on each tree branch is $k_1 \times T_{zs}$, while the total amount of delay insertion allowed on the tree is $0.5k_1 \times T_{zs} \times N$. As described in Section 4.2, such a correlation between k_1 and k_2 is used to have both constraints be binding as opposed to permitting excessive delay insertion for impractical clock period improvements. Additionally, this experimental setup enables a direct comparison with the previous work in [2] by providing the methodologies with identical total delay insertion resources. The comparison of results with the previous work in [2] is presented in Table 4. A "single"-domain application of the multidomain clock skew scheduling algorithm proposed in [2] is replicated in experimentation with skew scheduling ranges of 5% and 10% (0% case in [2] is the obvious zero clock skew case and needs not to be considered). In Table 4, the clock periods computed with both methodologies are presented as well as the *progress* of the improvement in clock period minimization. For instance, an improvement progress of 0% would indicate a zero clock skew operation while an improvement progress of 100% would indicate a design that is scheduled to operate at the minimum possible clock period with unlimited insertion. It is observed for both delay insertion bounds of 5% and 10% that the proposed post-CTS methodology consistently outperforms the multidomain clock skew scheduling methodology. On average, the proposed methodology is 2X and 1.6X better than [2] for skew scheduling ranges of 5% and 10%, respectively. As described in Section 3, the superiority of the proposed

Table 4: Clock period optimization with respect to the maximum possible improvement.

Circuit info	Clock period with total delay buffering 5%$T_{zs}N$				Clock period with total delay buffering 10%$T_{zs}N$			
Circuit	[2]	Improvement	Proposed	Improvement	[2]	Improvement	Proposed	Improvement
s1196	17.36	34%	14.24	66%	16.32	45%	12.16	86%
s1423	87.59	31%	82.98	62%	82.98	62%	77.40	100%
s1488	30.59	50%	29.09	97%	29.09	97%	29.00	100%
s6669	122.17	34%	115.74	68%	115.74	68%	109.80	100%
s9234	299.41	7%	295.62	13%	295.62	13%	288.04	27%
s13207	81.32	15%	77.04	30%	77.04	30%	68.48	60%
s15850	110.20	18%	104.4	36%	104.40	36%	92.8	72%
s15850.1	77.14	17%	73.08	34%	73.08	34%	65.18	67%
s35932	33.99	2%	33.77	3%	33.77	3%	33.35	6%
s38417	65.55	13%	62.10	26%	62.10	26%	55.20	51%
s38584	89.49	16%	84.78	32%	84.78	32%	75.36	65%
Average		22%		43%		41%		67%

Table 5: Post-CTS limited delay insertion clock periods for a suite of ISCAS'89 benchmark circuits.

Circuit info	Clock period									
Circuit	$k_1 = 0$	$k_1 = 0.1$	$k_1 = 0.2$	$k_1 = 0.3$	$k_1 = 0.4$	$k_1 = 0.5$	$k_1 = 0.6$	$k_1 = 0.7$	$k_1 = 0.8$	CSS ($k_1 = \infty$)
s1196	20.80	14.24	12.16	10.80	10.80	10.80	10.80	10.80	10.80	10.80
s1423	92.20	82.98	77.40	77.40	77.40	77.40	77.40	77.40	77.40	77.40
s1488	32.20	29.09	29.00	29.00	29.00	29.00	29.00	29.00	29.00	29.00
s6669	128.60	115.74	109.80	109.80	109.80	109.80	109.80	109.80	109.80	109.80
s9234	303.20	295.62	288.04	280.46	272.88	265.30	257.72	250.14	246.00	246.00
s13207	85.60	77.04	68.48	60.16	57.10	57.10	57.10	57.10	57.10	57.10
s15850	116.00	104.40	92.80	83.60	83.60	83.60	83.60	83.60	83.60	83.60
s15850.1	81.20	73.08	65.18	61.12	57.40	57.40	57.40	57.40	57.40	57.40
s35932	34.20	33.77	33.35	32.92	32.49	32.06	31.64	31.21	30.78	20.4
s38417	69.00	62.10	55.20	48.30	42.20	42.20	42.20	42.20	42.20	42.20
s38584	94.20	84.78	75.36	66.77	65.20	65.20	65.20	65.20	65.20	65.20

methodology is due to the flexibility of bounds on each clock branch and global monitoring of overall delay insertion.

Next, the parameters k_1 and k_2 are gradually increased to observe the change in clock period optimization through various levels of delay insertion.

In Table 5, the clock period improvements for varying delay insertion bounds between $k_1 = 0$ and $k_1 = 0.8$ are presented. The last column in Table 5 presents the unbounded clock skew scheduling result, that is, an upper bound of $k_1 = k_2 = \infty$. It is confirmed with experimentation that with increasing k_1 and k_2, the clock period is monotonously improved. An important observation is the delay insertion bounds at which significant progress is obtained in clock period minimization. For most of the circuits, the majority of the clock period improvement are achieved with delay insertion with an upper bound of 10% to 20% times T_{zs} on each branch. As demonstrated here, delay insertion budgets for clock period minimization can be devised more accurately so as to not waste design resources for relatively smaller improvements to be achieved for additional delay insertion over a certain bound.

In Figure 6(a), the minimum clock period with varying bounds of delay insertion is normalized with respect to the zero clock skew minimum clock period T_{zs}. In Figure 6(b), the clock period improvements with varying levels of insertion are presented as a percentage of maximum possible clock period improvement. In Figure 6(b), each curve starts from $k_1 = k_2 = 0$, which implies no delay insertion, thus no improvement in clock period minimization. The value "1" in the figure implies the maximum level of improvement (e.g., 100%) in clock period optimization is achieved. It is observed that nine (9) of the eleven (11) circuits can be optimized to more than 90% of the optimal solution with the post-CTS method using only a limit amount of delay insertion corresponding to $k_1 = 0.3$. Numerically, with k_1 set to 0.2 and k_2 set to 0.1, nine (9) out of eleven (11) circuits exhibit more than 20% of clock period improvement and seven (7) of them have improvements of over 50%. The average improvement in the clock period minimization is 67% for the selected suite of circuits, demonstrating the high level of improvement with limited delay insertion.

6. Conclusions

The post-CTS clock delay insertion method has the motivation of observing the limited amount of delay insertion space on an integrated circuit and utilizing this area more efficiently by simultaneously limiting the delay insertion per branch and the clock tree. The proposed method is analyzed for the objectives of fixing timing violations and clock period optimization. The proposed method has the following advantages.

(i) The proposed method is performed post-CTS, which requires lower efforts to fix timing violations after verification.

(ii) The proposed method starts off with the CTS results and only permits minimal delay insertion, which keeps the clock delays easy to realize.

A first set of experiments is performed to observe the advantages of limited delay insertion for circuits with timing violations. In experimentation with the generated ISCAS'89 clock networks, it is found that a 10% variation in the delay values of the clock network can result in 15.3% of the timing paths to fail the timing requirements. By applying the proposed method, timing violations are successfully resolved in ten (10) of the eleven (11) experimented circuits. A second set of experiments is performed to observe the advantages of limited insertion for circuits where no timing violations exist. By applying the clock period optimization method, the clock period can be improved by an average of 43% with a very limited amount of delay insertion. In practice, post-CTS delay insertion method can be used by designers to find quick solutions to timing violation or clock period optimization problems without having to go through lengthy synthesis-placement-routing iterations.

References

[1] I. S. Kourtev, B. Taskin, and E. G. Friedman, *Timing Optimization through Clock Skew Scheduling*, Springer, New York, NY, USA, 2009.

[2] K. Ravindran, A. Kuehlmann, and E. Sentovich, "Multi-domain clock skew scheduling," in *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design (ICCAD '03)*, pp. 801–808, San Jose, Calif, USA, November 2003.

[3] Q. K. Wu, *High-Speed Clock Network Design*, Kluwer Academic Publishers, Dordrecht, The Netherlands, 2003.

[4] E. G. Friedman, *Clock Distribution Networks in VLSI Circuits and Systems*, IEEE Press, New York, NY, USA, 1995.

[5] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to Algorithms*, MIT Press, Cambridge, Mass, USA, 2nd edition, 2001.

[6] M. A. B. Jackson, A. Srinivasan, and E. S. Kuh, "Clock routing for high-performance ICs," in *Proceedings of the ACM/IEEE Design Automation Conference (DAC '90)*, pp. 573–579, Orlando, Fla, USA, June 1990.

[7] R.-S. Tsay, "An exact zero-skew clock routing algorithm," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 12, no. 2, pp. 242–249, 1993.

[8] N.-C. Chou and C.-K. Cheng, "On general zero-skew clock net construction," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 3, no. 1, pp. 141–146, 1995.

[9] N. Ito, H. Sugiyama, and T. Konno, "ChipPRISM: clock routing and timing analysis for high-performance CMOS VLSI chips," *Fujitsu Scientific and Technical Journal*, vol. 31, no. 2, pp. 180–187, 1995.

[10] N. Gaddis and J. Lotz, "A 64-b quad-issue CMOS RISC microprocessor," *IEEE Journal of Solid-State Circuits*, vol. 31, no. 11, pp. 1697–1702, 1996.

[11] S. Rusu and G. Singer, "The first IA-64 microprocessor," *IEEE Journal of Solid-State Circuits*, vol. 35, no. 11, pp. 1539–1544, 2000.

[12] W. J. Bowhill, S. L. Bell, B. J. Benschneider, et al., "Circuit implementation of a 300-MHz 64-bit second-generation CMOS alpha CPU," *Digital Technical Journal*, vol. 7, no. 1, pp. 100–118, 1995.

[13] W.-K. Chen, Ed., *The VLSI Handbook*, CRC Press, Boca Raton, Fla, USA, 1st edition, 1999.

[14] S. R. Nassif, "Modeling and analysis of manufacturing variations," in *Proceedings of the IEEE Custom Integrated Circuits Conference (CICC '01)*, pp. 223–228, San Diego, Calif, USA, May 2001.

[15] A. B. Kahng, "A roadmap and vision for physical design," in *Proceedings of the IEEE International Symposium on Physical Design (ISPD '02)*, pp. 112–117, Del Mar, Calif, USA, April 2002.

[16] A. V. Mule, E. N. Glytsis, T. K. Gaylord, and J. D. Meindl, "Electrical and optical clock distribution networks for gigascale microprocessors," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 10, no. 5, pp. 582–594, 2002.

[17] B. Taskin and I. S. Kourtev, "Delay insertion method in clock skew scheduling," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 25, no. 4, pp. 651–663, 2006.

[18] Free Software Foundation (FSF), "GLPK (GNU Linear Programming Kit)," 2008, http://www.gnu.org/software/glpk/glpk.html.

Hindawi Publishing Corporation
VLSI Design
Volume 2010, Article ID 946710, 11 pages
doi:10.1155/2010/946710

Research Article

A Low-Power Digitally Controlled Oscillator for All Digital Phase-Locked Loops

Jun Zhao and Yong-Bin Kim

Department of Electrical and Computer Engineering, Northeastern University, Boston, MA 02115, USA

Correspondence should be addressed to Yong-Bin Kim, ybk@ece.neu.edu

Received 31 May 2009; Accepted 20 October 2009

Academic Editor: Gregory D. Peterson

A low-power and low-jitter 12-bit CMOS digitally controlled oscillator (DCO) design is presented. The Low-Power CMOS DCO is designed based on the ring oscillator implemented with Schmitt trigger inverters. The proposed DCO circuit uses control codes of thermometer type to reduce jitters. Performance of the DCO is verified through a novel All Digital Phase-Locked Loop (ADPLL) designed with a unique lock-in process by employing a time-to-digital converter, where both the frequency of the reference clock and the delay between DCO_output and DCO_clock is measured. A carefully designed reset process reduces the phase acquisition process to two cycles. The ADPLL was implemented using the 32 nm Predictive Technology Model (PTM) at 0.9 V supply voltage, and the simulation results show that the proposed ADPLL achieves 10 and 2 reference cycles of frequency and phase acquisitions, respectively, at 700 MHz with less than 67 ps peak-to-peak jitter. The DCO consumes 2.2 mW at 650 MHz with 0.9 V power supply.

1. Introduction

Phase-locked loops are widely used in many communication systems for clock and data recovery or frequency synthesis [1–5]. Cellular phones, computers, televisions, radios, and motor speed controllers are just a few examples that rely on PLLs for proper operation. With such a broad range of applications, PLLs have been extensively studied in literature.

The conventional PLLs are often designed using analog approaches. However, analog PLLs have to overcome the digital switch noise coupled with power through power supply as well as substrate-induced noise. In addition, the analog PLL is very sensitive to process parameters and must be redesigned if the process is changed or migrates to next generation process. Although many approaches have been developed to improve the jitter performance, it often results in long lock-in time and increasing design complexity. With the increasing performance and decreasing cost of digital VLSI design technology, all digital phase-locked loops have become more attractive. Although ADPLL will not have the same performance as its analog counterpart, it provides a faster lock-in time and better testability, stability, and portability over difference process [6, 7].

The controlled oscillator is a key component in PLL, which is a replacement of the conventional voltage or current controlled oscillator in the fully digital PLLs. They are more flexible and usually more robust than the conventional VCO. Furthermore, the design compromise for the frequency gain in voltage or current controlled oscillator is not necessary in DCOs because the immunity of their control input is very high. There are two main techniques for the DCO design as shown in Figure 1. One technique changes the driving strength dynamically using the fixed capacitance loading [8, 9] while the other uses shunt capacitor technique to tune the capacitance loading [10]. Although both of the approaches have a good linear frequency response and a reasonable frequency operating range, the power dissipation has not been taken into consideration. Moreover, for the DCO design, there is a tradeoff between the operating range and the maximum frequency that DCO can achieve. As a result, the increase of the operating range by adding more capacitance loading will result in a lower maximum frequency and higher power consumption. Since power

consumption is of extreme concern for portable battery-charged computing systems, the reduction of the power consumption has become a major concern in modern electronic systems.

This paper proposes a novel DCO circuit with significantly reduced power consumption using binary controlled pass transistors and Schmitt trigger inverters. The functionality and performance are verified through a novel ADPLL that uses the proposed DCO. Usually the ADPLL structure based on the second order negative feedback system has a faster lock-in time with a limited lock-in range [11, 12]. One the other hand, by separating the locking process into frequency and phase acquisition, a wide lock-in range is available [13, 14]. However, it takes more time due to the blind "ahead" or "behind" comparison as well as the extra phase acquisition process. In this paper, the new ADPLL also uses a separated frequency and phase lock-in process. Instead of "ahead" or "behind" comparison, a time-to-digital converter is used to measure the frequency difference accurately, which greatly reduces the lock-in time. The phase acquisition only takes two reference clocks. In the first cycle, the DCO is reset by the reference clock considering the delay between DCO_output and DCO_clock. In the second cycle, the DCO frequency changes back to the reference clock by updating the control bits. The ADPLL with the proposed DCO was implemented using a 0.9 V 32 nm practical transistor model.

2. DCO Principle and Design

2.1. DCO Principle. DCO should generate an oscillation period of T_{DCO}, which is a function of digital input word D and given by

$$T_{\mathrm{DCO}} = f\left(d_{n-1}2^{n-1} + d_{n-2}2^{n-2} + \cdots + d_1 2^1 + d_0 2^0\right). \quad (1)$$

Typically, the DCO transfer function is defined such that the period of oscillation T_{DCO} is linearly proportional to D with an offset. Therefore, the oscillation period is rewritten as

$$T_{\mathrm{DCO}} = T_{\mathrm{offset}} - D \cdot T_{\mathrm{step}}, \quad D: \text{ Digital Control Bits}, \quad (2)$$

where T_{offset} is a constant offset period and T_{step} is the period of the quantization step. For the conventional driving strength-controlled DCO shown in Figure 2, the constant delay of each cell is calculated as follows:

$$T_{\mathrm{constant}} = R_1(C_1 + C_2) + R_2C_2, \quad (3)$$

$$R_{1,2} \propto 1/W_{1,2}, \quad (4)$$

where R_1 and $R_2$are the equivalent resistances of M1and $M1'$ and C1 and C2 are the total capacitances at the drain of M1 and $M1'$, respectively, which mainly consist of drain to body and source to body capacitances. Assuming that they have the same driving strength, the delay tuning range of this standard cell is obtained as follows:

$$\frac{T_{\mathrm{tune}}}{2} = (C_1 + C_2)\left(R_1 // \frac{\Delta R}{d_0} // \frac{\Delta R}{d_1 2} // \cdots // \frac{\Delta R}{d_{n-1}2^{n-1}}\right) - (C_1 + C_2)R_1 \quad (5)$$

$$= \frac{R_1(C_1 + C_2)}{1 + (D \cdot \Delta W)/W_1} - (C_1 + C_2)R_1 \quad (6)$$

$$\approx R_1(C_1 + C_2)\frac{D \cdot \Delta W}{W_1}{}_1, \quad \left(\text{Only if } \frac{D \cdot \Delta W}{W_1} \ll 1\right). \quad (7)$$

In order to have a good linear tuning range, the width of the transistor M1 has to be increased as illustrated in (7). Consequently the equivalent resistance R1 will decrease, resulting in a smaller delay tuning range. One way to increase the tuning range while keeping the linear response is to increase the capacitance loading. However, this will minimize the maximum frequency that the DCO can accomplish and the power consumption will also be increased.

2.2. Proposed DCO Design. The proposed DCO employs a new approach to increase the delay tuning range using digitally controlled pass transistor arrays and Schmitt trigger-based inverters [15]. The Schmitt trigger-based inverter has a higher VM+ (low-to-high switching threshold) and lower VM− (high-to-low switching threshold) compared to the conventional inverters as shown in Figure 3. As a result, the proposed DCO circuit provides the same tuning range with a smaller capacitance loading, which is beneficial for power consumption reduction. Moreover, in the conventional DCO circuit, the slope of the input signal to each stage decreases gradually due to the large delay between each stage. This results in not only a nonideal rail-to-rail switch but also a poor power performance. The steep slope of the output signal from the Schmitt trigger-based inverter minimizes this problem to a certain extend.

The circuit diagrams of the conventional DCO and proposed DCO are shown in Figure 4. The conventional DCO consists of two identical binary controlled coarse cells as well as a similar fine cell with smaller tuning range. The proposed DCO also consists of two coarse cells and a fine cell. The coarse cells have tuning codes of 2 bits with PMOS array or NMOS array in form of thermometer code, which could provide a better duty cycle performance and linearity. The fine cell has tuning codes of 6 bits by only NMOS array in the form of thermometer code as shown in Figure 4(c). The thermometer code minimizes the jitters. Since they are grouped per 2 bits, the circuit to convert binary code to thermometer code is also minimized.

2.3. Improved Structure with Larger Operating Range. The binary controlled DCO structure has a limited linear operating range as discussed above. In this paper, three-stage constant delay chains and a 4 : 1 Mux are used to increase the operating range, and the three-stage constant delay is tuned by the fixed code such that each stage provides an accurate

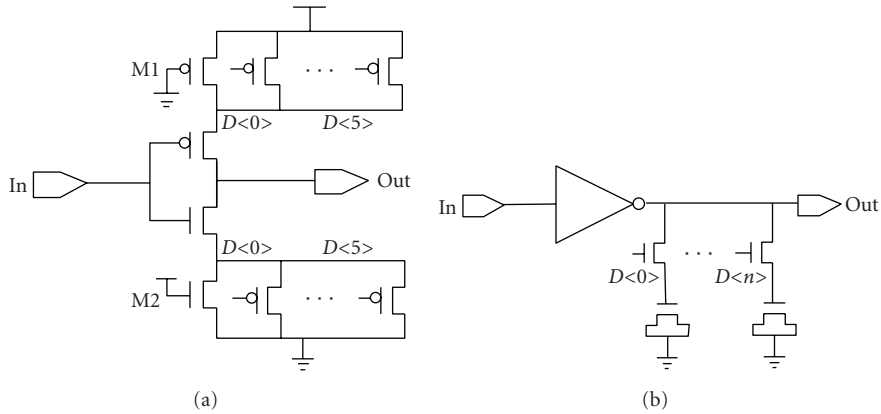


Figure 1: Standard cell of digitally controlled oscillator. (a) Driving strength controlled. (b) Shunt capacitance controlled.

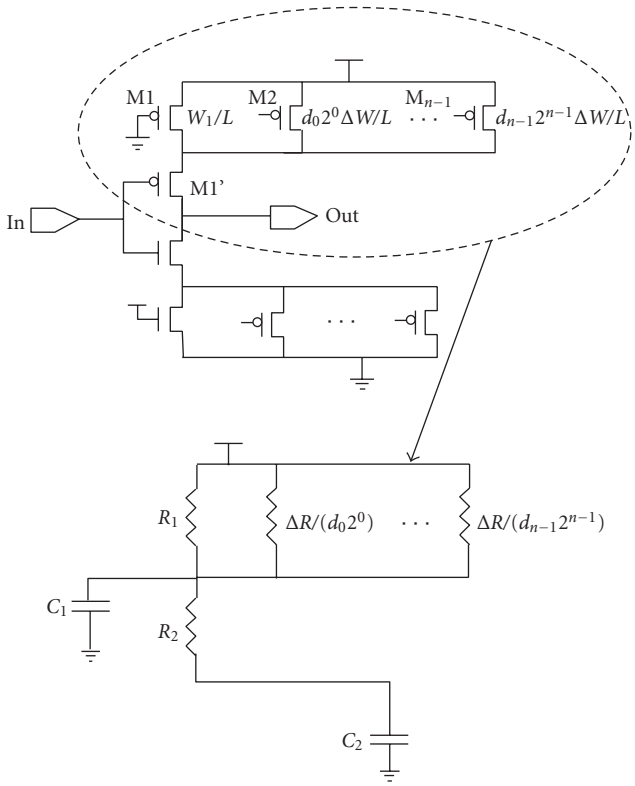


Figure 2: Equivalent circuit for the calculation of constant delay and delay tuning range.

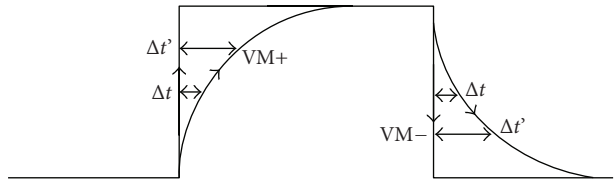


Figure 3: Delay comparison of Schmitt inverter and conventional inverter.

Table 1: Impact of each control bit on the DCO period

Control Bits	The Conventional DCO		The Proposed DCO	
	Period (ns)	Delta (ps)	Period (ns)	Delta (ps)
111111	1.8022	257.7	1.7918	270.6
100000	1.544.5	134.6	1.5212	139
010000	1.4099	69.8	1.3822	70.8
001000	1.3401	34.4	1.3114	36.8
000100	1.3057	19.8	1.2746	19.1
000010	1.2859	9.1	1.2555	9.4
000001	1.2768	9.2	1.2461	8.5
000000	1.2676	—	1.2376	—

delay as shown in Figure 5. As a result, the operating range can be four times larger compared to the original design.

2.4. Comparison between the Two DCO Structures. The proposed DCO and the conventional DCO are simulated and compared using 32 nm CMOS PTM (Predictive Technology Model) with a supply voltage of 0.9 Volts. The choice of 12 bits is a compromise between the DCO resolution, operating range and circuit complexity.

Table 1 shows the impact of each control bit on the period of the two DCO structures. Both structures have the same linear tuning range. Since the two DCO structure-have the same operating ranges, it is more reasonable for us to compare their power consumption.

Compared to the conventional DCO, the proposed DCO saves approximately 40% power consumption as shown in Figure 6. As discussed above, this reduction is due to the comparatively smaller capacitance loading for the Schmitt trigger-based inverter than the conventional inverter at the same operating frequency. The proposed DCO is significantly more power efficient than the conventional DCO. However, this DCO design has a limited operating frequency range, which is improved in this paper by employing the fixed delay blocks shown in Figure 5.

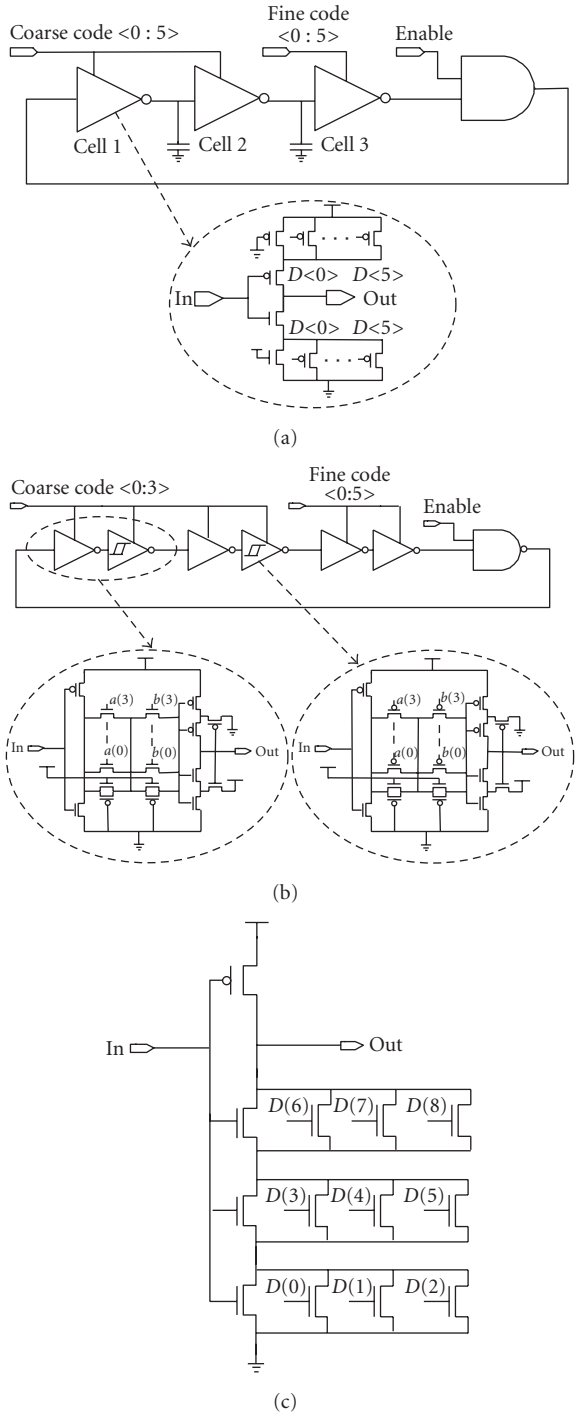


Figure 4: Digitally controlled oscillator. (a) conventional DCO structure). (b) Proposed DCO structure. (c) Fine delay cell.

2.5. Simulation Result of the Proposed DCO. The proposed DCO structure with increased operating range is designed and simulated using 32 nm PTM model. Figure 7 shows operating frequency ranges of the coarse and fine tuning frequency of the novel DCO. The curves have good monotonousness, which is a key factor in PLL performance. The frequency ranges are about 570 MHz ~ 800 MHz at

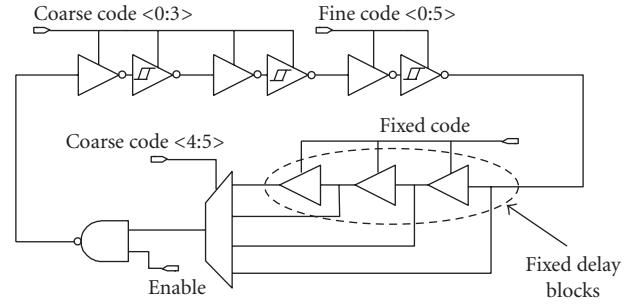


Figure 5: The proposed DCO structure with increased operating range.

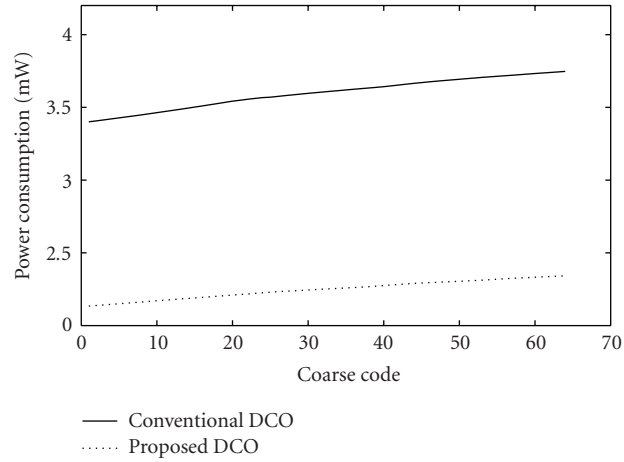


Figure 6: Power consumption of the conventional and the proposed DCO structures.

Table 2: Impact of each control bit on the DCO Period.

Items	Coarse Delay	Fine Delay
Resolution	6 bit	6 bit
Max. DCO Gain	10 ps	1 ps
Avg. DCO Gain	7 ps	0.7 ps
Operation Range	570 MHz~800 MHz	
Power Consumption	2.2 mW @ 650 MHz	

the condition of 0.9 V supply voltage at 25°C and the delay range of the fine delay chain is about 45 ps. The characteristics of the proposed DCO are summarized at Table 2.

The operational frequency response to the process, temperature, and voltage variation is shown in Figure 8. The curves show the normalized data with respecte to the center frequency. Figure 8 shows that the relative delay per code is almost same regardless of the process, temperature, and, voltage variations. In other words, the proposed DCO design is very robust to PVT variations.

Table 3 shows the measurement results to compare with a few recent state-of-the-art DCO designs [6, 10, 16, 17]. The proposed DCO achieves the finest LSB resolution and

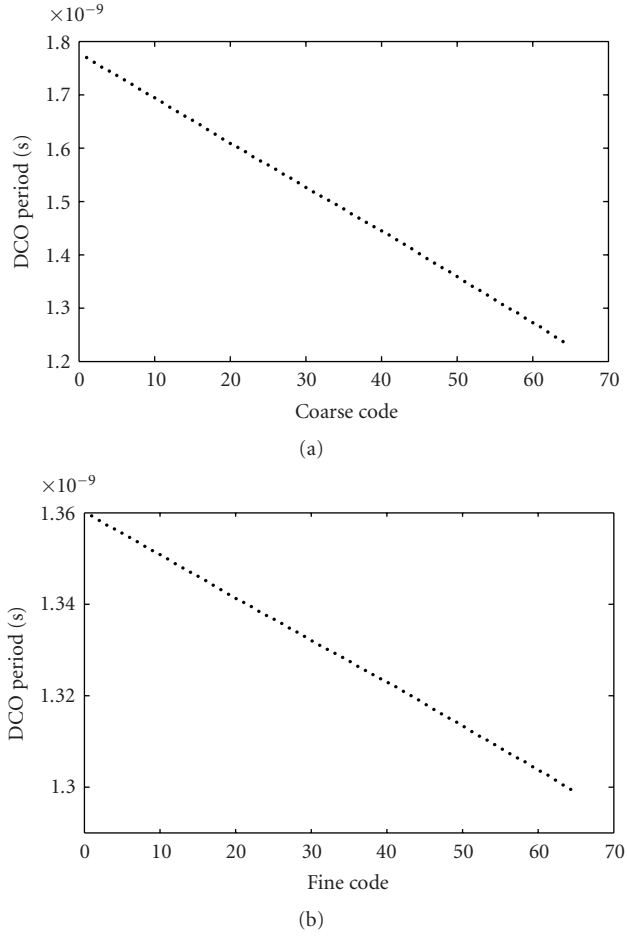


Figure 7: Operating range of the proposed DCO: (a) coarse loop and (b) fine loop.

the highest operating frequency. In addition, the proposed DCO consumes less power than others.

3. Performance Verification of the Proposed DCO

In this section, performance of the proposed DCO is verified through a novel ADPLL with the proposed DCO circuit. The proposed ADPLL is designed with a unique lock-in process based on the good monotonousness of the DCO.

3.1. ADPLL Architecture Overview. The block structure of the new ADPLL is shown in Figure 9. The control word corresponding to the period of the reference clock T_{ref} is stored in register1. In register 2, the control word corresponds to a new period of $T_{\text{ref}} - T_{\text{delay}}$, which is the period of reference clock subtracted by the delay between DCO_output and DCO_clock. Unlike the conventional ADPLL designs, the clock signals to all the logical blocks are generated from the DCO_output.

Phase lock begins with frequency acquisition. In this mode, a time-to-digital converter measures the time difference between the reference clock and the DCO_clock. As shown in Figure 10(a), it converts time difference into the digital word T_1 and T_2, which are the time difference between DCO_clock's rising edge and the reference clock's rising and falling edges, respectively. As a result, the frequency (period) difference can be defined as follows:

$$\Delta T = (T_1 - T_1') + \frac{(N-2)T}{2}, \tag{8}$$

$$T = 2(T_2 - T_1). \tag{9}$$

N represents the reference clock's low-to-high or high-to-low transition number during one DCO_clock period from the 4-bit counter. T_1' is the stored value of T_1 in the previous DCO period.

Compared to other frequency acquisition approaches, the DCO does not have to be reset at the beginning of every reference cycle for the initial phase alignment, which reduces the design complexity. Moreover, the frequency acquisition process can be reduced to less than ten cycles if the DCO has a good linearity performance. However, as shown in Figure 10(b), due to the nonzero setup time, the last transition of the reference clock may be ignored if the time difference T_1' is smaller than the register's setup time, which will results in an incorrect transition number N. The improved integer counter, which is designed to address this problem, will be discussed in the circuit design part.

The ADPLL in this paper starts with frequency and phase acquisition followed by maintenance mode. Once the frequency and phase are acquired using the coarse code, the acquired frequency and phase are maintained by updating the fine codes to correct the phase and frequency drift due to noises.

During the frequency acquisition mode, the coarse control bits are generated by the algorithm (arithmetic) blocks in Figure 9 and applied to the DCO. Since the DCO has a good linearity, this acquisition process takes fewer reference cycles compared to the previous blind fast or slow comparison. When the frequency is locked, the control bits are stored in the coarse bit register and the lock-in process is switched from the frequency acquisition to the phase acquisition process by the state machine.

In the phase acquisition, the DCO_clock edge will be aligned to the reference clock edge. In reality, there are several stages of logic separating DCO_output and DCO_clock such as the duty-cycle corrector shown in Figure 9. As a result, the DCO_clock edge cannot be aligned to the reference clock by a simple reset process as shown in Figure 11. The delay time T_{Delay} results from the logic blocks between the DCO_output and the DCO_clock.

A phase acquisition process is required to get the phase aligned, which is usually done by comparing the phase position of the two signals. The adjustment on the control word is made based on the "behind" or "ahead" signal until there is a polarity change. However, such kind of acquisition process takes many cycles, which results in a slow lock-in process.

A novel reset process is presented in this paper, which is able to reduce the phase lock process to two cycles as shown in Figure 12. In the first cycle, the DCO is still

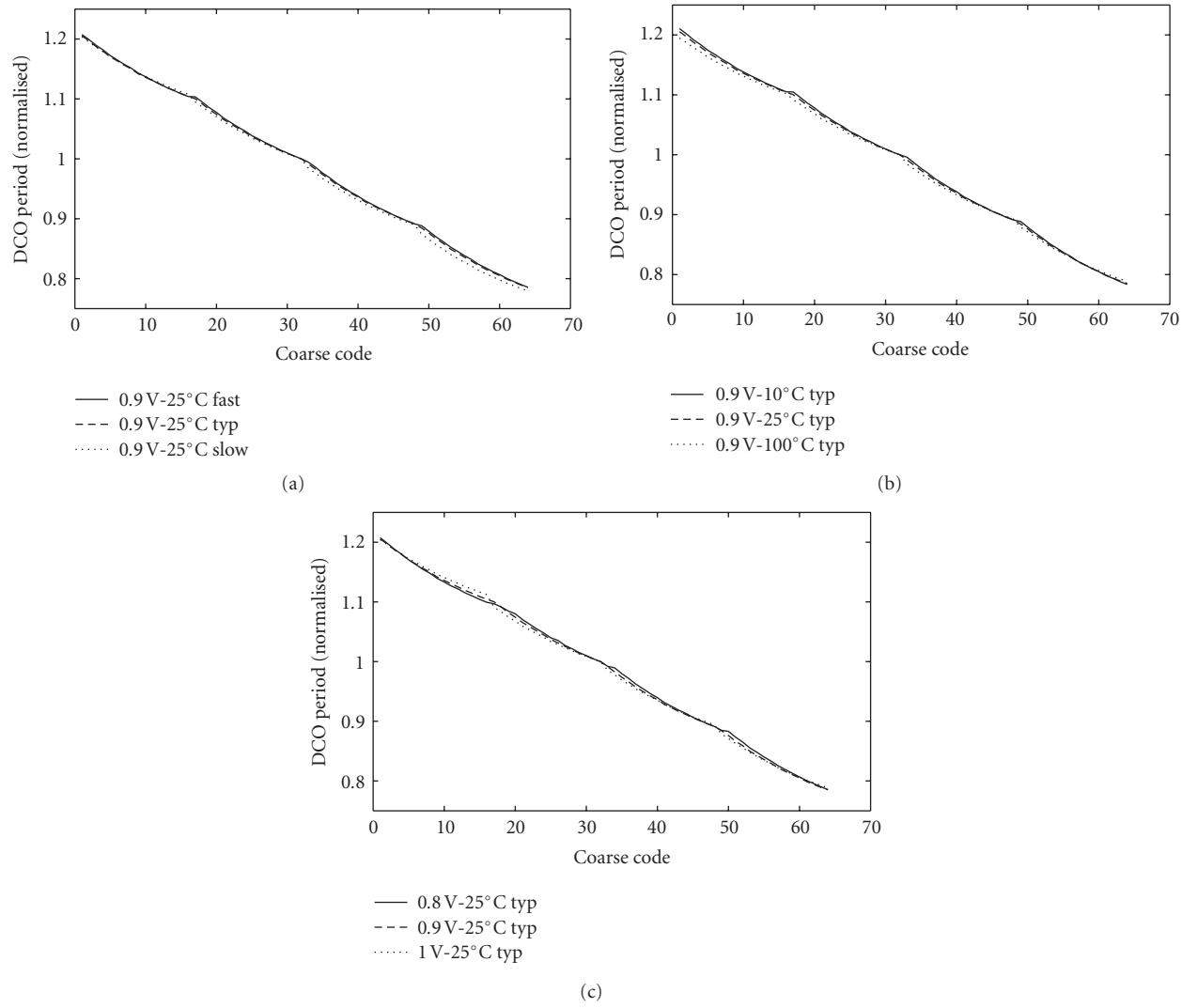


Figure 8: Delay characteristics of the coarse loop for the Process, Voltage, and Temperature variations. (a) Process variation. (b) Temperature Variation. (c) Voltage Variation.

Table 3: Comparis on with existing DCOs.

Function	Proposed	Items TCAS-II′ 05 [16]	ISQED′ 02 [10]	ISSCC′ 03 [17]	JSSC′ 03 [6]
Process	32 nm @ 0.9 V	0.35 um @ 3.3 V	0.13 um @ 1.65 V	0.6 um @ 5 V	0.35 um @ 3.3 V
DCO control world length	12 bits	15 bits	8 bits	10 bits	12 bits
LSB Resolution	1 ps	1.55 ps	40 ps	10 ps	5 ps
DCO output frequency	570 ~ 800 (MHz)	18 ~ 214 (MHz)	150 (MHz)	10 ~ 12.5 (MHz)	45 ~ 450 (MHz)
Power Consumption	2.2 mW @ 650 MHz	18 mW @ 200 MHz	1 mW @ 150 MHz	164 mW @ 100 MHz	100 mW @ 450 MHz

reset by reference clock with control word corresponding to $T_{\text{ref}} - T_{\text{Delay}}$. The control word is found in a similar way as mentioned in the frequency acquisition:

$$\Delta T = (T_1 - T_1') + \frac{(N-2)T}{2} + T_{\text{Delay}}. \tag{10}$$

Without the delay, the second rising edge will lead the reference clock by T_{Delay} such as the DCO_output. However, as for the DCO_clock signal, this can be compensated by the existing delay T_{Delay} and the second rising edge will be aligned to the reference clock. In the second cycle, the control word in the register1 will be reloaded and DCO frequency will be the same as reference clock again.

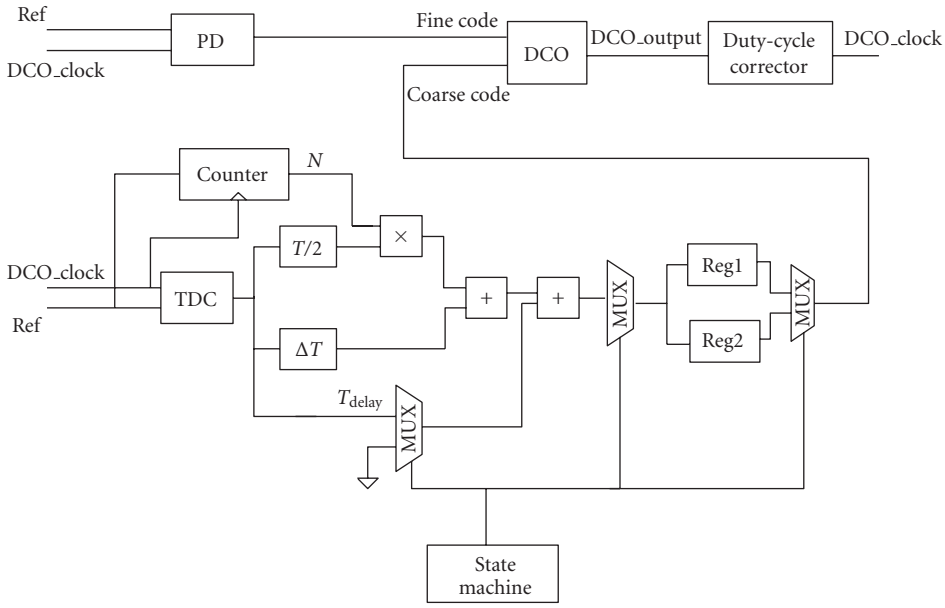


Figure 9: Block diagram of an ADPLL using the proposed DCO.

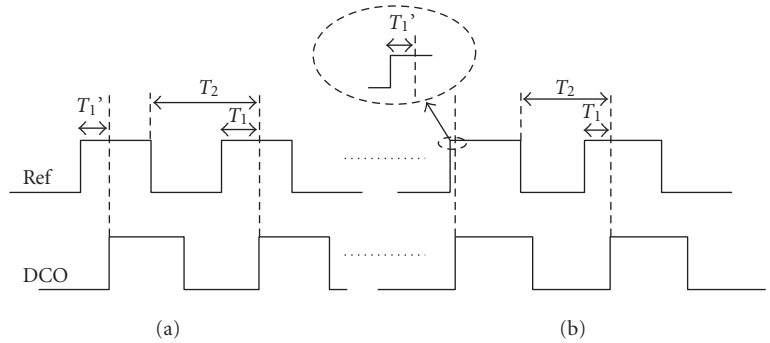


Figure 10: Principle of frequency acquisition.

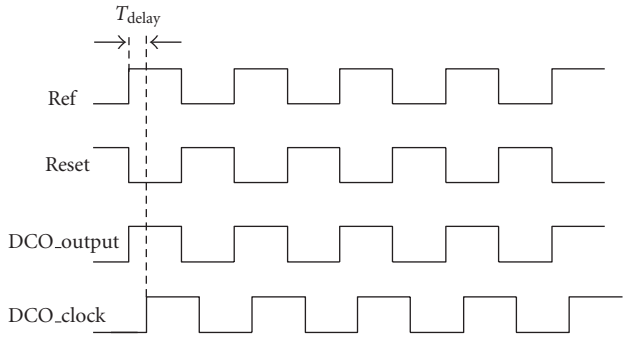


Figure 11: Failure in phase alignment due to logic blocks between DCO_clock and DCO_output.

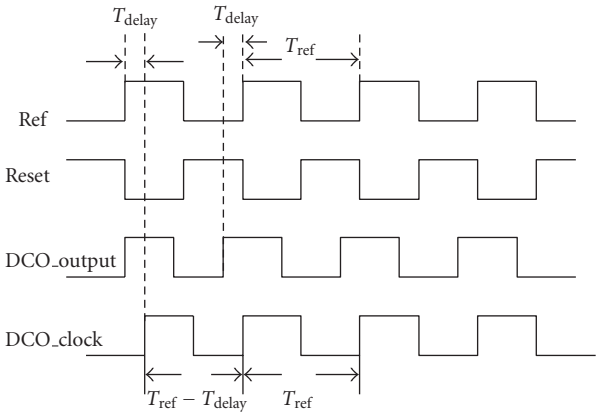


Figure 12: A new phase acquisition process with two lock-in cycles.

After the phase acquisition, a maintenance mode is applied to preserve the phase alignment of the DCO_clock relative to reference clock. The phase detector generates "ahead" or "behind" signal based on the rising edges of the reference clock and DCO_clock. The ADPLL increments or decrements the control word every cycle. The magnitude of the change, which is the value held in the phase-gain

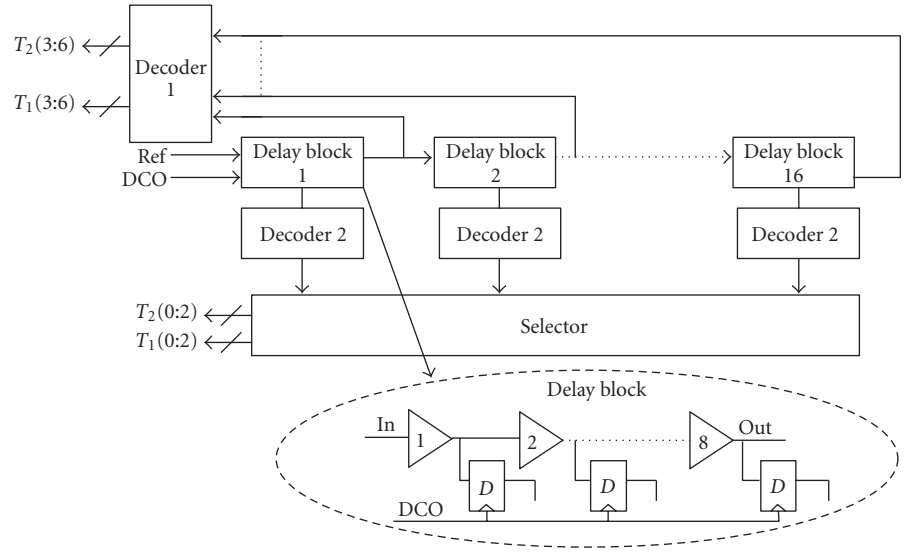


Figure 13: Block diagram of fractional TDC structure.

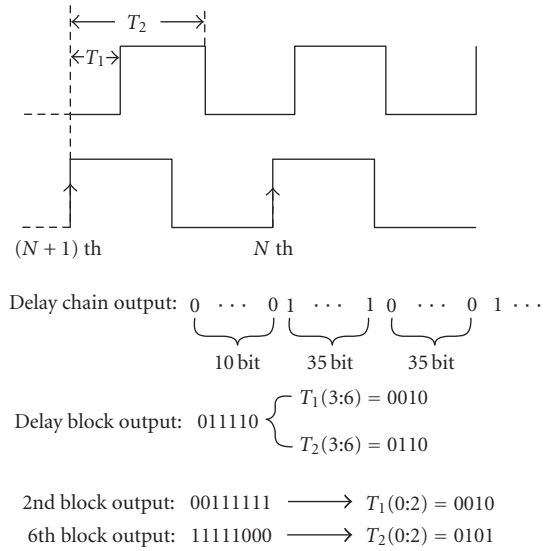


Figure 14: Decoding process of the TDC.

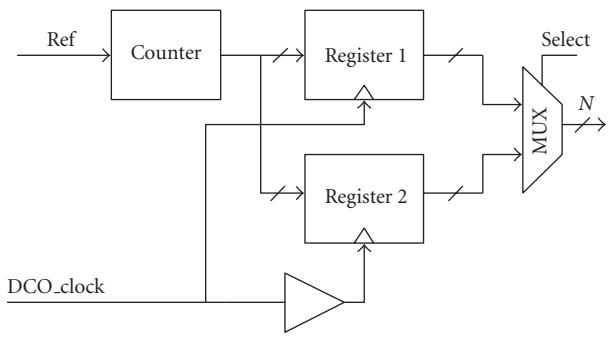


Figure 15: Block diagram of the integer counter.

Table 4: Impact of each control bit on the DCO period.

Status	Function
"000"	Fine the control word for $T = T_{\text{ref}}$
"001"	Reset DCO in order to find the delay value T_{Delay}
"010"	Find a new control word for $T = T_{\text{ref}} - T_{\text{Delay}}$
"011"	Reset DCO in order for the phase alignment
"100"	Maintenance Mode

register, is shifted to the left by one bit every cycle. When the polarity changes, the control word and the phase gain will be reset to the initial value stored during the frequency acquisition. The edge detector keeps comparing the phase difference and updating the fine control bits in order to maintain the phase lock. The fine resolution of the DCO as well as the bit shift strategy provides a fast phase lock-in time and better jitter performance.

3.2. Circuit Designs.

(1) Time-To-Digital Converter.

The TDC used in this paper is composed of two parts: an integer counter that counts the reference clock edges within one DCO clock period and a fractional counter that quantizes the residual phase difference, which helps to improve the resolution of the proposed TDC.

The block diagram of the fractional TDC structure is shown in Figure 13. It consists of 16 delay blocks and two types of independent decoders. The resolution of the TDC is the delay of a single buffer, which minimizes the delay mismatch compared to the delay of a single inverter. The reference clock waveform propagates through a chain of 8×16 delay elements whose outputs are sampled by 8×16 flip-flops at the rising edge of each DCO_clock.

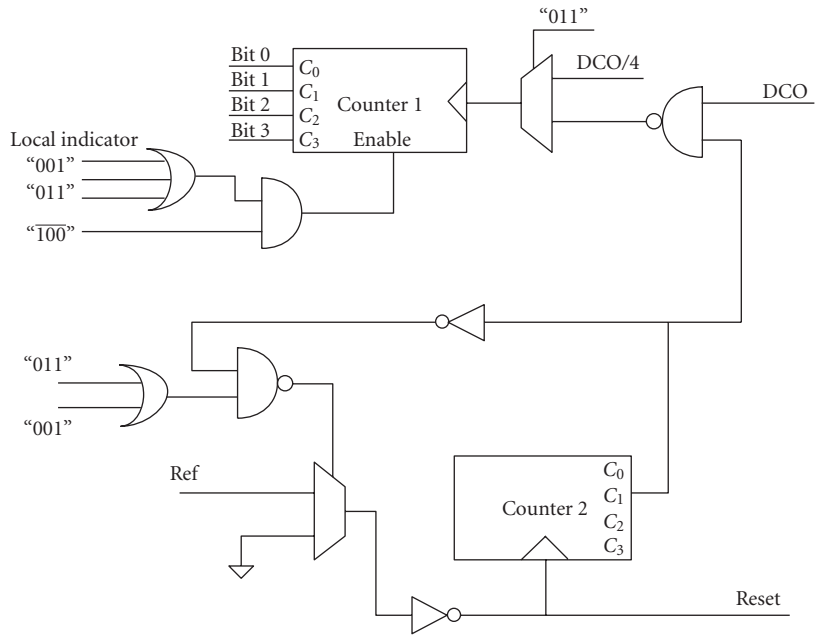


Figure 16: Block diagram of the state machine.

Table 5: Comparis on between conventional ADPLL and proposed ADPLL

	Approaches		
Items	Proposed ADPLL	Conventional ADPLL [11]	Conventional ADPLL [13]
Acquisition Time	10 cycles	18 cycles	50 cycles
Operation frequency range	570 MHz ~ 800 MHz	152 MHz ~ 366 MHz	50 MHz ~ 500 MHz
Lock-in range	Unlimited	Limited	Unlimited
Jitter	67 ps	150 ps	125 ps
Power consumption	19 mW @ 700 MHz	8.1–24 mW	40 mW @ 100 MHz

The decoding process is shown in Figure 14. The 16 bit output of the delay block is decoded into the higher 4 bits of T_1 and T_2. At the same time, the 8 bit output of each delay block is also decoded into a series of lower 3 bits of T_1 and T_2. Based on the output of decoder1, the proper set of T_1 $(0:2)$ and T_2 $(0:2)$ is selected. As is shown in Figure 14, the transition takes place in the 2nd and 6th blocks. As a result, the decoded output of those two blocks is selected as the lower 3 bits of T_1 and T_2. The separation of the decoder into two parts has greatly reduced the design complexity.

(2) Integer Counter.

As mentioned above, the nonideal setup time may result in an incorrect transition number N if T_1 is less than the setup time. The proposed integer counter that is designed to solve this problem is shown in Figure 15.

Using a delay buffer in the clock path, register2 is able to detect the closest rising edge of the reference clock while register1 cannot. The selected signal is the XOR output of the first delay block. When the rising edge of DCO_clock is slightly behind the edge of the reference clock, the transition will take place in the first delay block. As a result, the XOR output of this delay block will generate a logic high signal and the output of register2 will be selected. Apparently, when DCO_clock edge is slight ahead or T_1 is large enough, the XOR output of delay block 1 is logic low and the output of register1 is selected. This eliminates another possible error that register2 may store value of $N + 1$ instead of N when DCO_clock edge is slight ahead of reference clock edge.

(3) State Machine.

The state machine is the control unit of the proposed ADPLL, and it has five different kinds of working status as shown in Table 4. It takes the reference clock and DCO output as the input signals, and it outputs four-bit state signals such as bit0, bit1, bit2, and bit3 as well as a DCO reset signal as shown in Figure 16.

In the initial "000" status, the control word corresponding to T_{ref} is stored in register1. After that the lock indicator generates a high-voltage signal and ADPLL switches to "001" status. The delay between DCO_output and DCO_clock T_{Delay} will be measured by resetting the DCO using reference clock. Counter2 is used to make sure that the reset process only takes two cycles and it will be cleared after that. In "010" status, a new control word corresponding to $T_{\text{ref}} - T_{\text{Delay}}$ is stored in register2 and the corresponding lock indicates that signal will switch the status to "011". Then, the reset process will restart again with the control word corresponding to

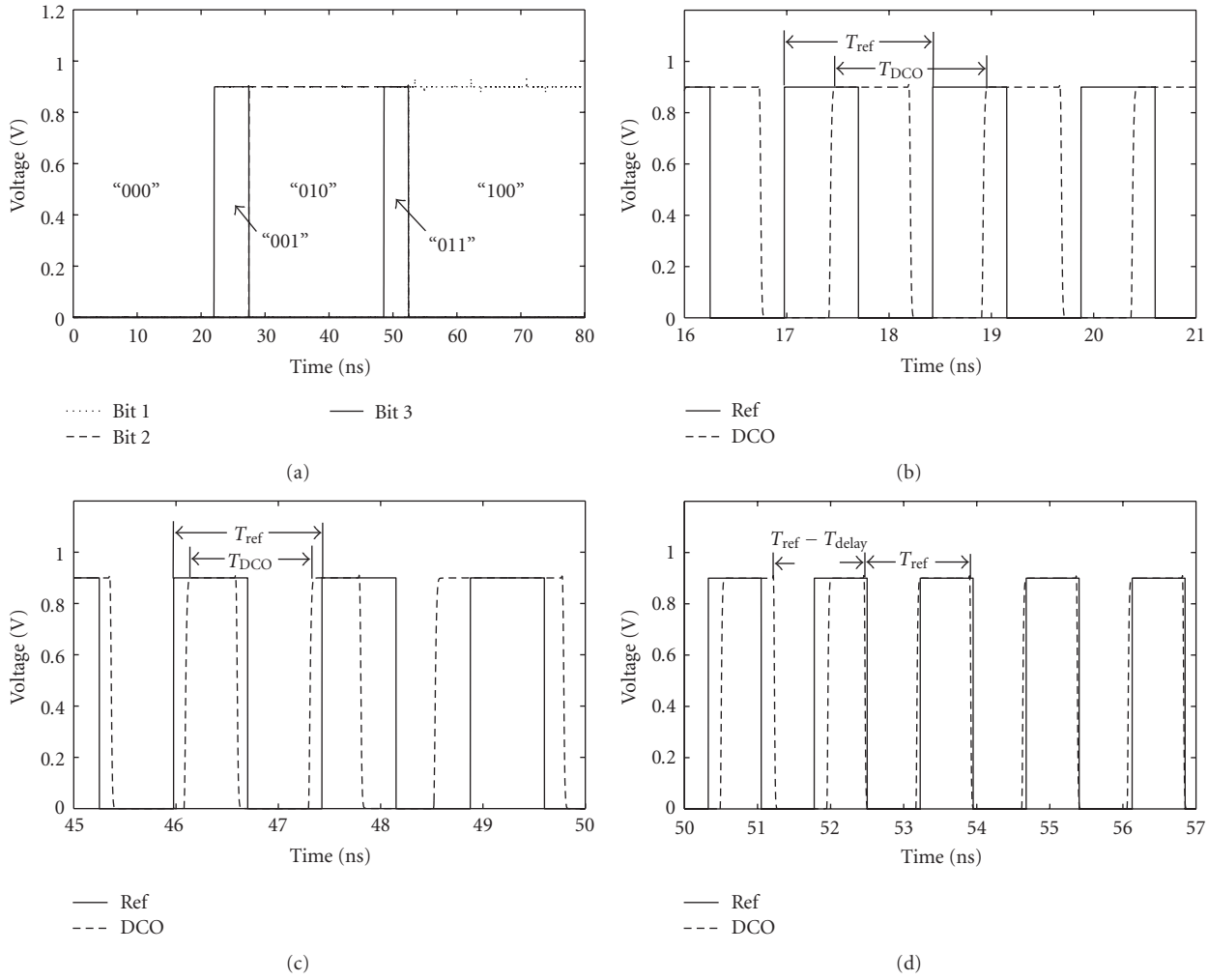


Figure 17: Lock-in process of the proposed ADPLL. (a) Lock process of the ADPLL with five different kinds of status from "000" status to "100" status. (b) Frequency acquisition at $T_{DCO} = T_{Ref}$ during "000" status, (c) Frequency acquisition at $T_{DCO} = T_{ref} - T_{delay}$ during "010" status, (d) Phase acquisition process during "011" status.

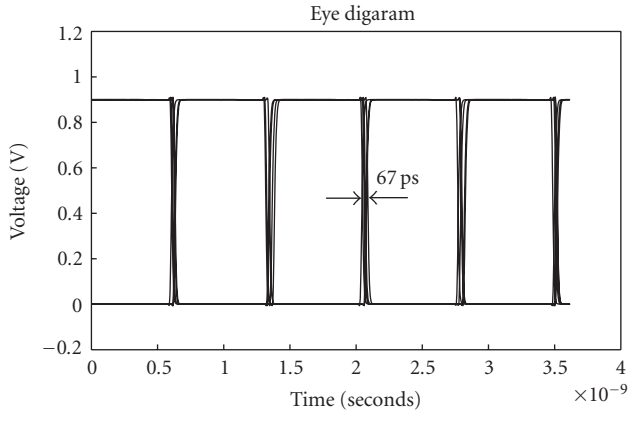


Figure 18: Output clock jitter (DCO @ 700 MHz)

$T_{ref} - T_{Delay}$ and T_{ref} in the first two cycles, respectively. After that, the ADPLL goes into the maintenance mode and the state machine will be locked. The five consecutive kinds of status ensure a fast lock-in and low-jitter ADPLL design.

4. Simulation Results of the ADPLL with the Proposed DCO

The proposed APDLL structure is designed and simulated using a 32 nm CMOS Predictive Transistor Model. The resolution of the TDC, which is the delay of a single buffer used in the delay chain, is 20 ps. The 12-bit digitally controlled oscillator has a coarse resolution close to 10 ps and fine resolution close to 1 ps with a tuning range from 570 MHz to 800 MHz.

The lock-in process of the proposed ADPLL is illustrated in Figure 17 when locking to 700 MHz. The output of the state machine ensures five consecutive kinds of status during the lock-in process as shown in Figure 17(a). Two-frequency lock-in processes are completed during "000" status and "010" status, respectively, and the corresponding control

words are stored in registers 1 and 2 as shown in Figures 17(b) and 17(c). The phase lock-in process takes 2 clock cycles, as in Figure 17(d). As a result, the whole lock-in process takes about ten reference cycles and phase acquisition process takes two cycles.

Figure 18 shows an eye diagram to show the DCO jitter performance during the maintenance mode after acquisition. As shown in the figure, this ADPLL achieves a peak-to-peak jitter of 67 ps at 700 MHz with the power supply of 0.9 V. Table 5 shows a comparison of the proposed ADPLL with the conventional ADPLL in items of acquisition time, jitter, operation frequency, power consumption, and locking range.

5. Conclusion

A 32 nm CMOS 12-bit digitally controlled CMOS oscillator design for low power consumption and low jitter is presented. The presented DCO demonstrates a good robustness to process, voltage, and temperature variations and better linearity comparing to the conventional design. The performance and the functionality of the DCO are verified through a novel ADPLL that uses the proposed DCO. This ADPLL is designed and implemented using 32 nm CMOS Predictive Technology Model for a frequency ranges of 570 MHz to 800 MHz at 0.9 V supply voltage. The overall lock-in process of the ADPLL takes about 12 reference cycles at 700 MHz with a peak-to-peak jitter less than 67 ps. The power consumption of the DCO is 2.2 mW at 650 MHz with supply voltage of 0.9 V. The presented results demonstrate that the proposed design is viable for various clock control systems for full digital implementations. The proposed work will be a good reference for future advanced ADPLL such as ADPLL that multiplies the reference clock frequency by a fractional number without using a fractional number divider.

References

[1] I. Hwang, S. Lee, S. Lee, and S. Kim, "A digitally controlled phase-locked loop with fast locking scheme for clock synthesis application," in *Proceedings of the 47th Annual IEEE International Solid-State Circuits Conference (ISSCC '00)*, pp. 168–169, San Francisco, Calif, USA, February 2000.

[2] D. W. Boerstler, "Low-jitter PLL clock generator for microprocessors with lock range of 340–612 MHz," *IEEE Journal of Solid-State Circuits*, vol. 34, no. 4, pp. 513–519, 1999.

[3] V. R. von Kaenel, "A high-speed, low-power clock generator for a microprocessor application," *IEEE Journal of Solid-State Circuits*, vol. 33, no. 11, pp. 1634–1639, 1998.

[4] P. Larsson, "A 2–166 MHz 1.2–2.5 V CMOS clock-recovery PLL with feedback phase-selection and averaging phase-interpolation for jitter reduction," in *Proceedings of IEEE International Solid-State Circuits Conference (ISSCC '99)*, pp. 356–357, San Francisco, Calif, USA, February 1999.

[5] I. A. Young, J. K. Greason, and K. L. Wong, "A PLL clock generator with 5 to 110 MHz of lock range for microprocessors," *IEEE Journal of Solid-State Circuits*, vol. 27, no. 11, pp. 1599–1607, 1992.

[6] C.-C. Chung and C.-Y. Lee, "An all-digital phase-locked loop for high-speed clock generation," *IEEE Journal of Solid-State Circuits*, vol. 38, no. 2, pp. 347–351, 2003.

[7] P. Nilsson and M. Torkelson, "A monolithic digital clock-generator for on-chip clocking of custom DSP's," *IEEE Journal of Solid-State Circuits*, vol. 31, no. 5, pp. 700–706, 1996.

[8] R. B. Staszewski and P. T. Balsara, "Phase-domain all-digital phase-locked loop," *IEEE Transactions on Circuits and Systems II*, vol. 52, no. 3, pp. 159–163, 2005.

[9] M. Saint-Laurent and G. P. Muyshondt, "A digitally controlled oscillator constructed using adjustable resistors," in *Proceedings of IEEE Southwest Symposium on Mixed-Signal Design*, pp. 80–82, Austin, Tex, USA, February 2001.

[10] P. Raha, S. Randall, R. Jennings, B. Helmick, A. Amerasekera, and B. Haroun, "A robust digital delay line architecture in a 0.13 μm CMOS technology node for reduced design and process sensitivities," in *Proceedings of the International Symposium on Quality Electronic Design (ISQED '02)*, pp. 148–153, San Jose, Calif, USA, March 2002.

[11] T. Olsson and P. Nilsson, "A digitally controlled pll for SoC applications," *IEEE Journal of Solid-State Circuits*, vol. 39, no. 5, pp. 751–760, 2004.

[12] R. B. Staszewski, J. Wallberg, S. Rezeq, et al., "All-digital PLL and GSM/EDGE transmitter in 90 nm CMOS," in *Proceedings of IEEE International Solid-State Circuits Conference (ISSCC '05)*, pp. 316–317, San Francisco, Calif, USA, February 2005.

[13] J. Dunning, G. Garcia, J. Lundberg, and E. Nuckolls, "An all-digital phase-locked loop with 50-cycle lock time suitable for highperformance microprocessors," *IEEE Journal of Solid-State Circuits*, vol. 30, no. 4, pp. 412–422, 1995.

[14] I.-C. Hwang, S.-H. Song, and S.-W. Kim, "A digitally controlled phase-locked loop with a digital phase-frequency detector for fast acquisition," *IEEE Journal of Solid-State Circuits*, vol. 36, no. 10, pp. 1574–1581, 2001.

[15] J. Zhao and Y.-B. Kim, "A 12-bit digitally controlled oscillator with low power consumption," in *Proceedings of the 51st IEEE International Midwest Symposium on Circuits and Systems (MWSCAS '08)*, pp. 370–373, Knoxville, Tenn, USA, August 2008.

[16] P.-L. Chen, C.-C. Chung, and C.-Y. Lee, "A portable digitally controlled oscillator using novel varactors," *IEEE Transactions on Circuits and Systems II*, vol. 52, no. 5, pp. 233–237, 2005.

[17] E. Roth, M. Thalmann, N. Felber, and W. Fichtner, "A delay-line based DCO for multimedia applications using digital standard cells only," in *Proceedings of IEEE International Solid-State Circuits Conference (ISSCC '03)*, vol. 1, pp. 432–505, San Francisco, Calif, USA, February 2003.

Hindawi Publishing Corporation
VLSI Design
Volume 2010, Article ID 213043, 11 pages
doi:10.1155/2010/213043

Research Article

FPGA Implementation of an Amplitude-Modulated Continuous-Wave Ultrasonic Ranger Using Restructured Phase-Locking Scheme

P. Sumathi[1] and P. A. Janakiraman[2]

[1] *DA-IICT, Department of Information and Communication Technology, Gandhinagar, Gujarat 382007, India*
[2] *Indian Institute of Technology Madras, Department of Electrical Engineering, Chennai 600036, India*

Correspondence should be addressed to P. Sumathi, sumichan04@yahoo.co.in

Received 30 May 2009; Revised 9 September 2009; Accepted 8 December 2009

Academic Editor: Ethan Farquhar

An accurate ultrasonic range finder employing Sliding Discrete Fourier Transform (SDFT) based restructured phase-locked loop (RPLL), which is an improved version of the recently proposed integrated phase-locking scheme (IPLL), has been expounded. This range finder principally utilizes amplitude-modulated ultrasonic waves assisted by an infrared (IR) pilot signal. The phase shift between the envelope of the reference IR pilot signal and that of the received ultrasonic signal is proportional to the range. The extracted envelopes are filtered by SDFT without introducing any additional phase shift. A new RPLL is described in which the phase error is driven to zero using the quadrature signal derived from the SDFT. Further, the quadrature signal is reinforced by another cosine signal derived from a lookup table (LUT). The pulse frequency of the numerically controlled oscillator (NCO) is extremely accurate, enabling fine tuning of the SDFT and RPLL also improves the lock time for the 50 Hz input signal to 0.04 s. The percentage phase error for the range 0.6 m to 6 m is about 0.2%. The VHDL codes generated for the various signal processing steps were downloaded into a Cyclone FPGA chip around which the ultrasonic ranger had been built.

1. Introduction

Ultrasonic sensors find applications generally in distance measurement, indoor mobile robot control, for environment information, gleaning, localization, and map building, vibration measurements, and safety systems like intelligent airbag control [1–4]. Many range-finding techniques are found in the literature, based on either the time of flight (TOF) or the continuous wave method [5–7]. A variety of continuous wave methods have been reported; notable among them are based on the multifrequency and amplitude-modulated (AM) schemes [8, 9]. The significant advantages of the AM continuous-wave method over the TOF were presented in [10]. The phase shift observed in the ultrasonic wave with respect to the distance traveled can be used to measure the range. For a 40 kHz sound wave, the maximum measurable range using phase shift is only 8.6 mm. In the present work, to enhance the measurable range to 6.86 m, the ultrasonic signal is amplitude modulated by a 50 Hz signal, which is more appropriate for mobile robot localization and navigation in indoor applications.

In this scheme, a low-frequency-modulated Infrared (IR) is used as a pilot signal. Another ultrasonic signal (US) modulated by the same low-frequency signal is utilized for estimating the range. A novel procedure using Sliding Discrete Fourier transform (SDFT) can be employed to extract the fundamental component of the envelope of the received ultrasonic signal [11]. The sampling pulse frequency of the SDFT block is tuned precisely by an integrated phase-locked loop so that exactly one full period of the envelope signal can be accommodated in a window of width 128 [12]. Two such PLL's, one for the extraction of the sinusoidal envelope of the infrared pilot signal, and another for ultrasonic signal are employed in the range-finding equipment. The PLL is basically a feedback circuit minimizing the phase error by correlating the given envelope signal with the quadrature

signal derived from the SDFT block. The envelope of the IR signal is the reference, against which the phase shift of the extracted envelope of the amplitude-modulated ultrasonic signal is compared. The integrated phase-locking scheme discussed in [12] shows a steady residual error in the NCO output frequency. The primary focus of the present paper is to describe the restructured phase-locking scheme, which makes use of a look up table to assist the quadrature signal derived from the SDFT block mainly to reduce the residual phase error. The hardware realization of the restructured phase-locking scheme which is implemented in an FPGA chip has also been described. The DSP builder tool, has been applied extensively for simulation and practical realization of an ultrasonic range finder.

The restructured phase locking scheme and its components have been described in Section 2. The proposed ultrasonic range measurement scheme based on RPLL, built using the DSP builder tool has been explicated in Section 3. The simulation and experimental results obtained using Cyclone FPGA have been presented in Section 4 and Section 5, respectively. The major conclusions are drawn in Section 6.

The schematic of the proposed ultrasonic range finder is shown in Figure 1(a). This consists of a transmitter and a receiver unit. The transmitter unit sends an IR pilot signal as well as an ultrasonic signal, both simultaneously modulated by a low-frequency sine wave. The receiver unit contains an IR receiver and an ultrasonic receiver. On the transmitter side a 40 kHz signal had been used to generate the ultrasonic carrier and a 50 Hz sinusoidal signal for amplitude-modulation. The 50 Hz sine wave is converted into square wave using a limiter circuit for modulating the IR pilot signal to serve as the instantaneous reference signal. An ultrasonic transducer transmits the amplitude modulated 40 kHz signal. Figure 1(b) shows the transmitted reference IR signal, ultrasonic signal, and the received ultrasonic signal. On the receiver side, the ultrasonic signal is amplified, rectified, and processed by an SDFT-based restructured phase-locking scheme to extract the sinusoidal envelope. The received reference IR signal is also processed using another SDFT based RPLL for extracting the fundamental component of the envelope. The detailed explanation of this scheme is presented in the next section.

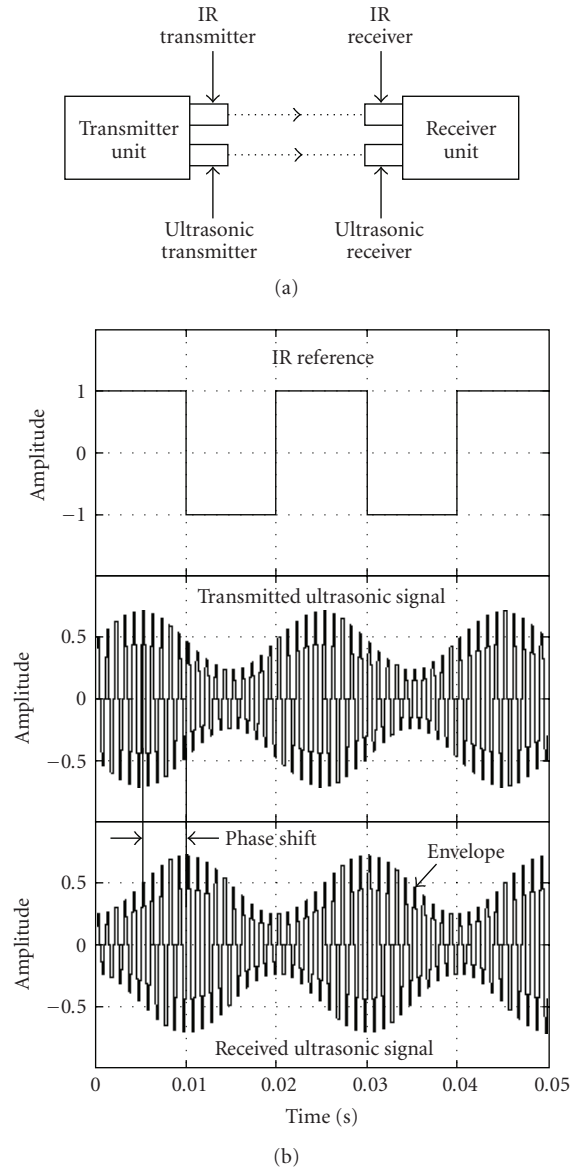


Figure 1: (a) Schematic of the proposed Ultrasonic Range Finder and (b) IR reference signal, transmitted ultrasonic signal, and received ultrasonic signal.

2. Restructured Phase-Locking Scheme

The operation of the restructured phase locked-loop (RPLL) based on SDFT is briefly explained here. The detailed block diagram of RPLL scheme is shown in Figure 2. When a periodic input signal is passed through a sliding DFT block tuned to a particular frequency, two distinct output signals are obtained. The first signal is basically the fundamental component of the incoming periodic signal, which may, however, be slightly shifted in phase, whenever the frequency of the incoming signal deviates from the frequency for which the SDFT block is tuned [12]. The SDFT block yields another signal, which is in quadrature with the first output signal. The correlation between the input signal and the quadrature signal from the SDFT can be made use of, for adjusting the frequency of the numerically controlled oscillator (NCO) which provides the sampling pulses to the SDFT block, resulting in the phase lock of the SDFT output [12]. Under these conditions, the enabling or the sampling frequency $f_s = fN$, where f is the cyclic frequency of the periodic input signal and N is the number of samples per cycle. Small phase errors may still persist which can be greatly reduced by the addition of an LUT-based pure quadrature signal. Essentially, this scheme differs from the simple integrated phase-locking scheme presented in [12]; in that, the quadrature signal output of the SDFT block is supplemented by a cosine wave from a lookup table accessed by an address counter which in turn is driven by NCO pulses. The added cosine

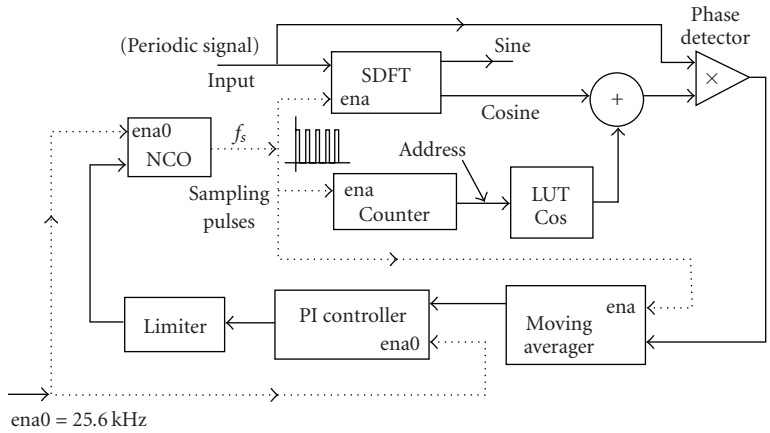


Figure 2: Block diagram of restructured phase-locking scheme.

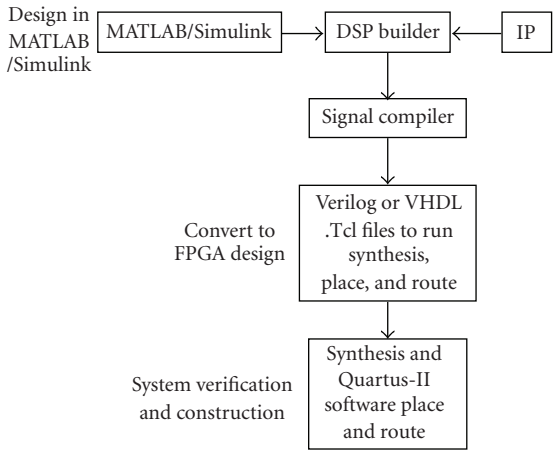


Figure 3: Synthesis flow of Matlab/Simulink-DSP builder and Quartus-II.

signal from the LUT reduces the steady residual error present in the control signal of the NCO [13]. The modification results in the NCO output frequency becoming very accurate compared to the simple integrated phase-locking scheme (IPLL), and hence provides fine phase locking with the fundamental component of the input signal.

2.1. DSP Builder-Quartus-II Tool. The MATLAB/Simulink-DSP builder software had been used for realizing the proposed Sliding DFT-based RPLL scheme. DSP Builder signal compiler block reads Simulink model files (.mdl) which are built using DSP builder and MegaCore blocks and generates VHDL files and Tool command languages (.Tcl) scripts for synthesis [14], hardware implementation, and simulation. The synthesis flow of the DSP builder tool and Quartus-II to generate the VHDL codes is shown in Figure 3. The generated VHDL codes can be downloaded into FPGA from PC through JTAG cable for processing.

2.2. Sliding DFT. The SDFT transfer function including the damping factor "r" can be expressed as

$$H_k(z) = \frac{(1 - r^N z^{-N}) z^{-1} \left(r e^{j2\pi k/N}\right)}{1 - r e^{j2\pi k/N} z^{-1}}, \quad r < 1, \ [k = 0, 1, 2, 3, \ldots, N-1], \tag{1}$$

where k is the bin index, which can vary from 0 to $N - 1$, and N is the window width. The damping factor $r < 1$ is introduced in the SDFT transfer function [15] to avoid numerical instability. The in-phase and quadrature signals can be obtained from SDFT block. For $k = 1$, the SDFT can extract the fundamental component present in the received signal. The real and imaginary parts of the SDFT transfer function can be written as

$$\mathrm{Re}[H_1(z)] = \frac{(1 - r^N z^{-N}) z^{-1} (r \cos(2\pi/N) - r^2 \ z^{-1})}{1 - 2r \cos(2\pi/N) z^{-1} + r^2 z^{-2}},$$
$$\mathrm{Im}[H_1(z)] = \frac{(1 - r^N z^{-N}) z^{-1} (r \sin(2\pi/N))}{1 - 2r \cos(2\pi/N) z^{-1} + r^2 z^{-2}}. \tag{2}$$

The block diagram realization of the SDFT transfer function given in equation in (1) is shown in Figure 4(a). A comb filter and a resonator are connected in cascade in the SDFT structure. For the sake of clarity, bus-width and port symbols were removed in the realization diagram shown in Figure 4(a). The SDFT transfer function has N number of zeros and a single pole lying on the unit circle in the z-plane. For $k = 1$, the single pole at $z = e^{j2\pi/N}$ cancels the zero corresponding to that location. The SDFT is a tuned filter, which passes only fundamental frequency present in the input signal and rejects all other harmonics and d.c. The pole-zero diagram of the SDFT is shown in Figure 4(b). The damping factor r is chosen as 0.9997. For maintaining the accuracy, a floating-point bit format of [2 : 18] (bus width) has been used for the variables in the SDFT block. The magnitude of the input signal is chosen small enough to avoid over-flow.

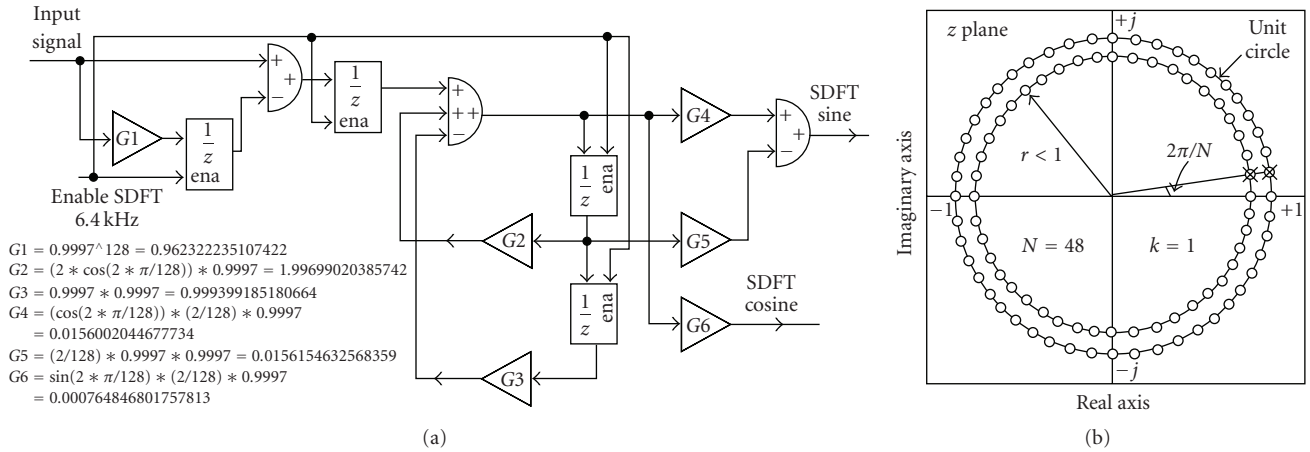


Figure 4: (a) Structure of Sliding DFT transfer function $H_k(z)$ for the kth bin and (b) pole-zero diagram of the SDFT.

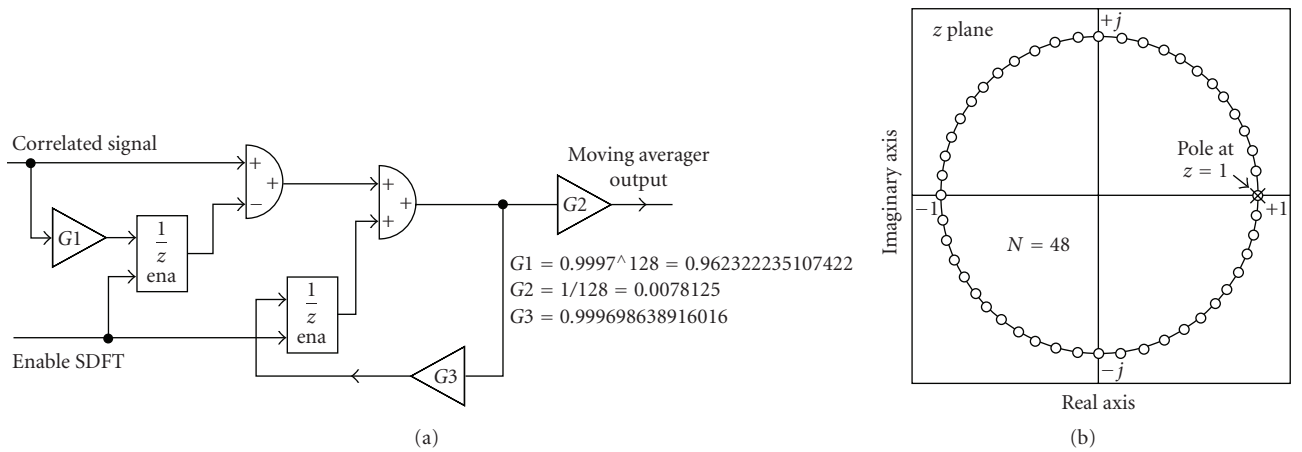


Figure 5: (a) Block diagram realization of the Moving Averager and (b) pole-zero diagram of Moving Averager.

2.3. Moving Averager. The phase error signal is obtained from the phase detector basically by multiplying the input signal with the quadrature signal generated by the SDFT block, and the Moving averager provides the average of the resultant product. The Moving Averager has N number of zeros and a pole at $z = 1$ on the unit circle. This arrangement results in the cancellation of the zero at $k = 0$ location, passing only the d.c. signal while filtering the fundamental and all other harmonics from the given periodic input signal. The block diagram realization and the pole-zero diagram of Moving Averager is shown in Figures 5(a) and 5(b), respectively.

2.4. PI Controller and Limiter. A PI controller processes the output from the Moving Averager. The PI controller is intended for reducing the steady residual phase error. The block diagram realization of PI controller is shown in Figure 6. The output of the PI controller is limited to ±1.

2.5. Numerically Controlled Oscillator. The NCO provides the sampling pulses required by the SDFT block and the Moving averager. The difference equation of NCO [16] can be expressed as

$$\begin{bmatrix} x_1(n+1) \\ x_2(n+1) \end{bmatrix} = \begin{bmatrix} \alpha & \alpha - 1 \\ \alpha + 1 & \alpha \end{bmatrix} \begin{bmatrix} x_1(n) \\ x_2(n) \end{bmatrix}, \quad x_1(0) = 1,\ x_2(0) = 0, \tag{3}$$

where $\alpha = \cos(\Psi)$ and $\Psi = 2\pi f_s / f_{\text{ena0}}$; f_s is the frequency of oscillation and f_{ena0} is the enabling (triggering) frequency of the delay elements. The block diagram realization of the difference equation of the simple NCO is shown in Figure 7. A nonlinear automatic feedback gain control is suggested for stabilizing the amplitude of NCO [16]. The enabling frequency of delay elements for the NCO is fixed at 25.6 kHz to get the sampling frequency of SDFT as 6.4 kHz so that a 50 Hz fundamental frequency signal is sampled 128 times per cycle. The comparison of NCO pulse frequency obtained from the IPLL and RPLL schemes is given in Table 1. For example, a cyclic frequency of 40 Hz and the window width $N = 128$, the bin index $k = 1$, $f = kf_s/N$, and the exact sampling frequency required by SDFT block being 5120 Hz

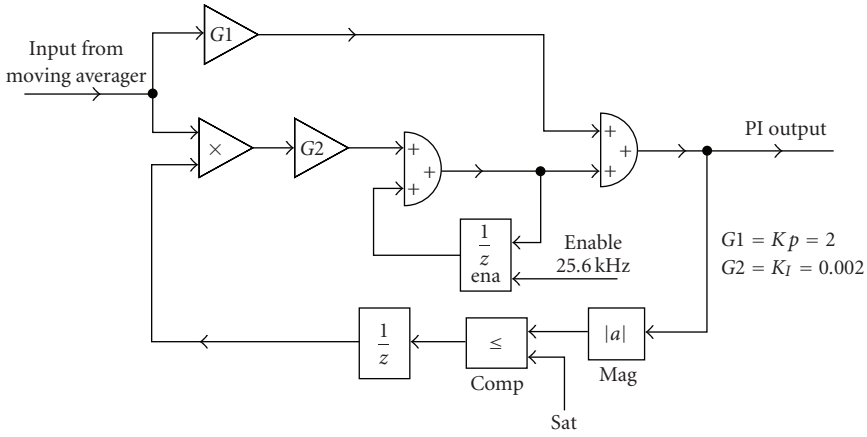


Figure 6: Block diagram realization of the PI controller and saturation.

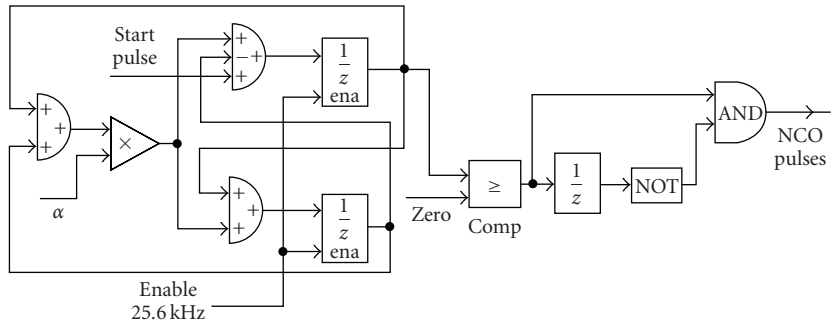


Figure 7: Block diagram realization of simple NCO.

Table 1: Comparison of NCO frequency.

S. no.	Input signal frequency (Hz)	IPLL NCO frequency (Hz)	RPLL NCO frequency (Hz)
1	40	5115	5120
2	45	5754	5760
3	50	6394	6400
4	55	7033	7040
5	60	7672	7680

are also given. From the entries in Table 1 it is seen that the design based on IPLL gives only 5115 sampling pulses per second resulting in an error of 5 pulses per second in the NCO output frequency. The RPLL scheme supplies the very accurate sampling pulse rate of 5120.

2.6. Cosine LUT, Counter, and Multiplier. The addition of cosine LUT has the effect of another PI controller, which acts in parallel to the existing PI controller in the RPLL scheme [17]. The increase in Cosine LUT magnitude improves the lock time of the PLL exhibiting overshoots in the response. The input signal to Cosine LUT ratio has been fixed as 0.5/0.25, for the simulation and experimental studies.

3. The Ultrasonic Range Measurement System Based on RPLL

The block diagram of the RPLL-based ultrasonic range measurement system is shown in Figure 8. This system comprises of the two channels, one for the reference IR pilot signal and the other for the ultrasonic signal. Both the IR and delayed ultrasonic signals are received at a targeted distance, and processed in the respective RPLL's. The phase shift between the IR and ultrasonic signals, which is proportional to the range, is computed using Park Transform. The realization of the proposed ultrasonic range measurement scheme is shown in Figure 9, which comprises of two RPLL's and Park transform blocks.

3.1. The PARK Transform. In the amplitude-modulated continuous-wave method, the phase shift information is measured at steady state, which is proportional to the range being measured.

The RPLL output corresponding to the block $SDFT_1$ shown in Figure 8 yields the reference IR in-phase and quadrature signals while the RPLL output of the $SDFT_2$ block gives the ultrasonic in-phase and quadrature signals.

The reference signals coming out from unit sine and cosine LUT's of RPLL of the IR channel are $\sin(\omega t)$ and $\cos(\omega t)$, and those from the ultrasonic RPLL block are $\sin(\omega t - \Delta\phi)$ and $\cos(\omega t - \Delta\phi)$. The Park transform

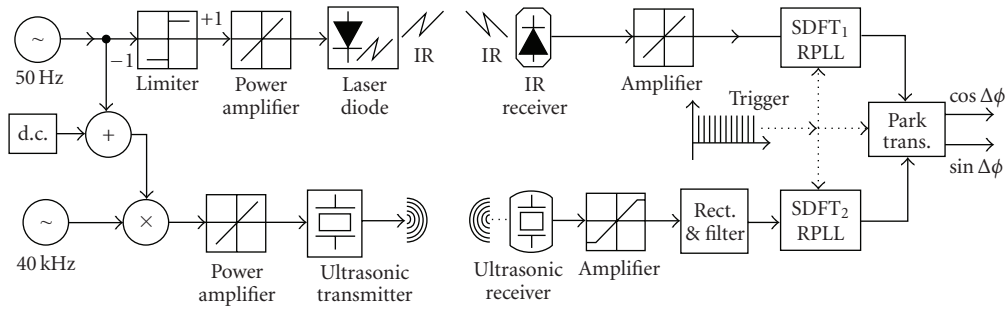


Figure 8: Block diagram of the RPLL-based range measurement scheme.

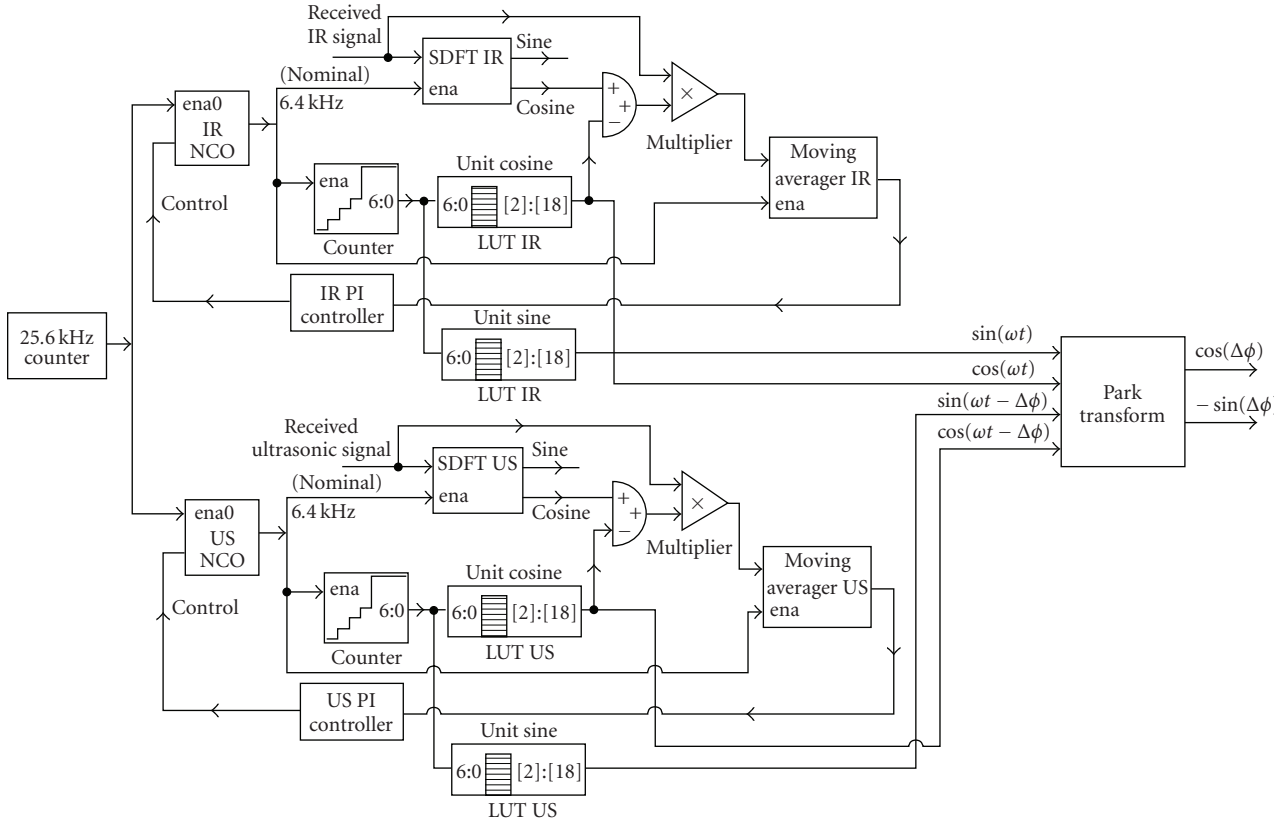


Figure 9: Realization of RPLL-based range measurement scheme.

procedure can be applied to compute the phase shift between the IR reference and the delayed ultrasonic signal, which can be written as

$$\begin{bmatrix} y \\ x \end{bmatrix} = \begin{bmatrix} \cos(\omega t) & \sin(\omega t) \\ -\sin(\omega t) & \cos(\omega t) \end{bmatrix} \begin{bmatrix} \cos(\omega t - \Delta\phi) \\ \sin(\omega t - \Delta\phi) \end{bmatrix}, \quad (4)$$

where

$$\begin{bmatrix} y \\ x \end{bmatrix} = \begin{bmatrix} \cos(\Delta\phi) \\ -\sin(\Delta\phi) \end{bmatrix}. \quad (5)$$

The signals y and x obtained from (5) are steady d.c. signals.

The phase shift can be computed using the atan2 function as

$$\Delta\phi = \text{atan2}(x, y). \quad (6)$$

The realization of the Park transform equations is shown in Figure 10.

3.2. Hardware Resource Utilization. The hardware resource utilization and the corresponding logic block details of the entire range measurement scheme have been listed in Table 2. The proposed algorithm demands only 56% of the total chip area. The Cyclone II chip contains 516 Logic Array Blocks

Table 2: Hardware resource utilization and logic blocks of the entire range measurement algorithm.

FPGA: Cyclone-II	
Device: EP2C8T144C8	
Total logic elements: 4,659/8,256 (56%)	
Combinational with no register	4148
Register only	1
Combinational with a register	510
Total registers: 511/8256 (6%)	
LC registers	511
Embedded multiplier 9-bit elements: 36/36 (100%)	
DSP elements	36
DSP 9×9	0
DSP 18×18	18
Total pins: 62/85 (72%)	
Total memory bits: 20,320/165,888 (12%)	

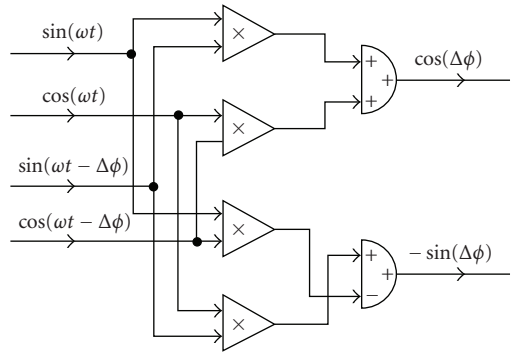


Figure 10: Realization of the Park Transform.

(LABs) with each LAB containing 16 Logic Elements (LEs), 18 Embedded multipliers (18×18) or 36 multipliers (9×9), 36 soft multipliers (16×16), and memory bits of 1,65,888 (4608 RAM bits $\times$ M4K RAM blocks). Table 2 indicates that 100% of the embedded multipliers are used. The synthesizer tool handles the available resources for optimum utilization and performance of the given algorithm. Among the available resources, first, the embedded multipliers are exhausted; the multipliers present in the remaining part of the algorithm had been realized using logic elements. The unusual bus width of 20 bits, not divisible by 9, in the algorithm, increases the multiplier utilization. Hence only 56% of the FPGA hardware space is used for the entire range finder. This summary in Table 2 contains the two RPLL schemes, one for the IR and another for ultrasonic envelope and a phase shift computation block. All the utilized resources like total logic elements and embedded multipliers are simultaneously operating in the FPGA. The signals from IR and ultrasonic sensors are sent to the FPGA by A/D converters. The processed digital signals from FPGA are converted into analog signals by D/A converters.

4. Simulation Results

The RPLL-based range measurement scheme has been simulated in MATLAB/Simulink-DSP builder environment. The simulation was carried out at a sampling frequency of 51.282 kHz. A 50 Hz sine wave with 128 samples per cycle had been chosen for simulation studies. Figure 11 shows the realization of the RPLL structure. When the range is about 86.25 cm, the phase shift quantifies to 45°. The signals from the sine LUT of the IR reference RPLL and the sine LUT of the received ultrasonic RPLL are shown in Figure 12. The corresponding unit cosine signals from the IR loop and the ultrasonic loop are shown in Figure 13. These four signals are processed by the Park Transform to yield the phase shift. Significantly, the unit sine and cosine signals from the LUT's present in the PLL make the computation of phase shift independent of the received signal magnitude. Additionally, the signals derived from LUT's make the NCO pulse frequency quite accurate. Also this improves the lock time for the 50 Hz input signal to 0.04 s whereas IPLL takes 0.15 s for the same 50 Hz input frequency.

5. Experimental Results

The bimorph-type ultrasonic transducer used in the experiments can be tuned to antiresonance around 37 ± 2 kHz. Hence the carrier frequency was fixed at its maximum value of 39 kHz. In the simulation studies a 40 kHz carrier signal was used, which is close enough to 39 kHz. The envelope frequency could be chosen as 50 Hz or 25 Hz. With 50 Hz, the measurable range could be 6.86 m while the use of 25 Hz envelope enhances the range to 13.72 m. The envelope frequency of 51.2 Hz ($\approx$50 Hz) has been chosen for modulation process and testing the proposed RPLL scheme.

These two signals were generated using two Wien bridge oscillators. The amplitude-modulated ultrasonic signal and the IR signals were sent and received at targeted distance by the matching receivers. The restructured phase-locking scheme had been employed for the IR and ultrasonic channels. The received IR reference square wave and extracted ultrasonic envelope from SDFT at a distance of 42 cm are shown in Figure 14. The reference square wave and unit sine obtained from the LUT present in the RPLL of the ultrasonic channel are shown in Figure 15. The unit sine obtained from LUT is comparatively purer than the envelope extracted from the SDFT, which makes the phase shift computation more accurate. The received rectified ultrasonic signal along with the corresponding LUT generated unit sine wave, which is very much equivalent to the extracted ultrasonic envelope is shown in Figure 16.

5.1. Hardware-In-Loop Test. The Hardware-In-Loop (HIL) [18] test was conducted for calibrating the ultrasonic range-finding equipment built around the Cyclone FPGA. The algorithm was developed in MATLAB-Simulink-Altera-DSP builder. In this environment the developed *.mdl files are converted to VHDL codes using Quartus-II software. The generated VHDL codes from PC are downloaded into the FPGA using JTAG cable. For conducting this test, first of

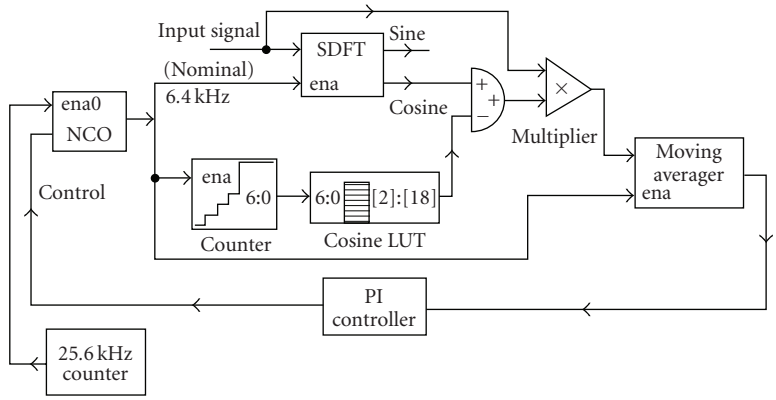


Figure 11: Realization of restructured phase-locking scheme.

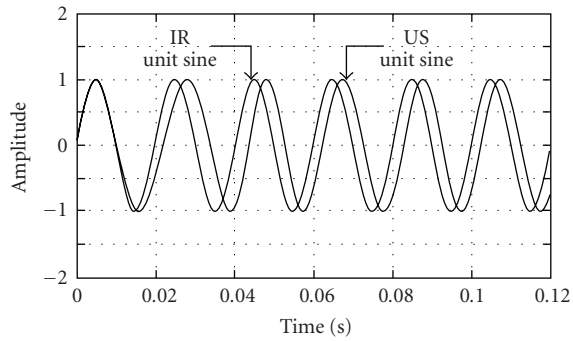


Figure 12: Reference and ultrasonic signals obtained from IR Sine LUT and US Sine LUT for 45° phase shift (Simulation).

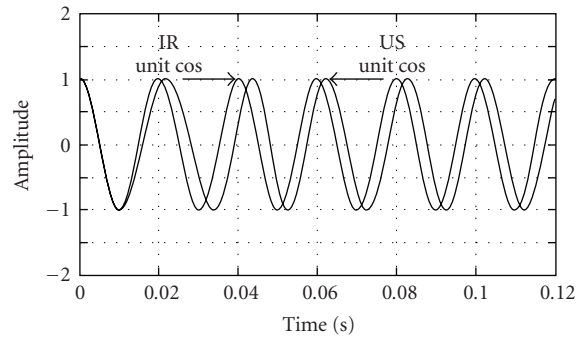


Figure 13: Reference and ultrasonic signals obtained from IR Cos LUT and US Cos LUT for 45° phase shift (Simulation).

all, the received IR and ultrasonic signals have been passed into the FPGA using suitable interfaces and the computed signals $\cos(\Delta\phi)$ and $\sin(-\Delta\phi)$ from the Park transform block are transferred through JTAG cable to the PC, where the phase shift is computed in the MATLAB environment using the atan2 function. The HIL test helps in computing the numerical value of phase shift in the same MATLAB environment. Adding the Hardware in the Loop (HIL) block to Simulink model allows cosimulating of a Quartus-II software design with a physical FPGA board implementing most of the design. A simple JTAG acts as interface between Simulink and the FPGA board. Figure 17 shows the block diagram of the HIL test configuration. The real time signals are accepted through ADC and processed in the FPGA. The processed signals are passed through the DAC for the displaying of the waveforms. The HIL loop operates between the Matlab/simulink environment and FPGA using JTAG interface.

The calibration graph is shown in Figure 18, proving that the phase shift and the range are linearly related. The percentage error obtained from calibration data is around 3% at closer distances (30 cm) and it reduces to 0.2% at larger distances (0.6 m to 6 m). At closer distance the observed percentage error is 3%. This is because the phase shift is very small and the quantization error in the [2] : [18] format becomes noticeable. A larger bit size for the variables (say [2] : [24]) reduces such errors. Phase errors are also observed due to finite ($N = 128$) timewise discretization of the period of the envelope signal. A larger bit size and a wider window ($N = 256$) may reduce maximum possible phase errors, demanding, however, extra hardware space.

Most of the commercially available range finders are based on the Time-Of-Flight (TOF) principle useing either single pulse or burst of ultrasonic waves. The comparison factors involve maximum measurable range, data update rate, hardware complexity, accuracy, cost, and performance in cluttered environment.

Literature shows that the available range finders have conflict among these parameters and are yet to be categorized. However, in TOF procedure, the measurable range appears to be 2 m to 10 m and the accuracy based on maximum distances is 1%. For example, the commercial ultrasonic ranger Maxsonar uses the ultrasonic bursts for target detection and the resolution is found to be 1 cm in the measurable range of 20 cm to 6.5 m. The reflected amplitude-modulated continuous wave method [9] with a 150 Hz envelope can, however, be compared with the proposed scheme. In this method, a range of 2 m and an accuracy of 2 mm for a 1.5 m distance have been reported.

5.2. Power Dissipation Analysis. The power planning is an important consideration when the circuit design turns out

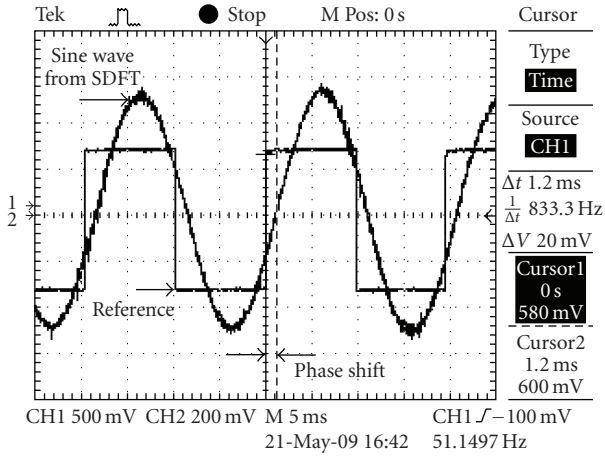


Figure 14: Reference square wave and extracted envelope using SDFT for 42 cm range showing 22 deg phase shift (Experiment).

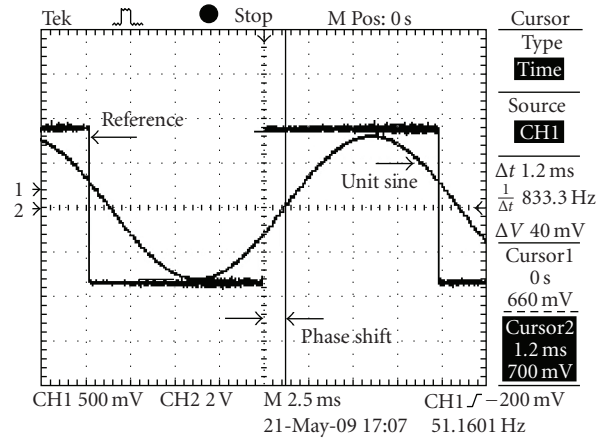


Figure 15: Reference square wave and unit sine from LUT equivalent to extracted envelope for 42 cm range showing 22 deg phase shift (Experiment).

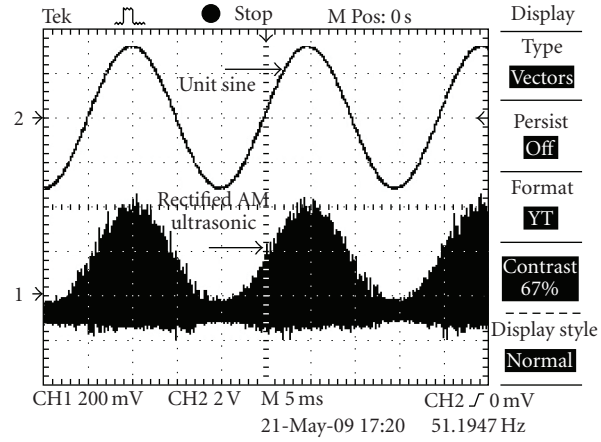


Figure 16: Received rectified, AM ultrasonic signal and the unit sine from LUT (Experiment).

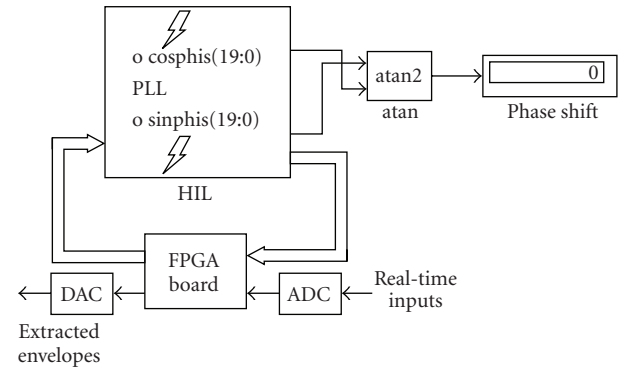


Figure 17: Block diagram of the MATLAB/Simulink-FPGA board for HIL test.

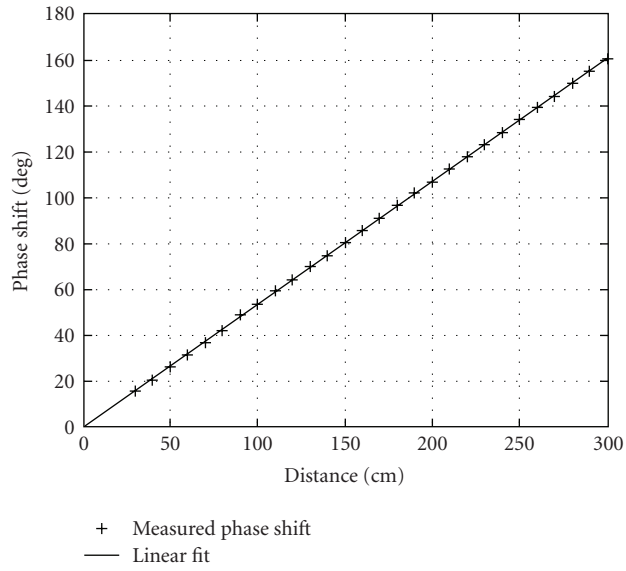


Figure 18: Distance versus phase shift (Experiment).

to be larger. The power dissipation analysis [19] for the design using DSP builder blocks had been carried out in the Quartus-II environment on the generated equivalent VHDL code. The signal activity file (.saf) corresponding to the VHDL code had been generated using the default toggle rates for the input signal. The summary obtained from the power dissipation analysis using the Quartus-II power analyzer tool is listed in Table 3.

From Table 3 total thermal power dissipation is sum of the dynamic and static power dissipation. Routing plays an important role in power dissipation. Routing is formed by the combination of resistors and capacitors. The resistors dissipate static power but the capacitors consume dynamic power.

Dynamic power is dissipated in the resources utilized as blocks in the algorithm and routing. The logic cell registers, pins loaded with capacitance, and memory bits consume thermal dynamic power of 0.74 mW. The dynamic power dissipated in clock control blocks due to routing is 0.29 mW. The total dynamic power dissipation amounts to be 1.03 mW.

Table 3: Power dissipation analysis.

Total thermal power dissipation = 44.56 mW							
Dynamic thermal power dissipation = 1.03 mW				Static thermal power dissipation = 43.53 mW			
The block thermal Dynamic power (mW)		Routing thermal Dynamic power (mW)		The block thermal Static power (mW)		Routing thermal Static power (mW)	
Logic cell registers	0.28	Clock control blocks	0.29	Pins	5.95	Resistors formed by routing	37.58
Pins	0.02	—	—	—	—	—	—
Memory bits	0.44	—	—	—	—	—	—
	0.74		0.29		5.95		37.58
Total dynamic thermal power Dissipation = 0.74 + 0.29 = 1.03 mW				Total static thermal power Dissipation = 5.95 + 37.58 = 43.53 mW			
Total thermal power dissipation = 1.03 + 43.53 = 44.56 mW							
Block average toggle rate (millions of transitions/sec)							
Pins				0.271			
Clock control blocks				2.667			

Similarly, the static power is dissipated in pins if they are used as terminated I/O's and in routing the algorithm. From Table 3, the pins consume 5.95 mW and the resistors formed during routing dissipate 37.58 mW. The total static power dissipation quantifies to be 43.53 mW.

The power analyzer examines the algorithm for the total power dissipation as heat in the FPGA device, and the estimation was found to be 44.56 mW.

5.3. Application of the Proposed Range Finder. The proposed range measurement system is very much suitable for localization and navigation of mobile robots in an indoor environment [20]. The IR and the US transmitters could be located in the ceiling or high on the wall of a room. The number of IR and US transmitters could be positioned at the required specific locations. The receiver unit could be mounted on the mobile robot to receive the signals transmitted by the sensors on the ceiling. The signals received by the sensors fixed on the mobile robot are processed for the distance information, which helps in position determination of mobile robot localization. The received signals in this direct path scheme are less prone to noise than reflected or indirect method also enables to get quicker measurement readings.

6. Conclusion

A phase-locking scheme has been presented which essentially uses the Sliding Discrete Fourier Transform block. The basic PLL built around the SDFT block had been restructured, by introducing an LUT to assist the quadrature signal of the SDFT. This restructured phase-locking arrangement has been found to reduce the errors in the phase measurement, basically by making the sampling frequency provided by the NCO very accurate simultaneously reducing the lock time. These restructured PLL's yield pure equivalent unit sinusoidal signals corresponding to the IR pilot signal and the envelope of the ultrasonic signal. These signals have enabled the measurement of the phase shift via the well-known Park transform procedure. The experimental results have validated the simulation of the proposed range finder. The extraction of the envelopes is carried out under steady state using the accurate RPLL scheme and the measurement ignores any portion of the ultrasonic signals missed initially. These factors lead to percentage errors as low as 0.2% at larger distances in the range of 0.6 m to 6 m in controlled environments. Interestingly, the summary of hardware resource utilization gives a convincing proof that the proposed ultrasonic-IR range measurement system can be accommodated in a single Cyclone-II FPGA chip.

References

[1] C.-Y. Lee, H.-G. Choi, J.-S. Park, K.-Y. Park, and S.-R. Lee, "Collision avoidance by the fusion of different beam-width ultrasonic sensors," in *Proceedings of IEEE Sensors Conference*, pp. 985–988, October 2007.

[2] G. Hueber, T. Ostermann, T. Bauernfeind, R. Raschhofer, and R. Hagelauer, "New approach of ultrasonic distance measurement technique in robot applications," in *Proceedings of International Conference on Signal Processing (WCCC-ICSP '00)*, vol. 3, pp. 2066–2069, August 2000.

[3] O. Manolov, Sv. Noikov, P. Bison, and G. Trainito, "Indoor mobile robot control for environment information gleaning," in *Proceedings of IEEE Intelligent Vehicles Symposium*, pp. 602–607, October 2000.

[4] A. Nemecek, K. Oberhauser, and H. Zimmermann, "Distance measurement sensor with PIN-photodiode and bridge circuit," *IEEE Sensors Journal*, vol. 6, no. 2, pp. 391–397, 2006.

[5] D. Marioli, C. Narduzzi, C. Offelli, D. Petri, E. Sardini, and A. Taroni, "Digital time-of-flight measurement for ultrasonic sensors," *IEEE Transactions on Instrumentation and Measurement*, vol. 41, no. 1, pp. 93–97, 1992.

[6] C.-C. Tong, J. F. Figueroa, and E. Barbieri, "A method for short or long range time-of-flight measurements using phase-detection with an analog circuit," *IEEE Transactions on*

Instrumentation and Measurement, vol. 50, no. 5, pp. 1324–1328, 2001.

[7] M. Parrilla, J. J. Anaya, and C. Fritsch, "Digital signal processing techniques for high accuracy ultrasonic range measurements," *IEEE Transactions on Instrumentation and Measurement*, vol. 40, no. 4, pp. 759–763, 1991.

[8] F. E. Gueuning, M. Varlan, C. E. Eugene, and P. Dupuis, "Accurate distance measurement by an autonomous ultrasonic system combining time-of-flight and phase-shift methods," *IEEE Transactions on Instrumentation and Measurement*, vol. 46, no. 6, pp. 1236–1240, 1997.

[9] H. Hua, Y. Wang, and D. Yan, "A low-cost dynamic range-finding device based on amplitude-modulated continuous ultrasonic wave," *IEEE Transactions on Instrumentation and Measurement*, vol. 51, no. 2, pp. 362–367, 2002.

[10] P. Sumathi and P. A. Janakiraman, "Sliding DFT based ultrasonic ranger," in *Proceedings of the 51st Midwest Symposium on Circuits and Systems (MWSCAS '08)*, pp. 862–865, Knoxville, Tenn, USA, August 2008.

[11] P. Sumathi and P. A. Janakiraman, "A new demodulation scheme for AM signals based on sliding DFT," in *Proceedings of International Conference on Modeling and Simulation (Ms '07)*, vol. 2, pp. 861–865, AMSE, December 2007.

[12] P. Sumathi and P. A. Janakiraman, "Integrated phase-locking scheme for SDFT-based harmonic analysis of periodic signals," *IEEE Transactions on Circuits and Systems II*, vol. 55, no. 1, pp. 51–55, 2008.

[13] M. T. Hill and A. Cantoni, "A digital implementation of a frequency steered phase locked loop," *IEEE Transactions on Circuits and Systems I*, vol. 47, no. 6, pp. 818–824, 2000.

[14] P. Molson, "Accelerating intellectual property design flow using Simulink for system on a programmable chip," in *Proceedings of the 35th Asilomar Conference on Signals, Systems and Computers*, vol. 1, pp. 454–457, November 2001.

[15] E. Jacobsen and R. Lyons, "The sliding DFT," *IEEE Signal Processing Magazine*, vol. 20, no. 2, pp. 74–80, 2003.

[16] C. S. Turner, "Recursive discrete-time sinusoidal oscillators," *IEEE Signal Processing Magazine*, vol. 20, no. 3, pp. 103–111, 2003.

[17] H. Alasady and M. Ibnkahla, "Design and hardware implementation of look-up table predistortion on ALTERA stratix DSP board," in *Proceedings of the Canadian Conference on Electrical and Computer Engineering (CCECE '08)*, pp. 1535–1538, Niagara Falls, Canada, May 2008.

[18] C. Siriteanu, S. D. Blostein, and J. Millar, "FPGA-based communications receivers for smart antenna array embedded systems," *EURASIP Journal on Embedded Systems*, vol. 2006, Article ID 81309, 13 pages, 2006.

[19] R. X. Gu and M. I. Elmasry, "Power dissipation analysis and optimization of deep submicron CMOS digital circuits," *IEEE Journal of Solid-State Circuits*, vol. 31, no. 5, pp. 707–713, 1996.

[20] J. J. Leonard and H. F. Durrant-Whyte, "Mobile robot localization by tracking geometric beacons," *IEEE Transactions on Robotics and Automation*, vol. 7, no. 3, pp. 376–382, 1991.

Hindawi Publishing Corporation
VLSI Design
Volume 2010, Article ID 512312, 14 pages
doi:10.1155/2010/512312

Research Article

FPGA-Based Software Implementation of Series Harmonic Compensation for Single Phase Inverters

K. Selvajyothi[1] and P. A. Janakiraman[2]

[1] *Department of Electrical Engineering, Indian Institute of Information Technology Design & Manufacturing Kancheepuram, IIT Madras campus, Chennai-600036, India*
[2] *Department of Electrical Engineering, Indian Institute of Technology Madras, Chennai-600036, India*

Correspondence should be addressed to K. Selvajyothi, ksjyothi@iiitdm.ac.in

Received 10 June 2009; Accepted 20 October 2009

Academic Editor: Gregory D. Peterson

This paper presents a single chip FPGA (Altera Cyclone II) controlled single phase inverter, programmed for the reduction of harmonics in the output voltage. Separate composite digital observers have been designed for extracting the fundamental and harmonic components of the voltage and the highly distorted current signals, particularly when the inverter supplies nonlinear loads. These observers have been embedded into the FPGA along with the controllers and I/O interfaces. The multiple observers yield very pure in-phase and quadrature voltage signals for use in the outer loop and similar signals for stabilizing the inner current loop. The Inverter could be modeled as a feed back control system with the fundamental component of the voltage as the desired output while the voltage harmonics take the role of noise creeping into the output. To obtain a very low total harmonic distortion in the voltage waveform, the well-known control strategy of using a very large feed back around the noise signal has been employed.

1. Introduction

Stand-alone inverters are commonly used in the case of power failure, to deliver power for critical loads, which demand purely sinusoidal voltage at the specified magnitude, frequency, and low total harmonic distortion (THDv). The THDv in industry should not exceed 5% as per the guidelines given in the IEEE Standard 519-1992. Fixed passive filters may not perform well, particularly when the operating frequency drifts far away from the set resonance frequency. Alternatively, active filters can be employed. Many control methods have been proposed basically for obtaining pure sinusoidal output with good voltage regulation and fast dynamic response [1–9]. Sinusoidal pulse width modulation (SPWM) schemes for stand-alone inverters have been shown to perform well with linear loads [10]. However, with nonlinear loads the SPWM scheme does not guarantee low distortion in the output voltage. The availability of low cost microprocessors has led to discrete-time methods, such as repetitive control [1, 2], sliding mode control [3], and deadbeat control [4, 5] to improve the performance. To get zero steady-state error in the output voltage and fast response virtual inductor, capacitor and a resistor were used in [6], while internal model control scheme (IMC) was employed in [7]. The control methods presented in [8, 9] employ two-feedback control loops. The inner loop is used for current control and the outer loop is used for voltage control. Many of these methods have not specifically considered the reduction in distortion due to nonlinear loads.

The emergence of FPGAs has drawn much attention due to their shorter design cycle, lower cost, and higher density. The simplicity and programmability of FPGAs make them a most favorable choice for prototyping digital systems. When comparing the dynamic performance and control capabilities in PWM-controlled Power converters FPGA-based digital techniques are better than DSPs [11].

Basically a Luenberger observer (simple observer) can be used for obtaining the filtered fundamental component from the periodic output voltage and current waveforms, which leads to the indirect estimation of the total harmonics in the output voltage due to the nonlinear loads [12]. The Inverter can be modeled as a feed back control system with

the fundamental component as the desired output, while the harmonics take the role of noise creeping into the output. The well-known control strategy of using a large feed back around the noise signal was employed, to reduce the effect of noise at the output. The net effect is a smoother output voltage showing negligible total harmonic distortion, even with nonlinear loads. In this paper, an attempt has been made to show the usefulness of controlling inverters by composite observers [13, 14] rather than by using simple observers. The typical composite observer provides the pure filtered fundamental in-phase signal along with the companion quadrature signal. Alongside, the various harmonics are also estimated as instantaneous in-phase and quadrature signal pairs [13, 14]. The composite observer has a repetitive parallel structure, which can be easily implemented in an FPGA.

2. System Overview

An inverter supplying a rectifier with an RC load is shown in Figure 1. The parameters of the inverter are listed in Table 1.

The proposed control scheme can be split into two parts:

(a) use of in-phase and quadrature fundamental output voltage and current signals for conventional D-Q control,

(b) reduction of the distortion in the output voltage waveform, by feedback of voltage harmonic signals.

2.1. Design of Discrete Composite Observer. Observability means the ability to estimate the initial state from an infinite number of input-output observations. Any periodic signal $y(kT)$ rich in harmonics and a DC bias can be modeled as if $y(kT)$ emanates from a system described by

$$\begin{aligned} x((k+1)T) &= A \cdot x(kT), \\ y(kT) &= C^t \cdot x(kT), \end{aligned} \tag{1}$$

where

$$A = \begin{bmatrix} A_0 & 0 & 0 & - & 0 & - & 0 \\ 0 & A_1 & 0 & - & 0 & - & 0 \\ 0 & 0 & A_2 & - & 0 & - & 0 \\ - & - & - & - & - & - & - \\ 0 & 0 & 0 & - & A_m & - & 0 \\ - & - & - & - & - & - & - \\ 0 & 0 & 0 & - & 0 & - & A_N \end{bmatrix}; \quad A_0 = 1, \tag{2}$$

$$[C]^t = \begin{bmatrix} 1 & 1 & 0 & 1 & 0 & - & - & 1 & 0 \end{bmatrix}.$$

The typical mth subblock is given by

$$Am = \begin{bmatrix} \alpha_m & \alpha_{m-1} \\ \alpha_{m+1} & \alpha_m \end{bmatrix}, \quad \text{where } \alpha_m = \cos(m \cdot \omega_1 T) \tag{3}$$

Table 1: Inverter system parameters for simulation.

Battery Voltage V_{dc}	24 V
Dither Frequency	10 kHz
Filter Inductance L_f; R_f	1.0 mH; 1 Ω
Filter Capacitance C_f; R_c	96 μF; 0.1 Ω
Rectifier Load: Resistance R_L & Capacitance C_L	10 Ω, 1 mF
Resistive Load R_L	10 Ω

V: Volt; kHz: kilo Hertz; Ω: Ohm; mH: milli-Henry; μF: micro-Farad; mF: milli-Farad.

with state vector $x_m(kT)$ and output variable $y_m(kT)$ defined by

$$\begin{aligned} x_m(kT) &= \begin{bmatrix} x_{m1}(kT) \\ x_{m2}(kT) \end{bmatrix}, \qquad y_m(kT) = C_m^t x_m(kT), \\ C_m^t &= \begin{bmatrix} 1 & 0 \end{bmatrix} \quad \text{for } m = 1, 2, 3, \ldots, N, \ C_0 = 1. \end{aligned} \tag{4}$$

The Discrete Composite Observer is a closed loop model of the system, with an open loop part consisting of N controlled digital oscillators as sub-blocks, one for each harmonic, arranged in the parallel format along with a DC block. The mth block of the observer can be modeled with a state vector $\hat{x}_m(kT)$ and an output variable $\hat{y}_m(kT)$ defined by:

$$\begin{aligned} \hat{x}_m(kT) &= \begin{bmatrix} \hat{x}_{m1}(kT) \\ \hat{x}_{m2}(kT) \end{bmatrix}, \qquad \hat{y}_m(kT) = \hat{x}_{m1}(kT), \\ \hat{x}_m((k+1)T) &= A_m \hat{x}_m(kT) + D_m e(kT), \\ \hat{y}_m(kT) &= C_m^t \hat{x}_m(kT); \quad m = 0, 1, 2, \ldots N; \\ D_m &= \begin{bmatrix} d_{m1} \\ d_{m2} \end{bmatrix} \quad \text{for } m = 1, 2, 3, \ldots, N, \ D_0 = d_0. \end{aligned} \tag{5}$$

The observation error $e(kT)$ is defined by

$$e(kT) = y(kT) - \hat{y}(kT) = C^t[x(kT) - \hat{x}(kT)] \ = C^t E(kT), \tag{6}$$

where

$$\hat{y}(kT) = \sum_{m=0}^{m=N} \hat{y}_m(kT), \qquad E(kT) = [x(kT) - \hat{x}(kT)]. \tag{7}$$

The sum of all the individual $(N+1)$ output variables $\hat{y}_m(kT)$ shown in (7) is the scalar output of the observer.

Under steady state, $\hat{y}(kT) \rightarrow y(kT)$ as $k \rightarrow \infty$.

Each sub-block has two fixed parameters $Dm = (dm1, dm2)$ and only one tunable parameter α_m. This facilitates easy tuning of the observer under wandering frequency conditions. Only one multiplier is required rather than two required for tuning the two-dimensional sub blocks [14].

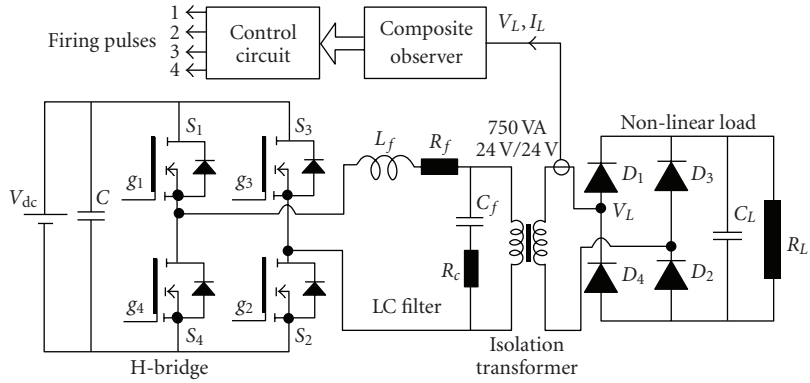


Figure 1: Single phase inverter circuit under rectifier load with RC filter.

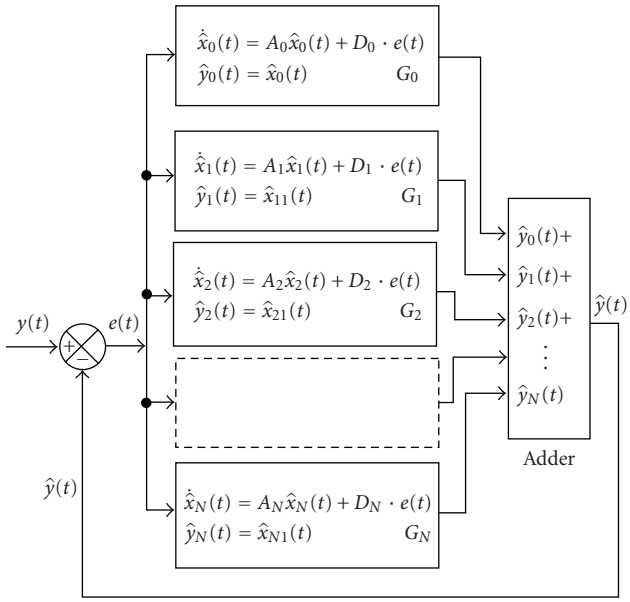


Figure 2: Structure of Discrete Composite Voltage Observer.

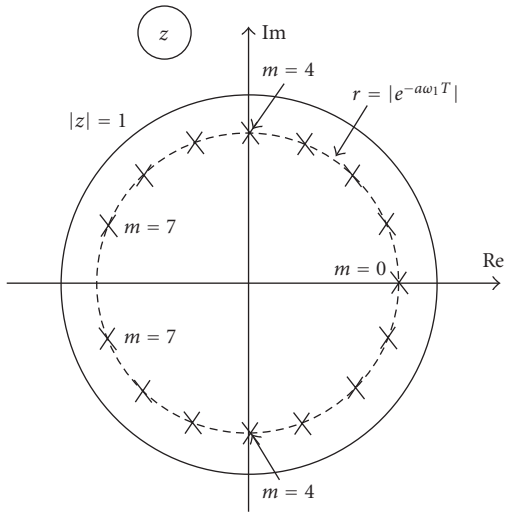


Figure 3: Poles of Discrete Observer for $(2N + 1) = 15$.

The composite observer defined so far can be concisely written as

$$\hat{x}((k+1)T) = [A]\hat{x}(kT) + [D] \cdot e(kT),$$
$$\hat{y}(kT) = C^t\hat{x}(kT), \tag{8}$$

where D the gain vector is given by

$$D = [d_0, (d_{11}, d_{12})\ (d_{21}, d_{22}) \cdots (d_{m1}, d_{m2}) \cdots (d_{N1}, d_{N2})]^t. \tag{9}$$

The $2N + 1$ observation-error-vector $[E(kT) = x(kT) - \hat{x}(kT)]$ is composed of the individual estimation errors in the DC, fundamental as well as the real and imaginary (orthogonal) harmonic components of the given signal.

Using (1), (6), and (8), we get

$$E((k+1)T) = [A - \mathrm{DC}^t]E(kT). \tag{10}$$

The characteristic equation, for the error-difference equation in (10), can be obtained in the z-domain as

$$\mathrm{Det}[zI - A + \mathrm{DC}^t] = 0, \tag{11}$$

where I is a $(2N + 1) \times (2N + 1)$ Identity matrix.

The $(2N + 1)$ roots of the characteristic equation, which are also defined as the observer poles, can be located within the unit circle in the z-plane (Figure 3). Such an "observer pole placement" makes the closed loop observer stable and the norm of the error vector E vanishes as the time tends to infinity, that is, $\|E(kT)\| \to 0$, as $k \to \infty$.

Let the $2N + 1$ closed loop poles to be equidominant and located such that $z = e^{-\delta T}$ for $m = 0$ and

$$z = e^{-\delta T}[\alpha_m \pm j\beta_m], \tag{12}$$

where $\alpha_m = \cos(m \cdot \omega_1 T)$, $\beta_m = \sin(m \cdot \omega_1 T)$ for $m = 1, 2, \ldots, N$, $\alpha_0 = 1, \beta_0 = 0$, $\delta = a \cdot \omega_1, a > 0$, which controls the observation speed, and ω_1 is the Fundamental frequency.

In the structure chosen, for a typical sub-block "m", the state variable $\hat{x}_{m1}(kT)$ will ultimately merge with the mth harmonic in the input signal. Each sub-block in the observer acts like a comb filter, accepting the signal of the

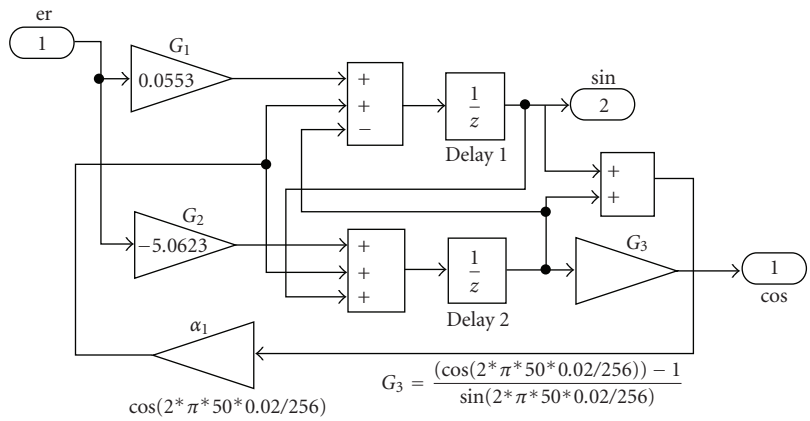


Figure 4: Structure of the Fundamental block in the Discrete Composite Voltage Observer.

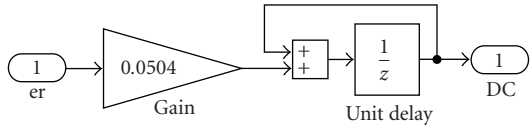


Figure 5: DC block in the Discrete Composite Voltage Observer.

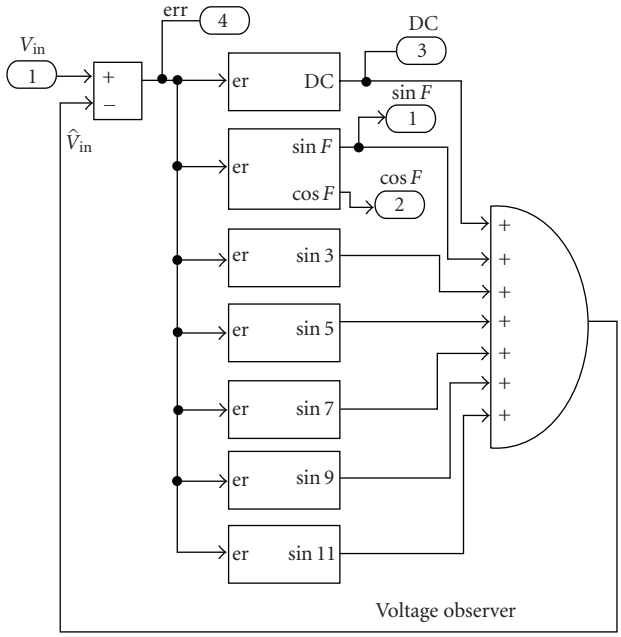


Figure 6: Structure of Discrete Composite Voltage Observer.

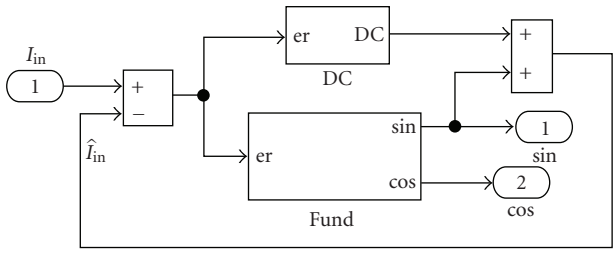


Figure 7: Structure of Current Observer.

respective tuned frequency and rejecting all other harmonic frequencies. The state variable $\hat{x}_{m2}(kT)$ will exhibit a phase-shift of 90° with respect to the signal $\hat{x}_{m1}(kT)$. Further, a magnitude scaling factor $g_m = (\alpha_m - 1)/\beta_m$ needs to be introduced for this orthogonal signal, for equalizing the amplitudes of the two signals $\hat{x}_{m1}(kT)$ and $\hat{x}_{m2}(kT)$. The observer can be tuned when the frequency of the input signal $y(kT)$ to the observer drifts. This is done by adjusting the parameter α_m using the correlation between the error signal $e(kT)$ and the fundamental quadrature signal $\hat{x}_{12}(kT)$ [14]. The fundamental block in the composite observer for $a = 1$ is shown in Figure 4. This "s" domain specification corresponds to $|z| = 0.9758$ in the digital domain (see Figure 3), for a sampling frequency of 12.8 kHz. The various harmonic blocks also have the same structure. However, the quadrature signal ("cos") need not be taken out for the harmonics, thereby reducing the number of gain elements required in the FPGA realization. The structure of the DC block is shown in Figure 5. The gains in these blocks correspond to real part of observer poles at $a = 1$. The composite observer used for voltage signal processing is shown in Figure 6.

The current observer can also be made to have the same structure as the voltage observer. However, to reduce the number of DSP elements required, only the DC block and the fundamental block were used for the current observer as shown in Figure 7.

2.2. D-Q Control Using Fundamental Components. In a 2-phase system the steady-state errors could be overcome by introducing DC reference-commands and the corresponding DC variables for representing the sinusoidal trajectories. When DC variables converge to constant values under steady-state, it is easier to make the steady-state error zero by including a conventional PI controller in the control loop.

Using a discrete composite voltage observer, having DC and odd harmonic blocks up to 11th $(0, 1, 3, \ldots, 11)$, the in-phase (sine) and the fictitious quadrature (cosine) fundamental voltage signals of the inverter can be derived. Applying the Park transformation, in a manner which is

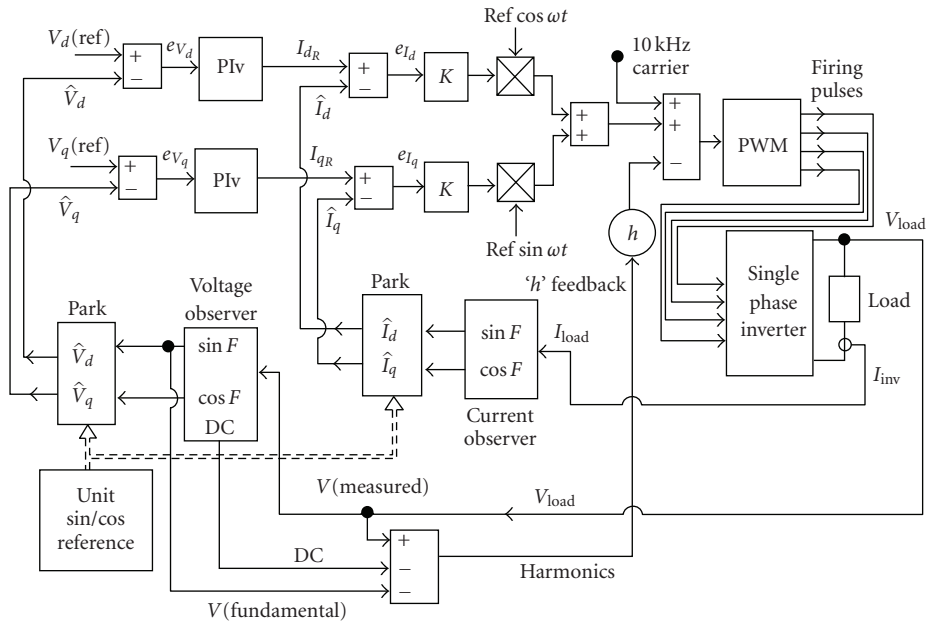


Figure 8: Schematic of the Proposed Control Strategy for the inverter.

analogous to 2-phase systems, these components can be transformed to "D-Q" coordinates. The Park transform requires unit sine and cosine reference signals which are generated internally for the stand-alone inverters:

$$\begin{bmatrix} V_d \\ V_q \end{bmatrix} = \begin{bmatrix} \sin(\omega_1 t) & \cos(\omega_1 t) \\ \cos(\omega_1 t) & -\sin(\omega_1 t) \end{bmatrix} \cdot \begin{bmatrix} \sin(\omega_1 t + \varphi_2) \\ \cos(\omega_1 t + \varphi_2) \end{bmatrix}. \quad (13)$$

Ideally, the load voltage is expected to contain only the fundamental components, whereas all the other harmonic components including random noise in the voltage loop must be zero. Now, the estimation speed of the observer depends on the pole-location. In the continuous domain, observer poles closer to the origin of the s-plane make the estimation sluggish, while the poles located farther from the origin of the s-plane make the observer faster. On the other hand, for a digital version of the observer, placing the poles closer to the origin in the "z" plane can speed up the estimation while placing the poles near to the unit circle makes it sluggish. In any case, very high-speed observers show a heavy distortion in the extracted fundamental and other components. Placing the observer poles closer to the unit circle in the z-domain (closer to origin of the s-plane) makes the extracted fundamental component pure and accurate, at the cost of increased time for estimation, which may even lead to instability of the voltage control system. The steady-state error-vector between the desired d-q components and the actual d-q components obtained from the composite voltage-observer is processed through PI controllers to obtain d-q current references. In the simulation and experimental studies presented in this paper, the decay factor "a" for voltage observer and current observer were set at 1.

The control scheme for reducing harmonics in the inverter is shown in Figure 8. The steady-state error-vector $[e_{vd}, e_{vq}]$ between the desired D-Q component values and the actual D-Q component values obtained from the voltage-observer is processed through PI controllers to obtain D-Q current references. Similarly, using another simple observer, the fundamental in-phase and fictitious quadrature current signals are estimated. This facilitates transformation of the current to the D-Q frame. The current error-vector $[e_{Id}, e_{Iq}]$ in the D-Q frame is processed by simple gain elements (k) even though PI or lead type compensators could have been used. The steady outputs of the two controllers in the current loop are converted into the in-phase and quadrature time signals, via the single-phase-inverse Park transform. The two time signals are added to get the sinusoidal reference signal for pulse width modulation as shown in Figure 8. Without any compensation for harmonics, the output voltage is heavily distorted for a "nonlinear load", consisting of a rectifier driving a RC load ($10\,\Omega \| 1\,\text{mF}$). The THDv is about 15.34% even though the control signal appears to be a pure sine wave. The distorted output voltage, pulsed load current, and the control voltage are shown in Figure 9. The distortion is mainly due to the voltage drop caused by the harmonic currents in the series R-L filter of the inverter.

2.3. Series Compensation by Harmonic Feedback. Let us consider the harmonics creeping into the output due to the voltage drop in the inductor as shown in Figure 10(a). The sum of all the harmonics can be assumed to be equivalent to a noise signal, superimposed on the fundamental component. In Figure 10(b), both the noise and the signal have unity gain. If a control loop could be built around the noise signal, with a high feedback gain "h" for the noise alone, as shown in

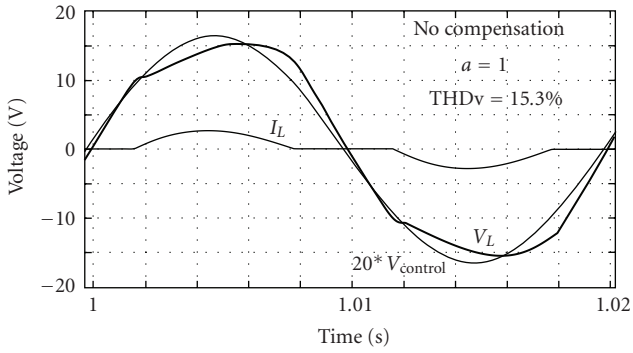


Figure 9: Distorted output voltage, current, and control signal with no harmonic feedback.

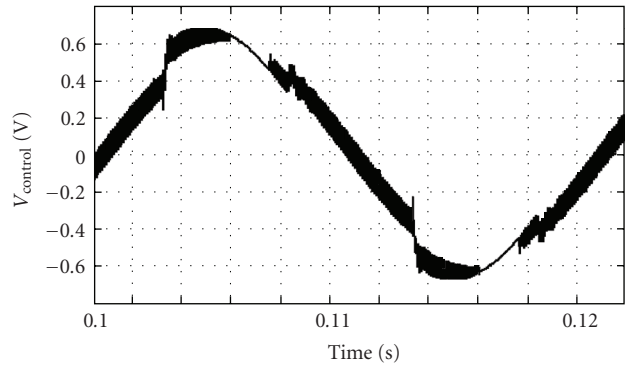


Figure 11: Control signal which yields a pure sine Output.

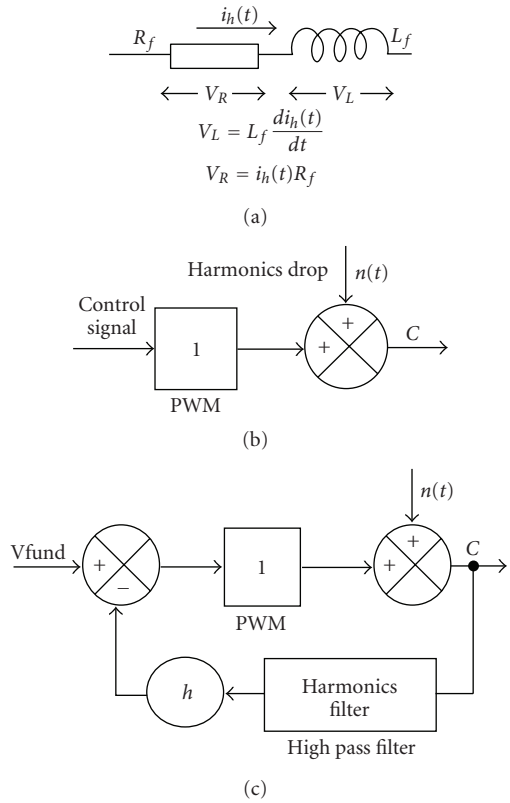


Figure 10: (a) Voltage drop across the inductor. (b) Equivalence of harmonic distortion to noise. (c) Attenuation of noise in closed loop.

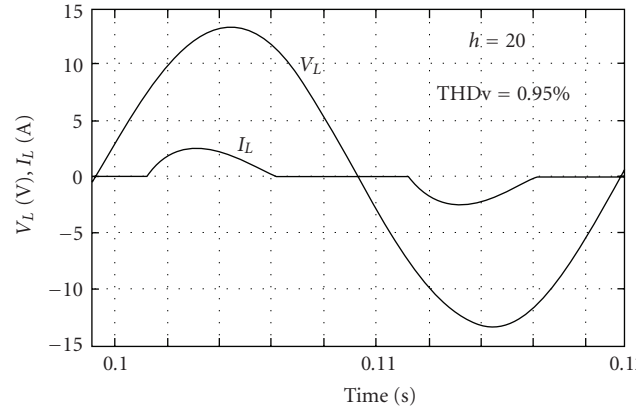


Figure 12: Load voltage signal after harmonic feed-back while supplying a current to a nonlinear load.

Figure 10(c), its effect in the output can be made negligible. This requires a highpass filter, which can be realized from the composite voltage observer, designed for extracting the various harmonics.

The high-frequency harmonic signal, which is devoid of the fundamental component and DC, can be fed back with a high gain "h" for reducing the distortion as shown in Figure 8. While the fundamental gain remains unaffected at unity, the closed loop gain (K_{nf}) for the harmonics becomes smaller:

$$K_{nf} = \frac{1}{(1+h)}. \tag{14}$$

For example, when $h = 20$, the effect of noise with this local loop reduces to $K_{nf}/1.0\sim = 5\%$.

Interestingly, due to harmonic feedback, the control signal becomes enriched with harmonics, as shown in Figure 11. In contrast, the output voltage waveform becomes a purer sine wave as shown in Figure 12.

2.4. Model of the Proposed Control System. A model for the fundamental components, as shown in Figure 13, of the inverter control system is obtained on the basis of the following assumptions. Since the loads may be nonlinear, the steady-state fundamental components of the voltage and current signals alone are taken into account for modeling. The inverter is assumed to drive near-unity power factor loads, which makes the system somewhat decoupled as far as the direct and the quadrature channels are concerned. So, we need to consider either the direct axis channel or the quadrature axis channel for analysis. The pulse width

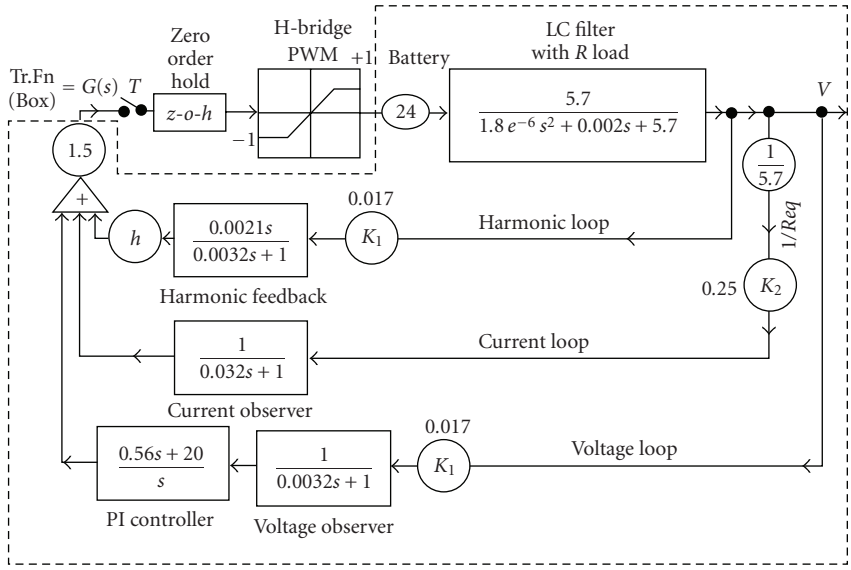


Figure 13: Block diagram for any one channel (V_d or V_q).

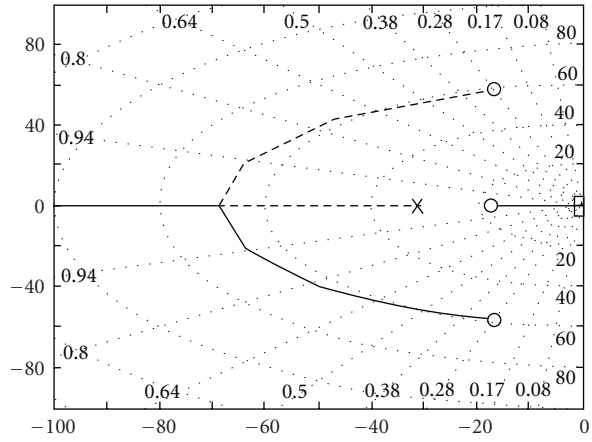


Figure 14: Root Locus for determining integral constant of the PI controller.

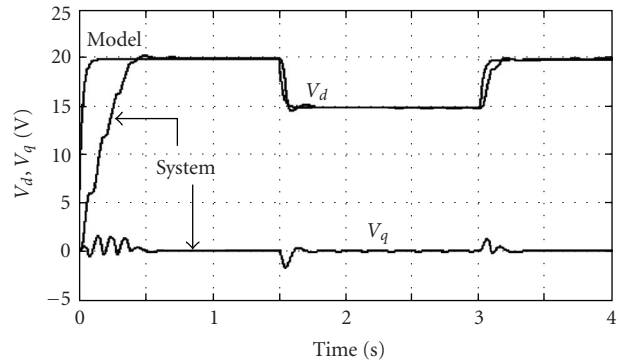


Figure 15: Validation of Model for a step disturbance in V_{dref}.

modulating (PWM) block is modeled as a unity-gain ($k \cong 1$) saturation-block, which feeds into a linear gain, of value = 24, the battery voltage. The input saturation level of PWM block is the peak of the carrier signal. The LC filter with resistive load can also be included in the model. The observer is equivalent to a lowpass filter in the d or q domains. A simple proportional controller with a fixed gain setting (say 1.5) is sufficient for the current loop because the PI controller in the voltage-loop will guarantee the accuracy of the amplitude of the output voltage, in the linear operating region. The real part of the current-observer pole was placed at -0.1ω. This is slow enough to filter out the harmonics of the current under nonlinear loads like a rectifier with RC filter. The real part of the voltage-observer pole is 10 times faster than the current-observer pole and is placed at -1ω. The slower current loop causes reduced phase margin and increased overshoot under transient conditions. The stability of the total control system depends mainly on the feedback factor "h", observer pole location, and the PI controller settings of the voltage loop. While some of the parameters listed above can be intuitively assigned, the major problem remaining is the design of the PI controller so that the control system would remain stable.

Sketching the root locus and enlarging it around the origin (Figure 14) reveal one real pole and a complex closed loop pole pair being dominant. By adjusting the zero of the PI controller, all these three poles can be made to have the same negative real part or remain equally dominant as the closed loop gain tends to infinity. For Req = 5.7 Ω, h = 20, the settings are Kp = 0.56 and Ki = 81. It is appropriate to carry out the stability analysis in the z-domain.

2.5. Validation of Model. For a step change in the reference V_d, the response of the model and system is shown in Figure 15. The system and the model initial conditions were different. However after convergence, the incremental responses of the system and the model coincide reasonably well.

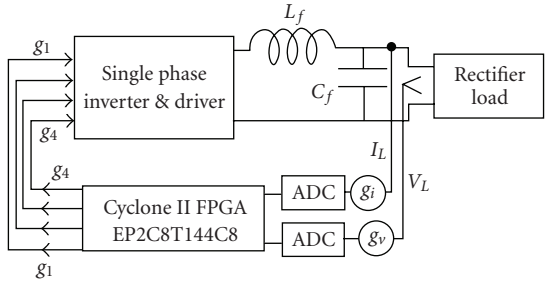


Figure 16: Overall Hardware implementation circuit of the inverter.

3. FPGA Implementation Schematic of the Controller

The overall hardware implementation circuit of the single phase inverter is shown in Figure 16. The measured output voltage and current are normalized to unity using g_v and g_i, which include voltage divider as well as the analog amplifier gains. Inside the FPGA the maximum voltage and current variables are made equal to unity. Hence the same setup could be easily extended to any voltage or current levels, by suitably adjusting the external range setting devices.

The implementation consists of two separate ADCs for feeding in the load voltage and current into the control circuit as shown in Figure 17. A serial +5V, 12-bit, MSOP-8 package analog-to-digital converter ADS7835 was interfaced with Cyclone II FPGA. Handshaking between FPGA and ADS7835 is through 3 signals named Convert, Clock, and Data. The Convert and Clock are input signals to ADC from FPGA, while the converted Data goes into the FPGA. Since the level of the digital I/O signals for the FPGA is at 3.3 V and the ADC operates at 5 V, a digital buffer (16 pin 74HC366) has been provided for the two channels. The serial data from ADS7835 has been converted into 12-bit parallel data in the integer format in the FPGA, which is shown in Figure 17. Since the integer signals are to be normalized to 18-bits floating point, that is, [3 : 15], further conversion to floating point is necessary which is illustrated in Figure 18. Bit-by-bit extraction and addition in the floating point has been carried out for achieving this. These two procedures were incorporated as subsystems.

The measured current and the voltage variables available in the floating point format are fed to separate observers, one for voltage and the other for the current as shown in Figure 19. The basic unit sinusoidal signal is generated by using a look up table and an address-generating counter, triggered by the same pulses, which drive the observers. The arrangement is shown in Figure 20. A 50 Hz unit sine wave is generated, by creating 256 data entries, obtained by sampling off-line, 256 times a full unit sine wave. The LUT which holds the data in the [3 : 15] floating format has an 8-bit address space, which can be accessed by an 8-bit address counter. The 12.8 kHz enabling pulses for the address counter are derived from a mod 313 counter, whose clock ticks at 4 MHz. The unit sine wave from the LUT is fed to an observer of the structure shown in Figure 4, so as to get the unit sine and cosine reference signals as shown in Figure 20. The extracted in-phase and quadrature fundamental signals from the voltage or current observer and the lookup table generated unit sine and cosine waves are used to derive the D-Q components of load voltage or current through the Park transform. A typical Park Transform block is shown in Figure 21. The references V_{dref} and V_{qref} are set as required in floating point format. The V_d and V_q feedback signals are derived from the voltage observer, after Park transformation as explained earlier. The errors in V_d and V_q are processed through separate PI controllers. The outputs of these PI controllers are taken to be the I_{dref} and I_{qref} signals for inner current loop. The current error signals are processed by proportional controllers and further converted into sinusoidal control signal via single-phase-inverse park transform as shown in Figure 22.

Along with this control signal, voltage harmonics are fed back with a large gain and then filtered to remove very high-frequency noise corrupting the control signal. A 10 kHz triangular carrier (dither) signal is used for the generation of sinusoidal PWM pulses required by the power MOSFETs of the inverter. The schematic is shown in Figure 23.

4. Experimental Results

Using the Altera DSP Builder, a digital version of the control scheme was designed and implemented in a Cyclone II (Altera) FPGA. The load voltage and current corresponding to a nonlinear load, with a harmonic feedback gain in the range $0 < h < 20$, were recorded, for validating the simulations. Figure 24(a) shows the heavily distorted load voltage (THDv 16.4%) for a nonlinear current since no harmonic feedback compensation was provided. Similarly even when a composite voltage observer is used along with a simple current observer, as long as no harmonic feedback compensation is provided, the voltage waveform will be distorted (THDv 15.29%) for a nonlinear load (Figure 24(b)). In Figure 25(a), the harmonic feedback gain "h" was set at 20, while supplying a rectifier load of $10\,\Omega \| 1\,\text{mF}$. It is seen that even a simple voltage observer gives a pure sine wave output, showing a THD of 1.2% When a composite voltage observer is used along with a simple current observer, the voltage waveform is still purer showing a THD of 0.98% (Figure 25(b)). Even though this result is attractive, very large harmonic feedback gains make the control system sluggish and reduce the stability margin appreciably. From Table 2, it is clear that a harmonic feedback gain of 5 is sufficient to get a THD of 4.67%.

5. Resource Utilization in FPGA

The realization of the control schemes has been done using DSP builder software in MATLAB simulink environment. DSP Builder tool provides a seamless design flow of algorithmic design and system-level integration in MATLAB and Simulink software and then ports the design to HDL for use in Altera's Quartus II design software. The DSP Builder

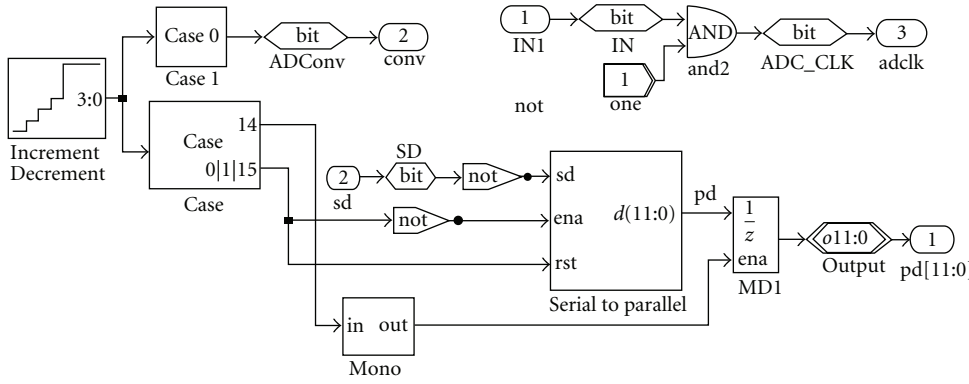


Figure 17: ADC Program for ADS 7835 in DSP Builder.

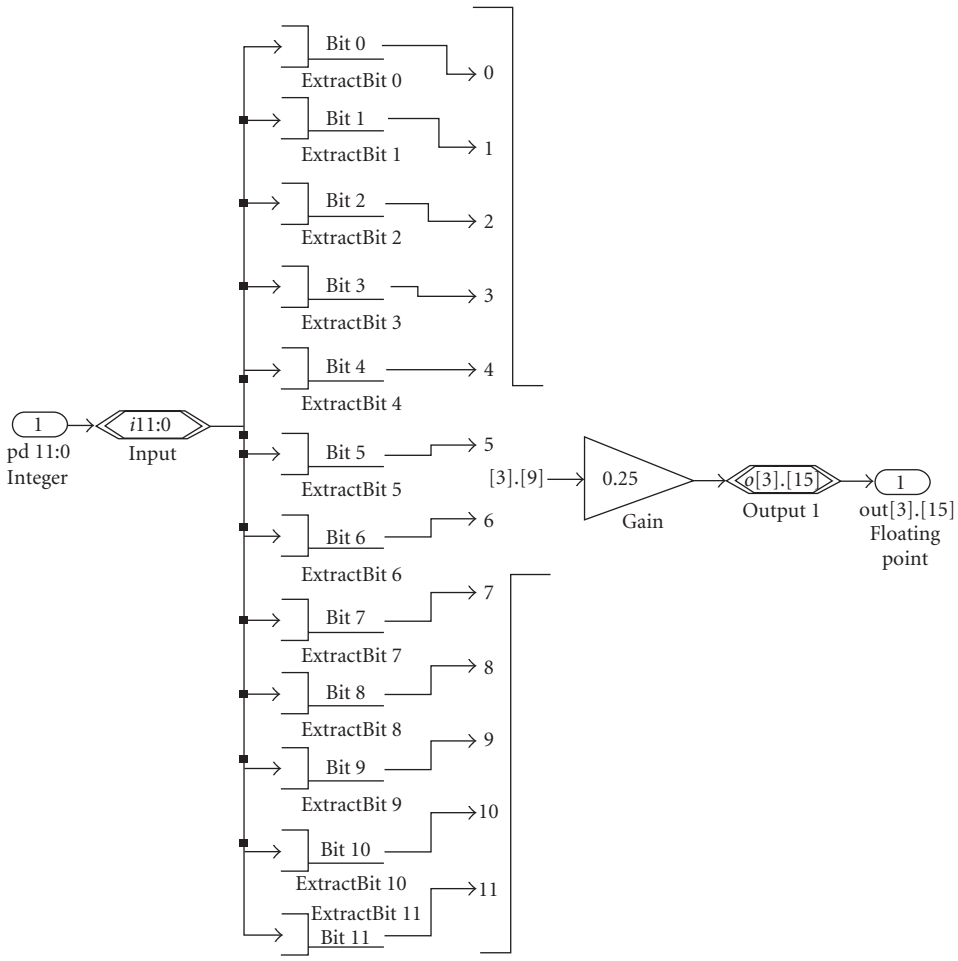


Figure 18: Integer to floating point block.

can automatically generate preverified RTL output files—including an RTL design and testbench—from the Simulink software. The DSP Builder design flow for Altera FPGA is shown in Figure 26. Quartus II software has been used for fitting the synthesised algorithm into the FPGA. The resource utilization summary for both the control schemes has been shown in Table 3. The total logic elements needed to implement the composite observer scheme using Cyclone II is only about 66%. Table 4 indicates the resources utilized by both the observers in Cyclone II.

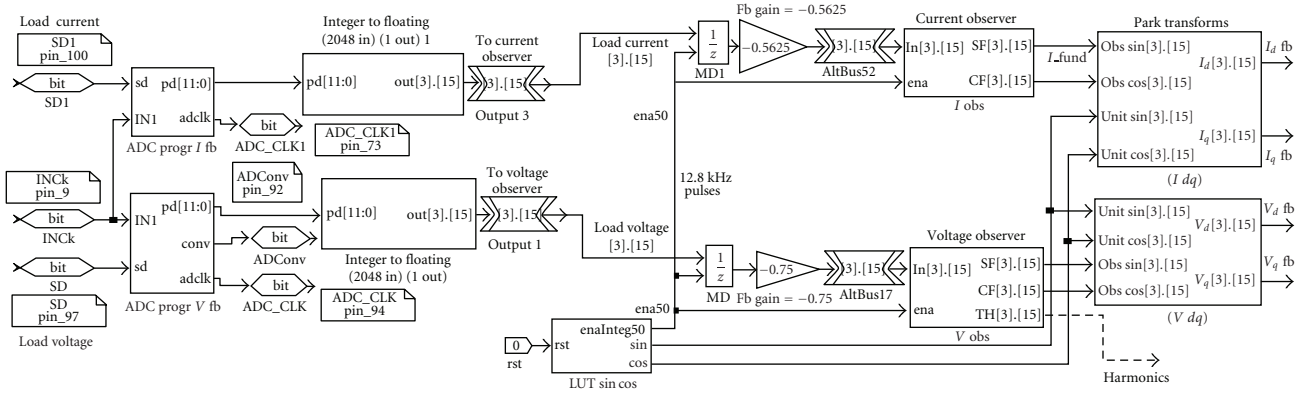


Figure 19: Generation of D-Q Voltage and Current Signals using the Output of the Inverter.

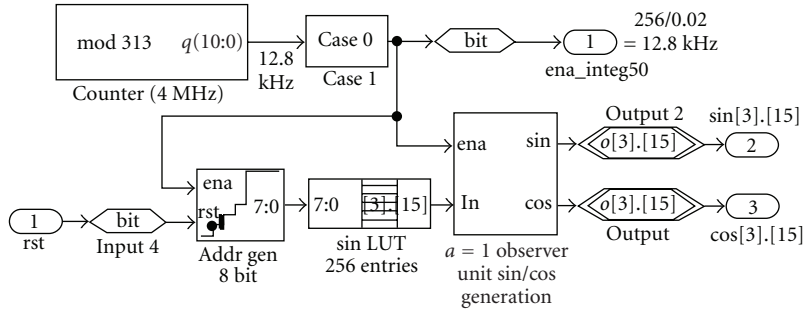


Figure 20: Internal Generation of Unit sine and cosine signals.

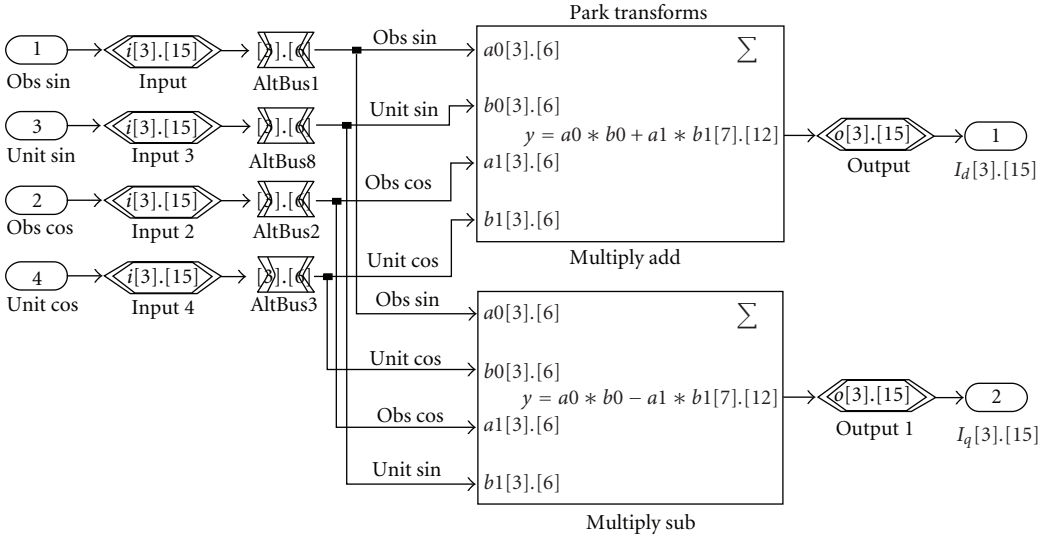


Figure 21: Computation of D-Q components (Park Transform).

Table 2: % THDv of the inverter output voltage with the two different schemes.

h	% THDv			
	Composite voltage observer and Simple current observer		Simple voltage and current observers	
	Simulation	Expt.	Simulation	Expt.
0	15.34	15.29	15.94	16.40
5	3.309	4.67	3.515	4.79
10	1.773	2.07	1.927	2.91
15	1.197	1.71	1.321	1.99
20	0.903	0.97	1.01	1.20

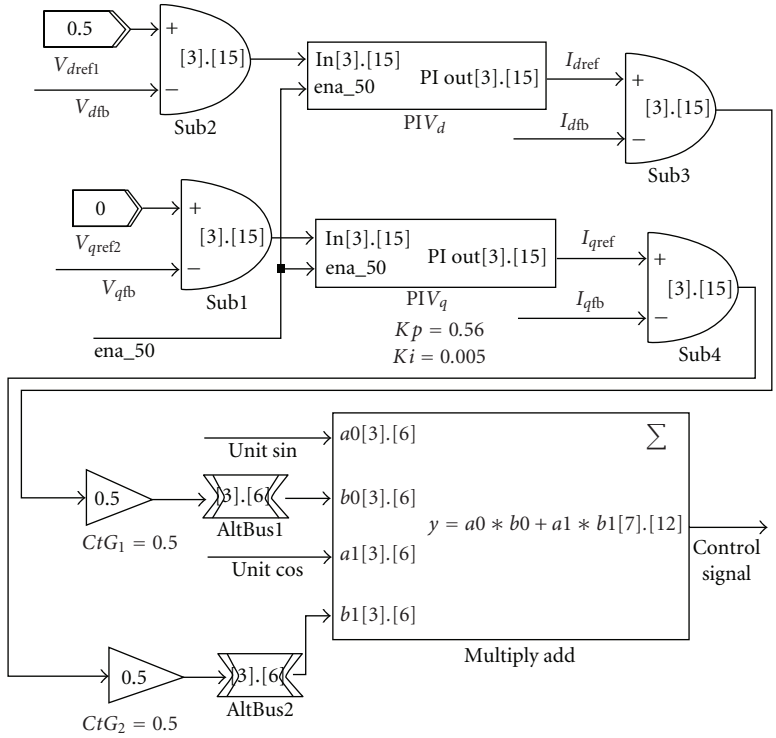


Figure 22: Fundamental Control signal generation via single-phase-inverse Park transform.

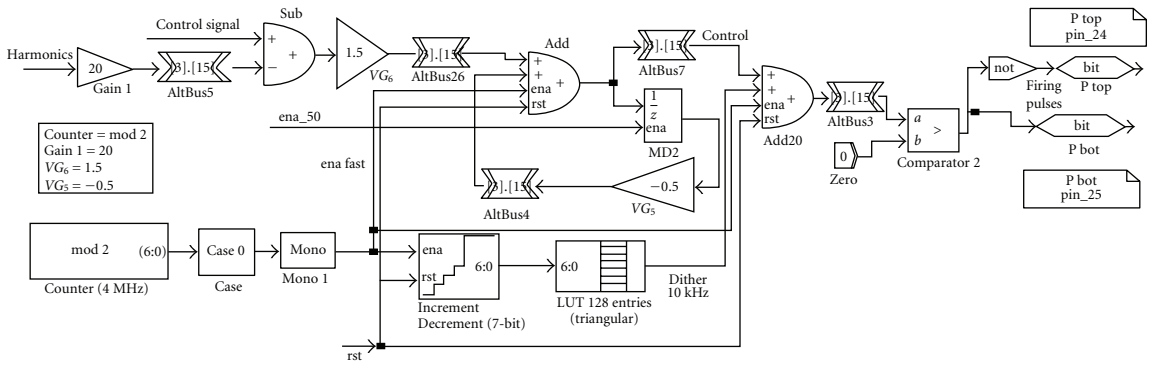


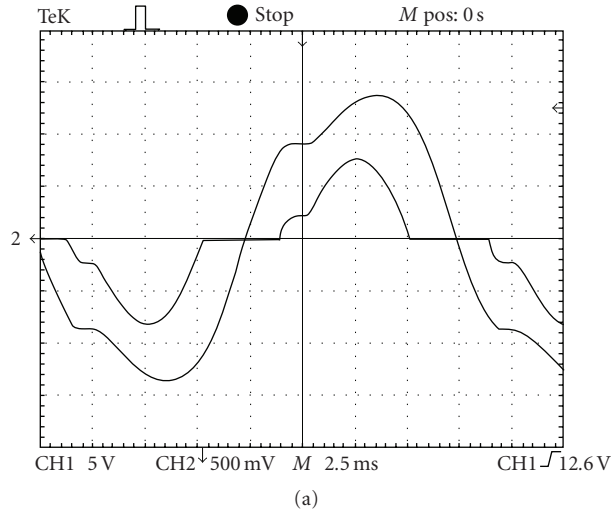
Figure 23: PWM pulse generation from the Control Signal.

Table 3: Resource utilization of the Simple Observer (SO) and Composite Observer (CO) Control scheme using Quartus II tool.

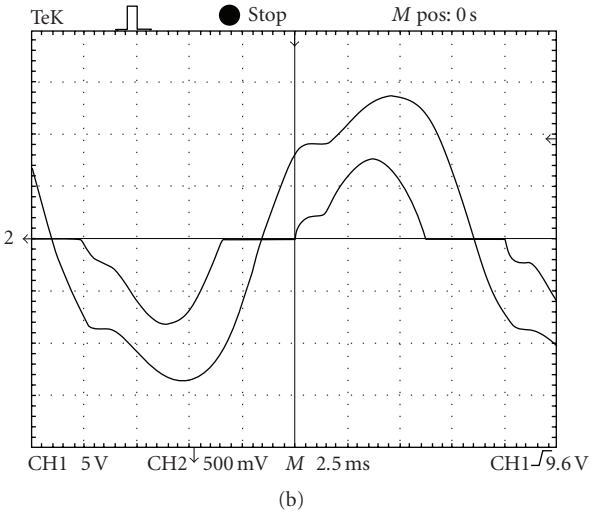
	SO	CO
Family	Cyclone II	Cyclone II
Device	EP2C8T144C8	EP2C8T144C8
Total logic elements	1,779/8,256 (21%)	5,517/8,256 (66%)
Combinational with no register	1448	4981
Register only	16	13
Combinational with a register	315	523
Logic element usage by number of LUT inputs		
4 input functions	215	662
3 input functions	1207	3967
<=2 input functions	341	875
Register only	16	13
Combinational cells for routing	13	32
Logic elements by mode		
Normal mode	574	1480
Arithmetic mode	1189	4024
Total registers	331/8,256 (4%)	536/8,256 (6%)
Total LABs	158/516(30%)	452/516 (87%)
Total pins	9/85 (10%)	9/85 (10%)
M4Ks	2/36 (5%)	2/36 (5%)
Total memory bits	5,120/165,888 (3%)	5,120/165,888 (3%)
Total RAM block bits	9,216/165,888 (5%)	9,216/165,888 (5%)
Embedded Multiplier 9-bit elements	36/36 (100%)	36/36 (100%)

Table 4: Resource utilization of the Simple Observer (SO) and Composite Observer (CO) using Quartus II tool.

	SO	CO
Family	Cyclone II	Cyclone II
Device	EP2C8T144C8	EP2C8T144C8
Total logic elements	265/8,256 (3%)	2,431/8,256 (29%)
Combinational with no register		
Combinational with a register	206	2167
Logic element usage by number of LUT inputs	59	264
4 input functions	0	6
3 input functions	240	2053
<=2 input functions	25	372
Combinational cells for routing	0	19
Logic elements by mode		
Normal mode	14	299
Arithmetic mode	251	2132
Total registers	59/8,256 (<1%)	264/8,256 (3%)
Total LABs	24/516 (4%)	207/516 (40%)
Embedded Multiplier 9-bit elements	16/36 (44%)	36/36 (100%)

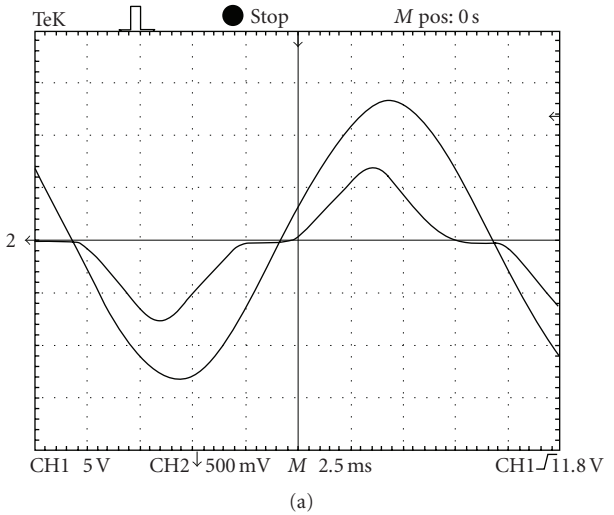


(a)

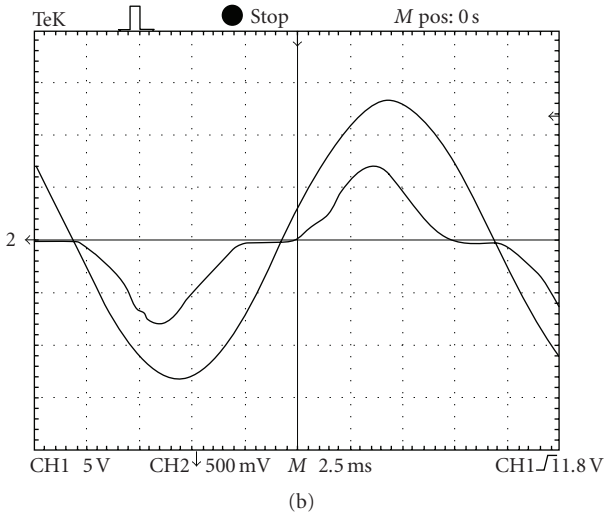


(b)

Figure 24: (a) Distorted output voltage waveform when a simple Voltage Observer and a simple Current Observer are used. (b) Distorted output voltage waveform when a Composite Voltage Observer and a simple Current Observer are used.



(a)



(b)

Figure 25: (a) Pure output voltage waveform when a Simple Voltage Observer and a simple Current Observer are used. (b) Pure output voltage waveform when a Composite Voltage Observer and a simple Current Observer are used.

6. Conclusion

The reduction of distortion in the voltage waveform in single phase Inverters, caused by the currents drawn by nonlinear loads, can be easily done by feeding back the instantaneous harmonic content of the voltage signal. This harmonic content is computed on-line by a composite observer in the voltage loop, which incidentally can also be used for obtaining the fundamental voltage signal for keeping the output voltage at the desired level. A simple Luenberger observer in the inner current loop serves the purpose of estimating the fundamental component of the load current, which is used for enhancing the stability of the overall scheme. Since the traditional trajectory control systems give rise to a steady-state error, the D-Q control procedure which is popular in 3-phase Inverters has been employed after suitable modifications for accurate voltage control of single-phase inverters. The local harmonic feedback gives rise to an enormous improvement in the quality of the waveform.

Interestingly the individual subblocks described in the work can be easily implemented in a Cyclone II FPGA, after trials in the Simulink environment. The composite observer has a parallel structure, which makes it desirable for implementation via the FPGA route. The digital multiply and add blocks were used for implementing the Park transform blocks. The various signals are almost simultaneously processed as if in an analog circuit, enabling a larger bandwidth for the control scheme. The total number of multipliers,

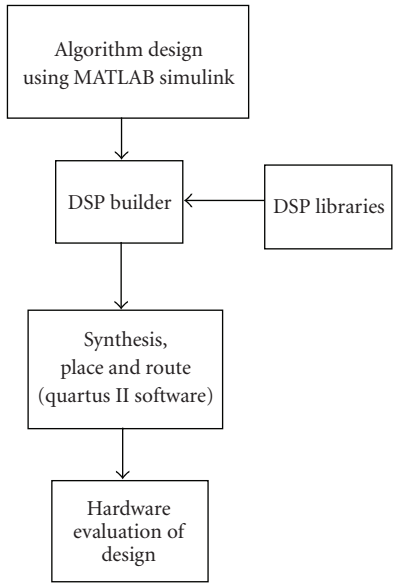


Figure 26: DSP Builder Design flow for Altera FPGAs.

adders, counters, integrators, and lookup tables was sufficient for implementing the inverter control system along with harmonic compensation procedure. Essentially, the harmonic distortion was cancelled by generating an equivalent opposing signal and adding it to the conventionally generated control signal within the FPGA. This procedure may be thought of as software-series compensation, which eliminates the need for external series transformers. It was found in the simulation and experimental studies that the composite observer shows a slightly faster response and purer output voltage waveform than a basic (simple) observer for the same feedback gain "h". Since the scheme works on the principle of "series software harmonic compensation," which is effectively carried out at the input terminal of the PWM block, a reduction in the maximum available sinusoidal peak voltage results. This is not a serious deficiency, since the output transformer turns ratio can be slightly increased, even at the design stage. Most importantly, the entire scheme could be embedded into a single cyclone FPGA Chip without any additional RAM elements. Some left over area in the FPGA can be used for protection of the Inverter under fault conditions.

References

[1] K. S. Low, K. L. Zhou, and D. W. Wang, "Digital odd harmonic repetitive control of a single-phase PWM inverter," in *Proceedings of the 30th Annual Conference of IEEE Industrial Electronics Society (IECON '04)*, pp. 6–11, Busan, South Korea, November 2004.

[2] C. Rech, H. Pinheiro, H. A. Gründling, H. L. Hey, and J. R. Pinheiro, "A modified discrete control law for UPS applications," *IEEE Transactions on Power Electronics*, vol. 18, no. 5, pp. 1138–1145, 2003.

[3] T.-L. Tai and J.-S. Chen, "UPS inverter design using discrete-time sliding-mode control scheme," *IEEE Transactions on Industrial Electronics*, vol. 49, no. 1, pp. 67–75, 2002.

[4] Z. Kai, K. Yong, X. Jian, and C. Jian, "Deadbeat control of PWM inverter with repetitive disturbance prediction," in *Proceedings of the 14th Annual Applied Power Electronics Conference and Exposition (APEC '99)*, vol. 2, pp. 1026–1031, Dallas, Tex, USA, March 1999.

[5] W. Guo, S. Duan, and Y. Kang, "A new digital multiple feedback control strategy for single-phase voltage-source PWM inverters," in *Proceedings of the 10th IEEE Region International Conference on Electrical and Electronic Technology*, vol. 2, pp. 809–813, Singapore, August 2001.

[6] P. A. Dahono and E. Taryana, "A new control method for single-phase PWM inverters to realize zero steady-state error and fast response," in *Proceedings of the 5th International Conference on Power Electronics and Drive Systems (PEDS '03)*, vol. 2, pp. 888–892, November 2003.

[7] Y. T. Woo and Y. C. Kim, "A digital control of a single-phase UPS inverter for robust AC-voltage tracking," in *Proceedings of the 30th Annual Conference of IEEE Industrial Electronics Society (IECON '04)*, vol. 2, pp. 1623–1628, Busan, South Korea, November 2004.

[8] N. M. Abdel-Rahim and J. E. Quaicoe, "Multiple feedback loop control strategy for single-phase voltage-source UPS inverter," in *Proceedings of the 25th Annual IEEE Power Electronics Specialists Conference (PESC '94)*, vol. 2, pp. 958–964, Taipei, Taiwan, June1994.

[9] N. M. Abdel-Rahim and J. E. Quaicoe, "Analysis and design of a multiple feedback loop control strategy for single-phase voltage-source UPS inverters," *IEEE Transactions on Power Electronics*, vol. 11, no. 4, pp. 532–541, 1996.

[10] M. A. Boost and P. D. Ziogas, "State of the art PWM techniques: a critical evaluation," *IEEE Transactions on Industry Applications*, vol. 24, no. 2, pp. 271–280, 1988.

[11] A. Fratta, G. Griffero, and S. Nieddu, "Comparative analysis among DSP and FPGA-based control capabilities in PWM power converters," in *Proceedings of the 30th Annual Conference of the IEEE Industrial Electronics Society (IECON '04)*, pp. 257–262, Novmeber 2004.

[12] K. Selvajyothi and P. A. Janakiraman, "Implementation of a control strategy for elimination of voltage harmonics in inverters," in *Proceedings of the 51st IEEE Midwest Symposium on Circuits and Systems (MWSCAS '08)*, pp. 13–16, Knoxville, Tenn, USA, August 2008.

[13] K. Selvajyothi and P. A. Janakiraman, "Analysis and simulation of single phase composite observer for harmonics extraction," in *Proceedings of the International Conference on Power Electronics, Drives and Energy Systems (PEDES '06)*, New Delhi, India, December 2006.

[14] K. Selvajyothi and P. A. Janakiraman, "Extraction of harmonics using composite observers," *IEEE Transactions on Power Delivery*, vol. 23, no. 1, pp. 31–40, 2008.

Hindawi Publishing Corporation
VLSI Design
Volume 2010, Article ID 251210, 25 pages
doi:10.1155/2010/251210

Research Article

Evolvable Block-Based Neural Network Design for Applications in Dynamic Environments

Saumil G. Merchant[1] and Gregory D. Peterson[2]

[1] *Department of Electrical and Computer Engineering, George Washington University, 20101 Academic Way, Ashburn, VA 20147-2604, USA*
[2] *Department of Electrical Engineering and Computer Science, University of Tennessee, 414 Ferris Hall, Knoxville, TN 37996-2100, USA*

Correspondence should be addressed to Saumil G. Merchant, smerchan@gwu.edu

Received 7 June 2009; Accepted 2 November 2009

Academic Editor: Ethan Farquhar

Dedicated hardware implementations of artificial neural networks promise to provide faster, lower-power operation when compared to software implementations executing on microprocessors, but rarely do these implementations have the flexibility to adapt and train online under dynamic conditions. A typical design process for artificial neural networks involves offline training using software simulations and synthesis and hardware implementation of the obtained network offline. This paper presents a design of block-based neural networks (BbNNs) on FPGAs capable of dynamic adaptation and online training. Specifically the network structure and the internal parameters, the two pieces of the multiparametric evolution of the BbNNs, can be adapted intrinsically, in-field under the control of the training algorithm. This ability enables deployment of the platform in dynamic environments, thereby significantly expanding the range of target applications, deployment lifetimes, and system reliability. The potential and functionality of the platform are demonstrated using several case studies.

1. Introduction

Artificial Neural Networks (ANNs) are popular among the machine intelligence community and have been widely applied to problems such as classification, prediction, and approximation. These are fully or partially interconnected networks of computational elements called artificial neurons. An artificial neuron is the basic processing unit of the ANN that computes an output function of the weighted sum of its inputs and a bias. Depending on the interconnection topology and the dataflow through the network, ANNs can be classified as feedforward or recurrent. The design process with ANNs involves a training phase during which the network structure and synaptic weights are iteratively tuned under the control of a training algorithm to identify and learn the characteristics of the training data patterns. The obtained network is then tested on previously unseen data. ANNs are effective at identifying and learning nonlinear relationships between input and output data patterns and have been successfully applied in diverse application areas such as signal processing, data mining, and finance.

Explicit computational parallelism in these networks has produced significant interest in accelerating their execution using custom neural hardware circuitry implemented on technologies such as application specific integrated circuits (ASICs) and field programmable gate arrays (FPGAs). The design process for ANNs typically involves offline training using software simulations and the obtained network is designed and deployed offline. Hence the training and design processes have to be repeated offline for every new application of ANNs. This paper is an extended version of [1] and presents an FPGA design of block-based neural networks (BbNNs) that can be adapted and trained online, on-chip under the control of an evolutionary algorithm. On-chip evolution under the control of evolutionary algorithms is called intrinsic evolution in the evolvable hardware community [2]. The capability of intrinsic evolution expands the potential applications of these networks to dynamic

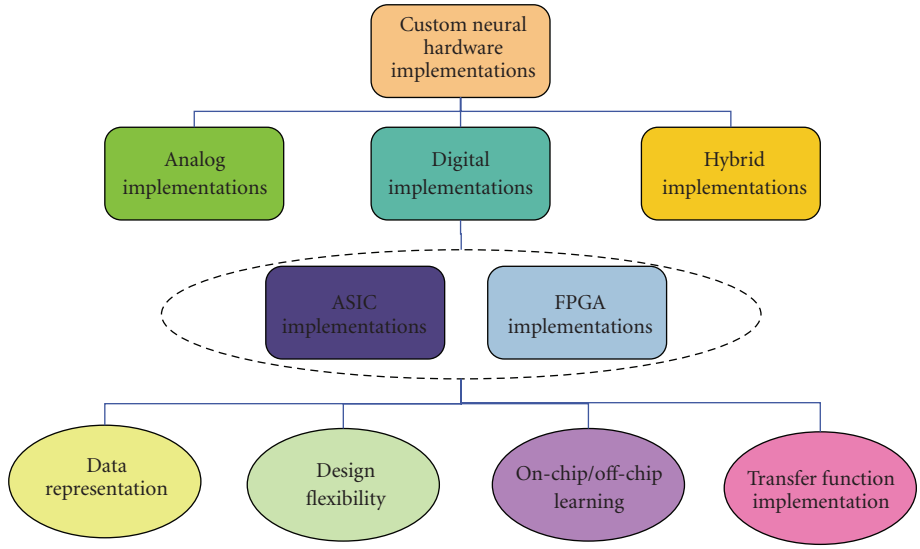


Figure 1: Neural network hardware classification.

environments and provides increased deployment lifetimes and improved system reliability. The paper also provides a detailed survey of reported ANN hardware implementations.

The rest of the paper is organized as follows. Section 2 provides a review of custom ANN implementations reported in the literature. Section 3 introduces block-based neural networks and their training procedure. Section 4 gives the details on the FPGA design of the evolvable BbNN implementation. Section 5 demonstrates the capabilities of the design using several case studies and Section 6 concludes the paper.

2. Review of ANN Implementations

There has been significant interest in custom ANN implementations and many have been reported in the literature over the years. Dedicated hardware units for artificial neural networks are called neurochips or neurocomputers [3]. Due to limited commercial prospects and the required development and support resources, these chips have seen little commercial viability. Also, due to the existence of wide-ranging neural network architectures and a lack of a complete and comprehensive theoretical understanding of their capabilities, most commercial neurocomputer designs are dedicated implementations of popular networks such as multilayer perceptrons (MLPs), hopfield networks, or kohonen networks targeting particular applications. Several classification and overview studies of neural hardware have appeared in the literature [3–13]. Heemskerk has a detailed review of neural hardware implementations until about 1995 [3]. He classified the neural hardware according to their implementation technologies such as the neurocomputers built using general purpose processors, digital signal processors, or custom implementations using analog, digital, or mixed-signal design. Zhu et al. have surveyed ANN FPGA implementations, classifying them based on design features such as precision and flexibility [13]. The review presented here follows and extends these surveys. Figure 1 shows the classification structure used in this review. Broadly, the implementations have been classified into digital, analog, and hybrid implementations. The digital hardware implementations have been grouped into FPGA and ASIC implementations and classified according to design features such as data representation, design flexibility, on-chip/off-chip learning, and transfer function implementation. A brief overview of analog and hybrid implementations is also offered. A detailed survey of these is beyond the scope of this work.

2.1. Digital Neural Network Implementations. Digital neural network implementations offer high computational precision, reliability, and programmability. They are targeted to either ASICs or FPGAs. The synaptic weights and biases for these implementations are typically stored in digital memories or registers, either on- or off-chip dictated by the design tradeoffs between the speed and the circuit size. ASIC neurochips can achieve higher processing speeds, lower power, and more density than corresponding FPGA implementations but have significantly higher design and fabrication costs. FPGAs are COTS chips and can be reconfigured and reused for different ANN applications significantly lowering implementation costs for low volume productions. The last decade has seen a lot of advancement in reconfigurable hardware technology. FPGA chips with built-in RAMs, multipliers, gigabit transceivers, on-chip embedded processors, and faster clock speeds have attracted many neural network designers. As compared to analog, digital implementations have relatively larger circuit sizes and higher power consumption.

2.1.1. Data Representation. Digital neural implementations represent the real-valued data such as weights and biases using fixed point, floating point, or specialized representations such as pulse stream encoding. The choice of a

particular representation is a tradeoff between arithmetic circuit size and speed, data precision, and the available dynamic range for the real values. Floating point arithmetic units are slower, larger, and more complicated than their fixed point counterparts, which are faster, smaller, and easier to design.

Generally, floating point representations of real-valued data for neural networks are found in custom ASIC implementations. Aibe et al. [14] used floating point representation for their implementation of probabilistic neural networks (PNNs). In PNNs, the estimator of the probabilistic density functions is very sensitive to the smoothing parameter (the network parameter to be adjusted during neural network learning). Hence, high accuracy is needed for the smoothing parameter, making floating point implementations more attractive. Ayela et al. demonstrated an ASIC implementation of MLPs using a floating point representation for weights and biases [15]. They also support on-chip neural network training using the backpropagation algorithm and are listed also in Section 2.1.3. Ramacher et al. present a digital neurochip called SYNAPSE-1 [16]. It consists of a 2-dimensional systolic array of neural signal processors that directly implement parts of common neuron processing functions such as matrix-vector multiplication and finding a maximum. These processors can be programmed for specific neural networks. All the real values on this chip are represented using floating point representation.

For FPGA implementations the preferred implementation choice has been fixed point due to chip capacity restrictions, but advances in FPGA densities may make floating point representations practical. Moussa et al. demonstrate implementations of MLP on FPGAs using fixed and floating point representations [17]. Their results show that the MLP implementation using fixed point representation was over 12x greater in speed, over 13x smaller in area, and achieved far greater processing density as compared to the MLP implementation using floating point representation. The works in [17–21] present precision analysis of ANN implementations and conclude that it is possible to train ANNs with fixed point weights and biases. But there is a tradeoff between minimum precision, dynamic data range, and the area required for the implementation of arithmetic units. Higher precision has fewer quantization errors but require larger arithmetic units, whereas lower precision arithmetic units are smaller, faster, and more power efficient but may have larger quantization errors that can limit the ANN's capabilities to learn and solve a problem. Tradeoff between precision, circuit area, and speed necessitates numerical analysis to determine the minimum precision for an application. Holt and Baker [19], Holt and Hwang [20], and Holt and Hwang [21] investigated the minimum precision problem on a few ANN benchmark classification problems using simulations and found 16-bit data widths with 8-bit fractional parts as sufficient for networks to learn and correctly classify the input datasets. Ros et al. demonstrate a fixed point implementation of spiking neural networks on FPGAs [22]. Pormann et al. demonstrate fixed point implementations of neural associative memories, self-organizing feature maps, and basis function networks on FPGAs [23]. Some other reported implementations that used fixed point representations can be found in [24–32].

As a third alternative many have proposed specialized data encoding techniques that can simplify arithmetic circuit designs. Marchesi et al. proposed special training algorithms for multilayer perceptrons that use weight values that are powers of two [33]. The weight restriction eliminates the need for multipliers in the design which are replaced with simple shift registers. Other approaches encode real values in stochastic bit streams and implement the multipliers in bit-serial fashion using simple logic gates instead of complex arithmetic units [34–37]. The advantage is that the product of the two stochastic bit streams can be computed using a simple bitwise "xor." But the disadvantage is that for multiplications to be correct, the bit streams should be uncorrelated. To produce these would require independent random sources which require more chip resources to implement. Also the precision limitation may affect the ANNs capability to learn and solve a problem. Murray and Smith's implementation of ANNs [38] used pulse-stream encoding for real values which was later adopted by Lysaght et al. [39] for implementations on Atmel FPGAs. Salapura used delta encoded binary sequences to represent real values and used bit stream arithmetic to calculate a large number of parallel synaptic calculations [40].

Chujo et al. have proposed an iterative calculation algorithm for the perceptron type neuron model that is based on a multidimensional, binary search algorithm [41]. Since the binary search does not require any sum of products functionality, it eliminates the need for expensive multiplier circuitry in hardware. Guccione and Gonzalez used a vector-based data parallel approach to represent real values and compute the sum of products [42]. The distributed arithmetic (DA) approach of Mintzer [43] for implementing FIR filters on FPGAs was used by Szabo et al. [44] for their implementation of neural networks. They used Canonic Signed Digit Encoding (CSD) technique to improve the hardware efficiency of the multipliers. Noory and Groza also used the DA neural network approach and targeted their design for implementation on FPGAs [45]. Pasero and Perri use LUTs to store all the possible multiplication values in an SRAM to avoid implementing costly multiplier units in FPGA hardware [46]. It requires a microcontroller to precompute all the possible product values for the fixed weight and stores it in the SRAM.

2.1.2. Design Flexibility. An important design choice for neural network hardware implementations is the degree of structure adaptation and synaptic parameter flexibility. An implementation with fixed network structure and weights can only be used in the recall stage of each unique application, thus necessitating a redesign for different ANN applications. For ASIC implementations this could be quite expensive due to high fabrication costs. An advantage of FPGA ANN implementations is the capability of runtime reconfiguration to retarget the same FPGA device for any number of different ANN applications, substantially reducing the implementation costs. There are several different

motivations of using FPGAs for ANN implementations such as prototyping and simulation, density enhancement, and topology adaptation. The purpose of using FPGAs for prototyping and simulation is to thoroughly test a prototype of the final design for correctness and functionality before retargeting the design to an ASIC. This approach was used in [47]. Runtime reconfigurations in FPGAs can be used for density enhancement to implement larger network sizes via time folding that a single FPGA device may not be able to hold. This increases the amount of effective functionality per unit reconfigurable circuit area of the FPGAs. Eldredge et al. used this technique to implement the backpropagation training algorithm on FPGAs [48, 49]. The algorithm was divided temporally in three different executable stages and each stage was reconfigured on the FPGA using runtime reconfiguration. More details on this and other follow-up implementations to Eldredge's technique are covered in Section 2.1.3 for on-chip learning. The runtime reconfiguration in FPGAs can also be used for topology adaptation. ANNs with different structures and synaptic parameters targeting different applications can be loaded on the FPGA via runtime reconfiguration. One of the earliest implementations of artificial neural networks on FPGAs, the Ganglion connectionist classifier, used FPGA reconfigurations to load networks with different structures for each new application of the classifier [50]. Similar approaches of using runtime reconfiguration to retarget the FPGA for different ANN applications are found in [22, 25–31, 51, 52].

Runtime reconfiguration provides the flexibility to retarget the FPGA for different ANN designs but is impractical for use with dynamic adaptations required for online training. The overheads associated with runtime reconfiguration are on the order of milliseconds. Thus the overheads of repetitive reconfigurations required in the iterative training procedures may outweigh any benefits associated with online adaptations. The design presented in this paper is an online trainable ANN implementation on FPGAs that supports dynamic structure and parameter updates without requiring any FPGA reconfiguration.

ASIC implementations of flexible neural networks that can be retargeted for different applications have been reported in literature. The Neural Network Processor (NNP) from Accurate Automation Corp. was a commercial neural hardware chip that could be adapted online [53]. It had machine instructions for various neuron functions such as multiply and accumulate or transfer function calculation. Thus the network can be programmed using the NNP assembly instructions for different neural network implementations. Mathia and Clark compared performance of single and parallel (up to 4) multiprocessor NNPs against that of the Intel Paragon Supercomputer (up to 128 parallel processor nodes). Their results show that the NNP outperformed the Intel Paragon by a factor of 4 [54].

2.1.3. On-Chip/Off-Chip Learning. ANN training algorithms iteratively adapt network structure and synaptic parameters based on an error function between expected and actual outputs. Hence an on-chip trainable network design should have the flexibility to dynamically adapt its structure and synaptic parameters. Most reported ANN implementations use software simulations to train the network, and the obtained network is targeted to hardware offline [22–31, 55, 56]. However, few implementations have been reported that support an on-chip training of neural networks. Eldredge et al. demonstrate an implementation of the backpropagation algorithm on FPGAs by temporally dividing the algorithm into three sequentially executable stages of the feedforward, error backpropagation, and synaptic weight update [48, 49]. The feed-forward stage feeds in the inputs to the network and propagates the internal neuronal outputs to output nodes. The backpropagation stage calculates the mean squared output errors and propagates them backward in the network in order to find synaptic weight errors for neurons in the hidden layers. The update stage adjusts the synaptic weights and biases for the neurons using the activation and error values found in the previous stages. Hadley et al. improved the approach of Eldredge by using partial reconfiguration of FPGAs instead of full-chip runtime reconfiguration [57]. Gadea et al. demonstrate a pipelined implementation of the backpropagation algorithm in which the forward and backward passes of the algorithm can be processed in parallel on different training patterns, thus increasing the throughput [58]. Ayala et al. demonstrated an ASIC implementation of MLPs with on-chip backpropagation training using floating point representation for real values and corresponding dedicated floating point hardware [15]. The backpropagation algorithm implemented is similar to that by Eldredge et al. [48, 49]. A ring of 8 floating point processing units (PUs) are used to compute the intermediate weighted sums in the forward stage and the weight correction values in the weight update stage. The size of the memories in the PUs limits the number of neurons that can be simulated per hidden layer to 200. A more recent FPGA implementation of backpropagation algorithm can be found in [59]. Witkowski et al. demonstrate an implementation of hyper basis function networks for function approximation [60]. Both learning and recall stages of the network are implemented in hardware to achieve higher performance. The GRD (Genetic Reconfiguration of DSPs) chip by Murakawa et al. can perform on-chip online evolution of neural networks using genetic algorithms [61]. The GRD chip is a building block for the configuration of a scalable neural network hardware system. Both the topology and the hidden layer node functions of the neural network mapped on the GRD chips can be reconfigured using a genetic algorithm (GA). Thus, the most desirable network topology and choice of node functions (e.g., Gaussian or sigmoid function) for a given application can be determined adaptively. The GRD chip consists of a 32-bit RISC processor and fifteen 16-bit DSPs connected in a binary-tree network. The RISC processor executes the GA code and each of the DSPs can support computations of up to 84 neurons. Thus each GRD chip can support 1260 neurons. Multiple GRD chips can be connected for a scalable neural architecture. Two commercially available neurochips from the early 1990s, the CNAPS [62] and MY-NEUPOWER [63], support on-chip training. CNAPS was an SIMD array

of 64 processing elements per chip that are comparable to low precision DSPs and was marketed commercially by Adaptive solutions. The complete CNAPS system consisted of a CNAPS server that connected to a host workstation and Codenet software development tools. It supported Kohonen LVQ (linear vector quantization) and backpropagation networks. MY-NEUPOWER supported various learning algorithms such as backpropagation, Hopfield, and LVQ and contained 512 physical neurons. The chip performed as a neural computational engine for software package is called NEUROLIVE [63].

The following references discuss analog and hybrid implementations that support on-chip training. Zheng et al. have demonstrated a digital implementation of backpropagation learning algorithm along with an analog transconductance-model neural network [64]. A digitally controlled synapse circuit and an adaptation rule circuit with an R-2R ladder network, a simple control logic circuit, and an UP/DOWN counter are implemented to realize a modified technique for the backpropagation algorithm. Linares-Barranco et al. also show an on-chip trainable implementation of an analog transconductance-model neural network [65]. Field Programmable Neural Arrays (FPNAs), an analog neural equivalent of FPGAs, are a mesh of analog neural models interconnected via a configurable interconnect network [66–70]. Thus, different neural network structures can be created dynamically, enabling on-chip training. Intel's ETANN (Electronically Trainable Analog Neural Network) [71] and the Mod2 Neurocomputer [72] are other examples of on-chip trainable analog neural network implementations. Schmitz et al. use the embedded processor on the FPGA to implement genetic algorithm operators like selection, crossover, and mutation [73]. This FPGA is closely coupled as a coprocessor to a reconfigurable analog ANN ASIC chip on the same PCB. A host processor initializes this PCB and oversees the genetic evolution process.

2.1.4. Activation Function Implementation. Activation functions, or transfer functions, used in ANNs are typically nonlinear, monotonically increasing sigmoid functions. Examples include hyperbolic tangent and logistic sigmoid functions. Digital ANN implementations use piecewise linear approximations of these to implement in hardware either as a direct circuit implementation or as a look-up table. Omondi et al. show an implementation of piecewise linear approximation of activation functions using the CORDIC algorithm on FPGAs [74]. Krips et al. show an implementation of piecewise linear approximation of activation functions precomputed and stored in LUTs [30]. Direct circuit implementation of the activation function requires a redesign of hardware logic for every application that uses a different activation function. In such scenarios the LUT approach maybe more flexible as the function values can be precomputed and loaded in the LUT offline. But the LUTs may occupy significant higher chip area as compared to direct implementations. Each extra bit in the data width more than doubles the size of the LUT. These tradeoffs are further discussed in Section 4.

2.2. Analog Neural Hardware Implementations. Analog artificial neurons are more closely related to their biological counterparts. Many characteristics of analog electronics can be helpful for neural network implementations. Most analog neuron implementations use operational amplifiers to directly perform neuron-like computations, such as integration and sigmoid transfer functions. These can be modeled using physical processes such as summing of currents or charges. Also, the interface to the environment may be easier as no analog-to-digital and digital-to-analog conversions are required. Some of the earlier analog implementations used resistors for representing free network parameters such as synaptic weights [75]. These implementations using fixed weights are not adaptable and hence can only be used in the recall phase. Adaptable analog synaptic weight techniques represent weights using variable conductance [76, 77], voltage levels between floating gate CMOS transistors [71, 78–80], capacitive charges [81, 82], or using charged coupled devices [83, 84]. Some implementations use digital memories for more permanent weight storage [85]. The works in [64–71, 73] are some other analog implementations previously discussed in Section 2.1.3 above. Although there are many advantages of implementing analog neural networks as discussed above, the disadvantage is that the analog chips are susceptible to noise and process parameter variations, and hence need a very careful design.

2.3. Hybrid Neural Hardware Implementations. Hybrid implementations combine analog, digital, and other strategies such as optical communication links with mixed mode designs in an attempt to get the best that each can offer. Typically hybrid implementations use analog neurons taking advantage of their smaller size and lower power consumption and use digital memories for permanent weight storage [85, 86]. The mixed-signal design of the analog neurons with the digital memories on the same die introduces a lot of noise problems and requires isolation of the sensitive analog parts from the noisy digital parts using techniques such as guard rings. Sackinger et al. demonstrate a high-speed character recognition application on the ANNA (Analog Neural Network Arithmetic and logic unit) chip [87]. The ANNA chip can be used for a wide variety of neural network architectures but is optimized for locally connected weight-sharing networks and time-delay neural networks (TDNNs). Zatorre-Navarro et al. demonstrate mixed mode neuron architecture for sensor conditioning [88]. It uses an adaptive processor that consists of a mixed four-quadrant multiplier and a current conveyor that performs the nonlinearity. Synaptic weights are stored using digital registers and the training is performed off-chip.

Due to the large number of interconnections, routing quickly becomes a bottleneck in digital ASIC implementations. Some researchers have proposed hybrid designs using optical communication channels. Maier et al. [89] show a hybrid digital-optical implementation that performs neural computations electronically, but the communication links between neural layers use an optical interconnect system. They demonstrate a magnitude of performance

improvement as compared to a purely digital approach. But on the flip side it increases hardware costs and complexity for converting signals between the electronic and the optical systems. Craven et al. [90] have proposed using frequency multiplexed communication channels to overcome the communication bottleneck in fully connected neural networks.

2.4. Summary. Custom neural network hardware implementations can best exploit the inherent parallelism in computations observed in artificial neural networks. Many implementations have relied on offline training of neural networks using software simulations. The trained neural network is then implemented in hardware. Although these implementations have good recall speedups, they are not directly comparable to the implementation reported here which supports on-chip training of neural networks. On-chip trainable neural hardware implementations have also been reported in literature. Most of the reported ones are custom ASIC implementations such as the GRD chip by Murakawa et al. [61], on-chip backpropagation implementation of Ayala et al. [15], CNAPS by Hammerstrom [62], MY-NEUPOWER by Sato et al. [63], and FPNA by Farquhar, et al. [66]. FPGA-based implementations of on-chip training algorithms have also been reported such as the backpropagation algorithm implementations in [48, 49, 57, 58]. An online trainable implementation of hyperbasis function networks has been reported in [60]. The implementation presented here differs from the reported ones in one or more of the following: (i) the artificial neural network supported (block-based neural networks in our case), (ii) the training algorithm, and (iii) the implementation platform. The design presented in this paper supports on-chip training without reliance on FPGA reconfigurations, unlike some of the approaches listed above. It uses genetic algorithms to train the BbNNs. The genetic operators such as selection, crossover, and mutation are implemented on the embedded processor PPC 405 on the FPGA die, similar to the approach of Schmitz et al. [73]. But unlike their approach the neural network designed is a digital implementation in the configurable logic portion of the same FPGA chip.

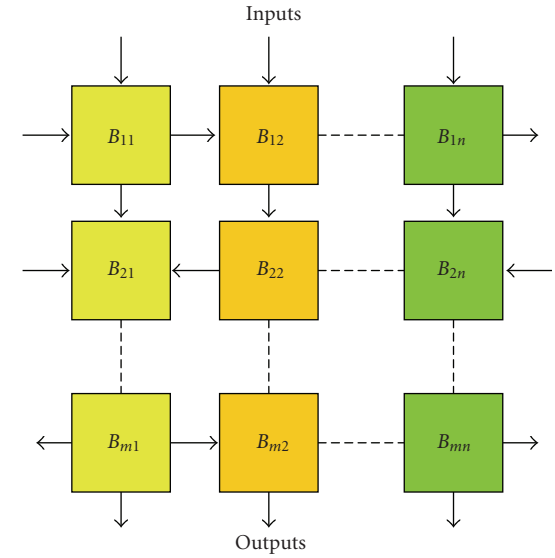


Figure 2: Block-based Neural Network topology.

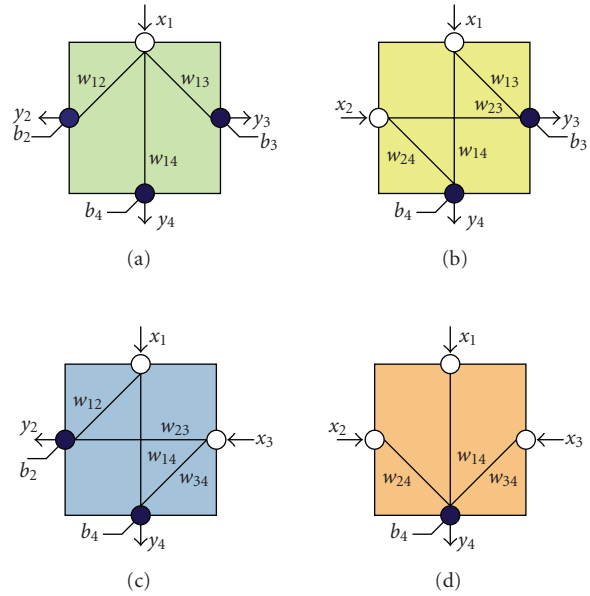


Figure 3: Four different internal configurations of a basic neuron block: (a) 1/3, (b) 2/2 (left), (c) 2/2 (right), and (d) 3/1 configurations.

3. Block-Based Neural Networks

A block-based neural network (BbNN) is a network of neuron blocks interconnected in the form of a grid as shown in Figure 2 [91]. Neuron blocks are the information processing elements of the network and can have one of four possible internal configurations depending on the number of inputs and outputs: (i) 1-input, 3-output (1/3), (ii) 2-input, 2-output (2/2) (left side output), (iii) 2-input, 2-output (2/2) (right side output), and (iv) 3-input, 1-output (3/1). These are shown in Figure 3.

Outputs of the neuron block are a function of the summation of weighted inputs and a bias as shown in (1) below. The internal block configurations and the dataflow through the network are determined by the network structure. Figure 4 shows three unique 2×2 BbNN structures:

$$y_k = g\left(b_k + \sum_{j=1}^{J} w_{jk} x_j\right), \quad k = 1, 2, \ldots, K, \tag{1}$$

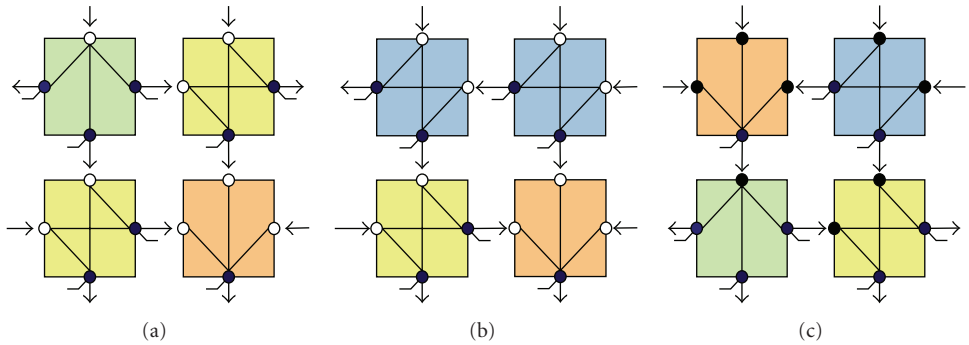


Figure 4: Three different 2×2 BbNN network structures.

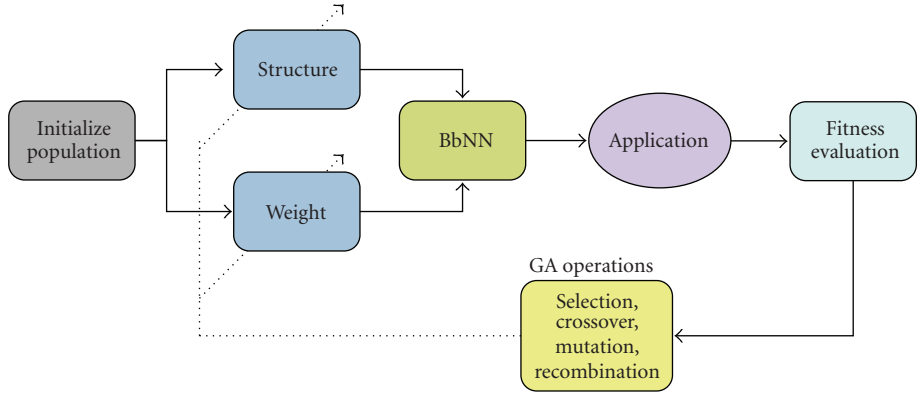


Figure 5: Flowchart of the genetic evolution process.

where,

y_k : is the kth output signal of the neuron block,

x_j : is the jth input signal of the neuron block,

w_{jk} : is the synaptic weight connection between jth input node and kth output node,

b_k : is the bias at kth output node,

J, K : are the number of input and output nodes respectively of a neuron block,

$g(\bullet)$: is the activation function.

Moon and Kong [91] have evaluated and compared BbNN and MLP characteristics. They state that any synaptic weight in an MLP network can be represented by a combination of more than one weight of the BbNN. Thus, a BbNN of $m \times n$ can represent the equivalent structure of the MLP network MLP$(n, m-1)$, where n denotes the maximum number of neurons per layer and $m-1$ represents the total number of hidden layers in an MLP network.

BbNN training is a multiparametric optimization problem involving simultaneous structure and weight optimizations. Due to multimodal and nondifferentiable search space, global search techniques such as genetic algorithms are preferred over local search techniques to explore suitable solutions. Genetic algorithms (GAs) are evolutionary algorithms inspired from the Darwinian evolutionary model where in a population of candidate solutions (individuals or phenotypes) of a problem, encoded in abstract representations (called chromosomes or the genotypes), are evolved over multiple generations towards better solutions. The evolution process involves applying various genetic operators such as selection, crossover, mutation, and recombination to the chromosomes to generate successive populations with selection pressure against the least fit individuals. Figure 5 shows a flowchart of the genetic evolution process. The network structure and weights of the BbNN are encoded as chromosomes as shown in Figure 6. Detailed information on the BbNN GA evolution process can be found in [92–94]. BbNNs have been applied to navigation [91], classification [1, 92–94], and prediction [95] problems in the past.

4. BbNN FPGA Implementation

Many FPGA ANN implementations are static implementations, targeted and configured offline for individual applications. The main design objective of this implementation is enabling intrinsic adaptation of network structure and internal parameters such that the network can be trained online without relying on runtime FPGA reconfigurations. Previous versions of this implementation were reported in [1, 96]. The target system environment for the implementation is an embedded computing system. As a result various design choices were made to optimize area and power constraints. It was assumed that the hardware resources available for

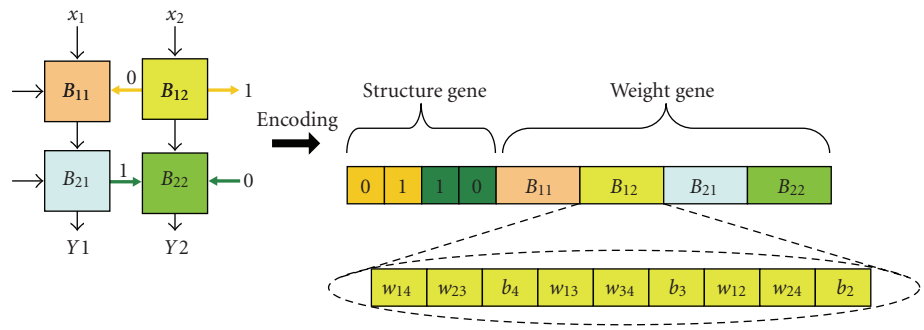


Figure 6: BbNN encoding.

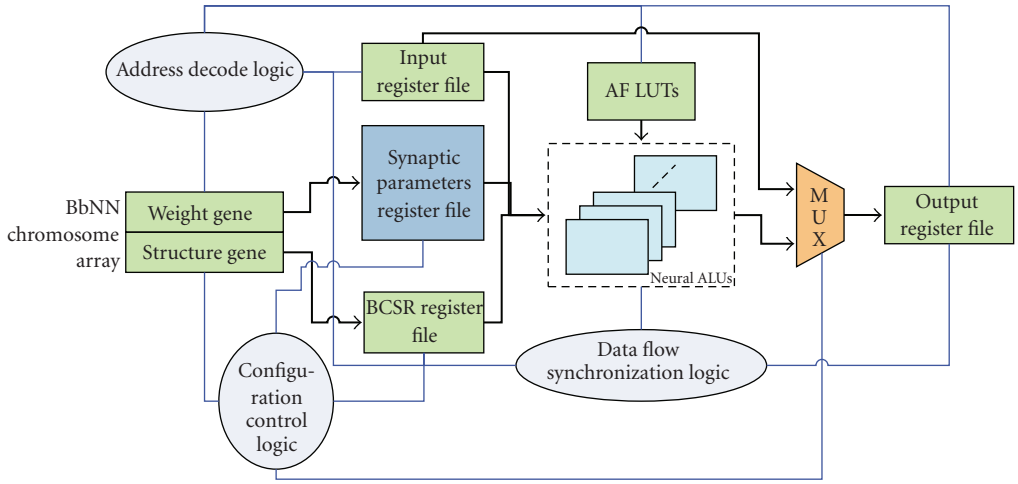


Figure 7: BbNN logic diagram.

this design in the target environment will be FPGA(s) and supporting memory and control circuits. Hence the design was prototyped on Xilinx Virtex-II Pro (XC2VP30) FPGA development boards [97]. The design was tested on two different prototyping boards, a stand-alone Digilent Inc. XUP development board [98] and an Amirix AP130 FPGA board [99]. The targeted FPGA has two on-chip PowerPC 405 embedded processor cores, 30,816 logic cells, 136 built-in 18×18 multipliers, and 2448 KBits (306 KBytes) of on-chip block RAM. The following sections describe the neuron block and network design in detail. Figure 7 shows the logic diagram of the design.

4.1. Data Representation and Precision. The ability to use variable bit-widths for arithmetic computations gives FPGAs significant performance and resource advantages over competing computational technologies such as microprocessors and GPUs. Numerical accuracy, performance, and resource utilization are inextricably linked, and the ability to exploit this relationship is a key advantage of FPGAs. Higher precision comes at the cost of lower performance, higher resource utilization, and increased power consumption. But at the same time, lower precision may increase the round-off errors adversely impacting circuit functionality.

The inputs, outputs, and internal parameters such as synaptic weights and biases in BbNN are all real valued. These can be represented either as floating point or fixed point numbers. Floating point representations often have a significantly wider dynamic range and higher precision as compared to fixed point representations. However, floating point arithmetic circuits are often more complicated, have a larger footprint in silicon, and are significantly slower compared to their fixed point counterparts. On the other hand fixed point representations have higher round-off errors when operating on data with large dynamic range. Although escalating FPGA device capacities have made floating point arithmetic circuits feasible for many applications, their use in our application will severely restrict the size of the network (i.e., the number of neuron blocks per network) that can be implemented on a single FPGA chip. Thus our implementation uses fixed point arithmetic for representing real-valued data. Also, [19–21] investigated the minimum fixed point precision needed for various benchmark classification problems on artificial neural networks and found 16 bits of precision as adequate for reliable operation. Hence all real valued data are represented as 16-bit fixed point numbers in our implementation.

4.2. Activation Function Implementation. Activation functions used in ANNs are typically nonlinear, monotonically increasing sigmoid functions. A custom circuit implementation of these functions may be area efficient as compared

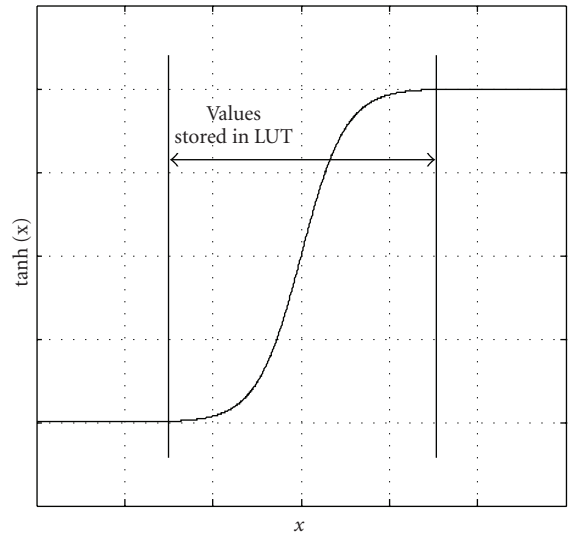


Figure 8: Illustrating activation function implementation in LUT.

to piecewise linearly approximated values stored in lookup tables (LUTs) but is inflexible and involves complicated circuitry. In our design we have chosen to use the internal FPGA BRAMs to implement LUTs which can be preloaded and reloaded as necessary with activation function values. The size of the lookup table required is directly associated with the data widths used. A 16-bit fixed point representation requires an LUT that is 16 bits wide and 2^{16} deep. This requires a total of 128 KBytes per LUT. Even though it may be simpler to use separate LUTs for each neuron block in the network such that each block can have parallel access to the LUTs, it is inefficient utilization of on-chip resources and the circuit will have longer setup times. For our design we have chosen to share an LUT across each column of neuron blocks in a network. This ensures that every neuron block that is actively computing at a given time will have independent access to the LUTs. This is guaranteed as no two blocks in a column can execute simultaneously for the case of feedforward networks implemented here. Sharing one LUT per column may also set an upper bound to the total number of columns implemented per FPGA due to limited BRAM resources per chip. But due to various optimizations described next the main bottleneck for our design is the number of logic resources on the FPGA used for neuron block designs and not the available internal memory. The 128 KB LUT is still large enough to restrict the number of columns that can be implemented per FPGA. For example, on the VirtexIIPro (V2P30) FPGA from Xilinx we would be restricted to only two columns. This severely restricts our ability to implement any interesting applications on the BbNN. Hence to further optimize the size of the LUT we restricted it to 16 bits $\times$ 2^{12} (i.e., 8 KB per LUT). Since the activation functions monotonically increase during only a small range of input data and saturate outside that window (e.g., hyperbolic tangent or logistic sigmoid functions) this optimization, in effect, stores only the transient portion of the activation function as illustrated in Figure 8.

4.3. Neuron Block Design. Kothandaraman designed a core library of neuron blocks with different internal block configurations for implementation on FPGAs [25]. A network can be stitched together with these cores using an HDL, but with fixed network structure and internal parameters. Thus a new network needs to be designed for each unique application. The objective to design an online, evolvable network necessitates dynamically adaptable network design that can change structure and internal parameters on-the-fly with little overhead.

A simplistic design can combine all cores in the library into a larger neuron block and use a multiplexor to select individual cores with the correct internal configuration. But the area and power overheads of such an approach render it impractical. Instead, a smarter block design is presented here that can emulate all internal block configurations dynamically as required and is less than a third the size of the simplistic larger block. For obvious reasons the block design is called the Smart Block-based Neuron (SBbN). An internal configuration register within each SBbN called the Block Control and Status Register (BCSR) regulates the configuration settings for the block. BCSR is a 16-bit register and is part of the configuration control logic section of the neuron block that defines the state and configuration mode of the block. All the bits of this register except 8 through 11 are read-only and the register can be memory or I/O mapped to the host systems address space for read and write operations. Figure 9 shows all the bit fields of the BCSR. States of BCSR bits 4 through 7 determine the current internal configuration of the neuron block. Setting bit 3 deactivates the block which then acts as a bypass from inputs to corresponding outputs. Figure 10 shows the BCSR settings and corresponding internal block configurations.

4.4. Configuration Control Logic. Configuration control logic uses the structure gene within the BbNN chromosome array as its input and sets the BCSR configuration bits of all the neuron blocks in the network. The translation process from the structure gene array into internal configuration bits of the neuron blocks is illustrated in Figure 11 for a 10×4 network. The translation logic extracts the structure gene from the chromosome array, divides it by the number of rows, and extracts the corresponding bits for each column from all the rows. The extracted column bits are divided across corresponding neuron blocks in the column and written to the internal BCSR of the corresponding neuron block. The neuron blocks not used are deactivated. Since the configuration control logic is combinational logic, the network is reconfigured to the correct network structure based on the structure gene loaded in the BbNN chromosome array immediately after a small delay.

4.5. Address Decode Logic. Address decoder provides a memory-mapped interface for the read/write registers and memories in the network design. It decodes address, data, and control signals for the input and output register files, the BbNN chromosome array, the BCSRs within each neuron block, and the activation function LUTs.

BCSR-block control and status register

15	14	13	12	11	10	9	8
0	0	0	0	AF1	AF2	AF3	AF4

7	6	5	4	3	2	1	0
ND1	ND2	ND3	ND4	DACT	0	CB	FM

(a)

Bits		Description
15-12		*reserved*
11	AF1	Node 1 Activation function enable ("0"-Purelin / "1"-LUT AF)
10	AF2	Node 2 Activation function enable ("0"-Purelin /"1"-LUT AF)
9	AF3	Node 3 Activation function enable ("0"-Purelin /"1"-LUT AF)
8	AF4	Node 4 Activation function enable ("0"-Purelin /"1"-LUT AF)
7	ND1	Node 1 direction (hardcoded as "0"-Input)
6	ND2	Node 2 direction ("0"-Input /"1"-Output)
5	ND3	Node 3 direction ("0"-Input /"1"-Output)
4	ND4	Node 4 direction (hardcoded as "1"-Output)
3	DACT	Deactivate block
2		*reserved*
1	CB	Configuration busy signal
0	FM	Neuron fire mode signal

(b)

Figure 9: Block control and status register (BCSR).

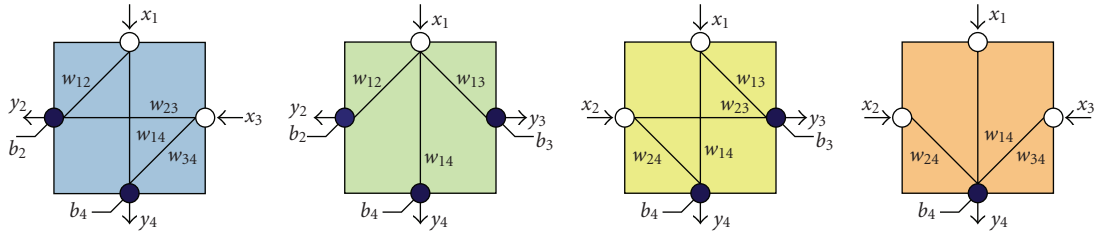


Figure 10: SBbN emulation of the internal block configurations based on BCSR settings.

4.6. Dataflow Synchronization Logic. To enable network scalability across multiple FPGAs the data synchronization between neuron blocks is asynchronous. Synchronization is achieved using generation and consumption of tokens as explained next. The logic associates a token with each input and output registers of every neuron block in the network. Each neuron block can only compute outputs (i.e., fire) when all of its input tokens are valid. On each firing the neuron block consumes all of its input tokens and generates output tokens. The generated output tokens in turn validate the corresponding input tokens of the neuron blocks next in the dataflow. This is illustrated in Figure 12. Black dots (•) in the figure represent valid tokens. Asynchronous communication mechanism between neuron blocks facilitates implementing larger network sizes either by scaling networks across multiple FPGAs or by folding the network in a time multiplexed fashion within a single FPGA.

4.7. Intrinsic BbNN Evolution. The design features explained above enable dynamic adaptations to network structure and internal synaptic parameters. Hence the network can be trained online, intrinsically under the control of the evolutionary training algorithm described in Section 3 above. There are several options for implementing the training algorithm based on the end system solution and other system design constraints. Examining the execution profiles of the serial genetic evolution code, the most computationally intensive section is found to be the population fitness evaluation function. Fitness evaluations involve applying

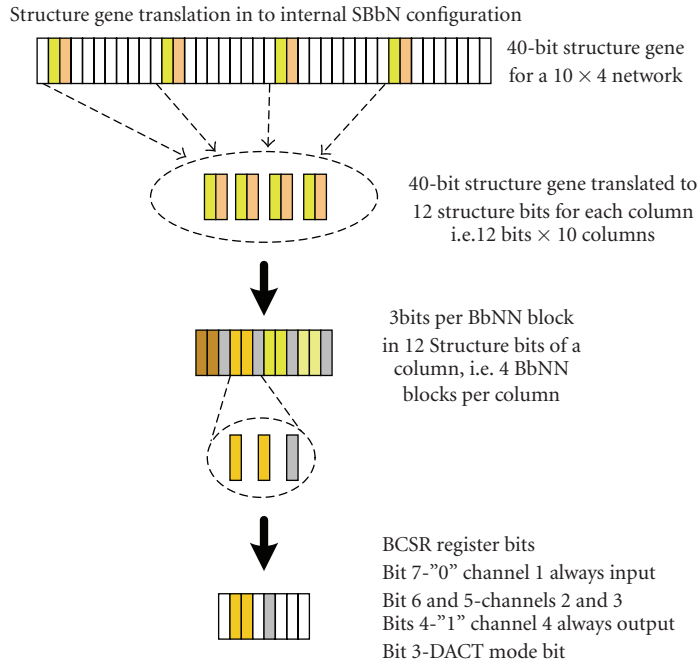


Figure 11: Gene translation process within the configuration control logic.

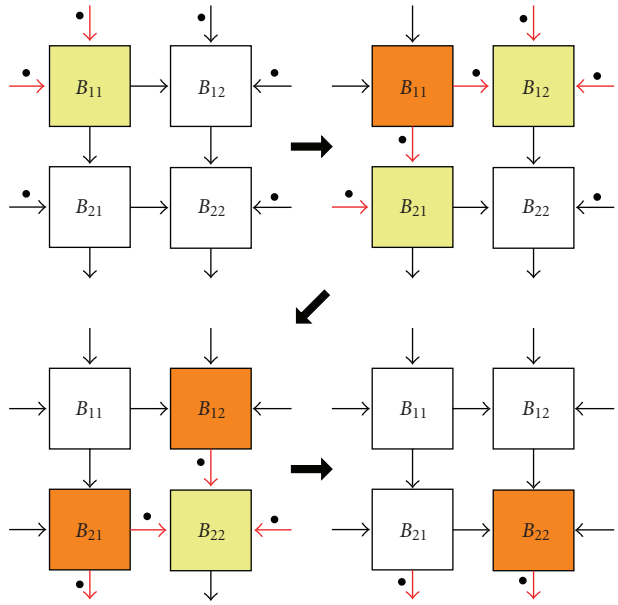


Figure 12: Dataflow synchronization logic.

the training patterns to the network and examining the measured outputs against expected values to determine the network fitness. This is performed for each network in the population to determine the average and maximum population fitness values. Genetic operations such as selection, crossover, and mutation are relatively less computationally intensive. Hence an obvious software-hardware partitioning is to perform genetic operations in software running on the host microprocessor and fitness evaluations in hardware executing in FPGAs. This has the benefit of dedicating all the available configurable logic space in the FPGAs to implement the neural network, facilitating implementation of larger network sizes.

Escalating FPGA logic densities has enabled building a programmable system-on-chip (PSoC) with soft or hard-core processors, memory, bus system, IO, and other custom cores on a single FPGA device. The Xilinx Virtex-II Pro FPGA used for prototyping our implementation has on-chip, hard-core PPC405 embedded processors which are used to design our PSoC prototyping platform. The platform is

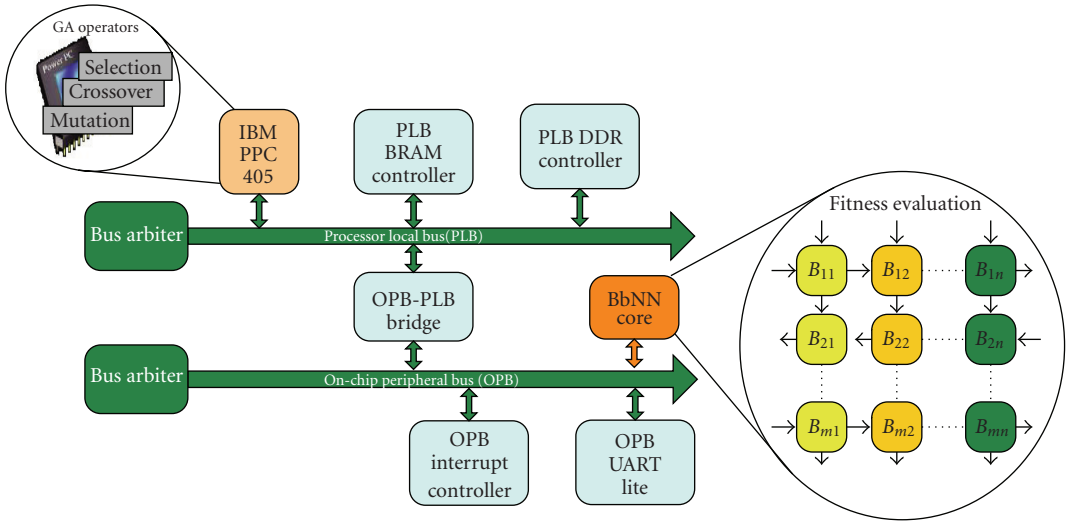


Figure 13: PSoC platform used for prototyping BbNN implementation.

Table 1: Peak and relative computational capacities and capacity per mW of commercial embedded processors. Relative values are normalized to PPC405 numbers.

Processor	Organization	Cycle freq	Power	MOPS	Relative MOPS	MOPS/mW	Relative MOPS/mW
MIPS 24Kc	1×32	261 MHz	363 mW	87	0.65	0.24	0.14
MIPS 4KE	1×32	233 MHz	58 mW	78	0.59	1.33	0.76
ARM 1026EJ-S	1×32	266 MHz	279 mW	89	0.67	0.32	0.18
ARM 11MP	1×32	320 MHz	74 mW	107	0.80	1.45	0.83
ARM 720T	1×32	100 MHz	20 mW	33	0.25	1.67	0.95
PPC 405	1×32	400 MHz	76 mW	133	1.00	1.75	1.00
PPC 440	1×32	533 MHz	800 mW	178	1.34	0.22	0.13
PPC 750FX	2×32	533 MHz	6.75 W	355	2.67	0.05	0.03
PPC 970FX	2×64	1 GHz	11 W	667	5.02	0.06	0.03

designed using Xilinx EDK and ISE tools. A block diagram for the designed platform is shown in Figure 13. The BbNN core is memory-mapped to the PPC405's address space via the on-chip peripheral bus (OPB). The genetic operations such as selection, crossover, mutation, and recombination are implemented in software executing on the PPC405 processor and the population fitness evaluation is implemented on the BbNN core as shown in Figure 13. On-board DDR is used as data memory to quickly access and store the chromosome populations and the training vectors for the evolution process. A fixed point version of the BbNN GA evolution algorithm is used due to the limited computational capacity of the on-chip PowerPC processor. The platform offers a very compact solution that is deployable and adaptable in field for embedded applications with area and power constraints.

In the case of target environments with less stringent area and power constraints, other higher-capacity embedded solutions such as single-board computers with FPGA accelerators can be used. The GA operators can thus be implemented on the on-board processor and the fitness evaluation can be performed in the FPGA using the BbNN core. Table 1 surveys computational capacities and capacity per mW of some commercial embedded processors. Also shown are relative values normalized to PPC405 numbers. The capacities are calculated to implement a single 2/2 BbNN block computation, that is 6 integer arithmetic operations. The calculations assume CPI of 1.0 and ignore instruction fetch and data transfer overheads. The Watt ratings are as published in the processor datasheets and do not include power consumption of other associated hardware resources. Based on these calculations, PPC405 offers the best computational capacity per mW out of the processors surveyed. The PPC970FX has 5 times the computational capacity of PPC405, but significantly higher power consumption. Although the numbers approximate capacities for neuron block computations, similar values for GA operators can be extrapolated from these numbers. For our assumed target

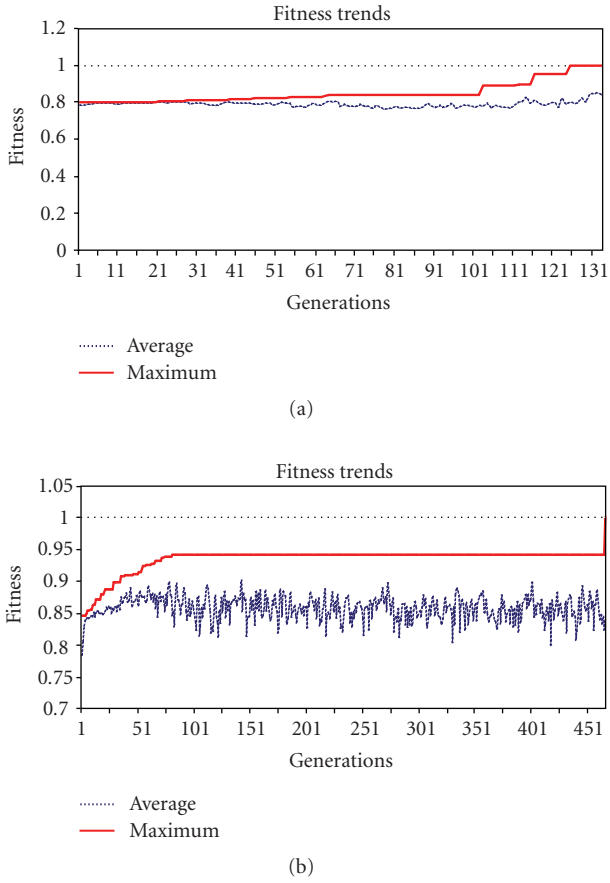


FIGURE 14: Fitness evolution trends for (a) 3-bit and (b) 4-bit parity examples.

system constraints, power efficiency is critical. Thus, the selection of the on-chip PPC405 to execute the GA operators in our prototype implementation is a sound design decision.

4.8. Performance and Resource Utilization. The postsynthesis timing analysis for the design reports a clock frequency of 245 MHz on the Xilinx Virtex-II Pro FPGA (XC2VP30). The designed block takes 10 clock cycles to compute outputs for six synaptic connections per block. Thus, each block has a computational capacity of 147 MCPS per block. Computational capacity is the measure of throughput defined as computational work per unit time. Hence for an artificial neural network it is determined by the number of synaptic connections processed per second (unit CPS). The computational capacity of the network is determined by the number of concurrent block executions, which in turn is dependent on the network structure. At peak computational capacity one block from each network column computes concurrently. Hence an $m \times n$ BbNN has a peak computational capacity of $147n$ MCPS. But the clock frequency on the actual implementation is limited by the on-chip peripheral bus (OPB) clock frequency which is set at 100 MHz for the prototyping platform. Thus, the peak computational capacity is limited to 60 MCPS per block or $60n$ MCPS for a $m \times n$ network size. The minimal platform (as shown in Figure 13) excluding the BbNN occupies about 13% of the Xilinx Virtex-II Pro FPGA (XC2VP30) resources.

Table 2 shows the postsynthesis device utilization summaries for various network sizes. According to the utilization summaries we can fit around 20 neuron blocks on a single FPGA chip along with the rest of the platform. Table 3 shows the postsynthesis device utilization results for a larger device (XC2VP70) from the same Xilinx Virtex-II Pro family of FPGA devices family. This device can hold around 48 neuron blocks.

5. Case Studies

Results from three case studies are presented. The case studies on n-bit parity classifier and Iris data classification demonstrate the functionality of the design. The case study on adaptive forward prediction demonstrates the benefit of online evolution capability.

5.1. n-bit Parity Classification. A parity classifier can be used to determine the value of the parity bit to get even or odd number of 1's in an "n" bit stream. The technique is widely used for error detection and correction with applications in communication and data storage. Results presented here show the outcome of BbNN training to determine the value of the parity bit for 3-bit and 4-bit data streams. A population size of 30 chromosomes is used for genetic evolution with crossover and mutation probabilities set at 0.7 and 0.1, respectively. The evolutionary algorithm uses tournament selection to choose parent chromosomes for crossover and elitism for recombination operations. A logistic sigmoid function was used as the activation function for the neuron block outputs. Figure 14 shows the average and maximum fitness curves for the 3-bit and 4-bit parity examples. The target fitness of 1.0 is reached after 132 generations in the case of the 3-bit parity problem and 465 generations for the 4-bit parity example. Figure 15 shows the evolution trends for the top five structures. Each color indicates a unique structure and the y-axis values determine the number of chromosomes per generation. Figure 16 shows the evolved networks for the 3-bit and the 4-bit parity examples. The average time per generation of the evolution with the PPC405 processor computing at 300 MHz was found to be 11 microseconds.

5.2. Iris Data Classification. This case study uses a well-known dataset in the machine learning community originally compiled by R. A Fisher [100]. The dataset has 150 samples of three classes of Iris plants, *Iris Setosa*, *Iris Versicolour,* and *Iris Virginica* with 50 samples per class. The dataset attributes are sepal length, sepal width, petal length, and petal width for the three classes of the Iris plants. The Iris Setosa class is linearly separable from the other two classes, Iris Versicolour and Iris Virginica. But the latter is not linearly separable from each other, which makes this an interesting problem to test neural classifiers. A BbNN was used to learn and correctly classify this dataset. The classification result is shown in Figure 17. The results show less than a 1.5% misclassification rate. A population size

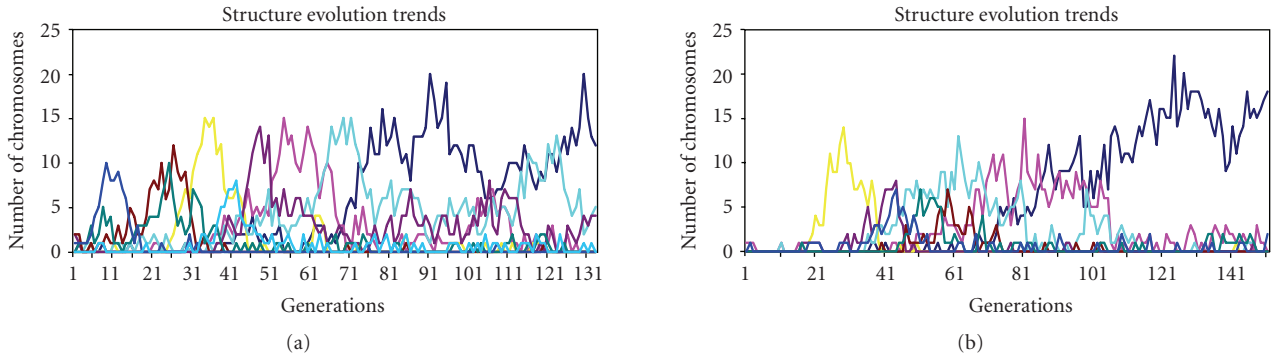


Figure 15: Structure evolution trends for (a) 3-bit and (b) 4-bit parity examples. Each color represents a unique BbNN structure.

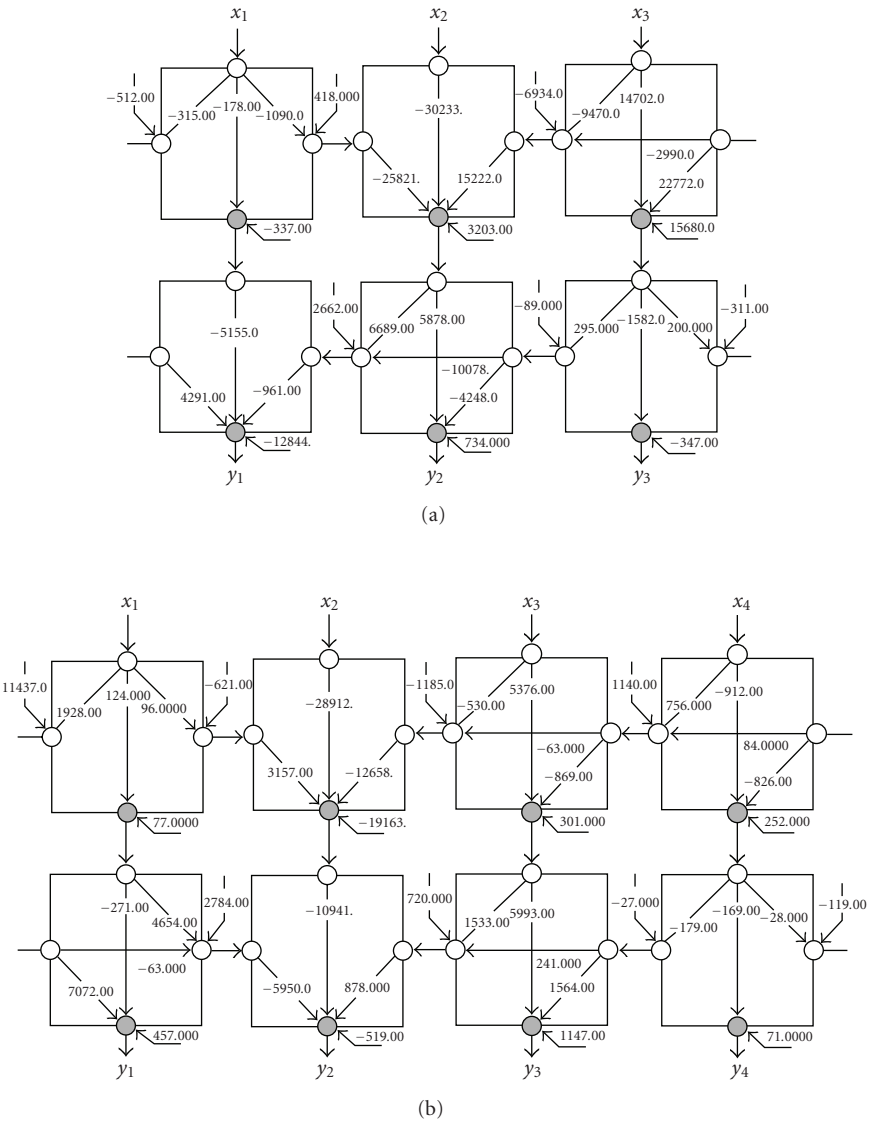


Figure 16: Evolved networks for (a) 3-bit and (b) 4-bit parity examples.

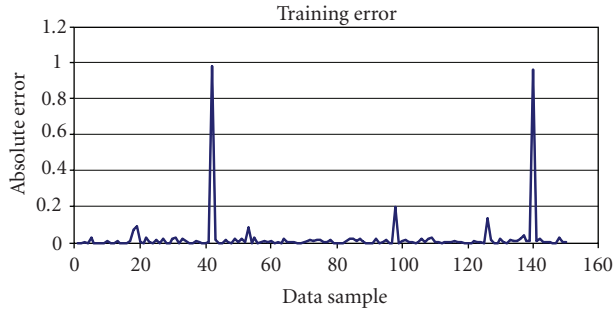


Figure 17: Training error for Iris data classification.

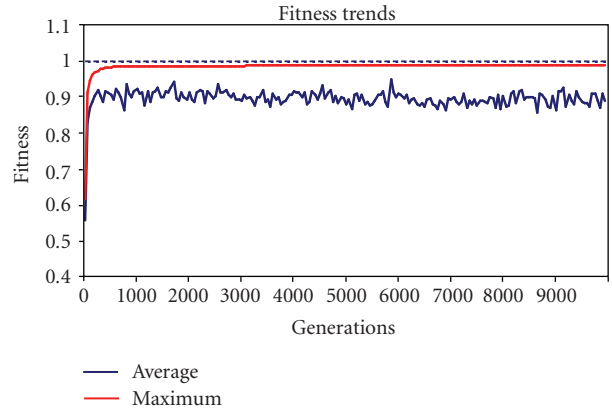


Figure 18: BbNN fitness evolution trends for Iris data classification.

of 80 chromosomes is used for evolution and crossover and mutation probabilities set at 0.7 and 0.2, respectively. Maximum fitness of 0.99 is achieved after 9403 generations. Figure 18 shows the average and maximum fitness trends. Figure 19 shows the structure evolution trends. Figure 20 shows the evolved network. The average evolution time to produce a new generation was found to be 23 microseconds.

5.3. Adaptive Forward Prediction. The objective of this case study is to demonstrate online training benefits of the BbNN platform. This feature is beneficial for applications in dynamic environments where changing conditions may otherwise require offline retraining and deployment to maintain system reliability. This simulation study will use a BbNN to predict future values of ambient luminosity levels in a room. The network will be pretrained offline to predict ambient luminosity levels in an ideal setup and then deployed in the test room. The actual ambient luminosity levels in the test room can be different from the training data due to various external factors such as a sunny or a cloudy day, number of open windows, and closed or open curtains. The actual levels can be recorded in real time using light sensors and used for online training of the BbNN predictor to improve its future predictions. This study could be applied to many applications sensitive to luminosity variations such as embryonic cell or plant tissue cultures.

5.3.1. Offline Training Experimental Setup and Results. The pretraining setup for our experiment is as follows. Figure 21(a) shows the normalized values of the ambient luminosity variations during the course of a day for the simulated training room. The plot also shows the training result for the BbNN. Figure 21(b) shows the training error. The training dataset for the network consists of past four luminosity observations as the inputs and the next luminosity level as the target output. Figure 22 shows the average and maximum fitness trends over the course of genetic evolution and the evolution parameters used for the training. Figure 23 shows the evolved network.

5.3.2. Online Training Experimental Setup and Results. The evolved network from the previous step is deployed in the simulated test room. Two case studies are considered that simulate the ambient luminosity variations in the test room. The first represents a cloudy day with lower ambient luminosity levels as compared to the ones considered in the offline training step and the second represents a sunny day with higher luminosity levels. These are shown in Figure 24.

The deployed BbNN platform is set up to trigger an online retraining cycle on observing greater than 5% prediction error. For the cloudy day test case the BbNN predicts the ambient luminosity reasonably well until 7:50 hours when the first retraining trigger is issued. The second retraining

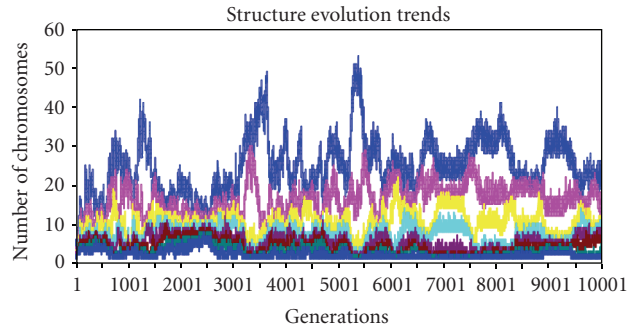


Figure 19: BbNN structure evolution trends for Iris data classification. Each color represents a unique BbNN structure.

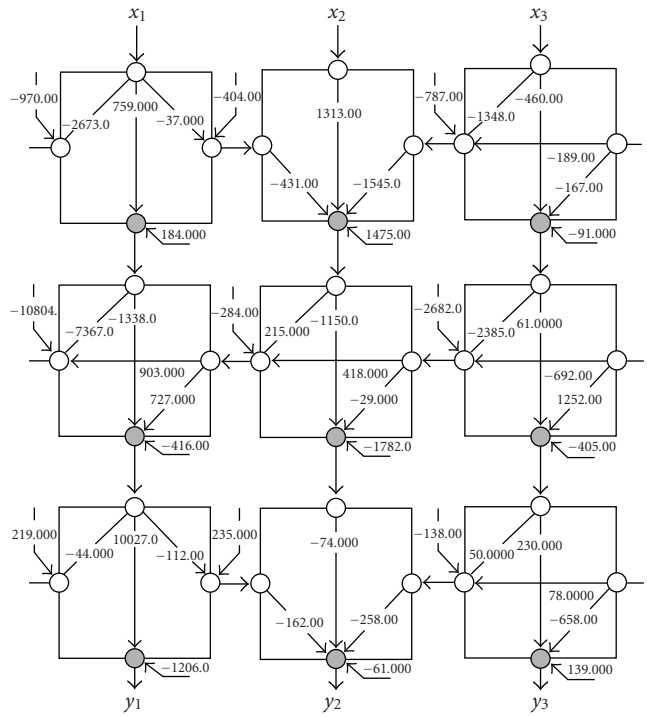


Figure 20: Evolved network for Iris data classification.

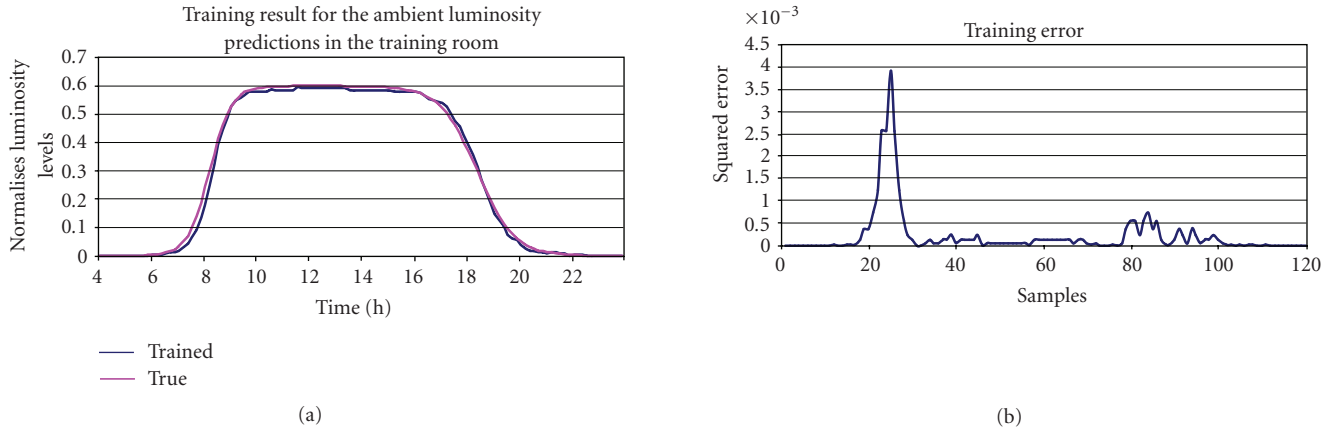
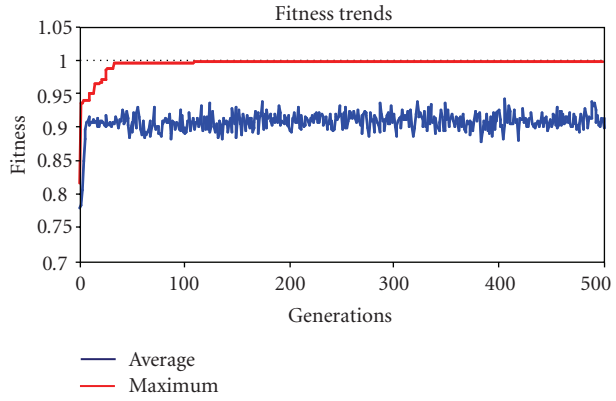


Figure 21: Pretraining result: (a) actual and predicted luminosity levels, and (b) training error.



(a)

Parameter	Value
Activation function	Hypergolic tangent function
Selection strategy	Tournament selection
Population size	80
Maximum generations	2000
Structure crossover probability	0.7
Structure mutation probability	0.3
Weight mutation probability	0.3
Number of patterns	120
Inputs per pattern	4
Evolution strategy	Ellitist evolution

(b)

Figure 22: (a) Average and maximum fitness values. (b) GA parameters used for training.

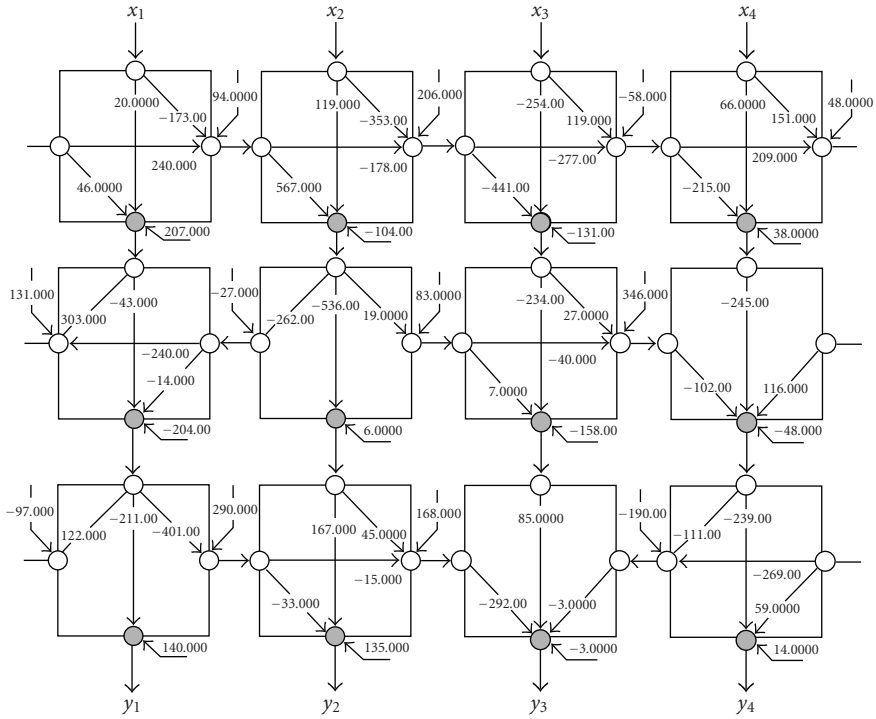


Figure 23: Evolved network after pre-training.

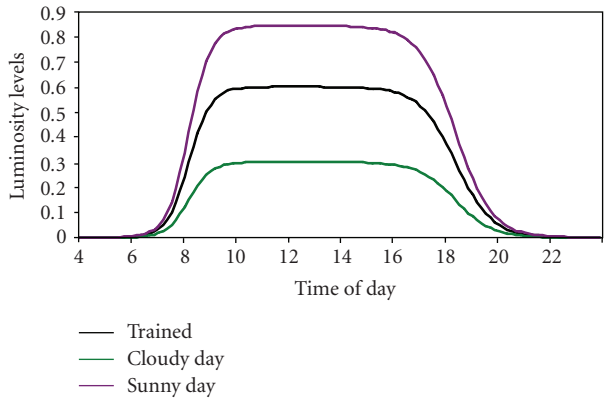


Figure 24: Luminosity levels used for offline training, cloudy day, and sunny day test cases.

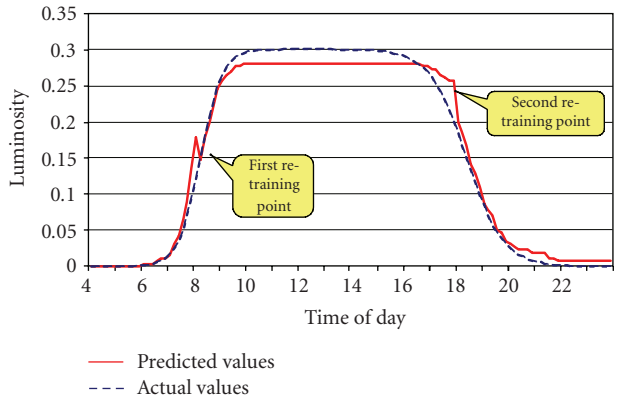


Figure 25: Online evolution operation for the cloudy day showing the two trigger points.

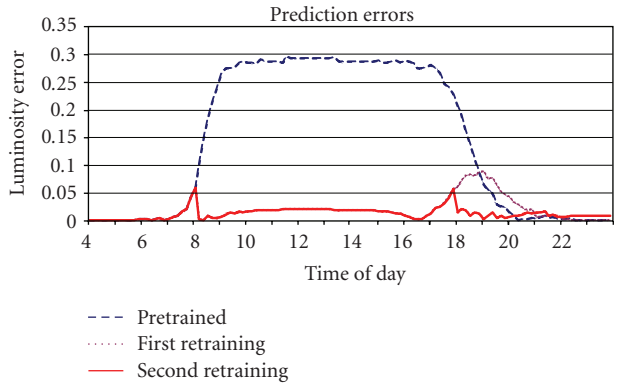


Figure 26: Prediction errors with and without online re-training.

Table 2: Device Utilization Summary on Xilinx Virtex-II Pro FPGA (XC2VP30).

Network size	Number of slice registers		Number of block RAMs		Number of MULT18 × 18s	
	Used	Utilization	Used	Utilization	Used	Utilization
2 × 2	2724	19%	8	5%	12	8%
2 × 4	4929	35%	16	11%	24	17%
2 × 6	7896	57%	24	17%	36	26%
2 × 8	10589	77%	32	23%	48	35%
2 × 10	12408	90%	40	29%	60	44%
3 × 2	3661	26%	8	5%	18	13%
3 × 4	7327	53%	16	11%	36	26%
3 × 6	11025	80%	24	17%	54	39%
3 × 8	14763	107%	32	23%	72	52%
3 × 10	18456	134%	40	29%	90	66%
4 × 2	4783	34%	8	5%	24	17%
4 × 4	9646	70%	16	11%	48	35%
4 × 6	14587	106%	24	17%	72	52%
4 × 8	19508	142%	32	23%	96	70%
4 × 10	24461	178%	40	29%	120	88%

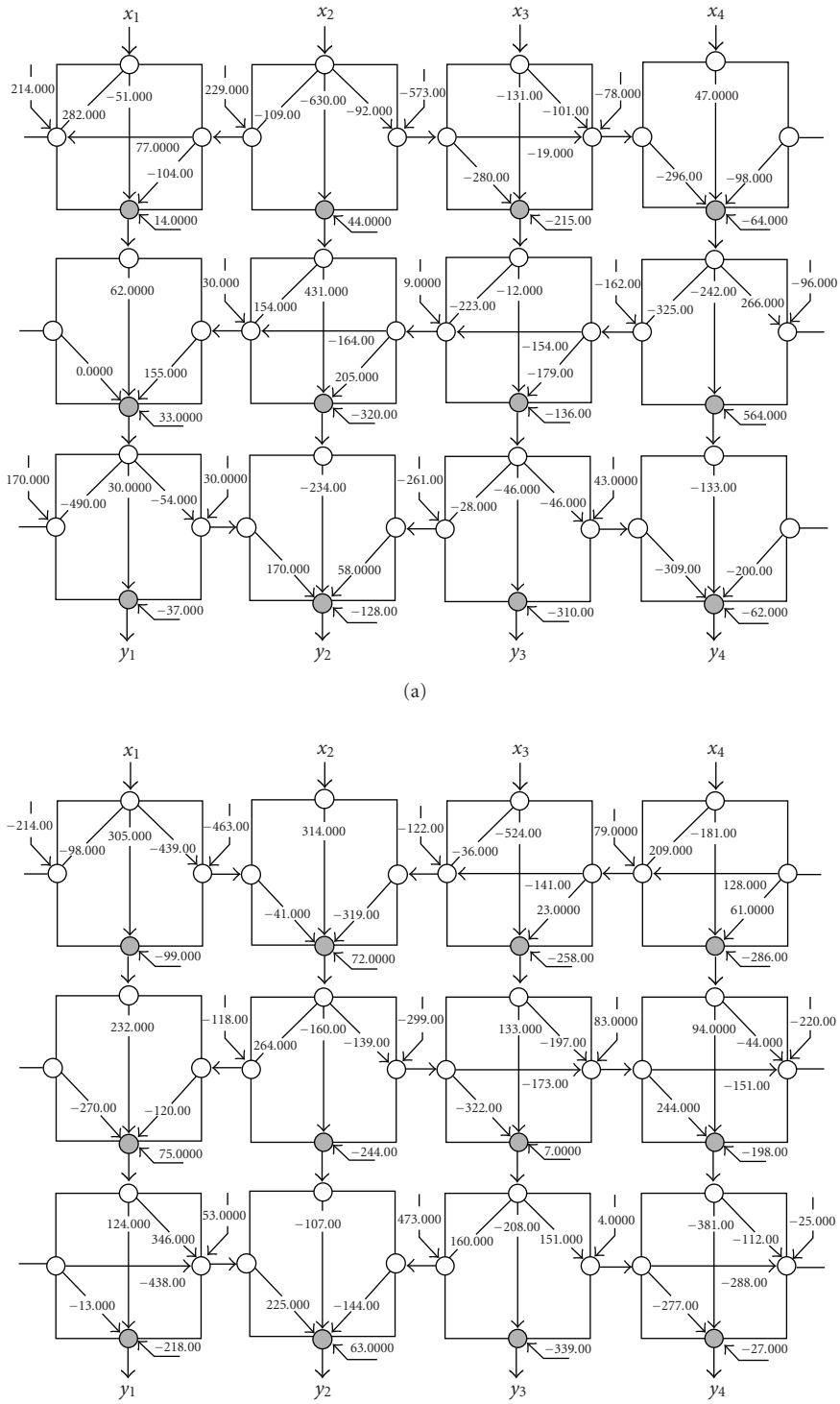


Figure 27: Evolved networks after retraining triggers: (a) first trigger point and (b) second trigger point.

trigger is issued at 17:50 hours. The actual and predicted luminosity values and the corresponding trigger points are shown in Figure 25. The prediction errors with and without online retraining are shown in Figure 26. Evolved networks after the first and second retraining cycles are shown in Figure 27.

In the sunny day test case the pretrained network performs poorly and requires eight retraining trigger points as shown in Figure 28. Figure 29 shows the prediction errors with and without retraining cycles. It can be seen that the network fails to correctly predict the transient rise in the luminosity level and over estimates after the first few

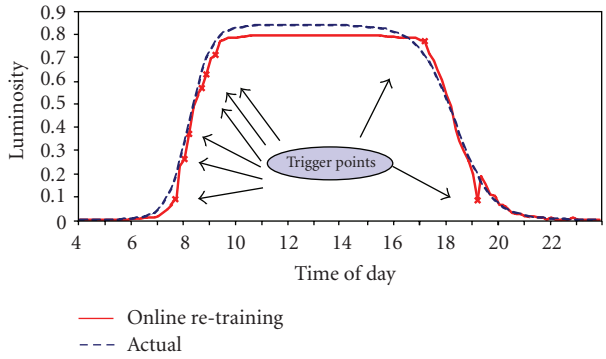


Figure 28: Online training result for the sunny day test case.

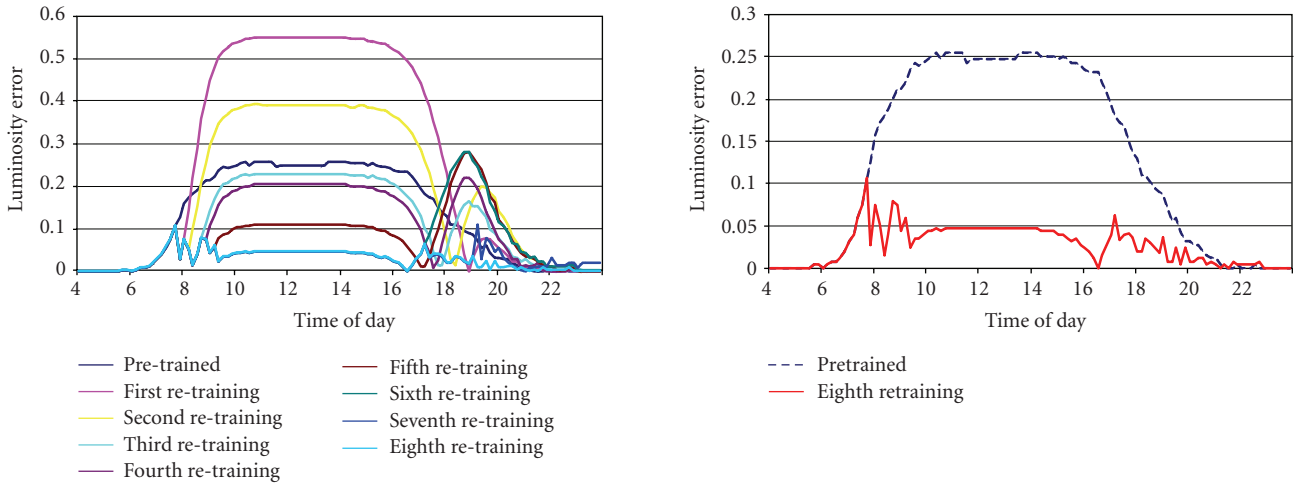


Figure 29: Prediction errors with and without retraining cycles for the sunny day test case.

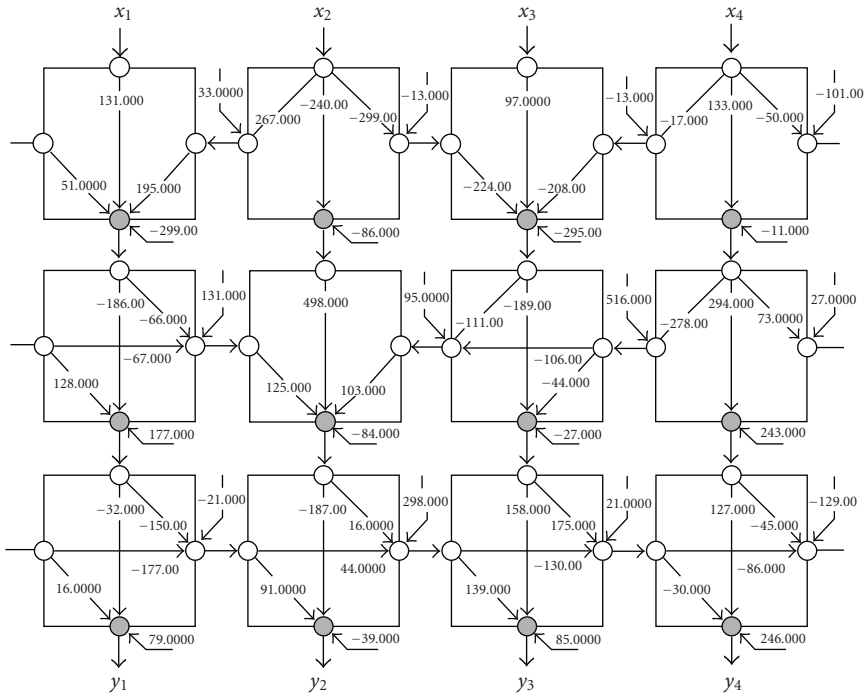


Figure 30: Evolved network after the eighth re-training cycle for the sunny day test case.

Table 3: Device utilization summary on Xilinx Virtex-II pro FPGA (XC2VP70).

Network size	Number of slice registers		Number of block RAMs		Number of MULT18 × 18s	
	Used	Utilization	Used	Utilization	Used	Utilization
2 × 2	2497	7%	8	2%	12	3%
2 × 4	4929	14%	16	4%	24	7%
2 × 6	7390	22%	24	7%	36	10%
2 × 8	9915	29%	32	9%	48	14%
2 × 10	12403	37%	40	12%	60	18%
3 × 2	3661	11%	8	2%	18	5%
3 × 4	7327	22%	16	4%	36	10%
3 × 6	11025	33%	24	7%	54	16%
3 × 8	14788	44%	32	39%	72	9%
3 × 10	18461	55%	40	12%	90	27%
3 × 12	22233	67%	48	14%	108	33%
3 × 14	25652	77%	56	17%	126	38%
3 × 16	29254	88%	64	19%	144	43%
4 × 2	4783	14%	8	2%	24	7%
4 × 4	9646	29%	16	4%	48	14%
4 × 6	14561	44%	24	7%	72	21%
4 × 8	19534	59%	32	9%	96	29%
4 × 10	24470	73%	40	12%	120	36%
4 × 12	29221	88%	48	14%	144	43%
4 × 14	34389	103%	56	17%	168	51%

retraining cycles necessitating multiple retrains. Figure 30 shows the evolved network after the eighth re-training cycle.

This case study demonstrates the online training benefits of the BbNN platform and its potential for applications in dynamic environments. Figures 26 and 29 demonstrate the performance of the network with and without re-training. Our study has not taken into consideration the time required to retrain the network. This time is dependent on the actual platform setup which can vary depending on the implementation constraints. Our design prototype currently uses the on-chip PPC 405 processor for executing the GA operators. For rapid convergence of the GA algorithm faster processors can be used with BbNN core executing on FPGAs for speeding up fitness evaluations.

6. Conclusions

In this paper we present an FPGA design for BbNNs that is capable of on-chip training under the control of genetic algorithms. Typical design process for ANNs involves a training phase performed using software simulations and the obtained network is designed and deployed offline. Hence, the training and design processes have to be repeated offline for every new application of ANNs. The novel online adaptability features of our design demonstrated using several case studies expand the potential applications for BbNNs to dynamic environments and provide increased deployment lifetimes and improved system reliability. The platform has been prototyped on two FPGA boards: a stand-alone Digilent Inc. XUP development board [98] and an Amirix AP130 development board [99], each with a Xilinx Virtex-II Pro FPGA. The ANN design can achieve peak computational capacity of 147 MCPS per neuron block on the Virtex-II Pro FPGAs. The paper presents three case studies. The first two, the n-bit parity classification and the Iris data classification, demonstrate the functionality of the designed platform. The case study on adaptive forward prediction demonstrates the benefits of online evolution under dynamically changing conditions. This work provides a platform for further research on design scalability, online unsupervised training algorithms, and applications of BbNNs in dynamic environments. To further speedup the evolution process, parallel GA algorithms can be used that can take advantage of multiple on-chip PowerPC processors per FPGA as well as scaling the design across multiple FPGAs. These are topics for further research.

Acknowledgments

This work was supported in part by the National Science Foundation under Grant nos. ECS-0319002 and CCF-0311500. The authors also acknowledge the support of the UT Exhibit, Performance, and Publication Expense Fund

and thank the reviewers for their valuable comments which helped them in improving this manuscript.

References

[1] S. G. Merchant and G. D. Peterson, "An evolvable artificial neural network platform for dynamic environments," in *Proceedings of the 51st Midwest Symposium on Circuits and Systems (MWSCAS '08)*, pp. 77–80, Knoxville, Tenn, USA, August 2008.

[2] H. de Garis, "Evolvable Hardware: Principles and Practice," *Communications of the Association for Computer Machinery (CACM Journal)*, August 1997.

[3] J. N. H. Heemskerk, "Overview of neural hardware," in *Neurocomputers for Brain-Style Processing: Design, Implementation and Application*, Unit of Experimental and Theoretical Psychology, Leiden University, Leiden, The Netherlands, 1995.

[4] H. P. Graf and L. D. Jackel, "Advances in neural network hardware," in *Proceedings of the International Electron Devices Meeting (IEDM '88)*, pp. 766–769, San Francisco, Calif, USA, December 1988.

[5] D. R. Collins and P. A. Penz, "Considerations for neural network hardware implementations," in *Proceedings of the 22nd IEEE International Symposium on Circuits and Systems (ISCAS '89)*, vol. 2, pp. 834–836, Portland, Ore, USA, May 1989.

[6] P. Ienne, "Architectures for neuro-computers: review and performance evaluation," Tech. Rep. 93/21, Swiss Federal Institute of Technology, Zurich, Switzerland, 1993.

[7] P. Ienne and G. Kuhn, "Digital systems for neural networks," in *Digital Signal Processing Technology*, P. Papamichalis and R. Kerwin, Eds., vol. 57, SPIE Optical Engineering, Orlando, Fla, USA, 1995.

[8] I. Aybay, S. Cetinkaya, and U. Halici, "Classification of neural network hardware," *Neural Network World*, vol. 6, no. 1, pp. 11–29, 1996.

[9] H. C. Card, G. K. Rosendahl, D. K. McNeill, and R. D. McLeod, "Competitive learning algorithms and neurocomputer architecture," *IEEE Transactions on Computers*, vol. 47, no. 8, pp. 847–858, 1998.

[10] T. Schoenauer, A. Jahnke, U. Roth, and H. Klar, "Digital neurohardware: principles and perspectives," in *Proceedings of the 3rd International Workshop on Neural Networks in Applications (NN '98)*, Magdeburg, Germany, February 1998.

[11] L. M. Reyneri, "Theoretical and implementation aspects of pulse streams: an overview," in *Proceedings of the 7th International Conference on Microelectronics for Neural, Fuzzy and Bio-Inspired Systems*, pp. 78–89, Granada, Spain, April 1999.

[12] B. Linares-Barranco, A. G. Andreou, G. Indiveri, and T. Shibata, "Special issue on neural networks hardware implementations," *IEEE Transactions on Neural Networks*, vol. 14, no. 5, pp. 976–979, 2003.

[13] J. Zhu and P. Sutton, "FPGA implementations of neural networks—a survey of a decade of progress," in *Proceedings of the 13th International Conference on Field-Programmable Logic and Applications (FPL '03)*, pp. 1062–1066, Lisbon, Portugal, September 2003.

[14] N. Aibe, M. Yasunaga, I. Yoshihara, and J. H. Kim, "A probabilistic neural network hardware system using a learning-parameter parallel architecture," in *Proceedings of the International Joint Conference on Neural Networks (IJCNN '02)*, vol. 3, pp. 2270–2275, Honolulu, Hawaii, USA, May 2002.

[15] J. L. Ayala, A. G. Lomeña, M. López-Vallejo, and A. Fernández, "Design of a pipelined hardware architecture for real-time neural network computations," in *Proceedings of the 45th Midwest Symposium on Circuits and Systems (MWSCAS '02)*, vol. 1, pp. 419–422, Tulsa, Okla, USA, August 2002.

[16] U. Ramacher, "Synapse-X: a general-purpose neurocomputer architecture," in *Proceedings of IEEE International Joint Conference on Neural Networks (IJCNN '91)*, vol. 3, pp. 2168–2176, Seattle, Wash, USA, July 1991.

[17] M. Moussa, S. Areibi, and K. Nichols, "On the arithmetic precision for implementing back-propagation networks on FPGA: a case study," in *FPGA Implementations of Neural Networks*, A. R. Omondi and J. C. Rajapakse, Eds., pp. 37–61, Springer, Berlin, Germany, 2006.

[18] K. R. Nichols, M. A. Moussa, and S. M. Areibi, "Feasibility of floating-point arithmetic in FPGA based artificial neural networks," in *Proceedings of the 15th International Conference on Computer Applications in Industry and Engineering (CAINE '02)*, San Diego, Calif, USA, November 2002.

[19] J. L. Holt and T. E. Baker, "Back propagation simulations using limited precision calculations," in *Proceedings of the International Joint Conference on Neural Networks (IJCNN '91)*, pp. 121–126, Seattle, Wash, USA, July 1991.

[20] J. L. Holt and J.-N. Hwang, "Finite precision error analysis of neural network electronic hardware implementations," in *Proceedings of the International Joint Conference on Neural Networks (IJCNN'91)*, pp. 519–525, Seattle, Washington, USA, July 1991.

[21] J. L. Holt and J.-N. Hwang, "Finite precision error analysis of neural network hardware implementations," *IEEE Transactions on Computers*, vol. 42, no. 3, pp. 281–290, 1993.

[22] E. Ros, E. M. Ortigosa, R. Agis, R. Carrillo, and M. Arnold, "Real-time computing platform for spiking neurons (RT-spike)," *IEEE Transactions on Neural Networks*, vol. 17, no. 4, pp. 1050–1063, 2006.

[23] M. Porrmann, U. Witkowski, H. Kalte, and U. Ruckert, "Implementation of artificial neural networks on a reconfigurable hardware accelerator," in *Proceedings of the 10th Euromicro Workshop on Parallel, Distributed and Network-Based Processing*, pp. 243–250, Canary Islands, Spain, January 2002.

[24] C. Torres-Huitzil and B. Girau, "FPGA implementation of an excitatory and inhibitory connectionist model for motion perception," in *Proceedings of IEEE International Conference on Field Programmable Technology (FPT '05)*, pp. 259–266, Singagore, December 2005.

[25] S. Kothandaraman, "Implementation of block-based neural networks on reconfigurable computing platforms," MS Report, Electrical and Computer Engineering Department, University of Tennessee, Knoxville, Tenn, USA, 2004.

[26] D. Ferrer, R. González, R. Fleitas, J. P. Acle, and R. Canetti, "NeuroFPGA—implementing artificial neural networks on programmable logic devices," in *Proceedings of the Design, Automation and Test in Europe Conference and Exhibition (DATE '04)*, vol. 3, pp. 218–223, Paris, France, February 2004.

[27] C. T. Yen, W.-D. Weng, and Y. T. Lin, "FPGA realization of a neural-network-based nonlinear channel equalizer," *IEEE Transactions on Industrial Electronics*, vol. 51, no. 2, pp. 472–479, 2004.

[28] Q. Wang, B. Yi, Y. Xie, and B. Liu, "The hardware structure design of perceptron with FPGA implementation," in *Proceedings of IEEE International Conference on Systems, Man and Cybernetics*, vol. 1, pp. 762–767, Washington, DC, USA, October 2003.

[29] M. M. Syiam, H. M. Klash, I. I. Mahmoud, and S. S. Haggag, "Hardware implementation of neural network on FPGA for accidents diagnosis of the multi-purpose research reactor of Egypt," in *Proceedings of the 15th International Conference on Microelectronics (ICM '03)*, pp. 326–329, Cairo, Egypt, December 2003.

[30] M. Krips, T. Lammert, and A. Kummert, "FPGA implementation of a neural network for a real-time hand tracking system," in *Proceedings of the 1st IEEE International Workshop on Electronic Design, Test and Applications*, pp. 313–317, Christchurch, New Zealand, January 2002.

[31] J. Zhu, G. J. Milne, and B. K. Gunther, "Towards an FPGA based reconfigurable computing environment for neural network implementations," in *Proceedings of the 9th International Conference on Artificial Neural Networks (ICANN '99)*, vol. 2, pp. 661–666, Edinburgh, UK, September 1999.

[32] S. Happe and H.-G. Kranz, "Practical applications for the machine intelligent partial discharge disturbing pulse suppression system NeuroTEK II," in *Proceedings of the 11th International Symposium on High Voltage Engineering (ISH '99)*, vol. 5, pp. 37–40, London, UK, August 1999.

[33] M. Marchesi, G. Orlandi, F. Piazza, and A. Uncini, "Fast neural networks without multipliers," *IEEE Transactions on Neural Networks*, vol. 4, no. 1, pp. 53–62, 1993.

[34] M. van Daalen, T. Kosel, P. Jeavons, and J. Shawe-Taylor, "Emergent activation functions from a stochastic bit-stream neuron," *Electronics Letters*, vol. 30, no. 4, pp. 331–333, 1994.

[35] E. van Keulen, S. Colak, H. Withagen, and H. Hegt, "Neural network hardware performance criteria," in *Proceedings of IEEE International Conference on Neural Networks*, vol. 3, pp. 1955–1958, Orlando, Fla, USA, June 1994.

[36] H. O. Johansson, P. Larsson, P. Larsson-Edefors, and C. Svensson, "A 200-MHz CMOS bit-serial neural network," in *Proceedings of the 7th Annual IEEE International ASIC Conference and Exhibit*, pp. 312–315, Rochester, NY, USA, September 1994.

[37] M. Gschwind, V. Salapura, and O. Maischbergeres, "Space efficient neural net implementation," in *Proceedings of the 2nd International ACM/SIGDA Workshop on Field-Programmable Gate Arrays*, Berkeley, Calif, USA, February 1994.

[38] A. F. Murray and A. V. W. Smith, "Asynchronous VLSI neural networks using pulse-stream arithmetic," *IEEE Journal of Solid-State Circuits*, vol. 23, no. 3, pp. 688–697, 1988.

[39] P. Lysaght, J. Stockwood, J. Law, and D. Girma, "Artificial neural network implementation on a fine-grained FPGA," in *Proceedings of the 4th International Workshop on Field-Programmable Logic and Applications (FPL '94)*, pp. 421–431, Prague, Czech Republic, September 1994.

[40] V. Salapura, "Neural networks using bit stream arithmetic: a space efficient implementation," in *Proceedings of IEEE International Symposium on Circuits and Systems (ISCAS '94)*, vol. 6, pp. 475–478, London, UK, May 1994.

[41] N. Chujo, S. Kuroyanagi, S. Doki, and S. Okuma, "An iterative calculation method of neuron model with sigmoid function," in *Proceedings of IEEE International Conference on Systems, Man and Cybernetics*, vol. 3, pp. 1532–1537, Tucson, Ariz, USA, October 2001.

[42] S. A. Guccione and M. J. Gonzalez, "Neural network implementation using reconfigurable architectures," in *Selected Papers from the Oxford 1993 International Workshop on Field Programmable Logic and Applications on More FPGAs*, pp. 443–451, Abingdon EE & CS Books, Oxford, UK, 1994.

[43] L. Mintzer, "Digital filtering in FPGAs," in *Proceedings of the 28th Asilomar Conference on Signals, Systems and Computers*, vol. 2, pp. 1373–1377, Pacific Grove, Calif, USA, November 1994.

[44] T. Szabo, L. Antoni, G. Horvath, and B. Feher, "A full-parallel digital implementation for pre-trained NNs," in *Proceedings of the International Joint Conference on Neural Networks (IJCNN '00)*, vol. 2, pp. 49–54, Como, Italy, July 2000.

[45] B. Noory and V. Groza, "A reconfigurable approach to hardware implementation of neural networks," in *Proceedings of the Canadian Conference on Electrical and Computer Engineering*, vol. 3, pp. 1861–1864, Montreal, Canada, May 2003.

[46] E. Pasero and M. Perri, "Hw-Sw codesign of a flexible neural controller through a FPGA-based neural network programmed in VHDL," in *Proceedings of IEEE International Joint Conference on Neural Networks (IJCNN '04)*, vol. 4, pp. 3161–3165, Budapest, Hungary, July 2004.

[47] C.-H. Kung, M. J. Devaney, C.-M. Kung, C.-M. Huang, Y.-J. Wang, and C.-T. Kuo, "The VLSI implementation of an artificial neural network scheme embedded in an automated inspection quality management system," in *Proceedings of the 19th IEEE Instrumentation and Measurement Technology Conference (IMTC '02)*, vol. 1, pp. 239–244, Anchorage, Alaska, USA, May 2002.

[48] J. G. Eldredge and B. L. Hutchings, "Density enhancement of a neural network using FPGAs and run-time reconfiguration," in *Proceedings of IEEE Workshop on FPGAs for Custom Computing Machines*, pp. 180–188, Napa Valley, Calif, USA, April 1994.

[49] J. G. Eldredge and B. L. Hutchings, "RRANN: a hardware implementation of the backpropagation algorithm using reconfigurable FPGAs," in *Proceedings of IEEE International Conference on Neural Networks, IEEE World Congress on Computational Intelligence*, vol. 4, pp. 2097–2102, Orlando, Fla, USA, June 1994.

[50] C. E. Cox and W. E. Blanz, "GANGLION—a fast field-programmable gate array implementation of a connectionist classifier," *IEEE Journal of Solid-State Circuits*, vol. 27, no. 3, pp. 288–299, 1992.

[51] A. Perez-Uribe and E. Sanchez, "FPGA implementation of an adaptable-size neural network," in *Proceedings of the 6th International Conference on Artificial Neural Networks (ICANN '96)*, pp. 383–388, Bochum, Germany, July 1996.

[52] H. F. Restrepo, R. Hoffmann, A. Perez-Uribe, C. Teuscher, and E. Sanchez, "A networked FPGA-based hardware implementation of a neural network application," in *Proceedings of IEEE Symposium on Field-Programmable Custom Computing Machines*, pp. 337–338, Napa Valley, Calif, USA, April 2000.

[53] I. Kajitani, M. Murakawa, D. Nishikawa, et al., "An evolvable hardware chip for prosthetic hand controller," in *Proceedings of the 7th International Conference on Microelectronics for Neural, Fuzzy and Bio-Inspired Systems (MicroNeuro '99)*, pp. 179–186, Granada, Spain, April, 1999.

[54] K. Mathia and J. Clark, "On neural network hardware and programming paradigms," in *Proceedings of the International Joint Conference on Neural Networks (IJCNN '02)*, vol. 3, pp. 2692–2697, Honolulu, Hawaii, USA, June 2002.

[55] D. Hajtas and D. Durackova, "The library of building blocks for an "integrate & fire" neural network on a chip," in *Proceedings of IEEE International Joint Conference on Neural Networks (IJCNN '04)*, vol. 4, pp. 2631–2636, Budapest, Hungary, July 2004.

[56] Jayadeva and S. A. Rahman, "A neural network with O(N) neurons for ranking N numbers in O(1/N) time," *IEEE Transactions on Circuits and Systems I*, vol. 51, no. 10, pp. 2044–2051, 2004.

[57] J. D. Hadley and B. L. Hutchings, "Design methodologies for partially reconfigured systems," in *Proceedings of IEEE Symposium on FPGAs for Custom Computing Machines*, pp. 78–84, Napa Valley, Calif, USA, April 1995.

[58] R. Gadea, J. Cerda, F. Ballester, and A. Macholi, "Artificial neural network implementation on a single FPGA of a pipelined on-line backpropagation," in *Proceedings of the 13th International Symposium on System Synthesis*, pp. 225–230, Madrid, Spain, September 2000.

[59] K. Paul and S. Rajopadhye, "Back-propagation algorithm achieving 5 Gops on the Virtex-E," in *FPGA Implementations of Neural Networks*, pp. 137–165, Springer, Berlin, Germany, 2006.

[60] U. Witkowski, T. Neumann, and U. Ruckert, "Digital hardware realization of a hyper basis function network for on-line learning," in *Proceedings of the 7th International Conference on Microelectronics for Neural, Fuzzy and Bio-Inspired Systems (MicroNeuro '99)*, pp. 205–211, Granada, Spain, April 1999.

[61] M. Murakawa, S. Yoshizawa, I. Kajitani, et al., "The GRD chip: genetic reconfiguration of DSPs for neural network processing," *IEEE Transactions on Computers*, vol. 48, no. 6, pp. 628–639, 1999.

[62] D. Hammerstrom, "A VLSI architecture for high-performance, low-cost, on-chip learning," in *Proceedings of the International Joint Conference on Neural Networks (IJCNN '90)*, vol. 2, pp. 537–544, San Diego, Calif, USA, June 1990.

[63] Y. Sato, K. Shibata, M. Asai, et al., "Development of a high-performance, general purpose neuro-computer composed of 512 digital neurons," in *Proceedings of the International Joint Conference on Neural Networks (IJCNN '93)*, vol. 2, pp. 1967–1970, Nagoya, Japan, October 1993.

[64] T. Tang, O. Ishizuka, and H. Matsumoto, "Backpropagation learning in analog T-model neural network hardware," in *Proceedings of the International Joint Conference on Neural Networks (IJCNN '93)*, vol. 1, pp. 899–902, Nagoya, Japan, October 1993.

[65] B. Linares-Barranco, E. Sanchez-Sinencio, A. Rodriguez-Vazquez, and J. L. Huertas, "A CMOS analog adaptive BAM with on-chip learning and weight refreshing," *IEEE Transactions on Neural Networks*, vol. 4, no. 3, pp. 445–455, 1993.

[66] E. Farquhar, C. Gordon, and P. Hasler, "A field programmable neural array," in *Proceedings of IEEE International Symposium on Circuits and Systems (ISCAS '06)*, pp. 4114–4117, Kos, Greece, May 2006.

[67] F. Tenore, R. J. Vogelstein, R. Etienne-Cummings, G. Cauwenberghs, M. A. Lewis, and P. Hasler, "A spiking silicon central pattern generator with floating gate synapses [robot control applications]," in *Proceedings of IEEE International Symposium on Circuits and Systems (ISCAS '05)*, vol. 4, pp. 4106–4109, Kobe, Japan, May 2005.

[68] G. Indiveri, E. Chicca, and R. J. Douglas, "A VLSI reconfigurable network of integrate-and-fire neurons with spike-based learning synapses," in *Proceedings of the 12th European Symposium on Artificial Neural Networks (ESANN '04)*, Bruges, Belgium, April 2004.

[69] B. Girau, "FPNA: applications and implementations," in *FPGA Implementations of Neural Networks*, pp. 103–136, Springer, Berlin, Germany, 2006.

[70] B. Girau, "FPNA: concepts and properties," in *FPGA Implementations of Neural Networks*, pp. 63–101, Springer, Berlin, Germany, 2006.

[71] M. Holler, S. Tam, H. Castro, and R. Benson, "An electrically trainable artificial neural network (ETANN) with 10240 'floating gate' synapses," in *Artificial Neural Networks: Electronic Implementations*, pp. 50–55, IEEE, New York, NY, USA, 1990.

[72] M. L. Mumford, D. K. Andes, and L. L. Kern, "The Mod 2 Neurocomputer system design," *IEEE Transactions on Neural Networks*, vol. 3, no. 3, pp. 423–433, 1992.

[73] T. Schmitz, S. Hohmann, K. Meier, J. Schemmel, and F. Schurmann, "Speeding up hardware evolution: a coprocessor for evolutionary algorithms," in *Evolvable Systems: From Biology to Hardware*, vol. 2606 of *Lecture Notes in Computer Science*, pp. 274–285, Springer, Berlin, Germany, 2003.

[74] A. Omondi, J. Rajapakse, and M. Bajger, "FPGA neurocomputers," in *FPGA Implementations of Neural Networks*, pp. 1–36, Springer, Berlin, Germany, 2006.

[75] L. D. Jackel, H. P. Graf, and R. E. Howard, "Electronic neural network chips," *Applied Optics*, vol. 26, no. 23, pp. 5077–5080, 1987.

[76] E. Farquhar and P. Hasler, "A bio-physically inspired silicon neuron," *IEEE Transactions on Circuits and Systems I*, vol. 52, no. 3, pp. 477–488, 2005.

[77] B. Linares-Barranco, E. Sanchez-Sinencio, A. Rodriguez-Vazquez, and J. L. Huertas, "A modular T-mode design approach for analog neural network hardware implementations," *IEEE Journal of Solid-State Circuits*, vol. 27, no. 5, pp. 701–713, 1992.

[78] C. Gordon, E. Farquhar, and P. Hasler, "A family of floating-gate adapting synapses based upon transistor channel models," in *Proceedings of IEEE International Symposium on Circuits and Systems (ISCAS '04)*, vol. 1, pp. 317–320, Vancouver, Canada, May 2004.

[79] E. Farquhar, D. Abramson, and P. Hasler, "A reconfigurable bidirectional active 2 dimensional dendrite model," in *Proceedings of IEEE International Symposium on Circuits and Systems (ISCAS '04)*, vol. 1, pp. 313–316, Vancouver, Canada, May 2004.

[80] G. Cauwenberghs, C. F. Neugebauer, and A. Yariv, "An adaptive CMOS matrix-vector multiplier for large scale analog hardware neural network applications," in *Proceedings of the International Joint Conference on Neural Networks (IJCNN '91)*, vol. 1, pp. 507–511, Seattle, Wash, USA, July 1991.

[81] O. Barkan, W. R. Smith, and G. Persky, "Design of coupling resistor networks for neural network hardware," *IEEE Transactions on Circuits and Systems*, vol. 37, no. 6, pp. 756–765, 1990.

[82] T. J. Schwartz, "A neural chips survey," *AI Expert*, vol. 5, no. 12, pp. 34–38, 1990.

[83] A. J. Agranat, C. F. Neugebauer, and A. Yariv, "A CCD based neural network integrated circuit with 64 K analog programmable synapses," in *Proceedings of the International Joint Conference on Neural Networks (IJCNN '90)*, vol. 2, pp. 551–555, San Diego, Calif, USA, June 1990.

[84] L. W. Massengill and D. B. Mundie, "An analog neural hardware implementation using charge-injection multipliers and neuron-specific gain control," *IEEE Transactions on Neural Networks*, vol. 3, no. 3, pp. 354–362, 1992.

[85] A. Passos Almeida and J. E. Franca, "A mixed-mode architecture for implementation of analog neural networks with digital programmability," in *Proceedings of the International Joint Conference on Neural Networks (IJCNN '93)*, vol. 1, pp. 887–890, Nagoya, Japan, October 1993.

[86] M. R. DeYong, R. L. Findley, and C. Fields, "The design, fabrication, and test of a new VLSI hybrid analog-digital neural processing element," *IEEE Transactions on Neural Networks*, vol. 3, no. 3, pp. 363–374, 1992.

[87] E. Sackinger, B. E. Boser, J. Bromley, Y. LeCun, and L. D. Jackel, "Application of the ANNA neural network chip to high-speed character recognition," *IEEE Transactions on Neural Networks*, vol. 3, no. 3, pp. 498–505, 1992.

[88] G. Zatorre-Navarro, N. Medrano-Marqués, and S. Celma-Pueyo, "Analysis and simulation of a mixed-mode neuron architecture for sensor conditioning," *IEEE Transactions on Neural Networks*, vol. 17, no. 5, pp. 1332–1335, 2006.

[89] K. D. Maier, C. Beckstein, R. Blickhan, W. Erhard, and D. Fey, "A multi-layer-perceptron neural network hardware based on 3D massively parallel optoelectronic circuits," in *Proceedings of the 6th International Conference on Parallel Interconnects*, pp. 73–80, Anchorage, Alaska, USA, October 1999.

[90] M. P. Craven, K. M. Curtis, and B. R. Hayes-Gill, "Consideration of multiplexing in neural network hardware," *IEE Proceedings: Circuits, Devices and Systems*, vol. 141, no. 3, pp. 237–240, 1994.

[91] S.-W. Moon and S.-G. Kong, "Block-based neural networks," *IEEE Transactions on Neural Networks*, vol. 12, no. 2, pp. 307–317, 2001.

[92] W. Jiang, S. G. Kong, and G. D. Peterson, "ECG signal classification using block-based neural networks," in *Proceedings of the International Joint Conference on Neural Networks (IJCNN '05)*, vol. 1, pp. 326–331, Montreal, Canada, July 2005.

[93] W. Jiang, S. G. Kong, and G. D. Peterson, "Continuous heartbeat monitoring using evolvable block-based neural networks," in *Proceedings of IEEE International Joint Conference on Neural Networks (IJCNN '06)*, pp. 1950–1957, Vancouver, Canada, July 2006.

[94] W. Jiang and S. G. Kong, "Block-based neural networks for personalized ECG signal classification," *IEEE Transactions on Neural Networks*, vol. 18, no. 6, pp. 1750–1761, 2007.

[95] S. G. Kong, "Time series prediction with evolvable block-based neural networks," in *Proceedings of IEEE International Joint Conference on Neural Networks (IJCNN '04)*, vol. 2, pp. 1579–1583, Budapest, Hungary, July 2004.

[96] S. Merchant, G. D. Peterson, S. K. Park, and S. G. Kong, "FPGA implementation of evolvable block-based neural networks," in *Proceedings of IEEE Congress on Evolutionary Computation (CEC '06)*, pp. 3129–3136, Vancouver, Canada, July 2006.

[97] Xilinx, "Virtex-II Pro and Virtex-II Pro X Platform FPGAs: Complete Data Sheet," Product Specification, DS083 (v4.6), March 2007.

[98] Xilinx, "Xilinx University Program Virtex-II Pro Development System Hardware Reference Manual," Hardware Reference Manual, UG069 (v1.0), March 2005.

[99] Amirix, "AMIRIX Systems Inc. PCI Platform FPGA Development Board Users Guide," User Guide, DOC-003266 Version 06, June 2004.

[100] R. Fisher, "The use of multiple measurements in taxonomic problems," *Annals of Eugenics*, vol. 7, pp. 179–188, 1936.

Hindawi Publishing Corporation
VLSI Design
Volume 2010, Article ID 930821, 11 pages
doi:10.1155/2010/930821

Research Article

Implementation of Hardware-Accelerated Scalable Parallel Random Number Generators

JunKyu Lee,[1] Gregory D. Peterson,[1] Robert J. Harrison,[2] and Robert J. Hinde[2]

[1] *Department of Electrical Engineering and Computer Science, University of Tennessee, Knoxville, TN 37996, USA*
[2] *Department of Chemistry, University of Tennessee, Knoxville, TN 37996, USA*

Correspondence should be addressed to JunKyu Lee, jlee57@utk.edu

Received 1 June 2009; Revised 13 November 2009; Accepted 21 December 2009

Academic Editor: Ethan Farquhar

The Scalable Parallel Random Number Generators (SPRNGs) library is widely used in computational science applications such as Monte Carlo simulations since SPRNG supports fast, parallel, and scalable random number generation with good statistical properties. In order to accelerate SPRNG, we develop a Hardware-Accelerated version of SPRNG (HASPRNG) on the Xilinx XC2VP50 Field Programmable Gate Arrays (FPGAs) in the Cray XD1 that produces identical results. HASPRNG includes the reconfigurable logic for FPGAs along with a programming interface which performs integer random number generation. To demonstrate HASPRNG for Reconfigurable Computing (RC) applications, we also develop a Monte Carlo π-estimator for the Cray XD1. The RC Monte Carlo π-estimator shows a 19.1× speedup over the 2.2 GHz AMD Opteron processor in the Cray XD1. In this paper we describe the FPGA implementation for HASPRNG and a π-estimator example application exploiting the fine-grained parallelism and mathematical properties of the SPRNG algorithm.

1. Introduction

Random numbers are required in a wide variety of applications such as circuit testing, system simulation, game-playing, cryptography, evaluation of multiple integrals, and computational science Monte Carlo (MC) applications [1].

In particular, MC applications require a huge quantity of high-quality random numbers in order to obtain a high-quality solution [2–4]. To support MC applications effectively, a random number generator should have certain characteristics [5]. First, the random numbers must maintain good statistical properties (e.g., no biases) to guarantee valid results. Second, the generator should have a long period. Third, the random numbers should be reproducible. Fourth, the random number generation should be fast since generating a huge quantity of random numbers requires substantial execution time. Finally, the generator should require little storage to allow the MC application to use the rest of the storage resources.

Many MC applications are embarrassingly parallel [6]. To exploit the parallelism, Parallel Pseudorandom Number Generators (PPRNGs) are required for such applications to achieve fast random number generation [2, 6, 7]. Random numbers from a PPRNG should be statistically independent each other to guarantee a high-quality solution. Many common Pseudorandom Number Generators (PRNGs) fail statistical tests of their randomness [8, 9]; so computational scientists are cautious in selecting PRNG algorithms.

The Scalable Parallel Random Number Generators (SPRNGs) library is one of the best candidates for parallel random number generation satisfying the five characteristics, since it supports fast, scalable, and parallel random number generation with good randomness [2, 7]. SPRNG consists of 6 types of random number generators: Modified Lagged Fibonacci Generator (Modified LFG), 48-bit Linear Congruential Generator with prime addend (48-bit LCG), 64-bit Linear Congruential Generator with prime addend (64-bit LCG), Combined Multiple Recursive Generator (CMRG), Multiplicative Lagged Fibonacci Generator (MLFG), and Prime Modulus Linear Congruential Generator (PMLCG).

We desire to improve the generation speed by implementing a hardware version of random number generators for simulation applications, since generating random numbers takes a considerable amount of the execution time

for applications which require huge quantities of random numbers [10]. FPGAs have several advantages in terms of speedup, energy, power, and flexibility for implementation of the random number generators [11, 12].

High-Performance Reconfigurable Computing (HPRC) platforms employ FPGAs to execute the computationally intensive portion of an application [13, 14]. The Cray XD1 is an HPRC platform providing a flexible interface between the microprocessors and FPGAs (Xilinx XC2VP50 or XC4VLX160). FPGAs are able to communicate with a microprocessor directly through the interface [15].

Therefore, we explore the use of reconfigurable computing to achieve faster random number generation. In order to provide the high-quality, scalable random number generation associated with SPRNG combined with the capabilities of HPRC, we developed the Hardware-Accelerated Scalable Parallel Random Number Generators library (HASPRNGs) that provides bit-equivalent results to operate on a coprocessor FPGA [16–18]. HASPRNG can be used to target computational science applications on arbitrarily large supercomputing systems (e.g., Cray XD-1, XT-5h), subject to FPGA resource availability [15].

Although the computational science application could be executed on the node microprocessors with the FPGAs accelerating the PPRNGs, the HASPRNG implementation could also be colocated with the MC application on the FPGA. The latter approach avoids internal bandwidth constraints and enables more aggressive parallel processing. This presents significant potential benefit from tightly coupling HASPRNG with Reconfigurable Computing Monte Carlo (RC MC) applications. For example, the RC MC π-estimation application can employ a huge quantity of random numbers using as many parallel generators as the hardware resources can support.

In this paper we describe the implementation of the HASPRNG library for the Cray XD1 for the full set of integer random number generators in SPRNG and demonstrate the potential of HASPRNG to accelerate RC MC applications by exploring a π-estimator on the Cray XD1.

2. Implementation

SPRNG includes 6 different types of generators and a number of default parameters [2, 7]. Each of the SPRNG library random number generators and its associated parameter sets are implemented in HASPRNG. The VHSICs (Very High-Speed Integrated Circuits) Hardware Description Language (VHDL) is used for designing the reconfigurable logic of HASPRNG and the RC MC π-estimator. To provide a flexible interface to RC MC applications, a one-bit control input is employed to start and stop HASPRNG operation. When HASPRNG is paused, all the state information inside HASPRNG is kept to enable the resumption of random number generation when needed. Similarly, a one-bit control output signals the availability of a valid random number. These two ports are sufficient for the control interface to a RC MC application developer.

To provide high performance, eight generators are implemented for HASPRNG on the Cray XD1. The modified lagged Fibonacci and multiplicative lagged Fibonacci generators have two implementations to exploit the potential concurrency associated with different parameter sets. We call the eight generators in HASPRNG as follows: Hardware-Accelerated Modified Lagged Fibonacci Generator for Odd-Odd type seed (1: HALFGOO), Hardware-Accelerated Modified Lagged Fibonacci Generator for Odd-Even type seed (2: HALFGOE), Hardware-Accelerated Linear Congruential Generator for 48 bits (3: HALCG48), Hardware-Accelerated Linear Congruential Generator for 64 bits (4: HALCG64), Hardware-Accelerated Combined Multiple Recursive Generator (5: HACMRG), Hardware-Accelerated Multiplicative Lagged Fibonacci Generator for Short lag seed (6: HAMLFGS), Hardware-Accelerated Multiplicative Lagged Fibonacci Generator for Long lag seed (7: HAMLFGL), and Hardware-Accelerated Prime Modulus Linear Congruential Generator (8: HAPMLCG). Since HASPRNG implements the same integer random number generation as SPRNG, every HASPRNG generator returns 32-bit positive integer values (most significant bit equals "0"). In consequence, HASPRNG converts different bit-width random numbers from different generators (e.g., 32 bits for LFG, 48 bits for LCG48, 64 bits for LCG64) to positive 32-bit integer random numbers and generates random numbers from 0 to $(2^{31}\text{-}1)$. For the conversion, HASPRNG masks the upper 31 bits of the different bit-width random numbers and prepends "0" as the most significant bit of the 31-bit data in order to produce positive 32-bit random numbers for all generators in HASPRNG. The techniques to produce 32-bit random numbers are the same as in SPRNG [2, 7]. The random numbers before the conversion are fed back to the generators to produce future random numbers in HASPRNG.

HASPRNG can be used not only for improving the speed of software applications by providing a programming interface but also for accelerating RC MC applications by colocating the application and the hardware accelerated generator(s) on the FPGAs (see Sections 2.6, 2.7, 3.4, and 4).

2.1. Hardware-Accelerated Modifed Lagged Fibonacci Generator (HALFG). The Modified LFG in SPRNG is computed by the XOR bit operation of two additive LFG products [2, 7]. The Modified LFG is expressed by (1), (2), and (3):

$$Z(n) = X'(n) \text{ XOR } Y'(n), \tag{1}$$

$$X(n) = \{X(n-k) + X(n-l)\} \text{Mod } (2^{32}), \tag{2}$$

$$Y(n) = \{Y(n-k) + Y(n-l)\} \text{Mod } (2^{32}), \tag{3}$$

where l and k are called the lags of the generator. The generator follows the convention of $l > k$. $X(n)$ and $Y(n)$ are 32-bit random numbers generated by the two additive LFGs. $X'(n)$ is represented by setting the least significant bit of $X(n)$ to 0. $Y'(n)$ is represented by shifting $Y(n)$ right by one bit. $Z(n)$ is the $(n/2)$th random number of the generator [2, 7]. In SPRNG the generator produces a random number every two step-operations. To provide bit-equivalent results and improve performance, two types of designs are used depending on whether k is odd or even [19, 20]. We employ the design for HALFG as in [19]. The

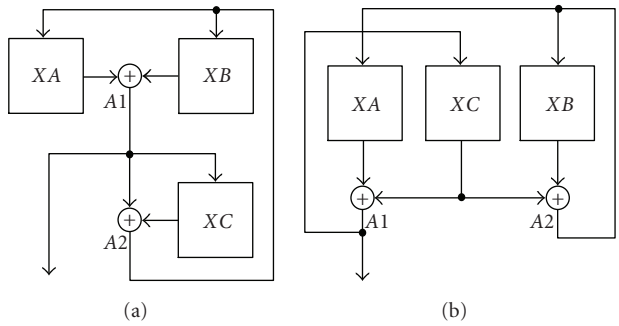


Figure 1: HALFGOO (a) and HALFGOE (b) architectures [19].

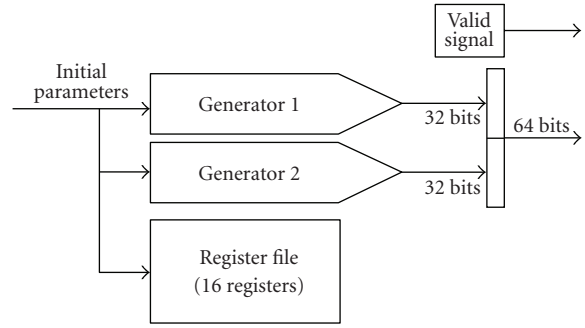


Figure 2: HALCGs architecture.

two types of designs are represented by HALFGOO (both l and k are odd numbers in (1), (2), and (3)) and HALFGOE (l is odd and k is even). The HALFGOO and HALFGOE employ block memory modules in order to store initial seeds and state. Note that small values of k result in data hazards that complicate pipelining. Hence, these designs are optimized for the specific memory access patterns dictated by l and k. Figure 1 shows the HALFGOO and HALFGOE architectures. The block memories are represented by XA, XB, and XC and the results from the two additions are represented by $A1$ and $A2$ in Figure 1. HALFG requires an initialization process to store l previous values in the three block memories, since $X(n)$ and $Y(n)$ in (2) and (3) require the l previous values. Therefore HALFG requires separate buffers of previous values to generate the different random number sequences for $X(n)$ and $Y(n)$.

After the initialization, the seeds are accessed to produce random numbers. For example, $A1$ represents a random number, $X(n)$ or $Y(n)$, and at the same time $A2$ is stored to XA and XB to produce future random numbers. This method is able to generate a random number every two step operations. The HALFGOO and HALFGOE generators produce a random number every clock cycle.

2.2. Hardware-Accelerated 48-bit and 64-bit Linear Congruential Generators (HALCGs). The two LCGs use the same algorithm with different data sizes, 48 bits and 64 bits. The LCG characteristic equation is represented by

$$Z(n) = \{\alpha \times Z(n-1) + p\} \text{Mod } (M), \tag{4}$$

where p is a prime number, α is a multiplier, M is 2^{48} for the 48-bit LCG and 2^{64} for the 64-bit LCG, and $Z(n)$ is the nth random number [2, 7]. The HALCGs must use the previous random number as in (4). Consequently, the HALCGs face data feedback (hazards) in the multiplication which reduces performance. Thanks to the simple recursion relation of (4) we are able to generate a future random number by unrolling the recursion in (4) based on pipeline depths. Equations (5) and (6) represent the modified equations which produce identical results as (4) in the HALCGs. HALCG implementations employing (5) and (6) generate two random numbers every clock cycle using two seven-stage pipelined multipliers:

$$Z(n) = \{\beta \times Z(n-1) + \gamma\} \text{Mod } (M), \tag{5}$$

$$\beta = \alpha^{16}, \qquad \gamma = p \times (\alpha^{15} + \alpha^{14} + \cdots + \alpha^2 + \alpha + 1). \tag{6}$$

The architecture for the HALCGs is shown in Figure 2. Two generation engines are employed to produce two random numbers. One generation engine (Generators 1 in Figure 2) produces the odd indexed random numbers (e.g., $Z(17)$, $Z(19)$, $Z(21)$, ...) and the other generator (Generator 2 in Figure 2) produces the even indexed random numbers (e.g., $Z(16)$, $Z(18)$, $Z(20)$, ...). For the multiplications in (5), we employ built-in 18×18 multipliers inside the generators (Generator 1 and 2 in Figure 2) (see Table 3).

Instead of having internal logic modules to obtain the β and γ in (6), the coefficients are precalculated in software to save hardware resources. Software also calculates 15 initial random numbers ($Z(1)$–$Z(15)$) to provide the initial state to the HALCGs. The pregenerated 15 random numbers and an initial seed are stored in the register file (Register File in Figure 2: $R[0] = Z[0]$ (Initial seed), $R[1] = Z[1], \ldots, R[15] = Z[15]$) having 16 48/64-bit registers during the initialization process. The HALCGs produce 15 random numbers during initialization before they generate random number outputs. In consequence, the generator generates two 32-bit random numbers every clock cycle. We provide two random numbers every clock cycle to exploit the Cray XD1 bandwidth since the Cray XD1 can support a 64 bit data transfer between SDRAM and FPGAs every clock cycle. A microprocessor can access the SDRAM directly (see Section 2.6).

2.3. Hardware-Accelerated Combined Multiple Recursive Generator (HACMRG). The SPRNG CMRG employs two generators. One is a 64-bit LCG and the other is a recursive generator [7]. The recursive generator is expressed by (8). The CMRG combines the two generators as follows:

$$Z(n) = \{X(n) + Y(n) \times 2^{32}\} \text{Mod } (2^{64}), \tag{7}$$

$$\begin{aligned} Y(n) = \{&107374182 \times Y(n-1) + 104480 \\ &\times Y(n-5)\} \text{ Mod } (2^{31} - 1), \end{aligned} \tag{8}$$

where $X(n)$ is generated by a 64-bit LCG, $Y(n)$ represents a 31-bit random number, and $Z(n)$ is the resulting random number [2, 7]. The implementation equation is represented by:

$$Z(n) = X'(n) + Y(n), \tag{9}$$

where $X'(n)$ is the upper 32 bits of $X(n)$. Equation (9) produces identical results as (7).

Figure 3 shows the HACMRG hardware architecture. The architecture has two parts, each generating a partial result. The first part is an HALCG64, and the second part is a generator having two lag factors as in (8).

The left part in Figure 3 is the HALCG64 employing a two-staged multiplier. The HACMRG HALCG64 produces one $X(n)$ random number every other clock cycle to synchronize with the $Y(n)$ generator composed of two two-staged multipliers, four-deep FIFO registers, and some combinational logic. In the $Y(n)$ generator, the multiplexor controlling the left multiplier inputs switches between an input value of "1" during the initial random number and the previous value $Y(n-1)$ thereafter.

For the "$2^{31}-1$" modulo operator implementation, the 62-bit value summed from the two multiplier's outputs is shifted right by 31-bits and added to the lower 31-bits of the value before the shifting. The shifted value represents the modulo value for the higher 31 bits of the 62 bit value and the lower 31 bits of the 62 bit value represents the modulo value itself. The summed value represents the total modulo value. The total modulo value is reexamined to represent the final modulo value. If the total modulo value is larger or equal than 2^{31}, the final modulo value is represented by adding "1" to the lower 31-bit data of the total modulo value since the $(2^{31}-1)$ modulo value of 2^{31} is "1". The value obtained from the modulus operation fans out three ways. The first one goes to the left multiplier input port in Figure 3 in order to save one clock cycle latency, the second one goes to the FIFO, and the third one goes to the final adder to add the value to the result generated by the HALCG64. The resulting upper 31 bits represent the next random number. The HACMRG generates a random number every other clock cycle.

2.4. Hardware-Accelerated Multiplicative Lagged Fibonnaci Generator (HAMLFG). The SPRNG MLFG characteristic equation is given by

$$Z(n) = \{Z(n-k) \times Z(n-l)\} \text{Mod } (2^{64}), \tag{10}$$

where k and l are time lags and $Z(n)$ is the resulting random number [2, 7].

The HAMLFGS and HAMLFGL employ two generators to produce two random numbers every clock cycle based on (10). The HAMLFGS and HAMLFGL require three-port RAMs to consistently keep previous random numbers since two ports are required to store two random numbers and one port is required to read a random number at the same time. However, only two ports are supported for the DPRAMs. Fortunately, (10) reveals data access patterns such that one port is enough for storing two random numbers. Tables 1 and 2 show the data accessing patterns in the case of the $\{17, 5\}$ and $\{31, 6\}$ parameter sets. In these tables, we reference the "odd-odd" case when the longer lag factor l is odd and the shorter lag factor k is odd and the "odd-even" case when the longer lag factor is odd and the shorter lag factor is even. Note that l is always odd for the SPRNG parameter sets [7].

Table 1: Data access patterns in odd-odd case $\{17, 5\}$.

ODD-ODD case $\{17, 5\}$ Two random numbers every clock cycle	Odd random number	Even random number
1st	$\mathbf{Z(17)} = Z(12) \times Z(0)$	$\mathbf{Z(18)} = Z(13) \times Z(1)$
2nd	$\mathbf{Z(19)} = Z(14) \times Z(2)$	$Z(20) = Z(15) \times Z(3)$
3rd	$Z(21) = Z(16) \times Z(4)$	$Z(22) = \mathbf{Z(17)} \times Z(5)$
4th	$Z(23) = \mathbf{Z(18)} \times Z(6)$	$Z(24) = \mathbf{Z(19)} \times Z(7)$
...	.....	.....

Table 2: Table 1 Data access patterns in odd-even case $\{31, 6\}$.

ODD-EVEN case $\{31, 6\}$ Two random numbers every clock cycle	Odd random number	Even random number
1st	$\mathbf{Z(31)} = Z(25) \times Z(0)$	$\mathbf{Z(32)} = Z(26) \times Z(1)$
2nd	$Z(33) = Z(27) \times Z(2)$	$Z(34) = Z(28) \times Z(3)$
3rd	$Z(35) = Z(29) \times Z(4)$	$Z(36) = Z(30) \times Z(5)$
4th	$Z(37) = \mathbf{Z(31)} \times Z(6)$	$Z(38) = \mathbf{Z(32)} \times Z(7)$
...	.....	.....

In the case of odd-odd parameter sets, the odd part always accesses the even part and the even part always accesses the odd part in Table 1. In the case of odd-even parameter sets, the odd part always accesses the odd part and the even part always accesses the even part in Table 2. The data access patterns are described in bold face in Table 2. Figure 4 shows the HAMLFGS and HAMLFGL architecture exploiting these data access patterns. HAMLFGS and HAMLFGL employ four DPRAMs to store random numbers and two multipliers to produce two random numbers every clock cycle, since the generator can access four values every clock cycle (see Tables 1 and 2). The two upper DPRAMs (DPRAM 1 and DPRAM 2 in Figure 4) are used to generate odd indexed random numbers and the two lower DPRAMs (DPRAM 3 and DPRAM 4) are used to generate even indexed random numbers.

HAMLFGS and HAMLFGL also employ one read-controller to read data from DPRAMs and one write-controller to write data to DPRAMs from the multipliers. The two multiplexers (MUX in Figure 4) exploit the data access patterns. They select the appropriate values for the odd-odd and odd-even parameter sets. Two-stage multipliers are employed for HAMLFGS for the $\{17, 5\}$ and $\{31, 6\}$

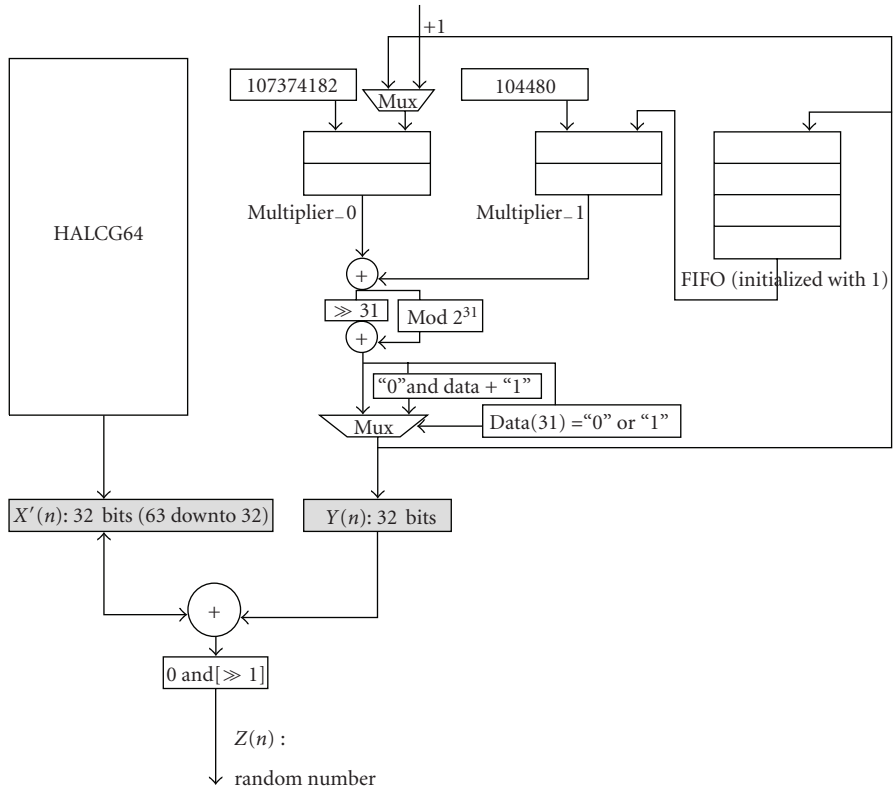


Figure 3: HACMRG architecture.

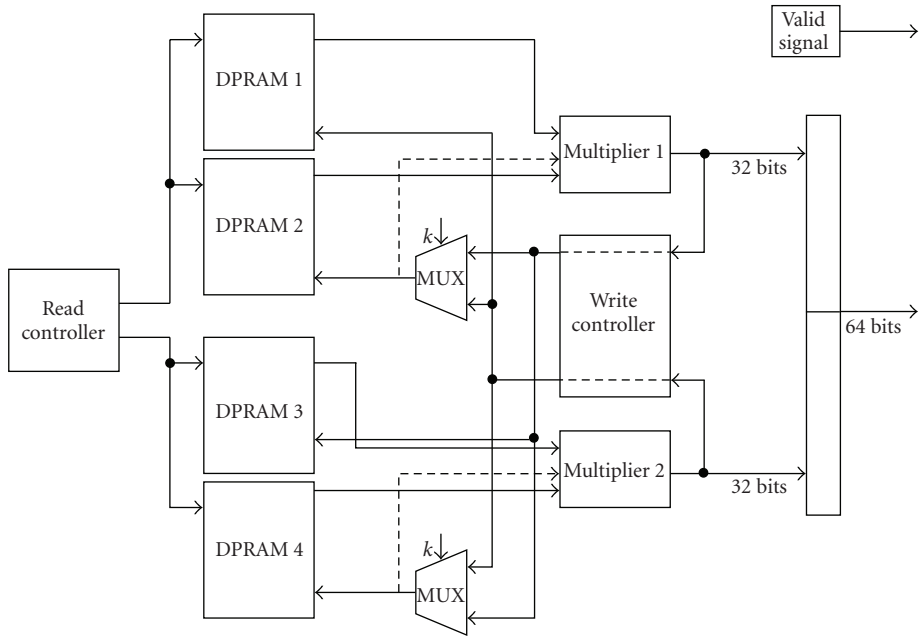


Figure 4: HAMLFGS and HAMLFGL architecture.

parameter sets and seven-stage pipelined multipliers are employed for HAMLFGL for the other nine parameter sets to avoid data hazards (SPRNG provides eleven parameter sets for MLFG) [7]. Even though two-staged multipliers are employed for the HAMLFGS implementation, data hazards still exist in Tables 1 and 2 since a two clock cycle latency is required due to the two-stage multipliers along with an additional two clock cycles for the DPRAM access, one for writing and one for reading. In order to avoid the two clock cycle delay from the DPRAM access, we employ forwarding

techniques when the parameter set is $\{17, 5\}$ or $\{31, 6\}$. Instead of storing the data into DPRAMs, the data is fed into the multipliers directly from the multiplier's outputs through the multiplexers. The forwarding is shown as dotted-lines in Figure 4. If the required latency is three clock cycles, such as for the odd part in $\{17, 5\}$ and both parts in $\{31, 6\}$, one stall is inserted to synchronize the data access. HAMLFGS and HAMLFGL produce two random numbers every clock cycle.

2.5. Hardware-Accelerated Prime Modulus Linear Congruential Generator (HAPLMCG). The SPRNG PMLCG generates random numbers as follows:

$$Z(n) = \{\alpha \times Z(n-1) \times 2^{32}\} \text{Mod } (2^{61} - 1), \qquad (11)$$

where α is a multiplier and $Z(n)$ is the resulting random number [2, 7]. Figure 5 shows the HAPMLCG architecture.

HAPMLCG employs four two-stage 32-bit multipliers that execute in parallel. The initial seed and the initial multiplier coefficient are divided by the lower 31 bits and upper 30 bits and are stored into the gray registers in Figure 5. In order to make those data 32 bits for 32-bit multiplications, the value of "0" is attached to the lower 31-bit data at the most significant bit position and the value of "00" is attached to the higher 30-bit data at the most significant bit position. All the shift and mask operations before the IF-condition black box are needed to make two 61-bit data multiplication (one is a multiplier coefficient and the other is a random number) as in (11). The last part of the implementation is described by the IF-condition black box in Figure 5. The IF-condition black box performs the modulus and data check operations. When the 62nd bit of the data is "1", the data is modified by adding "1" after the 61-bit mask operation. If the modified data is 2^{61}, the data is changed to the value "1" in order to prevent the generator from producing "0" as a random number [7, 19, 20]. The final data from the IF-condition black box fans out two directions. One is fed into one of the inputs of the multiplexer in order to generate the next random number. The other represents a resultant random number. After the first operation, the multiplexer always selects the resultant random numbers instead of the initial seed. The HAPMLCG generates a random number every other clock cycle.

The HAPMLCG did not use the HALCG technique for generating future random numbers, since it was extremely hard to derive an equation to generate future random numbers due to the complicated modulo value $(2^{61} - 1)$ while preventing the generation of a zero random number.

2.6. Programming Interface for HASPRNG. The programming interface for HASPRNG employs the C language and allows users to use HASPRNG the same way as SPRNG 2.0. The programming interface requires two buffers in main memory on the Cray XD1 to transfer data between the FPGAs and microprocessors. The main memory plays a role as a bridge to communicate between the microprocessor and the FPGA directly through HyperTransport (HT)/RapidArrayTransport (RT) [15].

HASPRNG provides the initialization function, the generation function, and the free function for the programming interface as in SPRNG 2.0 [7]. To the programmer, the functions for HASPRNG are identical to those for SPRNG, but with "sprng" replaced by "hasprng." The initialization function initializes HASPRNG. The generation function returns a random number per function call. The free function releases the memory resources when the random number generation is completely done. The three functions are implemented using Application Programming Interface (API) commands in C. Figure 6 describes the hardware architecture to support the programming interface of the Cray XD1.

The HASPRNG initialization function is responsible for reconfiguring the FPGA logic (FPGA in Figure 6) and registering buffers (Buffer 1 and Buffer 2 in Figure 6) in the Cray XD1 main memory. The initialization function requires five integer parameters to initialize HASPRNG. The five parameters represent the generator type, the stream identification number, the number of streams, an initial seed, and an initial parameter as with the initialization of SPRNG generators [7]. Refer to [7] for further explanation of the five initialization parameters. The logic for the specified generator type parameter is used to program the FPGAs. Once the generator core (Generator Core in Figure 6) is programmed in the FPGAs, the initialization function lets the generator core generate random numbers based on the five integer parameters and send them to a buffer until the two buffers are full of random numbers. A FIFO is employed to hide the latency for random numbers transfer and control the generator core operation. When the FIFO is full of data, the full flag signal is generated and it makes the generator core stop random number generation. When the full flag signal is released, the generator core starts generating random numbers again. The FIFO sends data directly to a buffer in the memory every clock cycle through HT/RT unless the buffer is full of data.

When the initialization function is complete, the random numbers are stored in two buffers so that a microprocessor can read a random number by a generation function call. The initialization function is called only once. Hence, the time initialization, including function call overhead, is negligible to overall performance.

The generation function allows a microprocessor to read random numbers from one buffer while the FPGA fills the other buffer. Once the buffer currently used by the processor is empty, the other buffer becomes active and the FPGA fills the empty buffer. In consequence the latency of random number generation can be hidden.

The free function stops random number generation and releases hardware resources. The free function makes the generator core inactive by sending a signal to the FPGAs and releases the two buffers and the pointer containing HASPRNG information such as the generator type and the buffer size.

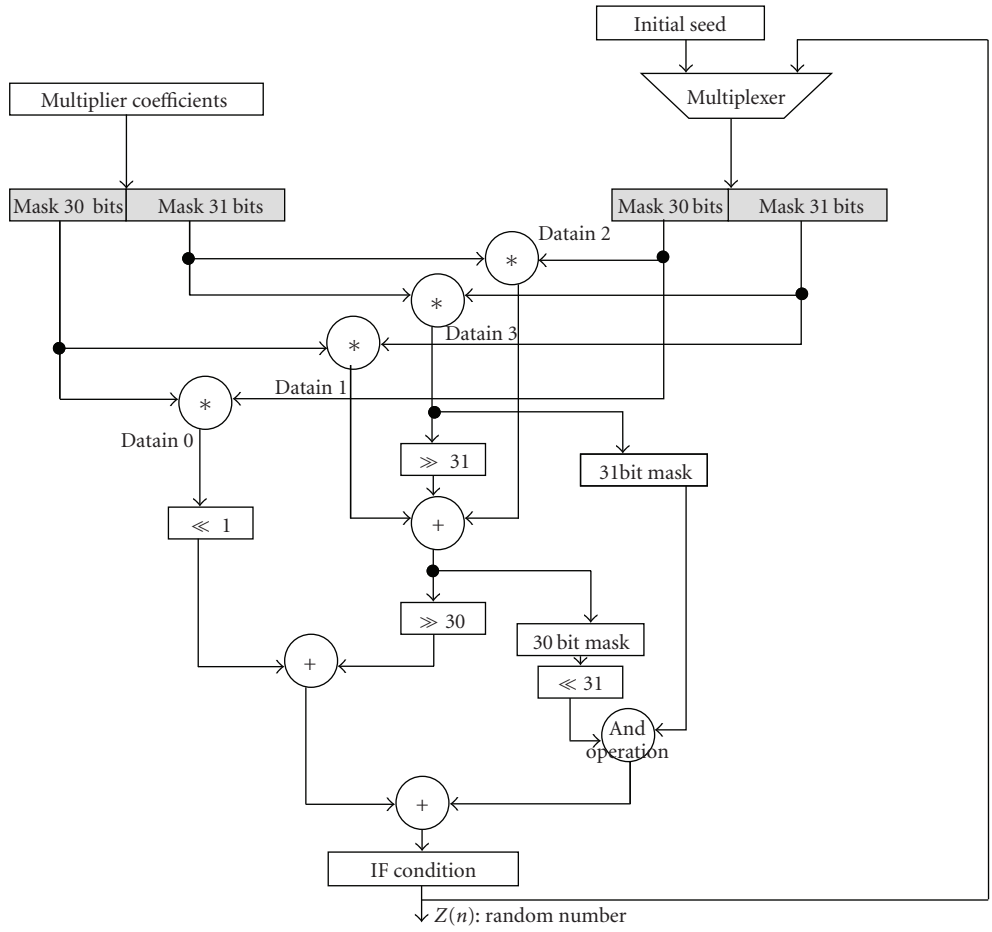


Figure 5: HAPMLCG architecture.

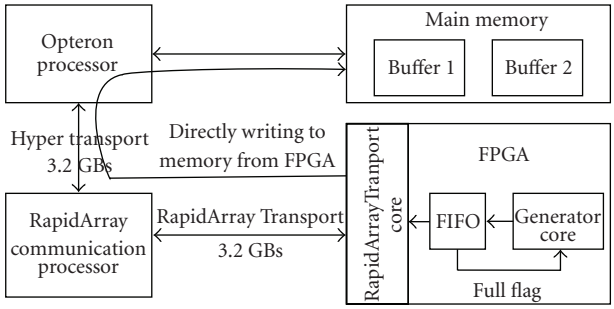


Figure 6: Programming interface architecture.

The performance of random number generation depends on the buffer size. We trade off the reduced overhead of swapping buffers by enlarging buffer size with fitting the buffer in cache. We optimized HASPRNG buffer size empirically, choosing 1MB as the most appropriate.

2.7. FPGA π-Estimator Using HASPRNG. We demonstrate an FPGA π-estimator implementation for the Cray XD1. The implementation of the π-estimation can be described by the following formula:

$$g(x,y) = \begin{cases} 1 & \text{when } x^2 + y^2 \le 1 \\ 0 & \text{when } x^2 + y^2 > 1, \end{cases} \tag{12}$$

$$\pi/4 = \int_0^1 \int_0^1 g(x,y)\,dx\,dy. \tag{13}$$

Based on the Law of Large Numbers (LLNs), the formula converges to the expected value of $g(x,y)$ as the number of samples increases [21]. Equation (14) describes the relation between the LLN and the MC integration for π estimation:

$$E(g(x,y)) = \left(\frac{1}{N}\right) \cdot \sum_{i=1}^{N} g(x_i, y_i) = \iint_0^1 g(x,y)\,dx\,dy = \frac{\pi}{4}, \tag{14}$$

where x_i and y_i represent samples and N is the number of sample trials of x_i and y_i. For example, if the π-estimation consumes two random numbers, one for x_i and one for y_i, then the value of N is "1".

The FPGA π-estimator implementation employs eight HALCG48 generators. Hence, the FPGA π-estimator is able to consume 16 random numbers per cycle (8 samples).

Table 3: HASPRNG XC2VP50 hardware usage.

	Hardware usage			Maximum number of copies
	Slices (23616)	Multipliers (232)	BRAM (232)	
HALFGOO	1022 (4%)	0 (0%)	40 (17%)	5
HALFGOE	1015 (4%)	0 (0%)	40 (17%)	5
HALCG48	1662 (7%)	12 (5%)	0 (0%)	14
HALCG64	2474 (10%)	20 (8%)	0 (0%)	9
HACMRG	683 (2%)	18 (7%)	0 (0%)	14
HAMLFGS	1123 (4%)	20 (8%)	32 (13%)	7
HAMLFGL	1670 (7%)	20 (8%)	40 (17%)	5
HAPMLCG	659 (2%)	16 (6%)	0 (0%)	16
Average LCG type	5%	7%	0 (0%)	13
Average LFG type	5%	8%	16%	5

Figure 7 describes the architecture of the FPGA π-estimator. Each "A" represents a HALCG48, each "B" represents a 32-bit multiplier, each "C" is an accumulator module, and the "D" is logic which produces the complete signal when the required number of iterations is done. A random number from the HASPRNG ranges from "0" to "$2^{31}-1$". Each pair of random numbers is interpreted as values in the unit square.

The multipliers compute the square of these numbers which are added as follows:

$$g(x, y) = x \times x + y \times y, \tag{15}$$

where x and y are random numbers generated by HALCG48s in the FPGA π-estimator. If the resultant $g(x, y)$ falls inside the unit circle, the accumulator adds "1" to the count value. When done, the FPGA π-estimator returns the count-value and the complete signal. When the complete signal is "1", the microprocessor reads the count values and computes the value of π based on (18):

$$\frac{\pi}{4} = \left(\frac{1}{N}\right) \times \sum_{i=1}^{N} g(x_i, y_i), \tag{16}$$

where N = (Number of random numbers)/2 = (16 × Number of Iterations (NI))/2 = 8 × NI,

$$\sum_{i=1}^{N} g(x_i, y_i) = \text{Count-Value (CV)}, \tag{17}$$

$$\pi = \frac{\text{CV}}{2 \times \text{NI}}. \tag{18}$$

3. Results

Each HASPRNG generator occupies a small portion of the FPGA, leaving plenty of room for RC MC candidate applications. Each of the HASPRNG generators is verified with over 1 million random numbers. Through the π-estimator results we demonstrate that HASPRNG is able to improve the performance significantly for RC MC applications. We use Xilinx 8.1 ISE Place And Route (PAR) tools to get hardware resource usage.

Table 4: HASPRNG performance.

	Clock rate	Actual (Theoretical) Speed in Millions of Random Numbers/second
HALFGOO	150 MHz	133 (150) MRN/s
HALFGOE	150 MHz	133 (150) MRN/s
HALCG	199 MHz	348 (398) MRN/s
HALCG64	199 MHz	349 (398) MRN/s
HACMRG	80 MHz	35 (40) MRN/s
HAMLFGS	80 MHz	142 (160) MRN/s
HAMLFGL	180 MHz	318 (360) MRN/s

Table 5: Generator Perfomance comparision.

	SPRNG	HASPRNG	Speedup	Speedup for max # of copies in FPGA
HALFGOO	77 MRNs	133 MRNs	1.7×	8.5×
HALFGOE	77 MRNs	133 MRNs		
HALCG48	167 MRNs	348 MRNs	2.1×	29.4×
HALCG64	200 MRNs	349 MRNs	1.7×	15.3×
HACMRG	91 MRNs	35 MRNs	0.4×	5.6×
HAMLFGS	111 MRNs	142 MRNs	2.6×	18.2×
HAMLFGL	111 MRNs	318 MRNs		13.0×
HAPMLCG	67 MRNs	35 MRNs	0.5×	8.0×
Average			1.5×	13.7×

3.1. HASPRNG Hardware Resources. Table 3 shows the hardware resource usage for the XC2VP50 FPGA in each Cray XD1 node and the maximum allowable number of generators on each FPGA

Each HASPRNG LFG consumes an average of about 5% of the slices, 8% of the built-in 18×18 multipliers, and 16% of the BRAMs. Multipliers are used when SPRNG characteristic equations contain multiplication operations and BRAMs are used when the initial seed data and random numbers are needed to be stored to generator random numbers. In consequence, the LCG type generators (HALCG48/64, HACMRG, and HAPMLCG) do not need BRAMs and

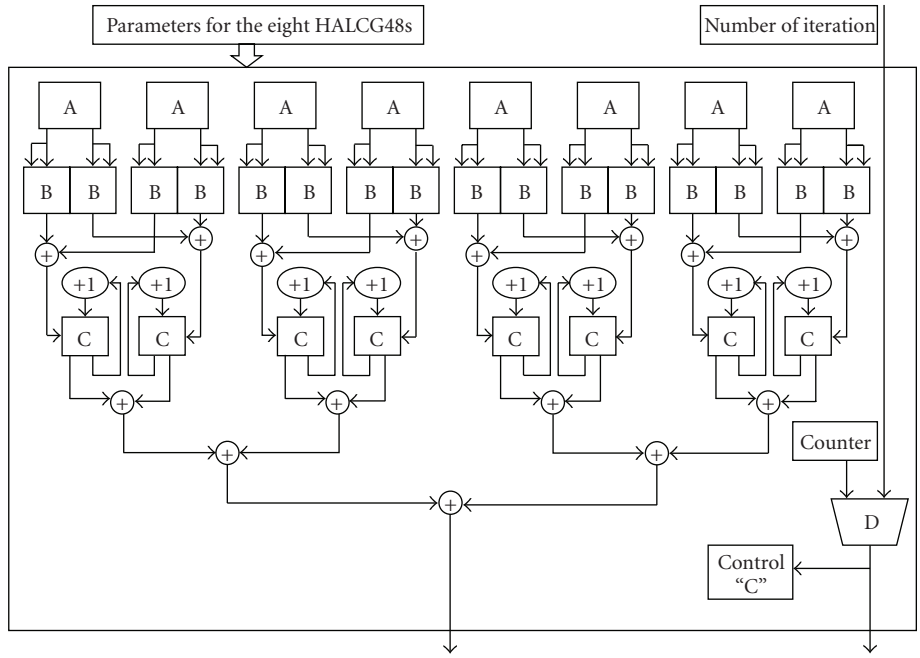


Figure 7: π-estimator for Cray XD1.

Table 6: π-estimator runtime comparison.

Number of Random Samples N	Software π-estimator (Seconds)	RC MC π-estimator (Seconds)	Speedup
8×10^4	5.6×10^{-4}	3.8×10^{-5}	14.7×
8×10^5	5.4×10^{-3}	3.4×10^{-4}	15.8×
8×10^6	5.4×10^{-2}	3.3×10^{-3}	16.3×
8×10^7	5.3×10^{-1}	3.3×10^{-2}	16.1×
8×10^8	5.5	3.3×10^{-1}	16.7×
8×10^9	5.1×10	3.3	15.5×
8×10^{10}	5.9×10^2	3.3×10	17.9×
8×10^{11}	6.3×10^3	3.3×10^2	19.1×

Table 7: Numerical results for the errors for π estimation.

Number of random samples N	Mean error	Standard deviation of error	95% Confidence interval upper limit of error
8×10^4	2.4×10^{-3}	2.1×10^{-3}	6.5×10^{-3}
8×10^5	3.1×10^{-3}	1.4×10^{-3}	5.5×10^{-3}
8×10^6	2.1×10^{-4}	1.2×10^{-4}	4.5×10^{-4}
8×10^7	1.2×10^{-4}	1.5×10^{-4}	4.1×10^{-4}
8×10^8	1.2×10^{-4}	1.1×10^{-4}	3.4×10^{-4}
8×10^9	3.0×10^{-5}	2.8×10^{-5}	8.5×10^{-5}
8×10^{10}	5.6×10^{-6}	3.2×10^{-6}	1.2×10^{-5}
8×10^{11}	1.2×10^{-6}	8.1×10^{-7}	2.8×10^{-6}

the HALFGOO/OE do not need multipliers. HASPRNG performance can be linearly improved by adding extra random number generator copies.

3.2. HASPRNG Performance. Each generator has a different clock rate. Table 4 shows applied clock rates and performance. The difference between actual performance and theoretical performance mainly comes from the data transfer overhead caused by transferring data from FPGAs to the main memory because the bus is saturated.

Table 5 shows performance evaluation for a single generator from HASPRNG compared to SPRNG. The gcc optimization level is set to O2. Because there are eleven parameter sets for the MLFG in SPRNG, we compute the HASPRNG and SPRNG performance in Table 5 using the average speed of each parameter set.

HASPRNG shows 1.5× overall performance improvement for a single HASPRNG hardware generator over SPRNG running on a 2.2 GHz AMD Opteron processor on the Cray XD1. Note that this speedup is limited by the bandwidth between the FPGA and microprocessor. Because SPRNG is designed to support additional random streams, we can easily add HASPRNG generators in the FPGA (up to the maximum copies in the last column of Table 3). This will not help applications on the Opteron processor because the link between the FPGA and Opteron's main memory is already saturated.

Other reconfigurable computing systems with faster links will obtain higher performance (with a speedup of up to 13.7× as shown in Table 5), or applications can be partially or entirely mapped to the FPGA for additional performance as seen in Section 2.7 with the π-estimator. Moreover, the Opteron microprocessor is now free to perform other portions of the computational science application.

Table 8: FPGA π-estimator XC2VP50 hardware usage.

	Hardware Usage		
	Slices (23616)	Multipliers (232)	BRAM (232)
FPGA π-estimator	18161 (77%)	160 (69%)	0 (0%)
Eight HALCG48s	13296 (56%)	96 (41%)	0 (0%)
FPGA π-estimator—Eight HALCG48s	4865 (21%)	64 (28%)	0 (0%)
FPGA π-estimator hardware resources per RNG	608 (3%)	8 (3%)	0 (0%)

3.3. HASPRNG Verification. For the verification platforms, each generator in HASPRNG was compared to its SPRNG counterpart to ensure bit-equivalent behavior [16, 18]. SPRNG was installed on the Cray XD1 to compare the results from SPRNG with the ones from HASPRNG. We observe that HASPRNG produces identical results with SPRNG for each type of generator and for each parameter set given in SPRNG. We verified over 1 million random numbers on the verification platform for each of these configurations.

3.4. Reconfigurable Computing π-Estimator. HASPRNG can improve the performance of RC MC applications as shown with the π-estimator. The π-estimator runs at 150 MHz and can consume 2.4 billion random numbers per second. The HASRPNG π-estimator shows 19.1× speedup over the software π-estimator employing SPRNG when the random samples are sufficiently large for Monte Carlo applications as shown in Table 6. We would expect the significant speedup for such computational science applications. It is worth noting that these results are for eight pairs of points generated per clock cycle. It takes less than 7 minutes to estimate π generating 1 trillion random numbers for the π-estimator on a single FPGA node.

Table 7 shows the numerical results for the absolute errors between the true value of π and the estimated π based on five different experiments for each sample size. Mean values and the standard deviations are computed assuming that the five sample errors follow a normal distribution [5, 21]. We seek 95% confidence intervals according to different random sample sizes. The error in the estimate of π decreases as N increases, proportional to $1/(N^{1/2})$ [3]. When similar MC integration applications consume an additional 100 times more random samples, they are able to obtain a solution having one more decimal digit of accuracy. Table 8 shows the hardware resource usage for the FPGA π-estimator.

4. Conclusions

Random number generation for Monte Carlo methods requires high performance, scalability, and good statistical properties. HASPRNG satisfies these requirements by providing a high-performance implementation of random number generators using FPGAs that produce bit-equivalent results to those provided by SPRNG. The bandwidth between the processor and FPGA is saturated with random number values, which limits the speedup on the Cray XD1. The reconfigurable computing Monte Carlo π estimation application using HASPRNG shows good performance and numerical results with a speedup of 19.1× over the software π-estimator employing SPRNG. Hence, HASPRNG promises to help computational scientists to accelerate their applications.

Acknowledgments

This work was partially supported by the National Science Foundation, Grant NSF CHE-0625598. The authors would like to thank Oak Ridge National Laboratory for access to the Cray XD1. They thank the reviewers for their helpful comments.

References

[1] T. Warnock, "Random-number generators," *Los Alamos Science*, vol. 15, pp. 137–141, 1987.

[2] M. Mascagni and A. Srinivasan, "Algorithm 806: SPRNG: a scalable library for pseudorandom number generation," *ACM Transactions on Mathematical Software*, vol. 26, no. 3, pp. 436–461, 2000.

[3] S. Weinzierl, "Introduction to Monte Carlo methods," Tech. Rep. NIKHEF-00-012, NIKHEF Theory Group, Amsterdam, The Netherlands, June 2000.

[4] N. Metropolis and S. Ulam, "The Monte Carlo method," *Journal of the American Statistica Association*, vol. 44, no. 247, pp. 335–341, 1949.

[5] R. L. Scheaffer, *Introduction to Probability and Its Applications*, Duxbury Press, Belmont, Calif, USA, 1995.

[6] M. Mascagni, "Parallel pseudorandom number generation," *SIAM News*, vol. 32, no. 5, pp. 221–251, 1999.

[7] Scalable parallel pseudo random number generators library, June 2007, http://sprng.fsu.edu.

[8] P. L'Ecuyer and R. Simard, "TestU01: a C library for empirical testing of random number generators," *ACM Transactions on Mathematical Software*, vol. 33, no. 4, article 22, 2007.

[9] G. Marsaglia, "DIEHARD: a battery of tests of randomness," 1996, http://stat.fsu.edu/pub/diehard/l.

[10] J. M. McCollum, J. M. Lancaster, D. W. Bouldin, and G. D. Peterson, "Hardware acceleration of pseudo-random number generation for simulation applications," in *Proceedings of the 35th IEEE Southeastern Symposium on System Theory (SSST '03)*, 2003.

[11] K. Underwood, "FPGAs vs. CPUs: trends in peak floating-point performance," in *Proceedings of the ACM International Symposium on Field Programmable Gate Arrays (FPGA '04)*, vol. 12, pp. 171–180, Monterrey, Calif, USA, February 2004.

[12] P. P. Chu and R. E. Jones, "Design techniques of FPGA based random number generator (extended abstract)," in *Proceedings of the Military and Aerospace Applications of Programmable Devices and Technologies Conference (MAPLD '99)*, 1999.

[13] D. Buell, T. El-Ghazawi, K. Gaj, and V. Kindratenko, "High-performance reconfigurable computing," *Computer*, vol. 40, no. 3, pp. 23–27, 2007.

[14] M. C. Smith and G. D. Peterson, "Parallel application performance on shared high performance reconfigurable computing resources," *Performance Evaluation*, vol. 60, no. 1–4, pp. 107–125, 2005.
[15] Cray Inc., "Cray XD1 FPGA development," 2005.
[16] J. Lee, G. D. Peterson, and R. J. Harrison, "Hardware accelerated scalable parallel random number generators," in *Proceedings of the 3rd Annual Reconfigurable Systems Summer Institute*, July 2007.
[17] J. Lee, Y. Bi, G. D. Peterson, R. J. Hinde, and R. J. Harrison, "HASPRNG: hardware accelerated scalable parallel random number generators," *Computer Physics Communications*, vol. 180, no. 12, pp. 2574–2581, 2009.
[18] J. Lee, G. D. Peterson, R. J. Harrison, and R. J. Hinde, "Hardware accelerated scalable parallel random number generators for Monte Carlo methods," in *Proceedings of the Midwest Symposium on Circuits and Systems*, pp. 177–180, August 2008.
[19] Y. Bi, *A reconfigurable supercomputing library for accelerated parallel lagged-Fibonacci pseudorandom number generation*, M.S. thesis, Computer Engineering, University of Tennessee, December 2006.
[20] Y. Bi, G. D. Peterson, G. L. Warren, and R. J. Harrison, "Hardware acceleration of parallel lagged Fibonacci pseudo random number generation," in *Proceedings of the International Conference on Engineering of Reconfigurable Systems and Algorithms*, June 2006.
[21] J. Jacob and P. Protter, *Probability Essentials*, Springer, Berlin, Germany, 2004.

Hindawi Publishing Corporation
VLSI Design
Volume 2010, Article ID 946486, 8 pages
doi:10.1155/2010/946486

Research Article

A Pipelined and Parallel Architecture for Quantum Monte Carlo Simulations on FPGAs

Akila Gothandaraman,[1,2] Gregory D. Peterson,[1] G. Lee Warren,[3] Robert J. Hinde,[4] and Robert J. Harrison[4]

[1] *Department of Electrical Engineering and Computer Science, The University of Tennessee, Knoxville, TN 37996, USA*
[2] *Center for Simulation and Modeling, Department of Chemistry, The University of Pittsburgh, Pittsburgh, PA 15260, USA*
[3] *Department of Chemistry, Yeshiva University, New York, NY 10033, USA*
[4] *Department of Chemistry, The University of Tennessee, Knoxville, TN 37996, USA*

Correspondence should be addressed to Akila Gothandaraman, agothan1@utk.edu

Received 5 June 2009; Accepted 19 November 2009

Academic Editor: Ethan Farquhar

Recent advances in Field-Programmable Gate Array (FPGA) technology make reconfigurable computing using FPGAs an attractive platform for accelerating scientific applications. We develop a deeply pipelined and parallel architecture for Quantum Monte Carlo simulations using FPGAs. Quantum Monte Carlo simulations enable us to obtain the structural and energetic properties of atomic clusters. We experiment with different pipeline structures for each component of the design and develop a deeply pipelined architecture that provides the best performance in terms of achievable clock rate, while at the same time has a modest use of the FPGA resources. We discuss the details of the pipelined and generic architecture that is used to obtain the potential energy and wave function of a cluster of atoms.

1. Introduction

Reconfigurable Computing (RC) using Field-Programmable Gate Arrays (FPGAs) is emerging as a computing paradigm to accelerate the computationally intensive functions of an application [1]. RC provides users the flexibility of a software solution with hardware-like performance. We develop a parallel and pipelined reconfigurable architecture for accelerating the computationally intensive functions of a Quantum Monte Carlo (QMC) application. QMC is a simulation technique that provides a practical and efficient way of solving the many-body Schrödinger equation [2]. Using this method, we can obtain the ground-state properties, such as energy and wave function, of a system of interacting particles. A software QMC application can be used to simulate a system of particles on general-purpose processors. To speed up the application and to provide better computational scaling with the system size, we use a hardware-software approach to speed up the computationally intensive functions of the application. In our implementation, we use hardware-based techniques to exploit parallelism using pipelining. Due to the complex nature of the functions involved, we use an interpolation-based approximation method for calculating these functions. The exact shape of the function depends on the nature of the atoms involved. Hence, the use of FPGAs gives us the flexibility to develop a programmable architecture so we can vary the parameters depending on the system. We propose a reconfigurable hardware architecture using Field-Programmable Gate Arrays (FPGAs) to implement the kernels of our application. In [3, 4], we presented our ongoing research and our hardware accelerated framework where we achieve promising results when the kernels of the application are ported to the Cray XD1 reconfigurable supercomputing platform [5]. Here, we focus on the architectural details, our design goals, and choices while designing the pipelined architecture to calculate the potential energy and wave function of atomic clusters.

Our design goals are performance, numerical accuracy, and flexibility. To quantify performance, we measure the speed up achieved by our hardware implementation over the

optimized software application. *Accuracy* is guaranteed by employing the numerical precision that delivers results that are comparable to the software version employing double-precision floating-point representation. We satisfy the above design goals by developing a deeply pipelined reconfigurable architecture that employs a fixed-point representation chosen after careful error analysis. *Flexibility* is provided by creating a general user-friendly framework [6] that can be used to simulate a variety of atomic clusters, which have the same overall function, with slightly different simulation parameters. The current framework can be extended to calculate other properties during the simulation. Our present implementation meets the above design goals by providing a speed up of 40*x* for the potential energy and wave function kernels targeted to the Xilinx Virtex-4 FPGA [7] on the Cray XD1 platform and accuracy comparable to the double-precision software implementation on the processor. A general interpolation framework allows us to calculate both the pairwise potential energies and wave function of atomic clusters with the flexibility to both vary the identity of the atoms as well as to realize different forms of potential energy or wave function.

The paper is organized as follows. In Section 2, we provide an overview of the QMC algorithm. Section 3 describes the different components of the pipelined reconfigurable hardware. In Section 4, we present the results of our implementation on the Cray XD1 platform. We provide conclusions and directions for future research in Section 5.

2. Algorithm

Quantum Monte Carlo (QMC) methods are widely used in physics and physical chemistry to obtain the ground-state properties of atomic or molecular clusters [2]. These methods are used to solve the many-body Schrödinger equation. Equation (1) represents the Schrödinger equation, the fundamental equation of quantum mechanics. In this equation, H refers to the Hamiltonian operator; E represents the energy of the system. In (2), we rewrite the Hamiltonian as the sum of kinetic and potential energy terms. This equation then gives the one-dimensional time-independent Schrödinger equation for a chargeless particle of mass, m, moving in a potential, $V(x)$. The analogous three-dimensional time-independent equation is given by (3). Solving this equation is trivial for small systems, but as the dimensions of the system increase, it is impractical to solve the equation analytically. In (2) and (3), $\hbar$ is the Planck's constant, ψ is the wave function, and ∇^2 is the Laplace operator,

$$H\psi = E\psi, \tag{1}$$

$$-\frac{\hbar^2}{2m}\frac{d^2\psi(x)}{dx^2} + V(x)\psi(x) = E\psi(x), \tag{2}$$

$$\left[-\frac{\hbar^2}{2m}\nabla^2 + V(r)\right]\psi(r) = E\psi(r). \tag{3}$$

We use a flavor of QMC called the Variational Monte Carlo (VMC) algorithm [2, 3]. In Step 1 of the algorithm, we initialize a reference configuration of atoms. In Step 2, we add a small random displacement to all the atoms in the configuration obtained in Step 1 to obtain a new configuration. Step 3 of the algorithm consists of the calculation of properties namely potential energy and wave function. These are pairwise functions and hence $O(N^2)$ in the number of atoms, N. In Step 4, we accept or reject the configuration using the ratio of the wave function values of the new and old configurations. Depending on the ratio, we use a uniformly distributed random number to decide whether or not to accept a configuration and its associated properties. These steps are repeated for all configurations and thousands of iterations, to accurately estimate the properties.

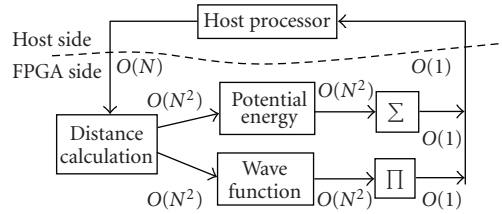


Figure 1: Data movement in the QMC algorithm.

Figure 1 shows the data movement in the QMC algorithm, which will help us decide how to partition the steps of the algorithm between the software application and reconfigurable hardware. The potential energy and wave function are both functions of the coordinate distances between the atoms. The coordinate distance calculation, calculation of potential energy and wave function are all $O(N^2)$ in the number of atoms. The total potential energy, V_{total}, is the sum of the $N(N-1)/2$ pairwise contributions as shown in (4). The total wave function, ψ_{total}, is approximated as the product of the pairwise contributions as shown in (5). In (4) and (5), r_{ij} denotes the radial distance between atoms, i and j. Upon the completion of Step 3 of the VMC algorithm, we have a single energy or wave function result that is transferred back to the host processor,

$$V_{\text{total}} \approx \sum_{i<j} V_2\left(r_{ij}\right), \tag{4}$$

$$\psi_{\text{total}} \approx \prod_{i<j} \psi_2\left(r_{ij}\right). \tag{5}$$

Table 1 shows the execution time for the different components, total time and compute time (for 1 iteration and N = 4096), and the percentage of total time for computing the kernels (Step 3 of the QMC algorithm). As we can see, the compute time (time for potential energy, and wave function calculations) dominates the total time for an iteration of the algorithm. Hence, we offload Step 3 of the algorithm to the FPGA. We have $O(N)$ coordinate positions to be moved from the processor to the FPGA for every iteration. A high-speed interconnect between the FPGA and processor helps keep the cost of the data movement low for each iteration, so we can speed up the computation of properties in Step 3.

Table 1: Total execution time versus compute time for 1 iteration.

Components	Time (in s) or %
Potential energy calculation	0.7615
Wave function calculation	2.739
Compute time for 1 iteration	3.500
Total time for 1 iteration	3.715
% of total time spent on compute	94%

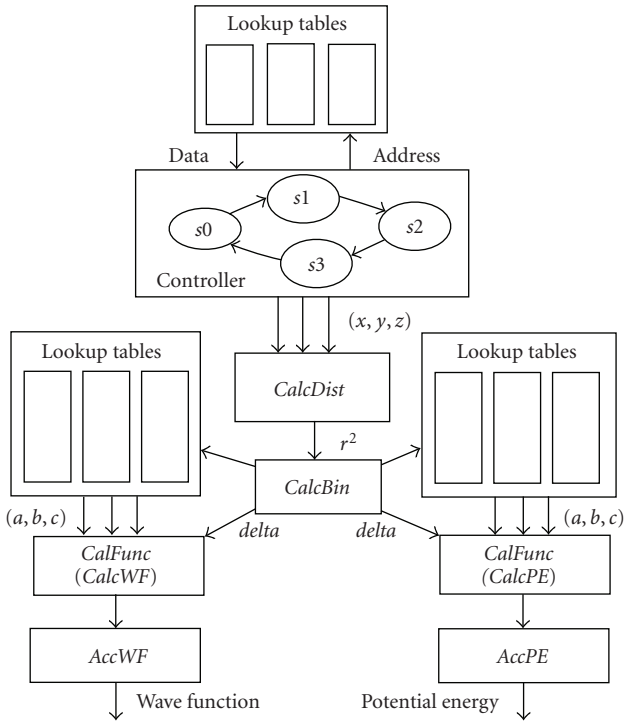


Figure 2: Top-level Block Diagram.

3. Architecture

Figure 2 shows the top-level block diagram of the pipelined architecture implemented on the FPGA. The architecture consists of the following components: *CalcDist* calculates the squared distances between the pairs of atoms. *CalcFunc* module uses as inputs the squared distances from *CalcDist* and evaluates the functions using an interpolation scheme. This is a generic module used in our architecture to compute the potential energy and wave function. These blocks are denoted as *CalcPE* and *CalcWF* in Figure 2. *AccFunc*, a generic module, accumulates the energies and wave functions to result in partial results that are delivered to the host processor. The accumulators for potential energy and wave function are labeled in Figure 2 as *AccPE* and *AccWF*. Different accumulator configurations are used depending on whether we compute the sum or product of the computed functions.

In addition to the above components, lookup tables using on-chip Block RAMs (BRAMs) are used to store the coordinate positions of atoms and the interpolation coefficients for the function evaluations. A state machine

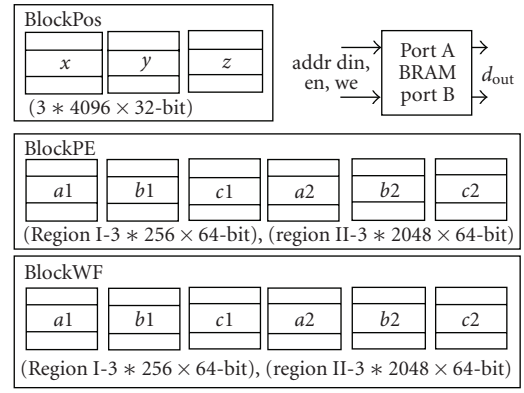


Figure 3: Memory Layout.

Table 2: BlockRAM Usage by the Lookup Tables.

	BlockPos	*BlockPE*	*BlockWF*
x/($a1$ and $a2$)	16 KB (8)	18 KB (10)	18 KB (10)
y/($b1$ and $b2$)	16 KB (8)	18 KB (10)	18 KB (10)
z/($c1$ and $c2$)	16 KB (8)	18 KB (10)	18 KB (10)
Total	48 KB (24)	54 KB (30)	54 KB (30)

controller is used to generate the addresses to the BRAMs that store the coordinate positions and transfer the (x, y, z) positions to the *CalcDist* module. The *CalcBin* module is used to compute the bin address, which is used to fetch the (a, b, c) interpolation coefficients. This module also produces an output, *delta,* that is used along with the interpolation coefficients by the *CalcFunc* modules. The distances from the *CalcDist* module are used by both *CalcPE* and *CalcWF* modules, which operate in parallel. Also, the components are deeply pipelined and designed to produce a result every clock cycle.

3.1. Lookup Tables. The 18-kbit embedded BRAMs on the Xilinx FPGA are used as lookup tables to store the (x, y, z) coordinate positions and (a, b, c) interpolation coefficients for the function evaluation. These memories are instantiated using the Xilinx Core Generator tools [8]. Figure 3 shows the layout of the BRAMs used to realize the above lookup tables. The BRAM used to store positions, called the *BlockPos* can support clusters of up to 4096 atoms. (Note that this is a limitation due to the size of the target FPGA. On the currently targeted FPGA, we use any remaining BRAMs to calculate additional properties of interest. On newer FPGAs, we can configure the BRAMs to support larger clusters due to larger memories). We use a single dual-port BRAM to store the 4096 32-bit positions. Three BRAMs are used to store the (x, y, z) positions for a total of 48 KB as shown in Table 2. This requires 24 BRAMs on the FPGA.

From our initial experiments varying the order of interpolation, we infer that quadratic interpolation with coefficients (a, b, c) guarantees the required accuracy with modest use of BRAM resources. The potential energy and wave function are divided into two regions, region I and

region II with each region approximated using a unique set of interpolation coefficients [3]. The lookup tables for each function store 256 ($a1$, $b1$, $c1$) 52-bit coefficients for region I and 1344 ($a2$, $b2$, $c2$) 52-bit coefficients for region II. The BRAMs to store the region I and II coefficients are configured with a depth of 256 and 2048 and width of 64-bit, respectively. These lookup tables are referred to as *BlockPE* and *BlockWF*.

For region I coefficients, we require 6 KB of memory, and for region II coefficients, we require 48 KB of memory for a total of 54 KB. For potential energy and wave function calculations, the coefficient memories consume a total of 60 BRAMs on the FPGA. The amount of memory needed to store the interpolation coefficients is given in Table 2. The number of Block RAMs needed for the above lookup tables is given in parenthesis in Table 2.

We use dual-port BRAMs, such that Port *A* writes the coordinate positions and Port *B* is used to read the positions. Ports *A* and *B* have a read pipeline latency of one clock cycle. In an RC setup, the host processor would load the positions to the *BlockPos* memory and the interpolation coefficients to the *BlockPE* and *BlockWF* BRAMs, respectively. The design implemented on the FPGA would read the *BlockPos* memory to calculate the distances. These distances would be used along with the interpolation coefficients from *BlockPE* or *BlockWF* to obtain the potential energy or wave function. Since the analytical functions for potential energy and wave function remain fixed throughout the simulation, the interpolation coefficients need to be loaded to the *BlockPE* or *BlockWF* memories by the processor only once prior to the beginning of the pipeline operation. However, the change in coordinate positions of the atoms, which relates to the physical movement of the atoms during the simulation, requires us to load the new positions to the *BlockPos* memory. As described earlier, the atoms are perturbed during every iteration of the algorithm and hence new positions are copied by the processor to the BRAMs for every iteration. The state machine transitions from one state to another to generate the addresses to the *BlockPos* memory so that the positions are read from the memory and transferred to the distance calculation and function evaluation modules. We calculate the energies and wave functions due to the atomic interactions in the shaded portion of the matrix (upper triangular part of the matrix) shown in Figure 4.

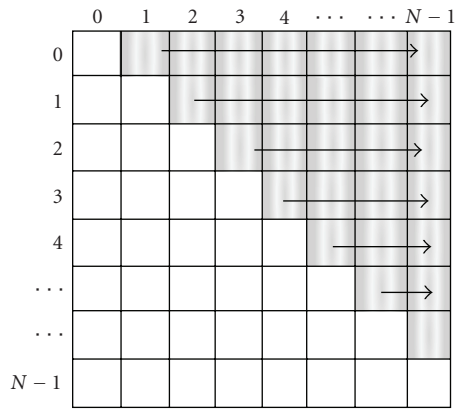


Figure 4: State machine generates addresses to read *BlockPos* BRAM in the order shown.

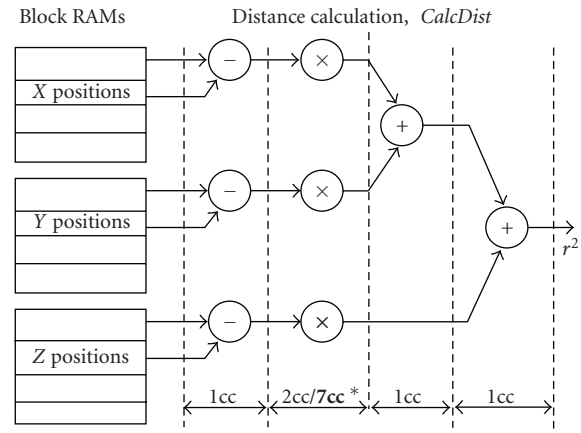


Figure 5: Squared distance calculation, *CalcDist*.

3.2. Distance Calculation (CalcDist). The ground-state properties such as potential energy and wave function are functions of the coordinate distance, r_{ij}, between a pair of atoms i and j. The distance calculation module, *CalcDist,* computes the squared distance between pairs of atoms. We eliminate the expensive square root on the FPGA by rewriting these properties as functions of the squared distance. Figure 5 shows the block diagram of this component. The latencies of the submodules are shown in clock cycles. The state machine controller provides the read addresses to the position memory, *BlockPos,* every clock cycle. The data outputs from the Position BRAMs are connected to the inputs of the *CalcDist* module. Since $r_{ij} = r_{ji}$ and $r_{ij} = 0$ when $i = j$, we only need to calculate $N(N-1)/2$ squared distances (upper or lower triangular portion of the matrix in Figure 4).

We use the Xilinx IP cores from the CoreGen library [8] to implement functions such as addition, subtraction, and multiplication. The cores are provided with a variety of pipelining options, variable input-output data-widths and customized for the target Xilinx FPGA architecture. For example, the multipliers are available with minimum or maximum pipelining with two- or seven-clock cycle latency respectively. We use the multipliers that are available as hard macros on the Xilinx Virtex4 series, in the form of DSP48s. To increase clock rates, the multipliers are configured for maximum pipelining, which corresponds to a latency of seven clock cycles. Table 3 shows the data widths and latencies of this module. The data widths for inputs and outputs are chosen after analyzing the errors with various fixed-point representations. The initial latency of this module is ten clock cycles after which the pipeline is full and it produces a 53-bit squared distance every clock cycle. Since we obtain a result every clock cycle, it takes $10 + N(N-1)/2$ clock cycles to obtain all the coordinate distances. However, since the entire architecture is pipelined,

Table 3: Data Widths and latencies of *CalcDist*.

Signal/Core	*Data Widths*	*Latency*
Input	32-bit (signed 12.20)	—
Subtractor	32-bit i/p, 33-bit o/p	1
Multiplier	33-bit i/p, 66-bit o/p	7
Adder1	65-bit i/p, 66-bit o/p	1
Adder2	66-bit i/p, 67-bit o/p	1
Output	53-bit (unsigned 27.26)	—

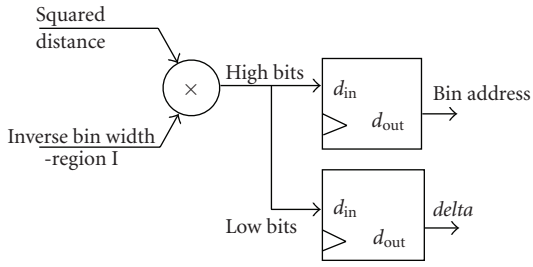


Figure 6: Region I Bin Calculation.

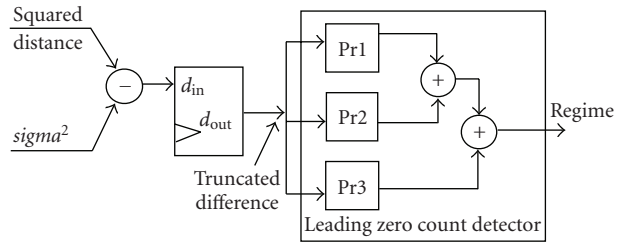


Figure 7: Region II Bin Calculation (First stage to obtain the regime).

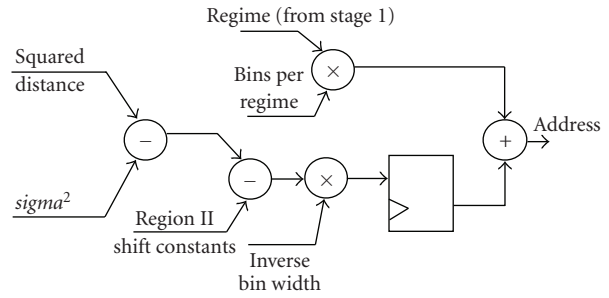


Figure 8: Region II Bin Calculation (Second stage to obtain the bin address).

the calculation of the pairwise distances is overlapped with the function evaluation.

3.3. Bin Calculation (CalcBin). The schemes to look up the interpolation coefficients are different for the two regions in each function, as the functions exhibit different numerical behavior in each region. The squared distance from the *CalcDist* module is compared with a cutoff value, $sigma^2$, to classify the distance as a region I or region II distance. Region I of the function is approximated using 256 bins. Using the inverse of the bin width, we can choose from the 256 values corresponding to the squared distances. The bin lookup scheme for region I is shown in Figure 6. The lower 8-bits of the product of the squared distance and the inverse bin width (by extracting the value to the left of the decimal point) are used to fetch the interpolation coefficients from the memory.

We employ a logarithmic binning scheme in region II. We divide the region II into 21 regimes. Each regime is divided into 64 bins for a total of 1344 coefficients. For regions I and II, we have a total of $[256 + 21 * 64] * 3$ coefficients for quadratic interpolation. First, we determine the regime and then determine the bin within the regime. We store the inverse of the bin widths corresponding to each bin in the BRAMs, eliminating the need for a division operation. The block diagram of the first stage of the lookup scheme for region II is shown in Figure 7. The difference between the squared distances and the sigma value is used by the leading zero count detector to compute the index of the regime. The detector logic is implemented using a set of three priority encoders (Pr1, Pr2, Pr3). Now that we have the index of the regime (i.e., the right power of 2), we use this to fetch the bin widths associated with this regime. Figure 8 shows the second stage of the lookup scheme to obtain the bin address to fetch the interpolation coefficients. To compute the bin location for region II, we require additional constants (region II shift constants and inverse bin width constants), which are obtained from BRAMs addressed using the regime. We can use the computed address to retrieve the interpolation coefficients for region II.

3.4. Calculation of Functions (CalcFunc). Figure 9 shows the data path of the *CalcFunc* module. This is a generic module that evaluates a function using polynomial interpolation. The interpolation coefficients, squared distances from the distance calculation module, and a delta value related to the squared distances are used to compute the function. In our reconfigurable architecture, we instantiate two copies of the *CalcFunc* module, one to calculate the potential energy, namely, the *CalcPE* and the other to compute the wave function, called the *CalcWF* as shown earlier in Figure 2. We load the corresponding (region I or region II) interpolation coefficients from the respective Block RAMs for each function. The address computed by the bin calculator, *CalcBin*, is used to fetch the (a, b, c) interpolation coefficients for the functions. The Xilinx IP cores (from Coregen) are used to implement the adders and multipliers. The multipliers can be configured to have a latency of two or seven clock cycles. We configure the multipliers for maximum pipelining (latency of seven clock cycles). This module outputs a 52-bit potential energy or wave function every clock cycle. Figure 9 shows the latencies (in clock cycles) in each step. Table 4 shows the data widths and latencies of the components of this module.

3.5. Accumulation of Functions (AccFunc). *AccFunc* is a generic module that is used to accumulate the results from the function evaluation module, *CalcFunc*. We instantiate two copies of the module, one for potential energy called

Table 4: Data widths and latencies of *CalcFunc* module.

Signal/Core	*Data Widths*	*Latency* (clock cycles)
Inputs	Interpolation coefficients (a, b, c): signed 0.52,delta: signed 0.52	—
Multiplier1	52-bit i/p, 104 bit o/p	7
Adder1	52-bit i/p, 52-bit o/p	1
Multiplier2	52-bit i/p, 104-bit o/p	7
Adder2	52-bit i/p, 52-bit o/p	1
Output	Pairwise Potential Energy/Wave Function: signed 0.52	—

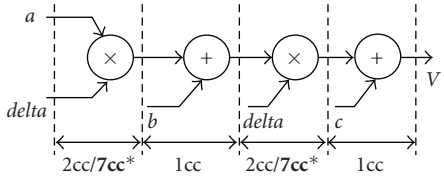


Figure 9: *CalcFunc* Module.

AccPE and the other for wave function denoted as *AccWF*. Figure 10 shows the *AccPE* module that accumulates the intermediate values of potential energy produced by *CalcPE*. The potential energy function is transformed prior to interpolation [3]; hence we use two types of accumulators: we multiply the intermediate results of potential energies for region I and we add the intermediate results of the energies for region II. The potential values which result from region I squared distances and interpolation constants are called region I potential values. Similarly, region II potential values result from region II squared distances and interpolation constants. The region II accumulator, which is instantiated from the Xilinx CoreGen library, accumulates the region II potential, if produced during that clock cycle or a zero if a region I potential is produced that clock cycle. For accumulating region I potential values, we have to form products of the potential energy result every clock cycle. The region I accumulator accumulates the region I potential, if produced during that clock cycle or a one if a region II potential is produced that clock cycle. The Xilinx CoreGen multipliers (with maximum pipelining) have a latency of seven clock cycles. We use eight instances of the multipliers (as the multiplier outputs are registered) and switch between the multipliers to increase the data rate and produce a partial product every clock cycle. At the end of our calculations for N atoms, we have a single partial sum (*PS*) from region II and eight partial products (*PP1-PP8*) from region I. The region I accumulator has an initial latency of eight clock cycles, after which one partial result is produced every clock cycle. The region II accumulator has a latency of two clock cycles. The worst-case latency that this module adds to the pipeline is eight clock cycles.

Figure 11 shows the *AccWF* module that accumulates the intermediate results of the wave function produced

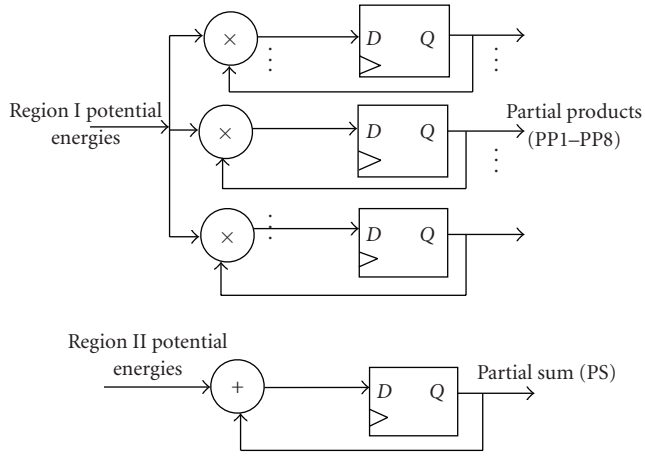


Figure 10: Potential energy accumulation (*AccPE*).

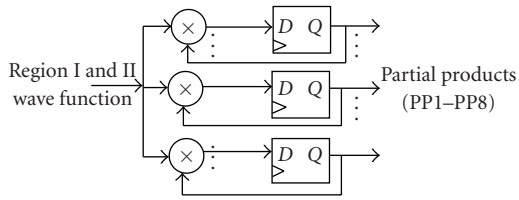


Figure 11: Wave Function accumulation (*AccWF*).

by *CalcWF*. The wave function does not undergo any transformation prior to interpolation. Hence we use one accumulator, which multiplies the intermediate results of wave function contributions from regions I and II. As shown in (5), the total wave function is the product of the pairwise wave functions. Hence the accumulator, that adds the intermediate results, is omitted to conserve FPGA resources. *AccWF* produces eight partial products (*PP1-PP8*) from region I and II wave function results.

We perform successive multiplications of region I potential values and region I and II wave function values. The distances we sample are such that most values of potential energy and wave function are close to one. Hence, repeated multiplication of a number of these values (especially for large N) makes the product tend towards zero. This leads to a loss of precision in a fixed-precision register, with the appearance of leading zeros as the product approaches zero. To guarantee sufficient accuracy, we bit-shift the result to the left after computing the partial product (if it is less than 2^{-1}) and keep track of the number of shifts. The partial results along with the number of shifts are transferred back to the host processor.

There are also overflow issues associated with the accumulator while accumulating region II potential values. To overcome this problem, we accumulate the region II potentials in a large register that can hold N evaluations of the maximum value of the rescaled potential. The accumulated results are processed by the host processor, which reconstructs the floating-point value of the total potential energy and wave function.

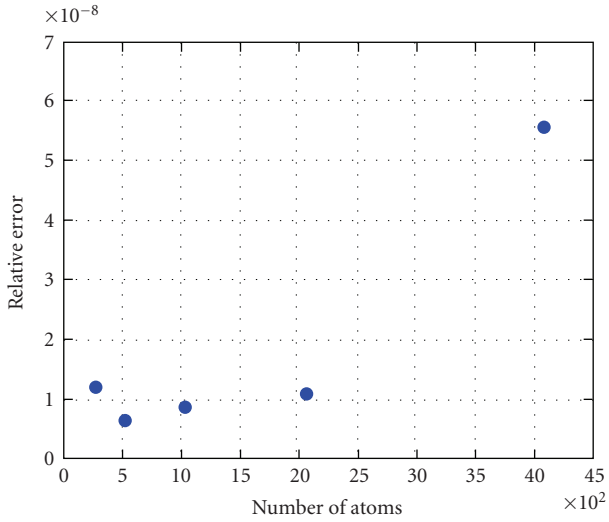


Figure 12: Plot of relative error versus number of atoms.

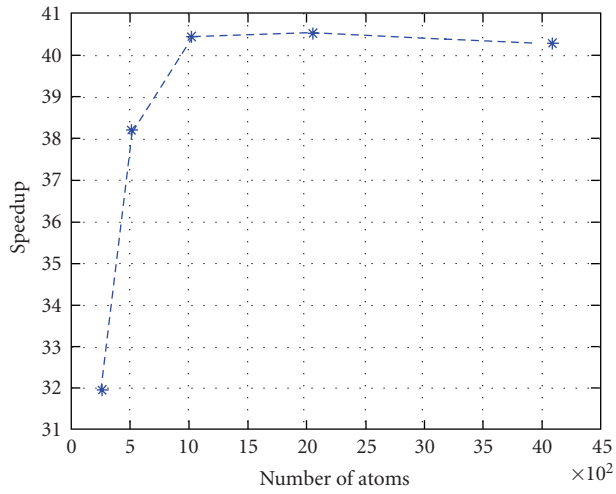


Figure 13: Speed up versus number of atoms.

Table 5: Resource usage on the Virtex-4 FPGA.

Resource type	Virtex-4 LX160 FPGA
SLICES	50%
BLOCK RAMs	51%
DSP48s	52%

4. Results

We target the FPGA implementation to the Cray XD1 high performance reconfigurable computing system [5]. This system incorporates Xilinx FPGAs to its compute nodes. We use a single-chassis Cray XD1 consisting of six compute nodes. Each compute node is equipped with a Xilinx Virtex-II Pro XC2VP50 or Virtex-4 LX160 FPGA [7, 9]. We use the compute node consisting of the Virtex-4 LX160 connected to a dual-processor dual-core 2.2 GHz AMD Opteron. The FPGA and processor are connected by the high-speed RapidArray interconnect, providing a bandwidth of 1.6 Gbytes/sec. As our baseline, we use an entirely software QMC application which uses double-precision that runs on the one core of the Opteron. We use the Scalable Parallel Random Number Generator (SPRNG) library [10] to generate the random numbers for perturbing the atom positions in Step 2 of the QMC algorithm as described earlier. Cray provides API functions that allow us to read from or write to the BRAMs and registers within our software application on the Opteron, simplifying the process of rewriting the software application to use the FPGA accelerators for the most computationally intensive functions. To port the hardware-accelerated QMC application, first we modify the software application to initialize the interpolation coefficients onto the BRAMs. We replace the potential energy and wave function calculations with calls to the FPGA. Next, we write the coordinate positions to the BRAMs after which a control signal is sent to the FPGA to begin the pipeline operation. Once the FPGA completes the operation and stores the results in the user registers, the FPGA sends a status signal back to the processor indicating that the results are ready. The results can be read within the hardware-accelerated application using the Cray API. Our hardware-accelerated framework is available from [6].

The design modules are developed using VHDL and synthesized using Xilinx XST tools. The implementation process, consisting of translating, mapping, and placing and routing of the design is done using the Xilinx ISE tools. The user design is clocked at 100 MHz. Table 5 shows the usage of resources—Slices, Block RAMs, and DSP48s (used as multipliers) when the design is targeted to a single Virtex-4 FPGA on one of the computing nodes of the Cray XD1 platform. The design consumes roughly 50 percent of the resources for wave function and potential energy. Careful redesign of the modules should allow us to fit an additional copy of both PE and WF pipelines on a single FPGA. We could also use the remaining resources to fit additional properties of interest, such as the kinetic energy. We compare the numerical results and the execution times of the software QMC application to the hardware-accelerated application targeted to a Virtex-4 LX160 FPGA. Figure 12 shows the relative error of the fixed-point FPGA potential energies compared to the double-precision results obtained on the CPU as we vary the number of atoms. The error performance is acceptable and within tolerance limits for the energies. Figure 13 shows the speed up obtained for the FPGA implementation over a single core of the Pacific XD1. This implementation provides a speed up of 40x over the software application running on a single core of the 2.2 GHz AMD Opteron.

5. Conclusions

We present a pipelined reconfigurable architecture to obtain the potential energy and wave function of clusters of atoms. We outline some of the goals that are critical for our design. From the design choices available for the building blocks of our architecture, we carefully evaluate the choices to obtain

optimal performance, numerical accuracy, and resource usage consumption. Our design choices including the use of a pipelined architecture, fixed-point representation, and a quadratic interpolation scheme to evaluate the function enable us to achieve a significant performance compared to the software version running on the processor. Our hardware design strategy for the Quantum Monte Carlo simulation offers a speed up of 40*x* on the Cray XD1 platform using a single FPGA over the software application while providing the desired numerical accuracy. Present extensions of our work include using multiple FPGAs on the Cray XD1 platform. This will allow us to distribute the configurations in the simulation among the FPGAs thereby exploiting additional parallelism. In this case, we can expect the total speed up to equal the number of FPGA equipped computing nodes times the speed up on a single FPGA node.

Acknowledgments

This work was supported by the National Science Foundation Grant NSF CHE-0625598, and the authors gratefully acknowledge prior support for related work from the University of Tennessee Science Alliance. The authors also thank Mr. Dave Strenski, Cray Inc., for providing them access to the Cray XD1 platform that was used to target our implementation.

References

[1] S. Hauck and A. DeHon, *Reconfigurable Computing: The Theory and Practice of FPGA-Based Computation*, Morgan Kaufmann, San Fransisco, Calif, USA, 2007.

[2] J. M. Thijssen, *Computational Physics*, Cambridge University Press, Cambridge, UK, 1999.

[3] A. Gothandaraman, G. D. Peterson, G. L. Warren, R. J. Hinde, and R. J. Harrison, "FPGA acceleration of a quantum Monte Carlo application," *Parallel Computing*, vol. 34, no. 4-5, pp. 278–291, 2008.

[4] A. Gothandaraman, G. D. Peterson, G. L. Warren, R. J. Hinde, and R. J. Harrison, "Design decisions in the pipelined architecture for Quantum Monte Carlo Simulations," in *Proceedings of the IEEE MidWest Symposium on Circuits and Systems*, August 2008.

[5] Cray Inc., http://www.cray.com/Home.aspx.

[6] A. Gothandaraman, G. D. Peterson, G. L. Warren, R. J. Hinde, and R. J. Harrison, "A hardware-accelerated quantum Monte Carlo framework (HAQMC) for N-body systems," *Computer Physics Communications*, vol. 180, no. 12, pp. 2563–2573, 2009.

[7] Xilinx Virtex-4 FPGA, http://www.xilinx.com/support/documentation/virtex-4.htm.

[8] Xilinx Inc., http://www.xilinx.com/ipcenter/index.htm.

[9] Xilinx Virtex-II Pro FPGA, http://www.xilinx.com/support/documentation/virtex-ii_pro.htm.

[10] Scalable Parallel Random Number Generator (SPRNG) Library, http://sprng.fsu.edu.

Hindawi Publishing Corporation
VLSI Design
Volume 2010, Article ID 156829, 12 pages
doi:10.1155/2010/156829

Research Article

A SiGe BiCMOS Instrumentation Channel for Extreme Environment Applications

Chandradevi Ulaganathan,[1] Neena Nambiar,[1] Kimberly Cornett,[2] Robert L. Greenwell,[1] Jeremy A. Yager,[3] Benjamin S. Prothro,[1] Kevin Tham,[1] Suheng Chen,[1] Richard S. Broughton,[2] Guoyuan Fu,[2] Benjamin J. Blalock,[1] Charles L. Britton Jr.,[1] M. Nance Ericson,[1] H. Alan Mantooth,[2] Mohammad M. Mojarradi,[4] Richard W. Berger,[5] and John D. Cressler[6]

[1] *Department of Electrical Engineering and Computer Science, The University of Tennessee, Knoxville, TN 37996, USA*
[2] *Department of Electrical Engineering, University of Arkansas, Fayetteville, AR 72701, USA*
[3] *Power and Sensor Electronics, Jet Propulsion Laboratory, Pasadena, CA 91109, USA*
[4] *Instrument Electronics and Sensors, Jet Propulsion Laboratory, Pasadena, CA 91109, USA*
[5] *Space Products and Systems, BAE Systems, Manassas, VA 20110, USA*
[6] *School of Electrical and Computer Engineering, Georgia Institute of Technology, Atlanta, GA 30332, USA*

Correspondence should be addressed to Chandradevi Ulaganathan, culagana@utk.edu

Received 6 June 2009; Accepted 2 November 2009

Academic Editor: Ethan Farquhar

An instrumentation channel is designed, implemented, and tested in a 0.5-μm SiGe BiCMOS process. The circuit features a reconfigurable Wheatstone bridge network that interfaces an assortment of external sensors to signal processing circuits. Also, analog sampling is implemented in the channel using a flying capacitor configuration. The analog samples are digitized by a low-power multichannel A/D converter. Measurement results show that the instrumentation channel supports input signals up to 200 Hz and operates across a wide temperature range of -180°C to 125°C. This work demonstrates the use of a commercially available first generation SiGe BiCMOS process in designing circuits suitable for extreme environment applications.

1. Introduction

There are numerous applications prompting interest in developing SoC (System-on-Chip) integrated systems. Space exploration presents a niche area wherein extreme environmental conditions pose several design constraints. At present, robotic exploration rovers have all the critical electronics inside a temperature-controlled Warm Electronics Box (WEB) [1]. This dictates a centralized architecture wherein data processing happens in the WEB and the generated control signals are communicated to various parts of the rover through extensive cables. Developing electronic circuits that are capable of reliable operation under extreme environmental conditions facilitates the use of a distributed architecture. This eliminates extensive cabling and thus increases the system reliability. In addition, significant reductions in power consumption, launch volume, and weight (and therefore, launch cost) are achieved.

The objective of this work is to develop a highly integrated front-end and signal processing circuitry capable of reliable operation in extreme conditions. In order to efficiently handle data, instrumentation channels (referred to as "universal" channels in this work) are required to interface external sensors with signal processing circuitry. Also, the use of a multi-channel analog-to-digital converter (ADC) aids in allowing high levels of circuit integration. In this work, numerous external sensors are interfaced to a multi-channel ADC via the universal channels as the front-end interface.

This paper presents the design and implementation of a universal channel suitable for low speed signals. Measurement results from the channel are also discussed. Section 2 presents an overview of the channel and its components. Section 3 discusses in detail the design and implementation of the individual components of the universal channel. The design techniques for reliable operation in extreme environment are discussed in Section 4. Section 5 presents

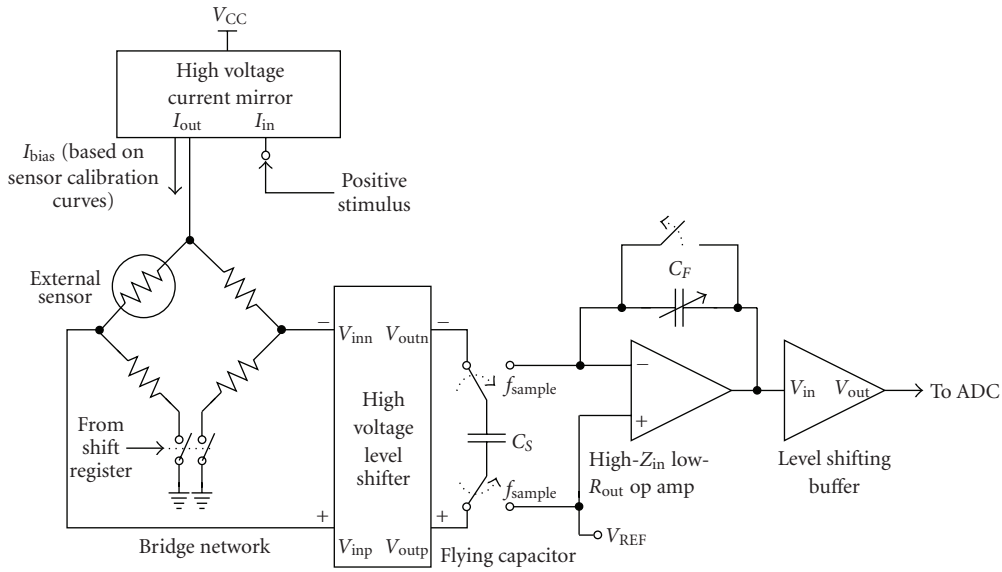


Figure 1: Simplified schematic of the universal channel.

the measurement results and finally Section 6 concludes this work.

2. Universal Channel—An Overview

The universal channel is a low-speed instrumentation channel that is capable of interfacing signal processing circuits with various transducers including thermocouples, RTDs (Resistance Temperature Detectors), pressure gauges, and strain gauges. The universal channel is designed to operate across a wide temperature range that includes −180°C through 125°C. Radiation-hardening-by-design (RHBD) circuitry is implemented in the digital circuits while special layout techniques render the analog circuits in the channel to be radiation tolerant. Numerous industrial channel designs [2–4] are available; however, this is the first multi-channel custom IC designed to operate across a wide temperature range extending down to cryogenic temperatures reported in the literature. Several versions of the universal channel were successfully fabricated and tested with the most recent version fabricated and tested in mid-2009 [5].

The channel is capable of supporting high-voltage sensors of up to 12 V and produces a 12-bit digital output from an ADC. Figure 1 shows, the Wheatstone bridge, the high-voltage level shifting buffer, the flying capacitor network, and the level shifting buffer form the front-end of the analog sampling channel.

The Wheatstone bridge is designed to be reconfigurable in order to interface with a variety of external transducers. The bridge includes three matched 350-Ω resistors that are used in combination with an external resistive sensor to make a complete Wheatstone bridge. The circuit works on the principle that any change in the bridge resistance, corresponding to a physical change in the sensor, is converted into a differential voltage that is sampled and processed.

The high common-mode voltage differential output from the Wheatstone bridge is level shifted to the low-voltage level (below 3.3 V) by a high-voltage level shifting buffer.

The flying capacitor network performs the analog sampling in the universal channel and provides high input common-mode rejection. The circuit includes a "flying" or sampling capacitor and uses a general purpose high-Z input operational amplifier (op amp) within a switched capacitor configuration to provide programmable gain. The flying capacitor's built-in calibration and correction circuitry, along with the low offset drift op amp, ensure that the sensor signals are sampled and transmitted to an input channel of the Wilkinson ADC [6]. Following this stage, a level shifter shifts the common-mode voltage of the channel to the center of the input range of the Wilkinson ADC, which is the next signal processing stage.

The sampled analog signal from the flying capacitor network is converted to digital data using the Wilkinson ADC. The 12-bit 16-channel Wilkinson ADC is a low-power, high-resolution, analog-to-digital converter capable of conversions at 20 KSps across the wide temperature range of −180°C to 125°C.

3. Design and Implementation

This section describes the design and operation of the individual blocks of the universal channel as well as the Wilkinson ADC.

3.1. Wheatstone Bridge Network. The Wheatstone Bridge Network is designed to interface with various sensors. Figure 2 presents a simplified schematic of the Wheatstone bridge network. This programmable bridge has three signal connections, two of which are used in any given configuration. This structure, in conjunction with the two switches in the bottom legs, provides the flexibility to have different

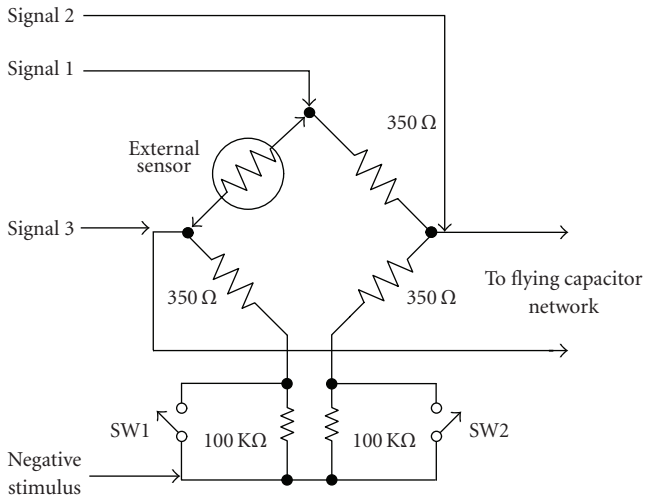


Figure 2: Schematic of the programmable Wheatstone bridge network.

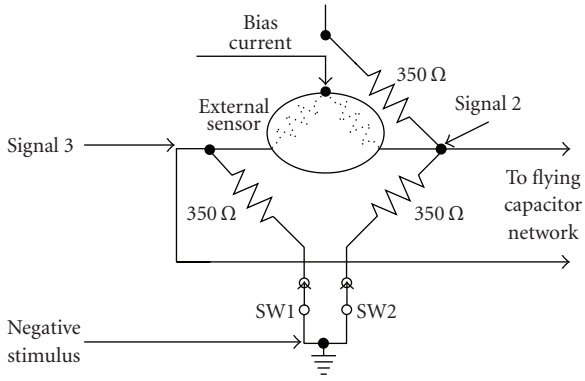


Figure 3: Wheatstone bridge network in half-bridge configuration.

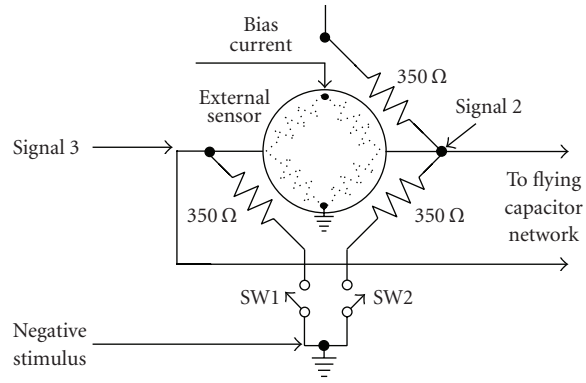


Figure 4: Wheatstone bridge network in full-bridge configuration.

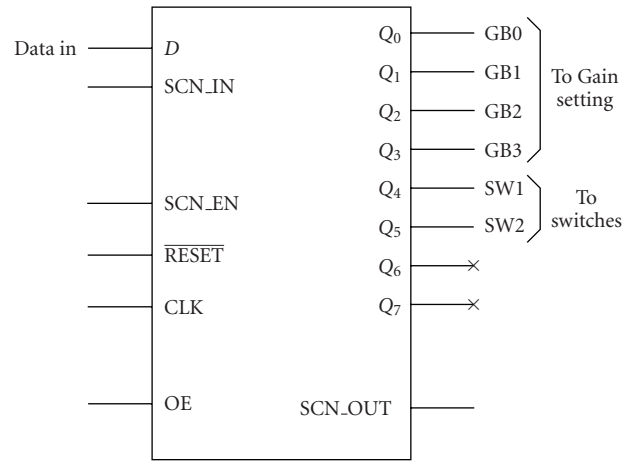


Figure 5: Pin-out of switches and gain setting control shift register.

configurations that interface with either full-bridge, half-bridge, or quarter-bridge resistive sensors.

By carefully constructing the layout, the 350-Ω resistors in the bridge are closely matched, making it relatively insensitive to temperature [7]. The bridge is biased by an external current source or an on-chip stimulus current that provides up to approximately 30 mA of bias current (Figure 1). The wiring of the bridge is designed to handle up to 35 mA of current. The n-type single-pole single-throw switches, at the bottom legs of the bridge, are designed to provide a low ON resistance (less than 10 Ω) over the operating temperature range. Since matching between the switch resistances is critical, common-centroid layout techniques are utilized. These switches set the bridge configuration used for measurements.

3.1.1. Control and Operation. The Wheatstone bridge network has three input terminals connected to the corners of the bridge. The fourth corner is connected to the "negative stimulus" terminal (typically tied to ground). The differential output voltage is sampled between *Signal 2* and *Signal 3* by the subsequent flying capacitor network.

The configuration of the bridge is set by controlling the switches SW1 and SW2. In the quarter-bridge and half-bridge configurations, both switches (SW1 and SW2) are closed. As depicted in Figure 2, the quarter-bridge configuration has an active external sensor connected between *Signal 1* and *Signal 3* as the upper left element of the bridge. The on-chip resistor network provides the bottom two resistor elements and the upper right element. With this setup, temperature variations and lead resistance reduce the accuracy of the sensor measurements. Using a dummy sensor in a half-bridge configuration mitigates these effects. Single element and lead compensated sensors may be used in this configuration [8].

In the half-bridge configuration (Figure 3), two active external sensors are connected between *Signal 2* and *Signal 3* to provide the upper half of the bridge, while the bottom half of the bridge is provided by the on-chip resistors. This configuration provides better sensitivity along with inherent temperature and lead compensation [8].

By setting both switches SW1 and SW2 to open, the on-chip resistors are not used. Four active external sensors are connected between *Signal 2* and *Signal 3* as a full bridge to provide a signal to the flying capacitor network (Figure 4).

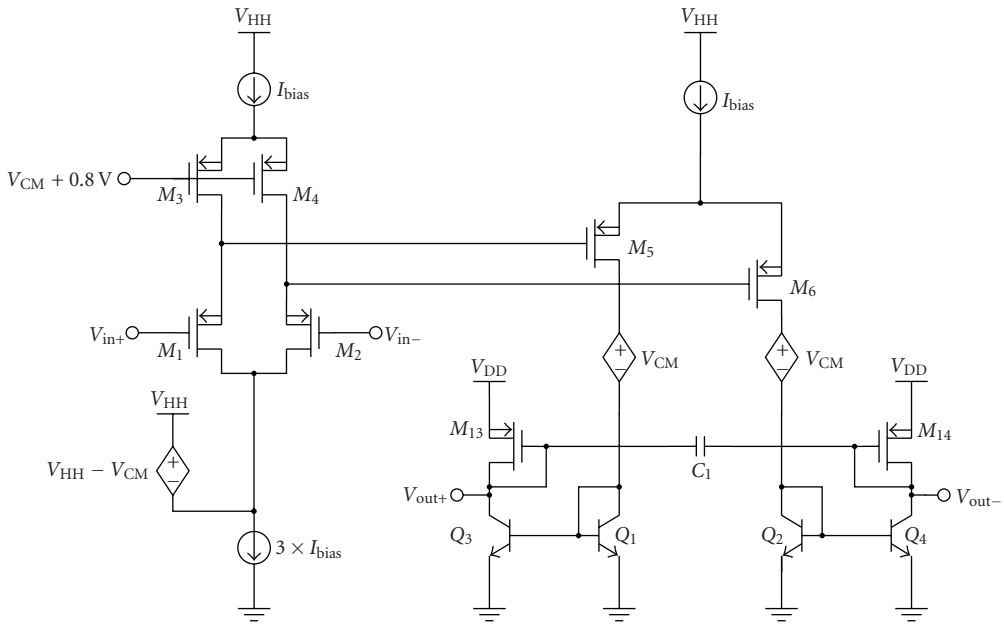


Figure 6: Simplified schematic of the high-voltage level shifter.

This configuration is known as a full-bridge configuration. This means that the sensor has a full-bridge inside it.

As shown in Figure 1, an on-chip stimulus current (positive stimulus) is applied to the top of the bridge. The stimulus functions as an adjustable current source that sets the common-mode voltage of the bridge. This current source is driven by a high-voltage current mirror that allows for stimulus voltages higher than those supported by the 3.3-V process. A current output digital-to-analog converter (DAC) is utilized to supply the stimulus input current to this current mirror. The circuit provides a 100× multiplication. The available bridge bias current ranges from 1 to 30 mA.

The bridge configuration and/or the gain setting of the channel are controlled by a serially loaded shift register. Figure 5 provides the pin-out of the control register. When a channel is integrated with other active channels in an application, the shift register is included in a scan path for test mode. Shift registers are also employed to program the stimulus and calibration data during testing and setup. These serial shift registers load the corresponding DACs. All of the shift registers contain multiplexers on each output bit. When the Output Enable (OE) control signal is *low*, the output bits are connected to ground irrespective of the data stored or being loaded into the register's latches. This feature protects the rest of the channel's circuitry as the data is loaded. When the data is fully loaded, Output Enable is set *high*. With Output Enable on *high*, the internal multiplexers will output the data that is stored in the register's latches.

3.2. High-Voltage Level Shifter. The high-voltage level shifter is a unity-gain buffer that translates the small differential output voltages (±200 mV) from the Wheatstone bridge network across a wide input common-mode range (V_{CM}) of 0–10 V to a differential signal at a lower common-mode voltage range of 1.8–2.2 V. This allows the rest of the analog processing and data conversion to occur at low voltages. A simplified schematic is illustrated in Figure 6.

The basic functionality of the circuit is as follows: the differential signal is fed into the gates of the dual level shifters formed by M_{1-4}. After level shifting (which ensures operation at low common-mode range), the differential voltage signal is converted to current by M_5 and M_6, and passed down through the V_{CM} level shifters into the current mirrors formed by Q_{1-4}. Once mirrored into the output branches, the differential signal is converted to a voltage by diodes M_{13} and M_{14}, where it is read by standard low-voltage CMOS analog processing circuitry. The pMOS transistors forming the high-side current sources are protected from breakdown by high-voltage N-wells as described by Najafizadeh et al. [9]. The low-voltage current source is formed by LDMOS transistors available in-process, also as described by Najafizadeh et al. [9].

3.3. Flying Capacitor Network. The flying capacitor network [10–12] used for sampling the analog input is implemented as shown in Figure 7. The main components of the circuit include sampling capacitors, C_S, holding capacitor, C_f, and a general purpose high-Z input op amp within a switched capacitor configuration. This circuit incorporates calibration and charge cancellation schemes in order to eliminate pedestal effects from parasitic capacitors and charge injection from the control switches. The array of feedback capacitors used for the op amp provides programmable gain (see Figure 8). During the channel calibration, the channel gain is programmed from −0.6 to −10 V/V using the desired combination of the gain switches. As discussed earlier, the switches are set by the shift register illustrated in Figure 5. An internally generated reference voltage, V_{REF}, is applied to

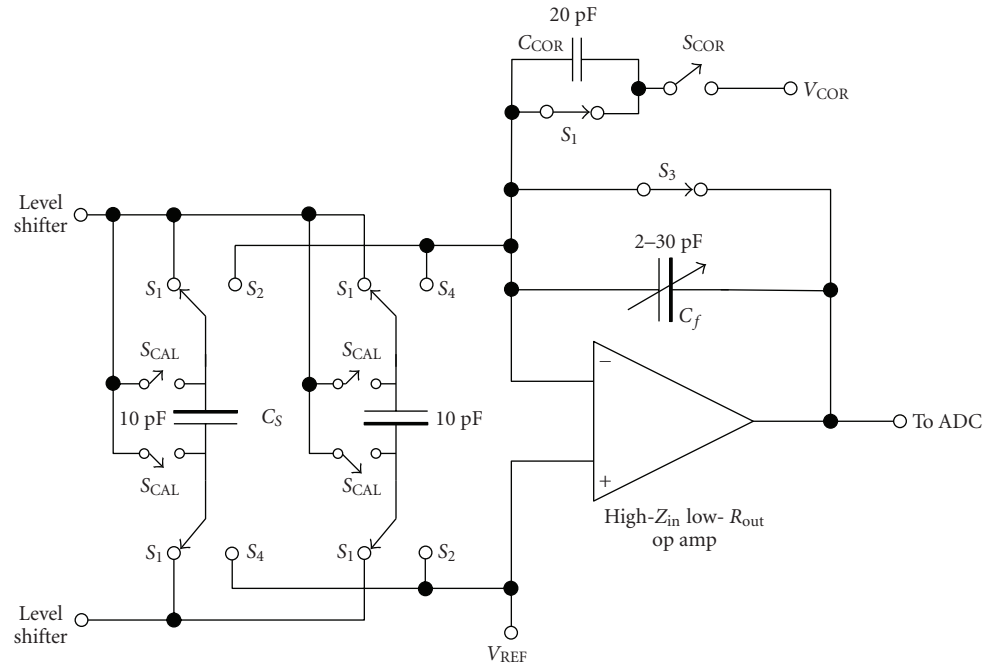


Figure 7: Schematic of the flying capacitor network.

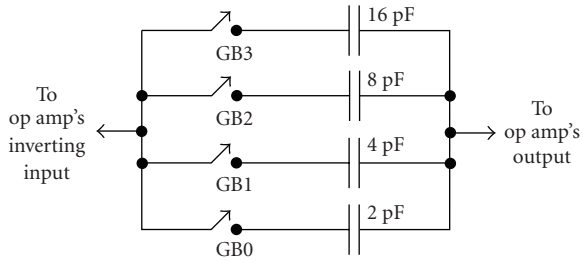


Figure 8: Programmable gain circuitry in the flying capacitor network.

the op amp to set the input common-mode voltage. Op amp offset voltage correction is implemented using an adjustable reference voltage, V_{COR}. The V_{COR} voltage is generated by a voltage DAC that is initialized using a serially loaded shift register during the calibration/setup of the channel. When the reference voltages are equal, the op amp's offset is compensated. Using a value for V_{COR} that is different from the reference voltage, V_{REF}, enables offset correction for the entire channel.

The general purpose op amp [13] provides the channel gain and is also employed as a level shifter to match the input range of the ADC. Since the ADC has 12 bits of resolution, the op amp is designed to provide at least 80 dB gain, low noise, and low offset drift. The schematic of the op amp is depicted in Figure 9. The amplifier senses input voltages near the lower supply and also provides rail-to-rail output voltage swing. The output stage is based on the topology presented in [14] and provides good drive capability while using relatively small transistors. Measurement characterizations of this op amp indicate that it provides a stable DC open-loop gain of 80 dB and gain-bandwidth product of 4 MHz across a wide temperature range of −180°C to 125°C [13]. Preliminary proton radiation testing induced negligible variation in its performance [15].

3.3.1. Timing Control and Operation. The switches control the different modes of operation of the flying capacitor network. Figure 10 provides the control signals that coordinate the switching. All switches employ transmission gates to reduce charge injection and eliminate threshold voltage effects on logic levels inherent to nMOS-only or pMOS-only gates.

When the circuit's operation begins with calibration mode, the sampling and holding capacitors are reset to 0 V and the op amp is biased in unity-gain mode. Switches S_{CAL} and S_3 are closed during this stage. The cross-coupling of the sampling capacitors aids in cancellation of charge from parasitic capacitance. Also, the hold capacitor (C_f) has its bottom plate connected to the output of the op amp rather than the inverting input, thereby reducing parasitic capacitance at the input of the op amp.

Calibration is followed by a calibration-hold stage (Figure 11). This phase incorporates a prehold phase for charge cancellation. Further, the switches have a "break-before-make" action which is essentially nonoverlapping control signals that aid in discharging the parasitic capacitors. The order of switching begins with the opening of the S_{CAL} switch and the closing of the S_2 switch. With switch S_2 closed, harmless discharge paths are formed for the parasitic capacitors found at the bottom and top plates of the sampling capacitors, whereas the sampling capacitors retain their charge. After the parasitic discharge is complete, switch S_3 opens to place C_f in the op amp's feedback loop.

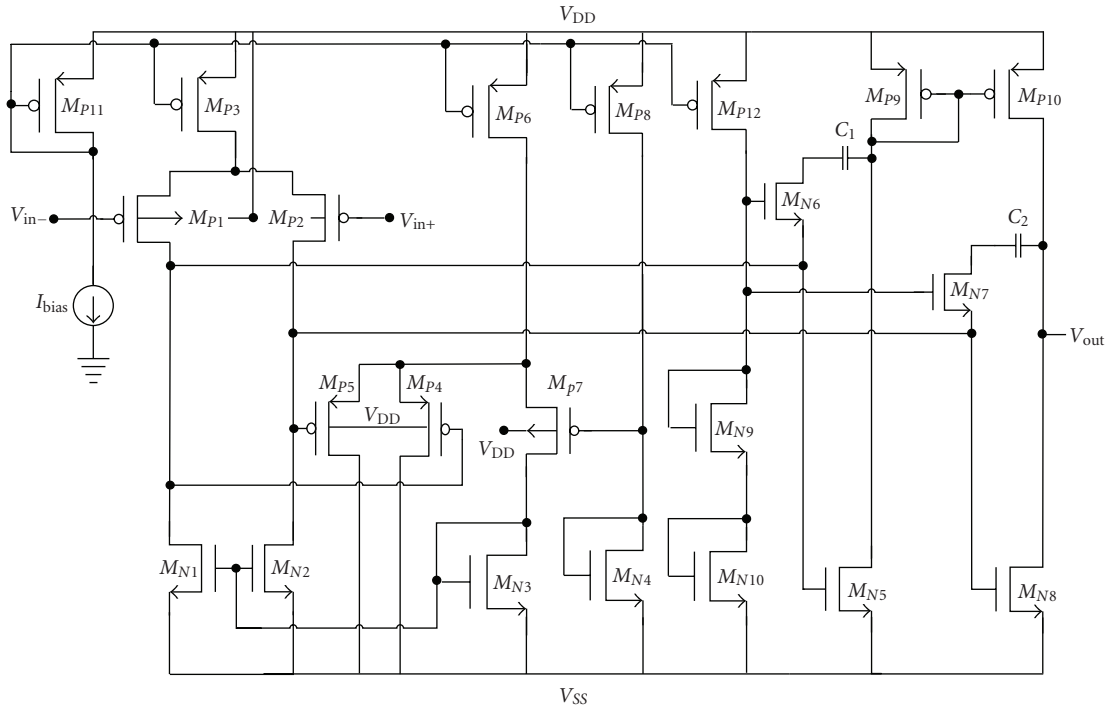


Figure 9: Schematic of the general purpose op amp [13].

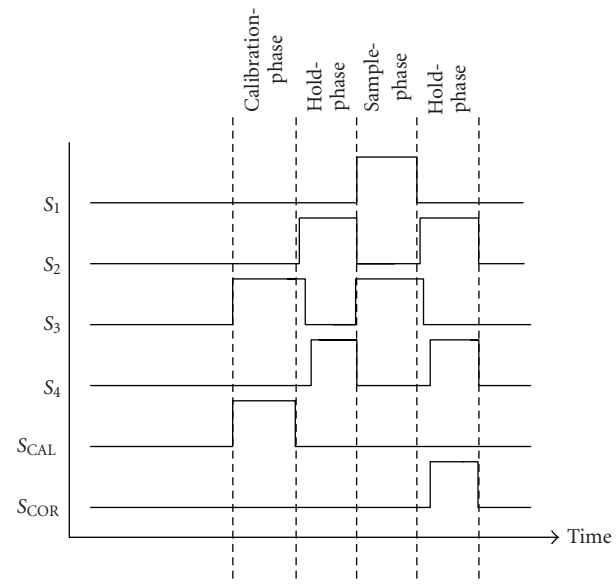


Figure 10: Coordination of timing signals for the flying capacitor circuit.

Then switch S_4 closes which allows charge from the sampling capacitors to move to the feedback capacitor. As a result, now the output of the op amp reflects the op amp's offset voltage.

After a cycle of calibration and hold, the circuit enters the sampling phase in which the switches S_1 and S_3 are closed and all others are opened. Figure 7 illustrates this phase. The sampling capacitors charge to the bridge's differential output voltage while the offset correction capacitor, C_{COR}, is reset. Next, the circuit enters the hold stage (Figure 12), in which the parasitic charges are canceled in the pre-hold phase and followed by the transfer of charge from the sampling to the holding capacitor, C_f. Also, C_{COR} is connected to a voltage equal to V_{REF} so that the voltage across the capacitor C_{COR} is equal to the op amp's offset voltage. With this correction voltage applied to the op amp's inverting input, the output voltage during the hold stage is V_{REF}-V_{CF}, where V_{CF} is the sampled analog input. Thus, the flying capacitor circuit performs analog sampling and also incorporates offset and parasitic charge cancellation.

The V_{REF} voltage is internally generated and V_{COR} is set during channel calibration. The switched capacitor's control signals, the 6-phase clocks, are brought into the chip from external sources. The next version of this chip includes a 6-phase clock generation circuit that provides the necessary control signals from a single input clock that is synchronized with the chip's master clock. This improves the signal integrity by minimizing crosstalk, substrate noise injection, and *IR* drop from long routes. It also reduces the required number of external pins. The 6-phase clock generator circuit halves the input clock frequency and employs it as an additional phase reference to generate the required control signals. Current-starved inverter delay cells [16] are used to provide the required delays for the non-overlapping clock phases.

In order to match the output range of the flying capacitor stage with the input range of the ADC, an op amp-based level shifter is employed. Figure 13 shows the two-stage level shifter that uses the general purpose high-Z input op amp

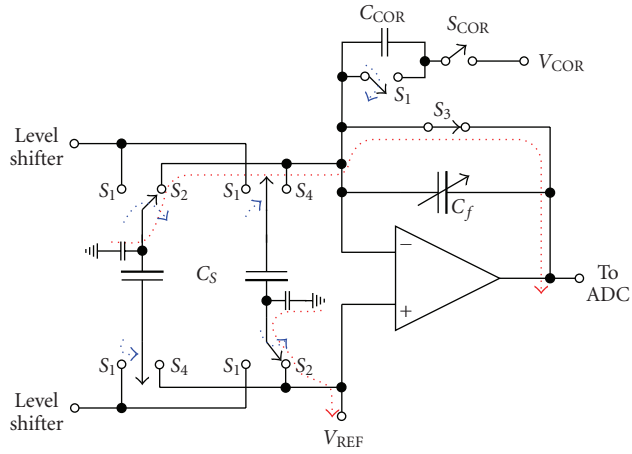


Figure 11: The flying capacitor network in charge cancellation mode in "pre-hold phase".

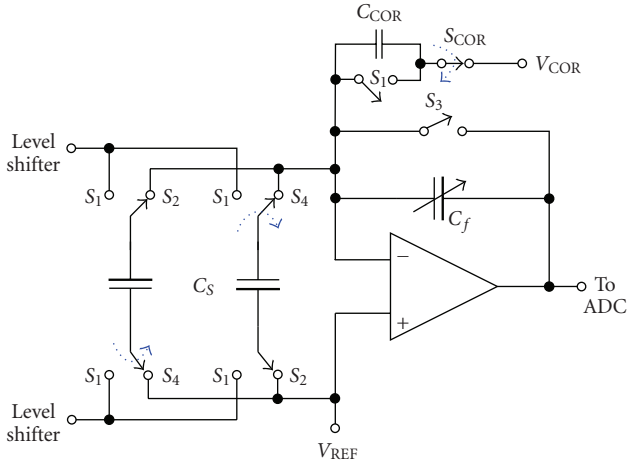


Figure 12: The flying capacitor network in hold phase.

of Figure 9. The first stage sets the required dc bias for the next stage such that the output of the second stage is centered about 0.6 V. This op amp configuration also increases the gain by a factor of two. A simplified schematic of the full universal channel is illustrated in Figure 13.

3.4. Multichannel Wilkinson ADC. The universal channel is interfaced to one of the 16 channels of the Wilkinson ADC [6]. The architecture of the Wilkinson ADC is based on [17]. The functional block diagram of the ADC is provided in Figure 14. The main components of the ADC include a single ramp generator and a single 12-bit Gray code counter that are shared across the multiple channels, thereby reducing the total power consumption. Additionally, an auto-zeroing comparator is integrated within each channel. This low-power comparator also incorporates the analog sampling function within the ADC.

3.4.1. Timing Control and Operation. To function properly, the ADC depends on precise synchronization of various signals that control the key components of the ADC. The clock to the Gray code counter is supplied via a low-voltage differential signaling (LVDS) link [18] to minimize the noise injected into the system by the clock. Figure 15 illustrates the timing sequence of the control signals for a conversion cycle. The analog signal from the universal channel is input directly to the comparator.

The conversion begins with a sampling phase, wherein the analog input is sampled while also auto-zeroing the comparator's offset. Signals *ICS* and *AZ* control this phase. The integration capacitor in the ramp generator is also reset during this phase by the *RAMP_FB* signal. Next, the comparator is connected to the ramp generator by the *RAMP_CONNECT* signal. The actual start of conversion begins with the release of the *RAMP_FB* switch and the start of the counter by the *COUNTER_ENABLE* signal. This generates a ramp voltage that linearly varies from 0 to 1.2 V that is proportional to the digital count. When the sampled voltage becomes equal to the ramp voltage, the comparator trips. The value of the counter is stored on latches for reading out the data. During the next sampling/auto-zero phase, the stored data is moved to another set of storage latches by the *REFRESH* signal and the first set of latches are reset by the *RESET* signal, thus preparing for the next conversion. Multiple universal channels can be connected to the ADC and the data can be read out using the channel select signals.

Since the flying capacitor network produces a discrete time output that is valid only during the hold phase (Figure 12), proper synchronization is essential to ensure that the ADC samples reflect the sensor output. This is accomplished by generating all the control signals from the same master clock through clock dividers. Thus the ADC's input sampling signal, *ICS*, is enabled only when the flying capacitor's S_{COR} signal is high.

4. Discussion

Designing circuits for extreme environment operation demands implementation of numerous design strategies at

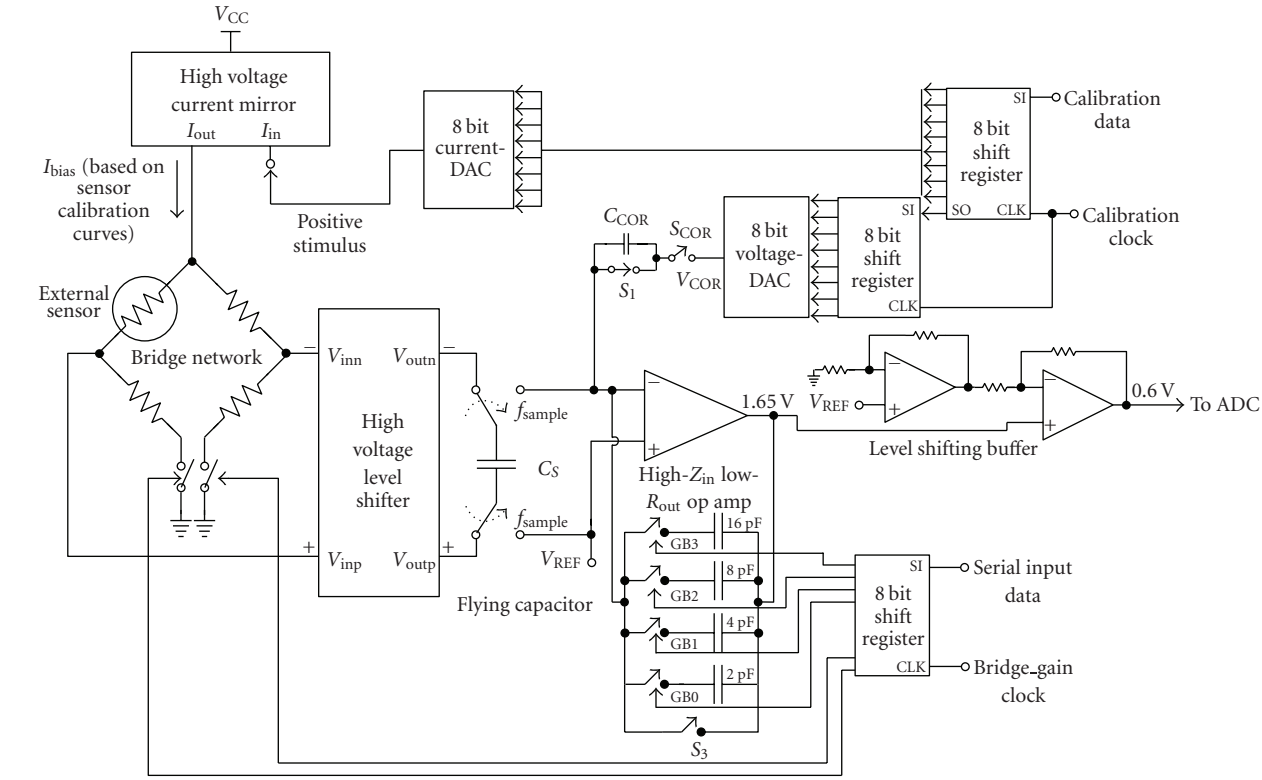


Figure 13: Simplified schematic of the Universal Channel along with the control blocks.

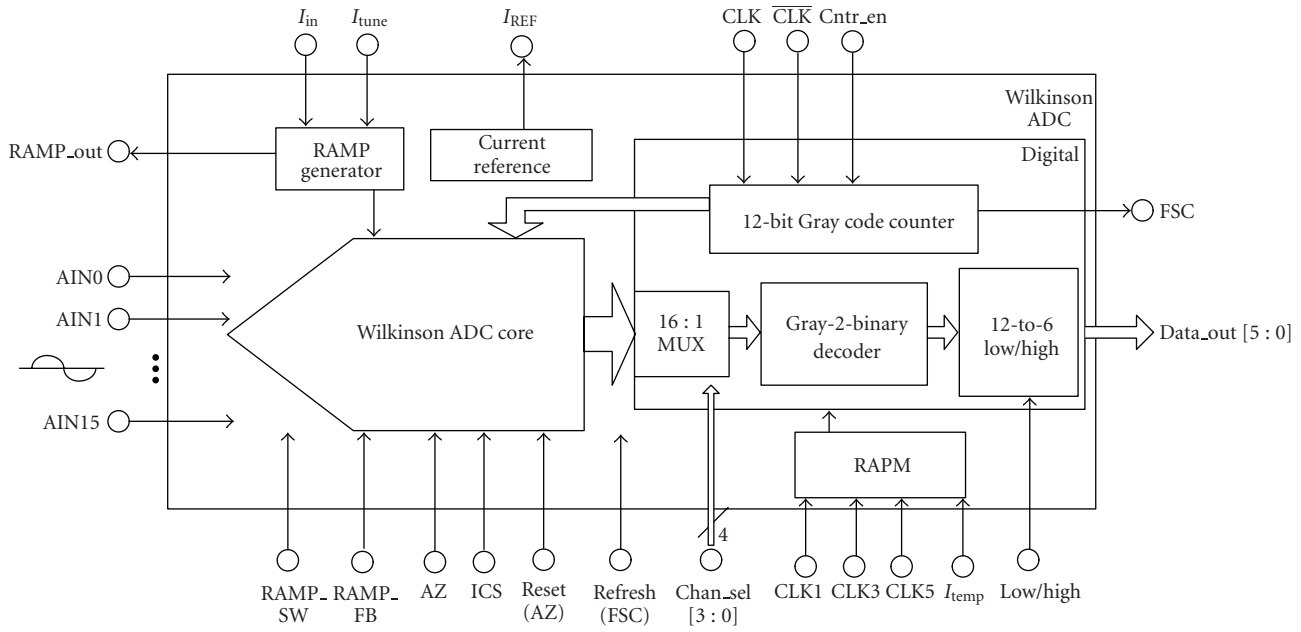


Figure 14: Functional block diagram of Wilkinson ADC.

the system level as well as at the individual circuit level, along with special layout techniques. The individual circuits in the channel are designed to operate reliably across the required temperature range of −180°C to 125°C. The choice of bias circuits is critical for the circuit's performance across temperature. This mixed-signal system includes a constant-current reference and a constant-inversion coefficient (IC) [19] current reference [20] for providing the bias currents. Using the constant-IC current reference to bias the op amps helps maintain a constant MOSFET IC across temperature,

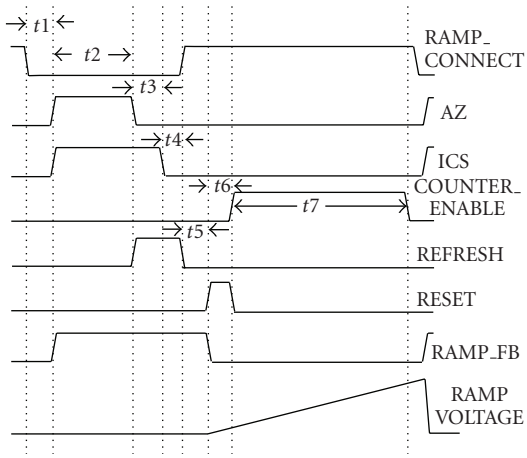


Figure 15: Timing diagram for the ADC.

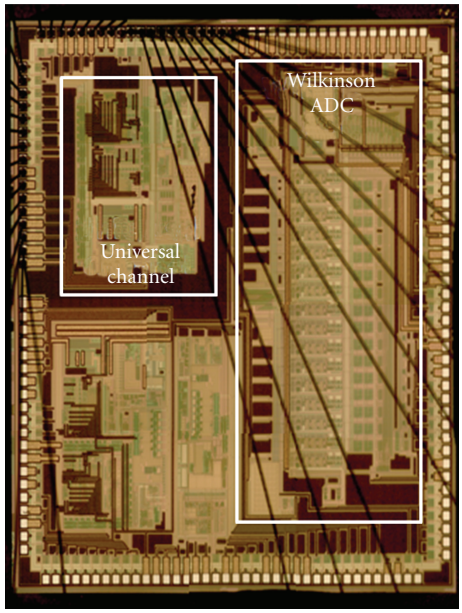


Figure 16: Die Photo of the chip (6×6 mm^2).

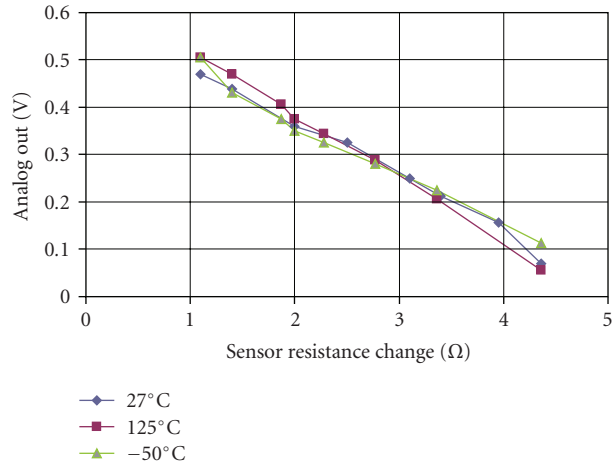


Figure 17: Output voltage versus resistance of the sensor at various temperatures. (quarter-bridge configuration, gain = −20 V/V) [5].

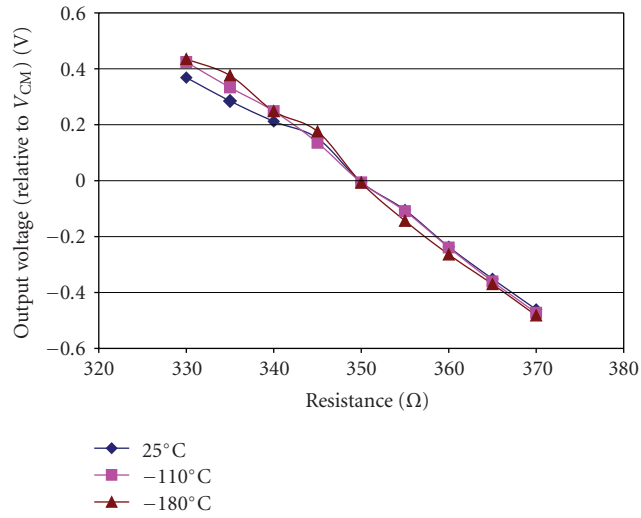


Figure 18: Output voltage versus resistance of the sensor at various temperatures. (quarter-bridge configuration, gain = −10 V/V).

thus minimizing performance variations. Circuits such as the ADC's comparator and LVDS link that require a constant bias current employ the constant-current reference circuit. The next version of this chip also includes a temperature compensation scheme within the ADC that maintains the accuracy across temperature.

Reliability and device life time depend on the operating temperature and tend to degrade at low temperatures. SiGe HBTs offer better performance across temperature and ionizing radiation when compared to CMOS devices [21]. Further, pMOS transistors exhibit better performance at cryogenic temperatures than nMOS transistors [22]. Thus, circuits designed with only pMOS transistors and HBTs operate more reliably across extreme environmental conditions. The reliability of CMOS circuits is improved by using longer channel length devices that reduce the increase in substrate current at low temperatures caused by the hot carrier effect [23]. To meet the device lifetime reliability specification in CMOS circuits for this work, all the 3.3-V powered nMOS transistors have a channel length $\geq 1\,\mu$m.

Radiation effects include total-ionization dose (TID), single-event upset (SEU), and single-event induced latch-up (SEL) [24]. In order to mitigate SEU, the digital circuits in the universal channel employ radiation-hardened-by-design (RHBD) cells such as DICE flip-flops and latches [25]. For the ADC's low-power Gray code counter (where DICE cells could not be used), local redundancy with voting logic is implemented to provide SEU tolerance. The TID effects are mitigated by layout techniques. The nMOS and pMOS transistors are surrounded by guard rings to alleviate the TID-induced leakage currents. The guard rings, along with n-well and substrate contacts, provide isolation to different transistors as well as circuits. This offers latch-up immunity and lower substrate noise coupling. Use of deep trench

isolation (DTI) also enhances the latch-up immunity of the circuits.

The low-power ADC also incorporates system-level SEL detection and mitigation using a radiation aware power management scheme. This scheme includes a voltage regulator that provides the power supply for the ADC's digital section. In the event of an SEL, the supply voltage is automatically collapsed and then restored, resetting the digital blocks. Therefore, SEL mitigation is implemented at the system level.

Electromigration is another failure mechanism that is addressed, wherein at high-current densities and temperatures, interconnects experience degradation due to the transport of metal ions [26]. The selected process's electromigration rules that govern current density limit, wire width, and minimum required contacts are strictly followed to improve the reliability and the life time of metal connections.

5. Measurement Results

Previous versions of the universal channel along with the 12-bit 16-channel Wilkinson ADC were fabricated and verified in a 0.5-μm SiGe BiCMOS process. The die photo of the most recently tested circuits is shown in Figure 16. The layout is designed to minimize noise and crosstalk in the chip. The analog and digital sections have separate power supplies and are isolated by guard rings. The high-frequency clock signal for the ADC is supplied as a complementary signal and is well shielded from other signals. Also, the bottom plate of the op amp's feedback capacitors in the flying capacitor circuit is connected to the output of the op amp so that the substrate-to-bottom plate crosstalk is reduced due to the low output impedance of the op amp.

A multilayer test board is designed to test the universal channel and the ADC. The control signals are generated from a master clock using an onboard reprogrammable FPGA. Separate ground planes and routing layers are provided for the analog and digital signals. Variable resistors are used as an input to the channel and the output of the ADC is recorded using a National Instruments data acquisition interface card [27].

The measurements of the channel are taken at different temperatures across the range of −180°C to +125°C. The output voltages of the channel, depending on the resistance of the sensor in the quarter-bridge configuration at various temperatures and gains, are shown in Figures 17, 18, and 19. Depending on the gain setting of the channel, the system can measure a maximum resistance change of approximately 67 Ω, and a minimum change of less than 1 Ω.

Figure 20 provides the output voltage of the channel across temperature. The variation in the output voltage results from the bridge becoming unbalanced when the resistance of the sensor remains unchanged at room temperature while the rest of the bridge varies with temperature.

The operation of the on-chip stimulus current is verified across temperature. The output current is measured by sweeping the current DAC over the 8-bit range. Figure 21 depicts the measured stimulus current at +27°C and −174°C. The simulation data at +27°C is also shown in this figure.

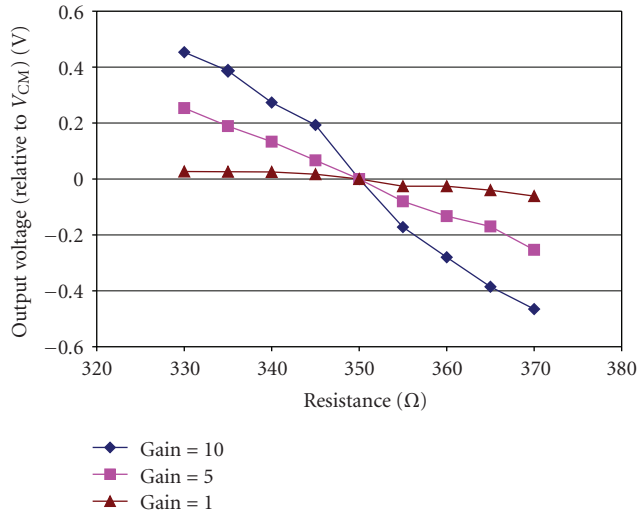


Figure 19: Output voltage versus resistance of the sensor at different gains. (quarter-bridge configuration, temp = −180°C).

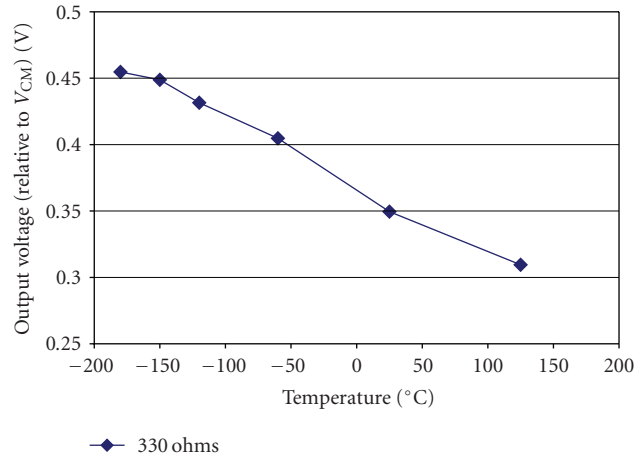


Figure 20: Output voltage versus temperature. (quarter-bridge configuration, gain = −10 V/V).

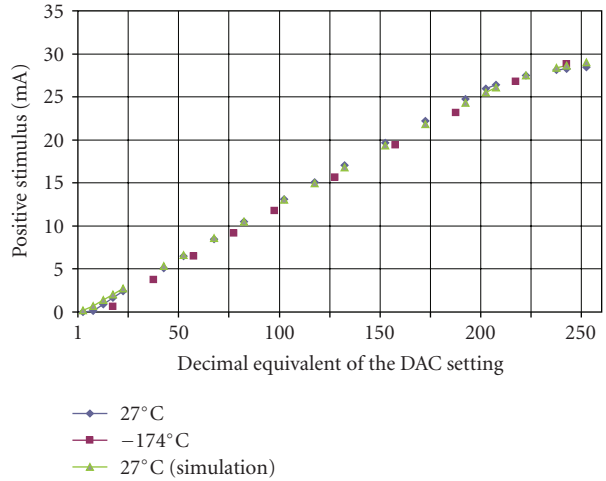


Figure 21: Measured and simulation data of the positive stimulus over the current-DAC 8-bit range [5].

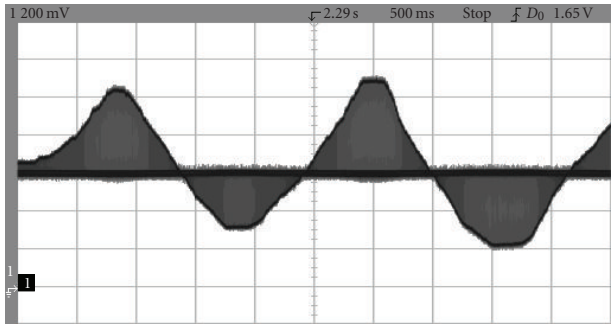


Figure 22: Measured output from universal channel at room temperature with manually adjusted sensor.

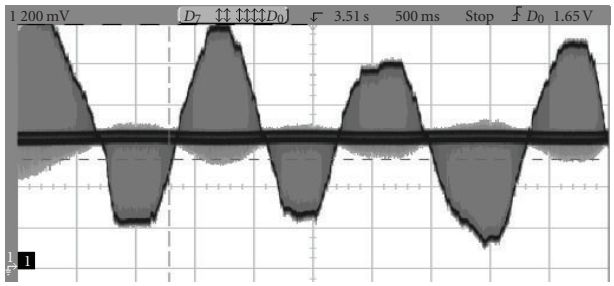


Figure 23: Measured output from universal channel at −174°C with manually adjusted sensor.

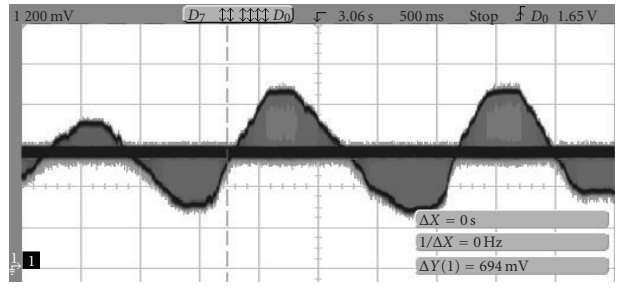


Figure 24: Measured output from universal channel at 125°C with manually adjusted sensor.

Table 1: Estimated power consumption.

Circuit Block	Simulated Average Power (mW)
Wheatstone Bridge	12–384 (204 typ.)
Current DAC	4
High-Voltage Level Shifter	1.2
Bridge Configuration circuits	4
Flying Capacitor Network	2.2
Level Shifting Buffer	3
ADC	≈1.1 per channel
Total Power	28–400 (220 typ.)

A potentiometer is used as a quarter bridge sensor to interface with the channel. This potentiometer is manually adjusted to represent a dynamically changing sensor. Figures 22, 23, and 24 illustrate the measured analog output of the channel before it is applied to the ADC. During the five seconds that these images were captured, the potentiometer was manually adjusted to provide an oscillating sensor input signal. As seen in the figures, the channel correctly samples the sensor as the resistance of the potentiometer is adjusted. Across the wide temperature range of +125°C to −174°C, the universal channel demonstrates full sampling functionality. The channel is designed for low-speed sensors that support input signals up to 200 Hz. The sampling frequency of the channel is approximately 20 kHz in these figures. Characterization of the Wilkinson ADC shows the differential-non-linearity (DNL) to be less than ±0.5 LSB across the temperature range [6].

Table 1 provides the estimated power consumed by the individual blocks and the total estimated power consumption of one universal channel. The bridge configuration circuits include the voltage DAC and three shift registers used for the calibration and configuration of the channel. It is evident that the input stage of the channel dominates the total power consumed. This is set by the bias requirements of the sensor.

In the final version of the universal channel and Wilkinson ADC, additional wide temperature testing of the circuits is expected to demonstrate full operation.

6. Conclusion

The design of the instrumentation channel requires many novel techniques in order to make it stable and reliable across a wide temperature range. This paper demonstrates the operating principles and experimental results of a channel which can be used in sensor data acquisition applications down to cryogenic temperature. In the future, the instrumentation channel will be integrated into a 16-channel Remote Electronics Unit Sensor Interface (RSI) ASIC for use in space applications.

Dedication

This paper is dedicated to the memory of Hung Hoang, Department of Electrical Engineering, University of Arkansas, Fayetteville, who contributed to the development of this work.

Acknowledgments

This work was supported by the NASA ETDP program, under Grant NNL06AA29C—Silicon Germanium Integrated Electronics for Extreme Environments. The authors are grateful to M. Watson, A. Keys, and the SiGe ETDP team led by Professor J. Cressler of Georgia Tech for their contributions.

References

[1] The NASA, "Mars exploration rover mission," http://marsrovers.nasa.gov/mission/.

[2] S. A. Kleinfelder, W. C. Carithers Jr., R. P. Ely Jr., et al., "A flexible 128 channel silicon strip detector instrumentation integrated circuit with sparse data readout," *IEEE Transactions on Nuclear Science*, vol. 35, no. 1, pp. 171–175, 1988.

[3] P. H. Garrett, *Multisensor Instrumentation 6σ Design*, John Wiley & Sons, New York, NY, USA, 3rd edition, 2002.

[4] BIOLAB, "Industrial technologies division," http://www.davidson.com.au/products/strain/mg/instruments/system-6000.asp.

[5] K. Cornett, *Design and verification of an integrated sensor interface for extreme environment applications*, M.S. thesis, University of Arkansas, Fayetteville, Ark, USA, 2009.

[6] N. Nambiar, C. Ulaganathan, S. Chen, et al., "SiGe BiCMOS 12-bit 8-channel low power Wilkinson ADC," in *Proceedings of the Midwest Symposium on Circuits and Systems (MWSCAS '08)*, pp. 650–653, August 2008.

[7] A. Hastings, *The Art of Analog Layout*, Pearson Prentice Hall, Upper Saddle River, NJ, USA, 2nd edition, 2006.

[8] National Instruments Technical staff, "Tutorial on strain gauge configuration types," October 2006.

[9] L. Najafizadeh, T. Vo, S. D. Phillips, et al., "The effects of proton irradiation on the performance of high-voltage n-MOSFETs implemented in a low-voltage SiGe BiCMOS platform," *IEEE Transactions on Nuclear Science*, vol. 55, no. 6, pp. 3253–3258, 2008.

[10] R. Gregorian and G. C. Temes, *Analog MOS Integrated Circuits for Signal Processing*, John Wiley & Sons, New York, NY, USA, 1986.

[11] C. C. Enz and G. C. Temes, "Circuit techniques for reducing the effects of op-amp imperfections: autozeroing, correlated double sampling, and chopper stabilization," *Proceedings of the IEEE*, vol. 84, no. 11, pp. 1584–1614, 1996.

[12] J. Williams, *Applications for a Switched-Capacitor Instrumentation Building Block*, Signal Conditioning Application Notes AN3, Linear Technology, Irvine, Calif, USA, 1985.

[13] C. Ulaganathan, *Design and analysis of a general purpose operational amplifier for extreme temperature operation*, M.S. thesis, The University of Tennessee, Knoxville, Tenn, USA, 2007.

[14] J. N. Babanezhad, "A rail-to-rail CMOS op amp," *IEEE Journal of Solid-State Circuits*, vol. 23, no. 6, pp. 1414–1417, 1988.

[15] L. Najafizadeh, A. K. Sutton, B. Jun, et al., "Radiation response of SiGe BiCMOS mixed-signal circuits intended for emerging lunar applications," in *Proceedings of the 9th European Conference on Radiation and Its Effects on Components and Systems (RADECS '07)*, pp. 78–82, Deauville, France, September 2007.

[16] M. Maymandi-Nejad and M. Sachdev, "A digitally programmable delay element: design and analysis," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 11, no. 5, pp. 871–878, 2003.

[17] O. B. Milgrome, S. A. Kleinfelder, and M. E. Levi, "A 12 bit analog to digital converter for VLSI applications in nuclear science," *IEEE Transactions on Nuclear Science*, vol. 39, no. 4, pp. 771–775, 1992.

[18] National Semiconductor Technical Staff, *LVDS Owner's Manual, Including High-Speed CML and Signal Conditioning*, National Semiconductor, 4th edition, 2008.

[19] C. C. Enz, F. Krummenacher, and E. A. Vittoz, "An analytical MOS transistor model valid in all regions of operation and dedicated to low-voltage and low-current applications," *Analog Integrated Circuits and Signal Processing*, vol. 8, no. 1, pp. 83–114, 1995.

[20] S. Chen, S. Terry, C. Ulaganathan, B. J. Blalock, and M. Mojarradi, "A SiGe current refernce for low temperature analog/mixed-signal applications," in *Proceedings of the 7th International Workshop on Low Temperature Electronics (WOLTE '06)*, pp. 276–280, Noordwijk, The Netherlands, June 2006.

[21] J. D. Cressler, "On the potential of SiGe HBTs for extreme environment electronics," *Proceedings of the IEEE*, vol. 93, no. 9, pp. 1559–1582, 2005.

[22] B. Jun, A. K. Sutton, R. M. Diestelhorst, et al., "The application of RHBD to n-MOSFETs intended for use in cryogenic-temperature radiation environments," *IEEE Transactions on Nuclear Science*, vol. 54, no. 6, pp. 2100–2105, 2007.

[23] T. Chen, L. C. Zhu, A. Diestelhorst, et al., "CMOS device reliablity for emerging cryogenic space electronics applications," in *Proceedings of the International Semiconductor Device Research Symposium (ISDRS '05)*, pp. 328–329, Washington, DC, USA, December 2005.

[24] J. E. Knudsen and L. T. Clark, "An area and power efficient radiation hardened by design flip-flop," *IEEE Transactions on Nuclear Science*, vol. 53, no. 6, pp. 3392–3399, 2006.

[25] T. Câlin, M. Nicolaidis, and R. Velazco, "Upset hardened memory design for submicron CMOS technology," *IEEE Transactions on Nuclear Science*, vol. 43, no. 6, pp. 2874–2878, 1996.

[26] J. R. Black, "Electromigration—a brief survey and some recent results," *IEEE Transactions on Electron Devices*, vol. 16, no. 4, pp. 338–347, 1969.

[27] National Instruments Technical staff, "Digital pattern I/O and handshaking," *NI 6534*, December 2007.

Hindawi Publishing Corporation
VLSI Design
Volume 2010, Article ID 864165, 8 pages
doi:10.1155/2010/864165

Research Article

Emerging Carbon Nanotube Electronic Circuits, Modeling, and Performance

Yao Xu,[1] Ashok Srivastava,[1] and Ashwani K. Sharma[2]

[1] *Department of Electrical and Computer Engineering, Louisiana State University, Baton Rouge, LA 70803-5901, USA*
[2] *Electronics Foundations Group, AFRL/VSSE, 3550 Aberdeen Avenue SE, Kirtland AFB, NM 87117, USA*

Correspondence should be addressed to Ashok Srivastava, ashok@ece.lsu.edu

Received 1 June 2009; Accepted 19 November 2009

Academic Editor: Gregory D. Peterson

Current transport and dynamic models of carbon nanotube field-effect transistors are presented. A model of single-walled carbon nanotube as interconnect is also presented and extended in modeling of single-walled carbon nanotube bundles. These models are applied in studying the performances of circuits such as the complementary carbon nanotube inverter pair and carbon nanotube as interconnect. Cadence/Spectre simulations show that carbon nanotube field-effect transistor circuits can operate at upper GHz frequencies. Carbon nanotube interconnects give smaller delay than copper interconnects used in nanometer CMOS VLSI circuits.

1. Introduction

A good amount of work on modeling carbon nanotube field-effect transistors (CNT-FETs) has been reported [1–4]. However, these models are numerical-based and require a mathematical/software realization. Recently, Srivastava et al. [5, 6] have obtained an analytical solution of current transport model for the CNT-FET for analysis and design of CNT-FET-based integrated circuits. Based on their work, a dynamic model [7, 8] for CNT-FETs is obtained and Verilog-AMS language [9] is used to predict static and dynamic characteristics of CNT-FETs and integrated circuits. Verilog-AMS requires less computational steps and easy to experiment with the developing model equations. In our work, we have used Verilog-AMS to describe the CNT-FET static and dynamic models and simulated CNT-FET circuits using Cadence/Spectre.

Fetter [10, 11] and Maffucci et al. [12] have investigated electron transport along the CNT and proposed a two-dimensional fluid model. In this model [10–12], electron-electron correlation, which is significant in CNT, has not been considered. In a recent work, we have made modification in two-dimensional fluid model to include electron-electron repulsive interaction and built a semiclassical one-dimensional fluid model [13], which is relatively easy to solve and apply in CNT transmission line modeling. We have also proposed circuit models for single-walled carbon nanotubes (SWCNTs) bundles as interconnects based on the one-dimensional fluid model.

2. CNT-FET Model

2.1. Static Model. The structure of a CNT-FET shown in Figure 1 [2] is similar to the structure of a typical MOSFET, where an SWCNT forms the channel between two electrodes, which work as the source and drain of the transistor. The structure is built on top of an insulating layer and a substrate which works as the back gate. The top gate is metal over the thin gate oxide. Current transport equations in a CNT-FET are developed in [5, 6] which are described here as follows and include both drift current and diffusion currents:

$$\begin{aligned} I_{ds} &= I_{drift} + I_{diff} \\ &= \beta\Big[f_{drift}\Big\{\Psi_{cnt,s}(L), V_{gs}\Big\} - f_{drift}\Big\{\Psi_{cnt,s}(0), V_{gs}\Big\}\Big] \\ &\quad + \beta\Big[f_{diff}\Big\{\Psi_{cnt,s}(L), V_{gs}\Big\} - f_{diff}\Big\{\Psi_{cnt,s}(0), V_{gs}\Big\}\Big], \end{aligned} \tag{1}$$

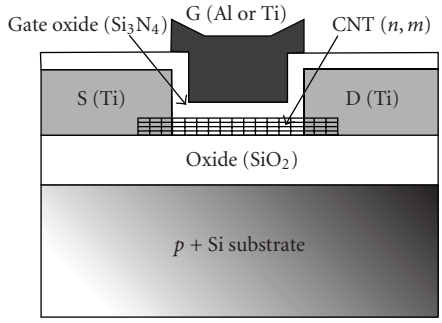


Figure 1: Plot of the vertical cross-section of a CNT-FET [2].

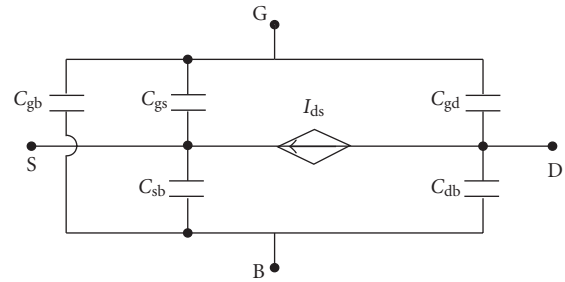


Figure 2: Meyer capacitance model for CNT-FETs.

where

$$f_{\text{drift}}\left(\psi_{\text{cnt},s}(x), V_{\text{gs}}\right) = \left(V_{\text{gs}} + V_{\text{sb}} - V_{\text{fb}}\right)\psi_{\text{cnt},s}(x) - \frac{1}{2}\psi^2_{\text{cnt},s}(x),$$

$$f_{\text{diff}}\left(\psi_{\text{cnt},s}(x), V_{\text{gs}}\right) = \frac{kT}{q}\psi_{\text{cnt},s}(x),$$

$$\beta = \gamma\frac{\mu C_{\text{ox1}}}{L^2}. \tag{2}$$

In (1), various parameters are defined as follows: we have L: gate length, μ: carrier mobility, k: Boltzmann constant, T: temperature,°K, V_{fb}: flat-band voltage, V_{gs}: gate-source voltage, V_{sb}: source-substrate voltage, $\psi_{\text{cnt},s}$: surface potential of CNT, and C_{ox1}: gate-oxide capacitance per unit area. For a carbon nanotube of length L and radius r in a CNT-FET, the oxide capacitance is given by [14]

$$C_{\text{ox1}} = \frac{2\pi\varepsilon_{\text{ox1}}L}{\ln\left(\left(T_{\text{ox1}} + r + \sqrt{T^2_{\text{ox1}} + 2T_{\text{ox1}}r}\right)/r\right)}. \tag{3}$$

In (3), T_{ox1} is the thickness of the gate oxide and r is the radius of the CNT. Equation (1) is modified to incorporate channel length modulation through the parameter λ as in a MOSFET. In saturation region, modified equation (1) is described as follows [6]:

$$I_{\text{ds}} = \beta\left[f\left\{\psi_{\text{cnt},s}(L), V_{\text{gs}}\right\} - f\left\{\psi_{\text{cnt},s}(0), V_{\text{gs}}\right\}\right](1 + \lambda V_{\text{ds}}). \tag{4}$$

2.2. Dynamic Model. The dynamic response of a CNT-FET can be modeled using Meyer capacitance model [7, 8, 15, 16] as shown in Figure 2. Recently, we have obtained capacitances: $C_{\text{gs}}, C_{\text{gd}}$, and C_{gb} based on current transport modeling of CNT-FETs described by (1), which are as follows [7, 8].

In linear region,

$$C_{\text{gb}} = 0,$$

$$C_{\text{gs}} = -\frac{\gamma\mu W|C_h|C'^2_{\text{ox1}}}{2\beta} \cdot \frac{\left[\delta Ie^{-1} - \delta\Delta\left(\mathfrak{C} - V_{\text{gs}}\right)\right]}{\left[\delta Ie^{-1} - \delta\Delta\left(\mathfrak{C} - (1/2)V_{\text{gd}} - (1/2)V_{\text{gs}}\right)\right]^2} \times \frac{\left[\delta Ie^{-1} - \delta\Delta\left(\mathfrak{C} - (2/3)V_{\text{gd}} - (1/3)V_{\text{gs}}\right)\right]}{\left[\delta Ie^{-1} - \delta\Delta\left(\mathfrak{C} - (1/2)V_{\text{gd}} - (1/2)V_{\text{gs}}\right)\right]^2},$$

$$C_{\text{gd}} = -\frac{\gamma\mu W|C_h|C'^2_{\text{ox1}}}{2\beta} \cdot \frac{\left[\delta Ie^{-1} - \delta\Delta\left(\mathfrak{A} - V_{\text{gs}}\right)\right]}{\left[\delta Ie^{-1} - \delta\Delta(\mathfrak{C} - (1/2)V_{\text{gd}} - (1/2)V_{\text{gs}})\right]^2} \times \frac{\left[\delta Ie^{-1} - \delta\Delta\left(\mathfrak{A} - (1/3)V_{\text{gd}} - (2/3)V_{\text{gs}}\right)\right]}{\left[\delta Ie^{-1} - \delta\Delta\left(\mathfrak{C} - (1/2)V_{\text{gd}} - (1/2)V_{\text{gs}}\right)\right]^2}, \tag{5}$$

where $\mathfrak{C}$ denotes $\phi_0 - (\Delta E_F/q) + (E_c/q) - (kT/q) + V_{\text{fb}}$, $\mathfrak{A}$ denotes $\phi_0 - (\Delta E_c/q) + (E_c/q) - (kT/q) + V_{\text{fb}}$. In saturation region,

$$C_{\text{gb}} = 0,$$

$$C_{\text{gs}} = \frac{1}{3}\frac{\gamma\mu W|C_h|C'^2_{\text{ox1}}}{\beta}, \tag{6}$$

$$C_{\text{gd}} = 0.$$

Considering C_{sb} and C_{db} to be equal to one half the insulator capacitance, $C_{\text{ox2}}/2$ in series with the depletion-layer

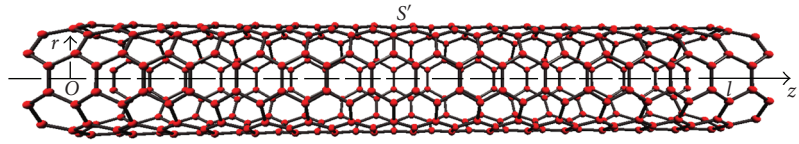


Figure 3: Geometry of a single-walled carbon nanotube (SWCNT).

capacitance, $C_{subs}/2$ [17, 18], we obtain

$$C_{ox2} = \frac{2\pi\varepsilon_{ox2}L}{\ln\left(\left(T_{ox1} + r + \sqrt{T_{ox2}^2 + 2T_{ox2}r}\right)/r\right)},$$

$$C_{subs} = \frac{N_A q\varepsilon_s}{4C_{ox2}\left(V_{gb} - \phi_{ms} - \psi_{cnt}\right)} + \frac{\sqrt{N_A q\varepsilon_s}\sqrt{8C_{ox2}^2 V_{gb} + N_A q\varepsilon_s - 8C_{ox2}^2\phi_{ms} - 8C_{ox2}^2\psi_{cnt}}}{4C_{ox2}\left(V_{gb} - \phi_{ms} - \psi_{cnt}\right)}, \quad (7)$$

where ε_s is the permittivity of the semiconductor, W_s is the depletion region width, and N_A is the doping concentration.

3. SWCNT Interconnect Model

Figure 3 shows the geometry of an SWCNT interconnect. Based on two-dimensional fluid model developed by Maffucci et al. [12] we have proposed one-dimensional fluid model to describe the electron transport in metallic CNT and built a transmission line model of metallic CNT interconnects [13]. When compared with a two-dimensional fluid model, one-dimensional fluid model is accurate and takes into account electron-electron interaction. The Lüttinger Liquid Theory [19] models SWCNT as a one-dimensional conductor from quantum concept and takes into account electron-electron correlation. However, our model is simple in mathematical modeling and easier to extend in modeling of CNT bundles as interconnections. The basic equation is Euler's equation, which is Newton's Second Law applied in fluid dynamics and is given by [13]

$$mn\left(\frac{\partial}{\partial t} + v_z\frac{\partial}{\partial z}\right)v_z = -\frac{\partial p}{\partial z} - en\left\{(1-\alpha)\ \boldsymbol{\mathcal{E}}_z|_{s'}\right\} - mnvv_z, \quad (8)$$

where n_0 is the equilibrium electron density, v_z is the electron mean velocity, p is the pressure, m is the electron mass, e is the electronic charge, and $\boldsymbol{\mathcal{E}}_z$ is electric field. The second term on the right-hand side represents Lorentz force and includes electron-electron interaction through the parameter α. The last term on the right-hand side represents the effect of scattering of electrons with the positive charge background and v is the electron relaxation frequency, $v = v_F/l_{mfp}$, where l_{mfp} is the mean-free path of electron in CNT and v_F is the Fermi velocity. Length of CNT is l and $\mathrm{sgn}(l)$ is the sign function defined as follows:

$$\mathrm{sgn}(l) = \begin{cases} 0 & \text{if } l < l_{mfp}, \\ 1 & \text{if } l \geq l_{mfp}. \end{cases} \quad (9)$$

The parameter α describes the classical electron-electron repulsive interaction given by [13]

$$\alpha \equiv \frac{\boldsymbol{\mathcal{E}}_{zP}}{\boldsymbol{\mathcal{E}}_z} = \frac{E_P}{E} = \frac{E_P}{E_K + E_P}, \quad (10)$$

where $\mathcal{E}_{zP}$ is the part of the electrical field which provides potential energy to electrons in z direction. E is the total energy of electrons. E_P and E_K are the potential and kinetic energies of electrons, respectively.

The equation relating current density, charge density, and electric field can be described as follows [13]:

$$\frac{\partial j(z,t)}{\partial t} + \nu j + u_e^2\frac{\partial\sigma(z,t)}{\partial z} = \frac{e^2 n_0}{m}(1+\alpha)\boldsymbol{\mathcal{E}}_z. \quad (11)$$

We consider a metallic single-walled CNT above a perfect conducting plane and assume [12] that the propagating EM wave is in quasi-TEM mode [12]. The voltage and current intensity are then expressed as follows:

$$i(z,t) = \oint \vec{j}\cdot\vec{z}\ dl \approx 2\pi r j(z,t),$$
$$q(z,t) = \oint \vec{\sigma}\cdot dl \approx 2\pi r\sigma(z,t). \quad (12)$$

Combining (12) and (11), following equation is obtained as

$$\boldsymbol{\mathcal{E}}_z = Ri + L_K\frac{\partial i}{\partial t} + \frac{1}{C_Q}\frac{\partial q}{\partial z}, \quad (13)$$

where $R \equiv L_K\,\mathrm{sgn}(l)\nu$ is the resistance per unit length of CNT, $L_K \equiv m/[(1+\alpha)2\pi r e^2 n_0]$ is the kinetic inductance per unit length, $C_Q \equiv 1/L_K u_e^2$ is quantum capacitance per unit length, and $u_e = v_F/\sqrt{1-\alpha}$ is the thermodynamic speed of sound of the electron fluid under neutral environment.

The magnetic inductance and electric capacitance per unit length of a perfect conductor on a ground plane are given by [20]

$$L_M = \frac{\mu}{2\pi}\cosh^{-1}\left(\frac{h}{r}\right) \approx \frac{\mu}{2\pi}\ln\left(\frac{h}{2r}\right),$$
$$C_E = \frac{2\pi\varepsilon}{\cosh^{-1}(h/r)} \approx \frac{2\pi\varepsilon}{\ln(h/r)}, \quad (14)$$

where h is the distance of CNT to the ground plane. Equation (14) is accurate enough for $h > 2r$.

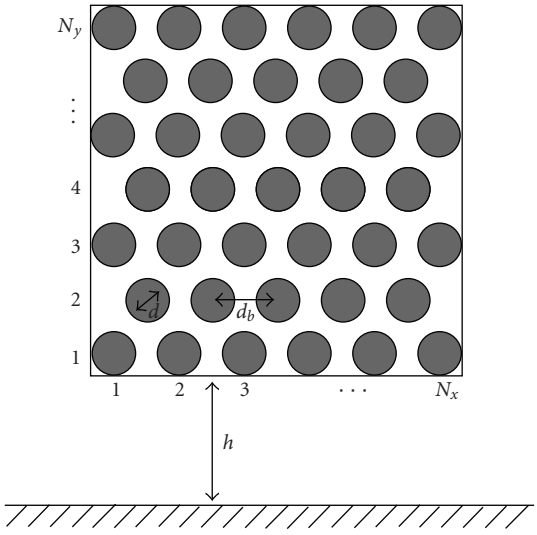


Figure 4: Cross-section of SWCNT bundle interconnect wire.

4. SWCNT Bundle Interconnect Model

Carbon nanotube can also be fabricated in bundles. The spacing between nanotubes in the bundle is due to the van der Waals forces between the atoms in adjacent nanotubes [21]. One of the most critical challenges in realizing high-performance SWCNT interconnect is in controlling the proportion of metallic nanotubes in a bundle. Current SWCNT fabrication techniques cannot effectively control the chirality of the nanotubes in a bundle [22, 23]. Therefore, SWCNT bundles have metallic nanotubes that are randomly distributed within the bundle. Avouris et al. [22] and Liebau et al. [23] have shown that metallic nanotubes are distributed with a probability $\beta = 1/3$ in a growth process. The proportion of metallic nanotubes can, however, be potentially increased using techniques introduced in [24, 25].

Figure 4 shows the cross-section of an SWCNT bundle. Since the van der Waals force between the carbon atoms in adjacent SWCNTs is negligible compared to covalent bond between carbon atoms in an SWCNT [26], the one-dimensional fluid model can be applied to each SWCNT in the bundle with modification.

Considering one of the SWCNTs, assuming electrons in SWCNT will be only affected by the electrons in the adjacent metallic SWCNTs and semiconducting SWCNTs have no effect on the conductance of the bundle. To calculate the potential energy, we first consider the potential energy of each SWCNT and then move them to be adjacent to each other to compose for the SWCNT bundle. Average potential energy of electrons in a single SWCNT can be described by the following equation:

$$E_P = 6\frac{e^2}{2\pi\varepsilon_0}\frac{1}{d} + \beta\sum_{i=1}^{\Gamma}\frac{16e^2}{2\pi\varepsilon_0}\frac{1}{d_b}, \tag{15}$$

where d is diameter of CNT and $d_b = \delta + d$ is the distance of the adjacent SWCNT shown in Figure 4. δ is the spacing between the SWCNTs in the bundle corresponding to the van der Waals distance between graphene layers in graphite. Γ is the average number of SWCNTs neighboring an SWCNT. The number of SWCNTs neighboring the corner SWCNT is 2, the number of SWCNTs neighboring the edge SWCNT is 4, and the number of SWCNTs neighboring the inside SWCNT is 6. Therefore, $\Gamma = [(6N_xN_y - 4N_x - 4N_y - 2[N_y/2])/(N_xN_y - 2[N_y/2])]$, where square brackets are the floor functions.

The kinetic energy of the electrons is described by

$$E_K = 4 \times \frac{1}{2}mv_F^2 \approx 7\,\text{eV}. \tag{16}$$

Therefore, electron-electron interaction parameter α for SWCNT bundle can be calculated using (10). Total number of metallic SWCNTs in a bundle can be described by $N = \beta(N_xN_y - [N_y/2])$. Following the derivation of electric field and current charge relation in [13], we get the similar equation for the electric field as described below:

$$\mathcal{E}_z = Ri + L_K\frac{\partial i}{\partial t} + \frac{1}{C_Q}\frac{\partial q}{\partial z}, \tag{17}$$

where $R \equiv L_K\,\text{sgn}(l)\nu$ is the resistance per unit length of an SWCNT in an SWCNT bundle.

In (17), $L_K \equiv \pi\hbar/4e^2\nu_F$ is the kinetic inductance per unit length of an SWCNT in a bundle and $C_Q \equiv 1/L_Ku_e^2$ is the quantum capacitance per unit length of an SWCNT in a bundle. $u_e = \nu_F/\sqrt{1-\alpha}$ is the thermodynamic speed of sound of the electron fluid under a neutral environment.

The SWCNTs at the bottom shield the upper SWCNTs from the ground plane. Therefore, the electric capacitance C_E does not exist in the upper SWCNTs. However, there exists electric capacitance per unit length C_b between the neighboring metallic SWCNTs and its value is given by [20]

$$C_b = \frac{\pi\varepsilon_0}{\ln\left((d_b/d) + \sqrt{(d_b/d)^2 - 1}\right)}. \tag{18}$$

Figure 5 shows the equivalent circuit of an SWCNT bundle as an interconnect wire. $N_b = N_x$ is the number of lowest level metallic SWCNTs, which shield upper levels SWCNTs from ground plane. $N_a = N - N_b$ is the number of upper levels metallic SWCNTs.

5. Result and Discussion

In a recent work [6], we have developed analytical CNT-FET models for I-V characteristics and verified them with the experimentally measured I-V characteristics. Table 1 summarizes some of the physical and electrical parameters of CNT-FETs and comparison with equivalent MOSFET parameters. CNT-FETs are described in [2, 6] and MOSFETs in [27, 28], respectively. It is noticed from Table 1 that the CNT-FET carries a higher current density compared with the equivalent bulk silicon and SOI MOSFETs. In the following, we have used our CNT-FET models [5–8] in studying the performance of a ring oscillator circuit and compared it with the measured performance.

Figure 6(a) shows schematic of a five-stage ring oscillator circuit similar to one fabricated by Chen et al. [29].

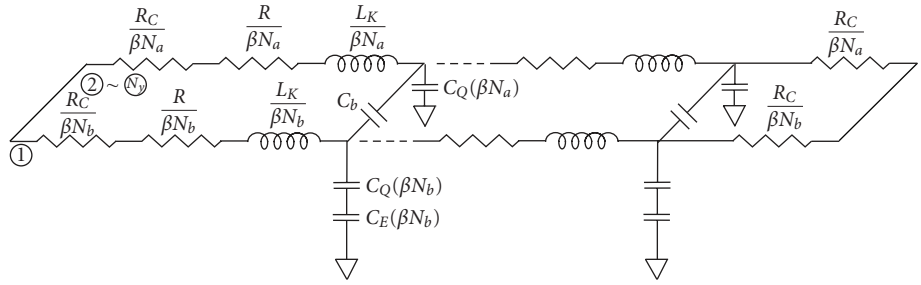


Figure 5: Equivalent circuit of SWCNT bundle interconnect.

Table 1: A comparison of modeled and measured parameters of CNT-FETs and MOSFETs.

FET Parameters	p-type CNT-FET* Model [6]	p-type CNT-FET* Measured [2]	Bulk-Si p-MOSFET Measured [27]	SOI p-MOSFET Measured [28]
Gate Length (nm)	260	260	15	50
Gate Oxide Thickness (nm)	15	15	1.4	1.5
Threshold Voltage (V)	−0.3	−0.5	~ −0.1	~ −0.2
Subthreshold Swing (mV/dec)	110	130	100	70
On-Current per Unit Width (μA/μm)	2000	2100	265	650

*Chiral Vector: (11, 9).

Figure 6(b) shows the simulation result of the ring oscillator output waveform at 0.92 V supply voltage. Figure 6(c) shows the oscillation frequency with varying supply voltage. The modeled curve does not include the effect of channel length modulation. The experimental data are taken from the work of Chen et al. [29]. Modeled and experimental curves show that the frequency of the ring oscillator is about 70–80 MHz at 1.04 V supply voltage. The frequency is low because CNT-FETs in this ring oscillator are 600 nm long and there are parasitic capacitances associated with the metal wire in the ring oscillator. Shorter length of CNT-FETs will increase the oscillating frequency as shown in Figure 7.

SWCNT exhibits large contact resistance when used as an interconnect wire [30]. However, CNT bundle gives low contact resistance when used as the circuit interconnect wire [31, 32]. Contact resistance in a bundle, however, will depend on the number of SWCNTs being metallic. Utilizing the models of CNT interconnects, we have also studied the performance of CNT-FET circuit inverter pair with different kinds of interconnects including the copper (Cu). One of the advantages of CNT interconnect is its large mean-free path l_{mfp}, which is on the order of several micrometers (as compared to 40 nm for Cu at room temperature). It provides low resistivity and ballistic transport in short-length interconnects [33]. In this work, first we have simulated a CNT-FET inverter pair with 0.1 μm Cu and SWCNT bundle as interconnect wires using Cadence/Spectre. We have utilized the process parameters from the 2016 node, 22 nm technology [34] and assumed a 22 nm width and 44 nm height of the SWCNT bundle. The spacing between nanotubes in the bundle is due to van der Waals forces between the atoms in adjacent nanotubes, which means that the spacing between adjacent SWCNTs is 0.34 nm. If we assume diameter of an SWCNT in a bundle is to be 1 nm, then there are nearly 500 SWCNTs in the 22 nm (width) × 44 nm (height) bundle following Figure 4.

Figure 8 shows input and output waveforms of a CNT-FET inverter pair using SWCNT bundle as an interconnect wire and comparison with the ideal interconnect (RC = 0) and Cu interconnect. Input signal is a 4 GHz square wave pulse. The average delay is 6 ps, which suggests that the CNT-FET inverter pair can respond up to 100 GHz input signal. The performance of SWCNT bundle as an interconnect wire ($\beta = 1$) is close to Cu interconnect. It is due to contact resistance and relatively larger capacitance in an SWCNT bundle. The average delays are smaller for $\beta = 1$ than those for $\beta = 1/3$ for SWCNT bundle interconnect. It can be explained that there are more metallic SWCNTs in a bundle when β increases.

Local interconnects are often used for connecting nearby gates or devices with lengths of the order of micrometers. Therefore, these have the smallest cross-section and largest resistance per unit length compared to global interconnects.

Figure 9 shows output waveforms of a CNT-FET inverter pair using 10 μm SWCNT bundle interconnect and the comparison with the ideal interconnect (RC = 0) and Cu interconnect wires. Input signal is a 15 MHz square wave pulse. The performance of SWCNT bundle ($\beta = 1$) interconnect is better than Cu interconnect. While the delay of SWCNT bundle ($\beta = 1/3$) interconnect is larger than the Cu interconnect due to contact resistance and relatively larger capacitance in an SWCNT bundle, the average delays are smaller when $\beta = 1$ than $\beta = 1/3$ for SWCNT bundle interconnect wires. It can also be explained that there are more metallic SWCNTs in a bundle when β increases.

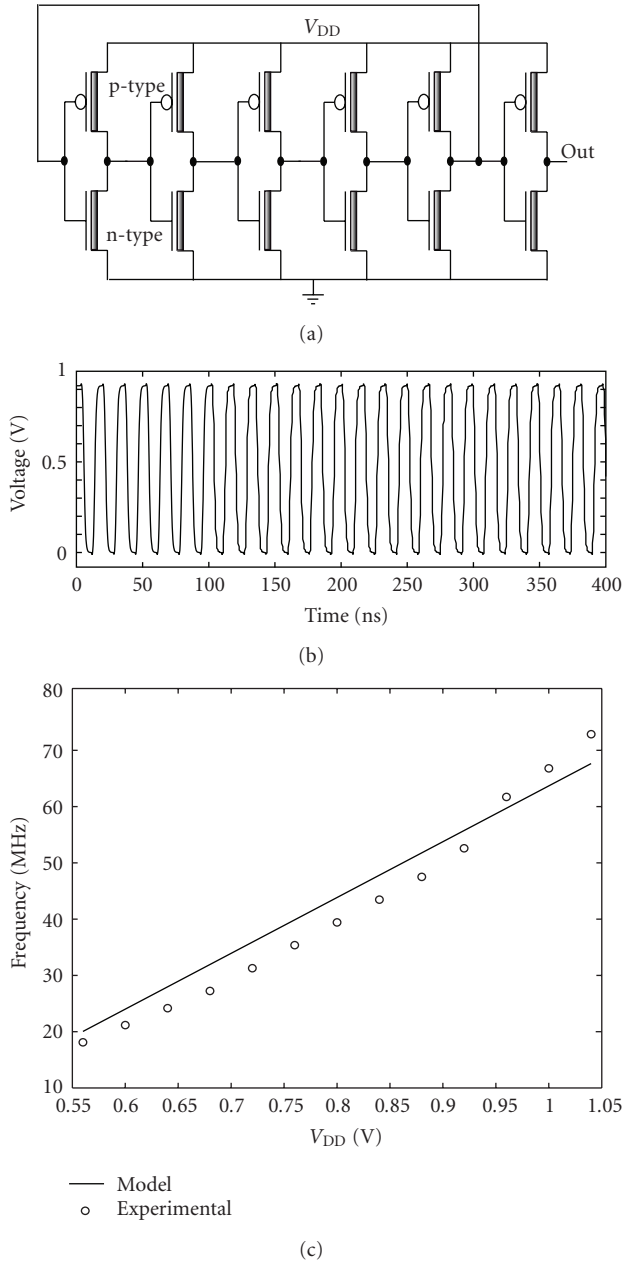


Figure 6: (a) Schematic of a 5-stage ring oscillator, (b) output waveform of ring oscillator, and (c) oscillating frequency versus supply voltage V_{DD}. Dimensions of both the n- and p-types CNT-FETs are $d = 2$ nm and $L = 600$ nm.

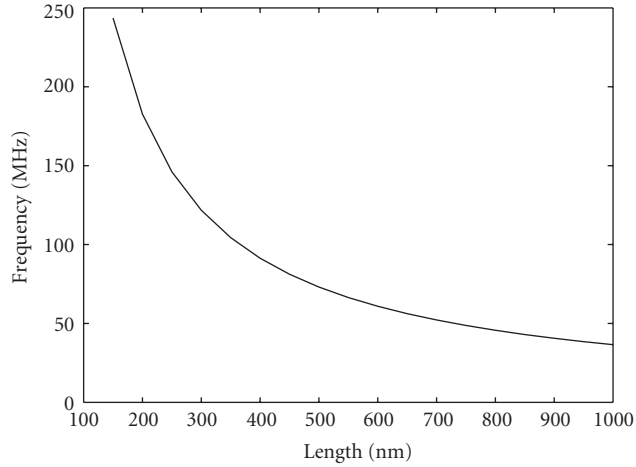


Figure 7: Oscillating frequency of a 5-stage ring oscillator versus length of the CNT-FETs.

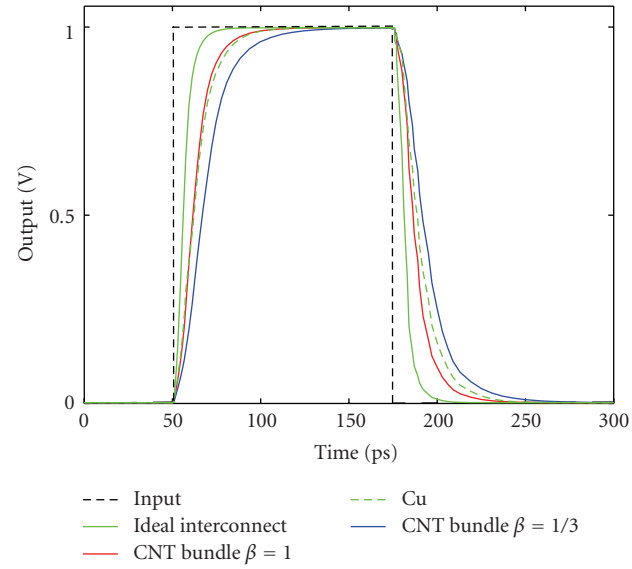


Figure 8: Input and output waveforms of a CNT-FET inverter pair with 0.1 μm length of different interconnect wires.

Global interconnects have larger cross-section and smaller resistivity. The lengths are of the order of hundred micrometers. Figure 10 shows output waveform of a CNT-FET inverter pair with 500 μm SWCNT bundle interconnect and comparison with the ideal interconnect (RC = 0) and Cu interconnect wires. Here we have utilized the process parameters from the 2016 node, 22 nm technology and assumed 33 nm width and 87 nm height of an SWCNT bundle [34]. It can be shown that there are nearly 1500 SWCNTs in the 33 nm (width) $\times$ 87 nm (height) bundle following Figure 4 assuming 1 nm diameter of an SWCNT. Input signal is a 2 MHz square wave pulse. The performance of SWCNT bundle ($\beta = 1$) interconnect is much better than the Cu interconnect. While the delay of SWCNT bundle ($\beta = 1/3$) interconnect is larger than the Cu interconnect, the average delays are smaller when $\beta = 1$ than $\beta = 1/3$ for SWCNT bundle interconnects. This explains further that there are more metallic SWCNTs in a bundle when β increases.

Our results show that the CNT-FET circuits can potentially operate up to 100 GHz. SWCNT bundle interconnect ($\beta = 1$) has better performance than the Cu interconnect contrary to bundle with $\beta = 1/3$. This result also compares well with the work of Nieuwoudt and Massoud [35] showing that the SWCNT bundle interconnects have larger delay than Cu interconnects for $\beta = 1/3$. The proportion of

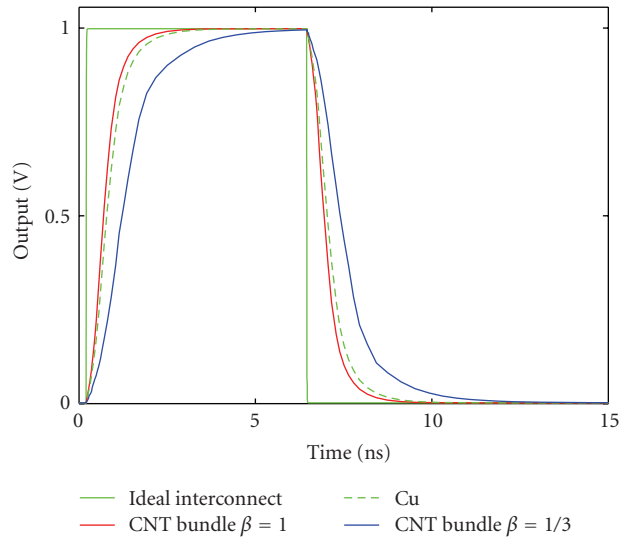


Figure 9: Output waveforms of a CNT-FET inverter pair with 10 μm length of different interconnect wires.

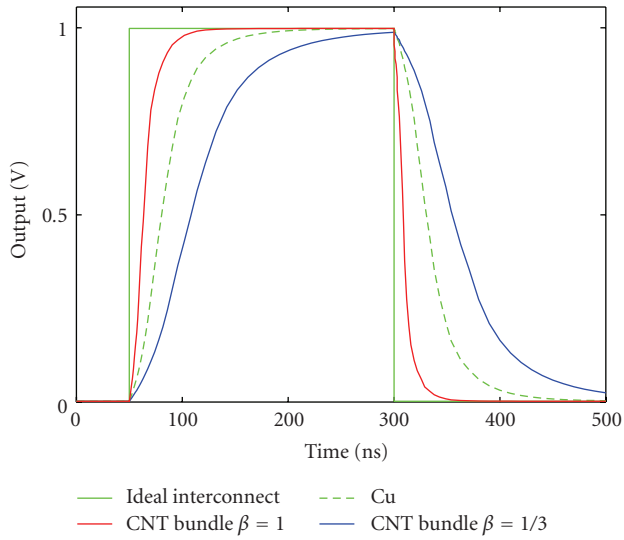


Figure 10: Output waveforms of a CNT-FET inverter pair with 500 μm length of different interconnect wires.

metallic nanotubes can be potentially increased using techniques introduced in [24, 25]; the delay of SWCNT bundle interconnect can be smaller than that of Cu interconnect when β approaches 1. The SWCNT bundle interconnect can potentially replace Cu interconnect in future CNT-FET circuits.

6. Conclusion

In this paper, static and dynamic models of CNT-FETs are introduced and models for SWCNT bundle interconnects are presented based on one-dimensional fluid model of SWCNTs. These models have been used to study the behavior of CNT-FET circuits such as ring oscillator and inverter pair and compared with the corresponding experimental behavior. The applicability of SWCNT bundle as interconnect wires in future design of integrated circuits has been explored theoretically and compared with the Cu interconnect wires for 22 nm technology node. Simulation results suggest that SWCNT bundle interconnect ($\beta = 1$) can replace Cu interconnects as the technology scales down.

Acknowledgments

Authors acknowledge the support provided by the Louisiana Economic Development Assistantship (EDA) program to carry out the proposed research. Part of the work is also supported by NSF (2009)-PFUND-138 and United States Air Force Contract no. FA9401-08-P-0129. Part of this material is also based on research sponsored by Air Force Research Laboratory under agreement number FA9453-10-1-0002. The U.S. Government is authorized to reproduce and distribute reprints for Government purposes notwithstanding any copyright notation thereon.

References

[1] F. Nihey, H. Kongo, Y. Ochiai, M. Yudasaka, and S. Iijima, "Carbon-nanotube field-effect transistors with very high intrinsic transconductance," *Japanese Journal of Applied Physics*, vol. 42, no. 10B, pp. L1288–L1291, 2003.

[2] S. J. Wind, J. Appenzeller, R. Martel, V. Derycke, and Ph. Avouris, "Vertical scaling of carbon nanotube field-effect transistors using top gate electrodes," *Applied Physics Letters*, vol. 80, no. 20, pp. 3817–3819, 2002.

[3] J. W. Park, J. B. Choi, and K.-H. Yoo, "Formation of a quantum dot in a single-walled carbon nanotube using the Al top-gates," *Applied Physics Letters*, vol. 81, no. 14, pp. 2644–2646, 2002.

[4] F. Nihey, H. Hongo, M. Yudasaka, and S. Iijima, "A top-gate carbon-nanotube field-effect transistor with a titanium-dioxide insulator," *Japanese Journal of Applied Physics*, vol. 41, no. 10A, pp. L1049–L1051, 2002.

[5] J. M. Marulanda, A. Srivastava, and A. K. Sharma, "Current transport modeling in carbon nanotube field effect transistors (CNT-FETs) and bio-sensing applications," in *Nanosensors and Microsensors for Bio-Systems*, vol. 6931 of *Proceedings of SPIE*, San Diego, Calif, USA, March 2008.

[6] A. Srivastava, J. M. Marulanda, Y. Xu, and A. K. Sharma, "Current transport modeling of carbon nanotube field effect transistors," *Physica Status Solidi A*, vol. 206, no. 7, pp. 1569–1578, 2009.

[7] Y. Xu and A. Srivastava, "Transient behavior of integrated carbon nanotube field effect transistor circuits and bio-sensing applications," in *Nano-, Bio, and Info-Tech Sensors and Systems*, vol. 7291 of *Proceedings of SPIE*, San Diego, Calif, USA, March 2009.

[8] Y. Xu and A. Srivastava, "Dynamic response of carbon nanotube field effect transistor circuits," in *Proceedings of the NSTI Nanotechnology Conference and Expo*, vol. 1, pp. 625–628, Houston, Tex, USA, May 2009.

[9] "Verilog-AMS Language Reference Manual," August 2008, http://www.designers-guide.org/VerilogAMS/.

[10] A. L. Fetter, "Electrodynamics of a layered electron gas. I. Single layer," *Annals of Physics*, vol. 81, no. 2, pp. 367–393, 1973.

[11] A. L. Fetter, "Electrodynamics of a layered electron gas. II. Periodic array," *Annals of Physics*, vol. 88, no. 1, pp. 1–25, 1974.

[12] A. Maffucci, G. Miano, and F. Villone, "A transmission line model for metallic carbon nanotube interconnects," *International Journal of Circuit Theory and Applications*, vol. 36, no. 1, pp. 31–51, 2008.

[13] Y. Xu and A. Srivastava, "A model for carbon nanotube interconnects," *International Journal of Circuit Theory and Applications*, pp. 1–17, 2009, Published online in Wiley InterScience (http://www3.interscience.wiley.com/).

[14] D. T. Thomas, *Engineering Electromagnetics*, Pergamon Press, New York, NY, USA, 1972.

[15] T. A. Fjeldly, T. Ytterdal, and M. S. Shur, *Introduction to Device Modeling and Circuit Simulation*, Wiley, New York, NY, USA, 1998.

[16] Y. Cheng and C. Hu, *MOSFET Modeling and BSIM3 User's Guide*, Springer, New York, NY, USA, 1999.

[17] B. Streetman and S. Banerjee, *Solid State Electronic Devices*, Prentice Hall, Upper Saddle River, NJ, USA, 5th edition, 2000.

[18] S. J. Wind, J. Appenzeller, R. Martel, V. Derycke, and Ph. Avouris, "Fabrication and electrical characterization of top gate single-wall carbon nanotube field-effect transistors," *Journal of Vacuum Science and Technology B*, vol. 20, no. 6, pp. 2798–2801, 2002.

[19] M. P. A. Fisher and L. I. Glazman, "Transport in a one-dimensional Lüttinger liquid," in *Mesoscopic Electron Transport*, L. Kouwenhoven, G. Schoen, and L. Sohn, Eds., vol. 345 of *NATO ASI Series E*, pp. 331–373, Kluwer Academic Publishers, Dordrecht, The Netherlands, 1996.

[20] S. Ramo, J. R. Whinnery, and T. V. Duzer, *Fields and Waves in Communication Electronics*, Wiley, New York, NY, USA, 1994.

[21] A. Thess, R. Lee, P. Nikolaev, et al., "Crystalline ropes of metallic carbon nanotubes," *Science*, vol. 273, no. 5274, pp. 483–487, 1996.

[22] Ph. Avouris, J. Appenzeller, R. Martel, and S. J. Wind, "Carbon nanotube electronics," *Proceedings of the IEEE*, vol. 91, no. 11, pp. 1772–1783, 2003.

[23] M. Liebau, A. P. Graham, G. S. Duesberge, E. Unger, R. Seidel, and F. Kreupl, "Nanoelectronics based on carbon nanotubes: technological challenges and recent developments," *Fullerenes Nanotubes and Carbon Nanostructures*, vol. 13, no. S1, pp. 255–258, 2005.

[24] N. Peng, Q. Zhang, J. Li, and N. Liu, "Influences of ac electric field on the spatial distribution of carbon nanotubes formed between electrodes," *Journal of Applied Physics*, vol. 100, no. 2, Article ID 024309, 5 pages, 2006.

[25] M. Zheng, A. Jagota, M. S. Strano, et al., "Structure-based carbon nanotube sorting by sequence-dependent DNA assembly," *Science*, vol. 302, no. 5650, pp. 1545–1548, 2003.

[26] D. M. Rosenthal and R. M. Asimow, *Introduction to Properties of Materials*, Van Nostrand Reinhold, New York, NY, USA, 1971.

[27] Y. Bin, W. Haihong, A. Joshi, X. Qi, I. Effiong, and L. Ming-Ren, "15 nm gate length planar CMOS transistor," in *Proceedings of IEEE International Electron Devices Meeting (IEDM '01)*, pp. 11.7.1–11.7.3, Washington, DC, USA, 2001.

[28] R. Chau, J. Kavalieros, B. Doyle, et al., "A 50 nm depleted-substrate CMOS transistor (DST)," in *Proceedings of IEEE International Electron Devices Meeting (IEDM '01)*, pp. 29.1.1–29.1.4, Washington, DC, USA, 2001.

[29] Z. Chen, J. Appenzeller, P. M. Solomon, Y.-M. Lin, and Ph. Avouris, "High performance carbon nanotube ring oscillator," in *Proceedings of the 64th Device Research Conference*, pp. 171–172, State College, Pa, USA, June 2006.

[30] P. J. Burke, "Lüttinger liquid theory as a model of the gigahertz electrical properties of carbon nanotubes," *IEEE Transactions on Nanotechnology*, vol. 1, no. 3, pp. 129–144, 2002.

[31] Y. Massoud and A. Nieuwoudt, "Modeling and design challenges and solutions for carbon nanotube-based interconnect in future high performance integrated circuits," *Journal on Emerging Technologies in Computing Systems*, vol. 2, no. 3, pp. 155–196, 2006.

[32] A. Nieuwoudt and Y. Massoud, "On the optimal design, performance, and reliability of future carbon nanotube-based interconnect solutions," *IEEE Transactions on Electron Devices*, vol. 55, no. 8, pp. 2097–2110, 2008.

[33] P. L. McEuen, M. S. Fuhrer, and H. Park, "Single-walled carbon nanotube electronics," *IEEE Transactions on Nanotechnology*, vol. 1, no. 1, pp. 78–85, 2002.

[34] "International Technology Roadmap for Semiconductors," 2007, http://www.itrs.net/Links/2007ITRS/Home2007.htm.

[35] A. Nieuwoudt and Y. Massoud, "On the impact of process variations for carbon nanotube bundles for VLSI interconnect," *IEEE Transactions on Electron Devices*, vol. 54, no. 3, pp. 446–455, 2007.