Cell Based Synthesized Low Noise All Digital Frequency Synthesizer, 0.13μm CMOS and FPGA Implementations

by

Tingjun Wen, B.Sc., M.Sc.

A thesis submitted to the Faculty of Graduate and Postdoctoral Affairs in partial fulfilment of the requirements for the degree of

Master's of Applied Science

in

Electrical Engineering

Ottawa-Carleton Institute for Electrical and Computer Engineering

Department of Electronics

Faculty of Engineering

Carleton University

Ottawa, Ontario

Canada

September 2011

Your file Votre référence
ISBN: 978-0-494-83122-9
Our file Notre référence
ISBN: 978-0-494-83122-9

The information used in this thesis comes in part from the research program of Dr. Tad Kwasniewski and his associates in the VLSI in Communication Group. The research results appearing in this thesis represent an integral part of the ongoing research program. All research results in this thesis, including tables, graphs, and figures, but excluding the narrative portion of the thesis are effectively incorporated into the research program and can be used by Dr. Kwasniewski and his associates for educational and research purpose, including publication in open literature with appropriate credits. The intellectual property may be pursued cooperatively with *Carleton University* and Dr. Kwasniewski when appropriate.

Abstract

This thesis proposed an all digital phase locked loop (ADPLL) that consists of a Bang-Bang phase frequency detector (BBPFD) without any cycle-slip and with a narrow dead-zone, a digitally controlled oscillator (DCO) with a built-in low pass filter (LPF), and a multi-modulus divider (MMD) with an extended divide range. The loop dynamics and phase noise is simulated by the behavioral simulation in *SystemVerilog*. The proposed ADPLL is fully synthesized in the field programmable gate array (FPGA) as a fast prototype and is also implemented in $0.13\mu m$ complementary metal-oxide semiconductor (CMOS) technology by the standard digital cells and two custom high speed cells. The measured open-loop and closed-loop phase noise of the fabricated CMOS ADPLL are $-123.10dBc/Hz$ and $-120.77dBc/Hz$ both at $1MHz$ offset of a carrier of $1.35GHz$.

Acknowledgments

I am wholeheartedly thankful to my supervisor, Professor Kwasniewski, for his continuous support throughout the course of my thesis project. Without his knowledgeable guidance and advice, I would not have been able to turn the complex ADPLL theories and the cuting-edge CMOS circuit design techniques into a real working chip.

Professor Kwasniewski set a high standard in my research project and gave me the opportunity to explore the problems and solutions independently. I enjoyed and appreciate the time he spent with me in *Tim Horton's* on technical subjects that made me scratch my head.

Special thanks also goes to Professor Calvin Plett who offered me helps on the circuit and pad design in the hallway. He taught me the wonderful Radio Frequency (RF) circuit course from which my ADPLL circuits benefit greatly.

It was a wonderful working experience in Professor Kwasniewski's research group with other team members. Dr. David Chen had many fruitful discussions with me on the PLL operating principles. Miao Li and Dr. Jingcheng Zhuang's past experiences with the design kits and methodologies passed on me are great treasure that saves me enormous design time.

Feng Liu and Hsu Ho from the *CMC Microsystems* were very helpful during chip fabrication and packaging stages. Without their prompt Email replies to clear up the design rule violations with their unmatched level of circuit design and layout experiences, the submission deadline would have had been missed.

Zhanjun Bai gave me help on the layout issues with the design kit during the layout stage. He also offered me his test gadgets used in his past chip measurement,

which sped up the control software development before the hardware test began.

The initial chip test environment was set up by Nagui Mikhail professionally in the Electronics Department. He spent a weekend helping me chasing down and eliminating an external switching power supply noise which was passed in to the core circuit through the shared power rail by the digital control signals. Mike Dziawa, Mike Wingrove, and Jonathon Sewter from *Ciena* helped me on the advanced phase noise measurement equipment E5052B. They also provided me with very useful tips on the input/output (IO) control signal level shifting and board level power supply noise filtering techniques in the test setup.

My graduate study is an enjoyable journey with my lovely wife Sherry behind me all the time. I am greatly indebted to her support, encouragement, and sacrifice. My cheerful daughters Sherilyn and Alina are two great motivators for me to learn at their amazing pace and speed.

The final thanks go to my parents who gave me unconditional love and support in my whole life.

Contents

List of Tables

List of Figures

List of Symbols and Abbreviations

α	Integral path coefficient
β	Proportional path coefficient
$\Delta\Sigma$	Delta sigma modulator
$\mathcal{L}$	Phase noise
μm	Micro meter
ω_n	Natural frequency
ω_{3dB}	3dB bandwidth
$\otimes$	Convolution operator
ρ	Proportional path coefficient
σ_{tv}	Root mean square jitter of the DCO output clock
τ	Jitter
$\Theta_o(s)$	DCO/VCO output phase in the s-domain

$\theta_o(t)$	DCO/VCO output phase in the time domain
$\Theta_o(z)$	DCO/VCO output phase in the z-domain
$\Theta_r(s)$	Reference clock phase in the s-domain
$\theta_r(t)$	Reference clock phase in the time domain
$\Theta_r(z)$	Reference clock Phase in the z-domain
ζ	Damping factor
A_0	Loop filter DC gain
B_L	Noise bandwidth
C	Capacitance
D	Number of delay elements in ADPLL
$E(s)$	Error transfer function
f_O	PLL Output frequency
f_R	PLL Reference frequency
$G(s)$	Open loop transfer function
$H(s)$	Closed loop transfer function
K_d	PFD gain
K_d	Phase frequency detector gain

K_i	Integral path loop gain
k_i	Integral path coefficient
K_p	Proportional path loop gain
k_p	Proportional path coefficient
K_T	DCO Period gain
L	Inductance
nm	Nano meter
Q	Quality factor
R	Ratio between the integral path coefficient and the proportional coefficient
R	Resistance
s	Laplace variable
$V_c(s)$	VCO input control voltage in the s-domain
$v_c(t)$	VCO input control voltage in the time-domain
$V_c(z)$	VCO input control voltage in the z-domain
$V_d(s)$	Phase detector output voltage in the s-domain
$v_d(t)$	Phase detector output voltage in the time domain
$V_d(z)$	Phase detector output voltage in the z-domain

z	z-transformation variable
ADPLL	All digital phase locked loop
BBPFD	bang-Bang phase frequency detector that outputs only 0 and 1
BBPLL	Bang-bang PFD based PLL
BPD	Bang-Bang phase detector
C#	A programming language
CML	Common mode logic
CMOS	Complementary metal-oxide semiconductor
CORDIC	COordinate Rotation DIgital Computer,a simple and efficient algorithm to calculate hyperbolic and trigonometric functions
CP-PLL	Charge-pump-based PLL
DAC	Digital to analog converter
DCO	Digitally Controlled Oscillator
DDS	Direct digital systehsis
DPI	Direct programming interface in SystemVerilog
DRC	Design rule check
DSP	Digital signal processing
EMSS	Electromagnetic simulation software

ESD	Electrostatic discharge
FCW	Frequency control word
FPGA	Field Programmable Gate Array
I2C	Inter-integrated circuit serial communication protocol
IIR	Infinite impulse response filter
IO	Input/output
IP	Intellectual property
JTAG	Joint test action group
LC	inductor-capactor tank in an oscillator
LPF	Low pass filter
MMD	Multi-modulus divider
PC	Personal computer
PFD	Phase Frequency Detector
PLL	Phase locked loop
PVT	Process voltage and temperature variations
RMS	Root mean square
RS latch	Reset-set latch

RTL	Register transfer language
SoC	System on a chip
SPI	Serial to peripheral interface protocol
TDC	Time-to-digital converter
USB	Universal serial bus protocol
VCO	Voltage controlled oscillator

Chapter 1

Introduction

The all digital phase locked loop (ADPLL) remains one of the most interesting research topics. Compared to the analog phase locked loop (PLL) architecture, it depends much less on process, temperature, and voltage (PVT) variations due to its digital nature. The device matching requirement in analog PLL to compensate the process variation is greatly relaxed. Complex circuits to compensate the PVT variations in the analog PLL circuit are not needed anymore.

The PLL circuit is basically a closed loop feedback control system that makes the output frequency signal track an input frequency signal. When the output frequency is faster than the input frequency, the feedback system slows down the output frequency; when the output frequency is slower than the input frequency, the feedback system speeds up the output frequency.

Albeit its simple operating principle, the PLL technique received much attention since it was first conceived by the French scientist H.de Bellescise in his paper, "La Réception Synchrone", published in *Onde Electrique*, volume 11, 1932 [4].

The PLL started as the discrete element implementation and transformed into a partially integrated circuit, then to a fully integrated analog circuit, and finally to an all-digital circuit in recent years. Most of the early PLL implementations are analog circuits. The early digital form of the PLL often refers to the digital implementation with only a few building blocks due to the prohibitively high cost of the digital elements. The cost for the integrated circuit has been reduced to the point where the cost for the digital circuit is much less than the cost for the analog counterpart of the same functionality or the cost for the extra transistors required by the digital circuit can be ignored. This makes it possible to convert every single component in the PLL into a digital block. The state of the art digital PLL is called all digital phase locked loop (ADPLL) in that all the building blocks of the PLL are implemented digitally and have digital interface between them. The ADPLL itself is also fully integrated with other bigger digital circuits, such as a DSP, on a single die. Most often, there are more than one ADPLL in a single chip.

1.1 Comparison of ADPLL and Analog PLL

The first advantage of the ADPLL over the traditional PLL is its lower cost. Most of the analog PLLs require big capacitors in the low pass filter while the ADPLL implements the low pass filter as a digital circuit which occupies much smaller area than the capacitors required in the analog PLL.

The second advantage of the ADPLL over the traditional PLL is that it has higher noise immunity since a transistor generates much less noise when it is in the switched on-off state [5] [6] [7].

The third advantage of the ADPLL over the traditional analog PLL is that it is less sensitive to the PVT variations. The reason for this is obvious, the DC operating point of most of the transistors in an ADPLL is far away from their threshold voltage.

The fourth, but not the last, advantage of the ADPLL over the traditional analog PLL is that it can be easily ported from one technology to another technology. Since all the building blocks are made digital, migration from one technology to another technology does not require a full redesign and simulation of the whole ADPLL circuit.

The above advantages of the ADPLL do not come without any penalty. The most obvious disadvantage is its lower operating speed compared to the analog PLL. Most of the ADPLL published so far operate below 10GHz, with only a few in recent years between 10GHz and 40GHz [8] [9] [10], while the analog PLL can run at speed over 100GHz [11]. The ADPLL also suffers from high quantization noise due to its pure digital implementation.

1.2 ADPLL Applications

The following is a list of typical ADPLL applications that are described in the papers published in the past 5 years:

- TV band [12]
- Wimax/Wlan [13]
- Frequency synthesizer [14]
- GSM/GPRS/EDGE [15][16]

- Clock generation [14]
- Portable devices [14]
- Hard disk/optical drive [17]
- Optical interconnect applications [18]
- Built-in speed grading for memory [19]
- Video pixel clock regeneration [20]
- FM-radio transmitter [21]
- Software defined radios [22]

Those ADPLL applications falls into the wireline category and wireless category in the communication industries. The wireless PLL applications require low power and fine frequency resolution while the wireline PLL applications require low jitter or low phase noise. The targeted applications of the ADPLL in this thesis are the high-speed chip-to-chip serial interconnections, the optical transceivers for the backplane interconnections, and the low phase noise clock generators..

Architecturally, the PLL can be classified as the integer-N PLL and the fractional PLL. The integer-N PLL's output frequency is always an integer multiple of the input reference frequency whereas the fractional PLL's output frequency can be an integer plus a fraction of the input reference frequency.

The main advantage of the fractional PLL over the integer-N PLL is its ability to produce finer output resolution with the same input reference clock. This advantage does not come without any cost. The drawbacks of the fractional PLL are higher

power consumption and the limit-cycle induced spurs. Simply put, the limit-cycle induced spurs are due to the fact that the digital control word is limited and will cycle through a repeated pattern and this will create spurs in the output frequency spectrum. Most often, an integer-N PLL can be easily converted into a fractional PLL by adding an extra sigma-delta modulator to its frequency divider.

The subject of this thesis is to design and implement the proposed ADPLL architecture in CMOS 0.13μm and FPGA based on the research and simulation of the ADPLL.

1.3 Thesis Organization

The thesis is organized as follows:

Chapter 2 provides the necessary theoretical background in order to understand how the analog and all digital PLL work.

Chapter 3 proposes the ADPLL that will be implemented in this thesis based on the theoretical findings in the previous chapter.

Chapter 4 provides a system level behavioral simulation of the proposed ADPLL to make sure that the proposed ADPLL is functionally correct.

Chapter 5 describes a FPGA implementation of a variant ADPLL that is only different in the DCO implementation from the proposed ADPLL. The FPGA is close to the final CMOS implementation so it is another fast way to validates the proposed ADPLL. The test result of the FPGA implementation is also included in this chapter.

Chapter 6 describes the test setup and the test results of the fabricated ADPLL in CMOS 0.13/mum.

Chapter 7 concludes the thesis by a summary of the key research tasks, findings, results, and the future directions.

Chapter 2

Background

This chapter is a brief literature review of the ADPLL. The first section reviews three published papers that are relevant to the research area conducted in this thesis. The general analog PLL and digital PLL concept and detailed PLL theories can be easily understood by the s-domain and z-domain models in the subsections. They can also be easily found in books [23] [24] [25] [26], although different books might use different notations.

2.1 Literature Review

2.1.1 All-Digital PLL and Transmitter for Mobile Phones

Any author of a paper about ADPLL after 2005 is likely to have already read this reference [1]. This paper has already been cited 110 times as of this writing. In this paper, the author presents the first ADPLL-based transmitter for *GSM/EDGE* mobile phones implemented in 90 nm digital CMOS process.

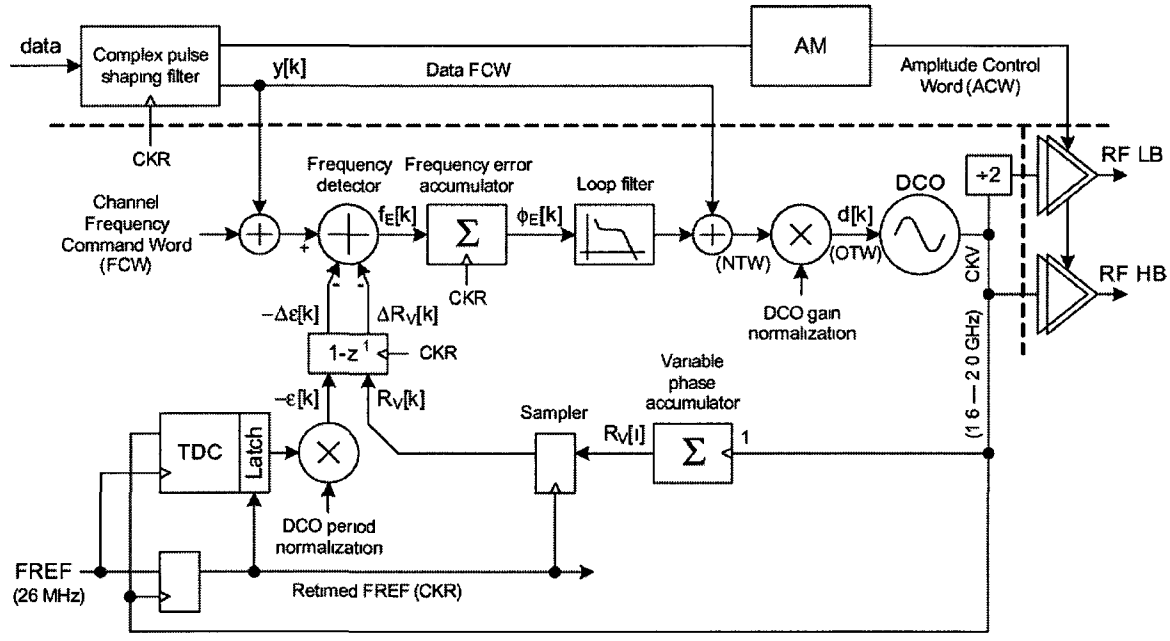

Figure 2.1: Polar Transmitter Based on All-Digital PLL (From [1]; ©2005 IEEE. Reprinted, with permission, from R. B. Staszewski, J. L. Wallberg, S. Rezeq, C.-M. Hung, O. E. Eliezer, S. K. Vemulapalli, C. Fernando, K. Maggio, R. Staszewski, N. Barton, M.-C. Lee, P. Cruise, M. Entezari, K. Muhammad, and D. Leipold, All-Digital PLL and Transmitter for Mobile Phones, IEEE J. Solid-State Circuits, vol. 40, no. 12, pp. 2469-2482, 2005.)

Figure 2.1 shows the ADPLL frequency synthesizer. The variable phase $R_V[i]$ is the output of the phase accumulator. It is then sampled by the retimed $FREF(CKR)$ and produces $R_V[k]$. This represents the fixed point phase value. The fixed point phase value is then concatenated with the TDC output $\epsilon[k]$ which represents the time differences between the FREF and DCO edges. The frequency error is calculated by the subtracting the differentiated phase error $R_V[k] - \epsilon[k]$ from the frequency control word (FCW):

$$f_E[k] = FCW - [(R_V[k] - \epsilon[k]) - (R_V[k-1] - \epsilon[k-1])] \tag{2.1}$$

The phase error $\phi_E[k]$ is the output of the accumulated frequency error $f_E[k]$ sampled by the retimed clock CKR:

$$\phi_E[k] = \sum_{l=0}^{k} f_E[k] \tag{2.2}$$

The phase error $\phi_E[k]$ is then filtered by a fourth order infinite impulse response (IIR) filter and scaled by the proportional coefficient α and integral coefficient ρ which is not shown in Figure 2.1. The filtered and scaled phase error is then multiplied by the DCO gain K_{DCO} normalization factor $f_R/\hat{K}_{DCO}$, where f_R is the reference frequency and $\hat{K}_{DCO}$ is the DCO gain estimator. The process produces the digital oscillator tuning word (OTW) to control the DCO. This is a high level description of the ADPLL in this paper.

The characteristic of this architecture is the use of the TDC which measures the fractional delay difference between the reference clock FREF and the DCO output CKV. The TDC resolution used in this paper is about 20 psec. The in-band phase noise is affected by the TDC resolution:

$$\mathcal{L} = \frac{(2\pi)^2}{12} \left(\frac{\Delta t_{inv}}{T_V} \right)^2 \frac{1}{f_R} \tag{2.3}$$

where Δt_{inv} is the resolution of a single inverter delay, T_V is the DCO clock period, and f_R is the reference or sampling frequency.

The circuit in this paper is implemented in the digital 90 nm CMOS process. The key parameters used in the circuit are $\alpha = 2^{-7}$, $\rho = 2^{-15}$, where α is the integral path coefficient and ρ is the proportional path coefficient.

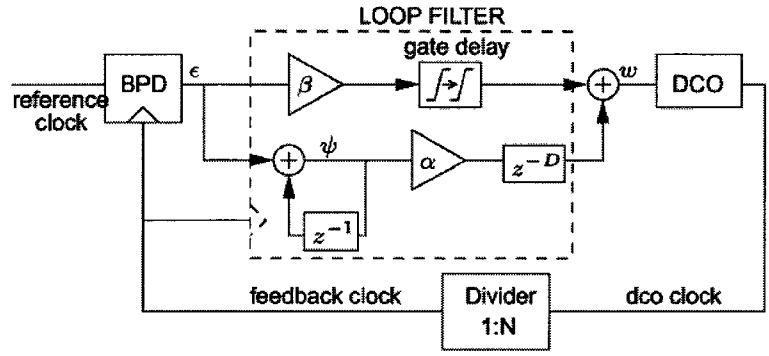

Figure 2.2: Linearized Analysis of a Digital Bang-Bang PLL and Its Validity Limits Applied to Jitter Transfer and Jitter Generation (From [2]; ©2005 IEEE. Reprinted, with permission, from N. Da Dalt, Linearized Analysis of a Digital Bang-Bang PLL and Its Validity Limits Applied to Jitter Transfer and Jitter Generation, IEEE Trans. Circuits Syst. I, vol. 55, no. 11, pp. 3663-3675, 2008.)

The in-band synthesizer phase noise is measured below -93 dBc/Hz for the loop bandwidth of 40 kHz and -121.6 dBc/Hz at offset 400 KHz with the carrier frequency of 824.2 MHz.

2.1.2 Linearized Analysis of a Digital Bang-Bang PLL and Its Validity Limits Applied to Jitter Transfer and Jitter Generation

Compared to [1] that uses a time-to-digital converter (TDC) as the way to measure the phase error, the circuit architecture described in [2] uses a simple Bang-Bang phase detector (BPD) which greatly simplifies the circuit design. The Bang-Bang terms refers to the fact that the phase detector output is of the digital form of either

zero or one. Figure 2.2 shows the block diagram of a second order digital Bang-Bang phase frequency detector (BBPFD) based phase locked loop (BBPLL) described in this paper. α is the integral path coefficient; β is the proportional path coefficient; and D is the integral path delay. Since the ADPLL is a highly non-linear circuit, the linearized analysis given by this paper only holds when the ADPLL is in the locked state.

The time domain operation of the BBPLL is governed by the following equation:

$$\begin{cases} \tau_{k+1} = \tau_k + x_0 - R \cdot \psi_{k-D} - sgn(\tau_k) \\ \psi_{k+1} = \psi_k + sgn(\tau_{k+1} \end{cases} \tag{2.4}$$

where R is the ratio between the integral path coefficient and the proportional coefficient α/β; $\tau = \Delta/(N\beta K_T)$ is the jitter seen at the BPD; K_T is the period gain of the DCO; ψ is the integrator output; and x_0 is the detuning of the DCO.

The key concept in this paper is to have the 2-Dimensional orbit representation of the limit-cycle caused by the limited word width in any digital circuit. Based on this concept, the author finds the stability condition of the BBPLL to be:

$$R < R_{crit} = \frac{2}{2D+1} \tag{2.5}$$

The BPD gain K_d expression

$$K_d \approx \frac{1}{\sqrt{2\pi\sigma_{t_r}}} \left[1 + e^{-\frac{1}{2}\left(\frac{N\beta K_T}{\sigma_{t_r}}\right)}\right] \approx \frac{1}{\sqrt{2\pi\sigma_{t_r}}} \tag{2.6}$$

derived in [27] by the same author is used in the linear model of the BPD to derive the variance of the output jitter:

$$\frac{\sigma_{t_v}^2}{N\beta K_T} = \frac{5}{4\sqrt[4]{2\pi}}\sqrt{\frac{\sigma_{t_r}}{N\beta K_T}} \tag{2.7}$$

where σ_{t_r} is the root mean square (RMS) jitter of the reference clock; σ_{t_v} is the RMS jitter of the DCO output clock; N is the divider ratio; β is the proportional path coefficient; and K_T is the DCO period gain.

From equation 2.6, it can be seen that the phase detector gain depends on the RMS jitter of the input reference clock. From equation 2.7, it can be seen that the output RMS jitter grows linearly with the square root of the input RMS jitter provided that the input jitter is larger than the minimum quantization step of the BBPLL.

The theory presented in this paper is validated with a BBPLL implemented in 0.13 μm CMOS technology. Two output bands centered at 2.2 GHz and 4.8 GHz are supported. The DCO phase noise of -110 dBc/Hz at 1 MHz with 2.2 GHz and 4.8 GHz carriers respectively.

2.1.3 A 4GHz Low Complexity ADPLL-based Frequency Synthesizer in 90 nm CMOS

Figure 2.3 shows a low complexity ADPLL-based frequency synthesizer implemented in 90 nm CMOS technology.

A conventional tri-state phase frequency detector (PFD) is used in this work, followed by the one-bit T2D. The frequency decision circuit consists of a decision clock to define the decision period (N), and a logical circuit to make decision at each rising edge of decision clock. The DCO in this paper is implemented as an LC tank oscillator with tunable varactor as the C. In order to achieve the fine DCO resolution of

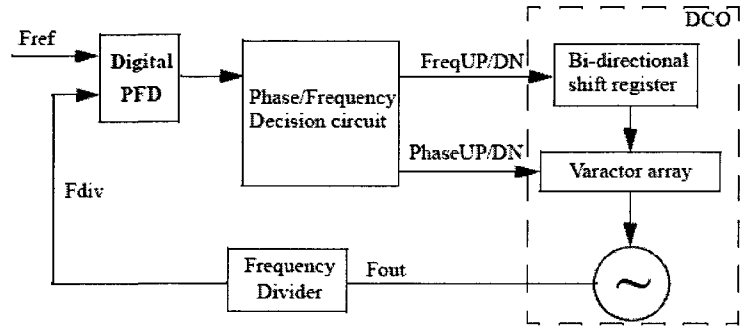

Figure 2.3: A 4GHz Low Complexity ADPLL-based Frequency Synthesizer in 90nm CMOS (From [3]; ©2005 IEEE. Reprinted, with permission, from N. Da Dalt, Linearized Analysis of a Digital Bang-Bang PLL and Its Validity Limits Applied to Jitter Transfer and Jitter Generation, IEEE Trans. Circuits Syst. I, vol. 55, no. 11, pp. 3663-3675, 2008.)

5 KHz, a capacitance tuning scheme with incrementally-sized varactors and matched varactor banks is employed. Instead of adding or removing a smallest possible varactor C_0 in a certain technology by thermometer coding, the author enables or disables any single varactor in an array with the increasing varactor step size $\Delta C0$ by one-hot coding, which means only one bit can be set at any time. Since ΔC_0 is usually one magnitude smaller than C_0, a finer frequency step size can be realized without violating the design rule checking (DRC).

The achieved phase noise of the circuit implemented in 90 nm CMOS in this paper is -106 dBc/Hz at 1 MHz offset of the 3.916 GHz carrier.

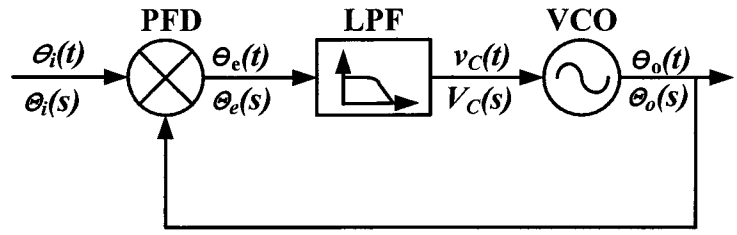

Figure 2.4: Phase Locked Loop Block Diagram

2.2 Linearized s-Domain Model of the Analog PLL

Although most PLL circuits, notably the digital PLLs, operate in non-linear fashion, the linearized modeling of the PLL is proved to be simple to understand and it is a power tool to gain insight into how a PLL works in the locked state.

In this section, the classical s-domain linearized model of the analog PLL is introduced based on the block diagram in Figure 2.4. Most steps of the derivations can be found in [23] [24] [25] and [26]. The mappings from the time domain convolution equations to the s-domain multiplication equations for the LPF's operation are added by the author for the sake of completeness.

Please note that the divider is normalized into the voltage controlled oscillator (VCO) in the derivations. Simply replacing all the VCO gain k_o with k_o/N in all the equations derived here results in equations for the de-normalized VCO, where N is the divider ratio.

The phase of the input reference clock signal is denoted by:

$$\theta_r(t) = \omega_r t \tag{2.8}$$

and the phase of the VCO is denoted by:

$$\theta_o(t) = \omega_o t + \phi_o(t) \tag{2.9}$$

The PFD compares these two phase inputs and produces the phase error:

$$\theta_e(t) = \theta_r(t) - \theta_o(t) \tag{2.10}$$

in the time domain. The s-domain phase error is simply:

$$\Theta_e(s) = \Theta_r(s) - \Theta_o(s) \tag{2.11}$$

Assume that the output of the PFD is a voltage and is linear in the phase locked condition so that its output can be written as:

$$v_d(t) = k_d[\theta_r(t) - \theta_o(t)] \tag{2.12}$$

where k_d is the the PFD gain. The s-domain PFD output is

$$V_d(s) = k_d[\Theta_r(s) - \Theta_o(s)] \tag{2.13}$$

The PFD output is processed by the low pass filter (LPF) to produce the control voltage of the VCO. The high frequency noise signal components are suppressed by this LPF. The loop dynamics are mainly determined by LPF as will be discussed shortly. The time domain output of the LPF is expressed as the convolution of $v_d(t)$ and LPF's impulse response function $f(t)$:

$$v_c(t) = f(t) \otimes v_d(t) = \int_0^t f(t-\tau) v_d(t) d\tau \tag{2.14}$$

To solve the output $v_c(t)$ in the time domain is quite tedious. Fortunetely, a loop filter can be analyzed easily in the s-domain. Please note that the s-domain signals in this thesis is represented by capital letters corresponding to its time domain lower case signal names. The s-domain LPF output can be easily converted by applying *Laplace Convolution Theorem*

$$V_C(s) = F(s)V_d(s) = k_d F(s)\Theta_e(s) \tag{2.15}$$

The phase deviation of the VCO caused by the control voltage is:

$$d\theta(t) = k_o v_C(t)dt \tag{2.16}$$

in the time domain, where k_o is the VCO gain.

Integrating both sides of the above equation results:

$$\theta_o(t) = \int k_o v_C(t)dt \tag{2.17}$$

in the time domain or

$$\Theta_o(s) = \frac{k_o V_C(s)}{s} = \frac{k_d k_o F(s)\Theta_e(s)}{s} \tag{2.18}$$

in the s-domain after applying the Laplace Integration Theorem.

The s-domain open loop transfer function is defined as:

$$G(s) \equiv \frac{\Theta_o(s)}{\Theta_e(s)} \tag{2.19}$$

It is clear from 2.18:

$$G(s) = \frac{K_d K_o F(s)}{s} \tag{2.20}$$

The s-domain closed loop transfer function is defined as:

$$H(s) \equiv \frac{\Theta_o(s)}{\Theta_r(s)} \tag{2.21}$$

It can be shown by combining 2.11, 2.19, and 2.21 that

$$H(s) = \frac{G(s)}{1 + G(s)} \tag{2.22}$$

Applying 2.18, we get

$$H(s) = \frac{k_d k_o F(s)}{s + k_d k_o F(s)} \tag{2.23}$$

The s-domain error transfer function is define as

$$E(s) \equiv \frac{\Theta_e(s)}{\Theta_i(s)} \tag{2.24}$$

Similarly, we can get

$$E(s) = \frac{1}{1 + G(s)} = 1 - H(s) \tag{2.25}$$

Applying either 2.20 or 2.23, we get

$$E(s) = \frac{s}{s + k_d k_o F(s)} \tag{2.26}$$

Equations 2.20 2.23 and 2.26 are generic equations that describe the dynamic of an analog PLL system regardless of what the loop filter is. In the following subsections,

we will introduce the PLL's design parameters and performance parameters as we apply certain filter types in the PLL system.

2.2.1 Analog PLL With Only Proportional Path

The simplest analog PLL is the one that has a filter with transfer function $F(s) = k_p$. The loop gain, open loop transfer function, closed loop transfer function, and the error transfer function are:

$$G(s) = \frac{K_p}{s} \tag{2.27}$$

$$H(s) = \frac{K}{s + K_p} \tag{2.28}$$

$$E(s) = \frac{s}{s + K_p} \tag{2.29}$$

where K_p is defined as:

$$K_p = k_d k_o k_p \tag{2.30}$$

2.2.2 Analog PLL With Only Integral Path

The other simple analog PLL is the one that has a filter with transfer function $F(s) = k_i/s$. The loop gain, open loop transfer function, closed loop transfer function, and the error transfer function are:

$$G(s) = \frac{K_i}{s^2} \tag{2.31}$$

$$H(s) = \frac{K_i}{s^2 + K_i} \tag{2.32}$$

$$E(s) = \frac{s^2}{s^2 + K_i} \tag{2.33}$$

where K_i is defined as:

$$K_i = k_d k_o k_i \tag{2.34}$$

2.2.3 Analog PLL With Both Proportional and Integral Path

The loop gain, open loop transfer function, closed loop transfer function, and the error transfer function for the analog PLL with both the proportional path and the integral path are:

$$G(s) = \frac{K_p s + K_i}{s^2} \tag{2.35}$$

$$H(s) = \frac{K_p s + K_i}{s^2 + K_p s + K_i} \tag{2.36}$$

$$E(s) = \frac{s^2}{s^2 + K_p s + K_i} \tag{2.37}$$

$$\omega_n = \sqrt{K_i} = \sqrt{k_d k_o k_i} \tag{2.38}$$

$$\zeta = \frac{K_p}{2\omega_n} = \frac{K_p}{2\sqrt{K_i}} = \frac{k_p}{2}\sqrt{\frac{k_d k_o}{k_i}} \tag{2.39}$$

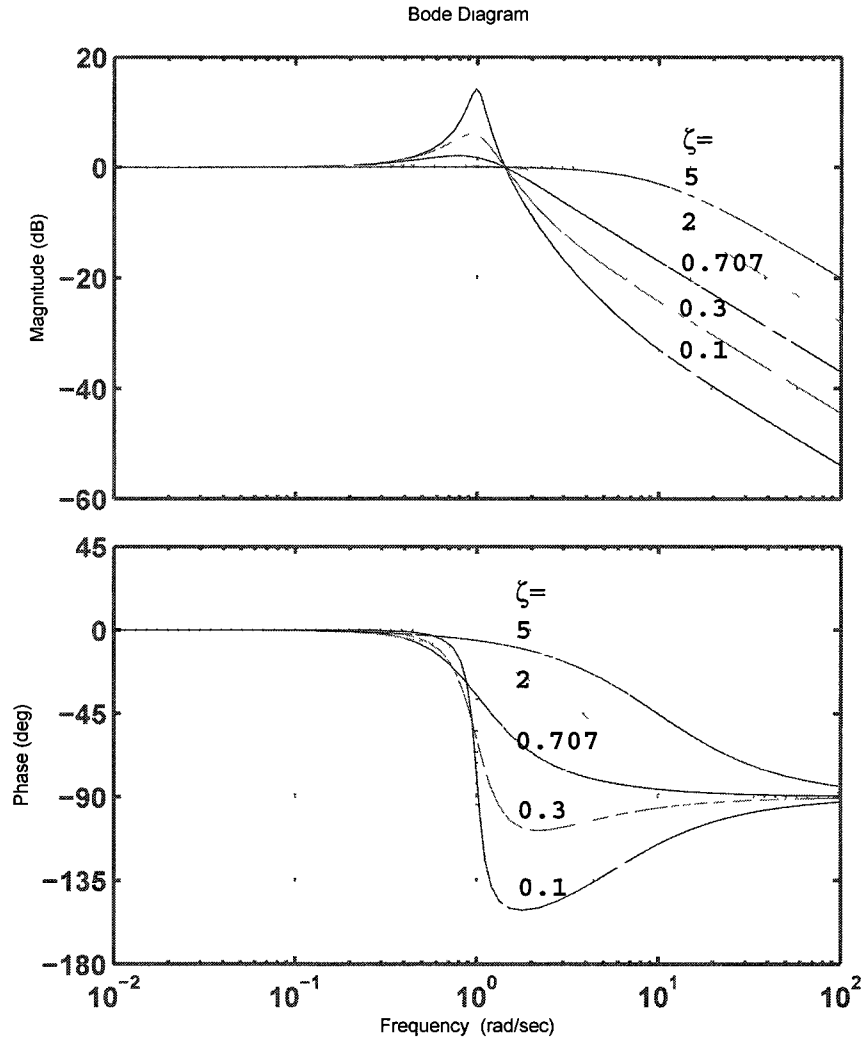

Figure 2.5: Bode Diagram of $|H(s)|$ for a Second-Order Type 2 PLL With Frequency Normalized to ω_n

2.2.4 Natural Frequency

Natural frequency ω_n provides a good indication of the PLL bandwidth, but it is a strong function of the damping factor ζ as can be seen from Figure 2.5. For this

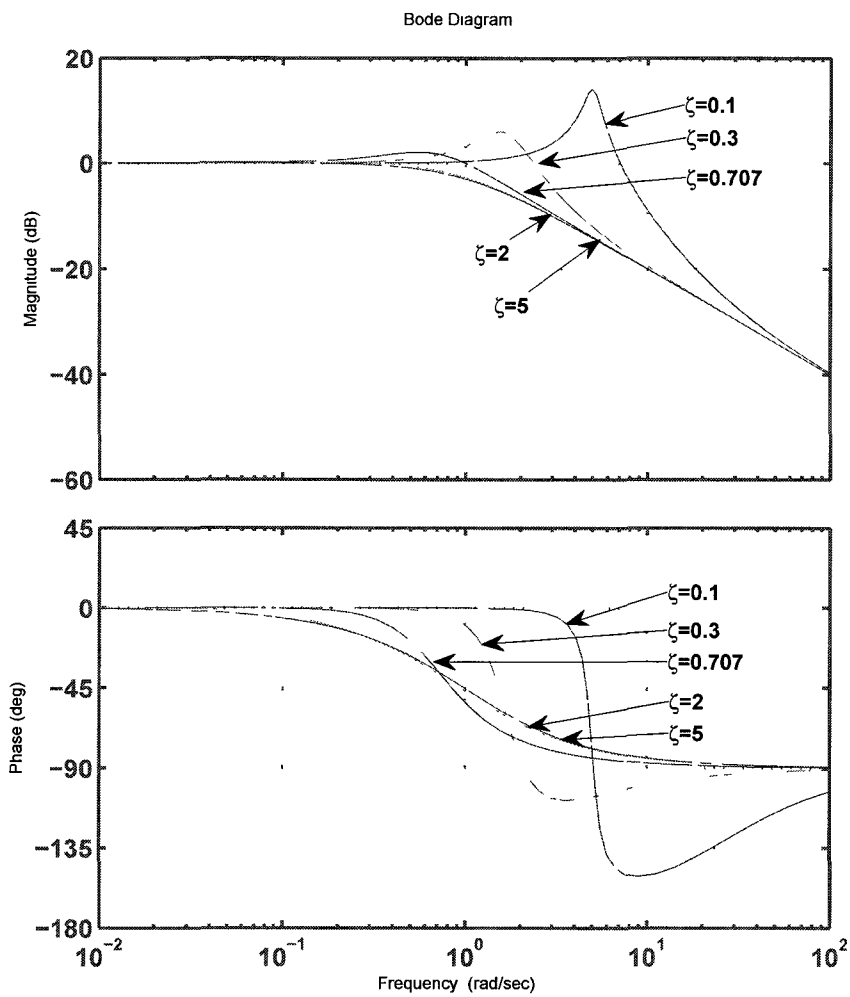

Figure 2.6: Bode Diagram of $|H(s)|$ for a Second-Order Type 2 PLL With Frequency Normalized to K_p

reason, ω_n is not a good choice for the PLL bandwidth.

2.2.5 Loop Gain

From Figure 2.6, we can see that the loop gain K_p is a good indication of corner frequency of $H(s)$. Unlike ω_n, which may have no definition for other order PLL, such as first order PLL, K_p is a choice of the PLL bandwidth for any type of PLL [24].

2.2.6 Noise Bandwidth

The noise bandwidth B_L is defined as:

$$B_L = \int_0^\infty |H(f)|^2 \, df \tag{2.40}$$

If the referred input reference signal has significant white noise, the noise bandwidth can be a good indication of $H(s)$ corner frequency and can be estimated easily from a phase noise measurement graph. However, if the input source has already been a noised shaped signal, such as a cascaded PLL source, or is a very low noise source, the noise bandwidth B_L will not be a good choice for the PLL bandwidth. In this cases, the loop gain K will be preferred.

2.2.7 3-dB Bandwidth

Since a PLL can be seen as a low pass filter and filters are commonly specified by the 3-dB bandwidth. The closed form of a the 3-dB bandwidth for a second order type 2 PLL can be derived as:

$$\omega_{3dB} = \omega_n \sqrt{1 + 2\zeta^2 + \sqrt{4\zeta^4 + 4\zeta^2 + 2}} \tag{2.41}$$

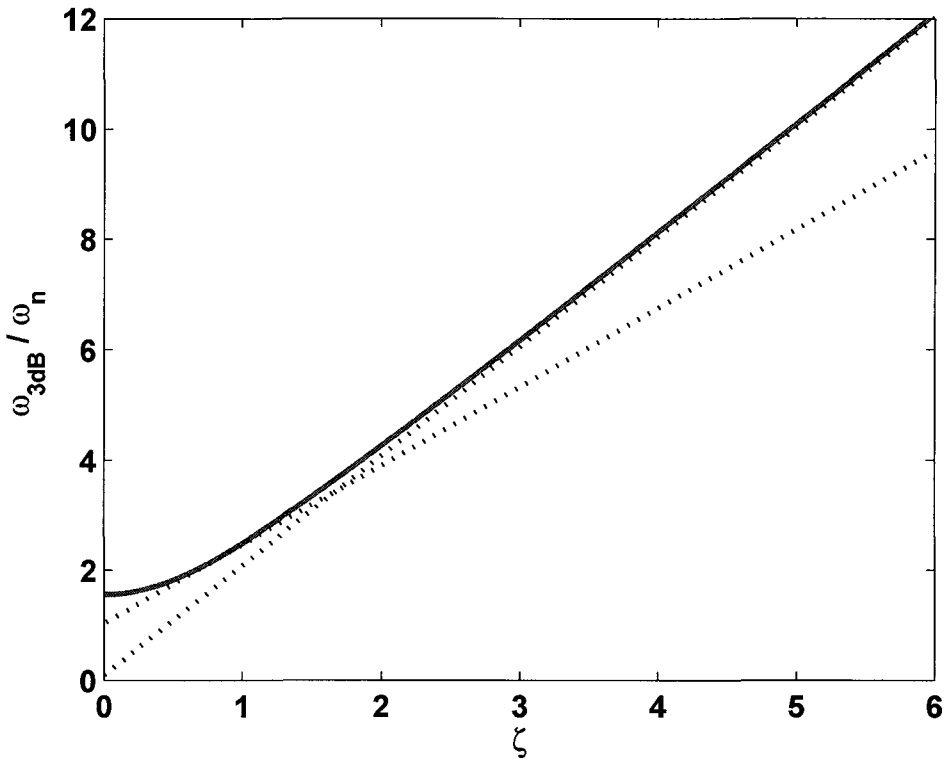

Figure 2.7: 3-dB Bandwidth vs Frequency Normalized to ω_n

The 3-dB bandwidth ω_{3dB} normalized to the natural frequency ω_n is plotted in Figure 2.7.

For quick estimate of the 3-dB bandwidth, the short form of the 3-dB bandwidth can be used:

$$\omega_{3dB} = \begin{cases} 2\zeta\omega_n & \text{if } \zeta > 1.7 \\ (1+\zeta\sqrt{2})\omega_n & \text{if } \zeta \leq 1.7 \end{cases} \tag{2.42}$$

As can be seen from Figure 2.7, the short form 3-dB bandwidth estimate is fairly accurate for $\zeta > 0.4$.

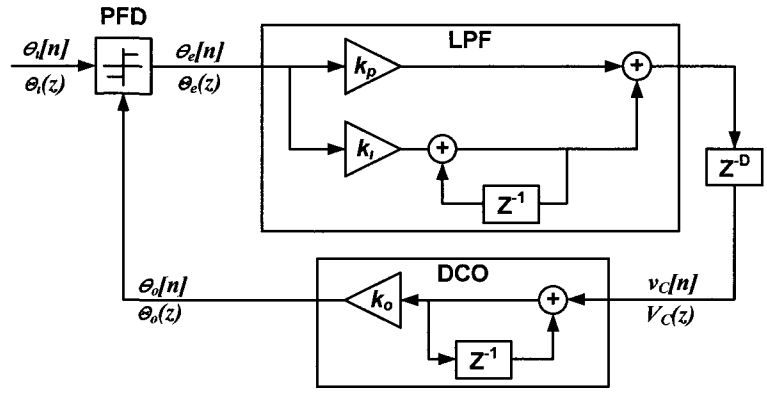

Figure 2.8: All Digital Phase Locked Loop Block Diagram

2.3 Linearized z-Domain Model of the ADPLL

The derivation of the linearized z-domain model of the ADPLL is similar to that of the linearized s-domain model of the ADPLL. Figure 2.8 shows the linerized z-domain model of the ADPLL. The phase of the input reference clock signal is denoted by:

$$\theta_r[n] = \omega_r n \tag{2.43}$$

and the phase of the voltage controlled oscillator (VCO) is denoted by:

$$\theta_o[n] = \omega_o n + \phi_o[n] \tag{2.44}$$

The PFD compares these two phase inputs and produces the phase error:

$$\theta_e[n] = \theta_r[n-1] - \theta_o[n-1] \tag{2.45}$$

in the time domain. The z-domain phase error is simply:

$$\Theta_e(z) = \frac{1}{z}(\Theta_r(z) - \Theta_o(z) \tag{2.46}$$

Assume that the output of the PFD is a voltage and is linear in the phase locked condition so that its output can be written as:

$$v_d[n] = k_d[\theta_r[n] - \theta_o[n]] \tag{2.47}$$

where k_d is the the PFD gain. The z-domain PFD output is:

$$V_d(z) = \frac{k_d}{z}[\Theta_r(z) - \Theta_o(z)] \tag{2.48}$$

The PFD output is processed by the LPF to produce the control voltage of the VCO. The high frequency noise signal components are suppressed by this LPF. We will see that the loop dynamics are mainly determined by LPF as will be discussed shortly. The time domain output of the LPF is expressed as the convolution of $v_d(t)$ and LPF's impulse response function $f(t)$:

$$v_c[n] = f[n] \otimes v_d[n] = \sum_{m=-\infty}^{\infty} f(n-m)v_d[m] \tag{2.49}$$

To solve the output $v_c(t)$ in the time domain is tedious. Fortunetely, a digital loop filter can be analyzed easily in the z-domain. Please note that the z-domain signals in this thesis represented by capital letters corresponding to its time domain lower case signal names. The z-domain LPF output can be easily converted by applying *Laplace Convolution Theorem*

$$V_C(z) = F(z)V_d(z) = k_d F(z)\Theta_e(z) \tag{2.50}$$

Applying D sampling clock delay to $V_C(z)$ and rewriting it without introducing a new symbol, we get:

$$V_C(z) = \frac{k_d F(z)\Theta_e(z)}{z^D} \tag{2.51}$$

The phase deviation of the VCO caused by the control voltage is:

$$\theta[n] - \theta[n-1] = k_o v_C[n-1] \tag{2.52}$$

in the time domain, where k_o is the DCO gain.

The corresponding z-domain equation by applying the Differential Theorem on the left and the Time Shift Theorem on the right results in:

$$(1 - z^{-1})\Theta_o(z) = k_o v_C(z) z^{-1} \tag{2.53}$$

re-arranging the above equation, we get:

$$\Theta_o(z) = k_o v_C(z)\frac{z^{-1}}{1 - z^{-1}} = k_d k_o \Theta_e(z)(z^{-D}F(z))\frac{z^{-1}}{1 - z^{-1}} \tag{2.54}$$

The z-domain open loop transfer function is define as:

$$G(z) \equiv \frac{\Theta_o(z)}{\Theta_e(z)} \tag{2.55}$$

It is clear from 2.54:

$$G(z) = k_d k_o (z^{-D}F(z))\frac{z^{-1}}{1 - z^{-1}} \tag{2.56}$$

The z-domain closed loop transfer function is defined as

$$H(z) \equiv \frac{\Theta_o(z)}{\Theta_r(z)} \tag{2.57}$$

It can be shown by 2.46 that:

$$H(z) = \frac{G(z)}{1 + G(z)} \tag{2.58}$$

Applying 2.54, we get:

$$H(z) = \frac{k_d k_o (z^{-D} F(z)) z^{-1}}{(1 - z^{-1}) + k_d k_o (z^{-D} F(z)) z^{-1}} \tag{2.59}$$

The z-domain error transfer function is define as:

$$E(z) \equiv \frac{\Theta_e(z)}{\Theta_i(z)} \tag{2.60}$$

Similarly, we can get:

$$E(z) = \frac{1}{1 + G(z)} = 1 - H(z) \tag{2.61}$$

Applying either 2.56 or 2.59, we get:

$$E(z) = \frac{1 - z^{-1}}{(1 - z^{-1}) + k_d k_o (z^{-D} F(z)) z^{-1}} \tag{2.62}$$

Equations 2.20 2.59 and 2.62 are generic equations that describe the dynamic of a PLL system regardless of the loop filter. In the following subsections, we will introduce the PLL's design parameters and performance parameters as we apply certain filter types in the PLL system.

2.3.1 ADPLL With Only Proportional Path

The simplest ADPLL is the one that has a filter with transfer function $F(z) = k_p$. The loop gain, open loop transfer function, closed loop transfer function, and the error transfer function are

$$G(z) = \frac{K_p z^{-1-D}}{1 - z^{-1}} \tag{2.63}$$

$$H(z) = \frac{K_p z^{-1-D}}{1 + z^{-1}(K_p - 1)} \tag{2.64}$$

$$E(z) = \frac{1 - z^{-1}}{1 + z^{-1-D}(K_p - 1)} \tag{2.65}$$

where K_p is defined as

$$Kp = k_d k_o k_p \tag{2.66}$$

2.3.2 ADPLL With Only Integral Path

The other simple ADPLL is the one that has a filter with transfer function $F(z) = k_i z^{-1}/(1 - z^{-1})$. The loop gain, open loop transfer function, closed loop transfer function, and the error transfer function are:

$$G(z) = K_i z^{-D} \left(\frac{z^{-1}}{1 - z^{-1}} \right)^2 \tag{2.67}$$

$$H(z) = \frac{K_i z^{-D}(1 - z^{-1})^2}{(1 - z^{-1})^2 + K_i z^{-2-D}} \tag{2.68}$$

$$E(z) = \frac{z^{-2}}{(1 - z^{-1})^2 + K_i (z^{-1-D})^2} \tag{2.69}$$

where K_i is defined as:

$$K_i = k_d k_o k_i \tag{2.70}$$

2.3.3 ADPLL With Both Proportional and Integral Path

The loop gain, open loop transfer function, closed loop transfer function, and the error transfer function for an ADPLL with both the proportional path and the integral path are:

$$G(z) = \frac{[K_p(1 - z^{-1}) + K_i z^{-1}]z^{-1-D}}{(1 - z^{-1})^2} \tag{2.71}$$

$$H(z) = \frac{K_p z^{-D}(1 - z^{-1}) + K_i z^{-1-D}}{(1 - z^{-1})^2 + K_p z^{-1-D} + K_i z^{-D}} \tag{2.72}$$

$$E(z) = \frac{(1 - z^{-1})^2}{(1 - z^{-1})^2 + K_p z^{-D}(1 - z^{-1}) + K_i z^{-1-D}} \tag{2.73}$$

Using $z = e^{-\jmath\omega t_s}$, where t_s is the sampling period, we can define the natural frequency and damping factor corresponding to those of the analog PLL. This is only valid if there is no delay D and when the bandwidth is less than the sampling frequency:

$$\omega_n = \sqrt{K_i} = \sqrt{k_d k_o}\sqrt{k_i} \tag{2.74}$$

$$\zeta = \frac{K_p}{2\omega_n} = \frac{K_p}{2\sqrt{K_i}} = \frac{\sqrt{K_d k_o}}{2}\frac{k_p}{\sqrt{k_i}} \tag{2.75}$$

The Bode plots of the ADPLL are similar to those of analog PLLs. If the bandwidth is relative small compared to the sampling rate of the ADPLL, then the ADPLL will work like an analog PLL.

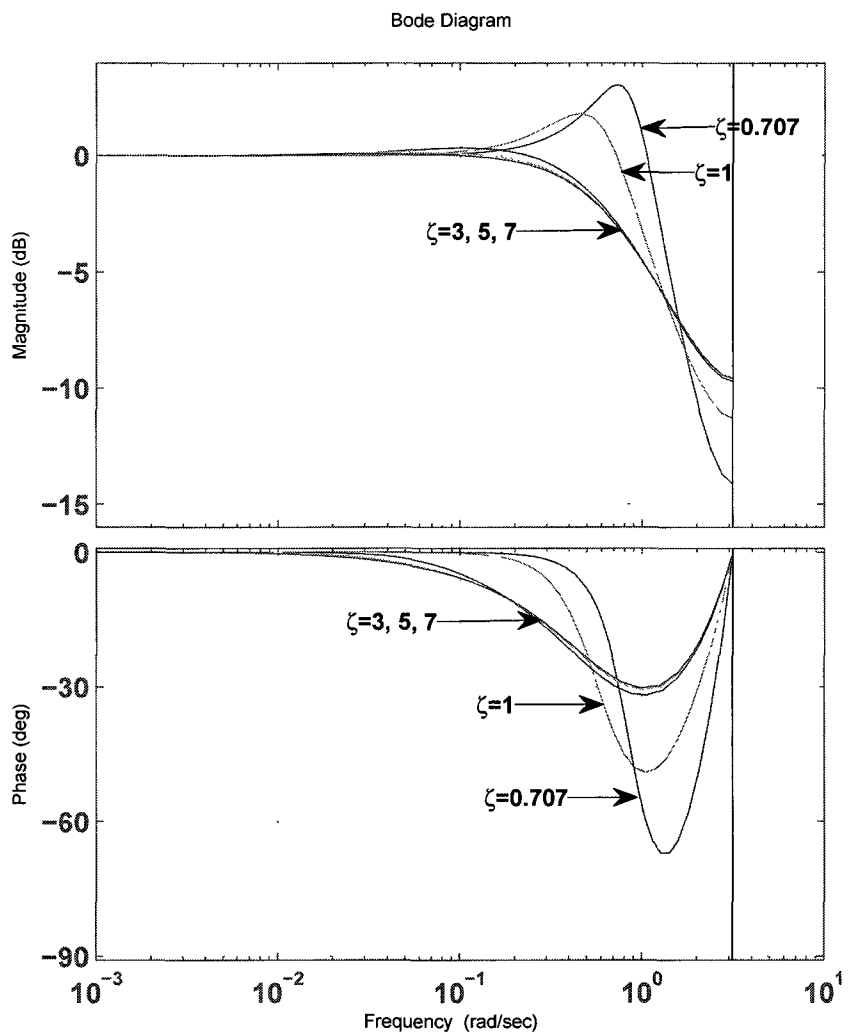

Figure 2.9: Bode Diagram of $|H(z)|$ for a Second-Order Type 2 ADPLL, D=1 With Frequency Normalized to Loop Gain K.

As can be seen that the Bode plots of the ADPLL shown in Figure 2.9 are similar to the Bode plots for an analog PLL, shown in Figure 2.6. The ADPLL in Figure 2.9 has large phase margin, therefore it is stable.

Chapter 3

The Proposed All Digital Phase Locked Loop

The proposed ADPLL in this chapter consists of three major modules as shown in Figure 3.1:

- The non-linear BBPFD without cycle-slip and dead-zone. This BBPFD takes the reference signal $Fref$ and the divider output signal $Fdiv$ and produces a single rectangle phase direction signal Dir instead of the pulsed signals UP and DN by a traditional linear PFD.
- The highly integrated DCO that contains both the integral and the proportional filter paths. This DCO takes the low speed PFD output signal Dir and the signal $Shift$ and produces the high speed synthesized clock output $Fout$. When running in open-loop mode, it can be characterized easily by the external test signals Dir and $Shift$ independent of all other ADPLL modules. K_p is the programmable proportional path coefficient.

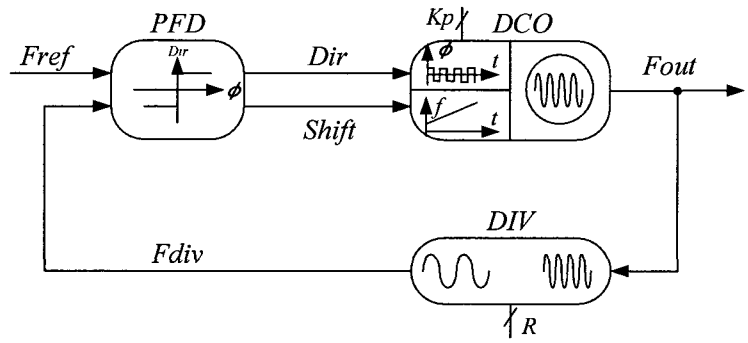

Figure 3.1: The Proposed All Digital Phase Loop (Kp and R are programmable proportional path coefficient and divide ratio)

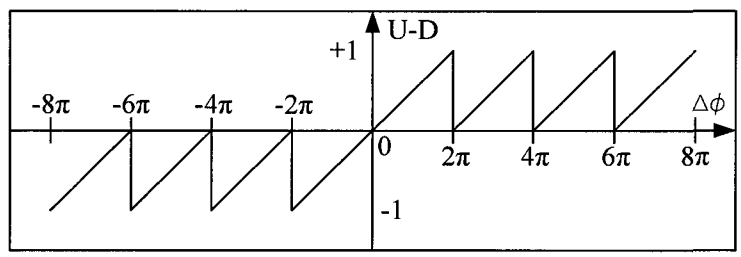

Figure 3.2: The Characteristic Curve of a Traditional 3-state PFD

- The wide divide range multi-modulus divider (MMD) that is capable of dividing by the divide ratio R from 2 to $2^n - 1$, where n is the number of divide-by-2/3 stages plus 1. The advantage of the wide divide range MMD is that the input reference frequency range is also wide, giving the designer more choices.

3.1 PFD Without Cycle-Slip and Narrow Dead-Zone

The traditional 3-state PFD takes the reference clock signal $Fref$ and the frequency divider output signal $Fdiv$ and produces two pulse outputs U and D. Followed by the charge pump circuit in the analog PLL, the traditional 3-state PFD is considered to be linear because the integrated area under the $U - D$ signals are proportional to the phase difference $\phi(Fref) - \phi(Fdiv)$. When in the phase locked state, the pulse U and D is not wide enough to drive the big capacitive load presented at the charge pump input due to the finite rise time and fall time, resulting in the metastability or the dead-zone problem. Within the dead-zone, the effect of the metastability is to smooth out the binary characteristic curve of the BBPFD thus leading to linear operation with small phase errors [28]. So the dead-zone problem is not a real problem per se. The real problem associated with the dead-zone is that the dead-zone can not be too wide. If the dead-zone is too wide, the loop will not be able to have enough loop gain to correct small phase error. For this reason, it is desirable to have a very narrow dead-zone.

Another less severe but unwanted effect of the traditional 3-state PFD when in frequency locked state is the cycle slip. The cycle slip is illustrated usually with a step response figure and is a direct effect result of the PFD characteristic curve shown in Figure 3.2. $U - D$ in Figure 3.2 is the averaged PFD output signal and $\Delta\phi$ is the the phase error between the two PFD inputs. In the analog PLL with the charge pump configuration, the cycle slip slows down the frequency step response and makes the settling time longer than needed.

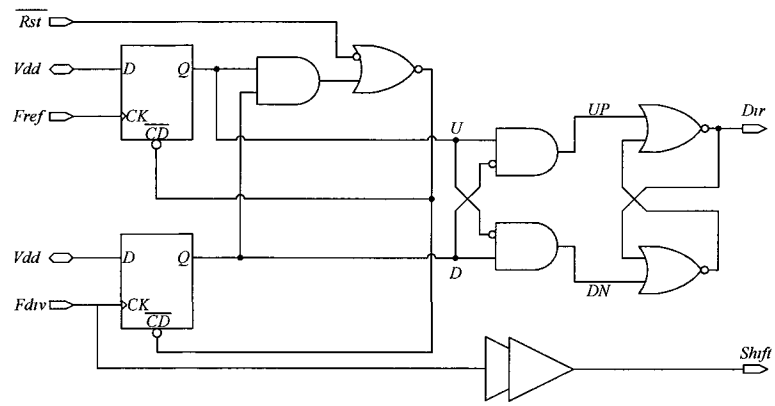

Figure 3.3: The Phase and Frequency Detector Without Cycle-Slip and Narrow Dead-Zone

Table 3.1: PFD Glitch Elimination Truth Table

U	D	UP	DN
0	0	0	0
0	1	0	1
1	0	1	0
1	1	0	0

The proposed PFD used in this thesis is shown in Figure 3.3. It consists of a traditional PFD followed by a glitch elimination stage and a cycle slip elimination stage. The glitch elimination stage consists of two AND gates with one inverted input connected to the other gate's input. Its operation follows the truth table 3.1:

As can be seen from this glitch elimination table that, when both U and D are high, the output UP and DN are low and the net effect of this stage is that the

Table 3.2: PFD RS Latch Truth Table

UP	DN	Dir
0	0	Keep State
0	1	0
1	0	1
1	1	Restricted

narrow glitches associated with the traditional 3-state PFD is eliminated, as can be seen in Figure 3.5. Please note that eliminating the state where both UP and DN are high also eliminates the undefined input state in the following cycle slip stage.

The cycle slip is eliminated with an reset-set (RS) latch following the dead-zone elimination circuit. The cycle elimination circuit truth table is shown in Table 3.2

With the cycle slip elimination circuit, the proposed PDF's characteristic curve is changed from Figure 3.2 to Figure 3.4 making it a non-linear Bang-Bang phase frequency detector. When in the phase locked condition, the ideal shape of the signal Dir is a squared waveform.

Since the glitch elimination stages and the cycle slip stage also serve as two high gain amplifiers, the flip-flops in the proposed PFD can be designed with very small gate width because the capacitive load seen by U and D signals is the glitch elimination stage only. This effect is called reverse scaling and it results in an extremely short time of metastability and a narrow dead-zone. The narrow the dead-zone, the higher speed that the the PDF can operate. This is confirmed with the measured

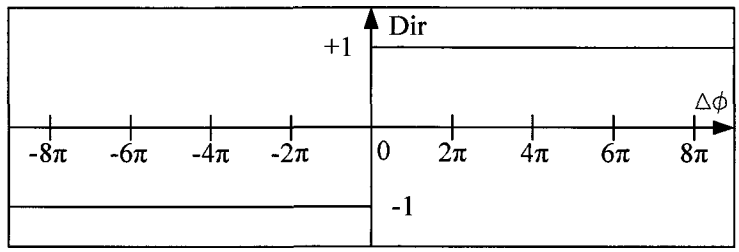

Figure 3.4: The Characteristic Curve of the Proposed PFD

output frequency spectrum with 1Hz resolution bandwidth in Figure 6.9.

Compared with the narrow pulsed up-down signal U and D used in the traditional analog PLL, the square waved signal Dir have the following advantages:

- Since the Dir signal is extended in the whole reference period, the proportional path phase correction is averaged out in the whole reference period and thus requires a much smaller proportional path varactor array.

- The square-waved Dir has less stringent rise and fall time requirements when applied to the DCO input flip-flops. With sufficient delay of the $Shift$ signal, the internal flip-flops that control the varactor arrays within the DCO will always be switch on or off. This makes the circuit work reliably under different temperature and power supply conditions.

The correct operation of the proposed PFD is illustrated in Figure 3.5. This figures shows how the glitch elimination and cycle elimination works and it also shows how the signal Dir changes value when applying a frequency step on the divider output signal $Fdiv$.

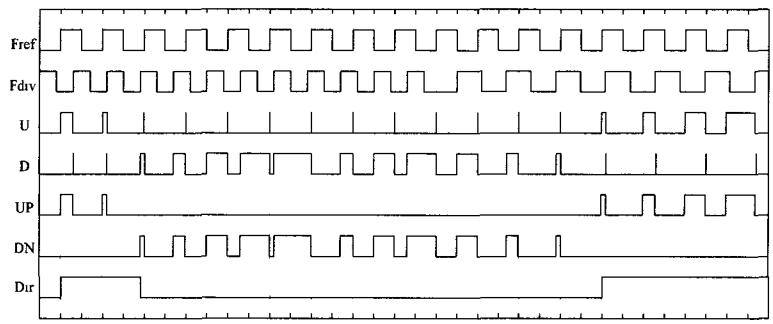

Figure 3.5: The Waveform of the Proposed PFD

3.2 DCO With the Integrated Low Pass Filter

The digital interface of the DCO is very simple by combining the low pass filter functionality inside the DCO itself. The DCO in the design takes only the signal *Dir* from the PFD and a signal *Shift* to clock the internal shift register. The output of the DCO is the synthesized output clock *Fout*.

Combining the low pass filter not only makes the interface simple and clean but also makes the test of the DCO in the open-looped mode easy. By inserting two MUXes between the PFD and the DCO, the PLL loop can be opened or closed. When in the closed mode, the *Dir* and *Shift* signals are connected to the PFD output *Dir* and the divider output *Fout*. When in the opened mode, the *Dir* and *Shift* signals are connected to two external control signals and the DCO's linearity and step size can then be characterized easily.

Conceptually, the DCO can be divided into the bidirectional shifter register controlled integral varactor array, the programmable proportional varactor array, and the DCO core.

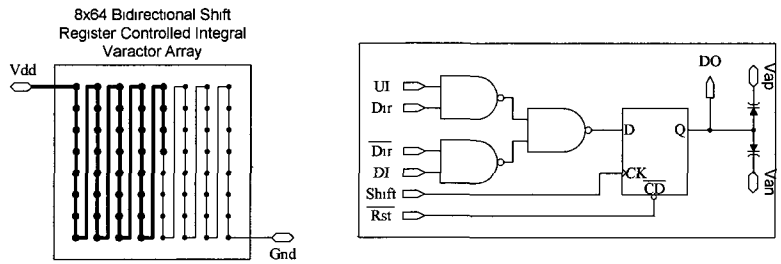

Figure 3.6: The Bidirectional Shift Register Controlled Integral Varactor Array

3.2.1 The Bidirectional Shift Register Controlled Integral Varactor Array

Without the capacitor counterpart in the analog PLL to serve as the memory element to remember the previous state, the integral path in this all digital PLL design is embedded in the DCO itself and is implemented by a long bidirectional shift register shown in Figure 3.6. The *Dir* signal determines the shift direction. The *Shift* signal serves as the clock input to the shift register. Signals *UI* and *DI DO* are used to concatenate the register cells together to form a serial array. For easy and regular layout purposes, each register cell also contains a pair of differential varactor pair which shares two common signals *Vap* and *Van* of the resonant tank from the DCO core.

The operation of the bidirectional shifter register controlled integral varactor array is similar to a thermometer. When the divider output signal *Fdiv* is ahead of the reference signal *Fref*, the *Dir* is low and the register shift one more '1' into the register and slows down the DCO output signal *Fout*; when the divider output *Fdiv* is behind the reference signal *Fref*, the *Dir* is high and the register shift one more '0' into the register and speed up the DCO output signal *Fout*.

Given a fixed varactor step size and the inductor value in the DCO core, the total number of the shift register cells determines the PLL's locking range. More cells results in a wide locking range but lower output frequency. The size of the varactor in each cell determines the minimal resolution of integral path in the DCO. Generally speaking, given the same proportional path to integral path coefficient ratio Kp/Ki, the finer the resolution in the integral path, the smaller the deterministic jitter caused by the proportional path.

3.2.2 The Programmable Proportional Varactor Array

The proportional path in the DCO as shown in Figure 3.7 is simpler than the integral path. Each cell in the proportional path consists of an AND gate and a differential varactor. The signal *Dir* is directly taken from the PFD output DIR. The programmable proportional coefficient Kp[7:0] is binary coded in the low 4 bits and unary coded in the upper 4 bits. The minimal coefficient is 0 and the maximal coefficient is 79 when all the bits of Kp[7:0] are all 0. A better choice would be to have all the bits binary coded to simplify the software programming interface and to allow for bigger proportional path coefficients.

3.2.3 The LC-Based DCO Core

The DCO core shown in Figure 3.8 is made up of two inverters cross connected with an spiral inductor. The bidirectional shift register controlled integral varactor array and the programmable proportional path varactor array provide the capacitive load in the resonant tank. A separate power supply $Vdco$ is provided for the DCO core

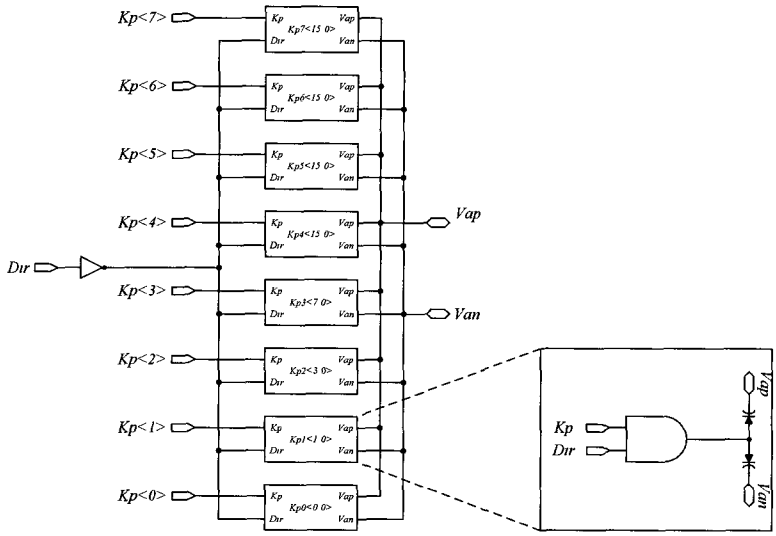

Figure 3.7: The Proposed Programmable Proportional Varactor Array

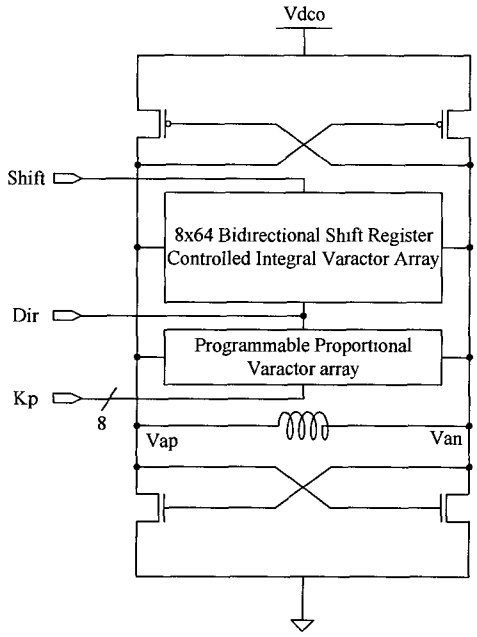

Figure 3.8: The Proposed DCO Core

to control the output center frequency and to control the output swing amplitude to match the high speed common mode logic (CML) logic in the divider stage described

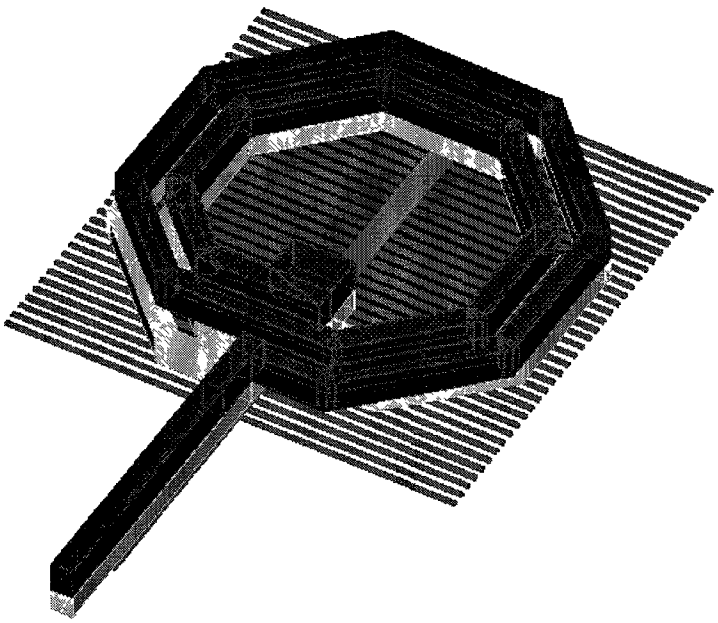

Figure 3.9: EMSS Model for the Inductor Used in the DCO

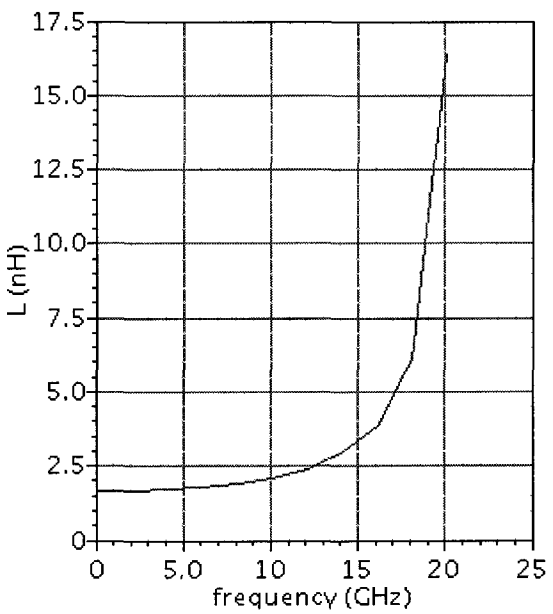

Figure 3.10: Inductor's Inductance by EMSS Simulation

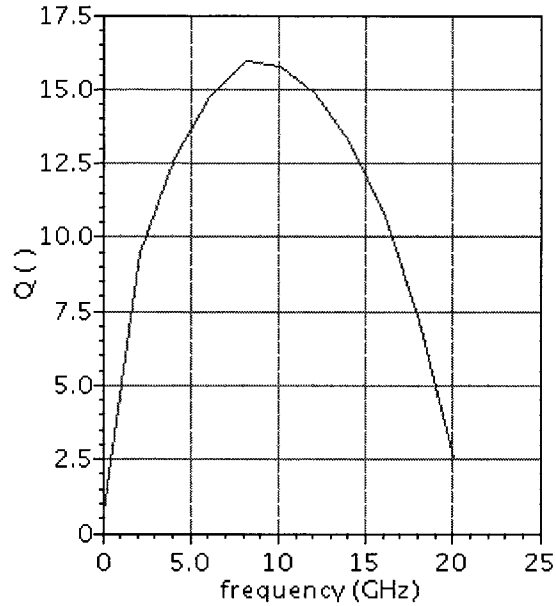

Figure 3.11: Inductor's Q by EMSS Simulation

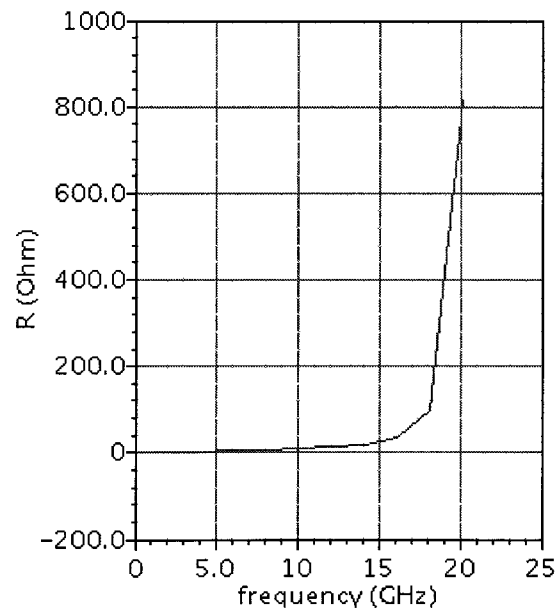

Figure 3.12: Inductor's R by EMSS Simulation

in the next section.

The inductor used in the DCO shown in Figure 3.9 is made up of top two layer metals with the bottom metal layer as the ground shield to reduce the substrate loss caused by the capacitive coupling. The result of the electromagnetic simulation software (EMSS) simulation of the inductor is shown in Figures 3.10, 3.11, and 3.12. The simulated inductance, quality factor Q, and the parasitic resistance are 1.66nH, 7.5, and 1.88Ω respectively at 1.35GHz.

3.3 MMD With Wide Divide Ratio Range

The MMD made up of n traditional divide-by-2/3 cells have the divide ratio range of 2^{n-1} to $2^n - 1$. For the applications that do not require wide divide ratio range, this type of MMD is sufficient. For a research project like this one or the frequency synthesizer design a wide divide ratio range will give the designer more freedom to explore various effects related to the divide ratio and find a best ratio range for a particular PLL architecture.

The modified divide-by-2/3 cell used in this design is illustrated in Figure 3.13. One OR gate and one NAND gate are added to the traditional divide-by-2/3 cell. The OR gate is used to sense if any higher bit $R[m+1:n]$ than the current bit $R[m]$ in the ratio word R has a '1'. If there is a '1' in the higher bit the current output $Tout[m]$ will be '1'. When cascaded together, the signal $Tin[m]$ with the next modulus signal $Mout[m-1]$ will activate or de-activate the next divide-by-2/3 cell, hence dynamically adjusting the effective length of the MMD depending on the input ratio R. This results in the wide divide ratio range 2 to $2^n - 1$. The extended range is

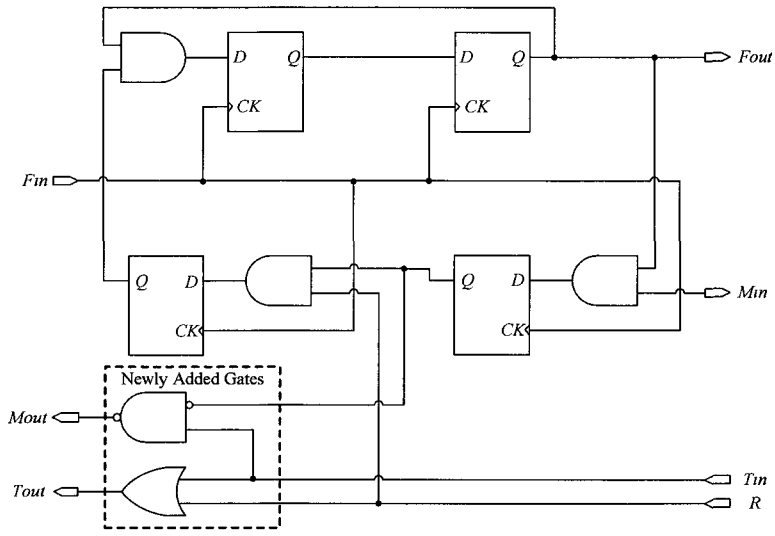

Figure 3.13: The Modified Divide-by-2/3 Cell

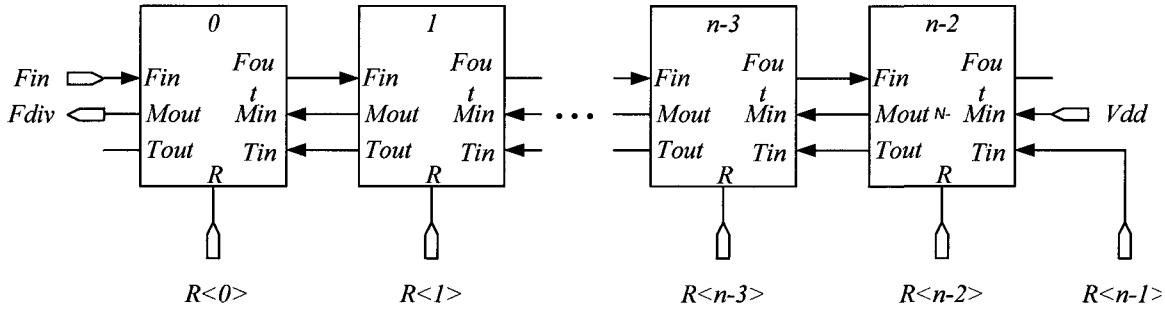

Figure 3.14: The Multi-Modulus Divider

achieved with only two additional gates per divide-by-2/3 cell. The cost in the layout area and power consumption is negligible.

The modified divide-by-2/3 cells are cascaded together to form a MMD as shown in Figure 3.14. For the divide range from 2 to $2^n - 1$, the required number of divide-by-2/3 cells is n-1.

Chapter 4

System Level Simulation of the ADPLL by SystemVerilog

Event-driven simulation technique of an ADPLL by *SystemVerilog* is presented in this chapter. It uses the *Mersenne-Twister* random number to generate the *uniformly distributed white noise* and then uses *Box-Muller* algorithm to generate the *normally distributed white noise*. The extremely simple *Stochastic Voss-McCartney* algorithm is used to generate the *pink noise* so that the $1/f$ phase noise effect can be modeled easily. Since the event-driven simulation is extremely fast compared to the circuit level simulation, it allows circuit designers to explore different ADPLL architectures at the early stage without going through the time-consuming circuit level simulation. Pure *SystemVerilog* implementation also makes it possible to simulate the phase noise effect of the ADPLL efficiently in a system-on-chip (SoC) dominated by the digital components.

4.1 Event-Driven Simulation by SystemVerilog

It is a well known issue that the PLL simulation by a Spice-like simulator is time consuming. In order to predict the noise in a circuit, the Spice simulator needs a quiescent operating point which is not always the case in a PLL circuit [29]. This means that it is impractical to use a Spice simulator to explore the PLL architecture due to its simulation speed and sometimes it is even impossible due to its inability to simulate the noise in a complex system. In order to solve the speed problem of the Spice simulation, phase-domain simulation technique was used. However, the phase domain simulation is only useful for predicting the loop dynamics and gives little intuitive insight into the circuit itself especially for the beginners. Moreover, the phase domain simulation can not be used to evaluate the phase noise performance which is the most important feature in a PLL system.

With the increased interest in the ADPLL, researchers began to simulate the AD-PLL by the event-driven technique. Reference [30] explores Matlab's event-driven programming technique to simulate the digital PLL at the system level. Because the Matlab simulation code uses the event-driven functions, the simulation speed is fast. It is easy to add the phase noise simulation in the Matlab code. However, it is not an easy task to integrate the Matlab simulation into a SoC simulation environment. Reference [29] is a comprehensive tutorial paper that uses the mixed signal simulator *VerilogA* to simulate a charge-pump-based PLL (CP-PLL). The best part of this reference is its jitter and phase noise simulation technique. Compared to the Matlab or other general purpose programming languages such as *C*, the hardware description languages *Verilog* or *VHDL* are intrinsically event-driven and programmers do not

need to worry about how event-driven functions work. Adding a $\Sigma\Delta$ modulator into the simulation code will make it possible to study the $\Sigma\Delta$ modulator phase noise effect. Reference [31] does this by a different mixed signal simulator *Spectre Verilog* from Cadence. With the advance of the CMOS process, more and more circuit components can be integrated into a single chip and most SoC designs have one or more PLL blocks. From the architectural point of view, it is desirable to use as many as possible digital parts in the PLL design. This results in the proliferation of the ADPLL in the SoC applications. Due to ADPLL's digital nature, it is very easy to incorporate the ADPLL simulation into the SoC simulation environment. It is a difficult task to effectively simulate the phase noise in a pure digital environment without slowing down the whole simulation because phase noise simulation requires complex noise generation algorithms. Reference [32] simulates the phase noise within the SoC environment with the pure digital VHDL language.

The key component in the phase noise simulation is the time domain noise generator. Once the noise in the DCO is generated correctly, the closed loop dynamics with the noise effect can be simulated. The most important noises in a DCO is the $-20dB/dec$ frequency modulated thermal noise, the $-10dB/dec$ flicker noise, and the flat white Gaussian noise. Noises generated in blocks that are not part of the PLL loop can be modeled as the additive white noise and they are not cumulative so they can be inferred into the DCO's output white noise.

In this chapter, SystemVerilog is used to model the ADPLL building blocks propose in Chapter 3 since *SystemVerilog*'s direct programmin interface (DPI) makes it extremely easy to call external noise generation functions written in C. The highly respected *Mersenne-Twister* random number generator [33] is used to generate the

white noise with uniform distribution. The white noise with Gaussian distribution is then generated from this uniform white noise by the *Box-Muller* algorithm. The 1/f noise is generated by a simple *Stochastic Voss-McCartney* algorithm [34] from the music DSP community. The higher order noises are generated by integration over the previously mentioned lower order noise sources.

4.1.1 PDF Simulation

The PFD used in this simulation is shown in Listing 4.1.

Listing 4.1: PFD Simulation Model in SystemVerilog

```
1  `include "adpll.vh"
2  module pfd (fref, fdiv, up, dn, rst_n);
3    output up, dn;
4    input fref, fdiv, rst_n;
5    reg up, dn;
6    wire pfd_rst;
7
8    // fref faster => up
9    always @(posedge fref or posedge pfd_rst) begin
10     if (pfd_rst) up <= 0; else up <= 1;
11   end
12
13   // fdiv faster => dn
14   always @(posedge fdiv or posedge pfd_rst) begin
15     if (pfd_rst) dn <= 0; else dn <= 1;
16   end
17
18   assign pfd_rst = ~rst_n | (up & dn);
19 endmodule
```

4.1.2 Divider Simulation

The frequency divider in a real circuit implementation may be the dual modulus divider. However, for the behavioral simulation purpose, we use a counter in the

simulation for simplicity and speed purpose. It is also used as a verification module for the multi-modulus divider we used in FPGA and CMOS implementation in the following chapters. Initially, the counter is reset to 0. Whenever the counter is greater than or equal to the programmed divider ratio, the output is flipped and the the counter is reset to 0 again. Listing 4.2 shows the frequency divider *SystemVerilog* code.

Listing 4.2: Divider Simulation Model in *SystemVerilog*

```
1  `include "adpll.vh"
2  module fdiv (clk, M, fdiv, reset);
3    parameter div_width = 8;
4    input clk, reset;
5    input [div_width-1:0] M;
6    output fdiv;
7    wire fdiv;
8    reg q;
9    integer i;
10
11   always @(negedge reset) begin
12     i = 0;
13     q = 0;
14   end
15
16   always @(clk) begin
17     if (~reset) begin
18       i = i + 1;
19       if (i >= M) begin
20         q = ~q;
21         i = 0;
22       end
23     end
24   end
25
26   assign fdiv = q & ~reset;
27 endmodule
```

4.1.3 DCO With the Built-in LPF Simulation

Compared to most of the traditional PLL configurations, the ADPLL does not have an explicit LPF. The LPF function in this ADPLL is tightly integrated inside the DCO. The DCO frequency is controlled by an array of switched capacitors which can also be used to perform addition operations. Integrating the LPF function inside the DCO has the benefit of eliminating the complex adders needed by the traditional LPF module.

If two capacitors are connected in parallel, the total capacitance is simply the addition of two individual capacitances. This property makes it possible to use the varactor array itself to perform the addition operation. The purpose of the multiplier is to control the ratio of the proportional coefficient and the integral coefficient. If we use extra control signals to control how many varactor units are used for the proportional path and how many varactor units are used for the integral path, we do not need complex multipliers any more in the LPF.

Listing 4.3 shows the DCO *SystemVerilog* code. Only the up/down counter method is shown in this listing for simplicity purpose. The noise generation related code will be explained in section 4.2.

Listing 4.3: DCO and the Built-in LPF Simulation Model in *SystemVerilog*

```
1  `include "adpll.vh"
2  module bbdco (up, dn, fout, fref, fdiv);
3    input up, dn;
4    output fout;
5    reg fout;
6    input fref, fdiv;
7    reg [1:0] P;
8    integer I;
9    real Ctrl, period;
10   real jitter_stddev, flicker_stddev, wander_stddev,
       saunter_stddev;
```

```
11    real pjitter,       pflicker,       pwander,       psaunter;
12    initial begin
13      fout = 1'b1;
14      P = 0;
15      I = 0;
16      Ctrl = 0.0;
17      pjitter  = 0.0;
18      pflicker = 0.0;
19      pwander  = 0.0;
20      psaunter = 0.0;
21      jitter_stddev  = white2stddev(`f0dco,
            `white_pn);
22      flicker_stddev = pink2stddev (`f0dco, `pink_corner,
            `pink_pn);
23      wander_stddev  = brown2stddev(`f0dco, `brown_corner,
            `brown_pn);
24      saunter_stddev = red2stddev  (`f0dco, `red_corner,
            `red_pn);
25    end
26    always @(up or dn) begin : lpf
27      case ({up, dn})
28        2'b01: I = I - 1;
29        2'b10: I = I + 1;
30        default: ;
31      endcase
32      P[0] = up ^~ dn;
33      P[1] = up & ~dn;
34    end
35    always @(posedge fdiv) begin : retiming
36      Ctrl <= (`Kp * P) + (`Ki * I);
37    end
38    always begin
39      period = 1.0 / (`f0dco + `Kdco * Ctrl);
40      period = period + jitter (jitter_stddev,  pjitter);
41      period = period + flicker(flicker_stddev, pflicker);
42      period = period + wander (wander_stddev,  pwander);
43      period = period + saunter(saunter_stddev, psaunter);
44      if (period <= 0) $$ stop;
45      else begin
46        #(period/1e-12/2) fout = ~fout;
47        #(period/1e-12/2) fout = ~fout;
48      end
49    end
50  endmodule
```

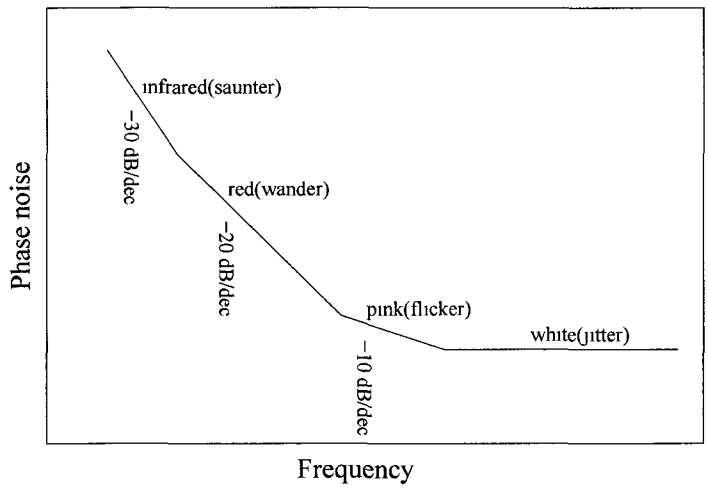

Figure 4.1: Phase Noise Spectral Density

4.2 Noise Simulation

The same noise terminologies from different publications can sometimes mean different types of noises. In order to have consistent noise terminologies, we use the terms white, pink, red, and infrared to designate the frequency domain phase noises with different slopes in this paper. White phase noise is flat; pink phase noise has $-10dB/dec$ slope; red phase noise has $-20dB/dec$ slop; infrared phase noise has $-30dB/dec$ slope. The corresponding time domain noise terms for the above phase noises are jitter, flicker, wander, and saunter, respectively, as shown in the parenthesis in Figure 4.1.

Please note that the term infrared phase noise and the saunter are newly introduced in this paper to refer to the integrated pink noise and the flicker noise in the frequency domain and in the time domain respectively.

Given the white phase noise $\mathcal{L}_w$ in dBc and the oscillator frequency f_0, the jitter noise σ_j can be calculated by the following equation [32]:

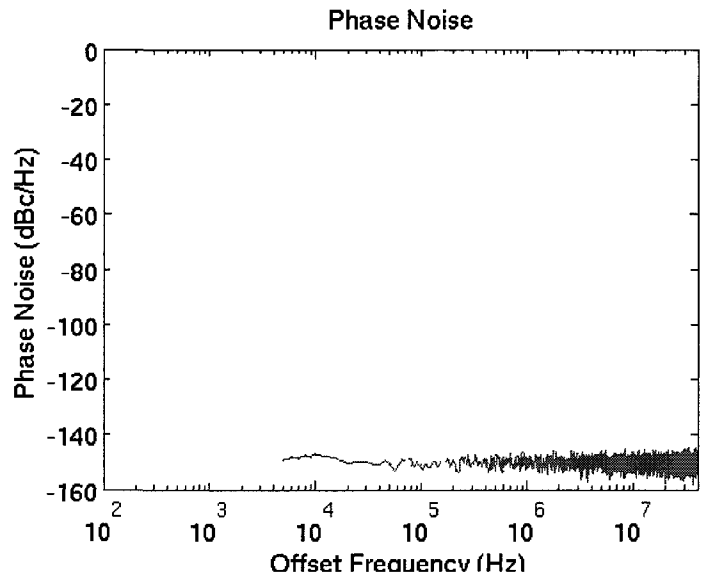

Figure 4.2: Generated White $-150dBc/Hz$ *Gaussian* Noise by *Box-Muller* Algorithm

$$\sigma_y = \frac{1}{2\pi}\sqrt{\frac{10^{\mathcal{L}_w/10}}{f_0}} \tag{4.1}$$

Figure 4.2 shows white $-150dBc/Hz$ *Gaussian* noise by the *Box-Muller* Algorithm.

Given the pink noise $\mathcal{L}_p$ in dBc and frequency offset Δf, the flicker noise σ_f can be calculated by the following empirical equation introduced in this paper:

$$\sigma_f = \frac{\Delta f}{f_0}\sqrt{\frac{10^{\mathcal{L}_p/10}}{2\pi f_0}} \tag{4.2}$$

Figure 4.3 shows the simulated $-10dB/dec$ pink noise generated by the *Voss-McCartney* algorithm.

Given the red noise $\mathcal{L}_r$ in dBc and frequency offset Δf, the wander noise σ_w can be calculated by the following equation [32]:

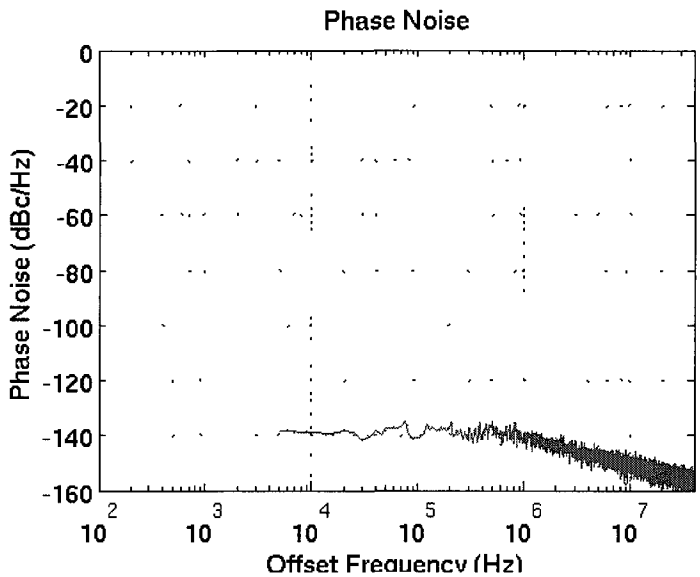

Figure 4.3: Generated $-140dBc/Hz@2MHz$ Pink Noise ($-10dB/dec$) by *Voss-McCartney* Algorithm

$$\sigma_w = \frac{\Delta f}{f_0}\sqrt{\frac{10^{\mathcal{L}_r/10}}{f_0}} \tag{4.3}$$

Figure 4.4 shows the simulated $-20dB/dec$ red noise generated by integration of the white noise.

Given the infrared noise $\mathcal{L}_i$ in dBc and frequency offset Δf, the saunter noise σ_s can be calculated by the following empirical equation introduced in this paper:

$$\sigma_s = \frac{\Delta f}{2\pi^2 f_0}\sqrt{\frac{10^{\mathcal{L}_i/10}}{2\pi f_0}} \tag{4.4}$$

Figure 4.5 shows the simulated $-30dB/dec$ infrared noise generated by integration of the pink noise.

Noise simulation in the behavioral simulation involves the generation of the white, pink, red, and the infrared noise. The simplest noise is the white noise. Theoretically,

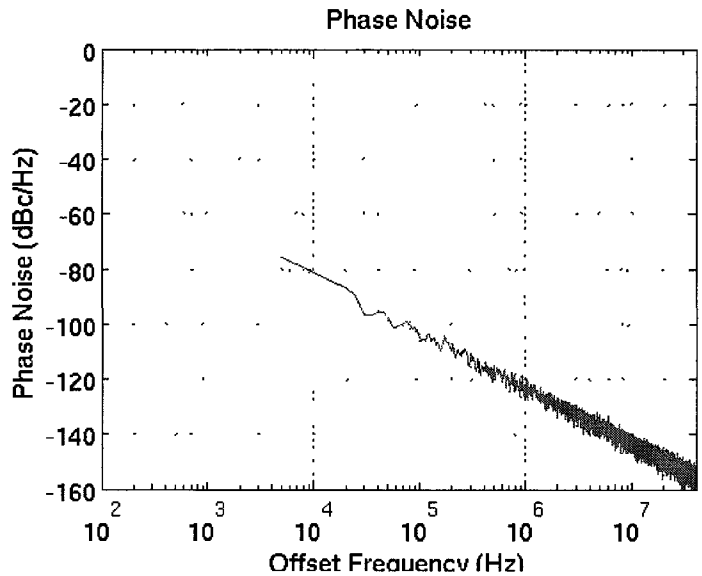

Figure 4.4: Generated $-123dBc/Hz@1MHz$ Red Noise ($-20dB/dec$) by Integration of the White Noise

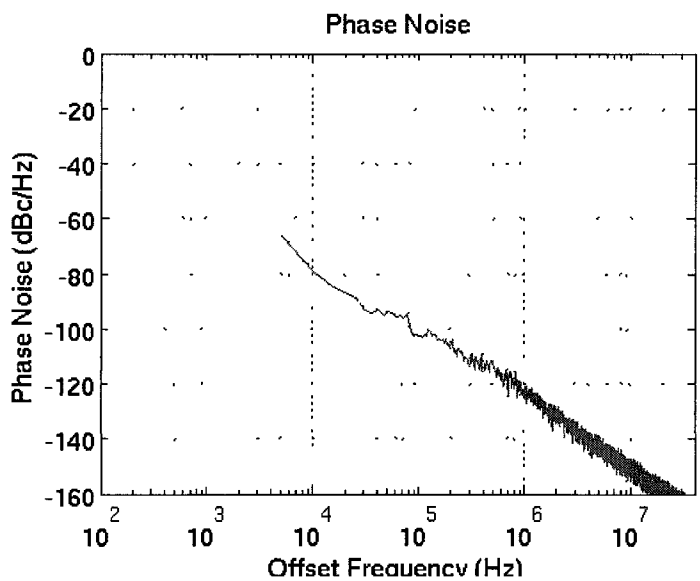

Figure 4.5: Generated $-80dBc/Hz@10KHz$ Infrared Noise ($-30dB/dec$) by Integration of the Pink Noise

the white noise can be generated by any uniform random number generator readily available from many software packages. However, a random uniform random number generator may not have the expected power spectral density value in the ADPLL simulation. The normal random number generator should be used in the simulation. In this paper, the normal random number generator is implemented by the *Box-Muller* algorithm and the uniform random number generator used by the *Box-Muller* algorithm is implemented by the highly respected *Mersenne-Twister* random number generator from [33].

Designers who are new to the noise simulation are often confused by the power spectral density and the random number probability distribution. Actually, these two properties do not have any direct relationship. Both the uniform random number generator and the normal random number generator can generate the time sequences that have flat spectral density. Noise with different power spectral density can be generated by integration or differentiation of the white noise. The integration and the differentiation serve the purpose of generating poles or zeros so that we can get the desired power spectral density graphs. This is called the filtering method or fitting method.

Like white noise, pink noise is also found to be universal in all kinds of applications such as electronics, economics, and music. However, unlike white noise that is the easiest one to generate, the flicker noise is the most difficult one to generate. It remains one of the most interesting topic of research in the art, science, and engineering fields. Reference [32] uses the filtering method to generate pink noise. The filtering method is straight forward but its running simulation efficiency is not the best. Reference [34] proposed an improved *Voss-McCartney* algorithm in the music DSP community to

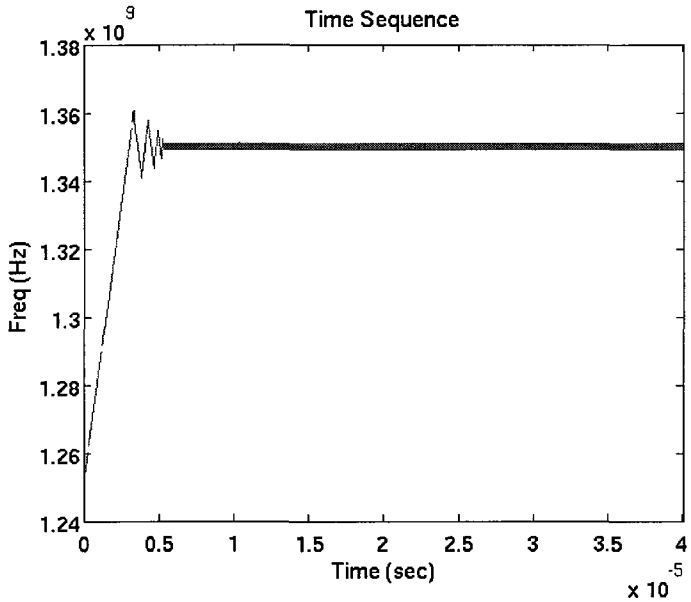

Figure 4.6: Step Response Without Noise

generate pink noise. This paper will use this improved *Voss-McCartney* algorithm in the ADPLL phase noise simulation.

The red and the infrared noise generators can be easily constructed with white noise and pink noise generators. The red noise generator is simply the integration of white noise and infrared noise generator is simply the integration of pink noise.

The noise generation functions are written in *C* language as the *SystemVerilog* DPI. DPI allows *SystemVerilog* code to call any user defined functions or operating system provided *C* libraries so it is extremely easy to integrate any external noise generators written in *C* into the simulation model written in register transfer language (RTL).

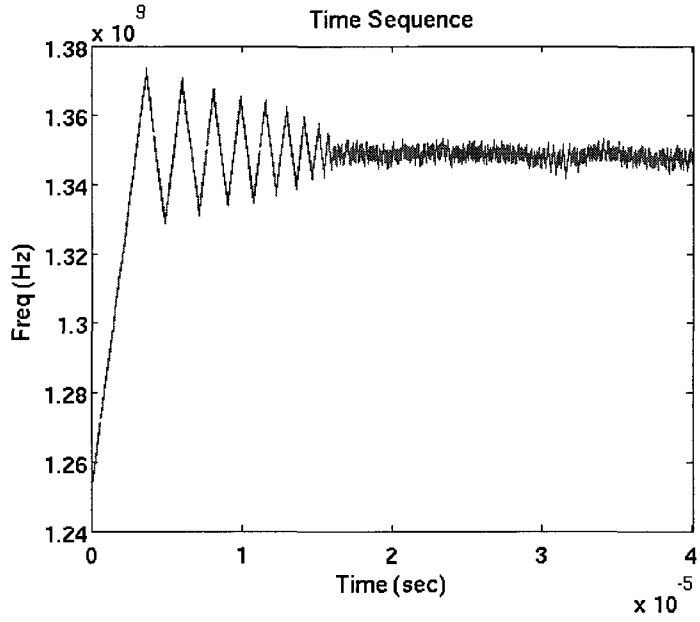

Figure 4.7: Step Response With Noise

4.3 Simulation Results

The *SystemVerilog* test bench writes the time domain periods into a file. After the simulation is done, a *Matlab* script is used to plot the time domain periods graph and to calculate the phase noise. The *Matlab* script is partially based on the script provided by [29]. The difference is that the *Matlab* script used by [29] does the noise integration by the *Matlab* while the noise is integrated by the PLL simulation itself. The noise integration is the transfer function of the DCO so it should be handled by the simulation. The phase noise is calculated by the *Welch* algorithm available in *Matlab* with the input period time sequence.

Figure 4.6 shows the time domain step response without any noise. The locking and the settling behavior can be clearly seen from this diagram. With the noise added in the simulation, the settling time is longer than that without the noise as show in

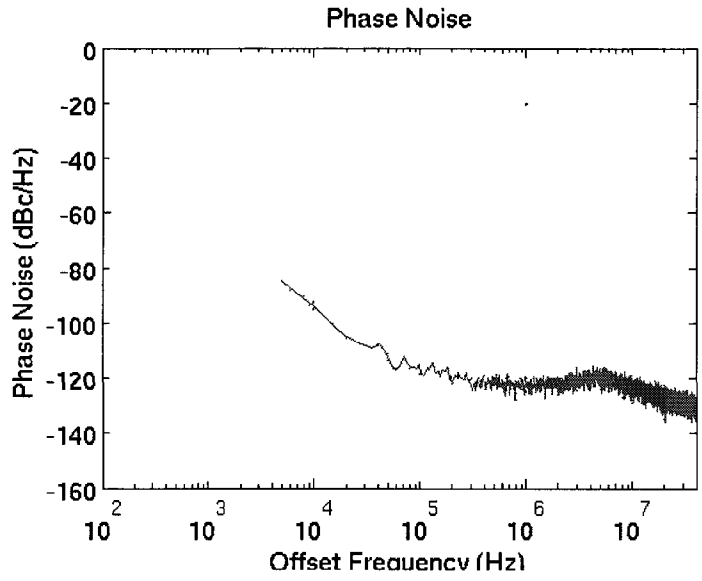

Figure 4.8: Simulated Phase noise

Figure 4.7. The time domain noise σ_t can be calculated from the period time sequence used to draw the step response diagram. However, this time domain noise increases with the the simulation time if the phase noise is not pure white.

Figure 4.8 shows the phase noise of the simulated ADPLL architecture. The phase noise values and shape are close to the calculated ones. For example, the white phase noise caused by peripheral circuits is expected to be $-150dBc/Hz$ and phase noise at $10MHz$ is expected to be $-120dBc/Hz + 3dBc = -117dBc/Hz$ mainly determined by the red noise corner and the infrared noise corner and the simulated result is $-116.6dBc/Hz$. We can see from this diagram that the ADPLL effectively suppresses the close-in phase noise. The loop bandwidth is estimated to be $30kHz$ from this diagram.

The program takes about 5 seconds to finish a 436 μs simulation without any noise and it takes 46 seconds to finish the same simulation with all four different noises on

a *Sun-Fire-V440* machine. This amounts to about $10\mu s$ simulation time per second.

4.4 Summary

Table 4.1 summaries different PLL simulation techniques described in the references. The simulation languages used in the references include *Matlab*, *VerilogA*, and *VHDL*. This paper chose *SystemVerilog* because the DPI interface provided by *SystemVerilog* makes it extremely simple to integrate the noise generators written in *C*. The noise performance is one of the most important parameters in any PLL design. Reference [30] does not include the noise simulation. Reference [29] and [31] only include the white noise and the red noise simulation. Reference [32] and this paper both include the $1/f$ noise simulation but with different algorithms. Reference [32] uses the IIR filter method to generate the $1/f$ noise while this paper uses the simple and efficient Voss-McCartney algorithm to generate the $1/f$ noise.

This paper also introduces a new infrared noise (saunter in the time domain) in the simulation to model the $-30dB/dec$ effect in addition to the white, pink, and red noises. Because not all the papers include all four noise sources and the simulation environments are different, the simulation speed values listed in Table 4.1 are for references only.

The pure digital language *SystemVerilog* is successfully used in this paper to simulate the step response and the locking behavior in the ADPLL. Since *SystemVerilog* is intrinsically event-driven, the simulation speed is fast enough to allow a SoC design to include the detailed ADPLL simulation.

The noise generation algorithms in *C* language are used in the *SystemVerilog*

Table 4.1: Comparison of ADPLL Behavioral Simulation

	Language	**Noise**	$1/f$ **Noise**	**Speed**
[29]	VerilogA	Yes	No	N/A
[30]	Matlab	No	No	N/A
[31]	VerilogA	Yes	No	$7\mu S/Sec$
[32]	VHDL	Yes	IIR Filters	$15\mu S/Sec$
This work	SystemVerilog	Yes	Voss-McCartney	$10\mu S/Sec$

model by DPI. The algorithms include the *Mersenne-Twister* random number generator, *Box-Muller* algorithm, and the *Voss-McCartney* algorithm. The noise simulation techniques can be used to investigate the phase noise behavior with any ADPLL system architecture at the very early design stage.

Chapter 5

Fully Synthesized ADPLL Prototype in FPGA

The purpose of this chapter is to provide a fast prototype of the proposed ADPLL by the FPGA which is much closer to the real CMOS implementation than the pure software simulation. The fully synthesized ADPLL in the FPGA can also be used separately in any applications that need both the ADPLL and other DSP modules to be implemented in the FPGA.

Analog circuit synthesis remains one of the chanllenging research area. Although its interfaces are all digital, the ADPLL works like a analog circuit. This makes it difficult to fully synthesize an ADPLL from *Verilog* or *VHDL*. The other reason that it is difficult to synthesize an ADPLL is that the ADPLL is basically a feedback system and the optimization algorithms used in *Verilog* or *VHDL* compiler will likely remove the feedback loop completely if no special constraints are used in the language. These constraints are vendor specific and are not standard *Verilog* or *VHDL* language

features.

Most of the ADPLL sub-circuits can be synthesized easily from the RTL language except for the DCO [35] [36][37] [38], [39] [40]. Most of the FPGA-based ADPLL uses the coordinate rotation digital computer (CORDIC) algorithm inside the FPGA and an digital-to-analog converter (DAC) to realize the DCO which limits the DCO output from a few KHz to tens of MHz.

This chapter describes a fully-synthesized ADPLL in the FPGA from the *Verilog* language including the DCO. The benefits of the digitally synthesized ADPLL are obvious. Firstly, the ADPLL can be offered as a soft IP module that can be integrated into SoC applications and can be simulated with the rest of the digital modules. Secondly, porting the ADPLL from one process to another requires much less time compared to porting the analog or mixed signal PLL. Lastly, the development cycle from functional specification to the tape-out and the lab characterization is short, since most of the tasks involved now can be automated by the software. The ADPLL synthesized on an FPGA chip can be served as the prototype to check if the circuit works and the circuits can be characterized immediately. This greatly improves the chance of first time tape-out success.

5.1 PFD

The tri-state PFD can be easily synthesized from *Verilog* as shown in Listing 5.1. The synthesized netlist schematic in the FPGA is shown in Figure 5.1.

Listing 5.1: Synthesizable PFD

```
1 module pfd (fref, fdiv, up, dn, rst_n);
2   output up, dn;
```

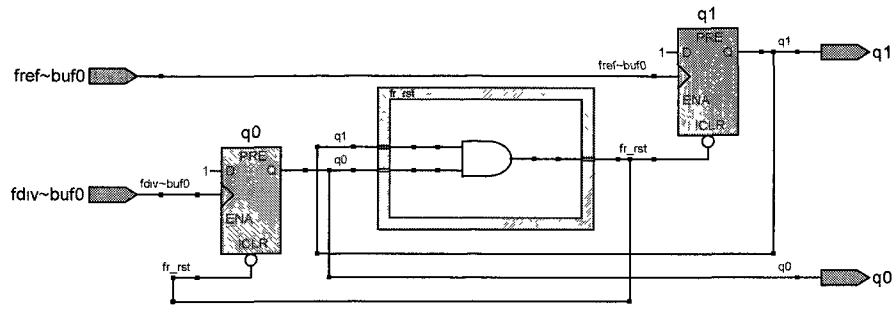

Figure 5.1: Synthesized PFD in PFGA

```
3    input fref, fdiv, rst_n;
4    wire fv_rst, fr_rst;
5    reg q0, q1;
6
7    assign fr_rst = (q0 & q1);
8    assign fv_rst = (q0 & q1);
9
10   always @(posedge fdiv or posedge fv_rst) begin
11     if (fv_rst) q0 <= 0; else q0 <= 1;
12   end
13
14   always @(posedge fref or posedge fr_rst) begin
15     if (fr_rst) q1 <= 0; else q1 <= 1;
16   end
17
18   assign up = q1;
19   assign dn = q0;
20 endmodule
```

5.2 LPF

The LPF *Verilog* code as shown in Listing 5.2 used in the FPGA is similar to the LPF *Verilog* code used in *SystemVerilog* simulation in [41]. The integral path is implemented as a bidirectional shift register that has a length of 288-bit and the proportional path is represented as 2-bit output. The integral and the proportional output

directly controls the DCO's tri-state load which tunes DCO's output frequency.

If the ADPLL is implemented in CMOS as a semi-customized circuit, the LPF module presented here will be tightly integrated in the DCO's varactor arrays such that the layout of the varactor array is compact and small due of regular wire routing and the interface of the DCO can be simply controlled by only two signals from the PFD. The DCO in the open loop mode can also be easily characterized with the built-in shift register by these two external control signals. Otherwise, it is impossible to control the DCO with hundreds or thousands of external signals.

Listing 5.2: Synthesizable LPF

```
1  module lpf(UP, DN, fsmp, rst_n, I, P);
2    input UP, DN, fsmp, rst_n;
3    output [`integral-1:0] I;
4    output [1:0] P;
5    reg [`integral-1:0] I;
6    reg [1:0] P;
7
8    always @(posedge fsmp)
9    begin
10     if (!rst_n) begin
11       I <= `integral'hf;
12       P <= 2'b00;
13     end
14     else begin
15       if (UP && ~DN)begin
16         I <= I >> 1;
17         I[`integral-1] <= 0;
18       end
19       else if (DN && ~UP) begin
20         I <= I << 1;
21         I[0] <= 1;
22       end
23     end
24     P[0] <= ~(UP ^~ DN);
25     P[1] <= ~(UP & ~DN);
26   end
27 endmodule
```

5.3 MMD

The MMD *Verilog* code that divides the input frequency by 2 to 2^n is shown in Listing 5.3. The code is partially based on the *Verilog* code in [23] that can only divide from 2^{n-1} to 2^n. Since the code is a simple sequential logic without any feedback, it can be easily synthesized as well.

A counter based divider can be synthesized easily from *Verilog* and is much easier to understand. However the synthesized netlist is bigger and slower than the compact MMD presented here. Remember that one of the purposes of the FPGA implementation is to validate the circuit design before a CMOS implementation is fabricated.

Listing 5.3: Synthesizable MMD With Divide Ratio of 2 to 2^n Partially Based on [23]

```
 1 module wlatch(Q, Q_N, D, clk, rst_n);
 2   input D, clk, rst_n;
 3   output Q, Q_N;
 4   reg Q;
 5   wire Q_N;
 6
 7   always @(clk or rst_n or D)
 8   begin
 9     if (~rst_n)
10       Q <= 1'b0;
11     else if (clk)
12       Q <= D;
13   end
14   assign Q_N = ~Q;
15 endmodule
16
17 module mmc(fin, fout, modin, modout, Tin, Tout, R, rst_n);
18   input fin, modin, Tin, R, rst_n;
19   output fout, modout, Tout;
20   wire a, b, c1, e, f, f1, g, h, h1, i, j, k;
21
22   assign modout = k;
23   assign fout   = j;
24
```

```
25    or  (Tout, Tin, R);
26    and (b,    i,   modin);
27    and (e,    k,   R);
28    and (g,    f1,  fout);
29    //        Q      Q_N D   clk  rst_n
30    wlatch Q1 (k, c1, b,  fin, rst_n);
31    wlatch P1 (f, f1, e, ~fin, rst_n);
32    wlatch Q2 (h, h1, g,  fin, rst_n);
33    wlatch P2 (i, j,  h, ~fin, rst_n);
34  endmodule
35
36  module mmd (fin, M, fdiv, rst_n);
37    input fin, rst_n;
38    input [`div_width-1:0] M;
39    output fdiv;
40    wire fdiv;
41
42    mmc MMC0 (fin, f0, mod1, mod0, T1,   T0, M[0], rst_n);
43    mmc MMC1 (f0,  f1, mod2, mod1, T2,   T1, M[1], rst_n);
44    mmc MMC2 (f1,  f2, mod3, mod2, T3,   T2, M[2], rst_n);
45    mmc MMC3 (f2,  f3, mod4, mod3, T4,   T3, M[3], rst_n);
46    mmc MMC4 (f3,  f4, mod5, mod4, T5,   T4, M[4], rst_n);
47    mmc MMC5 (f4,  f5, mod6, mod5, T6,   T5, M[5], rst_n);
48    mmc MMC6 (f5,  f6, mod7, mod6, T7,   T6, M[6], rst_n);
49    mmc MMC7 (f6,  f7, 1'b1, mod7, 1'b0, T7, M[7], rst_n);
50
51    assign fdiv = ~mod1;
52  endmodule
```

5.4 DCO

The most challenging task in the *Verilog* synthesis in the FPGA is the DCO since there is no synthesizable inductors or varactors. All the published FPGA implementations use either the CORDIC algorithm plus a DAC or a long ring oscillator whose stages can be switched on or off. Those implementation can only run at a few KHz or tens of MHz. Furthermore, those implementations have long delays in the close loop which are detrimental to the stability of the loop.

The DCO implemented in this thesis uses a 3-stage ring oscillator whose 3 output nodes O[2:0] drive a serials of tri-state loads Tp[80:0] and Ti[280:0] which are controled by the LPF shown in Listing 5.2. Compared to other ring oscillator-based DCOs which consist of an array of switchable inverters or CORDIC-based direct digital systhesis (DDS) system, the DCO used in Listing 5.4 runs over 500MHz in an entry level FPGA. It should be noted that the synthesized DCO on the FPGA can run above 1 GHz if it is synthesized into a single logical cell. However, this type of ring oscillator is not tunable since the tri-state load can not be directly connected to the ring oscillator inside a single logical cell.

Since the *Verilog* compiler will optimize out the entire ring loop, you need to tell the compiler not to do so by inserting the (*keep*) directive on line 8 of Listing 5.4.

The long and repetive tri-state loads are automatically generated by a *C* program into *varactor.v* which is included in Listing 5.5 but logically belongs here. If it is included here, the compiler optimization algorithm will transform it into something that is not what is needed originally.

The synthesized netlist schematic of the 3-stage ring oscillator is shown in Figure 5.2.

Listing 5.4: Synthesizable DCO

```
1  module dco(O);
2    (*keep*)output [2:0] O;
3    assign O[0] = ~O[2];
4    assign O[1] = ~O[0];
5    assign O[2] = ~O[1];
6  endmodule
7
8    // varactor.v
9    // Proportional path tunable tri-state load
10   assign Tp[0] = (SW[9] & P[0]) ? O[0] : 1'bZ;
11   assign Tp[1] = (SW[9] & P[1]) ? O[1] : 1'bZ;
```

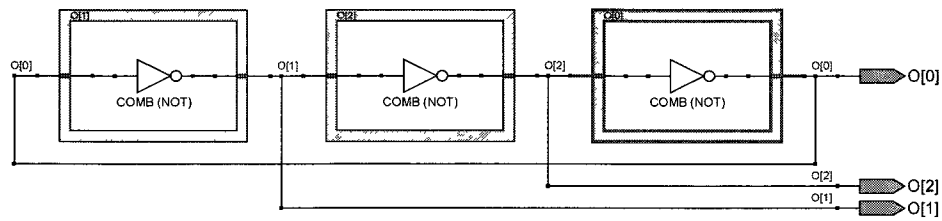

Figure 5.2: Synthesized Ring Oscillator Based DCO in PFGA

```
12    assign Tp[2] = (SW[9] & P[1]) ? O[2] : 1'bZ;
13    ...
14    assign Tp[78] = (SW[17] & P[0]) ? O[0] : 1'bZ;
15    assign Tp[79] = (SW[17] & P[1]) ? O[1] : 1'bZ;
16    assign Tp[80] = (SW[17] & P[1]) ? O[2] : 1'bZ;
17
18    // Integral path tunable tri-state load
19    assign Ti[0] = I[0] ? O[0] : 1'bZ;
20    assign Ti[1] = I[1] ? O[1] : 1'bZ;
21    assign Ti[2] = I[2] ? O[2] : 1'bZ;
22    ...
23    assign Ti[285] = I[285] ? O[0] : 1'bZ;
24    assign Ti[286] = I[286] ? O[1] : 1'bZ;
25    assign Ti[287] = I[287] ? O[2] : 1'bZ;
```

5.5 ADPLL

The complete ADPLL *Verilog* code is captured in the top level Listing 5.5. Another layer of code used for the pin assignment, key control, and LED display is omitted. The integral and proportional coefficients is controlled by the external keys on the *DE2* board. The LEDs are used to indicate the dynamic shift register value when you change the coefficients or the reference clock frequency.

Listing 5.5: Synthesizable ADPLL

```
1  `define proportional 81
2  `define integral 288
```

```
3  `define div_width   8
4  module adpll(CLOCK_50, SW, LEDR,
5             fref, fdiv, fdco, UP, DN, Tp, Ti, rst_n);
6    input CLOCK_50;
7    input [17:0] SW;
8    input rst_n;
9    output [17:0] LEDR;
10   (*keep*)output fref, fdiv, fdco, UP, DN;
11   output [`proportional-1:0] Tp;
12   output [`integral-1:0] Ti;
13
14   /* DCO output buffer */
15   (*keep*) wire N1;
16   (*keep*) wire N2;
17   (*keep*) wire N3;
18   assign N1 = ~O[0];
19   assign N2 = ~N1;
20   assign N3 = ~N2;
21   assign fdco = ~N3;
22
23   /* Internal wires */
24   wire UP, DN;
25   wire [1:0] P;
26   wire [`integral-1:0] I;
27   wire [2:0] O;
28
29   assign fref = f1M;
30   mmd FREF1 (.fin(CLOCK_50),    .R(`div_width'D5),
31              .fout(f10M),       .rst_n(rst_n));
32   mmd FREF2 (.fin(f10M),        .R(`div_width'D10),
33              .fout(f1M),        .rst_n(rst_n));
34   pfd PFD   (.fref(fref),       .fdiv(fdiv),
35              .up(UP), .dn(DN), .rst_n(rst_n));
36   lpf LPF   (.UP(UP), .DN(DN), .fsmp(fdiv),
37              .rst_n(rst_n),     .I(I), .P(P));
38   dco DCO   (O);
39 `include "varactor.v"
40   fdivby2  FPS0  (.fin(O[0]),   .fout(Fps));
41   mmd      FDIV  (.fin(Fps),    .R(SW[`div_width-1:0]),
42                   .fout(fdiv), .rst_n(rst_n));
43 endmodule
```

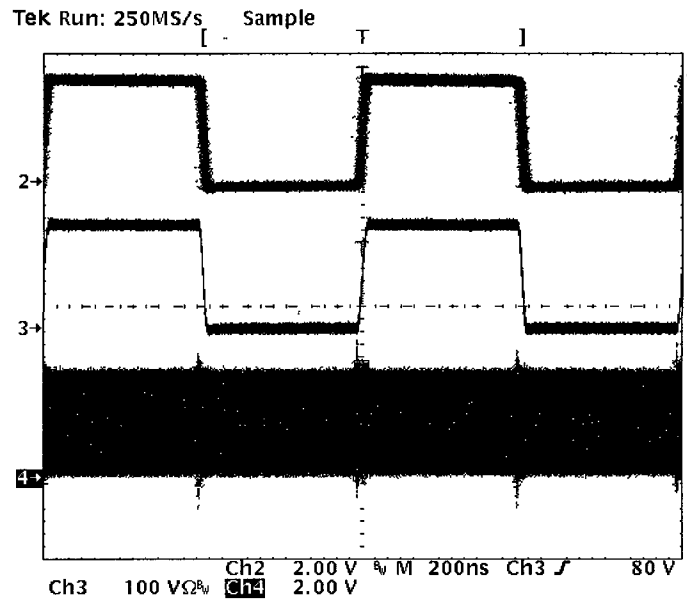

Figure 5.3: Waveforms of Reference Clock, Divider Ouput, and DCO Output

5.6 Test Results of the Synthesized ADPLL in FPGA

This ADPLL is synthesized into *Cyclone II EP2C35F672C6* on an Altera's *DE2* board. Since the tri-state gates are limited by the device, the number of the shift register cells are short and the proportional path coefficient is small in this implementation.

Figure 5.3 shows the waveforms of the input reference clock, the divided down clock, DCO output signals. The real time monitoring of these signals clearly shows that the output is locked to the input reference clock.

Figure 5.4 shows the measured phase noise graph. The loop bandwidth is 100kHz measured from this graph. The phase noise is $-98.33dBc/Hz@1MHz$ with the carrier frequency of $572MHz$. These results prove that the synthesized ADPLL is feasible.

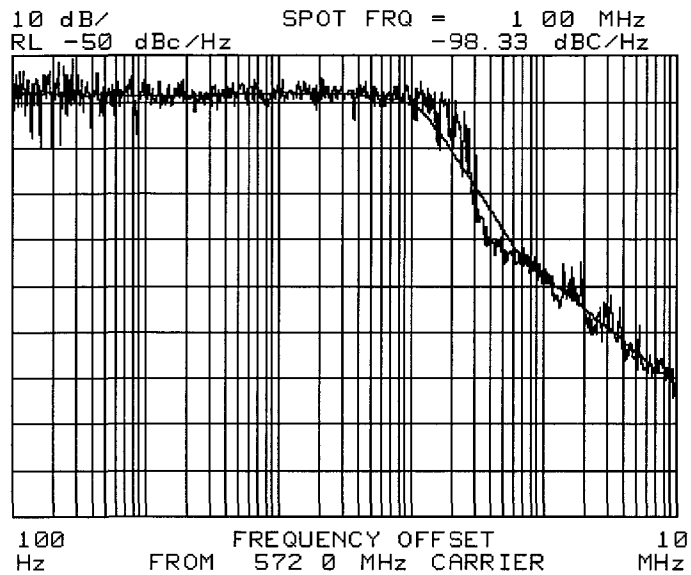

Figure 5.4: Measured Phase Noise of ADPLL implemented in FPGA

Table 5.1: Comparison of ADPLLs in FPGA

	[35]	[38]	[39]	[40]	This Work
Synthesizable	Yes	Yes	Yes	Yes	Yes
DCO type	Ring	CORDIC	CORDIC	CORDIC	3-stage Ring
AD/DA needed	No	Yes	Yes	Yes	No
f_{out}	$40MHz$	$117.4KHz$	$16.27KHz$	$60MHz$	$572MHz$
$\mathcal{L}^a$	N/A	N/A	N/A	N/A	$-98.33dBc/Hz$

[a]: Phase noise in $dBc/Hz@1MHz$

5.7 Summary

The comparison of the ADPLL FPGA implementations is shown in Table 5.1. It is clear that that the work presented in this paper is superior in that it runs a magnitude

faster than all the FPGA other implementations published known to the author due to the fact that the ring oscillator used in this work has only 3 stages. It is expected that with a newer and faster FPGA, the output frequency can run at a few GHz. The phase noise result is only available from this work and it is even better than some of the CMOS implementations. The uniqueness of the synthesized ADPLL presented in this paper is that the DCO is fully synthesized as a 3-stage ring oscillator tuned by an array of tri-state gates which is controlled by the bidirectional shift register. The bidirectional shift register serves as the integral path of the LPF.

Chapter 6

ADPLL in $0.13\mu m$ CMOS and Performance Evaluation

The test and measurement play a critical role in evaluating the performance of the ADPLL. Months of design effort will not be rewarded without any careful test preparation and proper setup. The input/output (IO) control circuits and software design is one of the major preparation tasks before the fabricated chips arrive. Test equipment user's manuals and operation theories need to be familiarized to save the precious alloted testing time.

The loose die micro-photo of the fabricated ADPLL chip is taken by a microscope as shown in Figure 6.1 with a layout annotation below it. The inductor implemented by the top level metals and the pads can be clearly seen in the die photo. All the squares seen in the die photo are top three level metal fills. All the low level metals and the transistor level structures are hidden below the top level metals.

The chip individual modules layout sizes are summarized in Table 6.1. It is clear

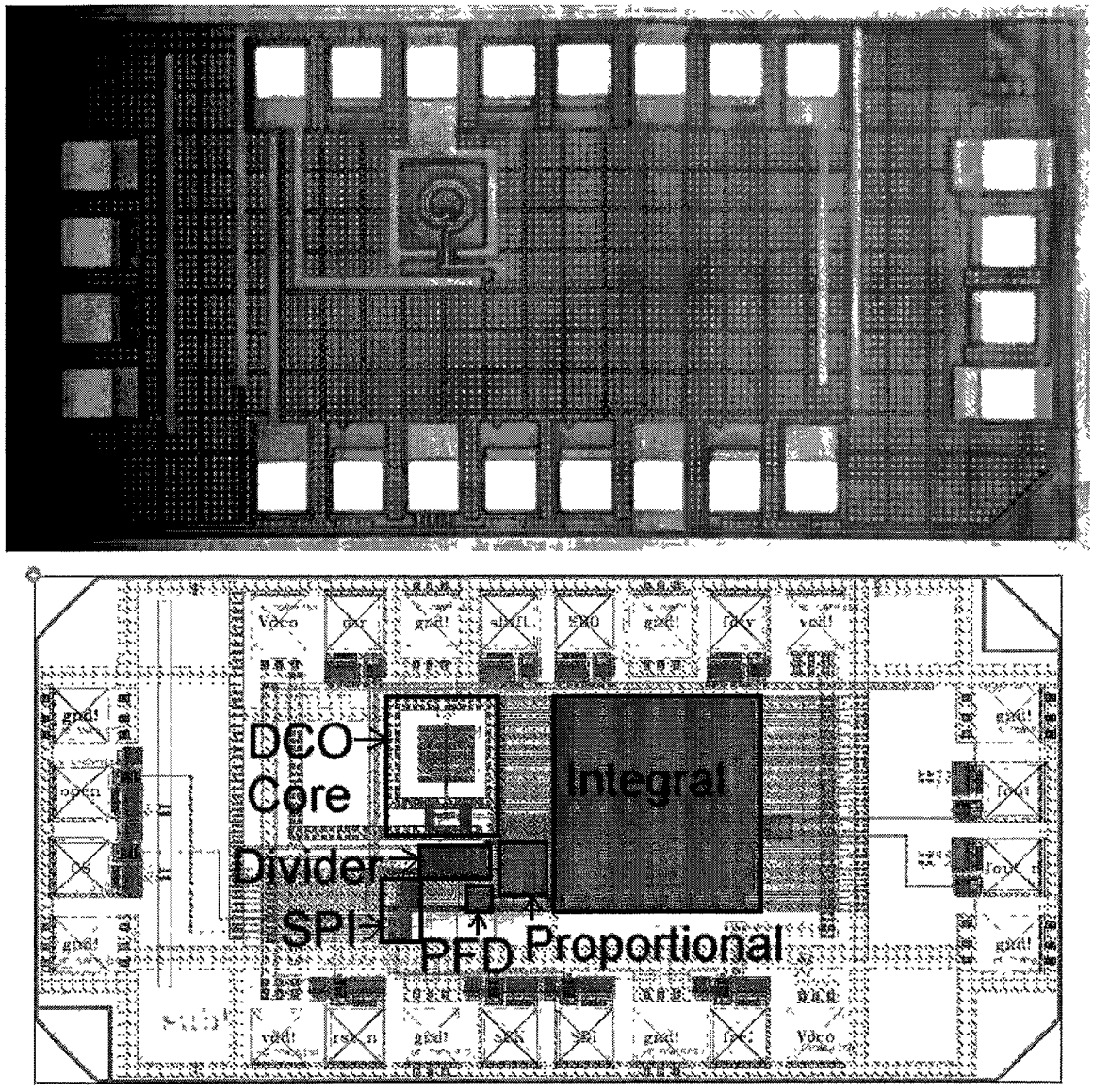

Figure 6.1: Die Micrograph With Layout Annotation

from this table that this test chip is pads limited. A single pad takes 108x108mum^2 and there are 24 in total. The biggest active module in this design is the integral varactor array which accounts for 8% of the total area. The electrostatic discharge (ESD) circuits are important to meet the reliability design rules and it accounts for 4.32% of the total area. The inductor used in this design is relatively small and only

Table 6.1: ADPLL Chip Layout Areas

	Size	**Percentage**
Chip Boundary	1000 μm x 2000 μm	100.00 %
Non-Active Area	-	71.61 %
PADs	108 μm x 108 μm x 24	14.00 %
Integral Varactor Array	400 μm x 400 μm	8.00 %
ESD Circuits	120 μm x 60 μm x 12	4.32 %
Inductor	110 μm x 110 μm	0.61 %
Proportional Varactor Array	100 μm x 100 μm	0.50 %
Divider	140 μm x 60 μm	0.42 %
SPI Controller	60 μm x 120 μm	0.36 %
DCO excluding Inductor	50 μm x 20 μm x2	0.10 %
PFD	40 μm x 40 μm	0.08 %

takes 0.61% of the total area. Although PFD only occupies 0.08% of the total area, it is a critical component since its phase resolution directly affects the phase noise performance of the DCO output.

The fabricated ADPLL chip is packaged in a CFP24 ceramic package and is mounted on the CFP24P test fixture provided by CMC Microsystems.

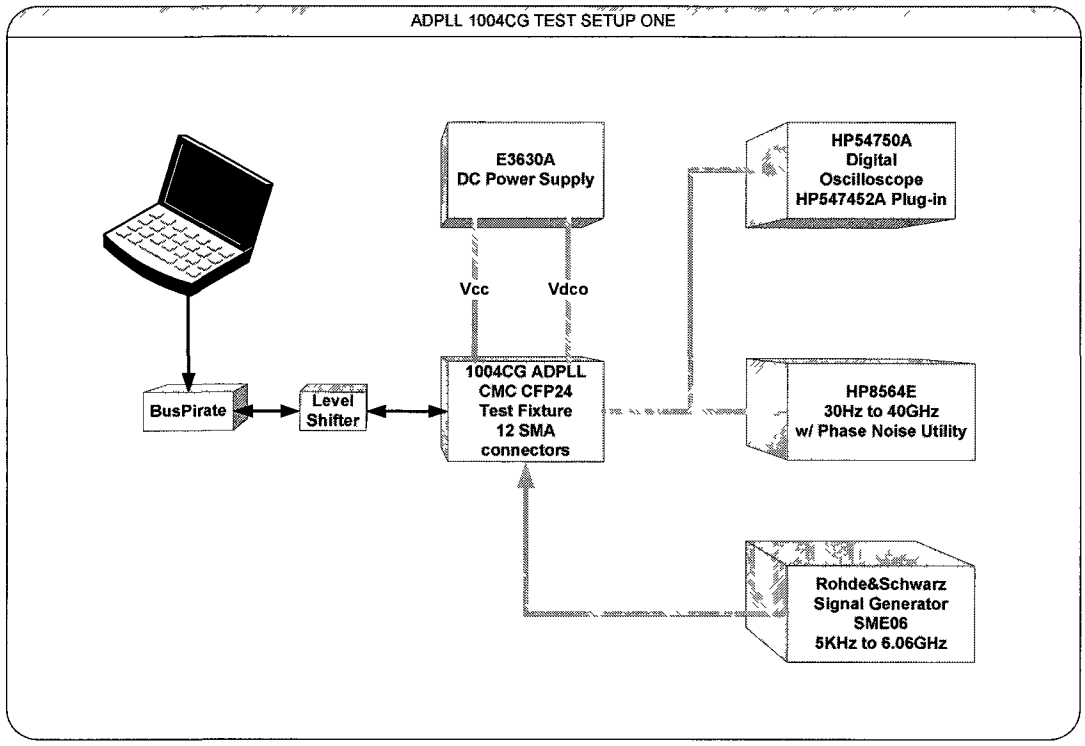

Figure 6.2: Test Environment Setup One

6.1 Test Environment Setup

There are two test environments used to characterize different aspects of the ADPLL chip with different test capability. The first test environment setup is shown in Figure 6.2.

The frequency spectrum and the phase noise is measured with *HP8564E*, which is a very good at measuring the frequency spectrum with a noise floor of -130dBC/Hz with some sharp discontinuities between different frequency bands. The phase noise software utility in *HP8564E* is rather slow when you need to measure the phase noise below 100KHz. The reference clock is generated by the *Rohde and Schwarz* signal

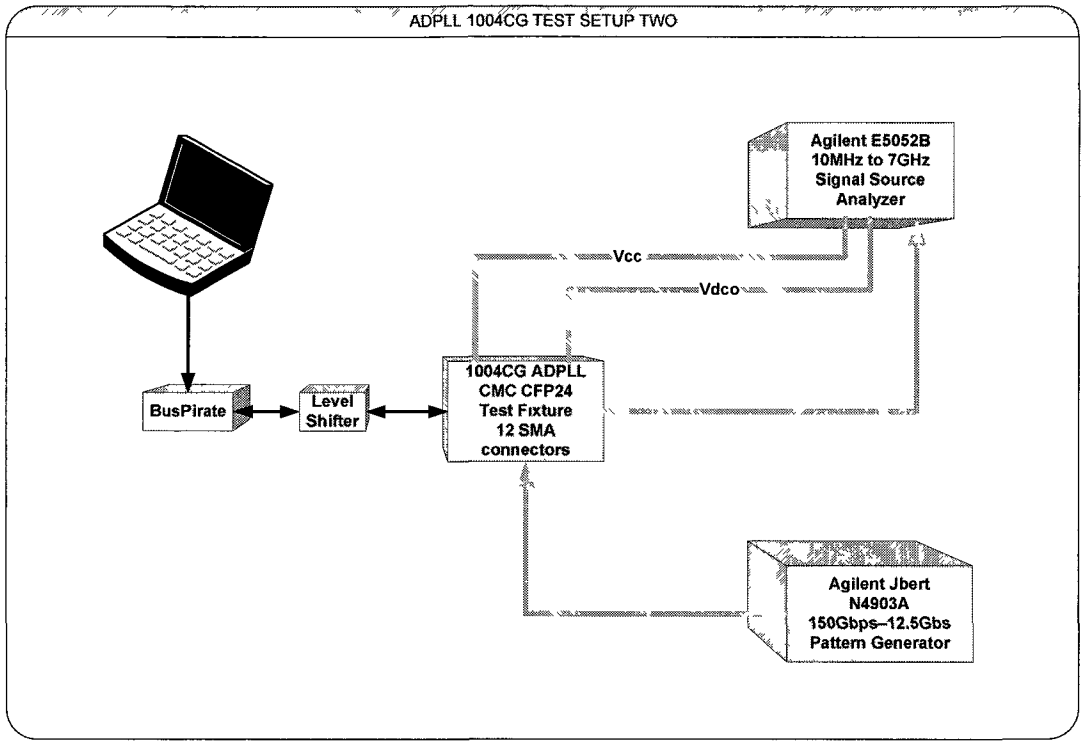

Figure 6.3: Test Environment Setup Two

generator *SME06* which is capable of generating signals from 5 KHz to 6.06 GHz. The system is handy to sweep the input frequency to measure the lock range. It can also generate signals with the phase modulation to study the behavior of how the ADPLL tracks the input reference noise. *HP54750a* digital oscilloscope with *HP547452A* plugin is used to monitor the synthesized high frequency clock waveform in the time domain.

The second test environment setup is shown in Figure 6.3. Equipped with modern CPU processors, *Agilent E5052B* is extremely fast and accurate to measure the phase noise. It can measure the phase noise above 100Hz within sub-second and above 1Hz within 15 seconds. The *E5052B* also has very low and continuous noise floor below

-160dBc/Hz. Another feature with E5052B is that it outputs two programmable very clean DC outputs as the ADPLL power supplies. The spectrum analyzer functionality provided by E5052B, however, is very weak since it only spans 15MHz such that you almost always need to manually locate the center frequency first before you can use the peak search and zoom keys. The sub clock output of the *Agilent JBert N4903A* is used as the signal generator. This reference clock has a noise floor of -155dBc/Hz above 100KHz offset frequency.

Since the ADPLL IO control circuits uses the standard 1.2V CMOS transistors to simplify the IO design, a voltage level shifter is needed when connecting the device under test to any external standard IO control and measurement equipment. In both setup environments, an AD325V20 breakout board with 8-BIT bidirectional voltage-level translator TXB0108 chip is used. Although the data sheet for TXB0108 specifies bidirectional translation between 1.2V to 5.5V, the test shows it can translate as low as 0.9V without any problem.

The open source hardware, *BusPirate*, created by Ian Lesnet, is used in the test environment setup to serve as the serial to peripheral interface (SPI) protocol bus master to control the ADPLL chip. The BusPirate hardware is capable of various serial bus protocols, such as the joint test action group (JTAG) protocol, the inter-integrated circuit (I2C) protocol, the 1-wire protocol, the raw wire bus protocol, and the SPI protocol. The interface from the *BusPirate* to the personal computer (PC) is through the prevalent universal serial bus (USB) bus.

You can use telnet to connect to the *BusPirate* to manipulate the serial bus and a few IO ports on the board. For any useful measurement, the software to automate the test is definitely desired to improve the test efficiency and to aid in data set

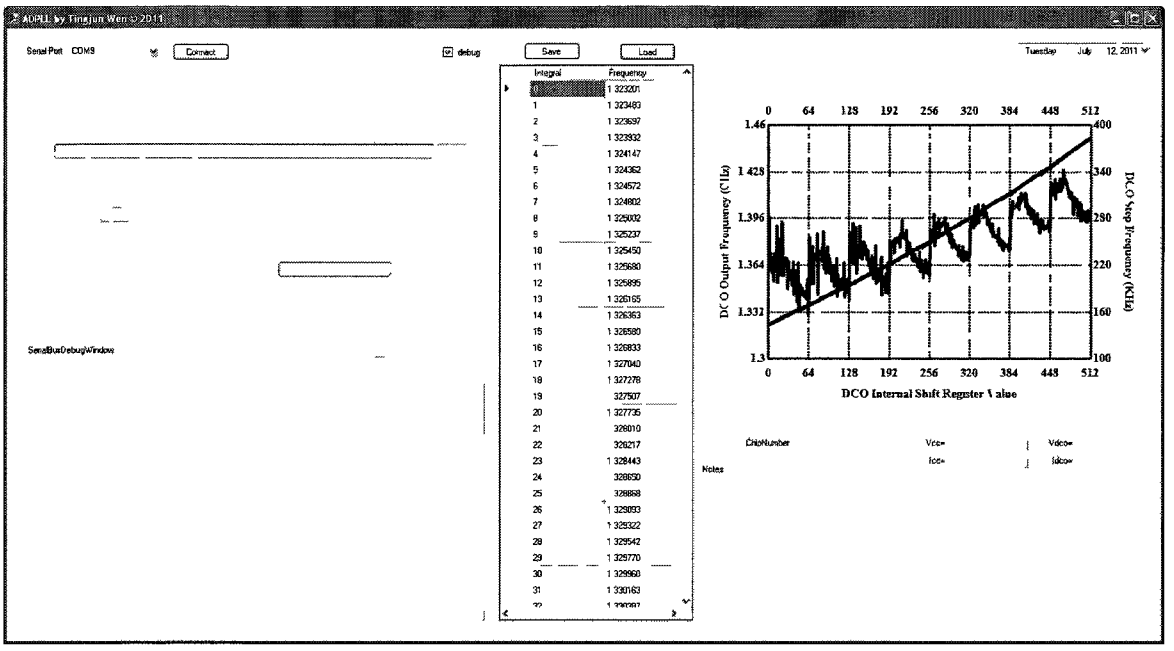

Figure 6.4: ADPLL Control and Measurement Software

management by time stamping and by comment insertion. The *Microsoft C#* is used to write the control and measurement software. A snapshot of the software GUI is shown in Figure 6.4 when it is used to obtain the DCO tuning curve.

6.2 CMOS DCO Performance

The ADPLL can operate in two different modes: open-loop mode and closed-loop mode. This is controlled by a dedicated pin. When the input of this pin is high, it operates in the open loop mode; when the input of this pin is low, the circuit is closed to form a complete ADPLL loop. In the open-loop mode, the SPI bus signals SCK and SDI can used to control the shift register chain in the integral path so the integral path tuning curve of the DCO can be easily characterized by measuring the

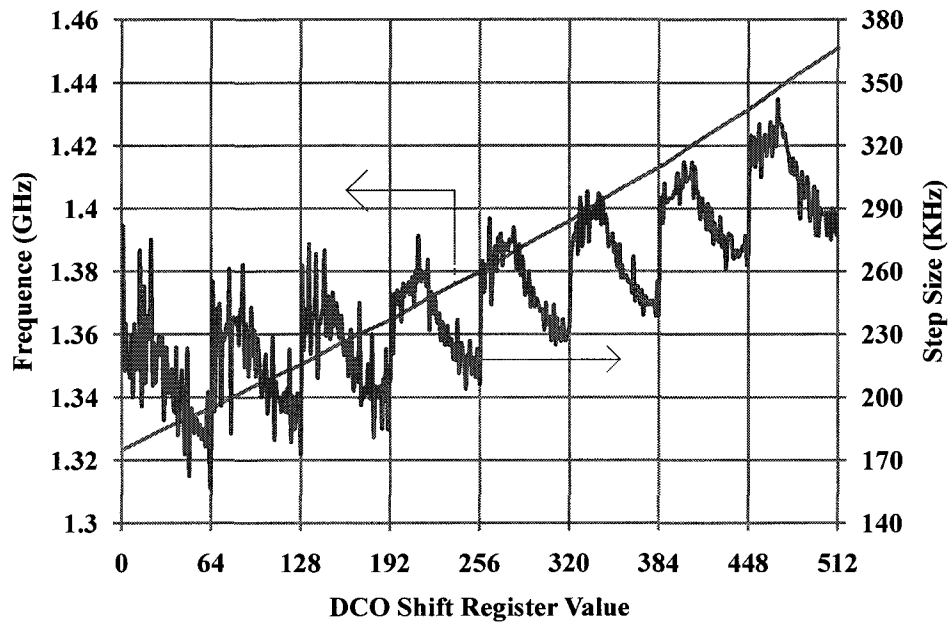

Figure 6.5: The Integral Path Tuning Curve

averaged carrier frequency by a spectrum analyzer. It takes about 20 to 30 seconds to characterize a single control word value. This amounts to about 3 to 4 hours to characterize all 512 control word values. A fully-automated control and measure software would be greatly appreciated.

Figure 6.5 shows the characterized integral path tuning curves when the proportional path coefficient is fixed to the constant 0. The straight line is shows the DCO output frequency in GHz vs the control word value. The saw tooth curve is the differentiated DCO step size in KHz at different control word values. The measured average step size from this diagram is $250KHz$ with a standard deviation of $46KHz$ within the output range from $1.32GHz$ to $1.45GHz$.

It is interesting to note that the step size curve is saw tooth shaped and also tips upward with the control word value. Since the integral path varactors with unit

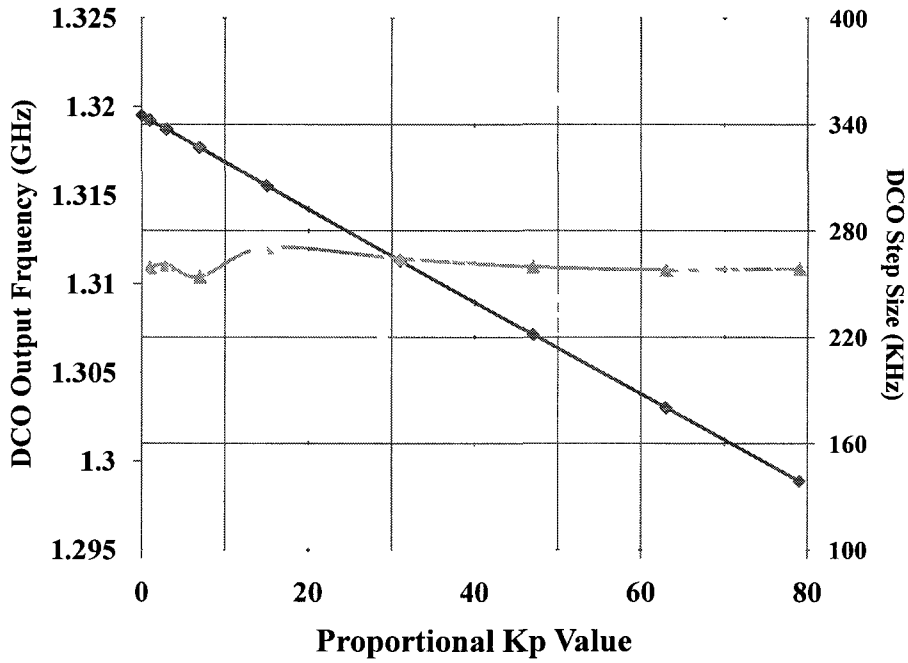

Figure 6.6: The Proportional Path Tuning Curve

size $2\mu m$ x $2\mu m$ are sparsely distributed and snake chained within $400\mu m$ x $400\mu m$ area, this can be easily explained by the large interconnect parasitic capacitance, inductance, resistance, or even the mutual inductance created by short and long wires associated with different control word values. It is also likely that the process variation across such a large area causes this effect. As long as the tuning curve is monotonic and the step sizes do not vary by too much, the effect of this second-order non-linear tuning curve on the phase noise should not be a big concern.

Similarly, the integral path tuning curve can also be obtained by fixing the integral shift register value to the constant 0. The obtained proportional path tuning curve is shown in Figure 6.6.

There are a few comments about this tuning curve as well. First, the proportional tuning curve has a negative slop different from the integral path due to the

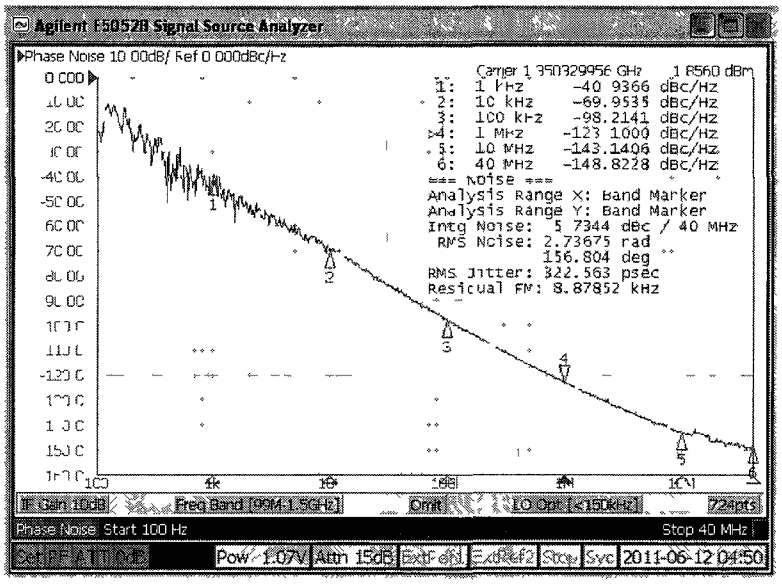

Figure 6.7: Measured Phase Noise of the DCO

way how the proportional path signal is connected in the loop. The average step size of the proportional path is $260KHz$. It is $10KHz$ bigger than the integral path step size even though both paths use the same unit varactor size. Again this is due to the wire parasitics and process variation across different regions. The coding method used for the proportional path only allows the control word values of [0,1,2,3,4,5,6,7,8,15,31,47,63,79] to simplify the control logic. Only the small proportional control word shows the step size variation. The bigger control word averages the step size to a relative constant value of $260KHz$.

The phase noise is the most important key parameter for a DCO used in any PLL as it partially determines how low the PLL phase noise can be. The measured phase noise of the DCO is presented in Figure 6.7. The achieved phase noise is $-123dBc/Hz@1MHz$ offset with the carrier frequency of $1.35GHz$.

The $-20dB$ slope due to the white flatten thermal noise integration is limited

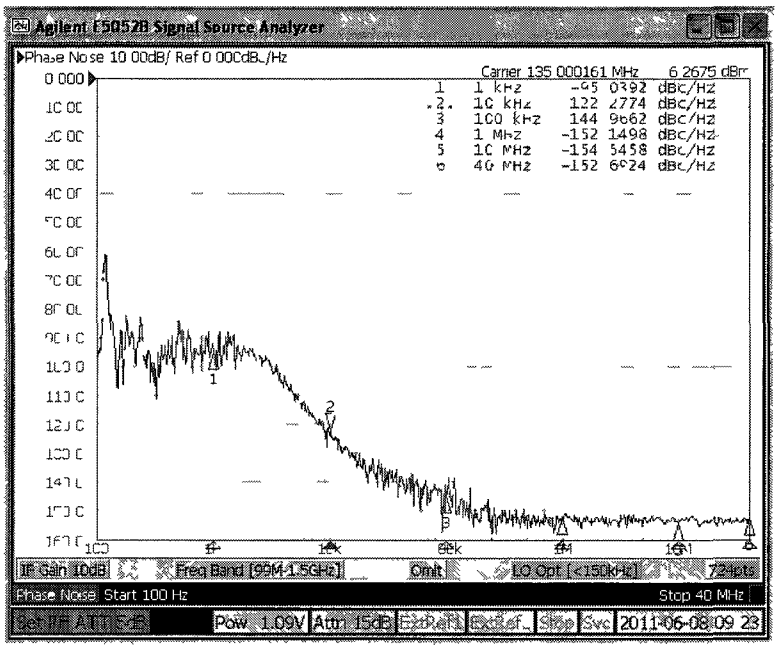

Figure 6.8: Measured Phase Noise of the Reference Clock of $135MHz$

above $1MHz$. Below $1MHz$, the phase noise exhibits $-30dB$ slope due the DCO integration filtering of the pink noise (flicker noise) with $-10dB$ slop. Since CMOS is notorious for the higher flicker noise compared to bipolar technology, this behavior is expected with a typical CMOS technology.

6.3 CMOS ADPLL Performance

Since the ADPLL output signal directly tracks input reference clock within the noise bandwidth, the close-in phase noise of the synthesized ADPLL output is usually higher than $20log(N)dB$ where N is the divider ratio. The close-in phase noise of the ADPLL output does not necessarily to be flat. It inherits whatever the shape of the input reference clock, including any reference clock spurs.

Figure 6.8 shows the input reference clock used by the ADPLL. The reference

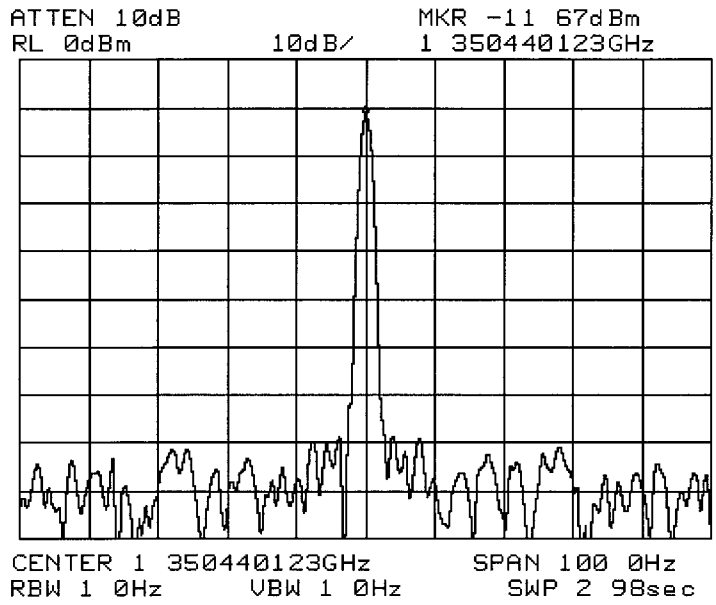

Figure 6.9: Frequency Spectrum of the ADPLL Output Clock of $1.35GHz$ With $1Hz$ Resolution Bandwidth

has a flat $-152dBc/Hz$ phase noise above $200KHz$ offset, $-30dB/dec$ slop between $2KHz$ and $200KHz$ offset, $-95dBc/Hz$ with a big spur below $2KHz$ offset.

The ADPLL output frequency spectrum is shown in Figure 6.9. Since the resolution bandwidth is $1Hz$, the close-in phase noise is just the difference between the carrier and the offset which is about $-80dBc/Hz$. This value is determined exclusively by the phase noise value of the reference clock and the divider ratio. This is confirmed by the subsequent more accurate phase noise measurement.

The measured phase noise of the ADPLL is shown in Figure 6.10. The close-in phase noise at $1KHz$ offset is $-75.9dBc/Hz$ and the phase noise at $1MHz$ offset is $-120.77dBc/Hz$.

As can be seen that the close-in phase noise truthfully tracks the reference phase

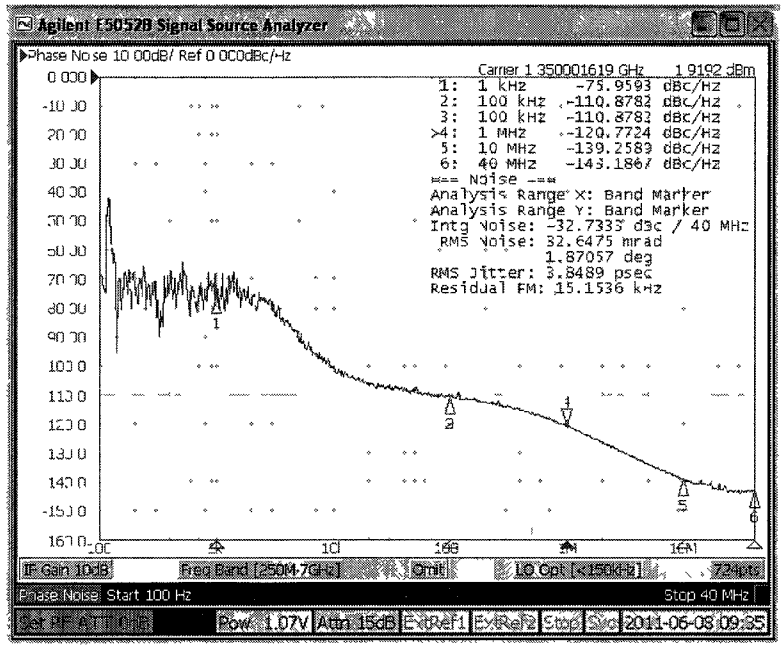

Figure 6.10: Measured Phase Noise of the ADPLL Output Clock of $1.35GHz$

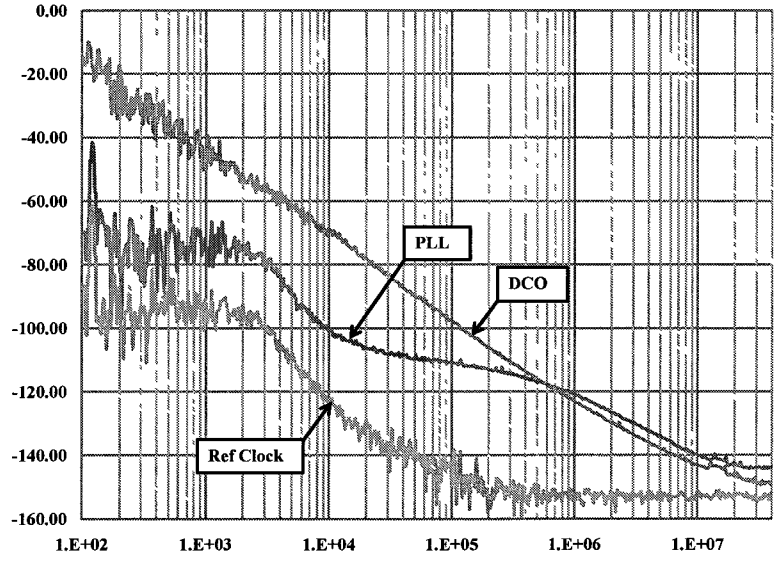

Figure 6.11: Measured Phase Noise of the ADPLL, the DCO, and the Reference Clock Superimposed

noise with a difference of $-20log(N)$ and the phase noise outside the noise bandwidth closely tracks the DCO phase noise. The output power level is $1.92dBm$. The integrated RMS jitter from $100Hz$ to $40MHz$ is 1.87^o or $3.85psec$.

Figure 6.11 combines the phase noise plots of the ADPLL, the DCO, and the reference clock together for easy comparison. The reference clock frequency used in the test is 135MHz and the divide ratio is 10. The reference spur of -78dB at 62.5MHz offset is observed. The spur offset is at half the reference frequency due to the one clock delay of the phase error from the BBPFD. This spur can be a problem for applications that use the ADPLL as the clock source and the application can lock to this spur accidentally. The commonly used solution is to apply a bandpass filter to the output of the ADPLL. This solution is not used in this thesis as it is out of the scope of this thesis.

6.4 Performance Summary

In this chapter, we presented the experimental results summarized in Table 6.2. Table 6.3 compares the performance results of similar PLL designs in the past two years with carrier frequencies from 1GHz to 3.28GHz in $0.13\mu m$ and below CMOS technologies. The results demonstrated the suitability of the ADPLL as the frequency synthesizer with the low phase noise of $-120.77dBc/Hz@1MHz$ at the core power consumption of $13mW$ and with an active area of $0.193mm^2$.

Table 6.2: ADPLL Performance Summary

Technology	0.13 μm CMOS
Chip Size	$2mm^2$
Active Area Size	$0.193mm^2$
DCO Supply Voltage	1.06 V
DCO current	17 mA
Other Circuit Supply Voltage	1.03 V
Other Circuit Current	13 mA
Open-loop Phase Noise	$-122.10dBC/Hz@1MHz$
Closed-loop Phase Noise	$-120.77dBC/Hz@1MHz$
Locking Range	$1.298GHz \sim 1.451GHz$

Table 6.3: Comparison of ADPLLs in CMOS

	[42]	[43]	[44]	[45]	[46]	This Work
Technolog	130nm	65nm	32nm	45nm	65nm	130nm
PLL Type	Analog	Digital	Analog	Digital	Hybrid	Digital
VCO	Ring	LC	CCO	RO/LC	Ring	LC
f_{out}	$1GHz$	$3.28GHz$	$1GHz$	$2.5GHz$	$1.24GHz$	$1.35GHz$
Power	16.8mW	4.5mW	1V	$2.5V$	$1.2V$	$1.1V$
Current	N/A	N/A	~1mA	$28/24mA$	N/A	$13mA$
Area	$0.31mm^2$	$0.22mm^2$	$0.046mm^2$	$0.277mm^2$	$0.12mm^2$	$0.193mm^2$
$\mathcal{L}^a$	-100	-109.92	-110	-106.6/-112.1	-119.67	-120.77

[a]: Phase noise $dBc/Hz@1MHz$

Chapter 7

Conclusions

7.1 Summary

This thesis introduces the ADPLL applications followed by the literature review with a few typical published ADPLL papers relevant to the topic in this thesis. The linearized s-domain model of the analog PLL and the linearized z-domain model of the ADPLL are presented to provide the solid theoretical background.

Based on the literature review and the ADPLL theories, a simple ADPLL architecture targeted for the low phase noise and wide locking range is proposed which consists of:

- a PFD that eliminates the cycle slips and has a narrow dead-zone,
- a DCO with a built-in LPF which is implemented as a bidirectional shift register, and
- an improved MMD divider that has wide divide ratio of 2 to $2^n - 1$.

An efficient top-down design methodology is used in this thesis to guarantee that the performance is met and the logic is working correctly at all levels. The *SystemVerilog* is used as the system level simulation tool to verify that the proposed ADPLL is operational in the time domain and can meet the phase noise expectation. All the *SystemVerilog* module used in the simulation can be directly synthesize into the FPGA except the LC DCO. The LC DCO is replaced with a fully synthesized 3-stage ring oscillator whose frequency is controlled by an array of tri-state load. The FPGA implementation validates most of the circuit at the gate level and serves as a propotype of the CMOS ADPLL. Compared with 5-month CMOS design turnaround time, the FPGA implementation is an efficient and economical way to guarantee the success of the first silicon tape-out.

Finally, the ADPLL based on the LC DCO is implemented in $0.13\mu m$ CMOS technology by using the standard digital cells and two custome high speed cells. The fabricated ADPLL chip is successfully characterized and evaluated with the open loop phase noise of $-123.10dBc/Hz@1MHz$ and closed loop phase noise of $-120.77dBc/Hz@1MHz$.

7.2 Significant Contributions

The significant contributions of this thesis are:

- Applied the *Mersenne-Twister* algorithm, *Box-Muller* Algorithm, and *Stochastic Voss-McCartney* algorithm in the phase noise simulation by *SystemVerilog*
- Presented a *completely digitally synthesized ADPLL in FPGA* including the

DCO with measured phase noise of $-98.33dBc/Hz$ at $1MHz$ offset with the carrier frequency of $572MHz$.

- Implemented and provided the detailed analysis of the *the DCO with a built-in low pass filter* in 0.13 μm CMOS technology with the phase noise of $-123.10dBc/Hz$ at $1MHz$ offset with carrier frequency of $1.35GHz$.

- Implemented the compact *multi-modulus divider (MMD) divide range from* 2 *to* $2^n - 1$ in CMOS 0.13 μm technology that is a good candidate for $\Delta\Sigma$ modulation in frequency synthesis.

- Implemented *the ADPLL by standard digital cells and two high speed cells* in CMOS 0.13 μm technology. The measured phase noise of the ADPLL is $-120.77dBc/Hz$ at $1MHz$ offset with a carrier of $1.35GHz$.

7.3 Potential Applications

The algorithms used in the SystemVerilog behavioral simulation is currently written in C connected by DPI interface. The SystemVerilog compiler vendors can easily turn those algorithms written in C into native tasks and functions so it is easy to include the noise simulation in the language.

The fully synthesized APDLL in FPGA and the cell-based ADPLL in CMOS are the pioneering work towards a fully synthesized ADPLL in any SoC applications. Almost all the SoC applications nowadays includes quite a few expensive and customized ADPLL IP blocks from third party vendors. The successfully designed ADPLL in both the FPGA and the CMOS technology in this thesis has been measured with fa-

vorable phase noise values which proves the feasibility of a fully synthesized ADPLL.

The ADPLL proposed in this thesis is suitable for the following applications: the high-speed chip-to-chip serial interconnections, the optical transceivers for the backplane interconnections, and the low phase noise clock generators. By adding the $\Delta - \Sigma$ modulator in the ADPLL, the applications can also be extended to the wireless transceivers for mobile devices.

7.4 Future Research Direction

The non-linear behavior of the ADPLL has not been fully studied yet. Since the ADPLL is a highly non-linear circuit, good understanding of the non-linear theory may aid in designing a better ADPLL. The book [26] and the papers [47] [48] [49] [50] [51] are good places to start with the non-linear PLL theory.

Based on the extensive measurement experience and review of some published papers [52] [53] [54] [55], the author feels that an ADPLL which combines both the phase lock and the injection lock mechnisms may lead to better phase noise result.

References

[1] R. B. Staszewski, J. L. Wallberg, S. Rezeq, C.-M. Hung, O. E. Eliezer, S. K. Vemulapalli, C. Fernando, K. Maggio, R. Staszewski, N. Barton, M.-C. Lee, P. Cruise, M. Entezari, K. Muhammad, and D. Leipold, "All-Digital PLL and Transmitter for Mobile Phones," *IEEE J. Solid-State Circuits*, vol. 40, no. 12, pp. 2469–2482, 2005.

[2] N. Da Dalt, "Linearized Analysis of a Digital Bang-Bang PLL and Its Validity Limits Applied to Jitter Transfer and Jitter Generation," *IEEE Trans. Circuits Syst. I*, vol. 55, no. 11, pp. 3663–3675, 2008.

[3] J. Zhuang, Q. Du, and T. Kwasniewski, "A 4GHz Low Complexity ADPLL-Based Frequency Synthesizer in 90nm CMOS," in *Proc. IEEE Custom Integrated Circuits Conf. CICC '07*, pp. 543–546, 2007.

[4] H. de Bellescise, "La Réception Synchrone," *Onde Electrique*, vol. 11, 1932.

[5] E. A. M. Klumperink, S. L. J. Gierkink, A. P. van der Wel, and B. Nauta, "Reducing MOSFET 1/f Noise and Power Consumption by Switched Biasing," *IEEE J. Solid-State Circuits*, vol. 35, no. 7, pp. 994–1001, 2000.

[6] D. Siprak, M. Tiebout, N. Zanolla, P. Baumgartner, and C. Fiegna, "Noise Reduction in CMOS Circuits Through Switched Gate and Forward Substrate Bias," *IEEE J. Solid-State Circuits*, vol. 44, no. 7, pp. 1959–1967, 2009.

[7] A. P. van der Wel, E. A. M. Klumperink, S. L. J. Gierkink, R. F. Wassenaar, and H. Wallinga, "Mosfet 1/f Noise Measurement under Switched Bias Conditions," *IEEE Electron Device Lett.*, vol. 21, no. 1, pp. 43–46, 2000.

[8] C.-C. Hung and S.-I. Liu, "A 40-GHz Fast-Locked All-Digital Phase-Locked Loop Using a Modified Bang-Bang Algorithm," *IEEE Trans. Circuits Syst. II*, vol. 58, no. 6, pp. 321–325, 2011.

[9] A. Rylyakov, J. Tierno, H. Ainspan, J.-O. Plouchart, J. Bulzacchelli, Z. T. Deniz, and D. Friedman, "Bang-Bang Digital PLLs at 11 and 20GHz with Sub-200fs Integrated Jitter for High-Speed Serial Communication Applications," in *Proc. IEEE Int. Solid-State Circuits Conf. - Digest of Technical Papers ISSCC 2009*, pp. 94–95, 2009.

[10] S.-Y. Yang, W.-Z. Chen, and T.-Y. Lu, "A 7.1 mW, 10 GHz All Digital Frequency Synthesizer With Dynamically Reconfigured Digital Loop Filter in 90 nm CMOS Technology," *IEEE J. Solid-State Circuits*, vol. 45, no. 3, pp. 578–586, 2010.

[11] K.-H. Tsai and S.-I. Liu, "A 104-GHz Phase-Locked Loop Using a VCO at Second Pole Frequency," *IEEE Trans. VLSI Syst.*, no. 99, pp. 1–9, 2010. Early Access.

[12] W. Altabban, P. Desgreys, H. Petit, K. Ben Kalaia, and L. du Roscoat, "Merged Digitally Controlled Oscillator and Time to Digital Converter for TV band AD-

PLL," in *Circuits and Systems (ISCAS), Proceedings of 2010 IEEE International Symposium on*, pp. 1987 –1990, June 2010.

[13] H.-H. Chang, C.-H. Fu, and M. Chiu, "A 320fs-RMS-Jitter and 300kHz-BW All-Digital Fractional-N PLL with Self-Corrected TDC and Fast Temperature Tracking Loop for WiMax/WLAN 11n," in *VLSI Circuits, 2009 Symposium on*, pp. 188 –189, June 2009.

[14] M.-H. Chang, Z.-X. Yang, and W. Hwang, "A 1.9mW Portable ADPLL-Based Frequency Synthesizer for High Speed Clock Generation," in *Circuits and Systems, 2007. ISCAS 2007. IEEE International Symposium on*, pp. 1137 –1140, May 2007.

[15] R. Staszewski, J. Wallberg, C.-M. Hung, G. Feygin, M. Entezari, and D. Leipold, "LMS-Based Calibration of an RF Digitally Controlled Oscillator for Mobile Phones," *Circuits and Systems II: Express Briefs, IEEE Transactions on*, vol. 53, pp. 225 – 229, March 2006.

[16] R. Staszewski, R. Staszewski, T. Jung, T. Murphy, I. Bashir, O. Eliezer, K. Muhammad, and M. Entezari, "Software Assisted Digital RF Processor (DRP tm) for Single-Chip GSM Radio in 90 nm CMOS," *Solid-State Circuits, IEEE Journal of*, vol. 45, pp. 276 –288, Feb. 2010.

[17] J. Deskur and A. Maciejuk, "Application of Digital Phase Locked Loop for Control of SRM Drive," in *Power Electronics and Applications, 2007 European Conference on*, pp. 1 –6, Sept. 2007.

[18] N. T. Hieu, T.-W. Lee, and H.-H. Park, "All-Digital Phase-Locked Loop for Optical Interconnect Applications," in *Advanced Communication Technology, The 9th International Conference on*, vol. 3, pp. 1829 –1832, Feb. 2007.

[19] H.-J. Hsu, C.-C. Tu, and S.-Y. Huang, "A High-Resolution All-Digital Phase-Locked Loop with its Application to Built-in Speed Grading for Memory," in *VLSI Design, Automation and Test, 2008. VLSI-DAT 2008. IEEE International Symposium on*, pp. 267 –270, April 2008.

[20] G. jun Xie and C. Wang, "An All-Digital PLL for Video Pixel Clock Regeneration Applications," in *Computer Science and Information Engineering, 2009 WRI World Congress on*, vol. 3, pp. 392 –396, April 2009.

[21] A. Neyer, J. Mueller, S. Kaehlert, R. Wunderlich, and S. Heinen, "A Fully Integrated All-Digital PLL Based FM-radio Transmitter in 90 nm CMOS," in *NEWCAS Conference (NEWCAS), 2010 8th IEEE International*, pp. 225 –228, June 2010.

[22] I. Syllaios, P. Balsara, and R. Staszewski, "On the Reconfigurability of All-Digital Phase-Locked Loops for Software Defined Radios," in *Personal, Indoor and Mobile Radio Communications, 2007. PIMRC 2007. IEEE 18th International Symposium on*, pp. 1 –6, Sept. 2007.

[23] J. Rogers, C. Plett, and F. Dai, *Integrated Circuit Design For High-Speed Frequency Synthesis.* Artech House, 2006.

[24] F. M. Gardner, *Phaselock Techniques Third Edition.* John Wiley & Sons, Inc, 2005.

[25] R. E. Best, *Phase-Locked Loops Design, Simulation, and Applications Sixth Edition.* The McGraw-Hill Companies, Inc, 2007.

[26] J. L. Stensby, *Phase-Locked Loops.* CRC Press LLC, 1997.

[27] N. Da Dalt, "Markov Chains-Based Derivation of the Phase Detector Gain in Bang-Bang PLLs," *IEEE Trans. Circuits Syst. II*, vol. 53, no. 11, pp. 1195–1199, 2006.

[28] J. Lee, K. S. Kundert, and B. Razavi, "Analysis and Modeling of Bang-Bang Clock and Data Recovery Circuits," *IEEE J. Solid-State Circuits*, vol. 39, no. 9, pp. 1571–1580, 2004.

[29] K. Kundert, "Predicting the Phase Noise and Jitter of PLL-Based Frequency Synthesizers," Aug. 2006.

[30] J. Zhuang, Q. Du, and T. Kwasniewski, "Event-Driven Modeling and Simulation of an Digital PLL," *Behavioral Modeling and Simulation Workshop, Proceedings of the 2006 IEEE International*, pp. 67–72, Sept. 2006.

[31] S. Huang, H. Ma, and Z. Wang, "Modeling and Simulation to the Design of $\Sigma\Delta$ Fractional-N Frequency Synthesizer," in *DATE '07: Proceedings of the conference on Design, automation and test in Europe*, (San Jose, CA, USA), pp. 291–296, EDA Consortium, 2007.

[32] R. Staszewski, C. Fernando, and P. Balsara, "Event-Driven Simulation and Modeling of Phase Noise of an RF Oscillator," *Circuits and Systems I: Regular Pa-*

pers, IEEE Transactions on [Circuits and Systems I: Fundamental Theory and Applications, IEEE Transactions on], vol. 52, no. 4, pp. 723–733, April 2005.

[33] M. Matsumoto and T. Nishimura, "Mersenne Twister: a 623-Dimensionally Equidistributed Uniform Pseudo-Random Number Generator," *ACM Trans. Model. Comput. Simul.*, vol. 8, no. 1, pp. 3–30, 1998.

[34] L. Tammell, "Improvements in the correlated pink noise generator evaluation," Jan. 2006.

[35] R. Stefo, J. Schreiter, J.-U. Schlussler, and R. Schuffny, "High Resolution ADPLL Frequency Synthesizer for FPGA-and ASIC-Based Applications," in *Proc. IEEE Int Field-Programmable Technology (FPT) Conf*, pp. 28–34, 2003.

[36] T. Olsson and P. Nilsson, "A Digitally Controlled PLL for SoC Applications," *Solid-State Circuits, IEEE Journal of*, vol. 39, pp. 751 – 760, May 2004.

[37] A. Wortmann, S. Simon, and M. Muller, "A High-Speed Transceiver Architecture Implementable as Synthesizable IP Core," in *Design, Automation and Test in Europe Conference and Exhibition, 2004. Proceedings*, vol. 3, pp. 46 – 51 Vol.3, 2004.

[38] U. Meyer-Baese, *Digital Signal Processing with Field Programmable Gate Arrays Second Edition*. Springer-Verlag, 2004.

[39] M. A. Tarar, J. Sun, A. Sampson, R. Wilcox, and Z. Chen, "The Implementation of a New All-Digital Phase-Locked Loop on an FPGA and its Testing in a

Complete Wireless Transceiver Architecture," in *Proc. Seventh Annual Communication Networks and Services Research Conf. CNSR '09*, pp. 238–244, 2009.

[40] M. Kumm, H. Klingbeil, and P. Zipf, "An FPGA-Based Linear All-Digital Phase-Locked Loop," *Circuits and Systems I: Regular Papers, IEEE Transactions on*, vol. 57, pp. 2487 –2497, Sept. 2010.

[41] T. Wen and T. Kwasniewski, "Phase Noise Simulation and Modeling of AD-PLL by SystemVerilog," in *Behavioral Modeling and Simulation Workshop, 2008. BMAS 2008. IEEE International*, pp. 29 –34, Sept. 2008.

[42] D.-W. Jee, Y. Suh, H.-J. Park, and J.-Y. Sim, "A 0.1-fref BW 1GHz Fractional-N PLL with FIR-Embedded Phase-Interpolator-Based Noise Filtering," in *Proc. IEEE Int. Solid-State Circuits Conf. Digest of Technical Papers (ISSCC)*, pp. 94–96, 2011.

[43] D. Tasca, M. Zanuso, G. Marzin, S. Levantino, C. Samori, and A. L. Lacaita, "A 2.9-to-4.0GHz Fractional-N Digital PLL with Bang-Bang Phase Detector and 560fsrms Integrated Fitter at 4.5mW Power," in *Proc. IEEE Int. Solid-State Circuits Conf. Digest of Technical Papers (ISSCC)*, pp. 88–90, 2011.

[44] H.-J. Lee, A. M. Kern, S. Hyvonen, and I. A. Young, "A Scalable Sub-1.2mW 300MHz-to-1.5GHz Host-Clock PLL for System-on-Chip in 32nm CMOS," in *Proc. IEEE Int. Solid-State Circuits Conf. Digest of Technical Papers (ISSCC)*, pp. 96–97, 2011.

[45] D. M. Fischette, A. L. S. Loke, M. M. Oshima, B. A. Doyle, R. Bakalski, R. J. DeSantis, A. Thiruvengadam, C. L. Wang, G. R. Talbot, and E. S. Fang, "A

45nm SOI-CMOS Dual-PLL Processor Clock System for Multi-protocol I/O," in *Proc. IEEE Int. Solid-State Circuits Conf. Digest of Technical Papers (ISSCC)*, pp. 246–247, 2010.

[46] A. Sai, T. Yamaji, and T. Itakura, "A 570fsrms Integrated-jitter Ring-VCO-based 1.21GHz PLL With Hybrid Loop," in *Proc. IEEE Int. Solid-State Circuits Conf. Digest of Technical Papers (ISSCC)*, pp. 98–100, 2011.

[47] M. Zabihi and H. M. Naimi, "Novel Nonlinear Fast Locking PLLs Based on Mathematical Models," in *Proc. Int. Conf. Application of Information and Communication Technologies AICT 2009*, pp. 1–4, 2009.

[48] T.-C. Wang, T.-Y. Chiou, and S. Lall, "Nonlinear Phase-Locked Loop Design Using Semidefinite Programming," in *Proc. 16th Mediterranean Conf. Control and Automation*, pp. 1640–1645, 2008.

[49] O. Scharf, O. Schnick, and W. Mathis, "A non-linear state-space model for plls and a comparison between plls and the entrainment effect," *Theoretical Engineering (ISTET), 2009 XV International Symposium on*, pp. 1 –4, June 2009.

[50] M. Ranaivoniarivo, S. Wane, E. Richalot, and O. Picon, "A System-Level Non-linear Behavioral Modeling of Pulling and Pushing Mechanisms in PLLs," in *Proc. IEEE Asia Pacific Conf. Circuits and Systems (APCCAS)*, pp. 947–950, 2010.

[51] B. Bohm, J. A. Schoonees, and J. Tapson, "Some Non-Linear Effects of Quantization in Phase-Locked Loops," in *Proc. IEEE South African Symp. Communications and Signal Processing COMSIG-94*, pp. 116–120, 1994.

[52] F. Plessas, F. Gioulekas, and G. Kalivas, "Phase Noise Performance of Fully Differential Sub-Harmonic Injection-locked PLL," *Electronics Letters*, vol. 46, no. 19, pp. 1319–1321, 2010.

[53] J. Lee, H. Wang, W.-T. Chen, and Y.-P. Lee, "Subharmonically Injection-Locked PLLs for Ultra-low-Noise Clock Generation," in *Proc. IEEE Int. Solid-State Circuits Conf. - Digest of Technical Papers ISSCC 2009*, pp. 92–93, 2009.

[54] S. Kudszus, M. Neumann, T. Berceli, and W. H. Haydl, "Fully Integrated 94-GHz Subharmonic Injection-Locked PLL Circuit," *IEEE Microw. Guided Wave Lett.*, vol. 10, no. 2, pp. 70–72, 2000.

[55] J. Matos, A. Gameiro, and J. Perez, "A 2GHZ PLL with Injection Lock of the VCO," in *Proc. 25th European Microwave Conf*, vol. 2, pp. 663–667, 1995.

Appendix A

Publications and Patents

The following paper was published as an integral part of this thesis:

- T. Wen and T. Kwasniewski, "Phase Noise Simulation and Modeling of ADPLL by SystemVerilog," in Behavioral Modeling and Simulation Workshop, 2008. BMAS 2008. IEEE International, pp. 29 34, Sept. 2008.

The following patents issued to the author while he was pursuing his Master's degree are not part of this thesis:

- T. Wen and T. Kwasniewski, "Packet Processors Having Multi-Functional Range Match Cells Therein,", U.S. Patent 7 298 636, Nov. 2007.
- T. Wen, "Content Addressable Memory (CAM) Devices Having NAND-Type Compare Circuits," U.S. Patent 7 355 890, April, 2008.
- T. Wen, "Integrated Circuit Search Engine Devices Having Priority Sequencer Circuits Therein That Sequentially Encode Multiple Match Signals," U.S. Patent 7 822 916, Oct. 2010.

- T. Wen, D.W. Carr, and T. Kwasniewski, "Packet Processors Having Comparators Therein That Determine Non-Strict Inequalities Between Applied Operands," U.S. Patent 7 825 777, Nov. 2010.