NTNU – Trondheim
Norwegian University of
Science and Technology

Python Framework for Design and Simulation of Integer-N ADPLLs

Cole Nielsen

Electronic Systems Design, Specialization Project

Submission date: December 2019

Supervisor: Trond Ytterdal, IET

Co-supervisor: Carsten Wulff, IET

Norwegian University of Science and Technology

Department of Electronic Systems

Abstract.

An open source Python-language design automation and simulation framework for the design of integer-N all digital phase locked loop (ADPLL) frequency synthesizers is presented in this paper. The framework enables (1) automatic design of optimized second order discrete time digital loop filters, and (2) behavioral ADPLL simulation to verify loop filter and PLL designs for satisfactory phase noise, lock-time and stability; optionally subject to variation using Monte-Carlo sampling. Simulation is implemented with a discrete-event time domain simulator utilizing behavioral PLL component models, permitting accurate modeling of effects of time-discretization and quantization in an ADPLL. Loop filter design automation is based upon a prototype second order filter, whose parameters are optimized to minimize total integrated output phase noise for a PLL provided specifications for reference frequency, divider ratio, time-to-digital converter (TDC) resolution, digitally controlled oscillator (DCO) gain, oscillator phase noise characteristics and maximum lock time. The optimizer utilizes continuous phase transfer function approximation of ADPLL dynamics and phase noise, which allows for computationally fast evaluation, but also requires conversion of the design filters from continuous-to-discrete time. Second order optimization is utilized post filter design to map the discrete filter into a fixed-point digital with acceptable quantization noise and filter error due to quantization effects.

Problem description.

The intent of this project is to develop a standalone PLL design and simulation framework to aid and facilitate a later master's thesis project regarding the design of an all-digital, ultra-low power PLL frequency synthesizer. This framework is intended to address and ease all-digital PLL design and simulation challenges at a high level. Specifically it should enable speedy PLL simulation defined with system and component- level specifications (e.g. desired frequency, gain of digitally controlled oscillator, phase detector resolution, divider ratio). This is to allow for development of component level specifications and verification of PLL performance (phase noise, lock time, stability) under ideal circumstances before transistor level implementation, thus accelerating overall implementation time for hardware.

A filter optimizer was made that correlates with simulation,

Contents

1 Introduction **11**

2 Theory **13**

2.1 Continuous PLL Model . . . 14
2.1.1 PLL Synthesizer architecture . . . 14
2.1.2 Divider . . . 14
2.1.3 Phase detector . . . 14
2.1.4 Loop Filter . . . 15
2.1.5 VCO . . . 15
2.1.6 Continuous PLL Transfer function . . . 15
2.2 ADPLL - digital, discretized PLL Model . . . 16
2.2.1 Divider . . . 16
2.2.2 TDC . . . 17
2.2.3 Discrete-time loop filter . . . 18
2.2.4 DCO . . . 19
2.2.5 Discrete-time PLL transfer function . . . 19
2.3 ADPLL Noise Model . . . 20
2.3.1 TDC noise . . . 20
2.3.2 DCO noise . . . 21
2.3.3 Divider noise . . . 22
2.3.4 Loop filter noise - direct form I . . . 23
2.3.5 PLL noise sensitivity transfer functions . . . 24
2.3.6 PLL phase signal and output PSD relationship for noise . . . 25
2.3.7 PLL output-referred noise PSD . . . 26
2.4 Modified ADPLL architecture for low TDC resolutions . . . 27

3 Loop Filter Design Automation **28**

3.1 Design sequence . . . 28
3.2 Loop filter Prototype . . . 29
3.2.1 Continuous PI-loop filter design . . . 30
3.2.2 PI-controller peaking compensation . . . 32
3.2.3 Alternative PID controller permutations . . . 33
3.2.4 Discretized Loop Filter Prototype . . . 33
3.2.5 Prototype PLL response . . . 34

4 Behavioral ADPLL simulation **35**

4.1 Simulation engine . . . 35
4.2 Clock behavioral model . . . 36

4.3 TDC behavioral model . . . 37
4.4 Loop filter behavioral model . . . 37
4.5 DCO behavioral model . . . 38
4.6 Divider behavioral model . . . 40
4.7 Post processing: lock time detection . . . 40
4.8 Post processing: phase noise power spectrum estimate . . . 41
4.9 Monte-Carlo sampling . . . 42

5 Loop filter optimization 43
5.1 Optimization rationale . . . 43
5.2 Loop filter optimization algorithm . . . 44
5.3 Fast estimation of PLL settling time . . . 46
5.4 Estimation of PLL phase noise . . . 47
5.5 Loop filter optimization - finite word effects . . . 48
5.5.1 Loop filter quantization noise optimization . . . 48
5.5.2 Loop filter transfer function error optimization . . . 48

6 Discussion and results 49
6.1 Design exercise using this work . . . 50
6.1.1 Result of filter optimization. . . . 50
6.1.2 Result of transient and phase noise simulation. . . . 50
6.1.3 Result of parametric sweep and variation analysis. . . . 51
6.1.4 Design method 2 . . . 53
6.2 Comparison to existing solutions . . . 56
6.3 Design choices and areas of improvement . . . 57
6.3.1 Divider noise constraint . . . 58

7 Conclusion 59

A Estimating PSD with autoregressive model 62

B Oscillator phase noise due to thermal noise 62

List of Figures

1 Basic phase feedback network. . . . 13
2 Basic PLL. . . . 14
3 Basic ADPLL. . . . 16
4 Digital divider signals. . . . 17
5 TDC model. . . . 17
6 Direct form I implementation of IIR filter. . . . 19
7 Discrete time PLL model. . . . 20
8 TDC quantization noise models. . . . 21
9 Quantization as via additive error signal. . . . 21
10 **(a)** Leeson model for phase noise, **(b)** DCO additive noise model. . . . 22
11 **(a)** Divider noise model, **(b)** Digital divider output jitter. . . . 22
12 Direct form I filter implementation with quantization. . . . 24
13 Full PLL additive noise model. . . . 25
14 PLL with parallel bang-bang phase detector and TDC. . . . 27
15 Filter design sequence. . . . 28
16 PI-controller PLL pole-zero locations. . . . 31
17 Example PI-PLL responses with varied ζ. . . . 31
18 PI-controller PLL phase margin versus damping ratio. . . . 32
19 Implementation of filter. . . . 33
20 Simulation process. . . . 35
21 **(a)** Discrete model for oscillator random walk, **(b)** Simulated phase noise of behavioral model. . . . 39
22 Detection of lock time from simulated PLL transient at loop filter. . . . 40
23 Power spectrum estimates example. . . . 42
24 **(a)** Example PLL phase noise from models in this work, **(b)** integrated phase noise power versus bandwidth for the same PLL. . . . 44
25 Bandwidth versus total integrated phase noise of PLL. . . . 45
26 Optimizer visualized. . . . 45
27 Quantization noise power power out of an example loop filter versus data word resolution. . . . 49
28 **(a)** Example filter error due to coefficient quantization, **(b)** Example MSE error of filter design due to coefficient quantization. . . . 49
29 Simulation with 0.5% initial frequency error: **(a)** Loop filter transient response, **(b)** PLL output instantaneous frequency. . . . 52
30 Simulation with 12 MHz (0.5%) initial frequency error: **(a)** BBPD/TDC detector responses, **(b)** PLL output phase noise power spectrum. . . . 52

31 **(a)** PLL lock time simulation with KDCO swept, 12 MHz (0.5%) initial frequency error, **(b)** PLL lock time simulation with initial frequency error swept. 53
32 Monte-Carlo simulation with 1000 samples, 20% RMS deviation in KDCO, and 60 MHz (2.5%) RMS deviation in initial frequency error **(a)** Frequency transient responses, **(b)** Lock time histogram. 53
33 Simulation with 0.5% initial frequency error: **(a)** Loop filter transient response, **(b)** PLL output instantaneous frequency. 54
34 Simulation with 12 MHz (0.5%) initial frequency error: **(a)** BBPD/TDC detector responses, **(b)** PLL output phase noise power spectrum. 54
35 **(a)** PLL lock time simulation with KDCO swept, 12 MHz (0.5%) initial frequency error, **(b)** PLL lock time simulation with initial frequency error swept. 55
36 Monte-Carlo simulation with 1000 samples, 20% RMS deviation in KDCO, and 60 MHz (2.5%) RMS deviation in initial frequency error **(a)** Frequency transient responses, **(b)** Lock time histogram. 55
37 DCO additive noise model. 62
38 Voltage to phase noise conversion. 63

List of Tables

1 System-level specifications 50
2 PLL parameters determined from filter design and optimization process. 51
3 Loop filter parameters after digitization and optimization for data word length. 51
4 PLL parameters extracted from variance and parameter sweep simulations. 54

Abbreviations.

ADPLL	All digital phase locked loop
BER	Bit error rate
BFGS	Broyden-Fletcher-Goldfarb-Shanno
BW	Bandwidth
CMOS	Complementary metal oxide semiconductor
DCO	Digitally controlled oscillator
DIV	Divider
DNL	Differential non-linearity
FFT	Fast Fourier transform
FM	Frequency modulation
IIR	Infinite impulse response
ISM	Industrial, scientific and medicine
LF	Loop filter
LSB	Least signigicant bit
OTW	Oscillator tuning word
PD	Phase detector
PI	Propotional-integral
PID	Proportional-integral-derivative
PLL	Phase locked loop
PN	Phase noise
PSD	Power spectral density
PVT	Process, voltage and temperature
RMS	Root mean squared
SSB	Single side band
TDC	Time to digital converter
TF	Transfer function
VCO	Voltage controlled oscillator

1 Introduction

Phase locked loops are extraordinarily useful frequency synthesizers that are vital to the operation of virtually all wired and wireless communication systems of today. The trend towards increasingly lower power wireless devices poses an acute need to reduce PLL power consumption. This is a challenge as PLLs typically rank among the highest power consuming components of a radio, and are necessarily so to limit oscillator phase noise. Reducing analog PLL power consumption can be a prohibitive challenge as the performance of analog loop filters degrade as a result of unavoidably lower charge pump current. However, recent CMOS process nodes with minimum gate lengths as small as 7nm allow for all-digital PLLs as an attractive alternative to analog designs due to low power consumption associated with the implementation of a digital loop filter. Digital loop-filters have the unique advantage where they can be scaled indefinitely as process nodes advance, suffering no loss in performance, while also having greatly reduced sensitivities to process, voltage, temperature (PVT) variations compared to analog implementations.

All-digital PLLs introduce new challenges in the process of design in the ultra-low power domain. Low power design is complemented by low complexity design, which in a PLL transcribes to low resolution of phase detectors, digitally tunable oscillators, and loop filter digital data paths. In other words, effects of quantization are strong. Quantization is an inherently nonlinear process, thus strong quantization is tied to strong nonlinear effects. Consequently, where high resolution digital PLLs with low quantization effects can be effectively analyzed with linear-time invariant transfer functions in the Z-domain, simulation and numerical analysis is necessary to accurately capture quantization effects in low power, low resolution ADPLLs . This, of course, presents a challenge in manual loop filter design of PLLs if simulation is an integral part to the most basic modeling, and thus motivates the creation of an automated design solution. Currently, no PLL design framework currently automates PLL loop filter design for the needs of ultra low power digital PLLs as characterized here.

Thus in this paper, a new framework written in pure-Python is introduced, which uniquely addresses issues of ultra-low power ADPLL design. Specifically, design of integer-N type PLLs is focused on, as the impetus of this work is an integer-N PLL design project. Topics presented are (a) automatic design and optimization of ADPLL loop filters given target system and component level specifications for the PLL, and (b) behavioral time domain PLL simulation for accurate analysis and verification of loop filter and PLL performance, with an integrated Monte-Carlo sampling variation analysis engine. Due to high phase noise associated with low power design, the optimization approach introduced in this paper focuses on minimization of total integrated phase noise power to allow for maximum PLL performance on a given power budget.

A brief outline of the paper is as follows. An introduction to PLL theory is in section 2. Simulation and optimization methods are discussed in sections 3-5. An example design exercise, a comparison to existing solutions, and general discussion considerations for using the framework are in section 6. Finally, section 7 concludes.

The main contributions of this work to PLL design are:

1. Fully automatic loop filter design and optimization for all digital integer-N PLLs in an open source Python framework, generating ready-for-hardware digital filter coeffients from high level PLL specifications (maximum lock time, reference frequency, DCO gain, divider ratio, oscillator phase noise). Design optimization performed in this work considers minimization of total phase noise and quantization errors in the final digital loop filter computed.
2. Generation of digital loop filter coefficients optimized for finite word effects (both quantization error and filter accuracy).
3. A behavioral time domain simulator of integer-N digital PLLs integrated with the loop filter designer. This simulator allows for verification of automatcally designed filters, with included analysis for variation to evaluate both lock-time and phase noise performance, as well as stability.

2 Theory

In its most basic form, a phase locked loop is a phase-based feedback system whose output tracks or maintains a fixed phase relationship to an input signal. As will be shown, such a system is uniquely suited to the task of frequency synthesis, which is the process of generating derivative frequencies from some reference frequency. Given reference and output phase signals Φ_{ref} and Φ_{out}, a PLL can be modeled as in figure 1, with feedforward and feedback networks A(s) and B(s).

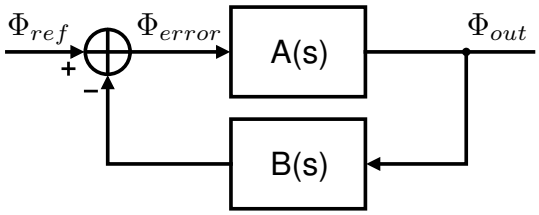


Figure 1: Basic phase feedback network.

The closed loop phase response T(s) for Φ_{ref} to Φ_{out} is therefore:

$$\mathrm{T}(s) = \frac{\Phi_{out}(s)}{\Phi_{ref}(s)} = \frac{A(s)}{1 + A(s)B(s)} \tag{1}$$

A particular case of interest is when B(s) = 1/N, where N is a constant, and the loop gain L(s) = A(s)B(s) $>>$ 1. The closed loop response for this case is:

$$\frac{\Phi_{out}(s)}{\Phi_{ref}(s)} \approx \frac{A(s)}{A(s)B(s)} = \frac{1}{B(s)} = N \tag{2}$$

We see that the phase through the PLL is multiplied by a factor of N. If the input phase signal is a sinusoid with frequency ω_{ref}, and likewise the output with ω_{out}, then $\phi_{ref}(t) = \omega_{ref}t$ and $\phi_{out}(t) = \omega_{out}t$. Thus:

$$\frac{\Phi_{out}(t)}{\Phi_{ref}(t)} = \frac{\omega_{out}t}{\omega_{ref}t} \approx N \quad \rightarrow \quad \omega_{out} \approx N\omega_{ref} \tag{3}$$

Therefore, it is observed that a PLL allows for the generation, i.e. synthesis, of a new frequency from a reference frequency signal. Given a feedback divider ratio of 1/N, the PLL multiplies the reference frequency by a factor of N. In the following sections, more advanced models for PLL will be developed, extending the concept introduced here. Specifically, the theory of digital, discrete-time PLLs will be developed and extended from a continuous phase model of a basic PLL.

2.1 Continuous PLL Model

Although this work is interested in discrete time sampling PLL design, as will later be seen continuous models can still be of use in the analysis and design of such systems. Thus a continuous PLL model is developed in this section.

2.1.1 PLL Synthesizer architecture

The traditional architecture for implementing a PLL frequency synthesizer [20] is shown in figure 2. This basic PLL is comprised of four components: (1) a phase detector, denoted by PD, (2) a loop filter, denoted by $\mathrm{H}_{LF}(\mathrm{s})$, (3) a voltage controlled oscillator, denoted by VCO, and (4) and phase divider, denoted by "÷ N" in the figure. These components are explained in the following sections.

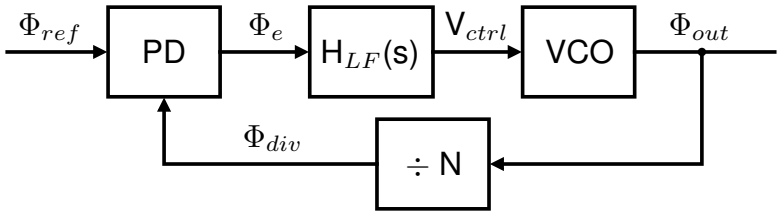


Figure 2: Basic PLL.

2.1.2 Divider

The phase divider is used as the feedback path in the PLL, where the division ratio N controls the phase and consequent frequency multiplication of the PLL. The transfer function of the divider is:

$$\mathrm{H}_{div}(s) = \frac{\Phi_{div}(s)}{\Phi_{out}(s)} = \frac{1}{\mathrm{N}} \tag{4}$$

2.1.3 Phase detector

The phase detector is used to measure the phase of the feedback signal (i.e. divider output) in relation to the reference phase, in order to establish a phase error signal Φ_e used to control the tuning of the PLL.

$$\Phi_e(s) = \Phi_{ref}(s) - \Phi_{div}(s) \tag{5}$$

2.1.4 Loop Filter

The PLL loop filter is used to control the phase-frequency response of PLL, which affects transient PLL behavior, as well as phase noise performance (see section 2.3). This can be designed to have P poles and Z zeros, and can be represented in the canonical form of equation 6 as a rational function of polynomials of s with coefficients given with $\{a_0, ..., a_P\}$ and $\{b_0, ..., b_Z\}$.

$$\mathrm{H}_{LF}(s) = \frac{\sum_{j=0}^{Z} b_j s^j}{\sum_{k=0}^{P} a_k s^k} \tag{6}$$

2.1.5 VCO

The voltage controlled oscillator is an oscillator with frequency controlled by an input signal V_{ctrl}. The VCO is characterized by its gain $K_{DCO} = \partial f / \partial \mathrm{V}_{ctrl}$, and the nominal oscillation frequency f_0. Analyzed in terms of phase, an oscillator can be seen as a time-phase integrator:

$$\Phi_{VCO}(t) = \Phi_{out}(t) = \int 2\pi(K_{DCO}\mathrm{V}_{ctrl}(t) + f_0)\mathrm{dt} \tag{7}$$

In the s-domain, where frequency offsets will be represented via initial conditions for modeling purposes, the VCO transfer function is therefore:

$$\mathrm{H}_{VCO}(s) = \frac{\Phi_{VCO}(s)}{\mathrm{V}_{ctrl}(s)} = \frac{\Phi_{out}(s)}{\mathrm{V}_{ctrl}(s)} = \frac{2\pi K_{DCO}}{s} \tag{8}$$

2.1.6 Continuous PLL Transfer function

Now that the continuous PLL synthesizer is understood at a component level, the closed loop dynamics of the PLL can be analyzed. First the PLL loop gain is determined:

$$\mathrm{L}(s) = \mathrm{H}_{LF}(s)\mathrm{H}_{VCO}(s)\mathrm{H}_{div}(s) = \frac{2\pi K_{VCO}}{\mathrm{N}} \frac{1}{s} \frac{\sum_{j=0}^{Z} b_j s^j}{\sum_{k=0}^{P} a_k s^k} \tag{9}$$

With the phase detector as the feedback summation point, the closed loop response of the PLL from reference to output is in equation 10.

$$\mathrm{T}(s) = \frac{\Phi_{out}(s)}{\Phi_{ref}(s)} = \frac{2\pi K_{VCO} \sum_{j=0}^{Z} b_j s^j}{\sum_{k=0}^{P} a_k s^{k+1} + \frac{2\pi K_{VCO}}{\mathrm{N}} \sum_{j=0}^{Z} b_j s^j} = \mathrm{N}\frac{\mathrm{L}(s)}{1 + \mathrm{L}(s)} \tag{10}$$

2.2 ADPLL - digital, discretized PLL Model

Based on the continuous PLL theory, a model for digital, discrete time sampled PLLs (i.e. ADPLLs) can be adapted. The general approach here is to utilize the bilinear transformation between continuous s-domain models to the discrete z-domain models and vice-versa. As commonly cited in PLL literature from a seminal paper by Gardner [5], if the PLL sampling frequency $f_s > 10 \cdot \mathrm{BW}_{loop}$, where BW_{loop} is the PLL loop bandwidth, the granularity effects due to time-sampling can be neglected. Thus the design methods established in this paper are predicated on $f_s > 10 \cdot \mathrm{BW}_{loop}$.

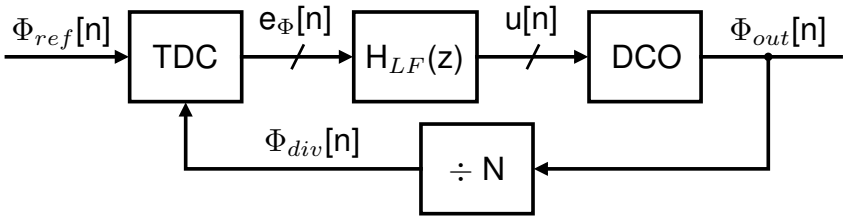


Figure 3: Basic ADPLL.

The basic architecture of an ADPLL is shown in figure 3. Here, compared to the continuous PLL of figure 2, the phase detector has been replaced with a time to digital converter (TDC), the loop filter $\mathrm{H}_{LF}(s)$ with a discrete-time loop filter $\mathrm{H}_{LF}(z)$, and the VCO with a digitally controlled oscillator (DCO). In this architecture, all components are fully or partially digital in implementation.

2.2.1 Divider

A digital divider functions by counting input cycles. With a divider modulus N, the output of the divider will have an active edge transition (considered to be rising edge as shown in figure 4) every N-cycles. Phase information is inferred from active edge timing, which occurs with time interval N/f_{osc}, and is equal to the point at which output phase equals a multiple of 2π. Thus a digital divider does not provider continuous phase information, but rather a sampled phase signal with rate f_{osc}/N.

For PLL transfer function modeling, a digital divider behaves identically to the continuous case:

$$\Phi_{div}[n] = \frac{\Phi_{out}[n]}{\mathrm{N}} \tag{11}$$

Application of the z- and s-domain transformations:

$$\mathrm{H}_{div}(z) = \mathrm{H}_{div}(s) = \frac{\Phi_{div}(z)}{\Phi_{out}(z)} = \frac{1}{\mathrm{N}} \tag{12}$$

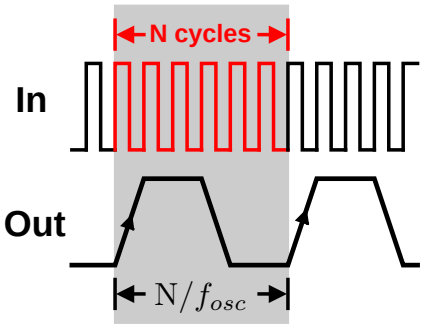


Figure 4: Digital divider signals.

2.2.2 TDC

The TDC is a digital, quantized representation of the phase detector. It takes input phase signals Φ_{div}[n] and Φ_{ref}[n], and outputs a digital phase error word $e_\Phi[n]$. Figure 5 shows the basic TDC model architecture. Being digitized, a TDC will have limited resolution in phase, equivalent to M steps per reference cycle. This is a minimum step size in time of Δt_{step} = $1/Mf_{ref}$. Since the output of the TDC is digital, the model applies a scale factor M/2π and floor rounding, so 1 least significant bit (LSB) of $e_\Phi[n]$ equates to Δt_{step} timing error between Φ_{div}[n] and Φ_{ref}[n].

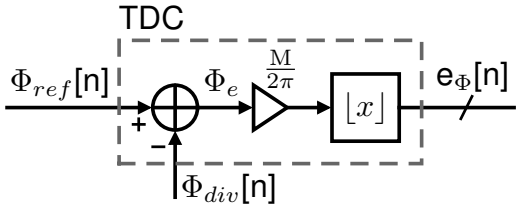


Figure 5: TDC model.

In sampled-time equation form:

$$e_\Phi[n] = \left\lfloor \frac{\mathrm{M}}{2\pi}(\Phi_{ref}[n] - \Phi_{div}[n]) \right\rfloor \tag{13}$$

For purposes of PLL loop gain calculation, the TDC z- and s-domain representation is equation 15, which accounts for phase-to-digital domain conversion gain. Effects of quantization will be handled in section 2.3.

$$e_\Phi(z) = \frac{\mathrm{M}}{2\pi}(\Phi_{ref}(z) - \Phi_{div}(z)) \tag{14}$$

$$\mathrm{H}_{TDC}(z) = \mathrm{H}_{TDC}(s) = \frac{\mathrm{M}}{2\pi} \tag{15}$$

2.2.3 Discrete-time loop filter

The discrete-time loop filter design will be derived from the continuous canonical loop filter (equation 6) via application of a s-to-z domain transformation. The bilinear transform [19] allows for such conversion from continuous transfer function to discrete representation. However, in the case presented in this work, where a high degree of sampling ($f_s > 10 \cdot \mathrm{BW}_{loop}$) is employed, a simpler transformation is permissible, derived through Taylor series approximation of z=$re^{s\Delta T}$ for values on the unit circle, i.e. r=1. This will be referred to as the approximate bilinear transform in this work. Given the $1/\Delta T_s$=f_{ref} as the relation for sampling rate, then:

$$\begin{aligned} z^{-1} &= e^{-s\Delta T_s} && \text{(definition of z-space on unit circle)} \\ &= \sum_{k=0}^{\infty} \frac{(-s\Delta T_s)^k}{k!} && \text{(exponential Taylor series)} \\ &\approx 1 - s\Delta T_s && \text{(if } |s\Delta T_s| = 2\pi \mathrm{BW}_{loop} \cdot \Delta T_s << 1) \end{aligned}$$

Thus the s-to-z and z-to-s identities for the approximated bilinear transform are:

$$z^{-1} = 1 - s\Delta T_s \tag{16}$$

$$s = \frac{1}{\Delta T_s}(1 - z^{-1}) \tag{17}$$

Applying 17 to equation 6 yields the z-domain loop filter:

$$\mathrm{H}_{LF}(z) = \mathrm{H}_{LF}(s)|_{s=\frac{1}{\Delta T_s}(1-z^{-1})} = \left.\frac{\sum_{j=0}^{Z} b_j s^j}{\sum_{k=0}^{P} a_k s^k}\right|_{s=\frac{1}{\Delta T_s}(1-z^{-1})} \tag{18}$$

$$= \frac{\sum_{j=0}^{Z} b_j (1 - z^{-1})^j}{\sum_{k=0}^{P} a_k (1 - z^{-1})^k} \tag{19}$$

Equation 19 is transformed to a digitally implementable representation by reorganizing into the canonical representation of 20, which then determines the tap coefficients for the sampled-time difference equation 21.

$$\mathrm{H}_{LF}(z) = \frac{\sum_{j=0}^{P} b'_j z^{-j}}{1 + \sum_{k=1}^{Z} a'_k z^{-k}} \tag{20}$$

$$y[n] = -\sum_{k=1}^{P} a'_k y[n-k] + \sum_{j=0}^{Z} b'_j x[n-j] \tag{21}$$

The obtained difference equation is directly implementable in digital hardware with a direct from I IIR filter [18] shown in figure 6. The filter coefficients $\{a'_1, ..., a'_P\}$ and $\{b'_0, ..., b'_Z\}$ must be quantized into finite resolution fixed point words for a complete digital implementation.

The delay elements (z^{-1} blocks) are implementable digitally as registers, the coefficient gains are implementable with array multipliers and the adders are implementable with digital adders. Effects of quantization will be discussed in section 2.3.

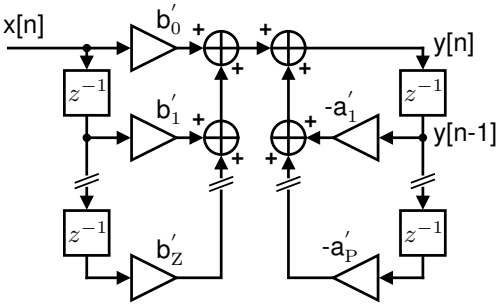


Figure 6: Direct form I implementation of IIR filter.

2.2.4 DCO

The digitally controlled oscillator varies from a VCO by only accepting a digital frequency tuning signal, called the oscillator tuning word (OTW). A DCO modeled in discrete time as a recursive phase integrator, dependent on (a) the DCO gain K_{DCO}, equal to the frequency tuning of the oscillator per LSB of the OTW, (b) the current state of the OTW u[n], and (c) the PLL sampling period T=f_{ref}^{-1}.

$$\Phi_{out}[n] = \Phi_{out}[n-1] + 2\pi K_{DCO} u[n] \Delta T_s \tag{22}$$

Application of the z-transform yields equation 23, and successive application of the bilinear transform to the DCO transfer function yields 24.

$$\mathrm{H}_{DCO}(z) = \frac{\Phi_{out}(z)}{u(z)} = \frac{2\pi K_{DCO} \Delta T_s}{1 - z^{-1}} \tag{23}$$

$$\mathrm{H}_{DCO}(s) = \frac{\Phi_{out}(s)}{u(s)} = \frac{2\pi K_{DCO} \Delta T_s}{1 - (1 - s\Delta T_s)} = \frac{2\pi K_{DCO}}{s} \tag{24}$$

2.2.5 Discrete-time PLL transfer function

The transfer function for the discrete-time PLL of figure 7 can be computed in the z-domain, and also approximated continuously. The open loop z-domain transfer function is:

$$\mathrm{L}(z) = \mathrm{H}_{TDC}(z)\mathrm{H}_{LF}(z)\mathrm{H}_{DCO}(z)\mathrm{H}_{DIV}(z) = \frac{\mathrm{M}}{\mathrm{N}} \frac{K_{DCO}\Delta T_s}{(1 - z^{-1})} \frac{\sum_{j=0}^{Z} b_j (1 - z^{-1})^j}{\sum_{k=0}^{P} a_k (1 - z^{-1})^k} \tag{25}$$

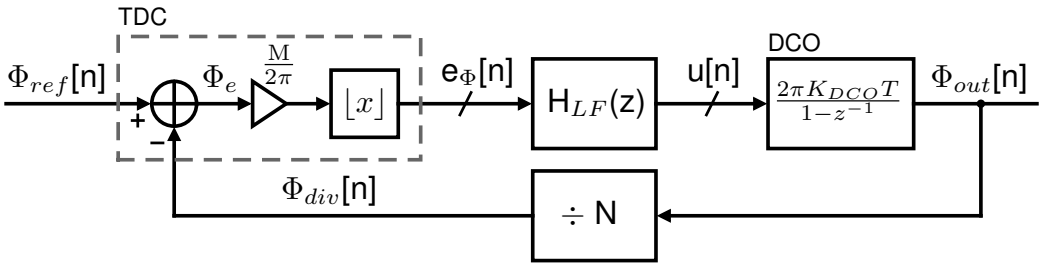


Figure 7: Discrete time PLL model.

The closed loop z-domain PLL phase transfer function is:

$$\mathrm{T}(z) = \frac{\Phi_{out}(z)}{\Phi_{ref}(z)} = \frac{\mathrm{M}K_{DCO}\Delta T_s \sum_{j=0}^{Z} b_j (1-z^{-1})^j}{\sum_{k=0}^{P} a_k (1-z^{-1})^{k+1} + K_{DCO}\Delta T_s \frac{\mathrm{M}}{\mathrm{N}} \sum_{j=0}^{Z} b_j (1-z^{-1})^j} \tag{26}$$

The s-domain approximation of the transfer function is:

$$\mathrm{L}(s) = \mathrm{H}_{TDC}(s)\mathrm{H}_{LF}(s)\mathrm{H}_{DCO}(s)\mathrm{H}_{DIV}(s) = \frac{\mathrm{M}}{\mathrm{N}} \frac{K_{DCO}}{s} \frac{\sum_{j=0}^{Z} b_j s^j}{\sum_{k=0}^{P} a_k s^k} \tag{27}$$

And in closed loop configuration the s-domain PLL phase transfer function is:

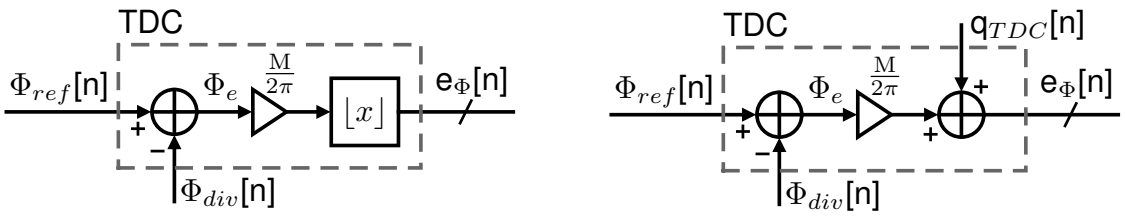
$$\mathrm{T}(s) = \frac{\Phi_{out}(s)}{\Phi_{ref}(s)} = \frac{\mathrm{M}K_{DCO} \sum_{j=0}^{Z} b_j s^j}{\sum_{k=0}^{P} a_k s^{k+1} 1 + \frac{\mathrm{M}}{\mathrm{N}} K_{DCO} \sum_{j=0}^{Z} b_j s^j} = \mathrm{N}\frac{\mathrm{L}(s)}{1+\mathrm{L}(s)} \tag{28}$$

2.3 ADPLL Noise Model

The noise in the discrete-time ADPLL result from quantization and from stochastic sources. Quantization results from round off errors introduced in the digitization of PLL components (in the TDC, loop filter, and DCO). Stochastic noise results from thermal and flicker noise in the PLL components when considered in an analog viewpoint (present in the DCO, divider and TDC). The noise generated by these quantization sources will be discussed in the following sections, based on a modeling and noise analysis approach from [15].

2.3.1 TDC noise

The predominant phase noise source in the TDC is due to quantization. A straightforward approach to model quantization noise is to utilize the model of figure 8b to represent quantization.



(a) TDC Model. **(b)** TDC additive noise model.

Figure 8: TDC quantization noise models.

Using this model, the quantized signal e_Φ[n] is the sum of the its unquantized representation $\Phi_e \frac{M}{2\pi}$ with a quantization error signal $q_{TDC}[n]$. Figure 9 illustrates this process.

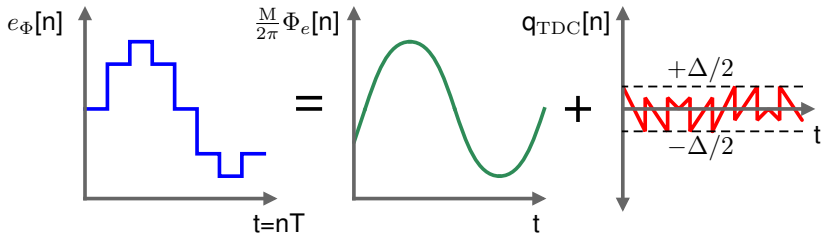


Figure 9: Quantization as via additive error signal.

The quantization noise signal has the statistical property that it is uniformly distributed in the range $[-\Delta/2, \Delta/2]$, i.e. $P_q(Q = q) = \mathrm{U}(-\Delta/2, \Delta/2)$ if Δ is the quantization step size. The power of the TDC quantization noise signal is:

$$\sigma^2_{q_{TDC}} = \int_{-\infty}^{\infty} q^2 P_q(Q = q) dq = \int_{-\Delta/2}^{\Delta/2} \frac{q^2}{\Delta} dq = \frac{\Delta^2}{12} \tag{29}$$

Since e_Φ[n] is a digital signal, the minimum step size is Δ=1 LSB. The TDC quantization noise power is therefore $\sigma^2_{q_{TDC}} = 1/12$ LSB^2. The po If the quantization noise is assumed to be white, and the TDC is sampled at f_{ref}, the quantization PSD is:

$$S_{qn_{TDC}}(f) = \frac{P_{q_{TDC}}}{\Delta f} = \frac{\sigma^2_{q_{TDC}}}{f_{ref}} = \frac{\Delta^2}{12 f_{ref}} = \frac{1}{12 f_{ref}} \quad \frac{[\mathrm{LSB}]^2}{[\mathrm{Hz}]} \tag{30}$$

2.3.2 DCO noise

Noise in a DCO is resulting from (a) quantization of the oscillator tuning word u[n], and (b) from thermal and stochastic sources. In the digital PLL, the OTW quantization occurs in the loop filter, so this will be analyzed in the later loop filter section (2.3.4). Thus oscillator thermal/stochastic noise will be considered, based on Leeson's model for oscillator phase noise [10]. Leeson's model considers noise power density at an offset Δf from the oscillator tone (carrier). Noise power density is represented with the function $\mathcal{L}(\Delta f)$, which is the noise

power density normalized to the power of the oscillator carrier tone, in other words in units of dBc/Hz. Leeson's model divides phase noise into three regions, illustrated in figure 10a: (1) flicker-noise dominated, with a slope of -30 dB/decade, (2) white frequency-noise dominated, with -20 dB per decade, and (3) a flat region, limited by the thermal noise floor or amplitude noise. It is noted that phase noise components are at frequencies different than the carrier, hence are orthogonal, and can be treated as independent components that are added to the main carrier frequency for analysis. Figure 10b demonstrates the application of this principle for modeling of the DCO phase noise $\Phi_{n_{DCO}}$ as an additive process to the oscillator phase Φ_{osc}, thus $\Phi_{out} = \Phi_{osc} + \Phi_{n_{DCO}}$.

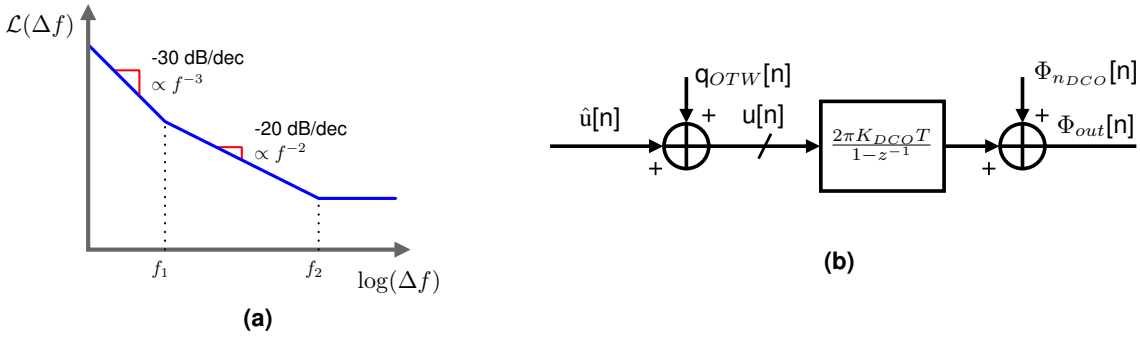


Figure 10: (a) Leeson model for phase noise, **(b)** DCO additive noise model.

The equation for $\mathcal{L}(\Delta f)$ (from [9]) is in 31, and is dependent on the temperature T, excess noise factor F, oscillator power P, oscillator Q factor, and the transition frequencies f_1 and f_2 that separate the different noise regions. It is of interest to note that the phase noise relative to the carrier will increase as power decreases, which provides challenge for creating low power oscillators with acceptable phase noise characteristics.

$$\mathcal{L}(\Delta f) = 10\log_{10}\left[\frac{2\mathrm{F}k_B\mathrm{T}}{\mathrm{P}}\left(1+\left(\frac{f_2}{2Q\Delta f}\right)^2\right)\left(1+\frac{f_1}{|\Delta f|}\right)\right] = S_{\Phi n_{DCO}}(\Delta f) \qquad (31)$$

For notational consistency, the following redefinition is used in the remainder of this paper: $S_{\Phi n_{DCO}}(f) = \mathcal{L}(\Delta f)|_{\Delta f = f}$

2.3.3 Divider noise

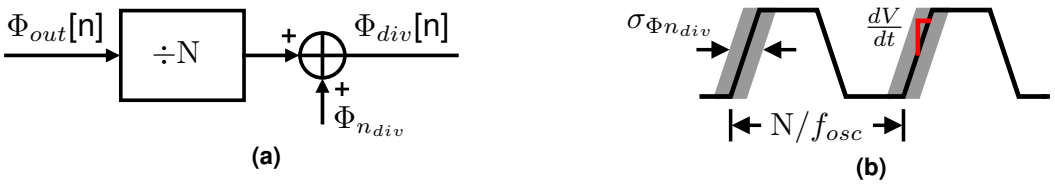


Figure 11: (a) Divider noise model, **(b)** Digital divider output jitter.

Divider noise is manifested as jitter (with RMS distribution in time of $\sigma_{tn_{div}}$) on the divider output. If the divider is a digital circuit, with edge rate dV/dt, and subject to thermal noise in

the form of an additive voltage v_n, with noise power of $\sigma^2_{v_n}$, the divider phase noise power added to the divider output is:

$$\sigma^2_{\Phi n_{div}} = \omega^2_{ref}\sigma^2_{tn_{div}} = \omega^2_{ref}\left(\frac{dV}{dt}\right)^{-2}\sigma^2_{v_n} \tag{32}$$

At lock, the output of a digital divider will have an update rate $f_{osc}/\mathrm{N} \approx f_{ref}$, which can be treated as the sampling rate of the output phase signal $\Phi_{div}[n]$ as mentioned in section 2.2.1. Thus if the divider phase noise power is confined into a bandwidth equal to f_{ref}, the spectral density of divider noise is:

$$S_{\Phi n_{div}}(f) = \frac{\sigma^2_{\Phi n_{div}}}{f_{ref}} = 2\pi\omega_{ref}\sigma^2_{tn_{div}} = 2\pi\omega_{ref}\left(\frac{dV}{dt}\right)^{-2}\sigma^2_{v_n} \quad \frac{[\mathrm{rad}]^2}{[\mathrm{Hz}]} \tag{33}$$

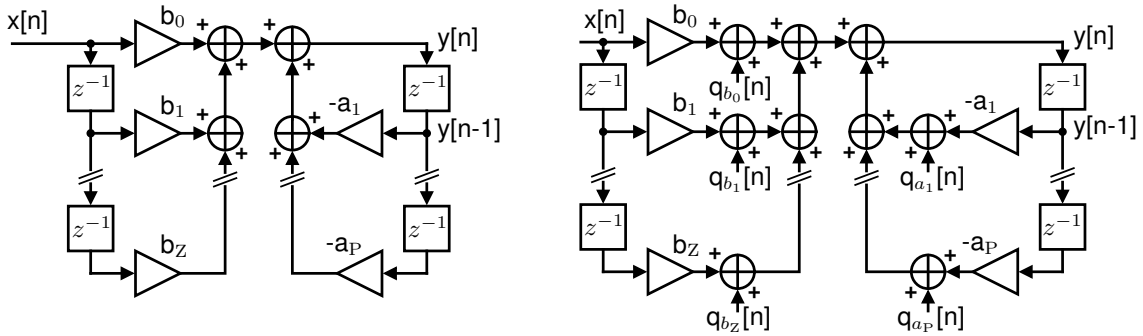
2.3.4 Loop filter noise - direct form I

In a digital loop filter, quantization noise arises from rounding errors due to finite precision in the arithmetic circuits that implement the filter. Quantization noise power here will be derived under the assumption of a direct form I filter implementation, with B bits in each fixed point word throughout the loop filter. In a digital implementation of the canonical z-domain transfer function 34 as the direct form I structure of figure 12a, delays are constructed using registers, adders with digital adders, and the filter coefficient gain terms $\{a_1, ...a_N; b_0, ..., b_M\}$ with digital multipliers. The registers and adders do not introduce extra round-off error beyond that already existing (if overflows do not occur). However, the multipliers will if the products resulting from two B bit words (nominally 2B bits), are mapped back onto B bit words.

$$\mathrm{H}_{LF}(z) = \frac{\sum_{j=0}^{Z} b_j z^{-j}}{1 + \sum_{k=1}^{P} a_k z^{-k}} \tag{34}$$

Quantization in this case can be represented by adding a quantization error signal q_x[n] to the result of each ideal multiplication, as shown in figure 12b. This is the same approach for TDC quantization noise in section 2.3.1. The noise power associated with each q_x[n] is identical, with $\sigma^2_{qx} = 1/12\ \mathrm{LSB}^2$.

Assuming the quantization error signals of each multiplier are uncorrelated with all other multipliers, the output-referred noise power of the filter can be computed as the sum of the output-referred individual contributions. These contributions can be determined via solving for the transfer function from each source q_x[n] of each quantization noise to the output y[n]. In the case of the direct form I filter structure, all quantization sources q_x[n] have the same



(a) Ideal structure. **(b)** With added quantization error signals.

Figure 12: Direct form I filter implementation with quantization.

transfer characteristic to the output y[n] given in 35.

$$\frac{Y(z)}{Q_x(z)} = \frac{1}{1 + \sum_{k=1}^{P} a_k z^{-k}} \tag{35}$$

Applying the bilinear transform to 35, with high oversampling where N·BW$_{loop}$10 $< f_{ref}$, and N is the number of poles in the system.

$$\left.\frac{Y(z)}{Q_x(z)}\right|_{z^{-1}=1-sT} \approx \frac{1}{1 + \sum_{k=1}^{P} a_k - s\sum_{k=1}^{P} k a_k} \tag{36}$$

The output power spectral density is then for one error source is, confined to a bandwidth defined by the (sampling) reference frequency f_{ref}:

$$S_{qx}(f) = \frac{\sigma_{qx}^2}{f_{ref}} \left|\frac{Y(s)}{Q_x(s)}\right|^2_{s=j2\pi f} \approx \frac{1}{12 f_{ref}} \left|\frac{1}{1 + \sum_{k=1}^{P} a_k - j2\pi f \sum_{k=1}^{P} k a_k}\right|^2 \quad \frac{[\text{LSB}]^2}{[\text{Hz}]} \tag{37}$$

Given P poles and Z zeros in the filter, the total output quantization PSD of the filter is 38. The total loop filter noise PSD linearly scales with the number of multipliers in the direct form I filter implementation.

$$S_{qn_{LF}}(f) = (P + Z + 1) S_{qx}(f) \quad \frac{[\text{LSB}]^2}{[\text{Hz}]} \tag{38}$$

2.3.5 PLL noise sensitivity transfer functions

Having developed models for noise of generated by each PLL component, noise sensitivity transfer functions must be computer to refer each noise source to the PLL output in terms of phase. In the developed noise theory, thus far all noise sources have been modeled as additive phase components. The full system model illustrating this is in figure 13.

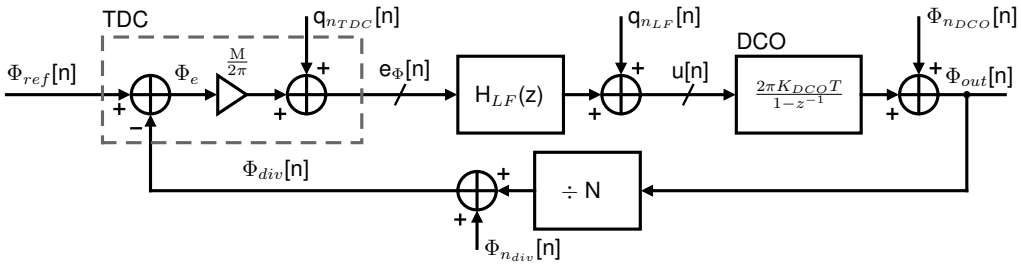


Figure 13: Full PLL additive noise model.

Following the approach of [15], it is useful to define a transfer function $\hat{\mathrm{T}}(s)$ which characterizes the normalized closed loop response from reference to output of the PLL. $\hat{\mathrm{T}}(s)$ is defined in terms of the loop gain L(s).

$$\hat{\mathrm{T}}(s) = \frac{\mathrm{L}(s)}{1+\mathrm{L}(s)} \quad \text{s.t.} \quad \mathrm{T}(s) = \frac{\Phi_{out}}{\Phi_{ref}} = \mathrm{N}\hat{\mathrm{T}}(s) \tag{39}$$

Solving for the closed transfer functions between each $q_{n_{TDC}}$, $q_{n_{LF}}$, $\Phi_{n_{DCO}}$ and $\Phi_{n_{div}}$ to the output Φ_{out} utilizing s-domain approximations yields equations 40-43.

$$\frac{\Phi_{out}(s)}{q_{n_{TDC}}(s)} = \frac{2\pi\frac{K_{DCO}}{s}\mathrm{H}_{LF}(s)}{1+\mathrm{L}(s)} = 2\pi\frac{\mathrm{N}}{\mathrm{M}}\frac{\mathrm{L}(s)}{1+\mathrm{L}(s)} = 2\pi\frac{\mathrm{N}}{\mathrm{M}}\hat{\mathrm{T}}(s) \tag{40}$$

$$\frac{\Phi_{out}(s)}{\Phi_{n_{DCO}}(s)} = \frac{1}{1+\mathrm{L}(s)} = 1-\hat{\mathrm{T}}(s) \tag{41}$$

$$\frac{\Phi_{out}(s)}{q_{n_{LF}}(s)} = \frac{2\pi\frac{K_{DCO}}{s}}{1+\mathrm{L}(s)} = 2\pi\frac{K_{DCO}}{s}(1-\hat{\mathrm{T}}(s)) \tag{42}$$

$$\frac{\Phi_{out}(s)}{\Phi_{n_{div}}(s)} = \frac{\mathrm{M}\frac{K_{DCO}}{s}\mathrm{H}_{LF}(s)}{1+\mathrm{L}(s)} = \mathrm{N}\frac{\mathrm{L}(s)}{1+\mathrm{L}(s)} = \mathrm{N}\hat{\mathrm{T}}(s) \tag{43}$$

2.3.6 PLL phase signal and output PSD relationship for noise

When analyzing PLL noise, the noise power spectral density of the PLL output is of most interest. Up to this point, noise has been defined in terms of noise phase signal Φ_n, or an unwanted added component to the oscillator phase signal $\Phi_{osc} = \omega_{osc}t$. The PLL output phase signal Φ_{out} is thus:

$$\Phi_{out}(t) = \Phi_{osc}(t) + \Phi_n(t) = \omega_{osc}t + \Phi_n(t) \tag{44}$$

Computation of PSD requires the PLL output voltage waveforms. These here will be defined in terms of complex exponentials. Given an oscillation amplitude A_0:

$$V_{out} = \Re\left(A_0 e^{j\Phi_{out}(t)}\right) = \Re\left(A_0 e^{j\omega_{osc}t} e^{j\Phi_n(t)}\right) \tag{45}$$

Assuming the phase noise signal is zero mean, $\mathbb{E}[\Phi_n(t)] = 0$, and the power of phase noise signal is small, $\text{Var}[\Phi_n(t)] << 1$, then the approximation $e^{j\Phi_n(t)} = 1 + j\Phi_n(t)$ can be applied by truncating the exponential Taylor series expansion.

$$V_{out} = \Re\left(A_0 e^{j\omega_{osc}t} e^{j\Phi_n(t)}\right) = \Re\left(A_0 e^{j\omega_{osc}t} + j\Phi_n(t) A_0 e^{j\omega_{osc}t}\right) \tag{46}$$

$$= A_0 \cos(\omega_{osc}t) - \Phi_n(t) A_0 \sin(\omega_{osc}t) \tag{47}$$

The result is a carrier cosine signal, and an orthogonal sine signal modulated by the phase noise Φ_n. From this, the spectral density of the phase noise relative to the carrier can be estimated. The power spectral density $S_{V_{out}}$ is computed in 48-50. Due to orthogonality of the sine/cosine components of 47, the cross terms that appear in the PSD are zero.

$$S_{V_{out}}(f) = \lim_{\Delta T \to \infty} \frac{1}{\Delta T} |\mathcal{F}\{V_{out}(t) \cdot \text{rect}(t/\Delta T)\}|^2 \tag{48}$$

$$= \lim_{\Delta T \to \infty} \frac{A_0^2}{\Delta T} |\mathcal{F}\{\cos(\omega_{osc}t) \cdot \text{rect}(t/\Delta T)\}|^2 \tag{49}$$

$$+ \lim_{\Delta T \to \infty} \frac{A_0^2}{\Delta T} |\mathcal{F}\{\Phi_n(t) \cdot \text{rect}(t/\Delta T)\} * \mathcal{F}\{\sin(\omega_{osc}t) \cdot \text{rect}(t/\Delta T)\}|^2 \tag{50}$$

The noise power spectral density $\mathcal{L}(\Delta f)$ is defined as the noise PSD at offset Δf from the carrier frequency f_{osc}, normalized to the carrier power. Here the PSD carrier component is given by 49, and the noise component by 50. Shifting 50 by ω_{osc} and performing normalization for carrier power results in:

$$\mathcal{L}(\Delta f) = \lim_{\Delta T \to \infty} \frac{1}{\Delta T} |\mathcal{F}\{\Phi_n(t) \cdot \text{rect}(t/\Delta T)\}|^2 \Big|_{f=\Delta f} = S_{\Phi_n}(\Delta f) \tag{51}$$

Thus, the PLL output noise PSD relative to the carrier $\mathcal{L}(\Delta f)$ is equal to the PSD of the phase noise signal $\Phi_n(t)$, $S_{\Phi_n}(\Delta f)$, provided $\text{Var}[\Phi_n(t)] << 1$.

2.3.7 PLL output-referred noise PSD

In terms of analysis, PLL noise PSD referred to the PLL output is of most interest. Thus far the following have been established: (a) noise spectrum generated by each individual PLL component, (b) the PLL phase noise sensitivity functions, and (c) the relationship between PLL output PSD and output phase noise. These can be combined to provide a final result for total PLL output-referred noise PSD. An assumption here is all noise sources are uncorrelated, so their independent noise power contributions may be summed to find the total noise PSD. The PLL output phase noise PSD for each noise source is simply found by multiplying magnitude

squared of the respective noise sensitivity function with the noise source PSD. Thus:

$$S_{\Phi n_{TDC,out}}(f) = S_{qn_{TDC}}(f)\left|\frac{\Phi_{out}(f)}{q_{n_{TDC}}(f)}\right|^2 = \frac{1}{12f_{ref}}\left|2\pi\frac{\mathrm{N}}{\mathrm{M}}\hat{\mathrm{T}}(f)\right|^2 \tag{52}$$

$$S_{\Phi n_{DCO,out}}(f) = S_{\Phi n_{DCO}}(f)\left|\frac{\Phi_{out}(f)}{\Phi_{n_{DCO}}(f)}\right|^2 = \frac{S_{0\Phi n_{DCO}}}{f^2}\left|1-\hat{\mathrm{T}}(f)\right|^2 \tag{53}$$

$$S_{\Phi n_{LF,out}}(f) = S_{qn_{LF}}(f)\left|\frac{\Phi_{out}(f)}{q_{n_{LF}}(f)}\right|^2 \approx \frac{K_{DCO}^2}{12f_{ref}f^2}\frac{(P+Z+1)|1-\hat{\mathrm{T}}(f)|^2}{\left|1+\sum_{k=1}^{P}a_k - j2\pi f\sum_{k=1}^{P}ka_k\right|^2} \tag{54}$$

$$S_{\Phi n_{div,out}}(f) = S_{\Phi n_{div}}(f)\left|\frac{\Phi_{out}(f)}{\Phi_{n_{div}}(f)}\right|^2 = f_{ref}\left|2\pi\sigma_{tn_{div}}\mathrm{N}\hat{\mathrm{T}}(f)\right|^2 \tag{55}$$

The output noise PSD at offset Δf relative to the carrier normalized to carrier power of PLL will be:

$$\mathcal{L}(\Delta f) = S_{\Phi n_{TDC,out}}(\Delta f) + S_{\Phi n_{DCO,out}}(\Delta f) + S_{\Phi n_{LF,out}}(\Delta f) + S_{\Phi n_{div,out}}(\Delta f) \tag{56}$$

2.4 Modified ADPLL architecture for low TDC resolutions

When a low resolution TDC is used in a PLL, the resolution in phase will limit the ability to detect phase (or equivalently frequency) errors at the output in steady state. With M TDC steps per reference cycle, a divider ratio of N and a frequency error at the PLL output of Δf, the time needed for enough phase error to accumulate for a 1 LSB change of the TDC output is:

$$t = \frac{\mathrm{N}}{\mathrm{M}\Delta f} \tag{57}$$

If M $<$ N, the response time t of the TDC to frequency disturbance Δf is greater than the period of that disturbance $1/\Delta f$. Under these conditions, the PLL will have impaired ability to correct phase error and the phase noise performance will be degraded. In order to introduce higher resolution phase-feedback in steady state, a bang bang phase detector (BBPD) is added in parallel with the TDC, as in figure 14, weighted with a gain K_{bb}. The BBPD outputs a 1 if the divider signal is late relative to the clock, and -1 if it is early, with resolution limited only by jitter. Post the K_{bb} gain, the power of the BBPD signal is K_{bb}^2. To ensure the noise generated by the BBPD is less than that expected for the TDC from quantization, $K_{bb}^2 < 1/12$.

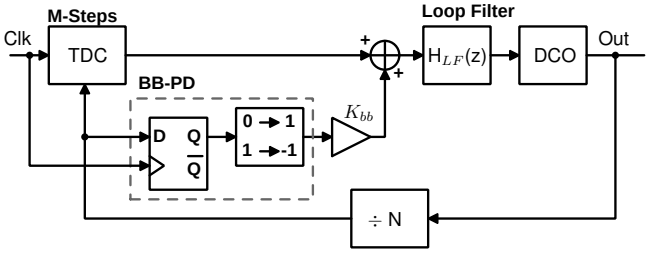


Figure 14: PLL with parallel bang-bang phase detector and TDC.

3 Loop Filter Design Automation

The automation approach for ADPLL loop filter design implemented in this work will be outlined here.

3.1 Design sequence

Design automation for ADPLL loop filter is implemented with a strategy that is illustrated in figure 15, that utilizes a continuous time-approximation based model of the PLL to generate a loop filter design which minimizes the total integrated phase noise power out of the PLL. The optimized continuous filter then undergoes discrete time conversion and mapping into digital representation of the design. Second order optimization is then applied to the discretized and digitized filter implementation, to minimize the effects of quantization error and filter design error due to finite word effects. A discrete-time, behavioral PLL simulator is then used to verify the filter design for proper lock-time, phase noise and stability. The following sections will detail these processes.

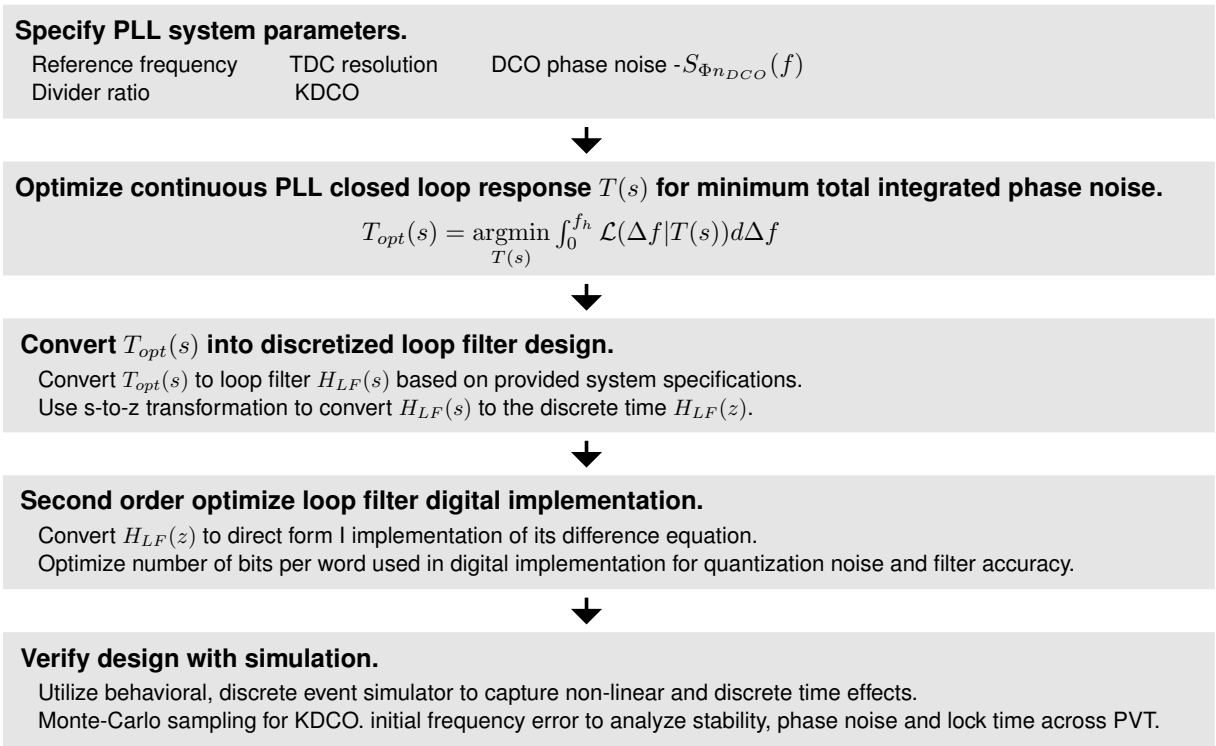


Figure 15: Filter design sequence.

3.2 Loop filter Prototype

The design automation approach developed utilizes a fixed loop filter as a prototype for all loop filter designs. This choice was derived from several criteria that were established for desirable ADPLL operation:

1. Zero steady state phase error, to ensure accuracy of synthesized frequency.
2. Loop filter output should remain constant if input (phase error) goes to 0.
3. Minimize complexity of implemented logic, i.e. minimize the number of poles and zeros.
4. Low pass response of PLL in closed-loop.

To satisfy criterion 1, the loop filter response must include an integrator (1/s) term in the transfer function [12]. A PLL with loop filter containing such an integrator is commonly classified as type II, and sans the integrator is type I [6]. A rational starting point is to consider a proportional-integral-derivative (PID) controller [13] for the loop filter, whose transfer function is given in equation 58. This filter contains coefficients K_d, K_p, K_i which set the gain of the derivative (s), proportional and integral (1/s) terms respectively.

$$\mathrm{H}_{LF}(s) = sK_d + K_p + \frac{K_i}{s} = \frac{K_d}{s}\left(s^2 + s\frac{K_p}{K_d} + \frac{K_i}{K_d}\right) \tag{58}$$

Application of the loop filter to the continuous ADPLL transfer function model from section 2.2.5 produces the PLL loop gain in 59 and the closed loop ADPLL response in 60.

$$\mathrm{L}(s) = \mathrm{H}_{TDC}(s)\mathrm{H}_{LF}(s)\mathrm{H}_{DCO}(s)\mathrm{H}_{DIV}(s) = \frac{\mathrm{M}}{\mathrm{N}}\frac{K_{DCO}K_d}{s^2}\left(s^2 + s\frac{K_p}{K_d} + \frac{K_i}{K_d}\right) \tag{59}$$

$$\mathrm{T}(s) = \frac{\Phi_{out}(s)}{\Phi_{ref}(s)} = \frac{\mathrm{M}K_{DCO}K_d\left(s^2K_d + sK_p + K_i\right)}{s^2\left(1 + \frac{\mathrm{M}}{\mathrm{N}}K_{DCO}K_d\right) + \frac{\mathrm{M}}{\mathrm{N}}K_{DCO}K_d\left(sK_p + K_i\right)} = \mathrm{N}\frac{\mathrm{L}(s)}{1 + \mathrm{L}(s)} \tag{60}$$

It should be noted that in the closed loop configuration, this PLL phase transfer function contains two poles and two zeros. This is not a low pass response as desired per criterion 4, needed for satisfactory PLL phase noise power spectrum as will later be discussed. In order achieve low pass operation, the derivative term K_d must be set to zero, yielding a proportional-integral (PI) controller for the loop filter:

$$\mathrm{T}(s) = \frac{\Phi_{out}(s)}{\Phi_{ref}(s)} = \frac{\mathrm{M}K_{DCO}\left(sK_p + K_i\right)}{s^2 + \frac{\mathrm{M}}{\mathrm{N}}K_{DCO}\left(sK_p + K_i\right)} = \mathrm{N}\frac{\mathrm{L}(s)}{1 + \mathrm{L}(s)} \tag{61}$$

Steady state zero phase error can be verified by solving the closed loop $\Phi_e(s)$ for s=0:

$$\Phi_e(s)|_{s=0} = \left(\Phi_{ref}(0) - \frac{\Phi_{out}(0)}{\mathrm{N}}\right) = \Phi_{ref}(0)\left(1 - \frac{\Phi_{out}(0)}{\mathrm{N}\Phi_{ref}(0)}\right) = \Phi_{ref}(0)\left(1 - \frac{\mathrm{N}}{\mathrm{N}}\right) = 0 \tag{62}$$

3.2.1 Continuous PI-loop filter design

Given a PI-controller loop filter, which can be optionally represented using a pole at zero and a zero with $\omega_z = K_i/K_p$:

$$\mathrm{H}_{LF}(s) = K_p + \frac{K_i}{s} = \frac{K_i}{s}\left(\frac{s}{\omega_z} + 1\right) \tag{63}$$

Selection of (not-necessarily optimal) PI controller gains can be easily derived from overall PLL settling time requirements. Suppose that settling time t_s is defined such that the PLL settles within $\pm\delta$ of the final value for a step input. If the initial and final PLL output frequencies are f_i and Nf_{ref}, and settling with $\pm f_{tol}$ is desired, $\delta = f_{tol}/|f_i - Nf_{ref}|$. Setting time is therefore:

$$t_s = -\tau \ln(\delta) \tag{64}$$

Thus, to find settling time, a value for the PLL time constant τ must be derived. Rewriting equation 61 with substitutions $\omega_z = K_i/K_p$ and $\mathrm{K} = MK_{DCO}K_i/\mathrm{N}$:

$$\frac{\Phi_{out}(s)}{\Phi_{ref}(s)} = \mathrm{N} \cdot \frac{s\frac{K}{\omega_z} + K}{s^2 + s\frac{K}{\omega_z} + K} \tag{65}$$

If the second order denominator can be redefined in terms of a natural frequency ω_n and damping ζ, such that:

$$s^2 + s\frac{K}{\omega_z} + K = s^2 + s2\zeta\omega_n + \omega_n^2 \tag{66}$$

It is then found that $\omega_n = \sqrt{K}$, and $\omega_z = \sqrt{K}/2\zeta$. The poles of equation 65 are then located at s = $\zeta\sqrt{K} \pm \sqrt{K}\sqrt{1-\zeta^2}$. The settling time of the PLL will be determined by the real portion of dominant pole of equation 65, specifically $\tau = 1/|\min(\Re(\{s_{p1}, s_{p2}\}))|$. Based on the pole-zero plot of figure 16, it can be observed that the dominant pole location is maximized with $\zeta = 1$. The pole-zero loci orientations are based on increasing ζ values. According to Razavi [21], ζ is usually "chosen to be $> \sqrt{2}/2$ or even 1 to avoid excessive ringing."

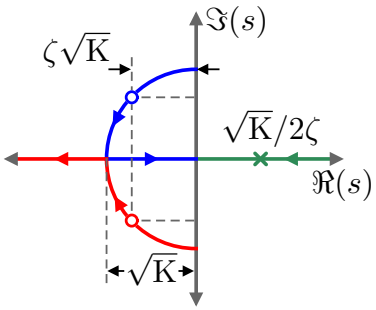


Figure 16: PI-controller PLL pole-zero locations.

To illustrate the effect of the damping coefficient ζ, figure 17 illustrates the example frequency and step responses of a PI-controlled PLL with N=1. Notice excessive peaking and ringing for $\zeta < \sqrt{2}/2$. The peaking observed in the frequency response is unavoidable with the PI-PLL due to the inherent zero in the transfer function. Its effect can be reduced with large ζ, however this will increase PLL settling time.

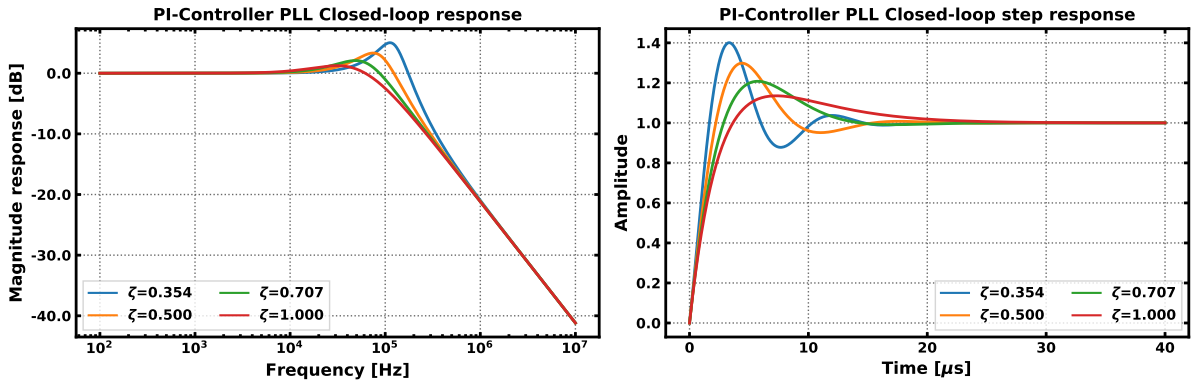


Figure 17: Example PI-PLL responses with varied ζ.

If ζ is constrained to ≤ 1:

$$\tau = \frac{1}{|\min(\Re(\{s_{p1}, s_{p2}\}))|} = \frac{1}{\zeta\sqrt{K}} \tag{67}$$

Thus:

$$t_s = \frac{-\ln(\delta)}{\zeta\sqrt{K}} = \frac{-\ln\left(\frac{f_{tol}}{|f_i - N f_{ref}|}\right)}{\zeta\sqrt{K}} \tag{68}$$

Based on specification for settling time and damping ζ, the values for K and ω_z can be determined. If K_{VCO} and N are also specified, the PI gain coefficients can be solved

additionally.

$$\omega_z = \frac{-\ln(\delta)}{2t_s} = \frac{-\ln\left(\frac{f_{tol}}{|f_i - N f_{ref}|}\right)}{2t_s} \tag{69}$$

$$K = \frac{\ln^2(\delta)}{\zeta^2 t_s^2} = \frac{\ln^2\left(\frac{f_{tol}}{|f_i - N f_{ref}|}\right)}{\zeta^2 t_s^2} \tag{70}$$

$$K_i = \frac{\mathrm{N}}{\mathrm{M}} \frac{\mathrm{K}}{K_{DCO}} \tag{71}$$

$$K_p = \frac{K_i}{\omega_z} \tag{72}$$

This controller has a predictable phase margin/stability

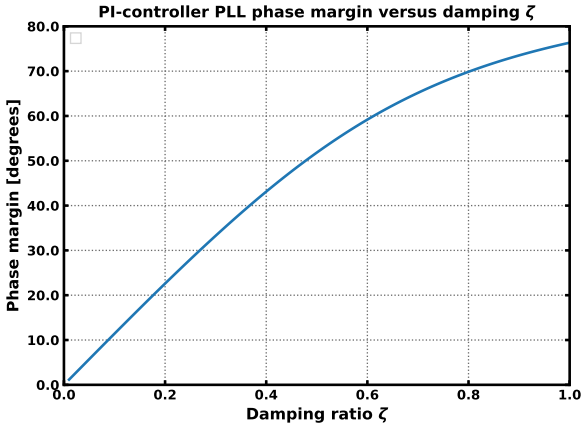


Figure 18: PI-controller PLL phase margin versus damping ratio.

3.2.2 PI-controller peaking compensation

To compensate for closed loop peaking, the original PI-controller loop filter of equation 63 can be modified with the addition of a single tunable pole at ω_p. The closed loop response becomes third order, which complicates direct analysis and design of the loop filter, but can be handled utilizing the numerical optimization approach described in this work. Not necessarily stable as closed loop configuration has 3 poles

$$\mathrm{H}_{LF}(s) = \frac{K_i}{s} \frac{\left(\frac{s}{\omega_z} + 1\right)}{\left(\frac{s}{\omega_p} + 1\right)} \tag{73}$$

3.2.3 Alternative PID controller permutations

If individual terms within the PID-controller are dropped, different controller permutations (PD, ID, PI, P, I, D) can be achieved. As mentioned before, inclusion of an integral term is needed to ensure the desired zero steady state error for a PLL, and the derivative term must be removed to achieve low pass response in the PLL. This leaves integral term only controller as the remaining candidate for PID controller design. Thus, setting the K_p and K_d terms of equation 60 to zero yields:

$$\frac{\Phi_{out}(s)}{\Phi_{ref}(s)} = \frac{2\pi K_{VCO} K_i}{s^2 + \frac{2\pi K_{VCO}}{\mathrm{N}} K_i} \tag{74}$$

This closed loop transfer function results in a pair of poles at $\pm\sqrt{2\pi K_{VCO} K_i / \mathrm{N}}$. This is not stable, as it can only be manifested as (1) a pair of poles on the imaginary axis, which is an oscillator, or (2) a real pole in the right-half plane and a real pole in the left-half plane, the former of which is not causally stable. Thus a PI-controller is the only viable PID-controller permutation for use in a PLL loop filter.

3.2.4 Discretized Loop Filter Prototype

Using the continuous filter discretization approach described in section 2.2.3 on the loop filter of equation 73 results in equation 75.

$$\mathrm{H}_{LF}(z) = \frac{K_i}{s} \frac{\left(\frac{s}{\omega_z} + 1\right)}{\left(\frac{s}{\omega_p} + 1\right)}\Bigg|_{s=\frac{1}{\Delta T_s}(1-z^{-1})} = k_i \Delta T_s \frac{\omega_p}{\omega_z} \frac{(1 + \omega_z \Delta T_s) - z^{-1}}{(1 + \omega_p \Delta T_s) - z^{-1}(2 + \omega_p \Delta T_s) + z^{-2}} \tag{75}$$

The transformation of 75 into a digitally implementable design as a direct form 1 IIR filter shown in figure 19. Its filter coefficients given by equations 76-77.

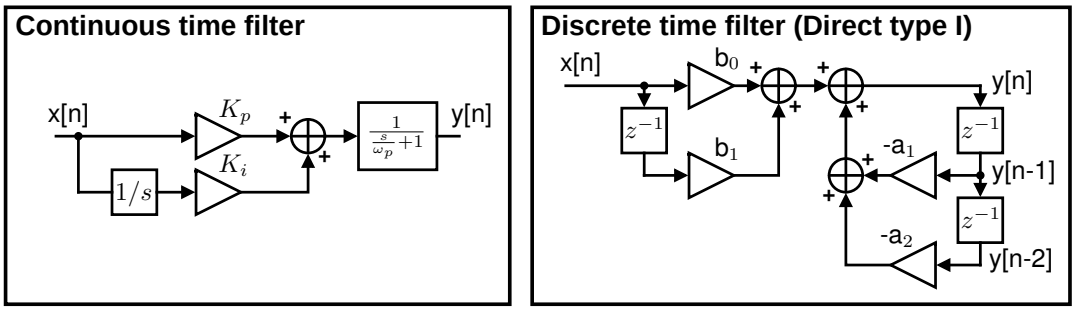


Figure 19: Implementation of filter.

$$a_1 = -\frac{2+\omega_p\Delta T_s}{1+\omega_p\Delta T_s} \qquad a_2 = \frac{1}{1+\omega_p\Delta T_s} \tag{76}$$

$$b_0 = \frac{K_i\omega_p\Delta T_s}{\omega_z}\frac{1+\omega_z\Delta T_s}{1+\omega_p\Delta T_s} \qquad b_1 = \frac{K_i\omega_p\Delta T_s}{\omega_z}\frac{1}{1+\omega_p\Delta T_s} \tag{77}$$

3.2.5 Prototype PLL response

Based on the prototype loop filter developed, the PLL closed loop response is developed for usage in the loop optimizer discussed in this work. Applying the discrete-time PLL transfer function model developed in section 2.2.5, the PLL loop gain is in equation 79, and the closed loop transfer function of the PLL is in 80.

$$\begin{aligned} \mathrm{L}(z) &= \mathrm{H}_{TDC}(z)\mathrm{H}_{LF}(z)\mathrm{H}_{DCO}(z)\mathrm{H}_{DIV}(z) \qquad (78)\\ &= 2\pi K_{DCO}K_i\Delta T_s^2\frac{\mathrm{M}}{\mathrm{N}}\frac{\omega_p}{\omega_z}\frac{(1+\omega_z\Delta T_s)-z^{-1}}{(1+\omega_p\Delta T_s)-z^{-1}(3+2\omega_p\Delta T_s)+z^{-2}(3+\omega_p\Delta T_s)-z^{-3}} \qquad (79)\end{aligned}$$

The closed loop z-domain PLL phase transfer function is:

$$\mathrm{T}(z) = \frac{\Phi_{out}(z)}{\Phi_{ref}(z)} = \frac{2\pi K_{DCO}K_i\Delta T_s^2\mathrm{M}\frac{\omega_p}{\omega_z}(1+\omega_z\Delta T_s)-z^{-1}}{\left((1+\omega_p\Delta T_s+2\pi K_{DCO}K_i\Delta T_s^2\frac{\mathrm{M}}{\mathrm{N}}\frac{\omega_p}{\omega_z}(1+\omega_z\Delta T_s))-z^{-1}(3+\right.\\ \left.2\omega_p\Delta T_s+2\pi K_{DCO}K_i\Delta T_s^2\frac{\mathrm{M}}{\mathrm{N}}\frac{\omega_p}{\omega_z})+z^{-2}(3+\omega_p\Delta T_s)-z^{-3}\right)} \tag{80}$$

Applying z-to-s domain transformation, the continuous s-domain approximation of the loop gain is given in equation 81, and the closed loop transfer function in equation 82.

$$\mathrm{L}(s) = \mathrm{H}_{TDC}(s)\mathrm{H}_{LF}(s)\mathrm{H}_{DCO}(s)\mathrm{H}_{DIV}(s) = \frac{\mathrm{M}}{\mathrm{N}}\frac{K_{DCO}K_i}{s^2}\frac{\left(\frac{s}{\omega_z}+1\right)}{\left(\frac{s}{\omega_p}+1\right)} \tag{81}$$

$$\mathrm{T}(s) = \frac{\Phi_{out}(s)}{\Phi_{ref}(s)} = \frac{\mathrm{M}K_{DCO}K_i\left(\frac{s}{\omega_z}+1\right)}{s^3\frac{1}{\omega_z}+s^2+\frac{\mathrm{M}}{\mathrm{N}}K_{DCO}K_i\left(\frac{s}{\omega_z}+1\right)} = \mathrm{N}\frac{\mathrm{L}(s)}{1+\mathrm{L}(s)} \tag{82}$$

4 Behavioral ADPLL simulation

To fully capture the effects of a discrete time PLL with digital quantization effects, the implementation of a behavioral, discrete event PLL simulator is described in the following. The simulator utilizes behavioral models to describe the individual components which comprise the PLL. These components are (1) a clock reference, (2) a TDC, (3) a loop filter, (4) a DCO, and (5) a divider. These behavioral models fully encapsulate effects of time-quantization and digitization. The simulator operates by iterating in fixed time steps of Δt=1/fs, where each node in the PLL is updated based upon the previous node values in a manner that is defined by each PLL component behavioral model. Each behavioral model is represented programmatically utilizing classes. In simulation, each component instance is represented by a object instance of the respective model class, constructed with any initial parameters (such as division ratio) that describe the component.

4.1 Simulation engine

The simulator engine for this work follows the sequence illustrated in figure 20 for running a simulation. Given specifications for simulation conditions (listed in the figure), the simulator accordingly initializes model objects that constitute the PLL components in simulation space. Then the simulation is run by entering a simulation loop.

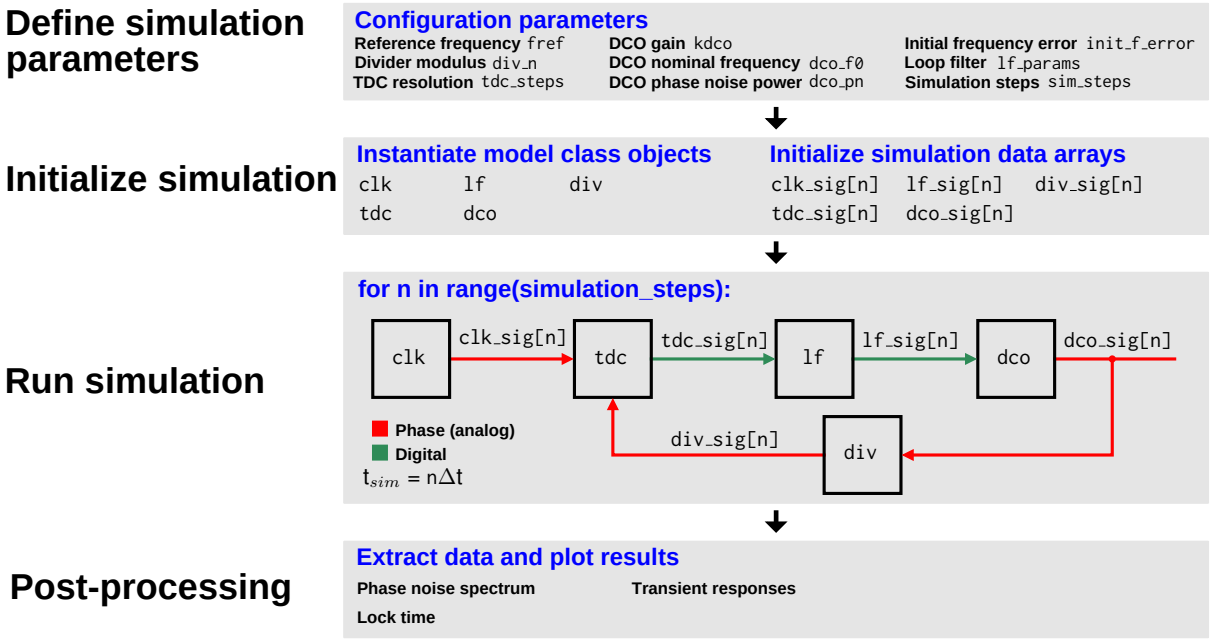


Figure 20: Simulation process.

The discrete event simulator loop is given in the following pseudocode of listing 1, with component model objects {`clk, tdc, lf, dco, div`}, and simulation data arrays {`clk_sig, tdc_sig, lf_sig, dco_sig, div_sig`}. The simulation operates with a fixed, discrete time step. Each model object is updated at each simulation step (loop iteration) utilizing the class method `update`, passing any relevant simulation data as arguments to the method. The output state of each component is saved into the respective array instance for each simulation step.

```
for n in range(simulation_steps):
    clk_sig[n] = clk.update()
    tdc_sig[n] = tdc.update(clk_sig[n-1], div_sig[n-1]))
    lf_sig[n]  = lf.update(tdc_sig[n-1], clk_sig[n-1]) #loop filter
    osc_sig[n] = dco.update(lf_sig[n-1])
    div_sig[n] = div.update(osc_sig[n-1], div_n)
```

Listing 1: PLL simulation loop Python pseudocode

After the simulation loop reaches completion, the results stored in the simulation data arrays can be post-processed to extract phase noise data, transient behavior and lock time. The following sections will discuss in more detail the implemented behavioral model classes and post-processing.

4.2 Clock behavioral model

An ideal behavioral clock model is utilized, given in the pseudocode of listing 2. The model is instantiated with the clock frequency `f` and the simulator time step `dt`. The model incrementing its phase every simulation step by a fixed amount $\Delta\Phi = 2\pi\texttt{f}\cdot\texttt{dt}$, as in 83. The model outputs an analog phase signal.

$$\Phi_{clk}[n] = \Phi_{clk}[n-1] + 2\pi\texttt{f}\cdot\texttt{dt} \tag{83}$$

```
class Clock:
  def __init__(self, f, dt):
    self.f = f      # clock frequency
    self.dt = dt    # simulation time step
    self.phase = 0.0  # clock phase state variable

  def update(self):
    self.phase += 2*pi*self.f*self.dt   # increment phase
    return self.phase
```

Listing 2: Ideal clock behavioral model Python pseudocode.

4.3 TDC behavioral model

The TDC behavioral model takes two analog inputs x and y that are in units of phase, and outputs a digital word that quantifies the phase separation of the signals. The model is instantiated with a resolution parameter tdc_steps, which defines the number of phase steps per cycle of the reference input x the TDC can resolve. The method round quantizes an floating point argument to the nearest integer.

$$\text{out}[n] = \left\lfloor 0.5 + \texttt{tdc\_steps}\frac{\text{x}[n] - \text{y}[n]}{2\pi} \right\rfloor \tag{84}$$

```
class TDC:
  def __init__(self, tdc_steps):
    self.tdc_steps = tdc_steps

  def update(x, y):
    ph_error = wrap(x-y)  # wraps phase to be within [0, 2*pi]
    return round(self.tdc_steps*(ph_error/(2*pi)))
```

Listing 3: TDC behavioral model Python pseudocode.

4.4 Loop filter behavioral model

The loop filter model implements a discrete-time filter via difference equation that operates on input x. The equivalent of one pole and two zeros are modelled, described using the filter coefficients $\{a_1, a_2; b_0, b_1\}$:

$$\text{H}_{LF}(z) = \frac{b_0 + b_1 z^{-1}}{a_0 + a_1 z^{-1} + a_2 z^{-2}} \tag{85}$$

$$\text{y}[n] = -a_1\text{y}[n-1] - a_2\text{y}[n-2] + b_0\text{x}[n] + b_1\text{x}[n-1] \tag{86}$$

Pseudocode for implementation of the model class is as follows. Data is to be represented with fixed point format, with number of fractional bits frac_bits and number of integer bits int_bits. The method fixed_point rounds floating point values to be equivalent to the nearest representation in the desired fixed point format. The filter coefficients are assumed to be pre-converted to the desired fixed-point equivalent values, and input x is assumed to be integer-valued.

```
class LoopFilter:
  def __init__(self, a1, a2, b0, b1, int_bits, frac_bits):
    self.a1 = a1; self.a2 = a2
    self.b0 = b0; self.b1 = b1
    self.xprev1 = 0;
    self.yprev1 = 0; self.yprev2 = 0
```

```
    self.int_bits=int_bits; self.frac_bits=frac_bits

  def update(x):
    ynew = -self.a1*self.yprev1 - self.a2*self.yprev2 \\
         + self.b0*x + self.b1*self.xprev1  # difference equation
    self.yprev2 = self.yprev1
    self.yprev1 = fixed_point(ynew, self.int_bits, self.frac_bits)
    self.xprev1 = x
    return round(self.yprev1) # convert to integer
```

Listing 4: Loop filter behavioral model Python pseudocode.

4.5 DCO behavioral model

The DCO is modeled similar to the clock model, with the inclusion of a digital input `otw` for tuning the frequency. Nominal oscillator frequency is given by `f0`, and the DCO gain in Hz/LSB of `otw` is `kdco`. Additionally, modeling of the $\propto f^-2$ oscillator phase noise component, which is equivalent to random phase walk [24], is included utilizing additive random phase walk. Random walk is implemented by stochastically either adding or subtracting a fixed magnitude random walk phase increment `krw` to the oscillator phase every simulation step. The sign of `krw` is randomly chosen with equal probability for positive and negative (sampling a bi-delta distribution), implemented with the method `choice`.

$$\Phi_{osc}[n] = \Phi_{osc}[n-1] + 2\pi(\texttt{f0} + \texttt{otw} \cdot \texttt{kdco}) \cdot \texttt{dt} \pm \texttt{krw} \tag{87}$$

```
class DCO:
  def __init__(self, f0, kdco, krw, dt):
    self.f0 = f0    # nominal frequency
    self.kdco = kdco  # DCO gain
    self.krw      # random phase walk gain
    self.dt       # simulation time step size
    self.phase = 0    # phase state variable

  def update(otw):
    self.phase += 2*pi*(self.f0 + otw*self.kdco)*self.dt + krw*choice
       ([-1,1])
    return self.phase
```

Listing 5: DCO behavioral model Python pseudocode.

Selection of the parameter `krw` to achieve a target phase noise level $\mathcal{L}(\Delta f)$ at offset Δf from the carrier is as follows.

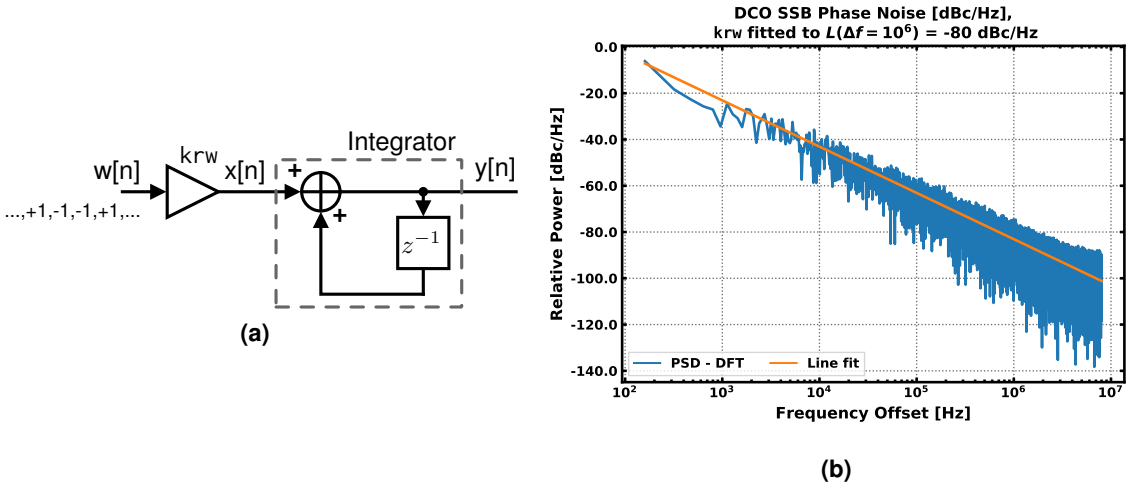


Figure 21: (a) Discrete model for oscillator random walk, **(b)** Simulated phase noise of behavioral model.

The random walk process described in the behavioral model is represented in figure 21a, where a random white-spectrum input sequence w[n] taking on values $\pm$ 1 with equal probability are multiplied by gain krw, yielding signal x[n] that is passed a discrete integrator. The output y[n] is then the phase noise signal that is summed with the oscillator phase trajectory. If the simulation is limited to sim_steps samples, application of Parseval's theorem for the discrete Fourier transform (DFT) of x[n] in 88 results in the estimate of the energy spectrum x[n] 89, having a constant value σ_x^2 .

$$\sum_{n=0}^{\texttt{sim\_steps}-1} |\texttt{krw} \cdot w[n]|^2 = \frac{1}{\texttt{sim\_steps}} \sum_{k=0}^{\texttt{sim\_steps}-1} |X[k]|^2 \tag{88}$$

$$\sigma_x^2 = |X[k]|^2 = \texttt{sim\_steps} \cdot \texttt{krw}^2 \tag{89}$$

Computation of the spectrum of the output $y[n]$ of figure 21a can be computed as follows, recalling that $x[n]$ is white with power σ_x^2, and approximating $z = 1 - s\texttt{dt}$ is in 90. Normalizing $Y(f)$ by the number of samples sim_steps to compute spectral density is performed in 91.

$$|Y(f)|^2 = |X(z)|^2 \frac{1}{|1 - z^{-1}|^2}\bigg|_{z^{-1}=(1-j2\pi f\texttt{dt})} = \frac{\sigma_x^2}{|j2\pi f\texttt{dt}|^2} = \frac{\texttt{sim\_steps} \cdot \texttt{krw}^2}{(2\pi f\texttt{dt})^2} \tag{90}$$

$$|\hat{Y}(f)|^2 = \frac{\texttt{krw}^2}{\texttt{sim\_steps} \cdot (2\pi f\texttt{dt})^2} \tag{91}$$

Given the target phase noise level $\mathcal{L}(\Delta f)$ at offset Δf, we set $f = \Delta f$ and $|\hat{Y}(\Delta f)|^2$=$\mathcal{L}(\Delta f)$, a reorganization of 91 results in the final expression for krw, equation 92. Figure 21b demonstrates simulated a test of this behavioral model, for krw fitted to $\mathcal{L}(\Delta f = 10^6)$ = -80 dBc/Hz.

$$\texttt{krw} = 2\pi\Delta f \cdot \texttt{dt}\sqrt{\mathcal{L}(\Delta f) \cdot \texttt{sim\_steps}} \tag{92}$$

4.6 Divider behavioral model

The divider model is defined with the divider modulus `div_n` and only performs a simple division of input phase.

```
1 class Divider:
2   def update(x, div_n):
3     return x/div_n
```

Listing 6: Divider behavioral model Python pseudocode.

4.7 Post processing: lock time detection

Lock time detection of the PLL start-up transient can be determined from the simulation data conditioned on a tolerance band for acceptable frequency error `lock_f_tol` is provided to the simulator by the user. Since the desired frequency of the PLL `fosc = div_n*fref`, nominal oscillator frequency `dco_f0` and simulation conditions of the PLL are known by the simulator, the ideal value of the loop filter at lock, `lf_lock_ideal` can be computed directly. Given the DCO gain value in the simulation `kdco`, the value of `lf_lock_ideal = round((fosc - dco_f0)/kdco)`. Lock is then detected at the first simulation step for which the current and all later time steps meet the following criteria for the loop filter signal `lf_sig` (i.e. frequency of oscillator is within the frequency tolerance band):

$$\texttt{abs(lf\_sig[n] - lf\_lock\_ideal)*kdco < lock\_f\_tol} \tag{93}$$

This measurement is predicated on the simulation being fully complete at the time of measurement. Lock time is computed by multiplying the simulation index of lock with the simulation time step, `1/fs`. Figure 22 demonstrates the lock detection technique.

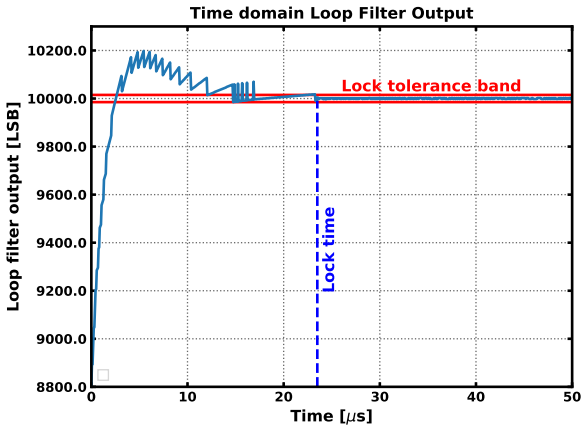


Figure 22: Detection of lock time from simulated PLL transient at loop filter.

4.8 Post processing: phase noise power spectrum estimate

The simulation data can be used to make an estimate of the phase noise power spectrum $\mathcal{L}(\Delta f)$ of the PLL (normalized to the carrier power). First, the phase error signal of the simulated PLL must be computed, `phase_error = div_n*clk_sig - osc_sig`. The phase error signal includes the phase noise signal in addition to any transient phase error components of the PLL. If the simulated PLL is in lock, the phase error can be is dominated by phase noise components, as the transient components become negligible. If `phase_error` is reduced to the simulation signal span after the detection of lock, with a span in samples of `n_steps`, the power spectral density of the phase noise normalized to the carrier tone, utilizing the fast Fourier transform (FFT) is in 94. This definition is based on the FFT implementation available in `numpy.fft` [11].

$$\texttt{psd} = \left| \frac{1}{\texttt{n\_steps}} \cdot \mathcal{FFT}\{\texttt{div\_n*clk\_sig - osc\_sig}\} \right|^2 \tag{94}$$

Provided the simulation sampling rate `fs`, the indices [0, `n_steps`/2-1] of the FFT and consequently `psd` correspond to frequencies [0, `fbin`·(`n_steps`/2-1)], where `fbin` = `fs`/`n_steps`. Slicing the power spectrum data `psd` with indices [1, `fbin`·(`n_steps`/2-1)] will yield the single side band spectrum of the oscillator, with the corresponding offset frequency of each index `k` being $\Delta f \cdot \texttt{k} \cdot \texttt{fbin}$. Thus:

$$\mathcal{L}(\Delta f) = \texttt{psd[round(}\Delta f\texttt{/fbin)]} \tag{95}$$

Computation of power spectrum based on the DFT has high variance about the true spectrum, thus an additional approach utilized here is parametric spectral estimation with autoregressive (AR) model [17], which allows for lower variance of the estimate. An AR(p) model is simply a transfer function with p poles, which can be minimum mean square error (MMSE) optimized to fit the spectrum of a signal. The MMSE optimization for an AR model can be performed using linear algebra with the Yule-Walker equations, which is based on the autocorrelation sequence of the signal to be fitted (see appendix A). This approach is utilized in this work. Figure 23 shows example power spectrum estimates utilizing the two implemented methods on a test data set.

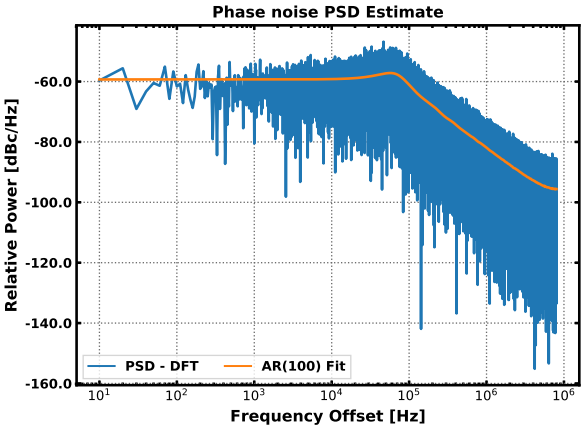


Figure 23: Power spectrum estimates example.

4.9 Monte-Carlo sampling

The simulation code implemented uses a Python dictionary containing the simulation parameters and the corresponding parameters for which the simulation engine should simulate the PLL for. This format makes the introduction of Monte-Carlo sampling into the simulator straightforward. This can be implementing utilizing a dictionary with the nominal simulation configuration, and then a second dictionary to define the parameters to be varied along with the corresponding parameter variance. Given a sample size of N for the Monte-Carlo simulation, a loop is implemented which creates a new simulation configuration dictionary each loop iteration with stochastically sampled values from a normal distribution based on the provided nominal configuration and variance dictionaries. That unique configuration dictionary instance is used to spawn a PLL simulation, and is stored. After N iterations, N sets of data are created with varied parameters.

5 Loop filter optimization

The loop filter optimization engine implemented minimizes total phase noise power of a PLL subject to a maximal lock time constraint. The optimizer uses a fixed prototype design for the loop filter in the form of equation 96 arrived at in section 3.2.2 with parameters $\{K_i, \omega_p, \omega_z\}$. Based on section 3.2.5, the PLL developed in this work will have open loop and normalized and continuous closed loop transfer functions $\mathrm{L}(s)$ and $\hat{\mathrm{T}}(s)$ in 97. The parameter K is equivalent to $\frac{\mathrm{M}}{\mathrm{N}} K_{DCO} K_i$, where M is the TDC resolution in steps, N is the divider ratio, K_{DCO} is the DCO gain in Hz/LSD and K_i is the loop filter integral term gain.

$$\mathrm{H}_{LF}(s) = \frac{K_i}{s} \frac{(\frac{s}{\omega_z} + 1)}{(\frac{s}{\omega_p} + 1)} \tag{96}$$

$$\mathrm{L}(s) = \frac{K}{s^2} \frac{(\frac{s}{\omega_z} + 1)}{(\frac{s}{\omega_p} + 1)}, \qquad K = \frac{\mathrm{M}}{\mathrm{N}} K_{DCO} K_i, \qquad \hat{\mathrm{T}}(s) = \frac{\mathrm{L}(s)}{1 + \mathrm{L}(s)} \tag{97}$$

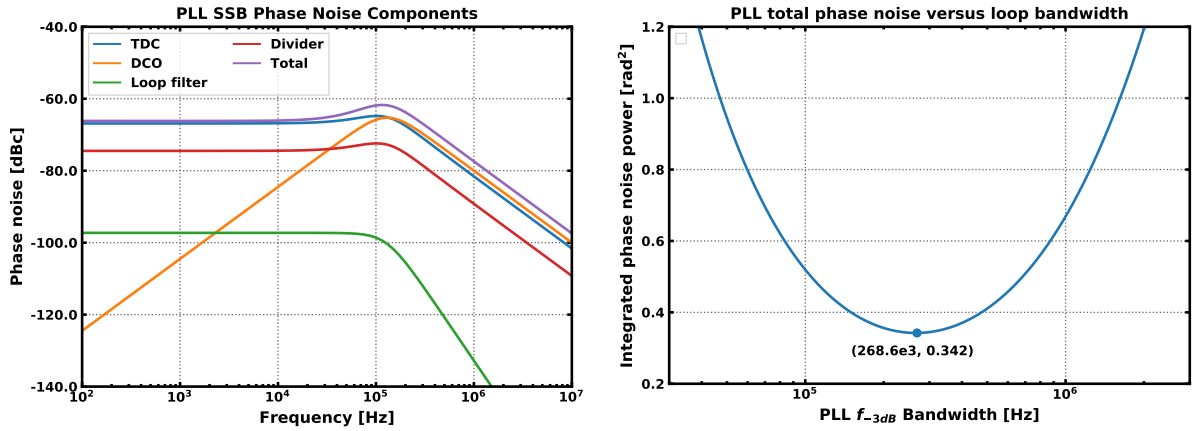
The filter optimization and design sequence is detailed below. The following sections cover this in more detail.

1. Optimize parameters $\{K, \omega_p, \omega_z\}$ of $\hat{\mathrm{T}}(s)$ for minimize total PLL phase noise power, using the phase noise model developed in section 2.3.7 parameterized by the transfer function $\hat{\mathrm{T}}(s)$ and PLL parameters $\{\mathrm{M}, \mathrm{N}, K_{DCO}, S_{0\Phi n_{DCO}}\}$.
2. Translate optimal parameters $\{K, \omega_p, \omega_z\}$ to prototype loop filter parameters $\{K_i, \omega_p, \omega_z\}$, perform continuous to discrete time filter conversion.
3. Optimize fixed point resolution of digital loop filter implementation for finite word effects.

5.1 Optimization rationale

The ADPLL transfer function in use (equation 97), coupled with the phase noise theory of section 2.3.7 results in an output PLL phase noise spectrum with component-wise breakdown of figure 24a. The showing figure uses an arbitrary selection of PLL and phase noise parameters, however, varying the model parameters will shift the individual components up and in power and frequency. It is important to note in the presented PLL design, TDC, divider and loop filter phase noise components are manifested with a low pass response, and the DCO noise with a band pass response. Also note the overall PLL response is low pass, owing to the low pass nature of the loop filter. Figure 24b shows the result of integrating the phase noise spectrum to yield total PLL phase noise power for the system modeled in figure 24a. The

parameters of the PLL were varied to change the half-power bandwidth of the closed loop response, resulting in a plot of total phase noise power versus PLL loop bandwidth. Here it is seen that there is an *optimal* selection of bandwidth that minimizes phase noise power; and that loss function for this optimization, i.e. the total phase noise power, is convex, and thus is easily optimizable.



(a) **(b)**

Figure 24: (a) Example PLL phase noise from models in this work, **(b)** integrated phase noise power versus bandwidth for the same PLL.

The take-away here is the PLL will have an optimal set of parameters which will minimize the overall PLL phase noise. There is then a question of criteria for optimization. In the optimizer introduced here, the TDC and DCO phase noise are assumed to be the principal phase noise components, and are used as the sole criteria for establishing phase noise optimality. This assumption follows that the loop filter and divider implementations can be tuned easily to have less phase noise than DCO and TDC. This will be discussed in more detail later. If the TDC and DCO parameters are kept fixed, but the loop filter is varied resulting in a change of PLL bandwidth, figure 25 illustrates the corresponding changes in the DCO and TDC phase noise components. At low bandwidths, DCO noise because dominant, and at high bandwidths the TDC noise is dominant. In both cases, the total integrated phase noise is high (sub-optimal) due to excessive contributions from either component. Optimal bandwidth occurs when the contributions from both sources are approximately equal. It is possible that the optimal bandwidth for phase noise does not result in the desired transient lock time for the PLL, thus the final criteria for optimality is defined as *the filter configuration that minimizes integrated TDC and DCO phase noise subject to a constraint for maximum lock time.*

5.2 Loop filter optimization algorithm

Based on the developed criteria for PLL optimality, the following algorithm is defined to establish the optimal parameters $\{K, \omega_p, \omega_z\}$ of $\hat{\mathrm{T}}(s)$. In order to verify phase noise and lock

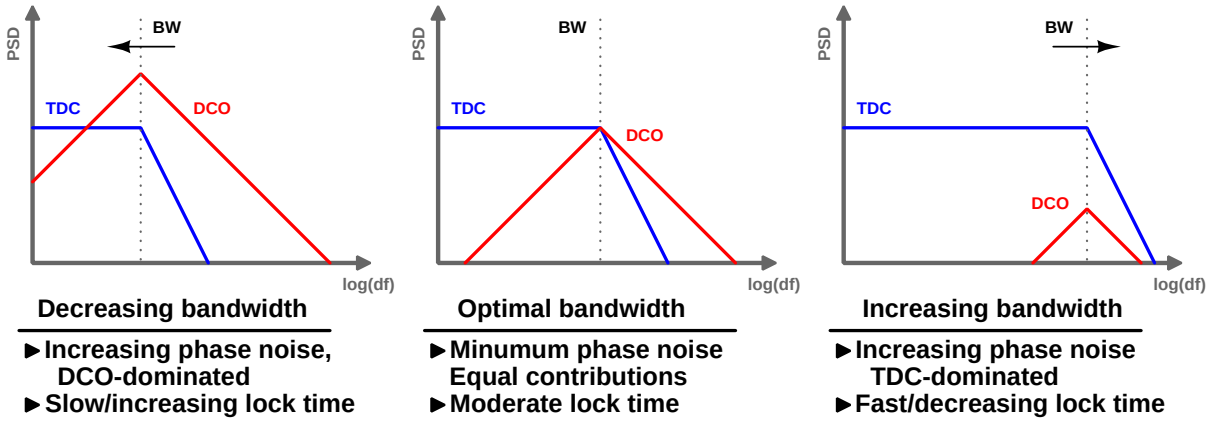


Figure 25: Bandwidth versus total integrated phase noise of PLL.

time, algorithms based on the transfer function $\hat{\mathrm{T}}(s)$ have been implemented (see sections 5.3 and 5.4). Usage of the transfer function $\hat{\mathrm{T}}(s)$ for these estimates is done as it is computationally fast compared to direct time-domain simulation of the ADPLL, allowing for fast optimization.

To allow the phase noise minimization and the constraint on maximum lock time to both have an impact in the optimization, the optimization is implemented with a divided approach. There is there is a lower level optimizer (optimizer B) which minimizes phase noise given a fixed target for settling time. There is also a higher level optimizer (optimizer A) that applies a constrained search for settling time that minimizes phase noise, using optimizer B to determine the optimal filter and minimum phase noise at each settling time value. Figure 26 illustrates this algorithm.

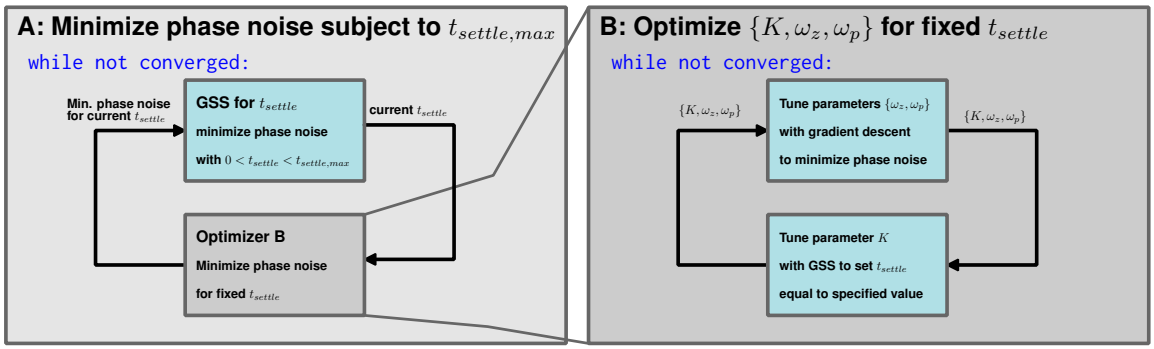


Figure 26: Optimizer visualized.

Optimizer A - Minimize PLL phase noise given a maximal constraint on settling time.

- Provided a maximum settling time $t_{settle,max}$ and PLL parameters $\{\mathrm{M},\mathrm{N},K_{DCO},S_{0\Phi n_{DCO}}\}$ one-dimensional golden section search (GSS) [16] is used with settling time t_{settle} as the varied parameter, trying to minimize total phase noise power. The cost function for this optimization is the minimum phase noise for each t_{settle} tried found using optimizer B.

Optimizer B - Minimize PLL phase noise for fixed settling time.

- Provided a fixed target for settling time = t_{settle}, and PLL parameters $\{\mathrm{M},\mathrm{N},K_{DCO},S_{0\Phi n_{DCO}}\}$, evaluate the two steps, satisfactorily converged of optimization parameters is observed:

  1. Minimize total phase noise power using pole/zeros locations $\{\omega_p, \omega_z\}$ using gradient descent with a two-point step size method from [1].
  2. Tune K such that settling time is equal to the fixed target value using unconstrained golden section search.

Convergence for gradient descent, GSS and optimizer B algorithms is decided by comparing the parameter values in each optimizer iteration to the previous. If the normalized change in a given iteration for all optimization parameters is below a tolerance specified by the user, convergence is detected and the optimization ends.

Once the optimal parameters $\{K, \omega_p, \omega_z\}$ for $\hat{\mathrm{T}}(s)$ are determined, the parameter K can be translated onto K_i with equation 98. The loop filter design then can be mapped into a difference equation which is implementable in digital hardware as a a direct form-I IIR architecture, as outlined in section 3.2.4.

$$K_i = \frac{\mathrm{M}}{\mathrm{N}} \frac{K}{K_{DCO}} \tag{98}$$

5.3 Fast estimation of PLL settling time

The following is the method implemented to estimate settling time from the PLL closed loop phase transfer function $\hat{\mathrm{T}}(s)$. $\hat{\mathrm{T}}(s)$ can be represented as a rational function of two polynomial functions of s, with P poles and Z zeros. Such a transfer function is computationally represented with arrays A and B, containing the denominator and numerator polynomial coefficients.

$$\hat{\mathrm{T}}(s) = \frac{\sum_{j=0}^{Z} b_j s^j}{\sum_{k=0}^{P} a_n s^n} \tag{99}$$

An estimate of the step response settling time of $\mathrm{T}(s)$ can by utilizing its representation in state space. This is given in 100, with input vector $\mathrm{U}(s)$, state vector $\mathbf{X}(s)$, and output $Y(s)$. The state-space representation from a s-domain transfer function can be quickly solved computationally with available signal processing packages such as `scipy.signal` [23] given the coefficient arrays A and B.

$$s\mathbf{X}(s) = \mathbf{A}\mathbf{X}(s) + \mathbf{B}\mathrm{U}(s) \tag{100}$$

$$Y(s) = \mathbf{C}\mathbf{X}(s) + \mathbf{D}\mathrm{U}(s) \tag{101}$$

The set of k eigenvalues $\{\lambda_1, ..., \lambda_N\}$ corresponding to poles for the system are found as the roots of 102 [2]. Matrix $\mathbf{A}$ is referred to as the state matrix.

$$|\mathbf{A} - \lambda\mathbf{I}| = 0 \tag{102}$$

Imposing the constraint of number of poles P > number of zeros Z, the system $\mathrm{T}(s)$ may be represented via partial fraction decomposition using the poles from the eigenvalues of state matrix $\mathbf{A}$ $\{\lambda_1, ..., \lambda_N\}$:

$$T(s) = \sum_{k=1}^{P} \frac{c_k}{s - \lambda_k} \tag{103}$$

Inverse Laplace transformation shows the step response of the system will be a sum of exponentials:

$$y(t) = c_1 e^{\lambda_1 t} + ... + c_k e^{\lambda_k t} \tag{104}$$

The dynamics of the step response are governed by the exponential components of y(t). If $\{\lambda_1, ..., \lambda_N\} \in \mathbb{C}$ where $\lambda_k = 1/\tau_k + j\omega_K$, the real portion of each λ_k will describe the transient behavior of each exponential, having time constant τ_k. The long term settling of y(t) will be dominated by the exponential with the largest τ_k, i.e. the dominant pole of the system. This estimate of settling time uses the dominant pole τ_k as a heuristic estimate for overall time constant of the system, τ. Finally, settling time t_s can be considered as the time interval required for the signal to drop within a tolerance band $\pm\delta_{tol}\mathrm{y}(\infty)$ about the final value $\mathrm{y}(\infty)$. Thus:

$$t_s = \tau \ln(\delta_{tol}) = \frac{\ln(\delta_{tol})}{\min(|\Re(\{\lambda_1, ..., \lambda_k\})|)} \tag{105}$$

This settling time estimate is computationally fast, as it requires only (a) computation of state matrix $\mathbf{A}$ from $\hat{\mathrm{T}}(s)$, (b) computation of the eigenvalues of $\mathbf{A}$, and (c) computation of settling time from the eigenvalue with minimum real component.

5.4 Estimation of PLL phase noise

The following is the method implemented to estimate total integrated phase noise power from the PLL closed loop phase transfer function $\hat{\mathrm{T}}(s)$, and PLL parameters $\{\mathrm{M}, \mathrm{N}, K_{DCO}, S_{0\Phi n_{DCO}}\}$. As has been stated, this optimizer assumes the dominant output-referred phase noise contributions of the PLL are due to the DCO thermal noise and TDC quantization. If S_{TDC} and S_{DCO} are the PLL output-referred noise PSD respectively for the TDC and DCO noise sources from section 2.3, the total PLL output noise PSD $S_\Sigma(f)$ is estimated as 106.

$$S_\Sigma(f) = S_{\Phi n_{TDC,out}}(f) + S_{\Phi n_{DCO,out}}(f) \tag{106}$$

$$= \frac{1}{12 f_{ref}} \left| 2\pi \frac{\mathrm{N}}{\mathrm{M}} \hat{\mathrm{T}}(f) \right|^2 + \frac{S_{0\Phi n_{DCO}}}{f^2} \left| 1 - \hat{\mathrm{T}}(f) \right|^2 \tag{107}$$

Given a bandwidth of interest Δf (i.e. baseband bandwidth for radio applications), the total integrated phase noise power is:

$$P_{\phi noise} = 2 \int_0^{\Delta f} S_\Sigma(f) df \tag{108}$$

This can be computed via numerical integration on a grid of K values in the interval Δf, where each point represents a frequency bin f_{bin} = Δf/K. The Romberg method for numerical integration [3] is utilized in this work as it has high accuracy for smooth integrands.

5.5 Loop filter optimization - finite word effects

Once a filter design has been optimized in the continuous domain following the algorithm of section 5.2, second order optimization is carried out to ensure the digitized, discrete implementation performs as expected in the presence of finite word effects. The optimized digital implementation provided here utilized fixed-point words for equal resolution throughout the datapath. The implemented second order optimization considers both the effect of loop filter quantization noise and transfer function error.

This optimizer works by converting the ideal discrete filter design into a fixed point implementation with direct form-I structure. All components of the data path in the optimization are constrained to have the same signed fixed point format, in twos complement format with 1 sign bit, a number of integer bits `int_bits`, and a number of fraction bits `frac_bits`. Thus the total data word representation bits is `B` = 1 + `int_bits` + `frac_bits`.

Representation of data throughout the digital implementation here

optimum = min number of bits that meets users accuracy and noise spec.

5.5.1 Loop filter quantization noise optimization

5.5.2 Loop filter transfer function error optimization

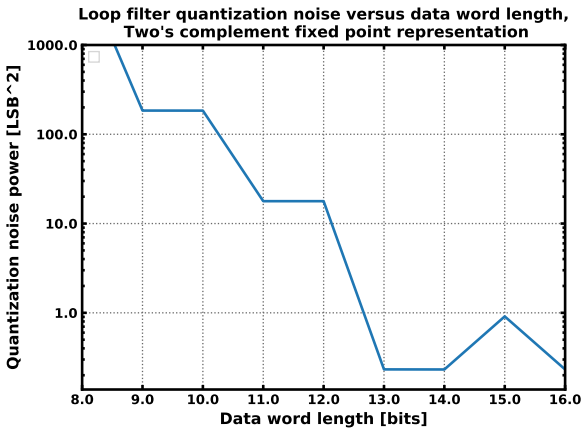
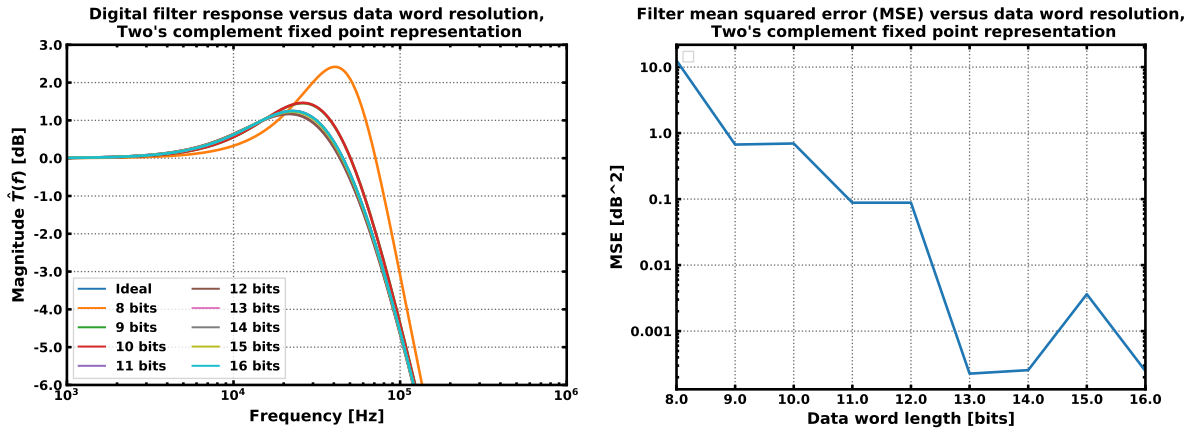


Figure 27: Quantization noise power power out of an example loop filter versus data word resolution.



(a) **(b)**

Figure 28: (a) Example filter error due to coefficient quantization, **(b)** Example MSE error of filter design due to coefficient quantization.

6 Discussion and results

Thus far in this work a solution to simplify and automate design the process of all digital PLL loop filter designs comprised of (1) an automated loop filter optimization and design engine, and (2) a discrete-event, time domain PLL simulator to evaluate the designed filters with full time-discretization and quantization nonlinearity effects. Now, in this discussion, the performance of the presented design solution will first be evaluated with a design example. Then, a comparison of the presented solution then will be made to existing solutions in literature, pointing out advantages and disadvantages will be made. Finally, a general discussion will be made covering areas of improvement, reasoning for the central design choices made, and considerations for usage the framework.

6.1 Design exercise using this work

The design of a loop filter for the PLL with the system level specifications of table 1 will be considered here, with the intent of optimizing phase noise. These specifications, where the TDC resolution in steps equals the divider ratio, is equivalent to a special case of a TDC-less PLL where a 150-step synchronous counter is used as a divider, and the loop filter directly samples the state of the synchronous counter. For filter design, a PI-controller loop filter prototype was utilized in the optimizer.

Parameter	Value	Unit
Output Frequency	2.4	GHz
Ref. frequency	16	MHz
Divider ratio	150	
TDC resolution	150	steps/reference cycle
DCO gain K_{DCO}	10^4	Hz/LSB
DCO Phase noise	-80	dBc/Hz at $\Delta f = 10^6$ Hz
Lock Time	≤ 25	μs
Lock Δf tolerance	10^5	Hz
Digital filter word resolution	≤ 16	bits
Residual phase modulation	minimize	

Table 1: System-level specifications

6.1.1 Result of filter optimization.

Table 2 contains the optimized filter parameters obtained from the filter design optimizer developed in this work. $\{K, K_i, K_p, f_z\}$ are the continuous model parameters of section 3.2.1, and $\{a_0, a_1, a_2, b_0, b_1\}$ are the filter coefficients for the discrete-time direct form-I filter structure of section 3.2.4. The estimated bandwidth for the filter is 144.8 kHz, and the lock time is estimated to be 19.3 μs. Table 3 contains the digitized version of the discrete time filter, based on a selection for word size determined via optimization for finite word effects. The final data words are 13 bits in length.

6.1.2 Result of transient and phase noise simulation.

The simulation engine implemented in this work was utilized to run a time domain simulation to verify the designed filter. Figures 29a and 29b demonstrate the transient response of the PLL with an initial frequency error of 12 MHz (0.5% of final frequency). It is observed that the PLL design stably approaches the target frequency in approximately 23 μs. Figures 30a

Parameter	Value	Unit
K	1.343792×10^{11}	
K_i	1.343792×10^{7}	
K_p	7.331074×10^{1}	
f_z	2.917324×10^{4}	Hz
b_0	$7.4150613906 \times 10^{1}$	
b_1	$-7.3310743796 \times 10^{1}$	
a_0	1.0×10^{0}	
a_1	-1.0×10^{0}	
a_2	0.0×10^{0}	
Estimated bandwidth	1.448234×10^{5}	Hz
Estimated lock time	1.934253×10^{-5}	seconds

Table 2: PLL parameters determined from filter design and optimization process.

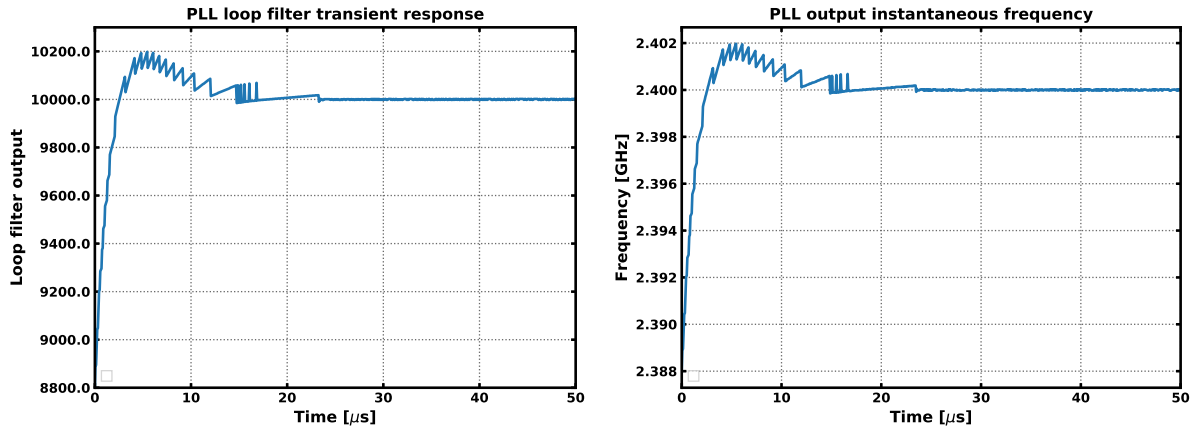
Parameter	Value	Value (digital)	Value Error
Total dataword bits	13		
Sign bits	1		
Integer bits	7		
Fractional bits	5		
b_0	7.415625×10^{1}	`0b0100101000101`	$+5.636094 \times 10^{-3}$
b_1	-7.331250×10^{1}	`0b1111011010110`	-1.756204×10^{-3}
a_0	1.0×10^{0}	`0b0000000100000`	0.0×10^{0}
a_1	-1.0×10^{0}	`0b1111111100000`	0.0×10^{0}
a_2	0.0×10^{0}	`0b0000000000000`	0.0×10^{0}

Table 3: Loop filter parameters after digitization and optimization for data word length.

illustrate the BBPD and TDC outputs during this inital transient. It is observed that the TDC output gives the dominant feedback, unit it reaches an output word of 0, where the BBPD then begins providing feedback in the steady state condition of the PLL. Figure 30b is the result of a phase noise calculation for the simulated PLL in an interval beginning immediately after detection of lock. The spectrum closely approximates that designed for, with some small additional peaking.

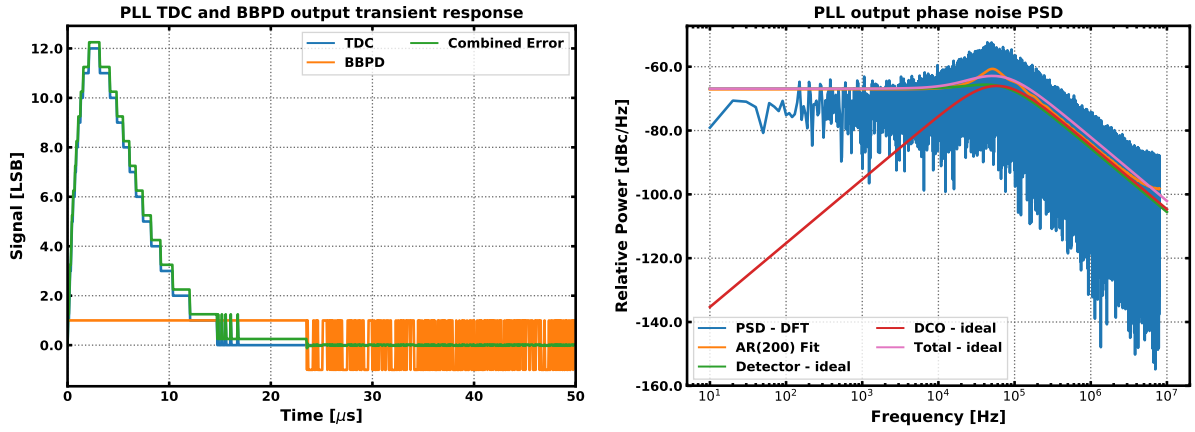
6.1.3 Result of parametric sweep and variation analysis.

The Monte-Carlo sampling and parametric sweep facilities in the simulator designed in this work were used to run analysis for effects of variation of DCO gain K_{DCO} and initial frequency error of the PLL. Figure 35a shows a parametric sweep of K_{DCO}, with lock time



(a) (b)

Figure 29: Simulation with 0.5% initial frequency error: **(a)** Loop filter transient response, **(b)** PLL output instantaneous frequency.

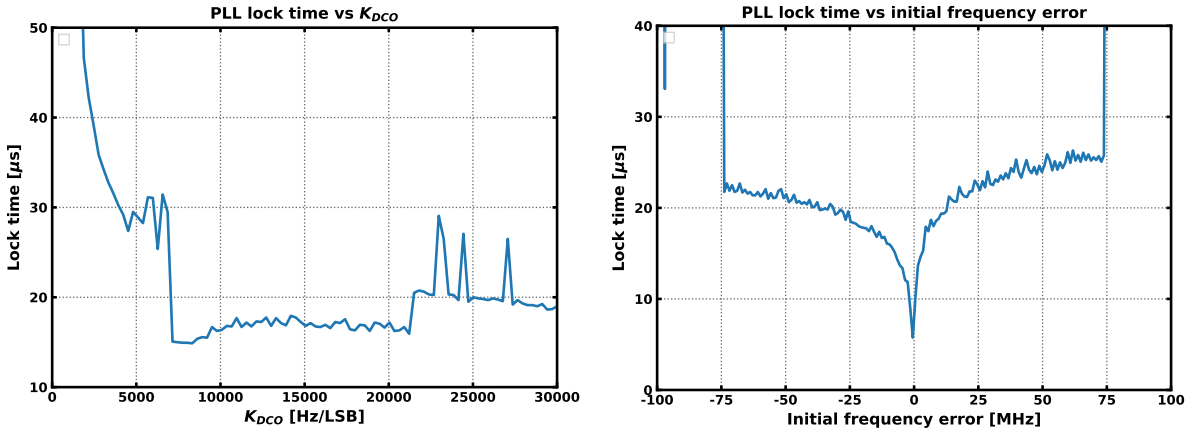


(a) (b)

Figure 30: Simulation with 12 MHz (0.5%) initial frequency error: **(a)** BBPD/TDC detector responses, **(b)** PLL output phase noise power spectrum.

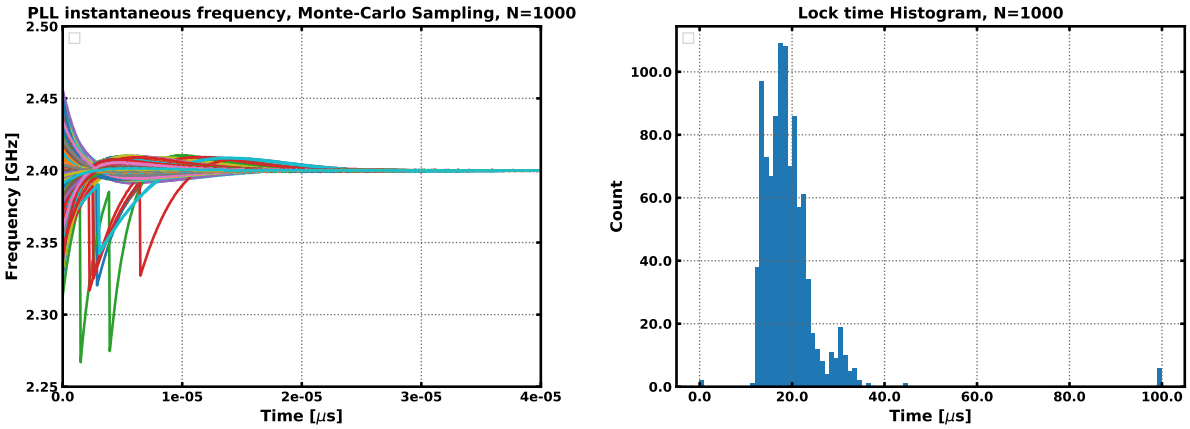
being measured. It is observed that the lock time is nearly flat in the range 7300-18000 Hz/LSB, meaning that there is a tolerance of -2700/+8000 Hz LSB for KDCO about the nominal 10000 Hz/LSB specified. This specification can be utilized in the design of a physical DCO to constrain maximum acceptable variation across PVT. Figure 35b demonstrates the simulated effect of initial frequency error on lock time. It is seen that PLL stably locks to the target frequency within the entire simulated interval of $\pm$ 60 MHz from 2.4GHz, implying that the capture range of the PLL is $>$ 120 MHz. This specification can be translated into a requirement for initial coarse frequency calibration needed before starting the PLL. Figures 36a and 36b are the results of a variation analysis simulation utilizing the Monte-Carlo sampling engine implemented in this work. The simulation was configured to vary K_{DCO} with a standard deviation of 20 % of the nominal value, and to vary the inital starting frequency with a standard deviation of 60 MHz (0.025 % of the final frequency). The simulation sample size is 1000. The transient responses from the simulation figure 36a are all stable, and figure

36b shows the histogram for measured lock time subject to the described variation. The mean lock time was 24.57 μs, meeting the 25 μs lock time set for the PLL, and the upper bound for a 99% confidence interval on the data is 50.75 μs. A set of extracted parameters from these simulations are in table 4. The Monte-Carlo simulations presented here are useful to analyze the range of variation in which the designed PLL can tolerate, as well as determine the expected performance varation, so well informed decisions on a PLL design can be made before moving into transistor level implementation and simulation.



(a) (b)

Figure 31: (a) PLL lock time simulation with KDCO swept, 12 MHz (0.5%) initial frequency error, **(b)** PLL lock time simulation with initial frequency error swept.



(a) (b)

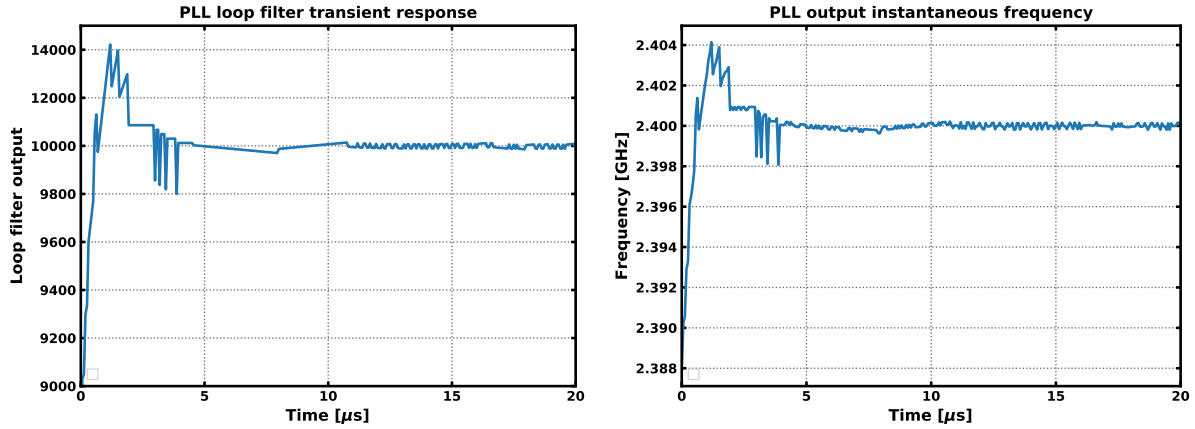
Figure 32: Monte-Carlo simulation with 1000 samples, 20% RMS deviation in KDCO, and 60 MHz (2.5%) RMS deviation in initial frequency error **(a)** Frequency transient responses, **(b)** Lock time histogram.

6.1.4 Design method 2

make sure captions and labels are unique

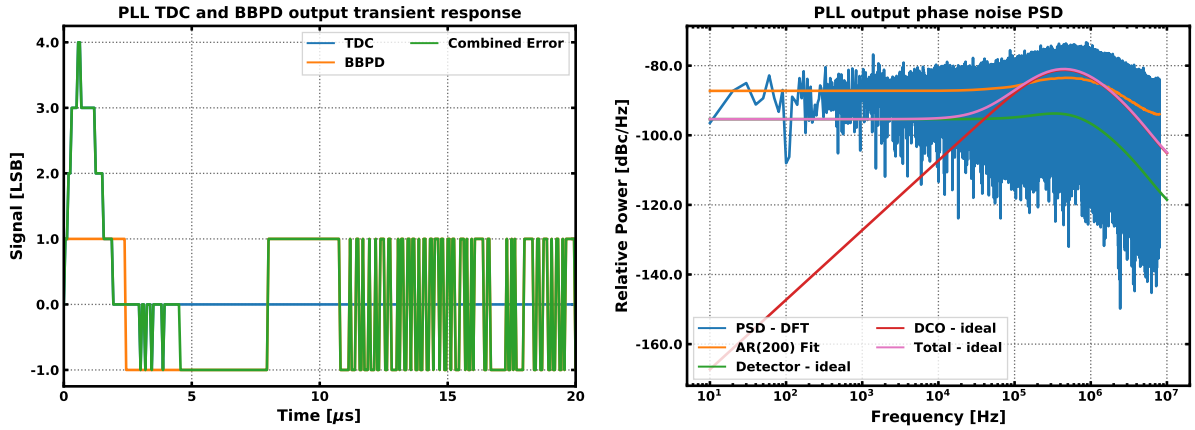
Parameter	Value	Unit
K_{DCO} Tolerance	-2700/+8000	MHz/LSB
Capture range	> 120	MHz
Mean lock time	24.57263	μs
Lock time σ	8.286061	μs
Lock time 99 % CI upper bound	50.75	μs

Table 4: PLL parameters extracted from variance and parameter sweep simulations.



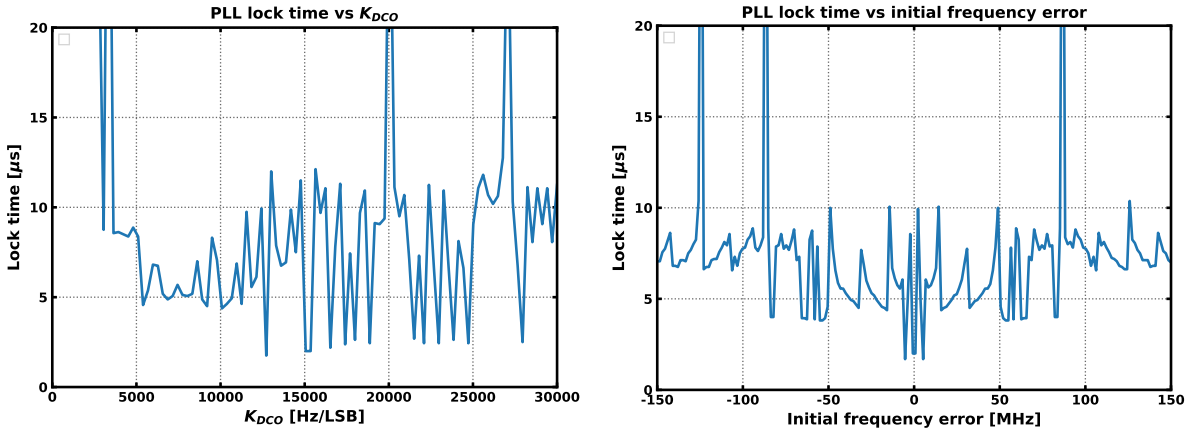
(a) (b)

Figure 33: Simulation with 0.5% initial frequency error: **(a)** Loop filter transient response, **(b)** PLL output instantaneous frequency.



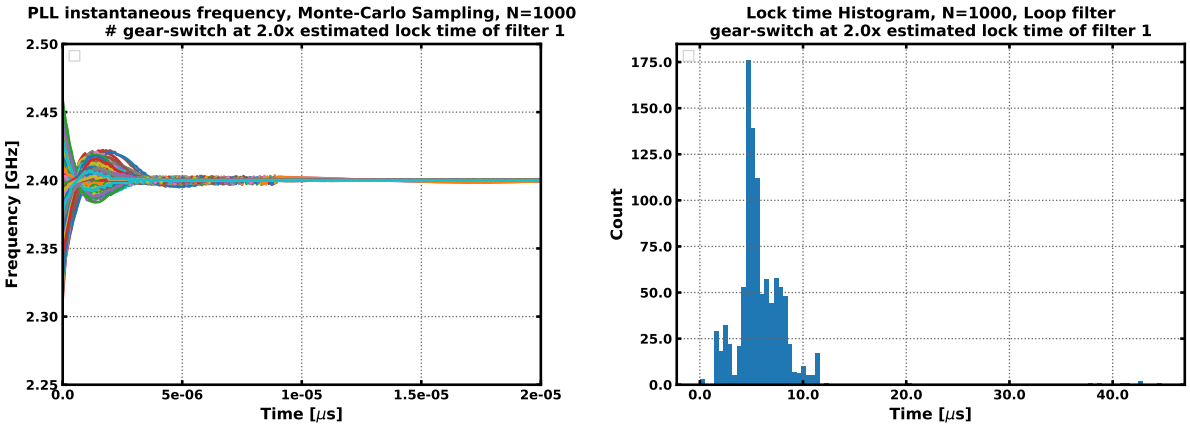
(a) (b)

Figure 34: Simulation with 12 MHz (0.5%) initial frequency error: **(a)** BBPD/TDC detector responses, **(b)** PLL output phase noise power spectrum.



(a) **(b)**

Figure 35: (a) PLL lock time simulation with KDCO swept, 12 MHz (0.5%) initial frequency error, **(b)** PLL lock time simulation with initial frequency error swept.



(a) **(b)**

Figure 36: Monte-Carlo simulation with 1000 samples, 20% RMS deviation in KDCO, and 60 MHz (2.5%) RMS deviation in initial frequency error **(a)** Frequency transient responses, **(b)** Lock time histogram.

6.2 Comparison to existing solutions

Due to the relative youth of all digital PLL design, and discontinuity between the continuous theory of analog PLL and digital PLL design, a smaller body of works exist pertaining to the loop filters for exclusively ADPLLs. Of the existing literature, the majority design approaches utilize discrete-time converted PI-controller loop filters with either bang-bang phase detectors [7][22][26][25] or phase-frequency detectors [8][4]. This work takes a similar approach to existing works, focusing on a fixed filter architecture to limit the scope of the problem at hand, and also utilizing a bang-bang phase detector in the PLL architecture. This work, however, differs in its approach to filter design, which is through numerical methods to find optimal filter design, whereas the other works largely focus on closed-form mathematical analysis. The usage of numerical methods here allows for full automation of the loop filter design process, removing work for the designer.

The criteria and motivation for loop design varies across the surveyed works. Of the surveyed works, several [7][8][4][22] do not consider optimization for phase noise as this work principal;y does, but rather consider stability/phase margin [7][8][22], lock time [4][22] and even approach simplicity [8]. The methods that consider optimization of phase noise [26][25] both provide a similar model of PLL dynamics based on a linearized model of the bang-bang phase detector. The design methods of the previous two works are presented with closed-form mathematical theory, using closed loop transfer function modeling to estimate phase noise. It is expected that these approaches yield a better optimization than that afforded with this work when using feedback from a bang-bang phase detector, as this work only attempts to reduce the BBPD noise below that expected for the TDC. This work, however, differs from the latter two as the loop filter is designed to utilize both bang-bang phase detector and TDC feedback, not just bang-bang detector feedback. In terms of optimization criteria, this work is unique in that it is solely focused on minimization of total phase noise power subject to constrained lock-time requirements, both of which are of high interest to the PLL designer.

Few of the existing works consider the implementation of the loop filter design into digital hardware, rather just provide continuous valued coefficients for the filter designs. Of those surveyed, only [8] provides an analysis for quantization noise in terms of SFDR out of the filter design, but lacks the connection to output phase noise. This work uniquely considers (a) the impact of quantization of the digitized filter design, in terms of filter accuracy and output phase noise, and (b) attempts to optimize the digital implementation of the filter in terms of number of bits representing the various components of the filter (i.e. filter coefficients, multipliers, adders) for complexity and performance.

A final advantage of this work not observed in the surveyed literature is the integrated PLL simulator with Monte-Carlo sampling which allows for verification of the designed loop filter with accurate modeling of time-discretization and digital quantization effects. This allows for

lock time, phase noise and stability to be verified on the design to ensure it meets the designer's requirements before moving onto testing with transistor level implementations and testing.

6.3 Design choices and areas of improvement

still a work in progess... Currently the simulation only handles integer-N PLLs. To extend simulation to fractional-N PLLs, a small simulation time steps (much smaller than reference cycle time used in integer-N case) is needed to reduce quantization noise to low enough levels for the fractional divider noise characteristics to not be overpowered [14]. This may prove a limitation in the current simulator, which runs on the order of 60s for the integer-N PLL phase noise simulations.

- Discuss choices made in simulator
- show effects of non-linearity : simulate PLL without BB-PD (far from ideal)
- Low resolution: feedback stops when within 1 LSB in phase lock, response time = t=(n/(m*df)), Use bbpd to add extra resolution.
  - Deficiencies, advantages, why they were made
  - Phase noise/lock time analysis
  - Analysis - monte-carlo variation to analyze stability/lock time
  - purpose: to validate filter design
- Filter structure choice - PI with added pole. Why is this best (low complexity, no phase error)
- Why only phase random walk in oscillator phase noise
- Why direct type I implementation
- Discuss choices made in loop filter designer/optimizer
  - Why only optimize for TDC/DCO phase noise initially - (other noise sources are easier to reduce below the limits of these two). Flicker noise ignore due to time resolution limitations, expectation that reference flicker¿ PLL flicker components as reference flicker with be multiplied by N at output.
  - Discuss why ref flicker noise doesn't matter (it can't be altered by PLL so it doesn't matter for optimization)

  - Second order optimization of filter design for data representation precision with discrete time considerations (first order design is with approximations from continuous PLL model). Discretized LF noise via simulation with input noise
  - Recommendations for divider noise limit
  - Design verification reasoning

- Discuss limitations and considerations for use of framework
  - Models are not accurate for frequencies near or greater than reference frequency???
  - Sampling rate recommendations (high oversampling)
  - Minimum choice of TDC resolution, constraints for divider jitter,
  - Optimizer needs poles¿zeros
  - doesn't handle non-linearity well.

Recommendations for maximum divider jitter, loop filter resolution

6.3.1 Divider noise constraint

Output referred phase noise PSD of TDC:

$$S_{\Phi n_{TDC,out}} = \frac{1}{12 f_{ref}} \left| 2\pi \frac{N}{M} G(f) \right|^2 \tag{109}$$

Output referred phase noise PSD of divider:

$$S_{\Phi n_{div,out}} = f_{ref} \left| 2\pi N \sigma_{tn_{div}} G(f) \right|^2 \tag{110}$$

The output-referred phase noise for the TDC and divider have the same frequency dependence. So by setting $S_{\Phi n_{div,out}} < S_{\Phi n_{TDC,out}}$, a constraint to force PLL output divider less than TDC noise can be found:

$$\sigma_{tn_{div}} < \frac{1}{\mathrm{M} f_{ref}} = \Delta t_{step_{TDC}} \tag{111}$$

Must simply ensure that jitter of divider is much less than TDC resolution, which is a reasonable demand. Thus, it is reasonable to ignore divider noise in the phase noise optimization if divider noise can reasonably be made insignificant in the overall output phase noise.

7 Conclusion

In this work, an automatic framework for the design and optimization of all digital PLL loop filters has been introduced. The framework allows input of target PLL specifications, for which a digital implementation ready filter design will be generated having optimized phase noise in terms of integrated power, subject to specified lock time constraints. The framework additionally performs second order optimization on the generated loop filter design to ensure performance including effects of quantization in the digital loop filter. An included time domain PLL simulation engine is included within the framework to verify the performance of the designed filters for acceptable phase noise, lock time and stability. This work will be applied in a later thesis project undertaken by the author of this work concerning the design of ultra-low power ADPLLS, in order to assist and accelerate the design process.

References

[1] Jonathan Barzilai and Jonathan M. Borwein. "Two-Point Step Size Gradient Methods". In: *IMA Journal of Numerical Analysis* 8.1 (1988), pp. 141–148. DOI: 10.1093/imanum/8.1.141.

[2] R. Brockett. "Poles, zeros, and feedback: State space interpretation". In: *IEEE Transactions on Automatic Control* 10.2 (1965), pp. 129–135. DOI: 10.1109/tac.1965.1098118.

[3] "Chapter 15: Numerical Integration". In: *A First Course in Numerical Methods* (2011), pp. 441–479. DOI: 10.1137/9780898719987.ch15.

[4] Yawgeng A. Chau and Chen-Feng Chen. "All-digital phase-locked loop with an adaptive bandwidth design procedure". In: *2009 International Symposium on Intelligent Signal Processing and Communication Systems (ISPACS)* (2009). DOI: 10.1109/ispacs.2009.5383893.

[5] F. Gardner. "Charge-Pump Phase-Lock Loops". In: *IEEE Transactions on Communications* 28.11 (Nov. 1980), pp. 1849–1858. DOI: 10.1109/tcom.1980.1094619.

[6] Floyd Martin Gardner. "Chapter Two Loop Fundamentals". In: *Phaselock techniques*. John Wiley, 2005.

[7] Volodymyr Kratyuk et al. "A Design Procedure for All-Digital Phase-Locked Loops Based on a Charge-Pump Phase-Locked-Loop Analogy". In: *IEEE Transactions on Circuits and Systems II: Express Briefs* 54.3 (2007), pp. 247–251. DOI: 10.1109/tcsii.2006.889443.

[8] Martin Kumm, Harald Klingbeil, and Peter Zipf. "An FPGA-Based Linear All-Digital Phase-Locked Loop". In: *IEEE Transactions on Circuits and Systems I: Regular Papers* 57.9 (2010), pp. 2487–2497. DOI: 10.1109/tcsi.2010.2046237.

[9] T.h. Lee and A. Hajimiri. "Oscillator phase noise: a tutorial". In: *IEEE Journal of Solid-State Circuits* 35.3 (2000), pp. 326–336. DOI: 10.1109/4.826814.

[10] D.b. Leeson. "A simple model of feedback oscillator noise spectrum". In: *Proceedings of the IEEE* 54.2 (1966), pp. 329–330. DOI: 10.1109/proc.1966.4682.

[11] *numpy.fft.fft*. URL: https://docs.scipy.org/doc/numpy/reference/generated/numpy.fft.fft.html.

[12] Katsuhiko Ogata. "5-7 Effects of Integral and Derivative Control Actions on System Performance". In: *Modern control engineering*. Prentice Hall, 2010.

[13] Katsuhiko Ogata. "Chapter 8 PID Controllers and Modified PID Controllers". In: *Modern control engineering*. Prentice Hall, 2010.

[14] M.h. Perrott. "Behavioral simulation of fractional-N frequency synthesizers and other PLL circuits". In: *IEEE Design & Test of Computers* 19.4 (2002), pp. 74–83. DOI: 10.1109/mdt.2002.1018136.

[15] M.h. Perrott, M.d. Trott, and C.g. Sodini. "A modeling approach for Sigma-Delta fractional-N frequency synthesizers allowing straightforward noise analysis". In: *IEEE Journal of Solid-State Circuits* 37.8 (2002), pp. 1028–1038. DOI: 10.1109/jssc.2002.800925.

[16] William H. Press et al. "10.2 Golden Section Search in One Dimension". In: *Numerical recipes: the art of scientific computing*. Cambridge University Press, 2007.

[17] John G. Proakis and Dimitris G. Manolakis. "12 Power Spectrum Estimation". In: *Digital signal processing: principles, algorithms, and applications*. Macmillan, 1993.

[18] John G. Proakis and Dimitris G. Manolakis. "7.3 Structures for IIR Systems". In: *Digital signal processing: principles, algorithms, and applications*. Macmillan, 1993.

[19] John G. Proakis and Dimitris G. Manolakis. "8.3.3 IIR Filter Design by the Bilinear Transformation". In: *Digital signal processing: principles, algorithms, and applications*. Macmillan, 1993.

[20] Behzad Razavi. "Design of monolithic phase-locked loops and clock recovery circuits tutorial". In: 1996.

[21] Behzad Razavi and Behzad Razavi. “16 Phase-Locked Loops”. In: *Design of analog CMOS integrated circuits*. McGraw-Hill.

[22] Sally Safwat, Maged Ghoneima, and Yehea Ismail. “A design methodology for a low power bang-bang all digital PLL based on digital loop filter programmable coefficients”. In: *2011 International Conference on Energy Aware Computing* (2011). DOI: 10.1109/iceac.2011.6136676.

[23] *Signal processing (scipy.signal)*. URL: https://docs.scipy.org/doc/scipy/reference/signal.html.

[24] V. Vannicola and P. Varshney. “Spectral Dispersion of Modulated Signals Due to Oscillator Phase Instability: White and Random Walk Phase Model”. In: *IEEE Transactions on Communications* 31.7 (1983), pp. 886–895. DOI: 10.1109/tcom.1983.1095902.

[25] Hao Xu and Asad A. Abidi. “Design Methodology for Phase-Locked Loops Using Binary (Bang-Bang) Phase Detectors”. In: *IEEE Transactions on Circuits and Systems I: Regular Papers* 64.7 (2017), pp. 1637–1650. DOI: 10.1109/tcsi.2017.2679683.

[26] M. Zanuso et al. “Noise Analysis and Minimization in Bang-Bang Digital PLLs”. In: *IEEE Transactions on Circuits and Systems II: Express Briefs* 56.11 (2009), pp. 835–839. DOI: 10.1109/tcsii.2009.2032470.

A Estimating PSD with autoregressive model

The following is based on [17]. Given a signal x[n] whose power spectrum should be estimated, its autocorrelation sequence $r_{xx}[l]$ with lag l must be computed:

$$r_{xx}[l] = \sum_{n=-\infty}^{\infty} x[n]x[n-l] \tag{112}$$

The autoregressive model for power spectrum, with p poles, that shall be fitted is given in 113

$$S_{XX}(f) = \left. \frac{1}{|1 + \sum_{n=1}^{p} a_n z^{-1}|^2} \right|_{z^{-1}=e^{-j2\pi f \Delta T}} \tag{113}$$

MMSE optimization of the distribution for coefficients $\{a_1, ..., a_p\}$ is done by solving the Yule-Walker equation in 114.

$$\begin{bmatrix} a_1 \\ a_2 \\ \vdots \\ a_p \end{bmatrix} = -\mathbf{R}_{xx}^{-1}\mathbf{r}_{xx} = - \begin{bmatrix} r_{xx}[0] & r_{xx}[1] & \dots & r_{xx}[p-1] \\ r_{xx}[1] & r_{xx}[0] & \dots & r_{xx}[p-2] \\ \vdots & \vdots & & \\ r_{xx}[p-1] & r_{xx}[p-2] & & r_{xx}[0] \end{bmatrix}^{-1} \begin{bmatrix} r_{xx}[1] \\ r_{xx}[2] \\ \vdots \\ r_{xx}[p] \end{bmatrix} \tag{114}$$

B Oscillator phase noise due to thermal noise

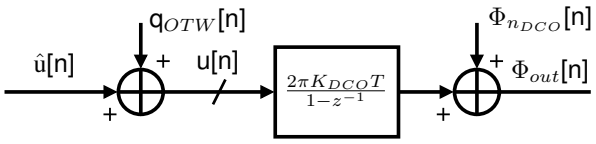


Figure 37: DCO additive noise model.

Oscillator phase noise due to stochastic and uncorrelated circuit and supply noise can be analyzed as additive voltage disturbance δv_n with variance $\sigma^2_{v_n}$ to the oscillator waveform V_{osc} at any given time. In a stable, noiseless oscillator, amplitude is inherently tied to signal phase, i.e. $V_{osc}(\Phi = \omega t)$. With additive noise, given $\frac{dV_{osc}(\Phi)}{d\Phi}$ is finite $\forall$t, a small voltage disturbance from noise δv_n will be coupled as a disturbance $\delta\Phi_n$ in the oscillator phase, shown in figure 38. The phase evolution of the noisy oscillator for an infinitesimal time increment δt with such a disturbance is:

$$\Phi_{out}(t + \delta t) = \Phi_{out}(t) + \delta\Phi_n + \omega_{osc}\delta t \tag{115}$$

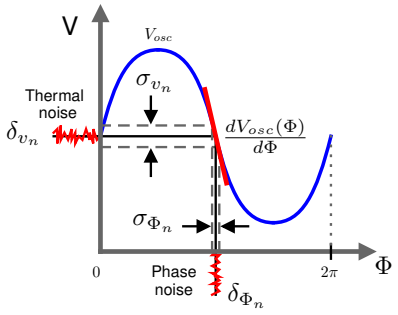


Figure 38: Voltage to phase noise conversion.

Assuming δv_n is Gaussian white noise, $\delta\Phi_n$ is sampled stochastically at any instant based on the probability distribution 116, dependent on the current oscillator phase Φ_{out} and the noiseless voltage-phase relation $V_{osc}(\Phi)$. It will be assumed that like the source noise δv_n, $\delta\Phi_n$ is white spectrum.

$$P(\delta\Phi_n|\Phi_{out}) = \text{Norm}\left(\mu = 0, \sigma = \sigma_{v_n}\left(\left.\frac{dV_{osc}(\Phi)}{d\Phi}\right|_{\Phi=\Phi_{out}}\right)^{-1}\right) \tag{116}$$

Spectral analysis of the noisy oscillator phase can be made utilizing discrete time-modeling. Converting 115 into a sampled signal with time step δt

$$\Phi_{out}[n+1] = \Phi_{out}[n] + \omega_{osc}\delta t + \delta\Phi_n[n|\Phi_{out}[n]] \tag{117}$$

Computing the z-transform, and splitting the result into the oscillation Φ_{osc} and phase noise Φ_n components:

$$\Phi_{out}(z) = \frac{\omega_{osc}\delta t}{z-1} + \frac{\delta\Phi_n(z)}{z-1} = \Phi_{osc}(z) + \Phi_n(z) \tag{118}$$

$$\Rightarrow \Phi_n(z) = \frac{\delta\Phi_n(z)}{z-1} \tag{119}$$

Application of the bilinear transform to 119 can be used to approximate the continuous phase noise spectrum, if $s = j\omega$

$$\Phi_n(s) = \Phi_n(z)|_{z=1-s\delta t} = \left.\frac{\delta\Phi_n(z)}{z-1}\right|_{z=1+s\delta t} = \frac{\delta\Phi_n(s)}{s\delta t} \tag{120}$$

The phase noise PSD is therefore:

$$S_{\Phi n_{DCO}}(f) = \lim_{\Delta T\to\infty}\frac{1}{\Delta T}\left|\frac{\delta\Phi_n(j2\pi f)}{j2\pi f\delta t} * \mathcal{F}\left\{\text{rect}\left(\frac{t}{\Delta t}\right)\right\}\right|^2 = \frac{S_{0\Phi n_{DCO}}}{f^2} \tag{121}$$

Following that the phase disturbance signal $\delta\Phi_n(t)$ is white spectrum, a constant value for its PSD $S_{0\Phi n_{DCO}}$ can be defined. The value for $S_{0\Phi n_{DCO}}$ is highly dependent on implementation and is best extracted by means of curve fitting simulation or physical measurement.