Charge-Sampling DTC based Fractional-N Phase-Locked Loop with Background DTC Gain Calibration

TU Delft

Charge-Sampling DTC based Fractional-N Phase-Locked Loop with Background DTC Gain Calibration

by

Rishabh Gurbaxani

to obtain the degree of Master of Science
at the Delft University of Technology,
to be defended publicly on Thursday October 13, 2022 at 14:00.

Student number:	5201853
Project duration:	15th November 2021 - 13th October 2022
Thesis Committee:	Dr. Fabio Sebastiano, TU Delft, Chair
	Dr. Masoud Babaie, TU Delft, Academic Supervisor
	Prof. Dr. Leo de Vreede, TU Delft

TU Delft

Contents

ABSTRACT

Phase-locked loops (PLLs) are ubiquitous in many RF applications such as frequency synthesizers in wireline and wireless transceivers. In this project, a charge-sampling PLL has been designed, which employs a charge-domain sub-sampling phase detector. The high gain of the phase detector helps suppress the in-band phase noise, while the lowered duty cycle of the sampling reference clock results in reduced reference spurs. A current-starved ring oscillator has been designed to support an output frequency range of 1-2 GHz. In order to achieve the synthesis of fractional frequencies, a capacitive DAC based constant slope digital-to-time converter (DTC) has been designed. In order to calibrate the DTC gain (K_{DTC}) over PVT variations, a digital background gain calibration loop has been introduced. The performance of the Fractional-N PLL is comparable to that of the state-of-the-art. In order to measure the PLL's performance in silicon, the design has been taped-out in TSMC 40-nm technology in July 2022.

Acknowledgements

Firstly, I would like to thank my academic supervisor, Dr. Masoud Babaie for his continuous guidance and support. He encouraged me to realize my full potential and achieve new levels. I am grateful to him for his time and for sharing his expertise in the field.

I would like to express my gratitude to Prof. Leo de Vreede and Dr. Fabio Sebastiano for reading my thesis and serving as my thesis committee.

My special thanks go to Jiang, Praneetha and Lennart who acted as my mentors during the course of my thesis. It was a truly enriching experience learning from each of them.

This acknowledgements page would be incomplete without mentioning Yu-Wen and David, who were on the same tumultuous tape-out boat as me. We shared some intense, yet exciting times together. I thank them both for the lively technical discussions at the office and equally lively badminton and tennis sessions on the field.

I also thank Job, Bagas, Niels, Ali and Luc for their help during the tape-out and all other Coolgroup members for the enjoyable lunch conversations and social events.

Finally, I am extremely grateful to my parents. Their unconditional love, understanding and support have helped me reach where I am today.

R. Gurbaxani
Delft, October 2022

1

INTRODUCTION

For the past few decades, phase-locked loops (PLLs) have been widely used to realize RF frequency synthesizers for a broad range of applications, such as wireless and wireline transceivers, high-performance analog to digital converters and optical serial data communication links. The ability of a PLL system to produce stable RF frequencies with low phase noise and jitter has not only led to its popularity but also made it an active field of research.

This chapter begins with a brief description of the basic building blocks of a PLL and the circuit design challenges, trade-offs and limitations associated with them. It then arrives at the motivation of this thesis and the targeted specifications for the PLL chip designed as part of this work. Subsequently, the chapter proceeds to delineate the thesis contributions and finally concludes with a description of the complete structure of the thesis.

1.1. THE CLASSICAL PLL

The "Classical PLL" architecture [1], [2] has been shown in Fig. 1.1. It is essentially a feedback network comprising of a fixed input reference frequency, f_{ref}, a phase-detector (PD), a charge-pump (CP), a loop filter (LF), a voltage-controlled oscillator (VCO) producing the RF output frequency, f_{VCO} and a divide-by-N divider. The PD/CP block detects the phase difference between the reference f_{ref} and the divided-by-N VCO frequency, f_{VCO}/N and converts it into a current signal which is integrated over the loop filter to form a voltage that is fed to the VCO as a control signal to tune the RF output frequency so that $f_{VCO} = N.f_{ref}$.

1

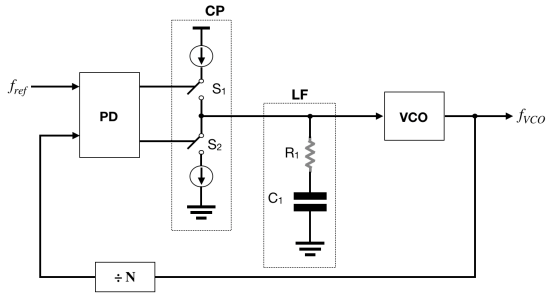

Fig. 1.1. Block diagram of the "Classical PLL" architecture.

A linear phase domain model of the PLL system in the s domain, including noise sources, has been presented in Fig. 1.2.

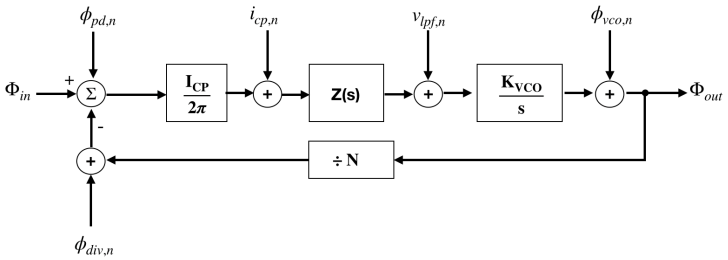

Fig. 1.2. Linear phase domain model of the PLL system.

The closed loop transfer function of the loop is given by:

$$H(s) = \frac{\Phi_{out}(s)}{\Phi_{in}(s)} = \frac{I_{CP}\frac{K_{VCO}}{s}Z(s)}{1+\frac{1}{N}I_{CP}\frac{K_{VCO}}{s}Z(s)} \tag{1.1}$$

where I_{CP} is the charge pump current, K_{VCO} is the sensitivity of the VCO expressed in in Hz/V, $Z(s)$ is the transfer function of the loop filter and has the form $Z(s) = \frac{1+sC_1R_1}{sC_1}$.

The above transfer function is represented in a normalized form as:

$$H(s) = \frac{2\zeta\omega_n s + \omega_n^2}{s^2 + 2\zeta\omega_n s + \omega_n^2} \tag{1.2}$$

where $\omega_n = \sqrt{\frac{I_P}{2\pi C_1}\frac{K_{VCO}}{N}}$ is the natural frequency and $\zeta = \frac{R_1}{2}\sqrt{\frac{I_P C_1}{2\pi}\frac{K_{VCO}}{N}}$ is the damping factor of the system.

The noise sources in Fig. 1.2 are the phase noise spectral densities $\phi_{pd,n}$, $\phi_{vco,n}$ and $\phi_{div,n}$ corresponding to the phase detector, the VCO and the divider,

respectively. Also included are the current and voltage noise spectral densities $i_{cp,n}$ and $v_{lpf,n}$ of the charge pump and the loop filter, respectively.

The output-referred noise contributions from each of the above mentioned sources can be expressed as:

$$\phi^2_{out,pd,n} = N^2|H(s)|^2\phi^2_{pd,n} \tag{1.3}$$

$$\phi^2_{out,cp,n} = N^2\left(\frac{2\pi}{I_{CP}}\right)^2|H(s)|^2 i^2_{cp,n} \tag{1.4}$$

$$\phi^2_{out,lpf,n} = N^2\left(\frac{2\pi}{I_{CP}Z(s)}\right)^2|H(s)|^2 v^2_{lpf,n} \tag{1.5}$$

$$\phi^2_{out,vco,n} = \left|\frac{s^2}{s^2+2\zeta\omega_n s+\omega_n^2}\right|^2 \phi^2_{vco,n} \tag{1.6}$$

$$\phi^2_{out,div,n} = N^2\phi^2_{div,n} \tag{1.7}$$

It can be observed from the above equations that the noise sources attributed to within the loop get multiplied by a factor of N^2. A typical phase noise spectrum of a Classical PLL plotted in [3] is shown in Fig. 1.3.

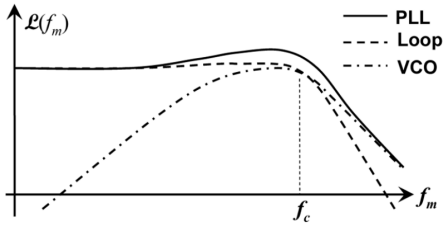

Fig. 1.3. Phase noise spectrum of the Classical PLL [3]. $\mathscr{L}(f_m)$ is the phase noise seen at an offset frequency f_m from the RF carrier. The dominant pole of the PLL system is formed by f_c.

It is clear from the above phase noise spectrum that the loop components, namely the PD and CP are major noise contributors in the PLL system.

1.2. THE SUBSAMPLING PLL

With the aim of reducing the phase noise arising from PD and CP, a subsampling PLL was first introduced in [3] and has since often been used in literature [4], [5], [6], [7]. The subsampling PLL removes the divider and directly samples the VCO output frequency using the reference signal. This technique not only eliminates the noise and power consumption arising from the divider, but also prevents the PD and CP noise from being multiplied by a factor of N^2. In the subsampling PLL of [7], the PD and CP blocks have been replaced by a sampler (made up of a switch and a sampling capacitor) and a G_M cell (see Fig. 1.4). Due to the subsampling

1

nature of the loop, the sampler can be realized with a high phase detection gain, thus suppressing the noise from the G_M cell, loop filter, and the PD itself.

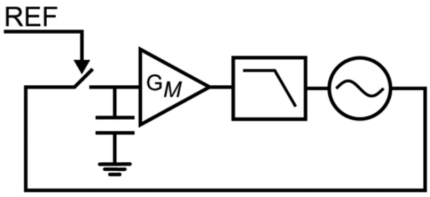

Fig. 1.4. Block diagram of the subsampling PLL [7].

The subsampling PLL thus promises improved PLL performance by significantly lowering phase noise and jitter. However, the direct sampling of the VCO voltage signal by the reference signal results in the formation of spurious tones at frequencies $f_{VCO} \pm m.f_{ref}$, where m is an integer. These tones, referred to as reference spurs are formed due to effects such as periodic modulation of oscillator tank capacitance at the reference clock rate, reference clock feedthrough and charge injection from the sampling switch to the oscillator. An expression to quantify the level of reference spur is given in [8]:

$$S_{REF} = 20.log_{10}\left[sin(\pi.D_{REF}).\frac{N}{2\pi}.\frac{C_{MOD}}{C_{TANK}}\right] \tag{1.8}$$

where D_{REF} is the duty cycle of the reference clock, C_{TANK} is the total capacitance of the LC oscillator tank and C_{MOD} is the modulated capacitance seen by the tank which is equal to the sum of the sampling capacitance used by the sampler and the parasitics in the sampling switch. Thus, it can be seen from equation 1.8 that in order to reduce the reference spurs, both D_{REF} and C_{MOD} must be lowered. Although equation 1.8 is intended for LC oscillators, the results drawn from it can also be extrapolated for inverter-based ring oscillators.

Directly lowering the sampling capacitor can lower C_{MOD} but at the cost of increased $\frac{kT}{C}$ sampling noise. This presents a trade-off during simultaneous reduction of phase noise and reference spur. To overcome this trade-off, an isolation buffer can be inserted between the oscillator and the sampler, and its operation can be reference-gated to save power and reduce D_{REF} [9] (see Fig. 1.5). However, the common-mode settling time of the buffer typically restricts the reference pulse width, T_P to at least 5-10 VCO cycles [8], which in turn limits the suppression of reference spurs.

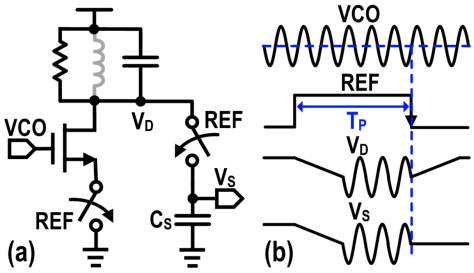

Fig. 1.5. (a) Schematic (b) waveforms of the voltage-sampler using a power-gated buffer [8].

A charge-domain subsampling PLL proposed in [8], [10] reports an appreciable reduction of reference spurs without compromising phase noise and jitter performance. The following section discusses this PLL architecture in detail.

1.3. THE CHARGE-SAMPLING PLL

The main difference between the charge-sampling PLL (CSPLL) [8] and a typical subsampling PLL is in the type of phase detector used by them. Unlike a typical subsampling PLL that samples the instantaneous VCO signal voltage to determine its phase, the CSPLL employs a charge-sampling phase detector (CSPD) that integrates an input current over a capacitor over a fixed time window and samples the resulting voltage. Fig. 1.6 shows the schematic of the CSPD and also gives insight into its operating principle.

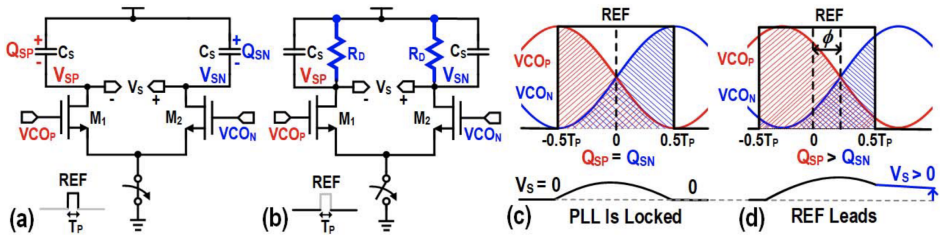

Fig. 1.6. Schematic of the CSPD when the REF pulse is (a) high (b) low; conceptual waveform for the CSPD when the PLL is (c) locked (d) unlocked [8].

Fig. 1.6 (a) shows the CSPD schematic when the reference pulse (REF) is high. During this time, the differential voltage applied to the MOS input pair $M_{1,2}$ is converted into a differential current that charges the C_S capacitors, thus producing a differential voltage at the output of the PD. Fig. 1.6 (c) shows the conceptual waveform for the case when the VCO zero-crossing is aligned with the center of the REF pulse, which is equivalent to the PLL locking condition. Since charge integration occurs symmetrically, the differential output voltage, V_S is zero. On the other hand, if the PLL is unlocked (Fig. 1.6 (d)), the charge integration over the C_S capacitors happens asymmetrically which results in the formation of a

non-zero differential voltage at the PD output. The schematic in Fig. 1.6 shows the behavior of the CSPD when REF is low. The CSPD output common-mode level returns close to V_{DD} via the R_D resistors, while the differential voltage V_S gradually discharges via R_D and C_S. This phase of operation can be thought of as the reset phase of the circuit. It is important to note that the value of R_D is kept quite high, so that it does not play a role during the charge-sampling phase when the REF is high.

The REF pulse width, T_P can be made extremely narrow ($T_P < T_{VCO}/2$) and thus the reference spur levels can be significantly reduced. It has also been shown in [8] that, for a properly designed circuit, the CSPD phase noise (PN) is nearly independent of C_S. Such behaviour is in stark contrast with that of the VSPD, where the value of C_S dictates the trade-off between jitter and spurs.

1.4. THESIS MOTIVATION: DESIGNING A FRACTIONAL-N CHARGE-SAMPLING PLL

The charge-sampling PLL proposed in [8], [10] simultaneously reduces the reference spur levels and phase noise of the PLL system. However, it can only operate in "Integer-N" mode which means that in the relationship $f_{VCO} = N.f_{ref}$, N assumes integer values and thus the VCO output can only lock on to integer multiples of the reference frequency. Unfortunately, high congestion in the spectrum of any modern communication standard has rendered the Integer-N operation largely unacceptable [7] and there is a need to develop high performance PLLs capable of operating in "Fractional-N" mode.

The primary motivation of this thesis is to design a high performance Fractional-N PLL which exploits the useful properties of a charge-domain subsampling PLL.

Unlike the classical PLL, where Fractional-N frequency synthesis is achieved by applying $\Delta\Sigma$ modulation to the divider [11], the subsampling PLL requires a completely new methodology. A technique has been proposed in [7], where phase modulation has been applied to the reference to match with the phase of the fractional VCO frequency. This has been achieved by inserting a digital-to-time converter (DTC) in the reference path as shown in Fig. 1.7. The DTC serves as a programmable delay generator to delay the reference pulse in accordance with the phase of the desired fractional-N VCO frequency. Although the operating principle of this technique has been presented in detail in chapter 2, it is worth mentioning at the outset that the DTC linearity and resolution play a crucial role in determining the performance of the Fractional-N PLL. These factors have been accounted for while selecting the DTC architecture and designing the DTC.

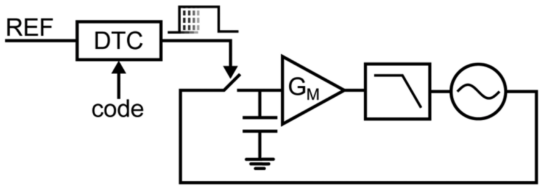

Fig. 1.7. Implementation of Fraction-N frequency synthesis with a subsampling PLL by inserting a DTC in the reference path [7].

It is also important to note that the DTC resolution, also referred to as the DTC LSB, t_{LSB} is susceptible to PVT variations. This calls for the implementation of a background calibration mechanism, preferably implemented in the digital domain, to track these PVT variations and apply appropriate corrections.

1.5. TARGETED SPECIFICATIONS

A ring oscillator (RO) has been chosen in the design of this PLL as it promises area efficiency and ease of silicon integration when compared to LC-based oscillators.

In order to derive the targeted specifications for the PLL designed as part of this thesis, as a first step, a comparison has been drawn with the recently published state-of-the-art ring oscillator-based Fractional-N PLLs (see Table 1.1).

Table 1.1: State-of-the-art RO-based Fractional-N PLLs

	ISSCC'22 C. Hwang [12]	**ISSCC'21 H. Park [13]**	**ISSCC'21 Q. Zhang [14]**	**ISSCC'20 T. Seong [15]**	**JSSC'20 Y. Zhang [16]**	**JSSC'19 A.Santiccioli [17]**
Architecture	DPLL	DPLL	MDLL	DPLL	CP PLL	MDLL
Technology	65nm	65nm	65nm	65nm	40nm	65nm
f_{OUT} (GHz)	4.4 to 5.4	5.2 to 6	0.8 to 2.0	4.5 to 6	1.67 to 3.12	1.6 to 3.0
Freq. resolution (kHz)	6.1	3.1	781.25	3.1	0.1	1960
F_{REF} (MHz)	100	100	50	100	50	100
Worst frac. spur (dBc)	-60	-63	-60	-58	-47	-52
Ref. spur (dBc)	-64	-77	-44	NA	-67	-56
rms jitter (fs)	188 (1k to 30M)	365 (10k to 30M)	1670 (10k to 10M)	648 (1k to 30 M)	2260 (1k to 100M)	397 (30k to 30M)
PN@1MHz (dBc/Hz)	-133.4	-128.8	-111.6	-124.9	-103.9	-122.4
Power (mW)	15.67	9.27	11.95	9.88	4.85	2.5
FoM_{jitter} (dB)	-242.6	-239.1	-224.8	-233.8	-226.1	-244
Area (mm^2)	0.139	0.146	0.180	0.108	0.086	0.0275

[1]

[1] $FoM_{jitter} = 20log_{10}(jitter_{rms}/1s) + 10log_{10}(power_{DC}/1mW)$

Aiming for a low power consumption design, a power budget of 1mW has been kept. To achieve a PLL Figure of Merit (FoM_{jitter}) of -240 dB, while honoring the aforementioned power budget, the rms jitter must be below $1ps$.

In a typical PLL, the oscillator's power consumption is dominant among all PLL blocks and can be estimated for a ring oscillator as:

$$Power_{osc} = C_{load} V_{DD}^2 . f_{OUT} \tag{1.9}$$

where f_{OUT} is the oscillator output frequency and C_{load} is the combined load driven by the oscillator.

Upon allocating the entire power budget to the oscillator and assuming that it drives a combined load of $C_{load} = 1pF$, the value of f_{OUT} is obtained around 1 GHz. For $f_{OUT} = 1GHz$ and rms jitter of $1ps$, equation 3.13, yields an in-band phase noise (PN) of $-114dBc/Hz$.

The targeted specifications are thus summarized in Table 1.2 :

Table 1.2: System parameter values

Specification	**Value**
Architecture	CSPLL
Technology	40nm CMOS
Freq. resolution	< 10 kHz
Worst frac. spur	< -60dBc
Ref. spur	< -75 dBc
rms jitter (fs)	1000
In-band phase noise (for 1 GHz f_{OUT})	-114dBc/Hz
Power(mW)	1
FoM_{jitter} (dB)	-240

1.6. THESIS CONTRIBUTIONS

The research contributions of this thesis are as follows:

1. A comprehensive understanding of state of the art PLL architectures has been developed with a special focus on realizing a Fractional-N frequency synthesizer utilizing a charge-domain subsampling PLL.

2. A literature review of the state of the art digital to time conversion techniques has been performed and a highly linear, high-resolution DTC has been designed.

3. A charge-domain subsampling PLL system has been designed and the DTC has been incorporated within this system to enable Fractional-N frequency synthesis.

4. A highly digital background DTC gain calibration mechanism has been designed to ensure robust Fractional-N PLL operation over PVT variations.

5. The complete PLL system has been designed and implemented in a 40-nm, 1.1V CMOS process.

1.7. THESIS STRUCTURE

The thesis has been structured as follows:

- Chapter 2 explains how a DTC can be introduced in the subsampling PLL to enable Fractional-N frequency synthesis. It also presents a literature review of the state of the art digital to time conversion techniques and motivates the selected DTC topology.
- Chapter 3 provides a block level description of the PLL system and discusses loop stability and noise sources using the linear phase domain model. This chapter also emphasizes the need for a DTC gain calibration mechanism and defines an approach to implement it.
- Chapter 4 elaborates on the design details pertaining to the analog and RF circuit blocks of the PLL system.
- Chapter 5 discusses the design of digital logic blocks required to implement the DTC gain calibration.
- Chapter 6 presents the full PLL chip simulation results and summarizes the PLL performance.
- Chapter 7 concludes the thesis and gives an overview of future work.

2

DIGITAL TO TIME CONVERTER

In the previous chapter, it was briefly illustrated how phase modulation can be introduced in the reference path by utilizing a digital-to-time converter (DTC) and enabling Fractional-N subsampling operation (Fig. 1.7).

This chapter presents a more detailed overview of a DTC-equipped subsampling PLL undergoing Fractional-N operation. It also compares the state of the art digital to time conversion techniques and motivates a suitable DTC topology for this work.

2.1. A DTC-EQUIPPED SUBSAMPLING PLL

As previously mentioned in section 1.4, a DTC can be employed within a subsampling PLL to modulate the phase of the reference frequency in order to match the phase of the fractional-N VCO frequency. This section explains this concept with the help of an example. Fig. 2.1 shows Integer-N operation for a subsampling PLL with $N = 2$. For a locked PLL, the sampling events take place at zero crossings of the VCO signal.

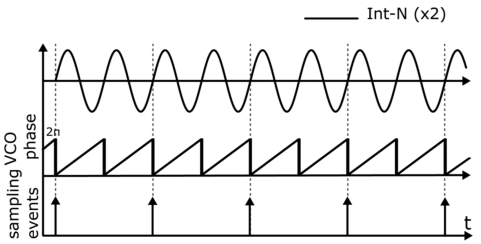

Fig. 2.1. Integer-N operation in a Subsampling PLL [7].

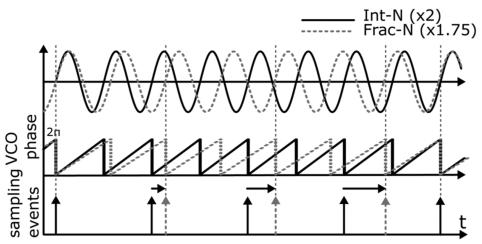

Fig. 2.2. Fractional-N operation in a Subsampling PLL [7].

The Fractional-N operation with $N = 1.75$ has been depicted in Fig. 2.2. The first sampling event happens at the same time as that for the integer-N case. For the next sampling event, there is a phase error of $0.25T_{VCO}$ between the reference and the VCO signal. To ensure PLL locking, this phase error can be compensated for by advancing (delaying) the reference by $0.25T_{VCO}$ in the time domain. Subsequent sampling events are constructed by advancing the reference by $0.5T_{VCO}$ and 0.75_TVCO. For the final sampling event shown in Fig. 2.2, the reference and VCO phases re-align which obviates the need to advance the reference at all. It can thus be observed that a periodic pattern emerges due to the "phase-wrapping" effect with a delay of $0.25T_{VCO}$ that accumulates every sampling event and the DTC must provide delay generation of up to one VCO period.

In order to achieve the above described phase modulation, the DTC must be appropriately programmed through a code generator. The DTC code generation logic can be implemented in the digital domain and its block diagram and computation flow are illustrated in Fig. 2.3.

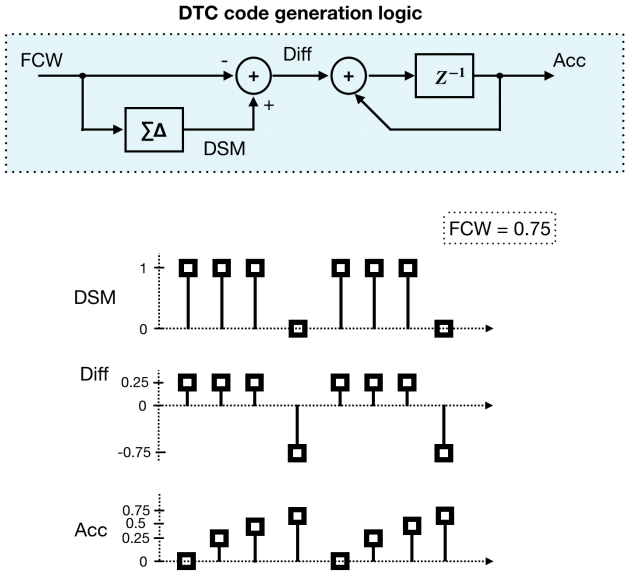

Fig. 2.3. Block diagram of the DTC code generator and its computation flow.

The Fractional command word (FCW) is fed as input to the digital DTC code generation logic. The computation flow in Fig. 2.3 has been shown for a 1st order ΔΣ modulator.

2.2. IMPACT OF DTC LINEARITY ON PLL PERFORMANCE

For a PLL undergoing Fractional-N operation, the DTC nonlinearity results in increased spurious tones at the PLL output. This effect can be better understood with the following example.

For a PLL having a reference frequency $f_{ref} = 100MHz$ and synthesizing a fractional output tone at $f_{VCO} = 1.001GHz$, the value of $N = 10.01$ and $FCW = 0.01$. Based on the functioning of the DTC code generation block (Fig. 2.3), the output of the accumulator, Acc will have a profile illustrated in Fig. 2.4. Since the $FCW = 0.01$, it will take 100 reference clock cycles before the accumulator is reset and begins to repeat the code pattern. For the reference having a clock period of $10ns$, this amounts to $1\mu s$.

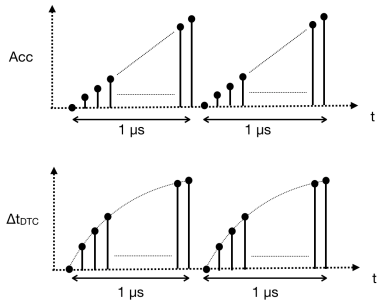

Fig. 2.4. DTC code generator output and the corresponding delay generated by a nonlinear DTC.

Even though the accumulator accumulates the DTC code in a linear fashion, the inherent DTC nonlinearity will manifest itself in the output time delay, Δt_{DTC} produced by the DTC, as shown in Fig. 2.4. During the sampling procedure of the phase detector, this nonlinear DTC characteristic will produce phase errors that exhibit a periodic pattern that repeats every $1\mu s$. These phase errors will be converted into voltage signals by the phase detector, which will be fed as the control voltage to the VCO, thereby producing spurious tones at offsets of multiples of $1/1\mu s = 1MHz$ from the fundamental carrier. These spurious tones are also referred to as fractional spurs.

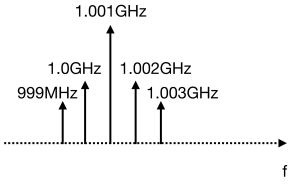

Fig. 2.5. Spectrum showing fractional spurs due to DTC nonlinearity produced at the PLL output.

The spectrum of Fig. 2.5 shows the fundamental VCO tone produced at $1.001GHz$ and the corresponding fractional spurs. The level of the highest fractional spur can be estimated as [18]:

$$\mathscr{L}_{inl,spur} = \frac{\pi^2}{4} \cdot \left(\frac{\Delta t_{inl}}{T_{VCO}}\right)^2 \tag{2.1}$$

where Δt_{inl} is the DTC integral nonlinearity (INL) and T_{VCO} is the VCO time period.

2.3. CHOICE OF DTC

A DTC can be described as a combination of a voltage ramp generator followed by a threshold comparator as shown in Fig. 2.6.

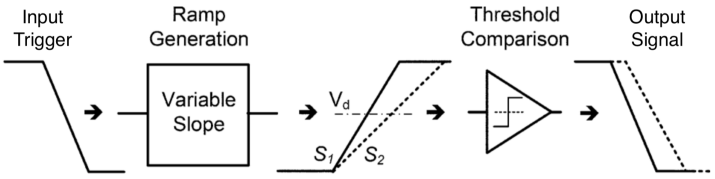

Fig. 2.6. Operation of a variable-slope DTC (Image modified from [19]).

The ramp generator itself can be modelled as a current source I, charging a capacitor C (Fig. 2.7), while the threshold comparator is designed as an inverter or an inverter chain.

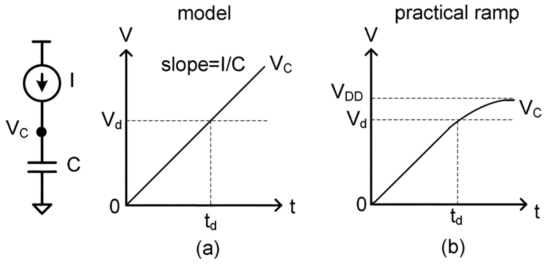

Fig. 2.7. Model of a ramp generator: (a) generating an ideal ramp voltage (b) generating a practically achievable ramp [19].

An input signal triggers the ramp generation block. The ramp voltage is compared against a threshold voltage V_d by the threshold comparator, which produces the output signal when the ramp voltage level equals V_d. If the ramp takes time t_d to go from zero voltage to V_d, t_d can be expressed as:

$$t_d = \frac{V_d}{S} \tag{2.2}$$

where $S = I/C$ is the slope of the ramp (see Fig. 2.7). To achieve programmability of t_d, one option is to vary the slope S of the ramp by varying either the current I or the capacitance C. This approach of DTC design is termed as the "variable-slope" approach (see Fig. 2.6). This approach has been exploited in [20] by using a switched capacitor load. Although it is possible to achieve a good resolution in delay with a variable-slope DTC, it fares poorly in terms of linearity due to the inherent nonlinear relationship between the ramp slope and the threshold comparison. This non linearity is reflected in the varying slope of the output signal as shown in Fig. 2.6.

The second option of varying the delay t_d is to vary the threshold voltage V_d for a fixed ramp slope. However, due to limitations posed by practical ramp generation, the ramp itself is not linear and hence the delay steps produced will also suffer from non-linearity. Moreover, implementing a linear variation in V_d may not prove straightforward.

An interesting approach has been presented in [19], where the ramp starting voltage, V_{st} is varied while the ramp slope remains constant. This approach of DTC design is termed as the "constant-slope" approach and is illustrated in Fig. 2.8.

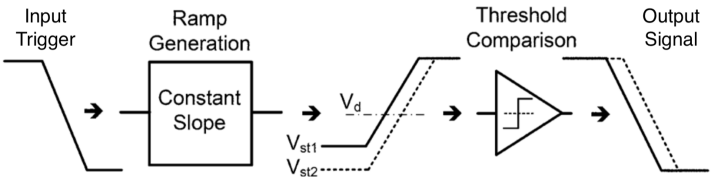

Fig. 2.8. Operation of a constant-slope DTC (Image modified from [19]).

Considering the case of a practical ramp being fed as input to a practical threshold comparator, it must be observed that neither the ramp slope is constant from GND to VDD, nor the threshold of the comparator is confined to single voltage V_d. As explained in [19], the threshold voltage can be thought of as a "threshold window", ranging from V_{th0} to V_{th1}, as depicted in Fig. 2.9.

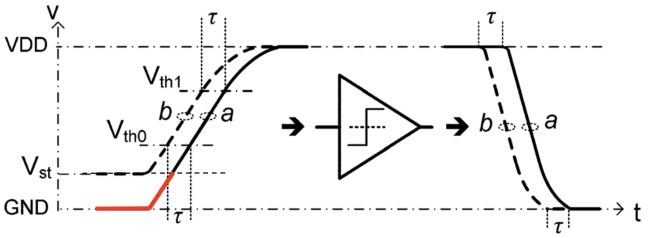

Fig. 2.9. Operation of a practical constant-slope DTC. The range over which the ramp starting voltage, V_{st} is varied is shown in red. For the ramps a and b, the comparator produces output signals having an overall identical profile with a delay of τ between them (Image modified from [19]).

Since the bulk of the nonlinear effects arise in the threshold window, the success of the constant-slope DTC is contingent on varying the ramp start voltage over a range that lies away from the threshold window. Additionally, it can be seen from Fig. 2.9 that the section of the ramp close to GND is linear and thus can be utilized to generate linear voltage steps.

The constant-slope DTC proposed in [19] utilizes a current domain digital-to-analog converter (I-DAC) to generate a varying ramp starting voltage. The top-level block diagram of this DTC is shown in Fig. 2.10.

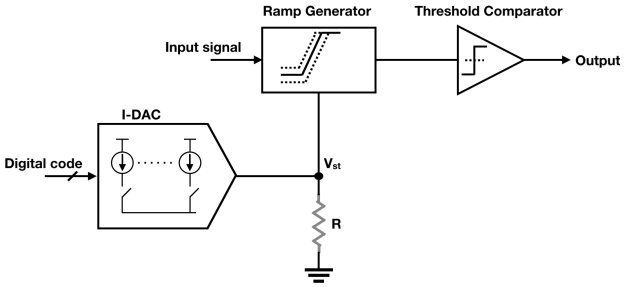

Fig. 2.10. Top-level block diagram of the I-DAC-based constant-slope DTC.

In addition to linearity, a factor that merits consideration during the design of a DTC is its power consumption. While the I-DAC-based constant slope DTC [19] promises good linearity and fine resolution it consumes a power of $1.8mW$. The major contribution of power consumption in this design is attributed to the I-DAC. An alternative approach has been proposed in the form of a Capacitor-DAC (C-DAC) based constant-slope DTC in [21] that offers similar linearity and resolution while consuming a mere $31\mu W$ of power. The C-DAC-based constant slope DTC architecture has been chosen in this thesis and its operating principle is described in the following section.

2.4. C-DAC-Based Constant Slope DTC

The C-DAC-based constant-slope DTC comprises of three main circuit blocks: a ramp generator, a threshold comparator and the C-DAC itself. These are illustrated in the DTC schematic in Fig. 2.11. The ramp generator is made up of an inverter and a capacitor, C_R while the threshold comparator is also made up of an inverter. The C-DAC consists of a digitally controlled capacitor array with unit capacitance represented by C_u and a fixed capacitance C_O, where typically $C_O >> C_u$. For the purpose of explaining the DTC operating principle The capacitor array has been depicted as a binary array. In practice, some segmentation can be applied to the array is as later shown in chapter 4.

It is important to note that this DTC works with a *discharging* ramp rather than the *charging* ramp that was discussed in the previous section. The voltage V_{common} serves as the ramp start voltage V_{st} and is dynamically adjusted by the C-DAC.

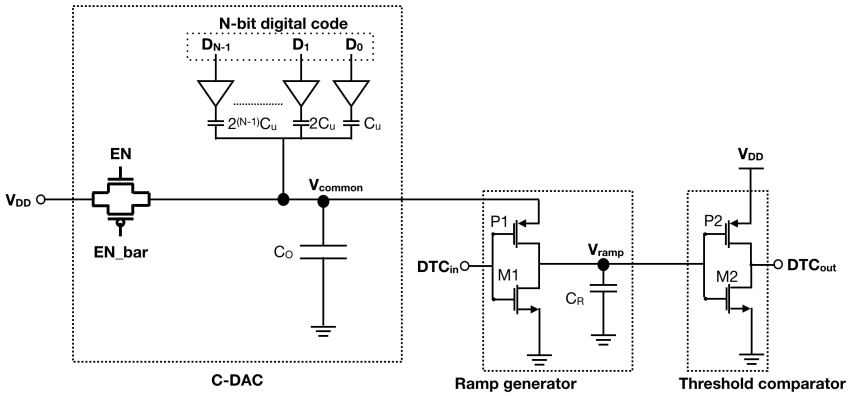

Fig. 2.11. Schematic of the C-DAC-based constant-slope DTC.

The threshold voltage of the threshold comparator can be adjusted by relative sizing of the NMOS $M2$ and PMOS $P2$ in Fig. 2.11. Ideally, since a discharging ramp is being used, the NMOS $M2$ must be sized much larger than the PMOS $P2$ to ensure that the threshold voltage remains below $V_{DD}/2$ while the ramp start voltage ranges from V_{DD} to $V_{DD} - \Delta$ where Δ is usually kept small (around $200\mu V$). Thus the basic function of the DTC can been seen as the generation of a variation ΔV_{st} in the ramp start voltage to produce a time delay Δt at the DTC output signal (Fig. 2.12).

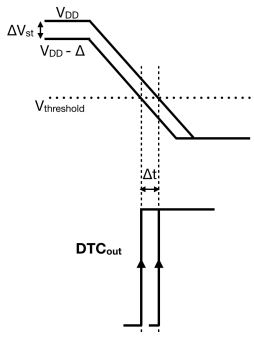

Fig. 2.12. Basic DTC function.

In order to achieve a variation in the ramp start voltage, the C-DAC is subjected to a two-phase operation, as shown in Fig. 2.13. In the first phase Φ_1, the EN signal is made high and the capacitance attached to the V_{common} node gets pre-charged to V_{DD} via the pass gate. The digital code is configured to selectively charge the capacitors in the array by either driving a zero voltage (the capacitors in black which do get charged) or by driving a V_{DD} voltage (the grey-shaded part of the array that does not get charged).

In the Φ_2 phase, the digital interface drives a zero voltage to all the capacitors in the array and this results in charge re-distribution at the V_{common} node, leading to a drop in its voltage level, as shown in the waveform in Fig. 2.13.

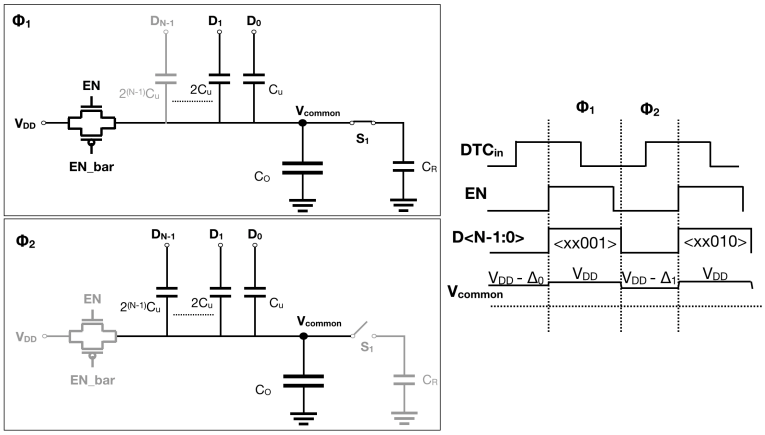

Fig. 2.13. Two-phase operation of the C-DAC.

The switch S_1 in Fig. 2.13 represents the ramp generator PMOS device $P1$ shown in Fig. 2.11. The switch is closed when the DTC_{in} signal goes low. Due to the overlap between the EN signal and the $\overline{DTC_{in}}$ signal, the ramp capacitance C_R also gets charged to V_{DD} and can be considered as a fixed capacitance during the

DTC operation.

The starting voltage of the ramp, referred to as V_{ramp} is made equal to V_{common} as switch S_1 is closed and can thus be computed based on the two-phase operation of the C-DAC as:

$$V_{ramp} = \frac{C_{fixed} + C_1}{C_{fixed} + C_2}.V_{DD} \tag{2.3}$$

where $C_{fixed} = C_O + C_R$, while C_1 is the part of the capacitor array selected during Φ_1, given by $C_1 = \Sigma_{i=0}^{N-1}(1 - D < i >)2^i C_u$ and C_2 is the total capacitance of the array, given by $C_2 = \Sigma_{i=0}^{N-1} 2^i C_u$. The notation $D < i >$ represents the bit value at the i^{th} bit position in the digital code and can assume a value of either 0 or 1. Thus for an increasing digital code in phase Φ_1, the value of C_1 decreases which results in a lower V_{ramp} voltage.

The complete timing diagram for the DTC has been sketched in Fig. 2.14.

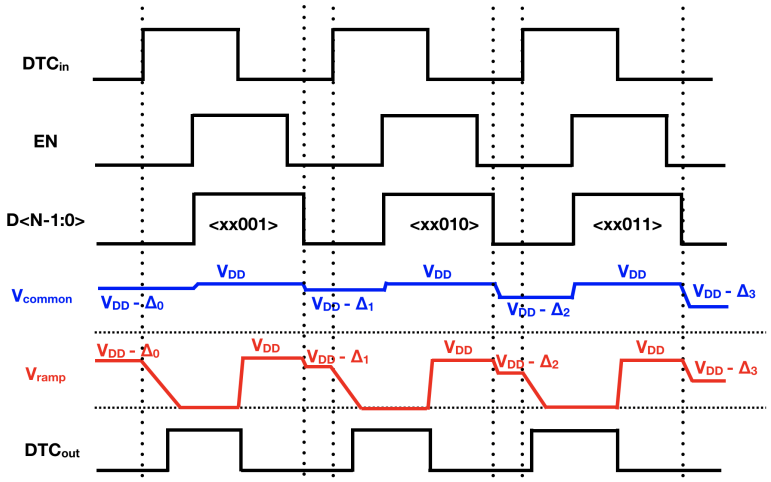

Fig. 2.14. Timing diagram of the DTC.

3

BLOCK LEVEL SYSTEM DESCRIPTION

This chapter gives an overview of the complete PLL system. This includes a description of the loop dynamics and phase noise contributions of the system blocks using an s-domain model. Additionally, this chapter also explains the motivation for the background DTC gain calibration technique used in the system.

3.1. PHASE DOMAIN MODEL

The block diagram of the PLL system is presented in Fig. 3.1.

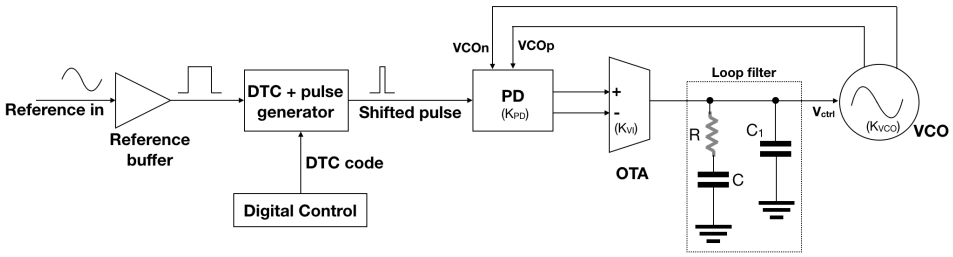

Fig. 3.1. PLL system block diagram.

A 100 MHz sinusoidal reference is fed as input from an off-chip crystal. The reference buffer converts it into a 100 MHz square wave with an amplitude of $1.1V_{pp}$. The DTC is used for fractional-N frequency synthesis and can be made dormant during integer-N operation using the digital control block. A pulse generator is used in conjunction with the DTC to produce pulse widths appropriate to maintain the gain and output CM levels of the phase detector (PD).

The PD gain is given by K_{PD} in V/rad. A differential input, single-ended output OTA converts the PD output voltage to a current that is integrated over the low-pass loop filter. K_{VI} is the transconductance of the OTA. The voltage-controlled oscillator (VCO) is made up of a ring oscillator that has a frequency tuning range from 1 to 2 GHz and is controlled by the control voltage V_{ctrl}. The PD receives feedback in the form of differential phases VCO_n and VCO_p from the VCO.

3.1.1. LOOP DYNAMICS IN S-DOMAIN

A linear phase domain model has been employed to describe the loop dynamics in the s domain. This model is illustrated in Fig. 3.2.

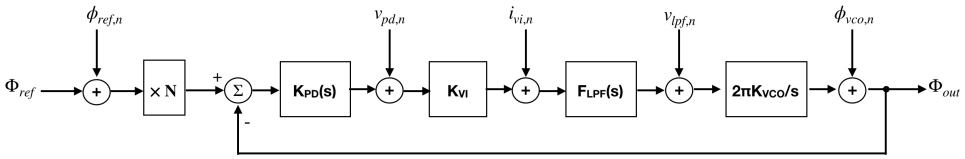

Fig. 3.2. Phase domain model of the PLL.

The closed-loop transfer function of the PLL can be given by:

$$H_{cl}(s) = \frac{H_{ol}(s)}{1 + H_{ol}(s)} \tag{3.1}$$

where H_{ol} is the open-loop transfer function and is expressed as:

$$H_{ol}(s) = K_{PD}(s).K_{VI}.F_{LPF}(s).\frac{2\pi K_{VCO}}{s} \tag{3.2}$$

which can be expanded as:

$$H_{ol}(s) = \frac{K_{PD}}{1 + sR_DC_S}.K_{VI}.\left(\frac{sCR + 1}{(sC)(1 + sC_1R)}\right).\frac{2\pi K_{VCO}}{s} \tag{3.3}$$

R_D and C_S form the load impedance of the PD.
The closed-loop transfer function can thus be obtained as:

$$H_{cl}(s) = \frac{H_{ol}(s)}{1 + H_{ol}(s)} = \frac{2\pi(K_{PD}.K_{VI}.K_{VCO})(1 + sCR)}{(1 + sCR)2\pi(K_{PD}.K_{VI}.K_{VCO}) + s(sC)(1 + sC_SR_D)(1 + sC_1R)} \tag{3.4}$$

The loop is typically designed such that $C_S << C$ and $C_1 << C$ leading to a simplified closed-loop transfer function that resembles a second order system:

$$H_{cl}(s) = \frac{s.RK_{PD}K_{VI}K_{VCO} + K_{PD}K_{VI}K_{VCO}/C}{s^2 + s.RK_{PD}K_{VI}K_{VCO} + K_{PD}K_{VI}K_{VCO}/C} = \frac{2\zeta\omega_n s + \omega_n^2}{s^2 + 2\zeta\omega_n s + \omega_n^2} \tag{3.5}$$

where the natural frequency, ω_n and the damping factor ζ are given by:

$$\omega_n = \sqrt{\frac{K_{PD}K_{VI}K_{VCO}}{C}} \tag{3.6}$$

$$\zeta = \frac{R}{2}\sqrt{K_{PD}K_{VI}K_{VCO}C} \tag{3.7}$$

The loop bandwidth can be expressed in terms of ω_n and ζ as:

$$f_{-3dB} = \frac{\omega_n}{2\pi}\sqrt{(2\zeta^2+1)+\sqrt{(2\zeta^2+1)+1}} \tag{3.8}$$

An estimate of the system's phase margin can be given by:

$$PM \approx tan^{-1}(\omega_u RC) - tan^{-1}(\omega_u RC_1) - tan^{-1}(\omega_u R_D C_S) \tag{3.9}$$

where $\omega_u \approx \omega_n\sqrt{2\zeta^2+\sqrt{4\zeta^4+1}}$ is the frequency at which $|H_{ol}(s)| = 1$.

The system has been designed taking into account loop stability by ensuring a loop bandwidth of about 10% of the reference frequency. Having a high loop bandwidth results in an improved lock-in range and reduced settling time [22]. Table 3.1 lists the values that have been chosen for the design parameters.

Table 3.1: System parameter values

Design parameter	Value
K_{PD}	0.31 V/rad
K_{VI}	$40\mu S$
K_{VCO}	20 MHz/V
R_D	$50k\Omega$
C_S	200 fF
R	$30k\Omega$
C	20 pF
C_1	100 fF

In addition to ensuring a loop bandwidth of $10MHz$, several individual block-level design considerations determine the choice of these parameters and have been explained in chapter 4.

A system Bode diagram has been plotted with the aid of MATLAB and is presented in Fig. 3.3.

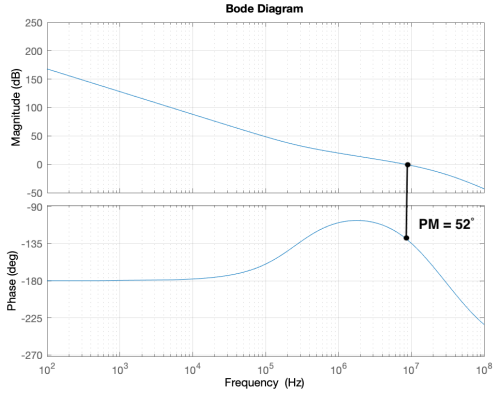

Fig. 3.3. Bode diagram of the PLL open loop transfer function

3.1.2. PHASE NOISE CONTRIBUTIONS

Various noise sources have been shown in the phase domain model of Fig. 3.2. The total phase noise seen at the output of the PLL can be expressed as:

$$\phi^2_{out,n} = |H_{cl}(s)|^2 N^2 \phi^2_{ref,n} + |H_{cl}(s)|^2 \left(\frac{1+sC_S R_D}{K_{PD}}\right) v^2_{pd,n} + |H_{cl}(s)|^2 \left(\frac{1+sC_S R_D}{K_{PD}K_{VI}}\right) i^2_{vi,n} + |H_{cl}(s)|^2 \left(\frac{1+sC_S R_D}{K_{PD}K_{VI}F_{LPF}(s)}\right) v^2_{lpf,n} + \left|\frac{1}{1+H_{ol}(s)}\right|^2 \phi^2_{vco,n} \tag{3.10}$$

The power spectral densities (PSD) of the noise sources can all be obtained from simulations. The PSD of the voltage and current noise from the loop filter and OTA respectively can also be computed as:

$$v^2_{lpf,n} = 4kTR \tag{3.11}$$

$$i^2_{vi,n} = 4kT\gamma K_{VI} \tag{3.12}$$

The relation between phase noise and rms jitter, $\sigma_{\phi,rms}$ is defined as [23]:

$$\sigma_{\phi,rms} = \frac{\sqrt{2\int_{f_{min}}^{f_{max}} \mathscr{L}(\Delta f).d(\Delta f)}}{2\pi f_0} \tag{3.13}$$

where $\mathscr{L}(\Delta f)$ is the phase noise spectral density at an offset frequency of Δf from the oscillator output frequency, f_0. The frequencies f_{min} and f_{max} are the limits of the integration window. The phase noise contributions from the PLL components are plotted in chapter 6.

3.2. INTRODUCING A DTC GAIN CALIBRATION LOOP

The DTC gain, denoted by $K_{DTC} = T_{VCO}/t_{LSB}$ is the ratio of the time period of the fractional frequency to be produced and the LSB of the DTC. Since the range of the DTC (and also t_{LSB}) is susceptible to PVT variations, K_{DTC} can not be programmed accurately. This induces non-linearity in the DTC operation and results in significant levels of fractional spurs [7]. A 9-bit DTC has been designed in this work and the variation in its range over corners is shown in table 3.2.

Table 3.2: Variation in DTC range over corners

Corner	DTC range (ps)
TT	780
FF	698
SS	935

In literature, this problem has been mitigated by designing a background DTC gain calibration loop utilizing the Least-mean squares (LMS) algorithm [18], [24]. However, these implementations have been designed for digital PLLs and it has been found in literature that for an analog PLL such as the one being implemented as part of this thesis, applying the LMS-based calibration technique is not straightforward [25], [7].

An integral part of the LMS-based calibration technique is the determination of the sign or polarity of the phase error between the fractional-N VCO signal and the reference. In [7], the sign of the current has been extracted at the output of the OTA in order to gauge the sign of phase error. However, extracting the sign of a voltage can prove less cumbersome when compared to that of a current. This is evident in [25], where the phase error polarity has been extracted from the output of the phase detector via a comparator. The sampling phase detector used in [25] has a single-ended architecture as opposed to the differential nature of the charge-sampling phase detector being used as part of this work, hence that technique can not be applied here.

This thesis proposes a novel technique to faithfully retrieve the phase error polarity in the voltage domain such that it can be utilized by the LMS-based DTC gain calibration loop. Fig. 3.4 illustrates the complete PLL system incorporating the DTC gain calibration loop.

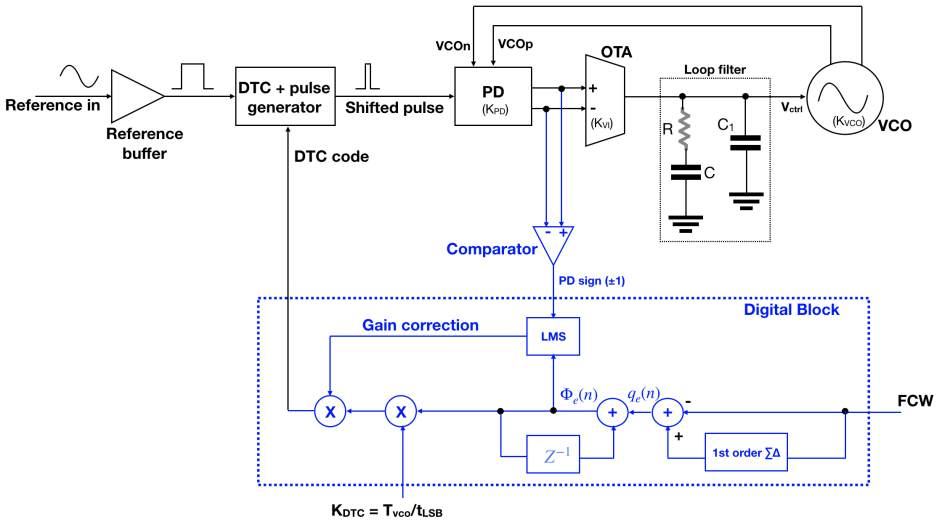

Fig. 3.4. Block diagram of the PLL system with DTC gain calibration loop

A comparator has been employed to detect the deviation in the DTC gain by sampling the output of the CSPD during fractional-N frequency synthesis. At the heart of the calibration loop lies a digital implementation of the least-mean squares (LMS) algorithm. The comparator output and the accumulated fractional shift $\Phi_e(n)$, are fed as inputs to the LMS block, which determines the gain correction term to be applied to K_{DTC}. Details of the LMS implementation and calibration loop simulation results are presented in chapter 5.

A serious challenge to the success of the proposed calibration technique is posed by the offsets in the comparator and the OTA. The ideal locking point of the CSPD is when the zero crossings of VCO signal align with the center of the reference pulse as shown in Fig. 3.5 (a), while Fig. 3.5 (b) shows deviation in the locking point due to offset in the OTA.

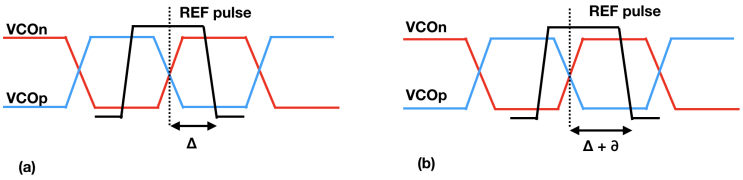

Fig. 3.5. CSPD locking point: (a) Without offset (b) Offset in PD or OTA

Although an offset in the CSPD alone will also result in the situation shown in Fig. 3.5 (b), it is not detrimental since the PD differential output sampled by the comparator will still be zero if the PLL is locked. However, PLL locking in

the presence of an offset in the OTA, $V_{OS,OTA}$ would result in the development of a differential voltage $V_{OS,PD}$ at the output of the CSPD, which is equal to $V_{OS,OTA}$ and opposite in polarity (Fig. 3.6). This voltage $V_{OS,OTA}$ erroneously adds to the actual DTC gain error and can not be rectified by the LMS algorithm.

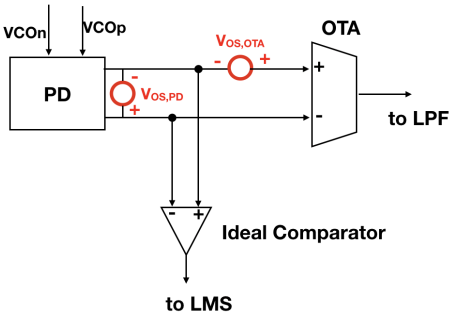

Fig. 3.6. Offset in OTA

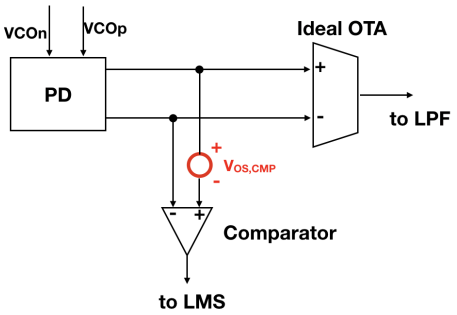

Fig. 3.7. Offset in comparator

Another source of offset is in the comparator itself, $V_{OS,CMP}$ (Fig. 3.7). The situations in both Figs. 3.6 and 3.7 present an identical threat to the decision making process of the comparator. It is interesting to note that for the operation of the comparator, it is possible to sum the offset voltages $V_{OS,OTA}$ and $V_{OS,CMP}$ as $V_{OS,EFF}$, which forms the effective offset seen by the comparator (Fig. 3.8). This effective offset must be removed or compensated to ensure proper functioning of the DTC gain calibration loop. To achieve this, a digital foreground offset compensation scheme in the comparator has been proposed. The merit of this scheme is that it compensates the cumulative offset in the OTA and comparator ($V_{OS,EFF}$) by applying additional circuitry in only the comparator. Details of the foreground offset compensation are presented in chapters 4 and 5.

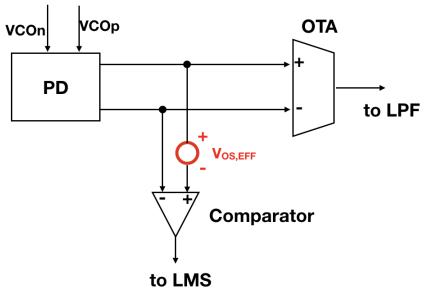

Fig. 3.8. Effective offset as seen by the comparator

Specification for the resolution upto which the effective offset must be compensated has been derived by simulating Fractional-N PLL operation

together with the background calibration loop and offset voltage $V_{OS,CMP}$ applied to the input of the comparator. A very realistic 10% DTC gain error has also been included as part of this simulation.

It can be seen from Fig. 3.9 that the un-calibrated PLL fares poorly with a worst case fractional spur as high as -19dBc, while Fig. 3.10 shows the frequency spectrum for the calibrated PLL. Figs. 3.11 - 3.13 show the efficacy of the calibration loop upon application of an offset voltage to the input of the comparator. As per expectation, the level of the fractional spur decreases with decreasing offset and an acceptable value is achieved for an offset of $100\mu V$. Consequently, it can be concluded that there is a need to compensate the effective offset seen by the comparator with a resolution better than $100\mu V$.

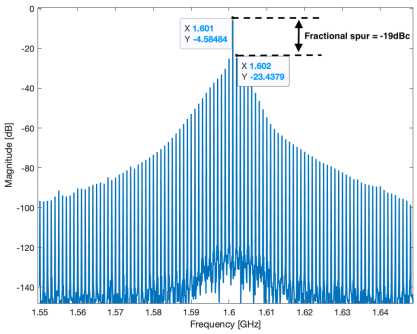

Fig. 3.9. PLL output spectrum for fractional-N frequency synthesis without DTC gain calibration

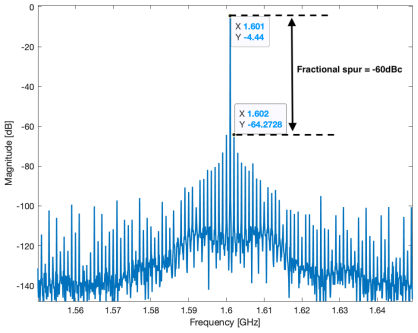

Fig. 3.10. PLL output spectrum with DTC gain calibration enabled and zero offset voltage applied to comparator

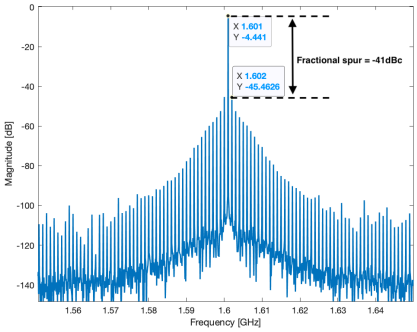

Fig. 3.11. PLL output spectrum with DTC gain calibration enabled and 1mV offset voltage applied to comparator

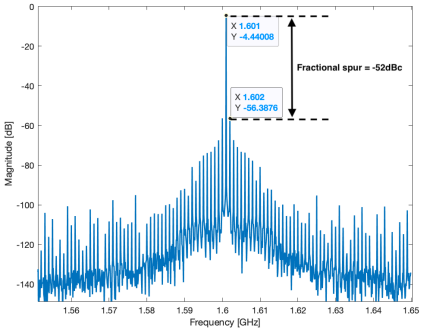

Fig. 3.12. PLL output spectrum with DTC gain calibration enabled and 250μV offset voltage applied to comparator

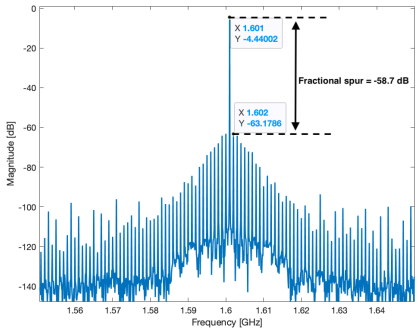

Fig. 3.13. PLL output spectrum with DTC gain calibration enabled and 100μV offset voltage applied to comparator

Applying the above mentioned offset voltage values to the input of the OTA instead of the comparator yields similar results, confirming the hypothesis regarding equivalence of $V_{OS,OTA}$ and $V_{OS,CMP}$ while considering comparator operation in the calibration loop.

Thus the calibration of DTC gain has been performed following a two-step procedure: First, the effective offset seen by the comparator is calibrated using a foreground calibration scheme. The PLL operates in Integer-N mode during this step to completely exclude the DTC gain error. Once the comparator offset is calibrated, the second step is initiated to make the PLL operate in Fractional-N mode by turning on the DTC and the background gain calibration loop.

4

Analog and RF circuit design

This chapter aims to provide detailed descriptions of the circuit design of analog and RF blocks in the PLL.

4.1. Digital to time converter

As explained in Chapter 2, a constant-slope capacitive DAC-based digital-to-time converter has been used to aid the Fractional-N frequency synthesis. Its principle of operation has also been described in Chapter 2. This section provides the design details of the DTC.

4.1.1. Deriving number of DTC bits

DTC Resolution

If the resolution of the DTC is Δt, the phase error due to quantization noise is given by:

$$\Delta\Phi_{DTC} = 2\pi f_{vco}\Delta t \tag{4.1}$$

For worst case phase error, f_{vco} can be taken as 2 GHz, which is the maximum frequency generated by the oscillator.

The rms value of the phase noise due to DTC quantization noise can be expressed as:

$$P_{n,rms,dtc} = \frac{\Delta\Phi_{DTC}^2}{12} \tag{4.2}$$

and the noise power spectral density is then given by:

$$S_{n,dtc} = \frac{P_{n,rms,dtc}}{f_{ref}} = \frac{\Delta\Phi^2}{12 f_{ref}} = \frac{(2\pi f_{vco}\Delta t)^2}{12 f_{ref}} \tag{4.3}$$

The specification for the PLL in-band phase noise is $S_{n,dtc} = -110dBc/Hz$. Keeping the quantization noise contribution 10dB below the in-band phase noise and $f_{ref} = 100MHz$, the DTC resolution can be obtained as $\Delta_t = 2.75ps$.

DTC RANGE

The range of the DTC is the time period of the smallest frequency produced by the oscillator which is 1 GHz. Hence the DTC range is 1 ns.

The number of DTC bits required to meet the above specifications are:

$$N_{DTC\ bits} = log_2\left(\frac{range}{resolution}\right) = 8.5 \tag{4.4}$$

A 9 bit DTC has been designed in this work.

4

4.1.2. SIZING THE CIRCUIT COMPONENTS

The schematic of the DTC has been shown in Fig. 4.1.

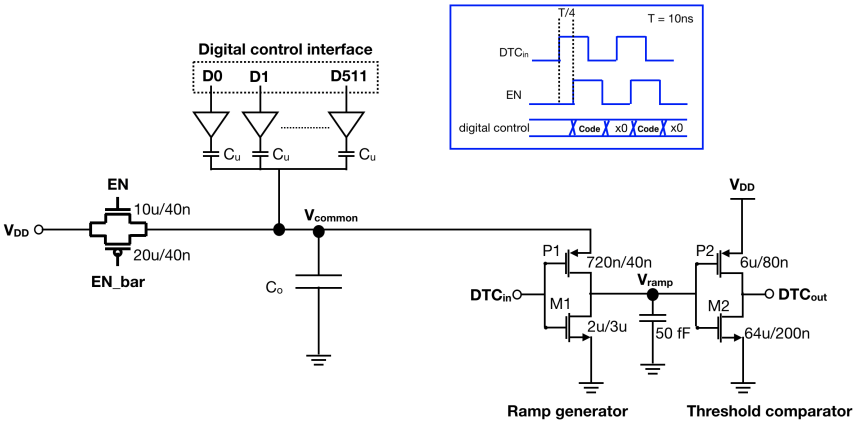

Fig. 4.1. DTC schematic

Based on charge redistribution during the two phase operation of the DAC, an expression defining the relationship between the unit capacitance of the DAC array C_u, the fixed large capacitance C_o and the voltage step V_{step} generated by the DAC at node V_{common} can be defined as:

$$V_{step} = \frac{C_u}{C_o + mC_u}.V_{DD} \tag{4.5}$$

where $m = 2^9$.

During the EN high phase, the capacitor bank of the DAC charged to V_{DD} and the worst case sampling noise is given by $\overline{V}_{n,sample} = kT/C_o$.

By selecting $V_{step} = 500\mu V$, the ramp start voltage at node V_{common} ranges between 1.1 V to 844 mV. By keeping $\overline{V}_{n,sample} = \frac{1}{15}V_{step}$, C_o can be computed to be 3.5 pF. C_o and V_{step} can be substituted in equation 4.5 to get $C_u = 2fF$.

The MOS device M1 in the ramp generator of Fig. 4.1 has a long length to ensure that the ramp is slow enough to accommodate the DTC range. The NMOS device M2 is sized to be much stronger than the PMOS device P2 in order to bring the threshold of the threshold comparator to 400mV. This is useful in keeping the non-linear section of the ramp near the threshold away from the linear range of operation which is 1.1 V to 844 mV.

4.1.3. SEGMENTATION CONSIDERATIONS IN THE DAC

The capacitive DAC can be designed as a binary weighted DAC or a thermometer-encoded DAC or in a segmented fashion, which is a combination of the two. A comparative analysis between binary weighted and thermometer-encoded DACs is presented in [26]. Briefly, a thermometer-encoded or unary implementation has a relaxed matching requirement, is not susceptible to glitches during code transitions and guarantees monotonicity. On the other hand, in a binary-weighted implementation the switching circuitry, digital control circuitry and routing complexity are all substantially reduced. Consequently, a segmented approach promises the best of both worlds.

Following the approach of [26], a MATLAB model has been developed to simulate various segmentation schemes in a 10-bit DAC. The mismatch in the 2fF unit capacitors of the DAC has been modelled based on the model provided in [27]:

$$\frac{\sigma_{\Delta C}}{C} = \frac{0.7\%}{\sqrt{C\ in\ fF}} = 0.5\% \tag{4.6}$$

100 Simulations were run taking a more pessimistic case of 1% random mismatch in the DAC unit capacitors and their consolidated results are presented in Figs. 4.2 - 4.5.

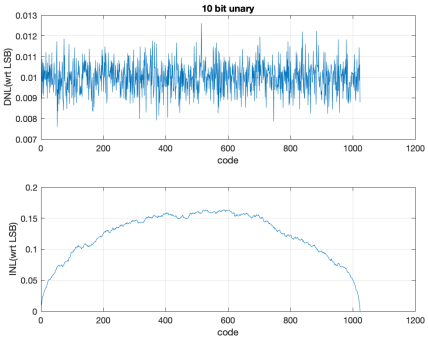

Fig. 4.2. RMS of 100 simulations for 10-bit unary DAC

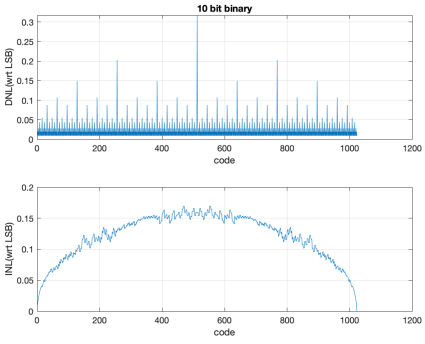

Fig. 4.3. RMS of 100 simulations for 10-bit binary DAC

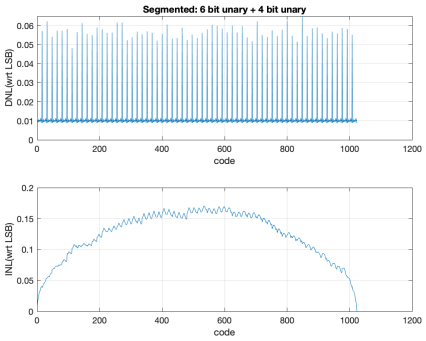

Fig. 4.4. RMS of 100 simulations for 6-bit unary + 4-bit unary DAC

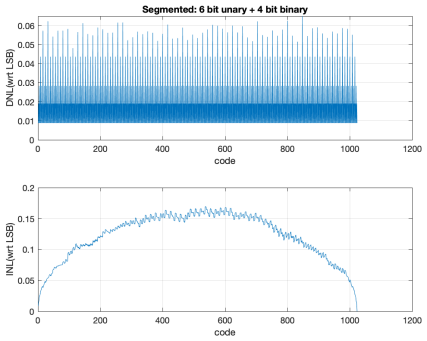

Fig. 4.5. RMS of 100 simulations for 6-bit unary + 4-bit binary DAC

The INL plots for the 10-bit unary and the 10-bit binary DAC exhibit similar characteristics with their peak INL agreeing with the theoretical value of $0.5\sigma\sqrt{1024} = 16\sigma = 0.16LSB$. The peak DNLs for the 10-bit unary and the 10-bit binary also match the theoretical values of σ and 32σ. For the segmentation cases, the INL behaviour remains similar to the previous two, however, the average DNL is lower for the 6 bit unary + 4 bit unary DAC.

Taking into account the above results, a 6-bit unary + 3-bit unary architecture has been chosen for the 9-bit capacitive DAC employed in the DTC. The schematic of the DTC with the segmented DAC is shown in Fig. 4.6.

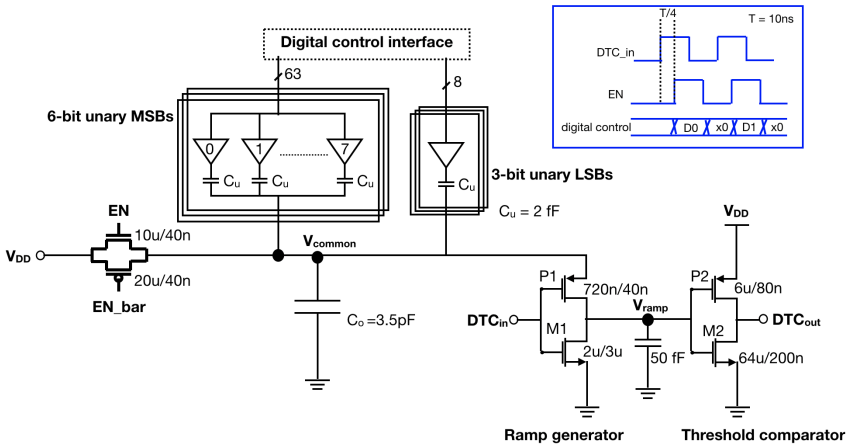

Fig. 4.6. DTC schematic with segmented DAC

The basic unit cell of the DAC is made up of the C_u capacitor a buffer. This makes the LSB. It is interesting to note that the MSB has been made by placing eight identical unit cells in parallel. Such a configuration is expected to yield favorable

device matching.

4.1.4. POST LAYOUT SIMULATIONS

The layout of the DAC unit cell is shown in Fig. 4.7 while Fig. 4.8 shows the placement of dummy cells added in the layout of the complete DAC array.

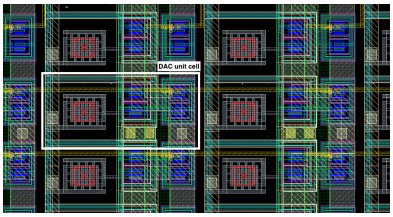

Fig. 4.7. Capacitive DAC unit cell comprising of a MOM cap and a buffer

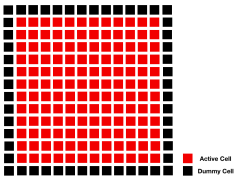

Fig. 4.8. Placement of dummy cells

The complete layout of the DTC is shown in Fig. 4.9.

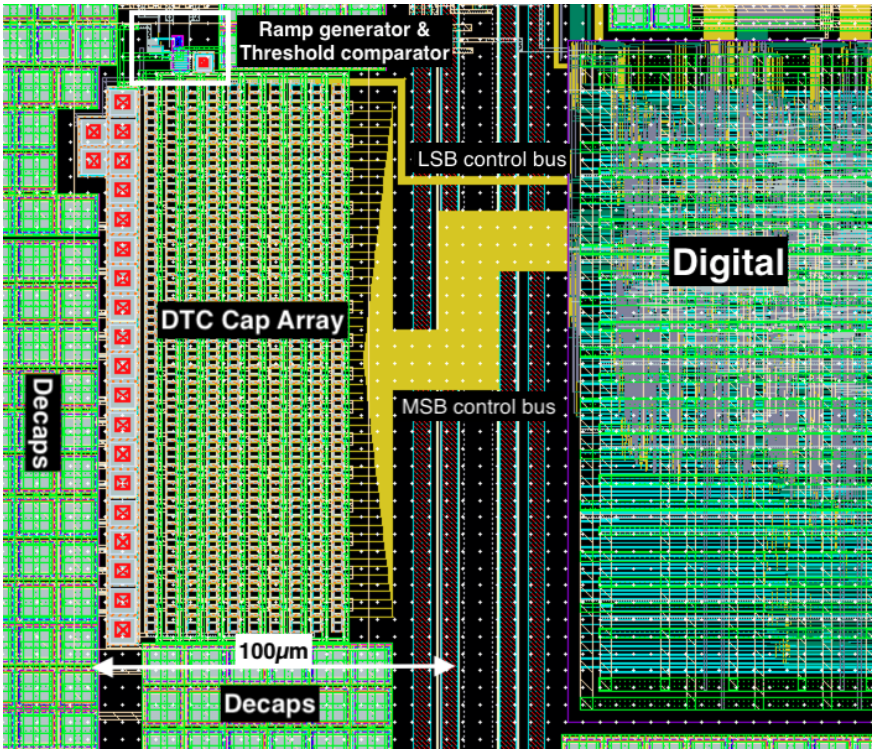

Fig. 4.9. Layout of the DTC in a 40nm process

Fig. 4.10 shows the thermal phase noise profile for the DTC while Fig. 4.11

illustrates the linearity characteristics of the DTC.

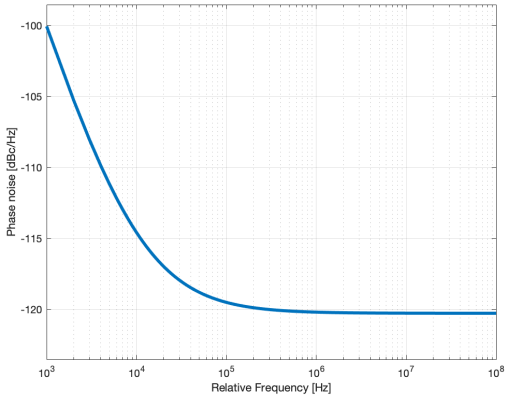

Fig. 4.10. Phase noise profile of the DTC

The thermal phase noise of the DTC is 10dB below the targeted in-band phase noise of the PLL.

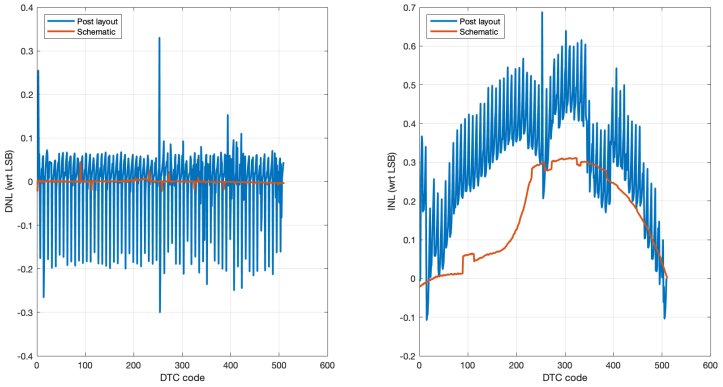

Fig. 4.11. Simulated DNL and INL of the DTC

The segmentation scheme used in the DTC is evident in the INL plot of Fig. 4.11, where a 'zigzag' pattern repeats after every 8 DTC codes.

Power consumed by the DTC is $140\mu W$.

4.2. CHARGE-SAMPLING PHASE DETECTOR

The basic principle of operation of the Charge-sampling phase detector (CSPD) was covered in chapter 1. Fig. 4.12 shows the circuit diagram of the CSPD.

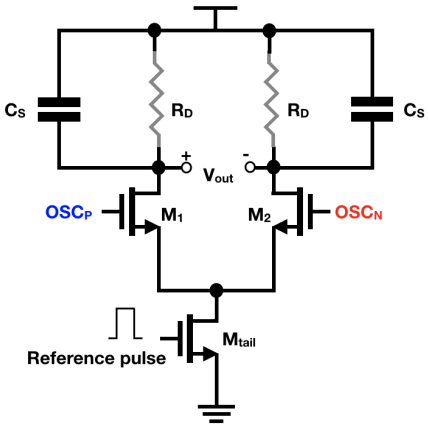

Fig. 4.12. Charge-sampling phase detector

4.2.1. PHASE DETECTOR GAIN

An expression for the phase detector gain has been derived for sinusoidal VCO output in [8] and is given by:

$$K_{PD} = \frac{2G_M A_{VCO} R_D}{N\pi}.sin(0.5\omega_{VCO}T_P).\frac{sin(\phi)}{\phi} \tag{4.7}$$

In equation 4.7, G_M is the transconductance of the MOS devices $M_{1,2}$, A_{VCO} is the amplitude of the VCO output signals $OSC_{P,N}$, N is the ratio between the VCO output frequency and the reference frequency, ω_{VCO} is the angular frequency of oscillations produced by the VCO, T_P is the pulse width of the reference pulse and ϕ is the phase error between the reference and VCO output.

When the reference pulse is high, the windowed current integration phase is engaged and phase comparison occurs. During this phase, a drop is observed in the common mode voltage at the output of the CSPD. This drop is given by:

$$\Delta V_{CM} = \frac{G_M V_{DC} T_P}{C_S} \tag{4.8}$$

While designing the CSPD, the circuit component values were selected by accounting for the following considerations:

1. K_{PD} should be sufficiently large in order to suppress the phase noise of loop components.
2. Loop stability should be maintained and the $R_D C_S$ pole acts as the dominant pole and significantly influences the phase margin.
3. The common mode voltage at the output of the CSPD should not fall below 800mV to ensure that the MOS devices $M_{1,2}$ remain in saturation and the CM level of the next stage is sufficient.

4. The differential input pair must not be sized too small in order to prevent susceptibility to random device mismatch. Additionally, both the flicker and thermal noise can be reduced by increasing the size of input pair MOS devices. However, increasing device size also presents a trade-off in the form of increasing feedthrough of the reference pulse into the oscillator, causing reference spurs.

Accounting for the above mentioned factors, the circuit component values were chosen: $R_D = 50k\Omega$, $C_S = 200fF$, $G_M = 600\mu S$. For $A_{VCO} = 550mV$, $T_P = 100ps$, VCO output frequency = 1GHz, reference frequency = 100MHz (N = 10) and small values of ϕ, such that $sin(\phi)/\phi = 1$, $K_{PD} = 0.5V/rad$ has been achieved.

The output characteristics of the designed CSPD have been simulated with the above stated circuit components and are presented in the plot of Fig. 4.13. The dotted line represents a slope of 0.5 rad/V which matches well with the plot for small values of ϕ.

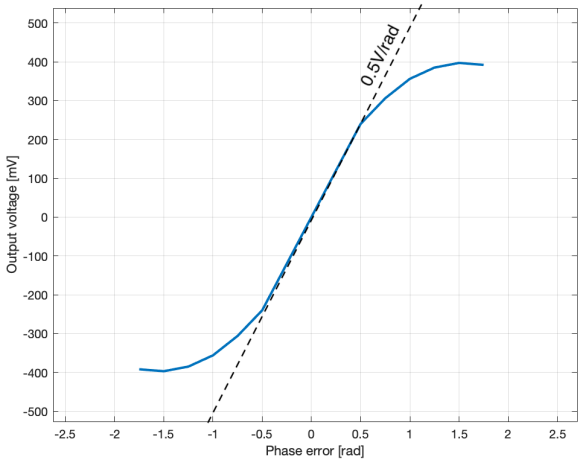

Fig. 4.13. Phase error Vs Output voltage plot for the CSPD

4.2.2. INTRODUCTION OF A RE-SAMPLING PHASE

It can be observed that during phase comparison, the CSPD generates a ripple at its output even when the PLL is locked. The red curve in Fig. 4.14 shows this ripple which was produced with the same frequency as the reference. This ripple, despite being attenuated by the loop filter, would eventually be fed to the VCO control voltage, thereby creating reference spurs.

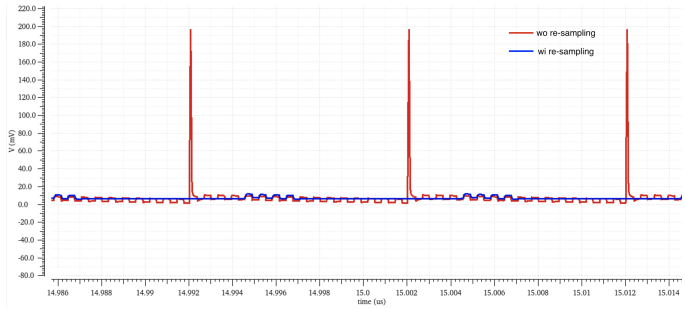

Fig. 4.14. Transient outputs generated by the CSPD with and without the re-sampling circuit for a locked PLL. The small square-shaped bumps are due to the feedthrough of the VCO having a square wave output.

The loop filter can be designed to suppress this ripple to a great extent, however a better solution incorporating re-sampling of the CSPD output has been presented in [28]. In this thesis, a simpler implementation has been conceived by using a re-sampling circuit comprising of capacitors C_O and switches $M_{3,4}$ as shown in Fig. 4.15.

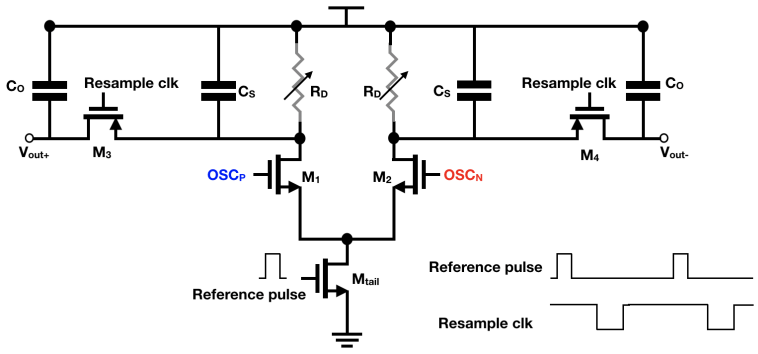

Fig. 4.15. Schematic of the CSPD with the re-sampling circuit

The position of the re-sampling phase has been shown in the timing diagram of Fig. 4.15. Sufficient time gap has been kept between the reference pulse and the re-sample pulse to account for the DTC shifting of the reference pulse during fractional-N operation. The value of the re-sampling capacitors C_O has been kept much smaller than C_S (around $C_S/6$) to prevent significant voltage drop at the CSPD output during charge redistribution in the re-sampling phase. The drop in voltage also reduces the K_{PD}.

The efficacy of the re-sampling circuit can be seen in the transient waveform of Fig. 4.14 and also in the spectra of Fig. 4.16 and Fig. 4.17.

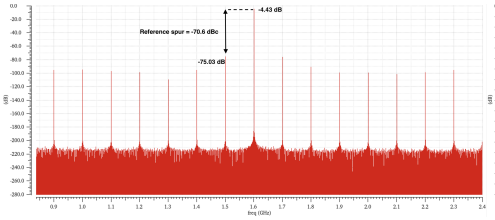

Fig. 4.16. Spectrum of the VCO output without re-sampling

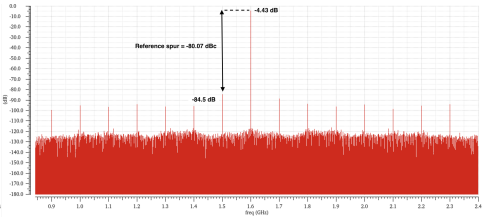

Fig. 4.17. Spectrum of the VCO output with re-sampling introduced in the CSPD

As seen in Fig. 4.16 and Fig. 4.17, the reference spur has been reduced by nearly 10dB after incorporating the re-sampling circuit. In the CSPD, R_D has been kept programmable as it acts as a tuning knob to control K_{PD}, which in turn acts as a tuning knob to control the loop bandwidth.

4.2.3. POST-LAYOUT SIMULATION OF THE CSPD

The layout of the CSPD has been presented in Fig. 4.18

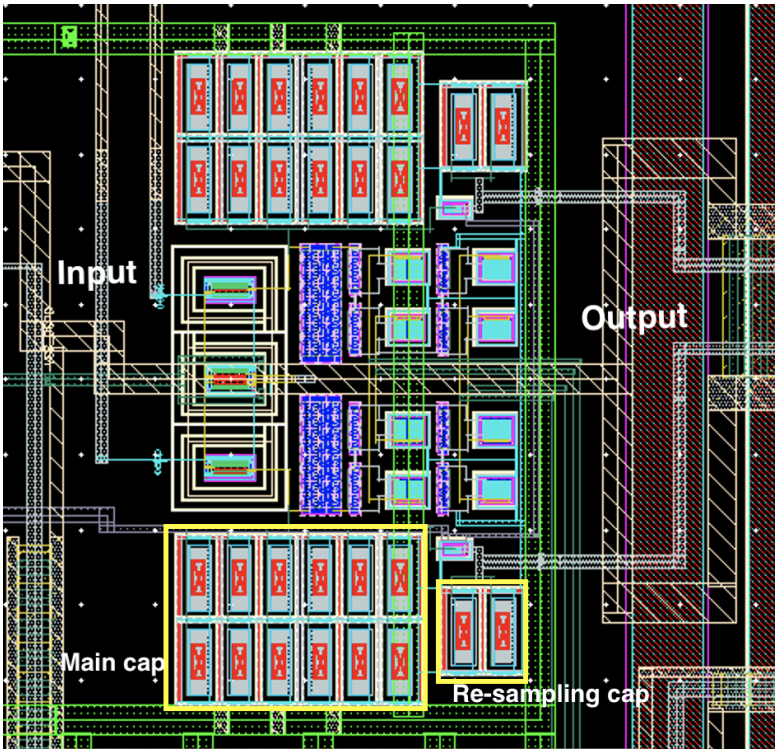

Fig. 4.18. Layout of the CSPD in a 40nm CMOS process

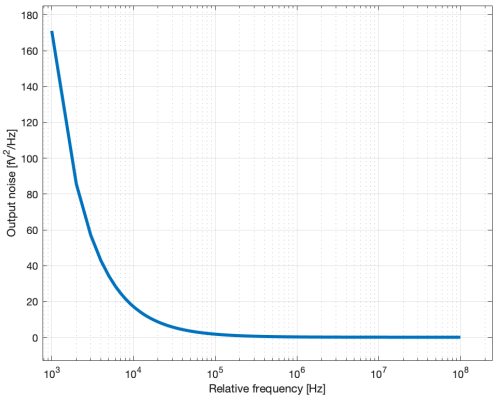

Fig. 4.19. Output noise characteristics of the CSPD

The output noise characteristics of the CSPD have been plotted in Fig. 4.19. The CSPD consumes a power of $12\mu W$.

4.3. OTA

The OTA in the PLL converts the differential voltage output from the CSPD into a single-ended current integrated by the loop filter. A folded cascode structure with differential input and single-ended output has been employed for this purpose. Fig. 4.20 shows the schematic of the OTA along with its biasing network.

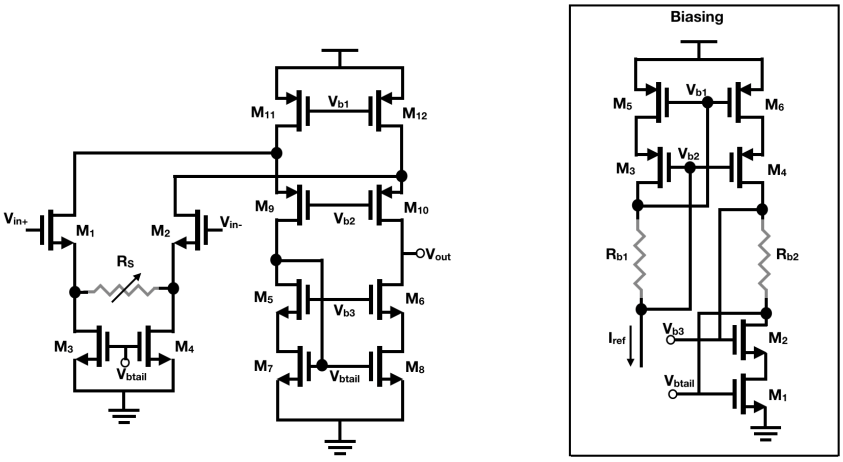

Fig. 4.20. Schematic of the core OTA along with its biasing network

The transconductance of the OTA is given by:

$$G_m = \frac{g_m}{1 + g_m R_S} \tag{4.9}$$

where g_m is the transconductance of the input pair NMOS devices $M_{1,2}$ and R_S is the value of the degeneration resistance. To enable tuning of the loop bandwidth, R_S has been made programmable by dividing it into 10 sections comprising of both coarse and fine steps. Fig. 4.21 shows the tunability that can be achieved in the G_m of the OTA. The fine tuning steps can be seen in the center (tuning range setting 4-8) while the remaing sections in the plot show the coarse steps.

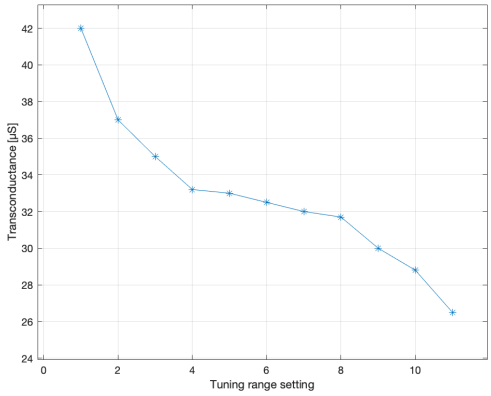

Fig. 4.21. Transconductance of the OTA for different values of the degeneration resistance

The layout of the OTA has been presented in Fig. 4.22

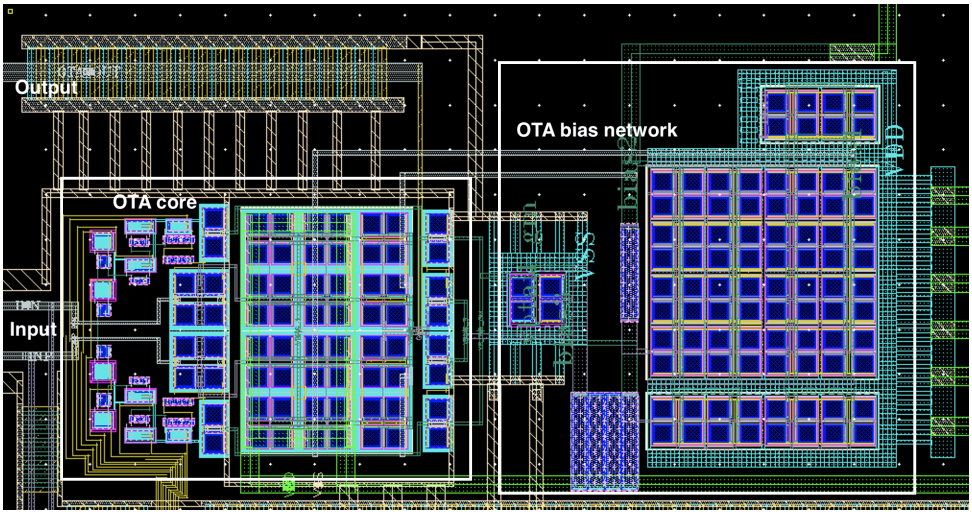

Fig. 4.22. Layout of the OTA in a 40nm process

The OTA consumes a power of $115\mu W$.

4.4. VOLTAGE CONTROLLED OSCILLATOR

Owing to its compactness and wide frequency tuning range, a ring oscillator (RO) has been chosen to serve as the oscillator for the PLL.

4.4.1. SELECTING RING OSCILLATOR TYPE

The most simple RO topology is the single-ended (SE) RO comprising of a N-stage inverter ring as shown in Fig. 4.23. The number of stgaes, N must be odd.

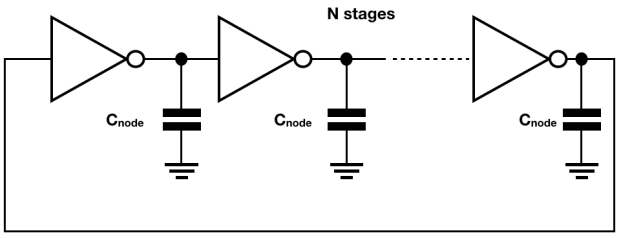

Fig. 4.23. Single-ended RO topology

An approximate expression for the free-running frequency of the SE RO is give by:

$$f_{osc} \approx \frac{\mu_{eff} W_{eff} C_{ox} (V_{DD}/2 - V_T)^2}{8\eta N L V_{DD} C_{node}} \tag{4.10}$$

where μ_{eff} is the effective mobility of electron and hole, W_{eff} is the sum of the widths of the PMOS and NMOS devices in an inverter stage, L is the device length, C_{ox} is the gate-oxide capacitance per unit area, η is a constant close to unity, C_{node} is the capacitance seen at each node of the ring and V_T is the threshold voltage of the MOS devices. The main drawbacks of the SE RO are its susceptibility to supply pushing as well as capacitive coupling of interference from other PLL blocks.

A fully differential (FD) RO topology, shown in Fig. 4.24, is much less susceptible to supply pushing (see K_{VDD} values in table 4.1) and the common-mode interference from different blocks is also cancelled owing to the inherent differential nature of the ring. Its free-running frequency is given by:

$$f_{osc} = \frac{1}{2NT_D} \tag{4.11}$$

where T_D is delay of each stage and is determined by the load resistor and capacitor. It can be observed from equation 4.11 that the oscillation frequency of the FD RO is independent of the supply voltage.

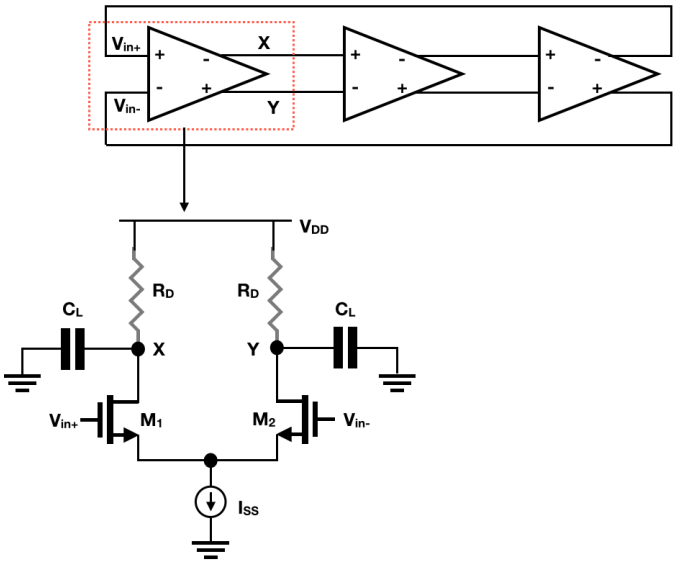

Fig. 4.24. Fully-differential RO topology

A third RO topology is the pseudo-differential RO, as shown in Fig. 4.25.

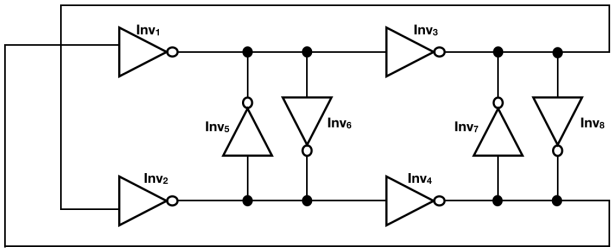

Fig. 4.25. Pseudo-differential RO topology

The inverters $inv_1 - inv_4$ form the main inverters of the ring, however since the number of inverters is even, the ring suffers from latch-up. Auxiliary inverters $inv_5 - inv_8$ are included to prevent latch-up. The strength of the auxiliary inverters with respect to the main inverters must be carefully chosen. If the auxiliary inverters are weak, the latch-up issue will not be resolved and if they are too strong, latch-up problem might persist in the case of device mismatch. Additionally, the auxiliary inverters also account for power consumption and must not be made excessively large. In the following simulations in this section, the strength of the auxiliary inverters has been chosen to be 0.7 times that of the main ones.

It must be noted that the pseudo-differential RO remains susceptible to supply pushing while the common-mode interference is cancelled.

A comparative study for different RO types has been compiled in [29]. Results presented in this study have been corroborated by designing and simulating the different RO types having a free-running frequency of 800 MHz.

Phase noise simulation results have been presented in Fig. 4.26 and 4.27.

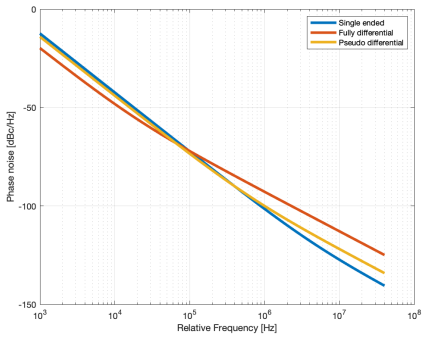

Fig. 4.26. Phase noise profile of different RO topologies

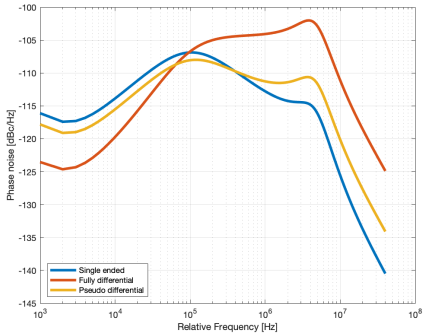

Fig. 4.27. Phase noise profile of different RO topologies at the PLL output after being shaped by the loop transfer function

For a PLL designed to have a loop bandwidth of 4 MHz, it can be observed that SE and pseudo-differential types fair better that the FD RO for the out of band noise performance but not for the in-band noise performance. This is because flicker noise up-conversion does not take place in the FD RO if its input pair transistors remain in triode region whereas in the inverter based rings, the flicker noise is also directly up-converted. On the other hand, the thermal noise regime of the FD RO experiences a degradation due to its limited output voltage swing.

Since the out of band phase noise of the oscillator is more important in a PLL circuit, a pseudo-differential RO has been chosen. It promises cancellation of common-mode interference and shows less supply sensitivity (see table 4.1 for K_{VDD} values) when compared to the SE RO.

Table 4.1 quantitatively summarizes the performance figures for each RO type.

Table 4.1: Summary of performance metrics for different RO types

RO type	**jitter(rms,ps) @800 MHz** (offset 1KHz to 40MHz)	**Power(μW)**	**FoM**	**K_{VDD} (GHz/V)**
Single-ended	1.56	481	163.2	1.47
Fully-differential	5.65	2035	148	0.14
Pseudo-differential	2.2	810	159	1.2

4.4.2. MAKING THE OSCILLATOR TUNABLE

In order to achieve tunability of the RO, the current starvation approach is followed where the frequency of oscillations is controlled by controlling the supply current of the inverters. In this case, a current-starved RO is implemented by inserting several PMOS devices acting as current sources between the actual voltage supply node and the V_{DD} node of the inverters. This circuit is presented in Fig. 4.28 and it serves as the tuning bank for the oscillator.

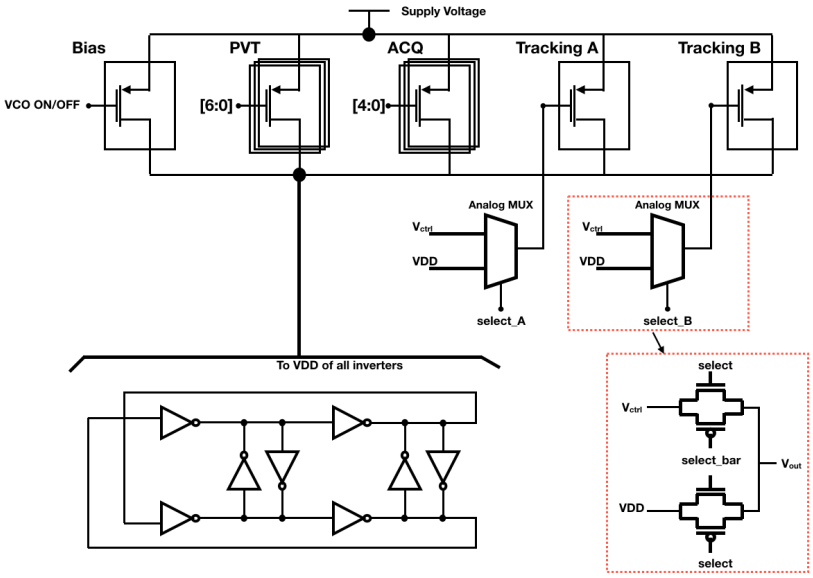

Fig. 4.28. Schematic of the current-starved VCO

The PMOS current sources have been grouped into different banks. There is a Bias PMOS devices responsible for turning on the VCO. The PVT and acquisition (ACQ) banks collectively provide a frequency tuning range of 1-2 GHz with a 12 bit resolution. There is also a tracking bank where the control voltage from the loop filter output is fed. Tracking A PMOS device provides a K_{VCO} of 20 MHz/V at 1 GHz while it dips to 4 MHz/V at 2 GHz. This dip in K_{VCO} is seen due to a decrease in the relative strength of the tracking A PMOS as other current sources in the bank are switched ON. To maintain the loop bandwidth and the loop stability, a second tracking B PMOS has been employed. The tracking B PMOS is sized larger and provides a K_{VCO} of more than 15 MHz/V at 2 GHz. It is also possible to enable both the tracking PMOS devices together, in which case their K_{VCO} values will add up and increase the loop bandwidth. However, both tracking devices must not be enabled at lower VCO frequencies (around 1 GHz) since the increase in loop bandwidth is substantial at these frequencies and could compromise the loop stability.

[1] FoM = -(PN@Δf) + $20log_{10}(f_o/\Delta f)$ - $10log_{10}(Power_{DC}/1mW)$

4.4.3. POST-LAYOUT SIMULATIONS

Fig. 4.29 shows the complete layout of the VCO.

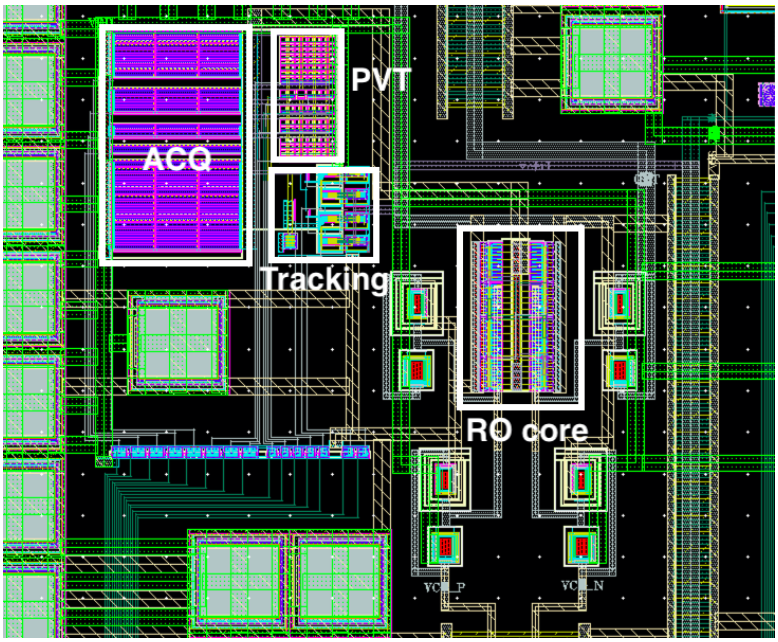

Fig. 4.29. Layout of the VCO in a 40 nm process

The post-layout phase noise profile for the RO oscillating at 2 GHz has been plotted in Fig. 4.30.

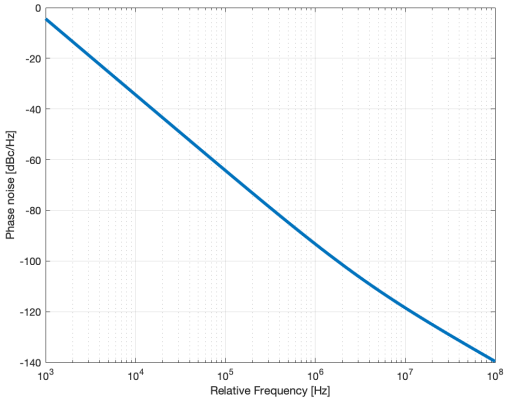

Fig. 4.30. Phase noise profile of the RO oscillating at 2 GHz

The oscillator consumes a power of $600\mu W$.

4.5. COMPARATOR

The comparator plays an essential role in the DTC gain calibration procedure by detecting the polarity of phase error at the output of the CSPD and feeding it to the digital calibration circuitry. For this purpose, a StrongARM latch has been used to serve as the comparator (Fig. 4.31). High operating speed and zero static power consumption are the main merits of this comparator architecture. Additionally, its input-referred offset can be primarily attributed to the input pair, making it a suitable choice for incorporating foreground calibration schemes. The operating principle of the StrongARM latch has been covered extensively in [30] and a brief overview has been presented in the following section.

4.5.1. OPERATING PRINCIPLE OF THE STRONGARM LATCH

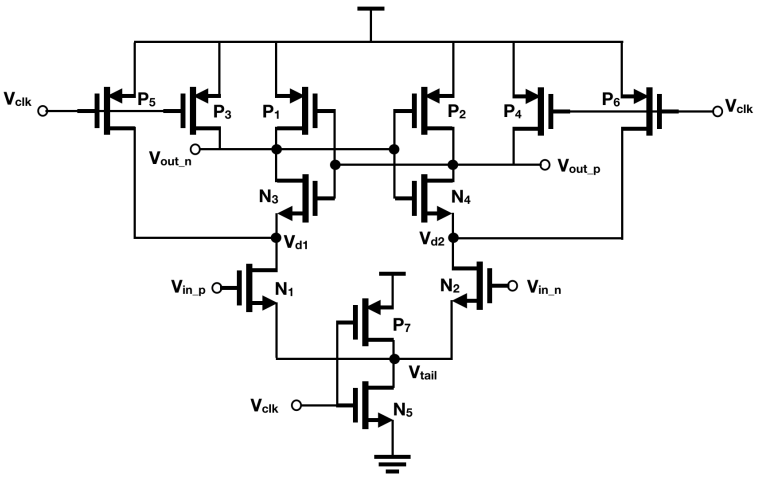

Fig. 4.31. Schematic of the StrongARM latch based comparator

The comparator in Fig 4.31 comprises of a differential input pair $N_{1,2}$, cross coupled inverters $N_3 - P_1$ and $N_4 - P_2$ and switches $N_5, P_3 \sim P_7$. The operation of the StrongARM latch can be explained in the following three phases:

1. Reset phase: In the reset phase, the clock signal V_{clk} is low and the nodes $V_{d1,d2}$, V_{out_p,out_n} and V_{tail} are precharged to V_{DD} through switches $P_3 \sim P_7$.

2. Amplification phase: When the clock signal goes high, the input pair NMOS devices turn on and draw a differential current proportional to $V_{in\_p} - V_{in\_n}$. This results in the development of an amplified differential voltage $V_{d1} - V_{d2}$ at the drain nodes of the input pair which increases with time. The amplification process can be described using the equation:

$$|V_{d1} - V_{d2}| = \frac{g_{m1,2}|V_{in\_p} - V_{in\_n}|t}{C_{d1,d2}} \tag{4.12}$$

where $g_{m1,2}$ is the transconductance of the input pair NMOS devices and $C_{d1,d2}$ is the capacitance at the drain nodes of the input pair. A common mode current I_{CM} is also drawn by the input pair which leads to a drop in the voltages V_{d1} and V_{d2}. The amplification phase lasts till V_{d1} and V_{d2} remain above $V_{DD} - V_{th\_N}$, V_{th_N} being the threshold voltage of the NMOS devices $N_{3,4}$. The amplification time can thus be approximately estimated by $t_{amp} = (C_{d1,d2}/I_{CM})V_{th\_N}$. This can be substituted in equation 4.12 to get the amplification gain:

$$A_v \approx \frac{g_{m1,2}V_{TH\_N}}{I_{CM}} \tag{4.13}$$

3. Regeneration phase: The regeneration phase commences as V_{d1} and V_{d2} fall below $V_{DD} - V_{th\_N}$, turning on the NMOS cross-coupled devices $N_{3,4}$ and thus leading to a drop in voltage level at the nodes V_{out_p,out_n}. Eventually, V_{out_p,out_n} drop to a value $V_{DD} - |V_{TH\_P}|$ thereby turning on the PMOS cros-coupled pair $P_{3,4}$. The positive feedback of the inverters then pulls one output to zero while the other remains at V_{DD}.

The comparator consumes dynamic power due to the charging and discharging of capacitances at nodes $V_{d1,d2}$ and V_{out_p,out_n} and is given by $f_{CK}(2C_{d1,d2} + C_{out\_p,out\_n})V_{DD}^2$, assuming that $C_{d1,d2}$ discharge nearly to ground while C_{out_p,out_n} discharge to $V_{DD}/2$. The clock frequency for the comparator is same as the reference clock frequency of 100 MHz.

Transient simulation results showing the comparator settling behavior for differential input voltage of $10\mu V$ are presented in Fig 4.32. It can bee seen that the comparator speed is better than 1ns.

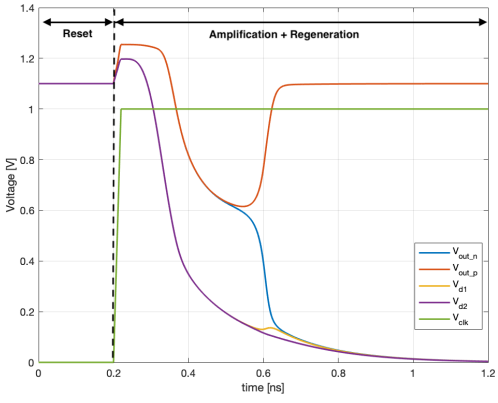

Fig. 4.32. Transient simulation of the designed comparator

Table 4.2: Comparator device sizing

Device	Sizing
N_1/N_2	8u/800n
N_3/N_4	1u/120n
P_1/P_2	2u/120n
N_5	1u/120n
$P_3 \sim P_7$	1u/120n

Table 4.2 shows the device sizing for the designed comparator. The input pair has been designed large enough so as to minimize input-referred offset due to random device mismatch. The switches have been sized to support the speed requirements dictated by the 100 MHz comparator clock. Subsequent sections discuss the noise and offset seen in the designed comparator.

4.5.2. COMPARATOR NOISE

The primary contributors of input-referred noise in the StrongARM latch are the input pair NMOS devices. Some share of the noise can also be attributed to the $\frac{kT}{C}$ noise injected due to the switching action of P_5 and P_6. The noise from other circuit components is small since they come into play much later in the decision making process of the comparator.

It is not straightforward to determine the noise of a comparator. This section describes a technique used to estimate the input-referred comparator noise. With a fixed differential voltage applied at the input of the comparator, several transient simulations are run with transient noise enabled. Due to noise, the comparator will not make the same decision every time, despite the input voltage being fixed. This effect can be quantified as the bit error rate (BER). The differential input is swept from $100\mu V$ to $1.5mV$, and a BER profile is plotted. As can be predicted, the BER must drop as the differential input is increased. This BER plot can be approximated as a Gaussian probability distribution [30]. The differential voltage corresponding to the 16% BER point forms the σ of the noise.

Fig. 4.33 shows the BER profile for the designed comparator. 100 transient simulations were run for each differential input voltage applied to the RC extracted version of the comparator.

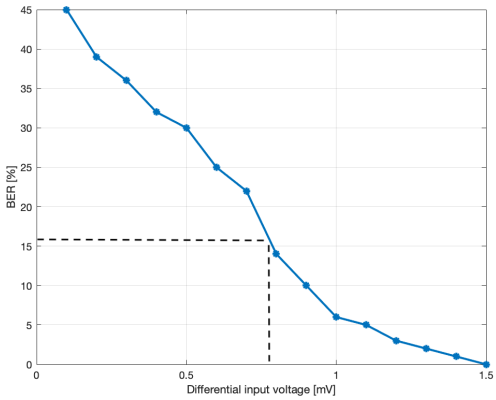

Fig. 4.33. Comparator BER vs applied differential input voltage

From the plot of Fig. 4.33, it can be seen that the input-referred comparator noise is about $800\mu V_{rms}$. Assuming that this noise gets integrated over a bandwidth of 100 MHz, which is the reference frequency of the sub-sampling PLL, the noise PSD can be obtained as $6.4fV^2/Hz$. This voltage noise PSD when referred to the PLL output shall be shaped by the transfer function $|H_{cl}(s)|^2\left(\frac{1+sC_SR_D}{K_{PD}}\right)$, which is same as that for the CSPD output noise (equation 3.10). It can be observed from the full chip phase noise simulation plot of chapter 6 that this noise level is acceptable as it does not limit the in-band phase noise of the PLL.

4.5.3. COMPARATOR OFFSET

Random device mismatch in the input pair NMOS devices acts as the primary contributors to the comparator input-referred offset. The MOS devices forming the cross-coupled inverters turn on much later in the comparator decision making process and hence their contribution is small.

In order to determine the input-referred offset in the designed comparator, Monte Carlo Sampling has been performed with the simulation setup shown in Fig. 4.34.

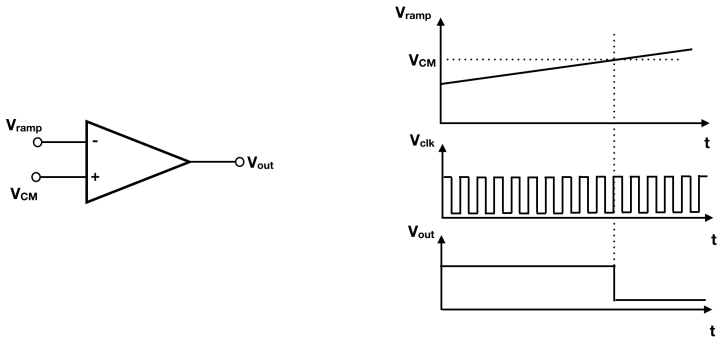

Fig. 4.34. Simulation setup to determine comparator input-referred offset

One of the comparator inputs is tied to the input common mode voltage V_{CM}, while a slowly increasing ramp voltage V_{ramp} is applied to the other input terminal. As V_{ramp} approaches V_{CM}, the comparator output toggles. In an ideal comparator, the output would toggle when V_{ramp} is exactly equal to V_{CM}. However, during Monte Carlo Sampling, device mismatch would induce deviations in the value of V_{ramp} for which the comparator output toggles. These deviations reflect the offset of the comparator. During the Monte Carlo Sampling, V_{CM} value is set to $800mV$, which is same as the common-mode level fed to the comparator input when it is plugged-in at the output of the CSPD (Fig. 3.4).

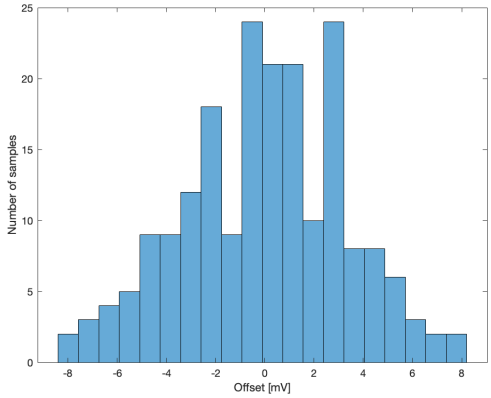

Fig. 4.35. Comparator input-referred offset (200 Monte Carlo samples)

From the histogram plot of Fig. 4.35, it can be observed that the comparator is expected to show a worst case offset of $10mV$.

4.5.4. FOREGROUND OFFSET CALIBRATION SCHEME

The concept of effective offset $V_{OS,EFF}$, which is the cumulative offset of the comparator and OTA experienced by the DTC gain calibration loop, was explained in section 3.2. Since the OTA input pair is sized larger than the comparator input pair, the input-referred OTA offset is not expected to be more than the input-referred comparator offset, which is $10mV$. In the worst case both OTA and comparator offsets can add up and hence $V_{OS,EFF}$ shall become as large as $20mV$.

As has been shown in section 3.2, the DTC gain calibration loop can not tolerate an effective offset of more than $100\mu V$. Therefore, there is a need to implement a foreground offset calibration mechanism having a range of $\pm 20mV$ and a resolution of $100\mu V$.

An offset cancellation method is presented in [30] which involves creating asymmetry in the capacitances C_{d1} and C_{d2} at the drain nodes of the input pair NMOS devices to cancel out the random device mismatch in the input pair. Considering asymmetry in C_{d1} and C_{d2}, the built-in offset in the comparator can be estimated as [30]:

$$\Delta V = \left(\frac{C_{d1}}{C_{d2}} - \frac{C_{d2}}{C_{d1}}\right).\frac{V_{TH\_N}}{2} \tag{4.14}$$

The schematic of Fig. 4.36 illustrates a technique used to create asymmetry in the capacitances seen at nodes V_{d1} and V_{d2}. PMOS devices P_8 and P_9 have been incorporated to behave as voltage controlled capacitors (or varactors) as their gate voltages are varied by resistive digital to analog converters (RDACs). A similar approach can be found in literature [31], where the asymmetry in capacitance is created at the output nodes of the comparator V_{out_p} and V_{out_n}. However, in this work the asymmetry has been created at the drain nodes of the input pair to make its effect more profound in the decision making process of the comparator. A wider offset cancellation range can thus be provided for a given size of P_8 and P_9.

The foreground calibration procedure begins with clocking the comparator for a zero differential input voltage and feeding the comparator decision to the digital control block. Based on the polarity of the comparator output, the digital control block selects one of the RDACs and tunes the varactor until the comparator decision toggles. It is important to note that the targeted offset resolution to be calibrated is about $100\mu V$, while the input-referred noise level of the comparator is $800\mu V_{rms}$. It is thus necessary to employ an approach where the comparator decisions sampled by the digital block are fed to an accumulator in order to gradually retrieve the offset information buried beneath the comparator noise. Details of the digital control block used in the foreground procedure are presented in chapter 5.

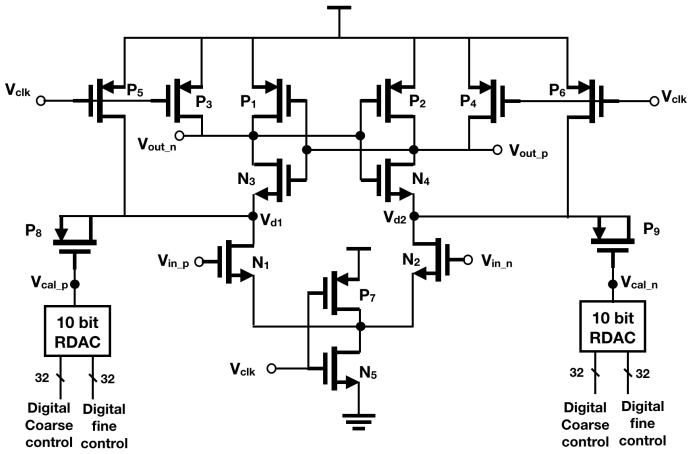

Fig. 4.36. Schematic of the comparator with offset calibration mechanism

For a PMOS device sized as $1u/1u$, the variation in the gate to source capacitance for gate volatge swept between $V_{DD} = 1.1V$ and zero has been plotted in Fig. 4.37. Both drain and source of the PMOS device have been tied to V_{DD}.

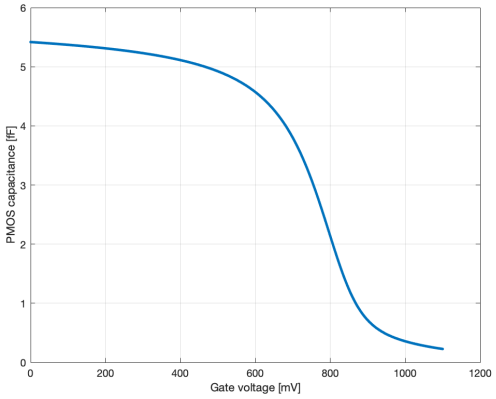

Fig. 4.37. PMOS capacitance vs gate voltage for device W/L = 1u/1u

For a 10 bit resolution in the RDAC, a step size of $1aF$ can be produced in the varactor for either high or low values of gate voltage. For the moderate gate voltages, the capacitance variation is quite steep and a step size of about $20aF$ is produced. These step sizes can be substituted in equation 4.14 along with the node capacitance values $C_{d1,d2} = 40fF$ (including the RC extracted parasitics). For $V_{TH\_N} = 400mV$, equation 4.14 yields best case and worst case offset resolution of $10\mu V$ and $200\mu V$ respectively. The RDACs can be appropriately programmed by

the digital block such that the varactors are swept in regions of finer step sizes during foreground calibration.

4.5.5. RDAC DESIGN

Monotonicity in the DAC is essential to ensure success of the foreground calibration procedure. In addition to being monotonic, the DAC must also have a high resolution. The implementation of a high resolution DAC (10 bit DAC in this case) can be facilitated by choosing a segmented architecture so as to reduce the number of switches and control circuitry. A review of segemented DACs is presented in [32]. A segmented RDAC architecture possessing intrinsic monotonicity [33] has been implemented. Since the implementation comprises of unary resistive ladders, it also promises good linearity [27]. The main concept of this architecture is illustrated with the help of a 6 bit RDAC shown in Fig. 4.38.

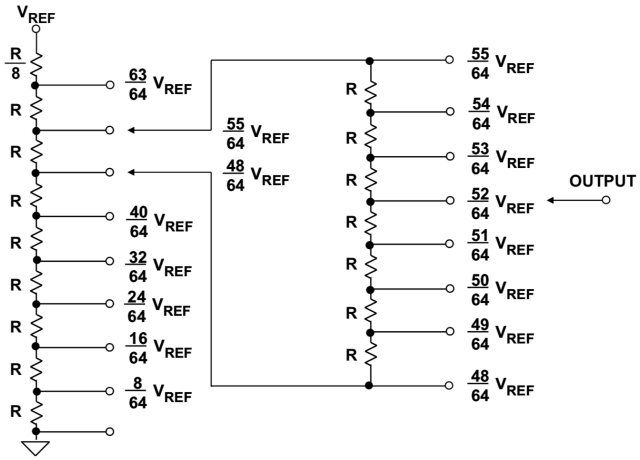

Fig. 4.38. An intrinsically monotonic segmented RDAC [33]

The RDAC is constructed using a 3 bit coarse unary ladder (MSB ladder) followed by a 3 bit fine unary ladder (LSB ladder). The coarse ladder contains 2^K MSB resistors while the fine ladder contains $2^K - 1$ LSB resistors. All resistors in Fig. 4.38 are of equal value except the topmost resistor in the coarse ladder, which has a value of $R/2^K$, with K being equal to 3 in this implementation. Since there are no buffers in this implementation, the LSB ladder when connected parallel to one of the MSB resistors tends to load it and causes a drop in voltage equal to one LSB, accross the chosen MSB resistor. The $R/2^K$ resistor has been added in the coarse ladder to compensate for the one LSB voltage drop. The desired output voltage can be obtained by appropriately selecting the voltage taps in the two ladders.

Schematic of the 10 bit RDAC designed as part of this work is presented in Fig. 4.39.

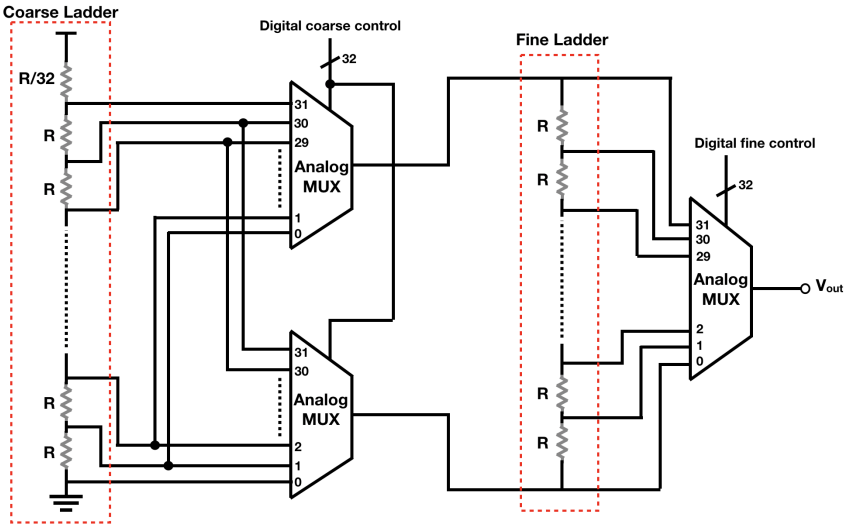

Fig. 4.39. 10 bit segmented RDAC

The coarse and fine ladders are made up of (32 + 1) and 31 resistors respectively. Control and switching circuitry has been designed using analog MUXes. The analog MUX itself has been designed using transmission gates, as shown in the schematic of Fig. 4.40. In order to control the MUXes, 'one-hot' encoding has been used, hence 32 select lines have been provided to each of the MUXes.

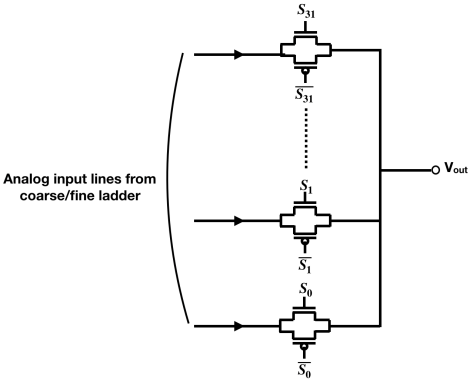

Fig. 4.40. Schematic of the analog MUX used

The value of R is chosen as $2k\Omega$, while the resistance of the transmission gates in their ON state is about 100Ω.

4.5.6. FOREGROUND CALIBRATION SIMULATION RESULTS

To simulate the foreground calibration procedure, an external positive offset voltage resembling $V_{OS,EFF}$ was applied to the positive terminal V_{in_p} of the comparator shown in the schematic of Fig. 4.36. As per expectation, the RDAC attached to the circuit node V_{cal_p} shall be selected by the digital control block and the corresponding varactor connected to this node shall be tuned to achieve offset cancellation. In order to co-simulate the digital control RTL with the analog comparator and RDACs, a mixed-signal simulation was performed with transient noise enabled.

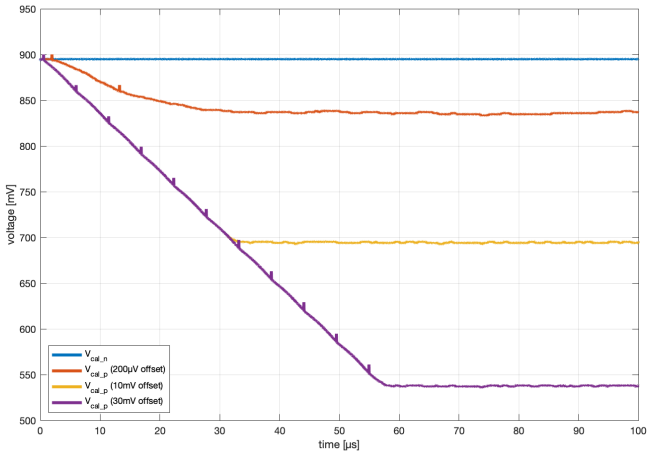

Fig. 4.41. Foreground offset calibration in comparator

The plot of Fig. 4.41 shows how V_{cal_p} is modulated by the digital logic in order to cancel the externally applied offset voltage. The voltage V_{cal_n} remains unchanged since the offset polarity was positive with respect to the positive input terminal of the comparator.

4.5.7. COMPARATOR AND RDAC LAYOUT

The layout of the comparator and RDAC are shown in Fig. 4.42 and Fig. 4.43 respectively.

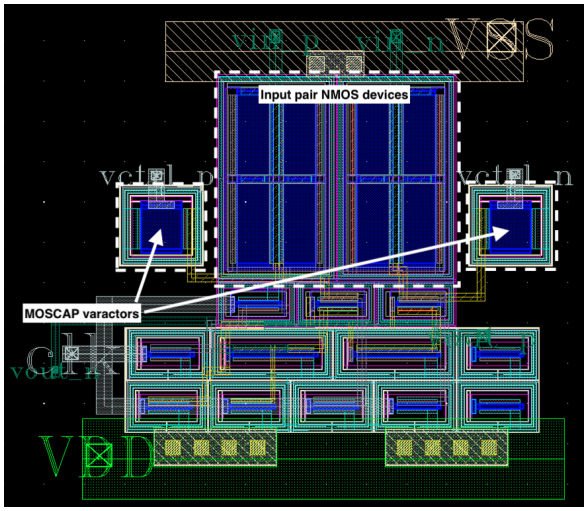

Fig. 4.42. Layout of the StrongARM latch comparator in a 40nm CMOS process

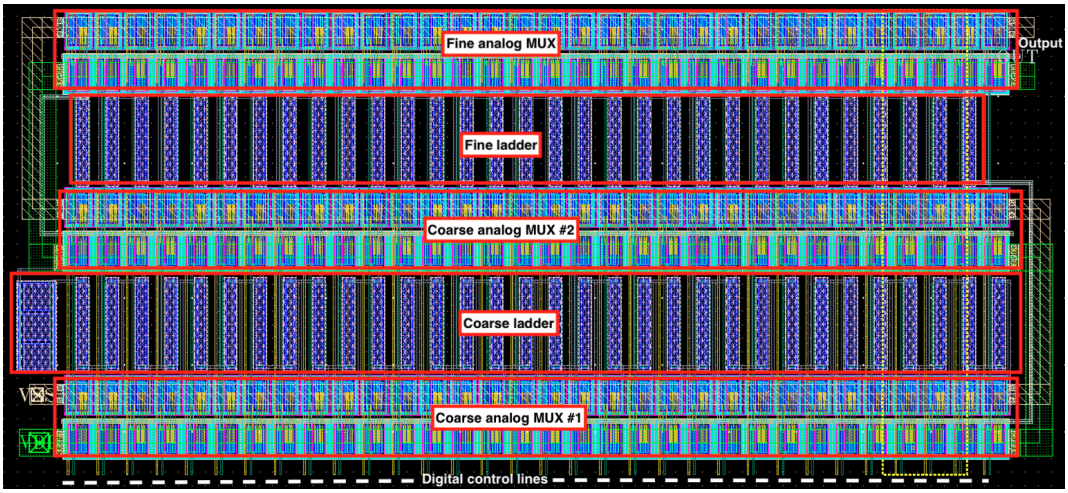

Fig. 4.43. Layout of the RDAC in a 40nm CMOS process

4.6. CLOCK GENERATION BLOCK

A complex network of clocks is required to support proper functioning of the entire PLL chip. Fig. 4.44 shows the various clocks required by different system blocks, while Fig. 4.45 illustrates the waveforms showing phase relationships among the different clocks.

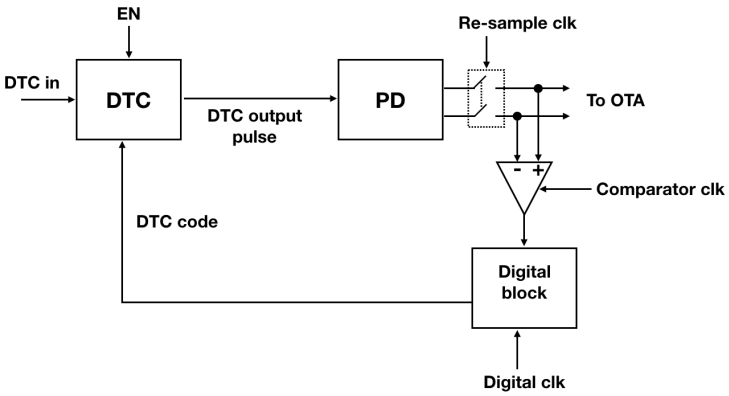

Fig. 4.44. Block diagram showing various clocks required by the system

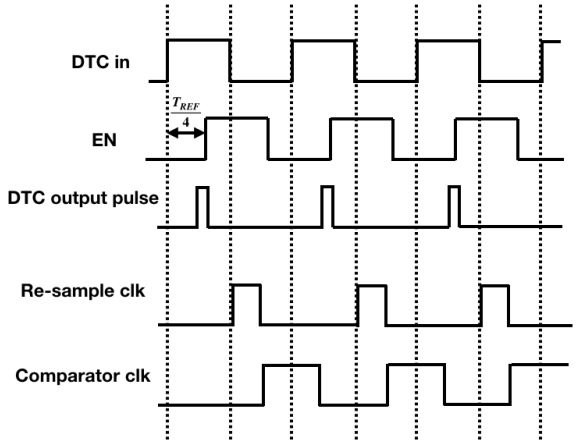

Fig. 4.45. Waveforms of the system clocks

The $DTC\ in$ clock is also fed as the digital clock. The digital logic utilizes a positive edge triggered flip-flop to sample the comparator decision. In Fig. 4.45, the re-sampling clock phase is the CSPD resampling phase and is shown as an active high signal for simplicity. It is inverted to drive the PMOS switches involved in the re-sampling of the CSPD output. The DTC operation has been simulated over PVT corners for all DTC codes to ensure that the DTC output pulse is always available in the positive half-cycle of the $DTC\ in$ signal. This guarantees that the CSPD output is correctly re-sampled. The comparator is clocked right after the re-sampled CSPD output is available.

The hardware implementation of the clock generation block is presented in Fig.

4.46.

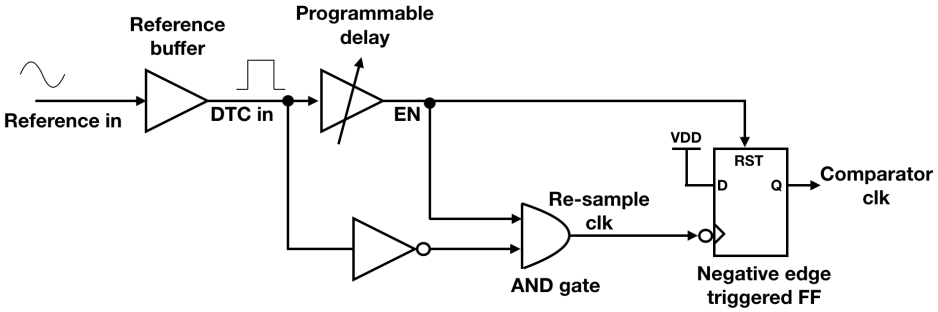

Fig. 4.46. Hardware implementation of the system clocks

5

RTL DESIGN

This chapter discusses the design of digital RTL blocks which are required to support key functionalities of the PLL chip. Fig. 5.1 shows the top level block diagram of the digital logic blocks and their interfaces.

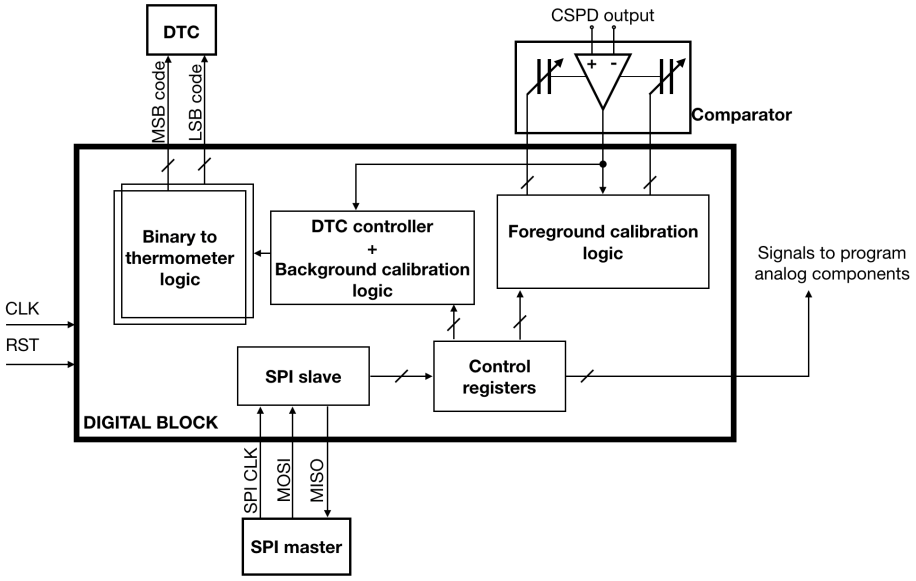

Fig. 5.1. Top level block diagram of the digital logic

The RTL implementation comprises of three primary blocks:

1. Foreground calibration block: This block contains the logic required to calibrate the offset seen by the comparator before it can be used for background calibration.

2. DTC controller: This block generates DTC codes required to produce fractional frequency tones based on the programmed FCW and K_{DTC}. The background calibration logic is also embedded within this block. Since the DTC is designed using unary segmentation, a binary to thermometer encoder has been employed to interface the DTC controller with the DTC.

3. Serial Peripheral Interface (SPI): The SPI slave block has been designed to interface with an off-chip master that programs the chip. This interface can also be used to read out information from the status and debug registers within the chip. Communication is achieved using a protocol involving only three signals: a) a clock (SPI clk), b) MOSI (Master-out Slave-in) and c) MISO (Master-in Slave-in)

The entire digital block (except the SPI slave) works on a 100 MHz clock signal, CLK. The RST signal is an active low asynchronous reset. The SPI clock frequency used is typically of the order of 1 MHz.

Subsequent sections provide details pertaining to the implementation of the foreground calibration logic and the DTC controller block.

5.1. FOREGROUND CALIBRATION

The circuit blocks and the interfaces involved in foreground calibration are illustrated in Fig. 5.2. The implementation details of the comparator, varactor and the RDAC have been covered in section 4.5. This section focuses on digital control logic used during foreground calibration.

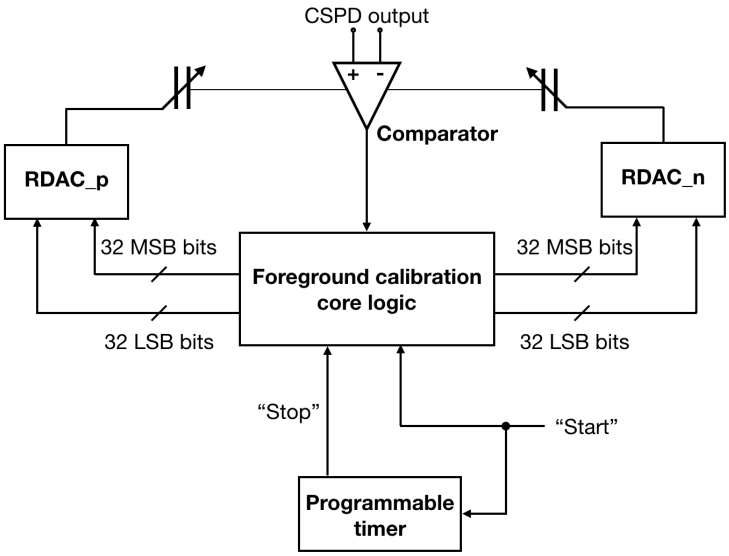

Fig. 5.2. Top level block diagram of the digital logic

The core logic used during foreground calibration (see Fig. 5.3) comprises of a digital accumulator which has been implemented in the form of a signed counter. Since the targeted comparator offset compensation resolution is smaller than its input-referred noise, it is important to accumulate the comparator decisions in order to faithfully retrieve the offset information. The polarity of the comparator decision dictates whether the counter should be incremented or decremented. The counter value is compared with a pre-programmed threshold value, N using a digital threshold comparator. The threshold comparator has been designed to detect threshold values of $+N$ (for positive comparator offset) and $-N$ (for negative comparator offset). Based on the counter full condition detected by the threshold comparator, the RDAC control logic is triggered to first select the appropriate RDAC and then alter its code one LSB at a time for every subsequent counter full detection. The counter is also cleared after every counter full condition is detected.

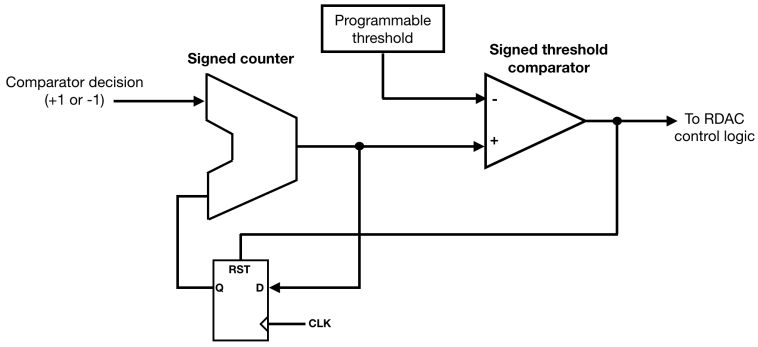

Fig. 5.3. Foreground calibration core logic

It is critical to program a suitable value for the threshold N. A very small value could lead to premature decision making leading to erroneous calibration. In the simulation results shown in Fig. 4.41, a threshold value of $N = 40$ has been chosen. The foreground calibration procedure must be commenced after integer-N locking has been achieved by the PLL. An externally triggered "Start" signal (Fig. 5.2) is used for this. Once the comparator offset has been calibrated, the foreground calibration logic must turned off, with the updated RDAC codes preserved for the remaining PLL operation. This ensures that a properly calibrated comparator is used during the background calibration procesure. A programmable timer (see Fig. 5.2) has been included to stop the foreground calibration logic. The plots in Fig. 4.41 reveal that $100\mu s$ is sufficient time to complete the foreground calibration procedure.

The algorithm used during the foreground calibration procedure is presented in Fig. 5.4.

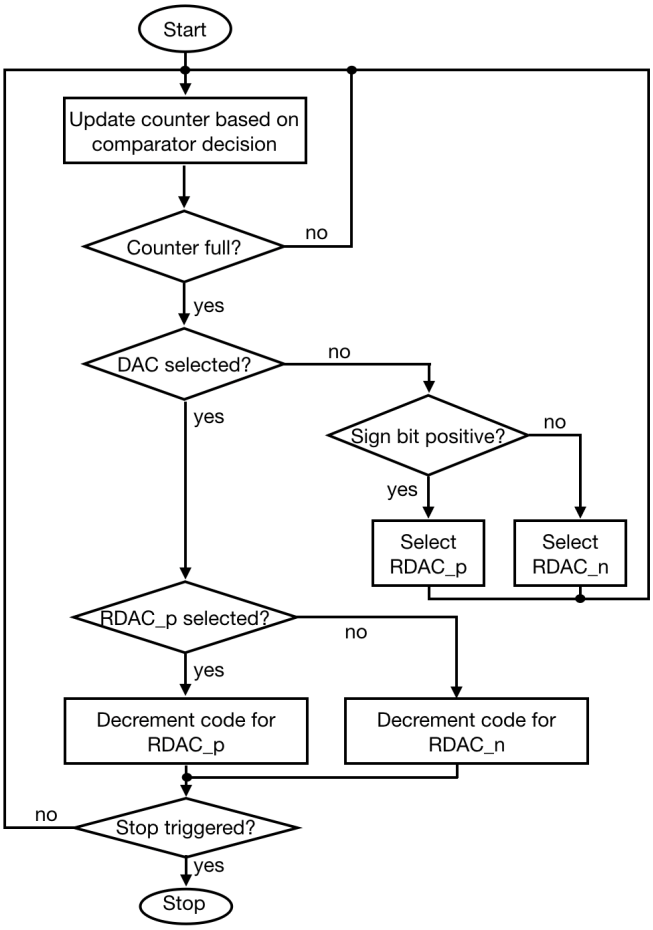

Fig. 5.4. Foreground calibration algorithm

5.2. DTC CONTROLLER AND BACKGROUND DTC GAIN CALIBRATION

This section provides a description of the DTC controller and the implementation of DTC gain calibration. The block diagram of the complete DTC controller and the background calibration logic is presented in Fig. 5.5. The operating principle of the basic DTC modulator has been presented in chapter 2.

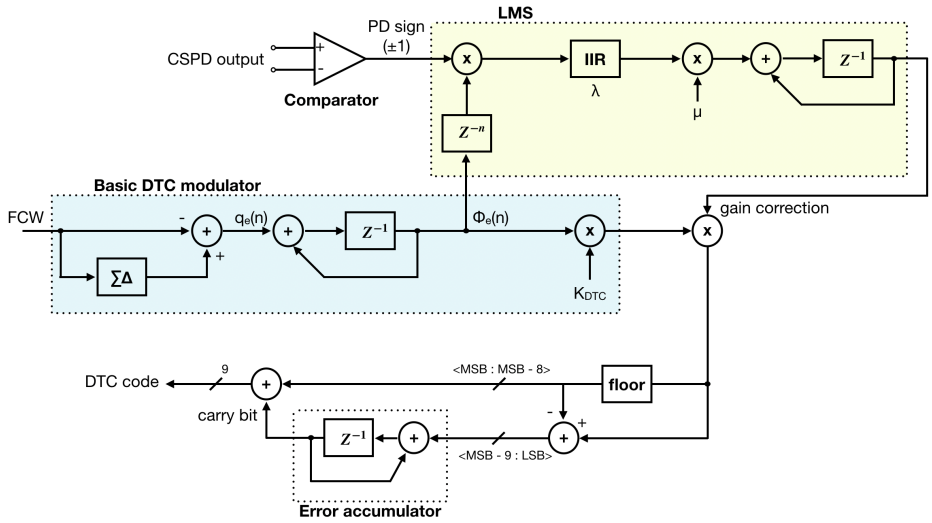

Fig. 5.5. Block diagram of the DTC controller and the background calibration logic

The need for calibrating the DTC gain has been explained in section 3.2. Digital calibration techniques involving the Least Mean Squares (LMS) algorithm have often been used in literature [25], [34], [18]. As a first step towards implementing the LMS algorithm, a given accumulated fractional shift, $\Phi_e(n)$, must be correlated to the CSPD output polarity (also referred to as the error polarity) detected by the comparator for that value of $\Phi_e(n)$. It must be noted that there is delay of a few reference clock cycles between the generation of $\Phi_e(n)$ and the detection of error polarity arising from $\Phi_e(n)$. This correlation can be achieved by inserting a delay element after $\Phi_e(n)$, represented by the Z^{-n} block in Fig. 5.5. Although it is possible to estimate the value n corresponding to the delay, owing to the complexity of the background calibration loop, this delay has been kept programmable. This programmable delay has been realized in hardware by using a chain of flip-flops with their outputs connected to a MUX, as shown in Fig. 5.6. The MUX select lines can be controlled to achieve programmability in the delay.

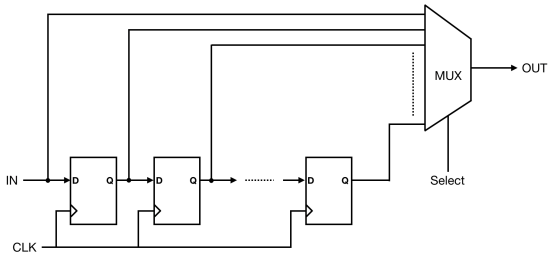

Fig. 5.6. Programmable delay stage used for correlation

The correlated $\Phi_e(n)$ is passed to a 1st order IIR filter (Fig. 5.7) which rejects non-dc components post correlation. The z domain transfer function of the 1st order IIR filter is given by:

$$\frac{Y(z)}{X(z)} = H(z) = \frac{\lambda}{1-(1-\lambda)z^{-1}} \tag{5.1}$$

A first order approximation of the s domain transfer function can be obtained from the z domain transfer function by replacing $z = 1 + \frac{s}{f_{ref}}$ to get:

$$H(s) = \frac{1+\frac{s}{f_{ref}}}{1+\frac{s}{\lambda f_{ref}}} \tag{5.2}$$

Thus the -3dB bandwidth of the IIR filter is given by:

$$f_{-3dB} = \frac{\lambda}{2\pi} f_{ref} \tag{5.3}$$

For $\lambda = 2^{-8}$ and $f_{ref} = 100MHz$, a bandwidth of $62kHz$ is obtained. λ has been implemented as a programmable parameter.

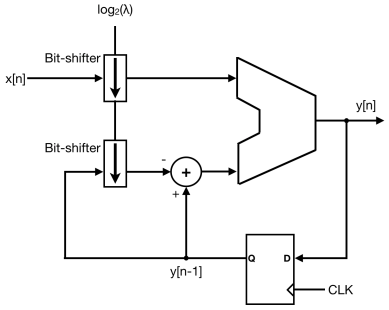

Fig. 5.7. 1st order IIR filter

In order to further control the bandwidth and hence the settling time of the calibration loop, the IIR filter output is passed through an accumulator whose integration step is controlled using the programmable parameter μ (see Fig. 5.5). The DTC code generated by the digital logic (including multiplication of gain correction term) has m bits with $m > 9$. The 9 most significant bits are directed towards the DTC while the remaining bits form the quantization error. To improve the phase wrapping accuracy, this error is accumulated and appended to subsequent DTC codes.

The digital DTC gain calibration loop has been simulated for the PLL with the designed 9 bit DTC having $t_{LSB} = 1.525ps$ and producing a fractional frequency tone at $1.601MHz$. Hence the ideal value of the DTC gain should be $K_{DTC} = T_{vco}/t_{LSB} = 410$. However, in order to test the robustness of the calibration

loop, DTC gain errors of 10% and 20% have been applied. Based on the variation of DTC range over corners (see Table 3.2), a 20% error is a pessimistic case. The calibration loop settles well within $100\mu s$ even for the case of 20% error. The K_{DTC} settling curves have been plotted for $\lambda = 2^{-8}$ and $\mu = 2^{-13}$ in Fig. 5.8.

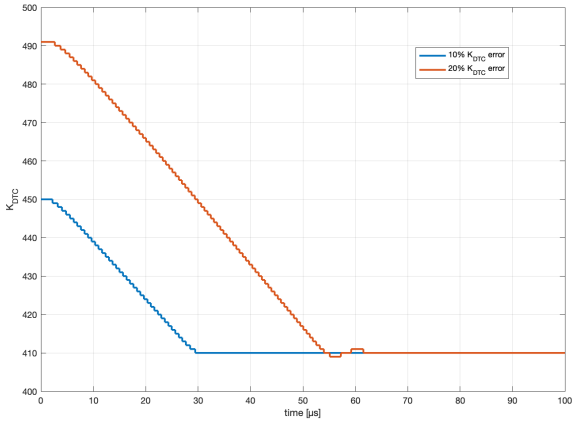

Fig. 5.8. Transient simulation results showing K_{DTC} settling

Larger values for λ and μ reduce the K_{DTC} settling time but tend to render the background calibration loop unstable, thereby preventing K_{DTC} from settling to a constant value. Although not implemented in this design, the settling time could be improved by exploiting a gear-shifting mechanism where the loop bandwidth is initially kept high with large λ and μ which are eventually reduced to lower values to ensure stability.

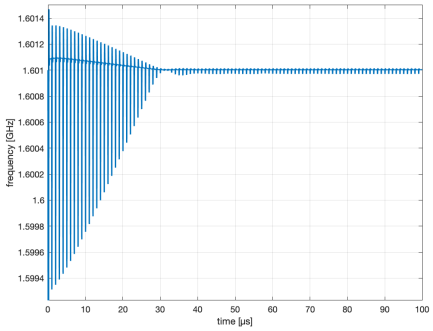

Fig. 5.9. Transient simulation results showing frequency settling

Fig. 5.9 shows the transient plot corresponding to the frequency settling profile for the background calibration loop simulated with a 10% K_{DTC} error.

6

FULL CHIP LAYOUT AND SIMULATION RESULTS

This chapter first presents the layout of the complete PLL chip which was carried out in a 40nm, 1.1V CMOS process. It then provides simulation results showing the PLL performance in both Integer-N and Fractional-N modes. Finally, a comparison has been made with the state-of-the-art.

6.1. CHIP LAYOUT

The complete chip layout containing all the PLL blocks is shown in Fig. 6.1. The core chip area excluding the I/O drivers and decoupling capacitors is $0.15mm^2$. Fig. 6.2 shows the I/O drivers and the different power domains used.

There is coupling of noise as well as signals among the different PLL blocks via the substrate. This can prove detrimental to the operation of the PLL. To prevent this coupling, the different PLL blocks have been placed within separate deep N-wells. The 'NTN' layer has also been placed between the deep N-wells to enhance the resistivity of the Silicon substrate. Since the VCO is most vulnerable to this coupling, an AC ground ring is placed all around it an connected to an AC ground pad on the padring.

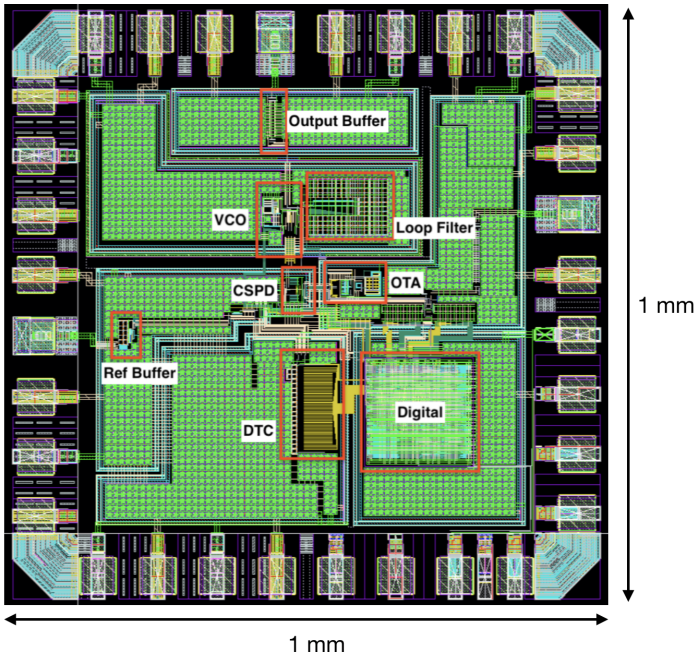

Fig. 6.1. Layout of the complete PLL system.

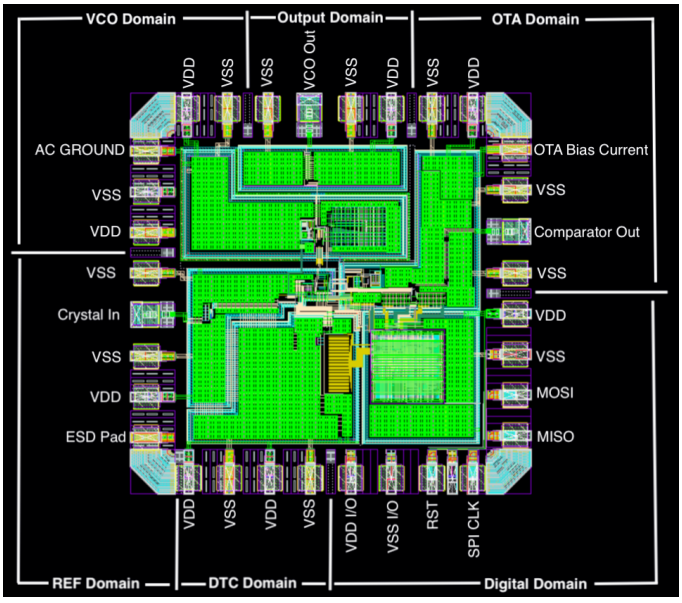

Fig. 6.2. Chip I/Os and power domains.

6.2. POST-LAYOUT SIMULATION RESULTS

This section presents the post-layout simulation results for the PLL chip.

6.2.1. PLL UNDER INTEGER-N OPERATION

Simulations have been performed for an Integer-N channel of 2 GHz. First, the frequency settling behaviour of the loop has been presented in Fig. 6.3. This is followed by plotting the spectrum of the VCO output signal after the PLL locked condition has been achieved (Fig. 6.4).

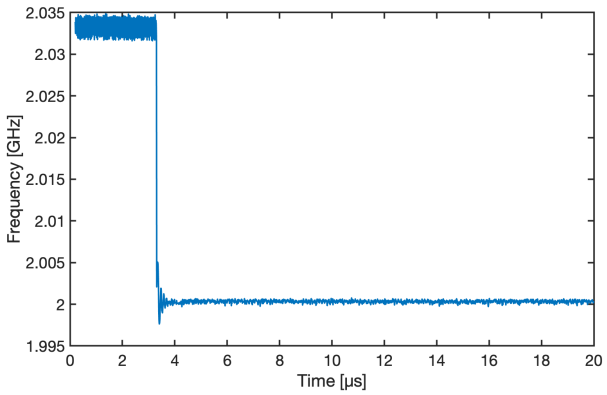

Fig. 6.3. Transient plot showing frequency settling when the PLL is operated in the Integer-N mode. During the first $3\mu s$, SPI registers were programmed to tune the VCO close to 2 GHz.

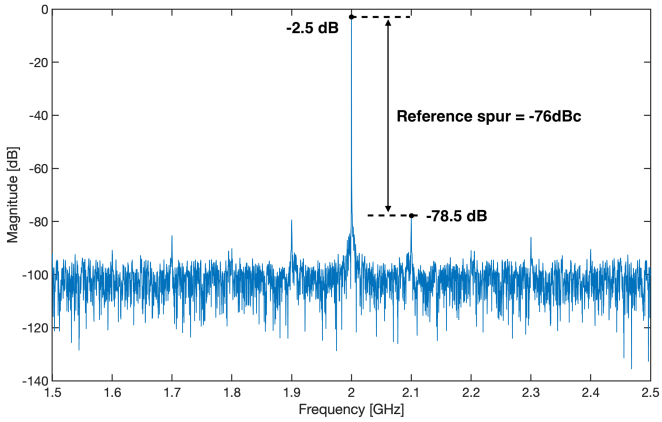

Fig. 6.4. Spectrum of the VCO output signal at 2 GHz.

Reference spur level of -76 dBc can be seen in Fig. 6.4.

6.2.2. PLL UNDER FRACTIONAL-N OPERATION

Simulations have been performed for a Fractional-N channel of 2.001 GHz. The fractional-N channel has been chosen close to an Integer-N channel to observe the worst-case fractional spurs produced. Fig. 6.5 shows the frequency settling plot as the PLL locks-on to a Fractional-N channel.

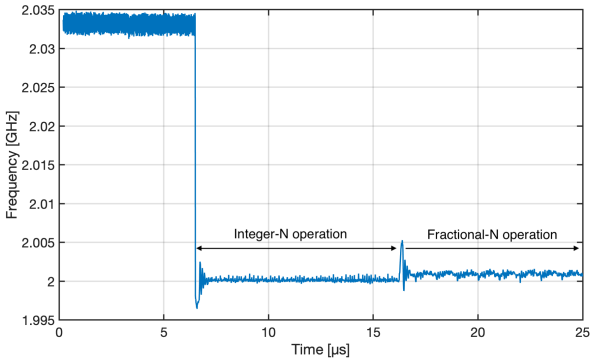

Fig. 6.5. Transient plot showing frequency settling when the PLL is operated in the Fractional-N mode. During the first $3\mu s$, SPI registers were programmed to tune the VCO close to 2 GHz. In the shown plot, the PLL first locks on to the integer-N channel, and then the DTC is switched on for Fractional-N synthesis. However, it is also possible to directly synthesize fractional frequencies.

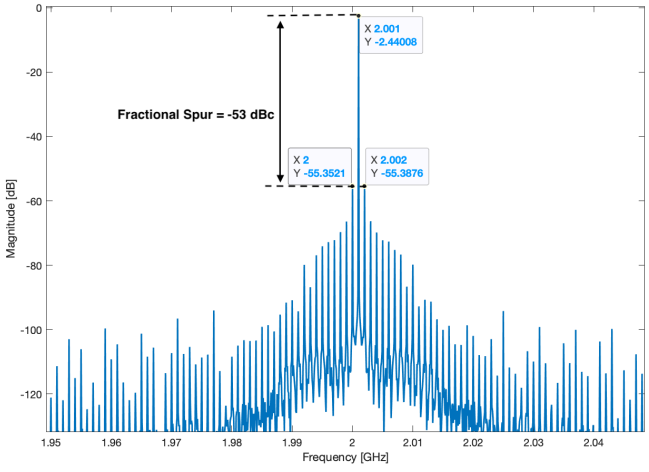

Fig. 6.6. Spectrum of the VCO output signal at 2.001 GHz.

Fig. 6.6 shows the spectrum of the VCO output signal at 2.001 GHz. The worst fractional spurs of -53 dBc are produced at 2 GHz and 2.002 GHz.

6.2.3. PLL PHASE NOISE

The phase noise of the PLL was computed by performing periodic steady-state (PSS) and periodic-noise (pnoise) simulations in Cadence Virtuoso. With the aid of equation 3.10, the phase noise contributions from individual PLL blocks referred to the PLL output are computed and plotted in Fig. 6.7.

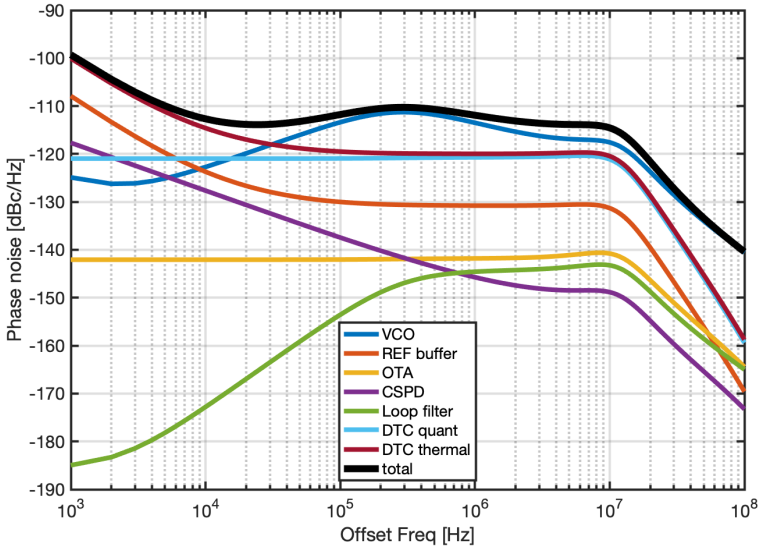

Fig. 6.7. Phase noise contributions of the PLL blocks for the PLL producing a 2 GHz VCO output signal.

The jitter contribution from each PLL block has been computed by using equation 3.13 and tabulated in Table 6.1.

Table 6.1: PLL Jitter contributions

Block	$Jitter_{rms}$ **(fs)**
VCO	890
DTC (quantization noise)	510
DTC (thermal noise)	550
CSPD	30
Ref Buffer	160
OTA	60
Loop filter	40
Total	**1186**

The rms sum of jitter from the in-band components is 780 fs while that from the

VCO is 890 fs. Thus, it can be seen that the loop bandwidth of 10 MHz is optimized to ensure equal jitter contribution from both the in-band and out-of-band (VCO) components.

6.2.4. PLL POWER CONSUMPTION

Table 6.2 illustrates the contribution of each PLL block towards the power consumption. As expected, a major chunk of the power consumption is attributed to the VCO.

Table 6.2: Power consumption of the PLL

Block	Power (μ W)
VCO	600
DTC	140
CSPD	12
Ref Buffer	100
OTA	115
Loop filter	10
Digital	205
Total	**1182**

Fig. 6.8 shows the % distribution of power across the chip.

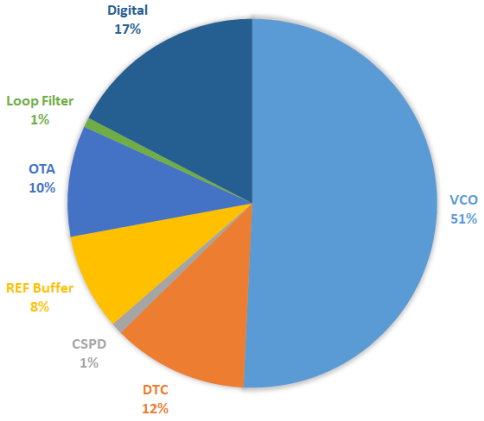

Fig. 6.8. Distribution of power consumed by the chip components.

6.3. COMPARISON WITH THE STATE-OF-THE-ART

Table 6.3 summarizes the comparison drawn between the PLL designed as part of this thesis and other recently published state-of-the-art Fractional-N PLLs employing a ring oscillator.

Table 6.3: Comparison with State-of-the-art RO-based Fractional-N PLLs

	This work	**ISSCC'22 C. Hwang [12]**	**ISSCC'21 H. Park [13]**	**ISSCC'20 T. Seong [15]**	**JSSC'20 Y. Zhang [16]**	**JSSC'19 A.Santiccioli [17]**
Architecture	CSPLL	DPLL	DPLL	DPLL	CP PLL	MDLL
Technology	40nm	65nm	65nm	65nm	40nm	65nm
f_{OUT} (GHz)	1 to 2	4.4 to 5.4	5.2 to 6	4.5 to 6	1.67 to 3.12	1.6 to 3.0
Freq. resolution (kHz)	6.1	6.1	3.1	3.1	0.1	1960
F_{REF} (MHz)	100	100	100	100	50	100
Worst frac. spur (dBc)	-53	-60	-63	-58	-47	-52
Ref. spur (dBc)	-76	-64	-77	NA	-67	-56
rms jitter (fs)	1186 (1k to 100M)	188 (1k to 30M)	365 (10k to 30M)	648 (1k to 30M)	2260 (1k to 100M)	397 (30k to 30M)
PN@1MHz (dBc/Hz)	-116	-133.4	-128.8	-124.9	-103.9	-122.4
Power (mW)	1.18	15.67	9.27	9.88	4.85	2.5
FoM_{jitter} (dB)	-237.8	-242.6	-239.1	-233.8	-226.1	-244
Area (mm^2)	0.15	0.139	0.146	0.108	0.086	0.0275

[1] [2]

The above comparison shows that the PLL designed as part of this thesis consumes the lowest power while achieving a decent Figure of Merit. The reference spur level is comparable to the lowest reference spur level reported while the fractional spur remains 10dB higher than the lowest reported fractional spur level.

The next chapter discusses a few improvements that can be made in the system to achieve a better performance.

[1] $FoM_{jitter} = 20log_{10}(jitter_{rms}/1s) + 10log_{10}(power_{DC}/1mW)$
[2] PN has been normalized to 1 GHz

7

CONCLUSIONS AND FUTURE SCOPE

A DTC-based subsampling Fractional-N PLL has been designed as part of this thesis. The PLL employs a high-gain charge-sampling phase detector which greatly suppresses the loop noise arising from the OTA, loop filter and the phase detector itself. The charge-sampling nature of the PLL also aids in reducing the reference spur levels without compromising the jitter performance of the PLL. To enable fractional-N frequency synthesis, a DTC has been inserted in the reference path to facilitate phase modulation of the reference to align with the fractional VCO tone. A highly linear DTC architecture has been chosen to ensure low levels of fractional spurs. To ensure robust PLL operation in the presence of PVT variations, a background DTC gain calibration technique has been introduced.

The Fractional-N PLL utilizes a 100 MHz reference and produces an output frequency range of 1 to 2 GHz while consuming a power of 1.18 mW. From simulations, it can be seen that the PLL has a reference spur level of -76 dBc and the worst fractional spur level of -53dBc. The PLL achieves a Figure of Merit (FoM_{jitter}) of -237.8 dB.

7.1. FUTURE SCOPE

This section provides a brief overview of the improvement points that can be applied to future versions of the PLL chip.

7.1.1. REDUCTION OF FRACTIONAL SPURS

The PLL produces fractional spurs having a relatively high level when compared to the lowest reported fractional spur level in literature. The fractional spur level can be reduced in this PLL system by using a 2nd or higher order $\Delta\Sigma$ modulator (DSM) in the DTC controller (see Fig. 2.3) to produce more randomness in the DTC

code, thereby reducing the spur level. However, increasing the order of the DSM also requires an increase in the dynamic range (DR) of the DTC, as given by [35]:

$$DR_{DTC} = 2^{n-1} T_{VCO} \tag{7.1}$$

where n is the DSM order. Thus, a 2nd-order DSM requires a DR of $2T_{VCO}$. A twofold increase in the dynamic range can be achieved by either reducing the slope of the DTC ramp signal to half or by increasing the range of the DTC ramp start voltage by two times. However both of these approaches pose limitations. While it would not be possible to slow down the DTC ramp due to the tight timing requirements of the DTC, increasing the range of ramp start voltage would introduce more nonlinearity in the DTC operation.

To meet the DR requirements of a 2nd-order DSM, the VCO can be designed to operate at twice the frequency (without any change in the PLL FoM), thereby reducing T_{VCO} to half its present value. This approach also ensures that no design changes are required in the DTC.

7.1.2. INCREASING REFERENCE FREQUENCY TO IMPROVE PLL FIGURE OF MERIT

In this subsection, the impact of increasing the reference frequency on the PLL Figure of Merit is considered. A frequency doubler (multiplier) can be designed without substantially increasing the system noise or power [25]. Based on the individual phase noise (Fig. 6.7) and jitter contributions (Table 6.1) of the PLL blocks, the DTC and the VCO can be identified as the dominant in-band and out-of-band noise sources respectively.

First considering the in-band noise, it can be seen from equation 4.3 that the DTC quantization noise $S_{n,DTC} \propto \frac{(f_{VCO}.\Delta t_{DTC})^2}{f_{ref}}$ where Δt_{DTC} is the DTC resolution. Doubling the reference frequency f_{ref} would result in reducing the quantization noise by half. Since f_{ref} acts as the clock for the DTC, doubling f_{ref} would mean reducing the DTC dynamic range by a factor of two. Thus the VCO frequency f_{VCO} is doubled and the DTC resolution, Δt_{DTC} is halved, leading to an overall reduction of DTC quantization noise by a factor of two.

The phase noise of a free-running ring oscillator at an offset frequency Δf is given by:

$$S_{n,osc} = \frac{16\gamma}{3\eta} \cdot \frac{kT}{P} \cdot \left(\frac{f_{osc}}{\Delta f}\right)^2 \tag{7.2}$$

where η is a constant close to unity and $\gamma = 2/3$ for long channel devices. For a fixed power consumed by the oscillator, it is true that the the phase noise becomes four times worse as the oscillator frequency is doubled. However, the oscillator jitter will remain the same. Increasing the reference frequency gives an opportunity to increase the loop bandwidth without compromising on the loop

stability. Thus an increased loop bandwidth will suppress the oscillator noise and jitter contribution, therefore reducing the out-of-band PLL noise.

7.1.3. ON-CHIP TUNING OF THE OSCILLATOR TUNING WORD (OTW)

The foreground offset compensation circuit used to compensate the effective comparator offset has a range of about 40 mV. If the Oscillator Tuning Word (OTW) is programmed such that the free-running VCO frequency is not close enough to that of the desired channel, the CSPD maintains a non-zero output even when the PLL is locked. This non-zero CSPD output might exceed 40mV, thus rendering the foreground calibration ineffective.

In the present chip architecture, the Oscillator Tuning Word (OTW) is manually tuned to bring the VCO frequency close to the desired Integer-N or Fractional-N channel. This condition is detected by bringing the comparator output to an I/O pad (see Fig. 6.2) and probing it off-chip to check for equal probability of '1s' and '0s'. This function can be performed on-chip by implementing suitable digital logic.

BIBLIOGRAPHY

[1] R.C.H. van de Beek et al. "A 2.5-10-GHz clock multiplier unit with 0.22-ps RMS jitter in standard 0.18-/spl mu/m CMOS". In: *IEEE Journal of Solid-State Circuits* 39.11 (2004), pp. 1862–1872. DOI: 10.1109/JSSC.2004.835833.

[2] Chun-Ming Hsu, Matthew Z. Straayer, and Michael H. Perrott. "A Low-Noise Wide-BW 3.6-GHz Digital ΔΣ Fractional-N Frequency Synthesizer With a Noise-Shaping Time-to-Digital Converter and Quantization Noise Cancellation". In: *IEEE Journal of Solid-State Circuits* 43.12 (2008), pp. 2776–2786. DOI: 10.1109/JSSC.2008.2005704.

[3] Xiang Gao et al. "A Low Noise Sub-Sampling PLL in Which Divider Noise is Eliminated and PD/CP Noise is Not Multiplied by N^2". In: *IEEE Journal of Solid-State Circuits* 44.12 (2009), pp. 3253–3263. DOI: 10.1109/JSSC.2009.2032723.

[4] Xiang Gao et al. "A 2.2GHz sub-sampling PLL with 0.16psrms jitter and -125dBc/Hz in-band phase noise at 700μW loop-components power". In: *2010 Symposium on VLSI Circuits*. 2010, pp. 139–140. DOI: 10.1109/VLSIC.2010.5560323.

[5] Juyeop Kim et al. "16.2 A 76fsrms Jitter and –40dBc Integrated-Phase-Noise 28-to-31GHz Frequency Synthesizer Based on Digital Sub-Sampling PLL Using Optimally Spaced Voltage Comparators and Background Loop-Gain Optimization". In: *2019 IEEE International Solid- State Circuits Conference - (ISSCC)*. 2019, pp. 258–260. DOI: 10.1109/ISSCC.2019.8662532.

[6] Luca Bertulessi et al. "A 30-GHz Digital Sub-Sampling Fractional- *N* PLL With -238.6-dB Jitter-Power Figure of Merit in 65-nm LP CMOS". In: *IEEE Journal of Solid-State Circuits* 54.12 (2019), pp. 3493–3502. DOI: 10.1109/JSSC.2019.2940332.

[7] Kuba Raczkowski et al. "A 9.2–12.7 GHz Wideband Fractional-N Subsampling PLL in 28 nm CMOS With 280 fs RMS Jitter". In: *IEEE Journal of Solid-State Circuits* 50.5 (2015), pp. 1203–1213. DOI: 10.1109/JSSC.2015.2403373.

[8] Jiang Gong et al. "A Low-Jitter and Low-Spur Charge-Sampling PLL". In: *IEEE Journal of Solid-State Circuits* 57.2 (2022), pp. 492–504. DOI: 10.1109/JSSC.2021.3105335.

[9] Dhon-Gue Lee and Patrick P. Mercier. "A Sub-mW 2.4-GHz Active-Mixer-Adopted Sub-Sampling PLL Achieving an FoM of -256 dB". In: *IEEE Journal of Solid-State Circuits* 55.6 (2020), pp. 1542–1552. DOI: 10.1109/JSSC.2019.2951377.

[10] Jiang Gong et al. “A 10-to-12 GHz 5 mW Charge-Sampling PLL Achieving 50 fsec RMS Jitter, -258.9 dB FOM and -65 dBc Reference Spur”. In: *2020 IEEE Radio Frequency Integrated Circuits Symposium (RFIC)*. 2020, pp. 15–18. DOI: 10.1109/RFIC49505.2020.9218380.

[11] T.A.D. Riley, M.A. Copeland, and T.A. Kwasniewski. “Delta-sigma modulation in fractional-N frequency synthesis”. In: *IEEE Journal of Solid-State Circuits* 28.5 (1993), pp. 553–559. DOI: 10.1109/4.229400.

[12] Chanwoong Hwang et al. “A 188fsrms-Jitter and -243d8-FoMjitter 5.2GHz-Ring-DCO-Based Fractional-N Digital PLL with a 1/8 DTC-Range-Reduction Technique Using a Quadruple-Timing-Margin Phase Selector”. In: *2022 IEEE International Solid- State Circuits Conference (ISSCC)*. Vol. 65. 2022, pp. 378–380. DOI: 10.1109/ISSCC42614.2022.9731646.

[13] Hangi Park et al. “32.1 A 365fsrms-Jitter and -63dBc-Fractional Spur 5.3GHz-Ring-DCO-Based Fractional-N DPLL Using a DTC Second/Third-Order Nonlinearity Cancelation and a Probability-Density-Shaping –M”. In: *2021 IEEE International Solid- State Circuits Conference (ISSCC)*. Vol. 64. 2021, pp. 442–444. DOI: 10.1109/ISSCC42613.2021.9365798.

[14] Qiaochu Zhang et al. “29.4 A Fractional-N Digital MDLL with Background Two-Point DTC Calibration Achieving -60dBc Fractional Spur”. In: *2021 IEEE International Solid- State Circuits Conference (ISSCC)*. Vol. 64. 2021, pp. 410–412. DOI: 10.1109/ISSCC42613.2021.9365819.

[15] Taeho Seong et al. “17.3 A -58dBc-Worst-Fractional-Spur and -234dB-FoMjitter, 5.5GHz Ring-DCO-Based Fractional-N DPLL Using a Time-Invariant-Probability Modulator, Generating a Nonlinearity-Robust DTC-Control Word”. In: *2020 IEEE International Solid- State Circuits Conference - (ISSCC)*. 2020, pp. 270–272. DOI: 10.1109/ISSCC19947.2020.9062948.

[16] Yanlong Zhang et al. “A Fractional- N PLL With Space–Time Averaging for Quantization Noise Reduction”. In: *IEEE Journal of Solid-State Circuits* 55.3 (2020), pp. 602–614. DOI: 10.1109/JSSC.2019.2950154.

[17] Alessio Santiccioli et al. “A 1.6-to-3.0-GHz Fractional- N MDLL With a Digital-to-Time Converter Range-Reduction Technique Achieving 397-fs Jitter at 2.5-mW Power”. In: *IEEE Journal of Solid-State Circuits* 54.11 (2019), pp. 3149–3160. DOI: 10.1109/JSSC.2019.2941259.

[18] Salvatore Levantino, Giovanni Marzin, and Carlo Samori. “An Adaptive Pre-Distortion Technique to Mitigate the DTC Nonlinearity in Digital PLLs”. In: *IEEE Journal of Solid-State Circuits* 49.8 (2014), pp. 1762–1772. DOI: 10.1109/JSSC.2014.2314436.

[19] Jiayoon Zhiyu Ru et al. “A High-Linearity Digital-to-Time Converter Technique: Constant-Slope Charging”. In: *IEEE Journal of Solid-State Circuits* 50.6 (2015), pp. 1412–1423. DOI: 10.1109/JSSC.2015.2414421.

[20] Nereo Markulic et al. "A 10-bit, 550-fs step Digital-to-Time Converter in 28nm CMOS". In: *ESSCIRC 2014 - 40th European Solid State Circuits Conference (ESSCIRC)*. 2014, pp. 79–82. DOI: 10.1109/ESSCIRC.2014.6942026.

[21] Peng Chen et al. "A 31- μ W, 148-fs Step, 9-bit Capacitor-DAC-Based Constant-Slope Digital-to-Time Converter in 28-nm CMOS". In: *IEEE Journal of Solid-State Circuits* 54.11 (2019), pp. 3075–3085. DOI: 10.1109/JSSC.2019.2939663.

[22] R. Best. *Phase-Locked Loops*. McGraw-Hill professional engineering. McGraw-Hill Education, 2003. ISBN: 9780071501231. URL: https://books.google.co.in/books?id=lasCAI23NBYC.

[23] B. Razavi. *Design of Integrated Circuits for Optical Communications*. McGraw-Hill Series in Electrical and Computer Engineering. McGraw-Hill, 2003. ISBN: 9780072822588. URL: https://books.google.co.in/books?id=Pl9TAAAAMAAJ.

[24] Davide Tasca et al. "A 2.9–4.0-GHz Fractional-N Digital PLL With Bang-Bang Phase Detector and 560-fs_{rms} Integrated Jitter at 4.5-mW Power". In: *IEEE Journal of Solid-State Circuits* 46.12 (2011), pp. 2745–2758. DOI: 10.1109/JSSC.2011.2162917.

[25] Wanghua Wu et al. "A 28-nm 75-fsrms Analog Fractional- N Sampling PLL With a Highly Linear DTC Incorporating Background DTC Gain Calibration and Reference Clock Duty Cycle Correction". In: *IEEE Journal of Solid-State Circuits* 54.5 (2019), pp. 1254–1265. DOI: 10.1109/JSSC.2019.2899726.

[26] Chi-Hung Lin and K. Bult. "A 10-b, 500-MSample/s CMOS DAC in 0.6 mm/sup 2/". In: *IEEE Journal of Solid-State Circuits* 33.12 (1998), pp. 1948–1958. DOI: 10.1109/4.735535.

[27] Marcel Pelgrom. "Analog-to-Digital Conversion". In: Jan. 2010, pp. 249–319. ISBN: 978-90-481-8887-1. DOI: 10.1007/978-90-481-8888-8_8.

[28] Jiang Gong et al. "A 2.7mW 45fsrms-Jitter Cryogenic Dynamic-Amplifier-Based PLL for Quantum Computing Applications". In: *2021 IEEE Custom Integrated Circuits Conference (CICC)*. 2021, pp. 1–2. DOI: 10.1109/CICC51472.2021.9431541.

[29] Behzad Razavi. "The Ring Oscillator [A Circuit for All Seasons]". In: *IEEE Solid-State Circuits Magazine* 11.4 (2019), pp. 10–81. DOI: 10.1109/MSSC.2019.2939771.

[30] Behzad Razavi. "The StrongARM Latch [A Circuit for All Seasons]". In: *IEEE Solid-State Circuits Magazine* 7.2 (2015), pp. 12–17. DOI: 10.1109/MSSC.2015.2418155.

[31] Chi-Hang Chan et al. "A voltage-controlled capacitance offset calibration technique for high resolution dynamic comparator". In: *2009 International SoC Design Conference (ISOCC)*. 2009, pp. 392–395. DOI: 10.1109/SOCDC.2009.5423836.

7

[32] Walt Kester. *Basic DAC Architectures III: Segmented DACs*. 2009. URL: https://www.analog.com/media/en/training-seminars/tutorials/mt-016.pdf.

[33] Dennis Dempsey and Christopher Gorman. *Digital-to-Analog Converter*. U.S. Patent 5,969,657. 1999.

[34] Yue Chen et al. "A Fractional-N Digitally Intensive PLL Achieving 428-fs Jitter and <-54-dBc Spurs Under 50-mVpp Supply Ripple". In: *IEEE Journal of Solid-State Circuits* 57.6 (2022), pp. 1749–1764. DOI: 10.1109/JSSC.2021.3123386.

[35] Tuan Minh Vo, Salvatore Levantino, and Carlo Samori. "Analysis of fractional-n bang-bang digital PLLs using phase switching technique". In: *2016 12th Conference on Ph.D. Research in Microelectronics and Electronics (PRIME)*. 2016, pp. 1–4. DOI: 10.1109/PRIME.2016.7519545.

A

RTL CODE FOR THE DTC CONTROLLER

```
module dtc_code_gen(
i_clk,
i_rst,
i_enable, // EN signal for the entire module
i_background_calibration_en,
i_frac_shift,
i_kdtc,
i_cmprtr_sign,
i_flip_sign,
i_fifo_delay,
lambda,
mu,
o_dtc_code);

// parameters

parameter FRAC_WIDTH = 14;
parameter KDTC_WIDTH = 10;
parameter GAIN_CORR_WIDTH = 14;
parameter CODE_WIDTH = 9;
parameter MULT_WIDTH = FRAC_WIDTH + KDTC_WIDTH +
    GAIN_CORR_WIDTH;
parameter MULT_1_WIDTH = FRAC_WIDTH + KDTC_WIDTH;
parameter ERR_WIDTH = MULT_WIDTH - CODE_WIDTH - 1;
parameter FIFO_DEPTH = 8;
```

```
parameter FIFO_PTR_WIDTH = $clog2(FIFO_DEPTH);

// I/Os

input i_clk;
input i_rst; // async rst
input i_enable;
input i_background_calibration_en;
input [FRAC_WIDTH-1 : 0] i_frac_shift;
input [KDTC_WIDTH-1 : 0] i_kdtc;
input i_cmprtr_sign;
input i_flip_sign;
input [FIFO_PTR_WIDTH-1 : 0] i_fifo_delay;
input [7 : 0] lambda;
input [7 : 0] mu;
output reg [CODE_WIDTH-1 : 0]  o_dtc_code;

// Registers and wires

reg [FRAC_WIDTH-1 : 0] acc_shift;
wire [FRAC_WIDTH-1 : 0] corr_acc_shift; // used for
    correlation
reg [ERR_WIDTH-1 : 0] acc_err;
reg acc_err_carry;
wire [CODE_WIDTH-1 : 0] w_dtc_code;
wire [MULT_WIDTH-1 : 0] gain_mult;
wire [MULT_1_WIDTH-1 : 0] gain_mult_1;
reg [MULT_1_WIDTH-1 : 0] gain_mult_1_reg;
wire [GAIN_CORR_WIDTH : 0] acc_shift_neg;

// FIFO related
reg [FRAC_WIDTH-1 : 0] context_fifo [0:7];
reg [FIFO_PTR_WIDTH-1 : 0] wr_ptr;
reg [FIFO_PTR_WIDTH-1 : 0] rd_ptr;
////////////////////////////////////////////

wire signed [GAIN_CORR_WIDTH : 0] iir_in;
reg signed [GAIN_CORR_WIDTH : 0] iir_out;
reg signed [GAIN_CORR_WIDTH : 0] gain_correct;

integer i;

////////////////////////////////////////////

// Accumulation of main shift
```

```
always @ (posedge i_clk or negedge i_rst)
begin
    if(~i_rst)
    begin
        acc_shift <= {FRAC_WIDTH{1'b0}};
    end
    else if(~i_enable)
    begin
        acc_shift <= {FRAC_WIDTH{1'b0}};
    end
    else
    begin // overflow is not a problem here: it is expected!
        acc_shift <= acc_shift + i_frac_shift;
    end
end

// Free-running FIFO to implement Z^(-n)

always @ (posedge i_clk or negedge i_rst)
begin
    if(~i_rst)
    begin
        for(i = 0; i < FIFO_DEPTH; i = i + 1)
        begin
            context_fifo[i] <= {FRAC_WIDTH{1'b0}};
        end
    end
    else if(~i_enable)
    begin
        for(i = 0; i < FIFO_DEPTH; i = i + 1)
        begin
            context_fifo[i] <= {FRAC_WIDTH{1'b0}};
        end
    end
    else
    begin
        context_fifo[wr_ptr] <= acc_shift;
    end
end

always @ (posedge i_clk or negedge i_rst)
begin
    if(~i_rst)
    begin
        wr_ptr <= {FIFO_PTR_WIDTH{1'b0}};
```

```
            rd_ptr <= {FIFO_PTR_WIDTH{1'b0}};
        end
        else if(~i_enable)
        begin
            wr_ptr <= {FIFO_PTR_WIDTH{1'b0}};
            rd_ptr <= {FIFO_PTR_WIDTH{1'b0}};
        end
        else
        begin
            wr_ptr <= rd_ptr + i_fifo_delay + 1'b1;
            rd_ptr <= rd_ptr + 1'b1;
        end
end

assign corr_acc_shift = context_fifo[rd_ptr];

//////////////////////////////////////////////
// Gain correction term using LMS algorithm

assign acc_shift_neg = ~{1'b0,(corr_acc_shift)} + 1'b1; //
    taking 2's complement
assign iir_in = ( ((i_cmprtr_sign == 1'b1) && (i_flip_sign
    == 1'b0)) || ((i_cmprtr_sign == 1'b0) && (i_flip_sign ==
    1'b1)) ) ? {1'b0,corr_acc_shift} : acc_shift_neg;

always @(posedge i_clk or negedge i_rst)
begin
    if(~i_rst)
    begin
        iir_out <= {(GAIN_CORR_WIDTH+1){1'b0}};
    end
    else if(~i_enable)
    begin
        iir_out <= {(GAIN_CORR_WIDTH+1){1'b0}};
    end
    else
    begin
        iir_out <= iir_out + (iir_in >>> lambda) - (iir_out
            >>> lambda) ;
    end
end

always @(posedge i_clk or negedge i_rst)
begin
    if(~i_rst)
```

```
    begin
        gain_correct <= 15'h2000;
        // MSB for gain_correct must always remain '0' since
            it is
        // positive
        // It has been defined as 'signed' to ensure proper
           signed addition
    end
    else if(~i_enable || ~i_background_calibration_en)
    begin
        gain_correct <= 15'h2000;
    end
    else
    begin
        gain_correct <= gain_correct + (iir_out >>> mu);
    end
end
//////////////////////////////////////////////


// Gain scaling multiplier
assign gain_mult = gain_mult_1_reg * gain_correct[
   GAIN_CORR_WIDTH-1 : 0];
assign gain_mult_1 = acc_shift * i_kdtc;
// sign bit of gain_correct has been discarded

// Intermediate multiplication result
always @ (posedge i_clk or negedge i_rst)
begin
    if(~i_rst)
    begin
        gain_mult_1_reg <= {MULT_1_WIDTH{1'b0}};
    end
    else if(~i_enable)
    begin
        gain_mult_1_reg <= {MULT_1_WIDTH{1'b0}};
    end
    else
    begin
        gain_mult_1_reg <= gain_mult_1;
    end
end

// Accumulation of error
always @ (posedge i_clk or negedge i_rst)
```

```
begin
    if(~i_rst)
    begin
        acc_err <= {ERR_WIDTH{1'b0}};
        acc_err_carry <= 1'b0;
    end
    else if(~i_enable)
    begin
        acc_err <= {ERR_WIDTH{1'b0}};
        acc_err_carry <= 1'b0;
    end
    else
    begin
        {acc_err_carry,acc_err} <= acc_err + gain_mult[
            MULT_WIDTH-CODE_WIDTH-2:0];
    end
end

assign w_dtc_code = gain_mult[MULT_WIDTH-2 : MULT_WIDTH-
    CODE_WIDTH-1];

// Register DTC code
always @ (posedge i_clk or negedge i_rst)
begin
    if(~i_rst)
    begin
        o_dtc_code <= {CODE_WIDTH{1'b0}};
    end
    else if(~i_enable)
    begin
        o_dtc_code <= {CODE_WIDTH{1'b0}};
    end
    else
    begin
        o_dtc_code <= w_dtc_code  + acc_err_carry;
        // o_dtc_code <= (gain_mult[MULT_WIDTH-1 -:
            CODE_WIDTH] + acc_err_carry);
    end
end


endmodule
```

B

RTL CODE FOR THE FOREGROUND CALIBRATION LOGIC

```
module foreground_calibration(
i_clk,
i_rst,
i_enable,
i_cmprtr_sign,
o_rdac_p_msb,
o_rdac_p_lsb,
o_rdac_n_msb,
o_rdac_n_lsb,
i_settle_time,
i_stop_time,
i_acc_full,
// Debug
o_settle_time,
o_stop_time,
o_dac_select_done,
o_dac_select);

// parameters

parameter NR_MSB = 32;
parameter MSB_WIDTH = $clog2(NR_MSB);
parameter NR_LSB = 32;
```

```
parameter LSB_WIDTH = $clog2(NR_LSB);
parameter TIMER_WIDTH = 16;
parameter COUNTER_WIDTH = 16;

// IOs

input i_clk;
input i_rst;
input i_enable; // enable foreground calibration block
input i_cmprtr_sign;

output [NR_MSB-1 : 0] o_rdac_p_msb;
output [NR_LSB-1 : 0] o_rdac_p_lsb;
output [NR_MSB-1 : 0] o_rdac_n_msb;
output [NR_LSB-1 : 0] o_rdac_n_lsb;

input [TIMER_WIDTH-1 : 0] i_settle_time; // Begin
    calibration after PLL settles in integerN mode
input [TIMER_WIDTH-1 : 0] i_stop_time; // Stop calibration
    when this count is reached
input [COUNTER_WIDTH-1 : 0] i_acc_full;// Threshold for
    accummulator (BW control)

// Debug
output [TIMER_WIDTH-1 : 0] o_settle_time;
output [TIMER_WIDTH-1 : 0] o_stop_time;
output o_dac_select_done;
output o_dac_select;

//
    ////////////////////////////////////////////////////////////////


// Registers and wires

// One-hot encoding
reg [NR_MSB-1 : 0] r_rdac_p_msb;
reg [NR_LSB-1 : 0] r_rdac_p_lsb;
reg [NR_MSB-1 : 0] r_rdac_n_msb;
reg [NR_LSB-1 : 0] r_rdac_n_lsb;
// binary encoding
reg [MSB_WIDTH-1 : 0] r_count_p_msb;
reg [LSB_WIDTH-1 : 0] r_count_p_lsb;
reg [MSB_WIDTH-1 : 0] r_count_n_msb;
reg [LSB_WIDTH-1 : 0] r_count_n_lsb;
```

```
//////////////////////////////////////

reg [TIMER_WIDTH-1 : 0] settle_timer;
reg [TIMER_WIDTH-1 : 0] stop_timer;
reg signed [COUNTER_WIDTH : 0] acc;
wire start; // indicates when to begin the calibration
wire stop; // indicates when to end the calibration
wire acc_full;
wire acc_sign;
wire [COUNTER_WIDTH : 0] acc_full_pos;
wire [COUNTER_WIDTH : 0] acc_full_neg;
reg dac_select;
reg dac_select_done;
//////////////////////////////////////////////////

// Synchronization FFs to prevent CDC
reg r_enable;
reg r_enable_2;
reg r_enable_3;

always @ (posedge i_clk or negedge i_rst)
begin
    if(~i_rst)
    begin
        r_enable   <= 1'b0;
        r_enable_2 <= 1'b0;
        r_enable_3 <= 1'b0;
    end
    else
    begin
        r_enable   <= i_enable;
        r_enable_2 <= r_enable;
        r_enable_3 <= r_enable_2;
    end
end



// Timer to ensure PLL settles in integerN mode
// Example settling time: 10us
always @ (posedge i_clk or negedge i_rst)
begin
    if(~i_rst)
    begin
        settle_timer <= {TIMER_WIDTH{1'b0}};
```

```
    end
    else if(~r_enable_3)
    begin
        settle_timer <= {TIMER_WIDTH{1'b0}};
    end
    else if(!start)
    begin
        settle_timer <= settle_timer + 1'b1;
    end
end

assign start = (settle_timer == i_settle_time) ? 1'b1 : 1'b0
    ;


// Comparator output flip counting logic
always @ (posedge i_clk or negedge i_rst)
begin
    if(~i_rst)
    begin
        stop_timer <= {TIMER_WIDTH{1'b0}};
    end
    else if(~r_enable_3)
    begin
        stop_timer <= {TIMER_WIDTH{1'b0}};
    end
    else if(start && !stop)
    begin
        stop_timer <= stop_timer + 1'b1;
    end
end

assign stop = (stop_timer == i_stop_time) ? 1'b1 : 1'b0;

// Accumulator
always @ (posedge i_clk or negedge i_rst)
begin
    if(~i_rst)
    begin
        acc <= {(COUNTER_WIDTH+1){1'b0}};
    end
    else if(~r_enable_3 || acc_full)
    begin
        acc <= {(COUNTER_WIDTH+1){1'b0}};
    end
```

```
    else if(start && !stop && i_cmprtr_sign)
    begin
        acc <= acc + 1'b1;
    end
    else if(start && !stop && !i_cmprtr_sign)
    begin
        acc <= acc - 1'b1;
    end
end
assign acc_full_pos = {1'b0,i_acc_full};
assign acc_full_neg = {1'b1,(~(i_acc_full)+1'b1)};
assign acc_full = ((acc == acc_full_pos) || (acc ==
   acc_full_neg));
assign acc_sign = acc[COUNTER_WIDTH]; // '0': +ve, '1': -ve

// Selecting DAC for tuning
// Selection is made based on first accumulator decision
always @ (posedge i_clk or negedge i_rst)
begin
    if(~i_rst)
    begin
        dac_select_done <= 1'b0;
    end
    else if(~r_enable_3)
    begin
        dac_select_done <= 1'b0;
    end
    else if(!dac_select_done && start && acc_full)
    begin
        dac_select_done <= 1'b1;
    end
end

always @ (posedge i_clk or negedge i_rst)
begin
    if(~i_rst)
    begin
        dac_select <= 1'b0;
    end
    else if (~r_enable_3)
    begin
        dac_select <= 1'b0;
    end
    else if(!dac_select_done && start && !acc_sign &&
       acc_full)
```

```
    begin
        dac_select <= 1'b1;
    end
end

// DAC_p tuning logic
always @ (posedge i_clk or negedge i_rst)
begin
    if(~i_rst)
    begin
        r_count_p_msb <= 'h1A;
        r_count_p_lsb <= {(LSB_WIDTH){1'b0}};
    end
    else if(~r_enable_3)
    begin
        r_count_p_msb <= 'h1A;
        r_count_p_lsb <= {(LSB_WIDTH){1'b0}};
    end
    else if(dac_select_done && dac_select && !stop &&
       acc_full)
    begin
        if(!acc_sign) // acc_sign = '0' means positive
        begin
            if(r_count_p_lsb =={(LSB_WIDTH){1'b0}})
            begin
                r_count_p_msb <= r_count_p_msb - 1'b1;
            end
            r_count_p_lsb <= r_count_p_lsb - 1'b1;
        end
        else
        begin
            if(r_count_p_lsb == 'h1F)
            begin
                r_count_p_msb <= r_count_p_msb + 1'b1;
            end
            r_count_p_lsb <= r_count_p_lsb + 1'b1;
        end
    end
end

// DAC_n tuning logic
always @ (posedge i_clk or negedge i_rst)
begin
    if(~i_rst)
    begin
```

```
            r_count_n_msb <= 'h1A;
            r_count_n_lsb <= {(LSB_WIDTH){1'b0}};
        end
        else if(~r_enable_3)
        begin
            r_count_n_msb <= 'h1A;
            r_count_n_lsb <= {(LSB_WIDTH){1'b0}};
        end
        else if(dac_select_done && !dac_select && !stop &&
            acc_full)
        begin
            if(acc_sign)
            begin
                if(r_count_n_lsb == {(LSB_WIDTH){1'b0}})
                begin
                    r_count_n_msb <= r_count_n_msb - 1'b1;
                end
                r_count_n_lsb <= r_count_n_lsb - 1'b1;
            end
            else
            begin
                if(r_count_n_lsb == 'h1F)
                begin
                    r_count_n_msb <= r_count_n_msb + 1'b1;
                end
                r_count_n_lsb <= r_count_n_lsb + 1'b1;
            end
        end
    end


    // Applying One-hot encoding for DAC control
    always @ (posedge i_clk or negedge i_rst)
    begin
        if(~i_rst)
        begin
            r_rdac_p_msb <= {1'b0,1'b0,1'b0,1'b0,1'b0,1'b1,{(
                NR_MSB-6){1'b0}}};
            r_rdac_p_lsb <= {{(NR_LSB-1){1'b0}},1'b1};
        end
        else if(~r_enable_3)
        begin
            r_rdac_p_msb <= {1'b0,1'b0,1'b0,1'b0,1'b0,1'b1,{(
                NR_MSB-6){1'b0}}};
            r_rdac_p_lsb <= {{(NR_LSB-1){1'b0}},1'b1};
```

```
    end
    else
    begin
        r_rdac_p_msb <= (1'b1 << r_count_p_msb);
        r_rdac_p_lsb <= (1'b1 << r_count_p_lsb);
    end
end

always @ (posedge i_clk or negedge i_rst)
begin
    if(~i_rst)
    begin
        r_rdac_n_msb <= {1'b0,1'b0,1'b0,1'b0,1'b0,1'b1,{(
            NR_MSB-6){1'b0}}};
        r_rdac_n_lsb <= {{(NR_LSB-1){1'b0}},1'b1};
    end
    else if(~r_enable_3)
    begin
        r_rdac_n_msb <= {1'b0,1'b0,1'b0,1'b0,1'b0,1'b1,{(
            NR_MSB-6){1'b0}}};
        r_rdac_n_lsb <= {{(NR_LSB-1){1'b0}},1'b1};
    end
    else
    begin
        r_rdac_n_msb <= (1'b1 << r_count_n_msb);
        r_rdac_n_lsb <= (1'b1 << r_count_n_lsb);
    end
end

assign o_rdac_p_msb = r_rdac_p_msb;
assign o_rdac_p_lsb = r_rdac_p_lsb;
assign o_rdac_n_msb = r_rdac_n_msb;
assign o_rdac_n_lsb = r_rdac_n_lsb;

// Debug port assignments
assign o_settle_time = settle_timer;
assign o_stop_time = stop_timer;
assign o_dac_select_done = dac_select_done;
assign o_dac_select = dac_select;

endmodule
```

C

RTL CODE FOR TOP MODULE AND SPI BLOCK

```
module top_dtc_control(
i_clk,
i_rst,

// input signals from analog blocks
i_cmprtr_sign,  // comparator decision
i_select,  // EN signal

// Dynamic control signals to analog blocks
o_upper_dtc_ctrl,
o_lower_dtc_ctrl,
o_rdac_p_msb,
o_rdac_p_lsb,
o_rdac_n_msb,
o_rdac_n_lsb,

// SPI Interface
i_spi_clk,
i_mosi,
i_ss_n,
o_miso,

// Static configuration signals to analog blocks
o_en_delay_select,
o_pulse_width,
```

```
o_cspd_resistor,
o_ota_gm,
o_lpf_r,
o_vco_en,
o_vco_pvt,
o_vco_acq,
o_vco_tracking
);
```

C

```
parameter FRAC_WIDTH = 14;
parameter KDTC_WIDTH = 10;
parameter GAIN_CORR_WIDTH = 14;
parameter CODE_WIDTH = 9;
parameter FIFO_DEPTH = 8;
parameter FIFO_PTR_WIDTH = $clog2(FIFO_DEPTH);

parameter NR_MSB = 32;
parameter NR_LSB = 32;
parameter TIMER_WIDTH = 16;
parameter COUNTER_WIDTH = 16;

parameter NR_THERM_BITS_UPPER = 64;
parameter NR_BIN_BITS_UPPER = $clog2(NR_THERM_BITS_UPPER);

parameter NR_THERM_BITS_LOWER = 8;
parameter NR_BIN_BITS_LOWER = $clog2(NR_THERM_BITS_LOWER);


input i_clk;
input i_rst;

input i_cmprtr_sign;
input i_select; // MUX select pin for DTC phase control

output [NR_THERM_BITS_UPPER - 1 : 0] o_upper_dtc_ctrl;
output [NR_THERM_BITS_LOWER - 1 : 0] o_lower_dtc_ctrl;

output [NR_MSB-1 : 0] o_rdac_p_msb;
output [NR_LSB-1 : 0] o_rdac_p_lsb;
output [NR_MSB-1 : 0] o_rdac_n_msb;
output [NR_LSB-1 : 0] o_rdac_n_lsb;

input i_spi_clk;
```

```
input i_mosi;
input i_ss_n;
output o_miso;

// Configuration registers
output [4:0] o_en_delay_select;
output [2:0] o_pulse_width;
output [3:0] o_cspd_resistor;
output [9:0] o_ota_gm;
output [7:0] o_lpf_r;
output o_vco_en;
output [6:0] o_vco_pvt;
output [4:0] o_vco_acq;
output [1:0] o_vco_tracking;

// Registers and wires
wire [CODE_WIDTH - 1 : 0] w_dtc_code;

// wire [NR_BIN_BITS_UPPER -1 :0] w_upper_dtc_code;
// wire [NR_BIN_BITS_LOWER -1 :0] w_lower_dtc_code;

wire [NR_THERM_BITS_UPPER - 1 : 0] w_upper_dtc_ctrl;
wire [NR_THERM_BITS_LOWER - 1 : 0] w_lower_dtc_ctrl;


wire [FRAC_WIDTH-1 : 0] w_frac_shift;
wire [KDTC_WIDTH-1 : 0] w_kdtc;
wire w_flip_sign;
wire [FIFO_PTR_WIDTH-1 : 0] w_fifo_delay;
wire [7 : 0] lambda;
wire [7 : 0] mu;

wire w_foreground_calib_en;
wire [TIMER_WIDTH-1 : 0] w_settle_time;
wire [TIMER_WIDTH-1 : 0] w_stop_time;
wire [COUNTER_WIDTH-1 : 0] w_acc_full;
wire [TIMER_WIDTH-1 : 0] w_settle_time_debug;
wire [TIMER_WIDTH-1 : 0] w_stop_time_debug;
wire w_dac_select_done;
wire w_dac_select;

// Selection between integer-N or fractional-N mode (
   Synchronization FFs also added)
reg r_int_or_frac;
reg r_int_or_frac_2;
```

```
reg r_int_or_frac_3;
wire w_int_or_frac; // From SPI : '0' is Integer-N, '1' is
    Fractional-N

always @ (posedge i_clk or negedge i_rst)
begin
    if(~i_rst)
    begin
        r_int_or_frac    <= 1'b0;
        r_int_or_frac_2  <= 1'b0;
        r_int_or_frac_3  <= 1'b0;
    end
    else
    begin
        r_int_or_frac   <= w_int_or_frac;
        r_int_or_frac_2 <= r_int_or_frac;
        r_int_or_frac_3 <= r_int_or_frac_2;
    end
end

// Synchronization FFs for background calibration enable
    signal
wire w_background_calibration_en;
reg r_background_calibration_en;
reg r_background_calibration_en_2;
reg r_background_calibration_en_3;

always @ (posedge i_clk or negedge i_rst)
begin
    if(~i_rst)
    begin
        r_background_calibration_en     <= 1'b0;
        r_background_calibration_en_2   <= 1'b0;
        r_background_calibration_en_3   <= 1'b0;
    end
    else
    begin
        r_background_calibration_en     <=
            w_background_calibration_en;
        r_background_calibration_en_2   <=
            r_background_calibration_en;
        r_background_calibration_en_3   <=
            r_background_calibration_en_2;
    end
end
```

```
dtc_code_gen #(
.FRAC_WIDTH (FRAC_WIDTH),
.KDTC_WIDTH (KDTC_WIDTH),
.GAIN_CORR_WIDTH (GAIN_CORR_WIDTH),
.CODE_WIDTH (CODE_WIDTH),
.FIFO_DEPTH (FIFO_DEPTH)
)
u_dtc_code_gen(
.i_clk (i_clk),
.i_rst (i_rst),
.i_enable (r_int_or_frac_3),
.i_background_calibration_en (r_background_calibration_en_3)
    ,
.i_frac_shift (w_frac_shift),
.i_kdtc (w_kdtc),
.i_cmprtr_sign (i_cmprtr_sign),
.i_flip_sign (w_flip_sign),
.i_fifo_delay (w_fifo_delay),
.lambda (lambda),
.mu (mu),
.o_dtc_code (w_dtc_code)
);

foreground_calibration #(
.NR_MSB (NR_MSB),
.NR_LSB (NR_LSB),
.TIMER_WIDTH (TIMER_WIDTH),
.COUNTER_WIDTH (COUNTER_WIDTH)
)
u_foreground_calibration(
.i_clk (i_clk),
.i_rst (i_rst),
.i_enable (w_foreground_calib_en),
.i_cmprtr_sign (i_cmprtr_sign),
.o_rdac_p_msb (o_rdac_p_msb),
.o_rdac_p_lsb (o_rdac_p_lsb),
.o_rdac_n_msb (o_rdac_n_msb),
.o_rdac_n_lsb (o_rdac_n_lsb),
.i_settle_time (w_settle_time),
.i_stop_time (w_stop_time),
.i_acc_full (w_acc_full),
// debug
.o_settle_time (w_settle_time_debug),
```

```
.o_stop_time (w_stop_time_debug),
.o_dac_select_done (w_dac_select_done),
.o_dac_select (w_dac_select)
);

// assign w_upper_dtc_code = w_dtc_code[CODE_WIDTH - 1 -:
    NR_BIN_BITS_UPPER];
// assign w_lower_dtc_code = w_dtc_code[NR_BIN_BITS_LOWER -
    1 : 0];


spi_reg #(
.FRAC_WIDTH (FRAC_WIDTH),
.KDTC_WIDTH (KDTC_WIDTH),
.GAIN_CORR_WIDTH (GAIN_CORR_WIDTH),
.FIFO_DEPTH (FIFO_DEPTH),
.TIMER_WIDTH (TIMER_WIDTH),
.COUNTER_WIDTH (COUNTER_WIDTH)
)
u_spi_reg(
// SPI Interface
.i_spi_clk (i_spi_clk),
.i_rst_asyn (i_rst),
.i_ss_n (i_ss_n),
.i_mosi (i_mosi),
.o_miso (o_miso),
// I/F with DTC controller
.o_int_or_frac (w_int_or_frac),
.o_background_calibration_en (w_background_calibration_en),
.o_frac_shift (w_frac_shift),
.o_kdtc (w_kdtc),
.o_flip_sign (w_flip_sign),
.o_fifo_delay (w_fifo_delay),
.lambda (lambda),
.mu (mu),
// I/F with foreground calibration
.o_foreground_calib_en (w_foreground_calib_en),
.o_settle_time (w_settle_time),
.o_stop_time (w_stop_time),
.o_acc_full (w_acc_full),
.i_settle_time_debug (w_settle_time_debug),
.i_stop_time_debug (w_stop_time_debug),
.i_dac_select_done (w_dac_select_done),
.i_dac_select (w_dac_select),
```

```
// Configuration registrers
.o_en_delay_select (o_en_delay_select),
.o_pulse_width (o_pulse_width),
.o_cspd_resistor (o_cspd_resistor),
.o_ota_gm (o_ota_gm),
.o_lpf_r (o_lpf_r),
.o_vco_en (o_vco_en),
.o_vco_pvt (o_vco_pvt),
.o_vco_acq (o_vco_acq),
.o_vco_tracking (o_vco_tracking)
);



bin2therm #(
.NR_THERM_BITS (NR_THERM_BITS_UPPER)
)
u_bin2therm_upper(
.i_clk (i_clk),
.i_rst (i_rst),
.i_binary (w_dtc_code[8:3]),
.o_therm (w_upper_dtc_ctrl)
);


bin2therm #(
.NR_THERM_BITS (NR_THERM_BITS_LOWER)
)
u_bin2therm_lower(
.i_clk (i_clk),
.i_rst (i_rst),
.i_binary (w_dtc_code[2:0]),
.o_therm (w_lower_dtc_ctrl)
);


assign o_upper_dtc_ctrl = (r_int_or_frac_3 && i_select) ?
    w_upper_dtc_ctrl : {NR_THERM_BITS_UPPER{1'b0}};
assign o_lower_dtc_ctrl = (r_int_or_frac_3 && i_select) ?
    w_lower_dtc_ctrl : {NR_THERM_BITS_LOWER{1'b0}};


endmodule
```

```
module bin2therm(
i_clk,
i_rst,
i_binary,
o_therm);

// parameters

parameter NR_THERM_BITS = 64;
parameter NR_BIN_BITS = $clog2(NR_THERM_BITS);


// I/Os

input i_clk;
input i_rst;
input [NR_BIN_BITS - 1 : 0] i_binary;
output reg [NR_THERM_BITS - 1 : 0] o_therm;


// Registers and wires
reg [NR_THERM_BITS - 1 : 0] r_therm;
integer i;

always @(*)
begin
    r_therm = 'h0;
    for(i = 0; i < NR_THERM_BITS; i = i + 1)
    begin
        if(i_binary == i)
            r_therm = {NR_THERM_BITS{1'b1}} << i;
    end
end

always @(posedge i_clk or negedge i_rst)
begin
    if(~i_rst)
    begin
        o_therm <= {NR_THERM_BITS{1'b0}};
    end
    else
    begin
        o_therm <= r_therm;
    end
end
```

```
endmodule


module spi_reg (
i_spi_clk,     // SPI clock
i_rst_asyn,    // asynchronous negative reset (same as the
    digital part)
i_ss_n,        //slave select connect to dig vss
i_mosi,        //master output slave input
o_miso,        //master input slave output

// configuration registers for DTC controller
o_int_or_frac,
o_background_calibration_en,
o_frac_shift,
o_kdtc,
o_flip_sign,
o_fifo_delay,
lambda,
mu,

// configuration registers for foreground calibration block
o_foreground_calib_en,
o_settle_time,
o_stop_time,
o_acc_full,
i_settle_time_debug,
i_stop_time_debug,
i_dac_select_done,
i_dac_select,

// Static configuration signals to analog blocks
o_en_delay_select,
o_pulse_width,
o_cspd_resistor,
o_ota_gm,
o_lpf_r,
o_vco_en,
o_vco_pvt,
o_vco_acq,
o_vco_tracking
);

parameter FRAC_WIDTH = 14;
```

```
parameter KDTC_WIDTH = 10;
parameter GAIN_CORR_WIDTH = 14;
parameter FIFO_DEPTH = 8;
parameter FIFO_PTR_WIDTH = $clog2(FIFO_DEPTH);
parameter TIMER_WIDTH = 16;
parameter COUNTER_WIDTH = 16;

parameter SPI_CMD_WRITE = 1'b0;
parameter SPI_CMD_READ = 1'b1;


// I/Os

input  wire i_spi_clk ;
input  wire i_rst_asyn ;
input  wire i_ss_n ;
input  wire i_mosi ;
output wire o_miso ;

// configuration registers for DTC controller
output o_int_or_frac;
output o_background_calibration_en;
output [FRAC_WIDTH-1 : 0] o_frac_shift;
output [KDTC_WIDTH-1 : 0] o_kdtc;
output o_flip_sign;
output [FIFO_PTR_WIDTH-1 : 0] o_fifo_delay;
output [7 : 0] lambda;
output [7 : 0] mu;

// configuration registers for foreground calibration block
output o_foreground_calib_en;
output [TIMER_WIDTH-1 : 0] o_settle_time;
output [TIMER_WIDTH-1 : 0] o_stop_time;
output [COUNTER_WIDTH-1 : 0] o_acc_full;
input [TIMER_WIDTH-1 : 0] i_settle_time_debug;
input [TIMER_WIDTH-1 : 0] i_stop_time_debug;
input i_dac_select_done;
input i_dac_select;


// Configuration registers
output [4:0] o_en_delay_select;
output [2:0] o_pulse_width;
output [3:0] o_cspd_resistor;
output [9:0] o_ota_gm;
```

```
output [7:0] o_lpf_r;
output o_vco_en;
output [6:0] o_vco_pvt;
output [4:0] o_vco_acq;
output [1:0] o_vco_tracking;

// Operation state definitions
localparam  [1:0]   STATE_IDLE  = 2'b10;    // idle
localparam  [1:0]   STATE_CMD   = 2'b00;    // command input
localparam  [1:0]   STATE_DATA  = 2'b01;    // data access


//
   ------------------------------------------------------------------

// Internal signal declarations
//
   ------------------------------------------------------------------


reg   [7:0]     spcr;               // SPI command register,
   including r/w and address
reg   [7:0]     spdw;               // data write shift
   register
reg   [7:0]     spdr;               // data read shift
   register
reg   [3:0]     counter;            // clock positive edge
   counter
wire  [1:0]     state;              // SPI operating state
wire            wr_en;              // write enable
wire            rd_en;              // read enable

//
   ////////////////////////////////////////////////////////////

// Registers used for DTC controller
reg [7:0] r_frac_shift_0;
reg [7:0] r_frac_shift_1;
reg [7:0] r_kdtc_0;
reg [7:0] r_kdtc_1;
reg [7:0] r_flip_sign;
reg [7:0] r_fifo_delay;
reg [7:0] r_lambda;
reg [7:0] r_mu;
reg [7:0] r_int_or_frac;
```

```
reg [7:0] r_background_calibration_en;

// Registers used for Foreground calibration
reg [7:0] r_foreground_calib_en;
reg [7:0] r_settle_time_0;
reg [7:0] r_settle_time_1;
reg [7:0] r_stop_time_0;
reg [7:0] r_stop_time_1;
reg [7:0] r_acc_full_0;
reg [7:0] r_acc_full_1;

// Registers used for configuring analog blocks
reg [7:0] r_en_delay_select;
reg [7:0] r_pulse_width;
reg [7:0] r_cspd_resistor;
reg [7:0] r_ota_gm_0;
reg [7:0] r_ota_gm_1;
reg [7:0] r_lpf_r;
reg [7:0] r_vco_en;
reg [7:0] r_vco_pvt;
reg [7:0] r_vco_acq;
reg [7:0] r_vco_tracking;


//
   ------------------------------------------------------------------

// Main codes
//
   ------------------------------------------------------------------

//
   ------------------------------------------------------------------

// Control signals generation
//
   ------------------------------------------------------------------

// Clock positive edge counter

always @ (posedge i_spi_clk, negedge i_rst_asyn)
begin
    if (!i_rst_asyn)
        counter <= 4'h0;
    else if (!i_ss_n)
```

```
        counter <= counter + 4'h1;
end


// SPI command shift in
always @ (posedge i_spi_clk, negedge i_rst_asyn)
begin
    if (!i_rst_asyn)
        spcr <= 8'h00;
    else if (state == STATE_CMD)
        spcr <= {spcr[6:0], i_mosi};
end


// SPI operating state and write/read enable signals
assign state = {i_ss_n, counter[3]};
assign wr_en = (state == STATE_DATA) && (spcr[7] ==
   SPI_CMD_WRITE);
assign rd_en = (state == STATE_DATA) && (spcr[7] ==
   SPI_CMD_READ );

//
   ------------------------------------------------------------------------

// SPI data write process
//
   ------------------------------------------------------------------------

// SPI write data shift in
always @ (posedge i_spi_clk, negedge i_rst_asyn)
begin
    if (!i_rst_asyn)
        spdw <= 8'h00;
    else if (wr_en)
        spdw <= {spdw[6:0], i_mosi};
end


always @ (posedge i_spi_clk, negedge i_rst_asyn)
begin
        if (!i_rst_asyn) begin

                r_frac_shift_0          <= 'h5C;
                r_frac_shift_1          <= 'h3F;
                r_kdtc_0                <= 'h8F;
```

C

```
                r_kdtc_1                    <= 'h02;
                r_flip_sign                 <= 'h00;
                r_fifo_delay                <= 'h04;
                r_lambda                    <= 'h08;
                r_mu                        <= 'h0D;
                r_int_or_frac               <= 'h00;
                r_background_calibration_en      <= 'h00;


                r_settle_time_0             <= 'hFF;
                r_settle_time_1             <= 'h03;
                r_stop_time_0               <= 'hFF;
                r_stop_time_1               <= 'hFF;
                r_acc_full_0                <= 'h3F;
                r_acc_full_1                <= 'h00;
                r_foreground_calib_en       <= 'h00;

                r_en_delay_select           <= 'h17;
                r_pulse_width               <= 'h05;
                r_cspd_resistor             <= 'h00;
                r_ota_gm_0                  <= 'hF7;
                r_ota_gm_1                  <= 'h01;
                r_lpf_r                     <= 'hF0;
                r_vco_en                    <= 'h01;
                r_vco_pvt                   <= 'h60;
                r_vco_acq                   <= 'h10;
                r_vco_tracking              <= 'h01;

        end
    else if (wr_en && counter == 4'hF) begin
            case (spcr[6:0])

                'h0     :       r_frac_shift_0[7:0]
                            <= {spdw[6:0], i_mosi};
                'h1     :       r_frac_shift_1[7:0]
                            <= {spdw[6:0], i_mosi};
                'h2     :       r_kdtc_0[7:0]
                              <= {spdw[6:0], i_mosi};
                'h3     :       r_kdtc_1[7:0]
                              <= {spdw[6:0], i_mosi};
                'h4     :       r_flip_sign[7:0]
                            <= {spdw[6:0], i_mosi};
                'h5     :       r_fifo_delay[7:0]
                           <= {spdw[6:0], i_mosi};
                'h6     :       r_lambda[7:0]
```

```
                    <= {spdw[6:0], i_mosi};
'h7     :         r_mu[7:0]
                        <= {spdw[6:0],
   i_mosi};
'h8     :         r_int_or_frac[7:0]
              <= {spdw[6:0], i_mosi};
'h9     :
   r_background_calibration_en[7:0]  <=
   {spdw[6:0], i_mosi};

'hA     :         r_settle_time_0[7:0]
            <= {spdw[6:0], i_mosi};
'hB     :         r_settle_time_1[7:0]
            <= {spdw[6:0], i_mosi};
'hC     :         r_stop_time_0[7:0]
              <= {spdw[6:0], i_mosi};
'hD     :         r_stop_time_1[7:0]
            <= {spdw[6:0], i_mosi};
'hE     :         r_acc_full_0[7:0]
             <= {spdw[6:0], i_mosi};
'hF     :         r_acc_full_1[7:0]
             <= {spdw[6:0], i_mosi};
'h10    :         r_foreground_calib_en
   [7:0]  <= {spdw[6:0], i_mosi};

'h11    :         r_en_delay_select[7:0]
          <= {spdw[6:0], i_mosi};
'h12    :         r_pulse_width[7:0]
              <= {spdw[6:0], i_mosi};
'h13    :         r_cspd_resistor[7:0]
            <= {spdw[6:0], i_mosi};
'h14    :         r_ota_gm_0[7:0]
                 <= {spdw[6:0], i_mosi};
'h15    :         r_ota_gm_1[7:0]
                 <= {spdw[6:0], i_mosi};
'h16    :         r_lpf_r[7:0]
                    <= {spdw[6:0], i_mosi
   };
'h17    :         r_vco_en[7:0]
                   <= {spdw[6:0], i_mosi};
'h18    :         r_vco_pvt[7:0]
                  <= {spdw[6:0], i_mosi};
'h19    :         r_vco_acq[7:0]
                  <= {spdw[6:0], i_mosi};
'h1A    :         r_vco_tracking[7:0]
```

C

```
                                    <= {spdw[6:0], i_mosi};

                endcase
        end
end

always @ (negedge i_spi_clk, negedge i_rst_asyn)
begin
        if (!i_rst_asyn)
        spdr <= 8'h00;

    else if (rd_en && (counter == 4'h8)) begin
                case (spcr[6:0])
                'h0     :       spdr  <=  r_frac_shift_0
                   [7:0];
                'h1     :       spdr  <=  r_frac_shift_1
                   [7:0];
                'h2     :       spdr  <=  r_kdtc_0[7:0];
                'h3     :       spdr  <=  r_kdtc_1[7:0];
                'h4     :       spdr  <=  r_flip_sign
                   [7:0];
                'h5     :       spdr  <=  r_fifo_delay
                   [7:0];
                'h6     :       spdr  <=  r_lambda[7:0];
                'h7     :       spdr  <=  r_mu[7:0];
                'h8     :       spdr  <=  r_int_or_frac
                   [7:0];
                'h9     :       spdr  <=
                   r_background_calibration_en[7:0];

                'hA     :       spdr  <=  r_settle_time_0
                   [7:0];
                'hB     :       spdr  <=  r_settle_time_1
                   [7:0];
                'hC     :       spdr  <=  r_stop_time_0
                   [7:0];
                'hD     :       spdr  <=  r_stop_time_1
                   [7:0];
                'hE     :       spdr  <=  r_acc_full_0
                   [7:0];
                'hF     :       spdr  <=  r_acc_full_1
                   [7:0];
                'h10    :       spdr  <=
                   r_foreground_calib_en[7:0];
```

```
                'h11    :       spdr <=
                    r_en_delay_select[7:0];
                'h12    :       spdr <= r_pulse_width
                    [7:0];
                'h13    :       spdr <= r_cspd_resistor
                    [7:0];
                'h14    :       spdr <= r_ota_gm_0[7:0];
                'h15    :       spdr <= r_ota_gm_1[7:0];
                'h16    :       spdr <= r_lpf_r[7:0];
                'h17    :       spdr <= r_vco_en[7:0];
                'h18    :       spdr <= r_vco_pvt[7:0];
                'h19    :       spdr <= r_vco_acq[7:0];
                'h1A    :       spdr <= r_vco_tracking
                    [7:0];

                'h30    :       spdr <=
                    i_settle_time_debug[7:0];
                'h31    :       spdr <=
                    i_settle_time_debug[15:8];
                'h32    :       spdr <=
                    i_stop_time_debug[7:0];
                'h33    :       spdr <=
                    i_stop_time_debug[15:8];
                'h34    :       spdr <= {7'h0,
                    i_dac_select_done};
                'h35    :       spdr <= {7'h0,
                    i_dac_select};

                default :       spdr <= 8'h00;
                endcase
        end
        else
                spdr <= spdr << 1;
end

assign o_miso =spdr[7];

assign o_frac_shift = {r_frac_shift_1[5:0],r_frac_shift_0};
assign o_kdtc = {r_kdtc_1[1:0],r_kdtc_0};
assign o_flip_sign = r_flip_sign[0];
assign o_fifo_delay = {r_fifo_delay[2:0]};
assign lambda = r_lambda[7:0];
assign mu = r_mu[7:0];
assign o_int_or_frac = r_int_or_frac[0];
assign o_background_calibration_en =
```

```
    r_background_calibration_en[0];

assign o_foreground_calib_en = r_foreground_calib_en[0];
assign o_settle_time = {r_settle_time_1,r_settle_time_0};
assign o_stop_time = {r_stop_time_1,r_stop_time_0};
assign o_acc_full = {r_acc_full_1,r_acc_full_0};

assign o_en_delay_select = r_en_delay_select[4:0];
assign o_pulse_width = r_pulse_width[2:0];
assign o_cspd_resistor = r_cspd_resistor[3:0];
assign o_ota_gm = {r_ota_gm_1[1:0],r_ota_gm_0};
assign o_lpf_r = r_lpf_r;
assign o_vco_en = r_vco_en[0];
assign o_vco_pvt = r_vco_pvt[6:0];
assign o_vco_acq = r_vco_acq[4:0];
assign o_vco_tracking = r_vco_tracking[1:0];

endmodule
```

D

VERILOGA MODEL FOR THE DTC

```
// VerilogA for a DTC embedded within a subsampling PLL

`include "constants.vams"
`include "disciplines.vams"

(* ignore_hidden_state *) module dtc_control_calibrate(
    ref_clk, vsp,vsn, dly_clk,dly_cal_clk,dly_clk_quant,debug
    ,acc_err_debug,sample_clk,dly_clk_err,kdtc);

input ref_clk;
input vsp;
input vsn;
output dly_clk;
output dly_cal_clk;
output dly_clk_quant;
output debug;
output acc_err_debug;
output sample_clk;
output dly_clk_err;
output kdtc;

electrical ref_clk;
electrical vsp;
electrical vsn;
electrical dly_clk;
```

```
electrical dly_cal_clk;
electrical dly_clk_quant;
electrical debug; // debug pin for acc_frac
electrical acc_err_debug; // debug pin for acc_error
electrical sample_clk;
electrical dly_clk_err;
electrical kdtc;

parameter real ttol = 0.01f;
parameter real tr = 20p;
parameter real tf = 20p;

// Freq related params
parameter real f_ref = 10e6;
parameter real t_ref = 100n;
parameter real f_vco = 32e6;
parameter real t_vco = 31.25n;
parameter real t_vco_max = 50n;
parameter real pulse_width = 5n;
parameter real frac_n = 0.2;
parameter real shift = 0.8; // 1 - frac_n
parameter real n_levels = 1024; // number of levels in
    quantizer

real acc_frac; // accumulated fraction
real delay_time;
real delay_time_cal;
real delay_time_err;
real delay_quant; // delay after quantization + INL
real pure_delay_quant; // delay after quantization
real sample_delay;
integer pulse;
integer pulse_err;
integer pulse_cal; // Pulse with INL
integer pulse_quant; // Pulse with quantized delay
real lsb; // LSB of quantizer
integer i = 0, j = 0;
real quant_thresh [1024-2:0];
real error;
real acc_error; // accumulate the quantization error

real dtc_gain_err;
integer k_dtc;
integer sample;
real lms_acc;
```

```
// real lms_acc_update;
real iir_out;
real iir_in;
real lambda;
real mu;
// real alpha;
integer sign;

// calibration loop BW control
integer counter_max;
integer counter;

analog
begin
        @(initial_step)
        begin
                acc_frac = 0.0;
                delay_time = 0.0;
                delay_time_err = 0.0;
                delay_time_cal = 0.0;
                delay_quant = 0.0;
                pure_delay_quant = 0.0;
                acc_error = 0.0;
        lsb = t_vco_max/n_levels;
                dtc_gain_err = 0.1;
                k_dtc = (t_vco/lsb)*(1+dtc_gain_err);
                lms_acc = 1;
                iir_out = 0;
                iir_in = 0;
                lambda = 5e-3;
                mu = 1e-4;
                sign = 0;
                counter_max = 6000;  // 10ns_clk*6000 = 60us
                counter = 0;

        // create quantizer thresholds (n_levels - 1
           thresholds)
        for(i=0; i<(n_levels-1); i=i+1)
                begin
                        quant_thresh[i] = (i+1)*lsb - lsb/2;
                end
        end

        @(cross(V(ref_clk)-0.5,+1,ttol)) begin
```

D

```
// BW control
                counter = counter + 1;
                if(counter > counter_max)
                        mu = 2.5e-5;
// BW control



                delay_time = acc_frac*t_vco;
                k_dtc = (t_vco/lsb)*(1+dtc_gain_err)*lms_acc
                    ;
                delay_time_err = acc_frac*t_vco*(1 +
                    dtc_gain_err);
                lms_acc = lms_acc + mu*iir_out;

                delay_time_cal = acc_frac*t_vco*(1 +
                    dtc_gain_err)*lms_acc;

                // Quantizing true delay
                delay_quant = 0.0;
                pure_delay_quant = 0.0;
                for(j=0; j < (n_levels-1); j=j+1)
                begin
                        if(delay_time >= quant_thresh[j])
                        begin
                                pure_delay_quant = (j+1)*lsb
                                    ;
                                delay_quant = (j+1)*lsb*(1+
                                    dtc_gain_err);
                        end
                end

                error = (pure_delay_quant - delay_time)/lsb;
                     // quant error normalized to lsb
                acc_error = acc_error + error; //
                    accumulating quant error in terms of lsb
                if(acc_error >= 1)
                begin
                        acc_error = acc_error - 1;
                        delay_quant = delay_quant - lsb*(1+
                            dtc_gain_err);
                end
                if(acc_error <= -1)
                begin
                        acc_error = acc_error + 1;
```

```
                        delay_quant = delay_quant + lsb*(1+
                            dtc_gain_err);
                end


                acc_frac = acc_frac + shift;
                if(acc_frac >= 1)
                        acc_frac = acc_frac - 1; //
                            effectively modulo 1 addition
                pulse = 1;
                pulse_err = 1;
        pulse_cal = 1;
                pulse_quant = 1;
        end

        @(cross(V(dly_clk)-0.5,+1,ttol)) begin
                pulse = 0;
                delay_time = pulse_width; // generating a
                    pulse
                sample = 1;
                sample_delay = t_ref/4;
        end

        @(cross(V(dly_clk_err)-0.5,+1,ttol)) begin
                pulse_err = 0;
                delay_time_err = pulse_width; // generating
                    a pulse
        end

        @(cross(V(dly_cal_clk)-0.5,+1,ttol)) begin
                pulse_cal = 0;
                delay_time_cal = pulse_width; // generating
                    a pulse
        end

        @(cross(V(dly_clk_quant)-0.5,+1,ttol)) begin
                pulse_quant = 0;
                delay_quant = pulse_width;
        end

// Sampling clks for Re-sampling PLL

        @(cross(V(sample_clk)-0.5,+1,ttol)) begin
                sample = 0;
                sample_delay = t_ref/4;
```

```
        end
//////////////////////////////////////////

// logic for calibration
        @(cross(V(sample_clk)-0.5,-1,ttol)) begin     //
            compare PD output on falling edge of re-sampling
            clk
                if(V(vsn) > V(vsp))
                        sign = +1;
                else
                        sign = -1;
                iir_in = sign*acc_frac;
                iir_out = (1-lambda)*iir_out + lambda*iir_in
                    ;
        end
//////////////////////////////////////////


        V(dly_clk) <+ transition(pulse,delay_time,tr,tf);
        V(dly_clk_err) <+ transition(pulse_err,
            delay_time_err,tr,tf);
    V(dly_cal_clk) <+ transition(pulse_cal,delay_time_cal,tr
        ,tf);
        V(dly_clk_quant) <+ transition(pulse_quant,
            delay_quant,tr,tf);

        V(sample_clk) <+ transition(sample,sample_delay,tr,
            tf);

    V(debug) <+ transition(sign,0,tr,tf);
    V(acc_err_debug) <+ transition(lms_acc,0,tr,tf);
        V(kdtc) <+ transition(k_dtc,0,tr,tf);

end


endmodule
```