Journal of Electronic Testing (2024) 40:31–43
https://doi.org/10.1007/s10836-023-06095-2

General Fault and Soft-Error Tolerant Phase-Locked Loop by Enhanced TMR using A Synchronization-before-Voting Scheme

Shun-Hua Yang[1] · **Shi-Yu Huang**[1]

Received: 23 August 2023 / Accepted: 11 December 2023 / Published online: 9 January 2024

Abstract
A Phase-Locked Loop (PLL) is indispensable in producing high-speed on-chip clock signals in an IC. For safety–critical applications, fault and soft-error tolerance are often desirable. However, how to achieve this goal for a PLL is still a challenge. In this paper, we address this challenge with a TMR-based FET-PLL design. Our unique contribution is a "synchronization-before-voting" scheme so that the fault and soft-error tolerance and the jitter performance can be maintained at the same time. Post-layout simulation using a 90 nm CMOS process demonstrates that our PLL can indeed withstand the attack of online faults as well as soft errors without suffering from significant jitter performance loss.

Keywords Delay-locked loop · Fault tolerant · Soft-error tolerant · Majority voting · Mutual synchronization

1 Introduction

A Phase-Locked-Loop (PLL) is often used to produce a high-speed on-chip clock signal in an IC. It can be made by analog/mixed-signal circuits [7, 15] or by all-digital circuits [2, 6, 13]. Furthermore, a compiler is available that fully automates the design process of a synthesizable PLL using only standard cells [18]. With the proliferation of safety–critical applications for automotive, biomedical, and aerospace electronics, it is often desirable that an IC should be able to withstand the attack of radiation or fault-induced performance hazards. Here, the radiation means high-energy particle bombardment on an IC. It could induce a short-time transient glitch at the bombarded site and is often referred to as **Single-Event-Transient (SET)**. In this paper, we use SET and "**soft error**" interchangeably without distinction. In addition to the radiation effect, an IC designed to last for more than 10 years may encounter sudden malfunction in the field due to reliability degradation of some worsening latent faults.

In the face of these safety and reliability threats, **F**ault and soft-**E**rror **T**olerance (**FET**) techniques have become more important than ever before. Various FET techniques exist for the logic and memory components in an IC. In this work, we will focus on the FET techniques for the PLLs.

For PLLs, radiation tolerance has been addressed in the literature [1, 12]. There have existed component-specific **Radiation-Hardening-By-Design (RHBD)** techniques for traditional analog PLLs [3, 8–10, 14, 17]. An analog PLL often consists of 5 major components – (1) a Voltage-Controlled Oscillator (VCO), (2) a Frequency-Divider, (3) a Charge-Pump circuit, (4) a passive Loop Filter, and (5) a Controller. Some of these components are more sensitive to radiation than others. For example, the Charge-Pump circuit is considered to be the most vulnerable part [12]. Therefore, several techniques have been proposed to protect it from radiation-induced malfunction [9, 10]. For the other parts, various radiation-hardening techniques can be applied as well to achieve the highest radiation tolerance. For example, the VCO using the Differential Cascade-Voltage Switch Logic (DCVSL) on stacked SOI (Silicon-On-Insulator) technology can mitigate the SET events [3, 14]. For the frequency divider, the usage of the Radiation-Hardened Flip-Flops boosts the radiation tolerance as well [3, 17].

However, these component-specific techniques still have two problems. First, the radiation-induced soft errors are only mitigated, not eradicated. Therefore, radiation tolerance is not guaranteed. Second, it may go through a sense-and-recovery process under a radiation attack. During the recovering process, the PLL may exhibit temporary instability with higher jitter displacement at the PLL's output clock signal.

Responsible Editor: X. Li

✉ Shi-Yu Huang
syhuang@ee.nthu.edu.tw

[1] EE Dept, National Tsing Hua University, Hsinchu, Taiwan

To address these problems, **TMR (Triple-Module-Redundancy)** based techniques have been exploited in [11, 16], providing another layer of defense. However, they only apply it to the VCO, not the complete PLL instance.

In this work, we propose a TMR-based architecture-level Fault and Soft-Error (FET) PLL design, with the following unique technical contributions. As well known, TMR can provide stronger and continuous protection from radiation-induced soft errors and single-module faults. Still, applying the TMR techniques to render a FET-PLL design is not trivial. Some nasty timing issues need to be resolved to keep the resulting jitter performance in check.

(1) First, our TMR module level is different from that in [11]. In [11], highly customized 3 VCOs are protected by the TMR. In our design, we take 3 **independently running PLL primitives** to form our TMR, and thus the scope of our protection is more complete. Any PLL primitive design can be employed in our architecture to enjoy fault and soft-error tolerance.
(2) Second, we developed a **synchronization-before-voting scheme** to ensure that the clock signals produced by the 3 TMR modules are synchronized first before being voted to produce the final output clock signal. This scheme is essential in our FET-PLL design.
(3) Third, we solved the **delay mismatch problem within a voter circuit** via a timing correction scheme we developed earlier for FET Delay-Locked Loops (DLLs) [19, 20]. This again helps prevent the degradation of the jitter performance by several dozens of picoseconds.
(4) Last, but not least, by post-layout simulation, we will demonstrate that our PLL can continue to function correctly in the presence of **online suddenly activated faults**, as long as the faults affect only one of the 3 modules. To our knowledge, this work is the first to tolerate this type of online fault.

The rest of this paper is organized as follows. In Section 2, we provide the preliminaries of a PLL. In Section 3, we provide the architecture of our Fault and Soft-Error Tolerant (FET) PLL. In Section 4, we discussed the detailed circuit. In Section 5, we present the post-layout simulation of our FET-PLL implemented in a 90 nm CMOS process to demonstrate its features and performances. In Section 6, we conclude.

2 Preliminaries

In this Section, we briefly review the basic architecture and functionality of a PLL. In general, the proposed fault and soft-error tolerant architecture be applied to either analog or digital PLLs. In the following, we use a digital PLL as an example.

Figure 1 shows a typical block diagram for a digital PLL. It consists of 4 major circuit blocks, (1) a Digitally Controlled Oscillator (DCO), (2) a Phase Detector, (3) a Low-Pass Filter, (4) a Frequency Divider, and (4) a Controller.

A PLL usually takes a low- to medium-frequency clock signal, *clk_ref*, and then multiplies its frequency by *N* times to become a higher-frequency output clock signal, *clk_out*. For example, when the frequency of *clk_ref* is 10 MHz and *N* is 100, then the frequency of *clk_out* will be 10 MHz × 100, which will be 1000 MHz or 1 GHz. The frequency of the Frequency Divider's output, *clk_div*, is the frequency of *clk_out* divided by *N* times and thus its frequency is the same as that of *clk_ref*.

The operation of a digital PLL can be divided into 3 stages after power-up, namely (1) frequency acquisition, (2) phase locking, and (3) phase tracking. The frequency acquisition is to quickly find a proper control code for the DCO so that its output signal (i.e., *clk_out*) will have a frequency closest to the desired frequency. Then the PLL starts the phase locking, which is to conduct a dynamic process in which the control code of the DCO is modified in a way to minimize the "phase difference" between *clk_div* and *clk_ref*. Once this is complete, the frequency of *clk_out* will remain steady and very close to the target frequency. Now, the PLL is ready to provide its service in that its output clock signal is stable and can be used by the system. However, in response to potential environmental changes (e.g. temperature increase in the chip), the PLL remains vigilant and will tune the control code of the DCO slightly whenever necessary (to step up or slow down the DCO's oscillating frequency) to maintain the phase-locking condition. This is a stage called phase-tracking.

The above phase-locking and phase-tracking operations are made possible by the feedback loop built into the PLL. This feedback loop consists of the following elements: (1) The frequency of *clk_out* is first divided by *N* times to become *clk_div*. (2) The phase of *clk_div* is then compared with that of *clk_ref* by the Phase Detector, producing a binary *lead/lad* signal that changes in time to indicate if the phase of *clk_div* is leading or lagging that of *clk_ref*.

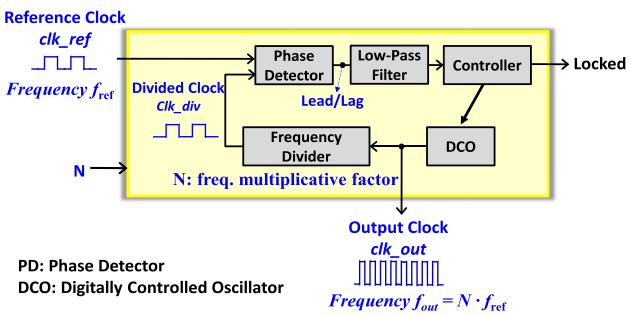

Fig. 1 General digital PLL architecture

(3) The controller of the PLL observes the sequence of *lead/lag* signals produced over time and decides if the DCO needs to speed up or slow down to prevent the phase difference between *clk_div* and *clk_ref* from drifting away.

The most important performance metrics associated with a PLL are the "frequency error" and the "jitter performance". The frequency error is the difference between the average frequency of the output clock signal, *clk_out*, and its target frequency – which is *N* times the frequency of the reference clock signal. The smaller this frequency error, the better the PLL design. Generally, speaking, when a PLL is locked (with the output signal "locked" raised to '1'), the frequency error is usually very small. Otherwise, it would have experienced some "out-of-lock" situations due to the presence of manufacturing faults or the attack of soft errors. On the other hand, the "clock jitter" (or simply called jitter in the sequel) is a more time-varying phenomenon. Even though a PLL produces a very accurate average frequency, its clock cycle time samples from one to another exhibit variation. Therefore, we have the clock jitter definition as the following:

(Definition 1): The **clock jitter** refers to the statistical "deviation of the clock cycle time away from its ideal value". Usually, it is a statistical profile as illustrated in Fig. 2 and characterized by its Root-Mean-Square jitter value (or **RMS jitter** for short) denoting its nominal value, and its peak-to-peak jitter value (or **peak-to-peak jitter** for short) denoting its widest span over a window of time.

Figure 2 shows the profile of 2000 clock cycle time samples taken from a 1 GHz clock signal produced by a PLL. The clock cycle time is within the range of [990, 1010], with an average of 1000.06 ps. The peak-to-peak jitter amount is thus (1010–990) = 20 ps.

For a PLL to operate very reliably at all times, we need to maintain not just a very small frequency error but also a reasonably small peak-to-peak jitter. When a PLL is attacked by a fault or a soft error, its output clock signal may be jittery for some time. Due to its inherent self-correction ability, a PLL may eventually return to its correct and stable condition. However, during this jittery process, the maximum peak-to-peak constraint may have been violated and the logic system driven by the PLL's output clock signal could have failed due to timing violation. Therefore, we need to use FET techniques to cope with the jittery situation even though it is just momentary.

3 Proposed Fault and Soft-Error PLL Design

3.1 Predicament of Naïve TMR-based PLL

The principle of Triple-Module-Redundancy (TMR) cannot be applied to PLL directly to achieve fault and soft-error tolerance. Figure 3(a) shows a naïve TMR PLL. Three identical PLL instances {PLL-1, PLL-2, PLL-3} are used to create 3 basic clock signals, namely {pclk1, pclk2, pclk3}. These 3 basic clock signals are used to drive a voter circuit to produce the final output clock signal, clk_out. Post-layout transistor-level simulation as shown in Fig. 3(b) reveals that the three clock signals {pclk1, pclk2, pclk3} indeed are locked to the same frequency but with different phases, as indicated that the rising edges of {pclk1, pclk2, pclk3} are not aligned. As a result, the final clk_out after voting is a total mess up.

It is worth mentioning that throughout this paper the voter circuit is simply a traditional majority function. That is,

$$clk\_out = pclk1 \cdot pclk2 + pclk2 \cdot pclk3 + pclk3 \cdot pclk1$$

This example leads to one important observation: To make a TMR PLL feasible, we need to synchronize the 3 basic clock signals {pclk1, pclk2, pclk3} first before they are voted. Then the voting can be effective in **masking** out a soft error or a faulty effect affecting one of the 3 basic PLL instances.

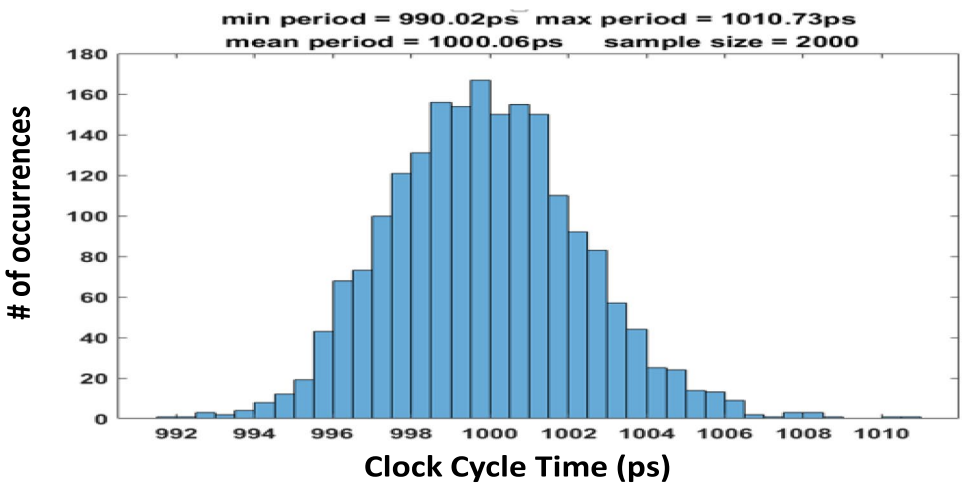

Fig. 2 The clock cycle time profile for a PLL instance producing a 1 GHz clock signal. These results are derived by direct waveform analysis

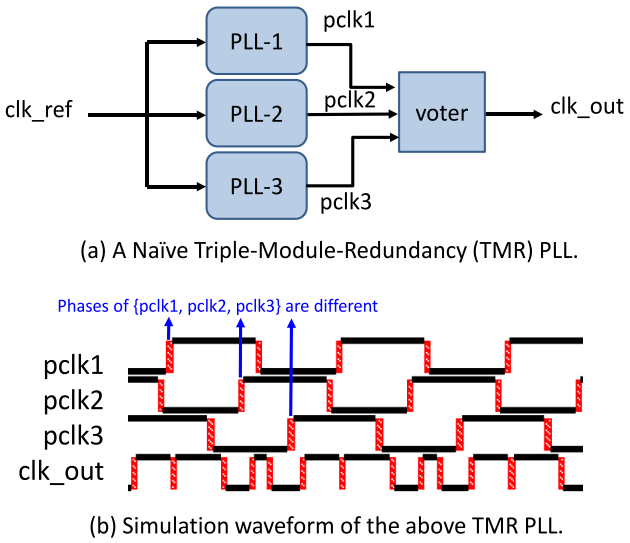

Fig. 3 Baseline Triple-Module-Redundancy (TMR) PLL

3.2 The Architecture of Proposed FET-PLL

Our **synchronization-before-voting** principle leads to a FET-PLL architecture as shown in Fig. 4(a). In addition to the 3 PLL instances, we have added 3 so-called **Mesh Delay-Locked Loop** (Mesh-DLL) instances, namely {MDLL-1, MDLL-2, MDLL-3}. The details of a Mesh-DLL will be explained later. Their joint function is to take the 3 potentially out-of-phase clock signals of the same clock frequency, {pclk1, pclk2, pclk3}, and then produce 3 clock signals, namely {dclk1, dclk2, dclk3}, with synchronized rising edges. When these 3 synchronized clock signals {dclk1, dclk2, dclk3} are voted, they exhibit a voting-in-time property as shown in Fig. 4(b) that guarantees that even

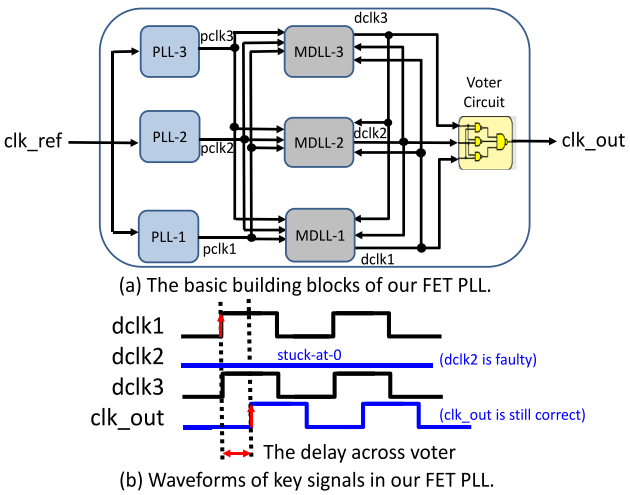

Fig. 4 Architecture of our proposed Fault-and-Soft-Error Tolerant (FET) PLL

though one of them is faulty (such as stuck-at-0 or glitching), the voter's output signal is a clean clock signal.

3.3 Overall Mesh-DLLs Operation

As mentioned above, the 3 Mesh-DLL instances jointly form a component that synchronizes the 3 clock signals {pclk1, pclk2, pclk3} produced by the 3 PLL instances {PLL-1, PLL-2, PLL-3}, while producing the 3 synchronized clock signals {dclk1, dclk2, dclk3}. Its operation takes two stages – (1) the pivot-based synchronization stage, and (2) the mutual-synchronization stage.

In the pivot-based synchronization stage, we first pick one clock signal out of {pclk1, pclk2, pclk3} as the **pivot** clock signal. Then the 3 Mesh-DLLs operate in a way to make their output signals {dclk1, dclk2, dclk3} all synchronized with that pivot clock signal. Once this is done, the final output clock, *clk_out*, is stable and ready to support system operation.

After the above initial synchronization process, the FET-PLL enters its tracking stage to continuously tune itself in response to environmental changes. The 3 Mesh-DLLs perform the **mutual synchronization** to keep {dclk1, dclk2, dclk3} mutually synchronized to one another. It is notable that with such a scheme we can achieve the Fault and Soft-Error Tolerance in a sense that when any one of {pclk1, pclk2, pclk3}is faulty, or any one of {dclk1, dclk2, dclk3} is faulty, at least two dclk signals driving the output voter is still synchronized to ensure the correct timing of *clk_out*.

3.4 Mutual Synchronization Scheme

Mutual synchronization is a key technique in this work. Its challenges can be summarized as follows:

(1) It is jointly achieved by 3 independent Mesh-DLL instances.
(2) It should be fault and soft-error tolerant and maintain its correct operation in the presence of a fault or a soft error.
(3) It needs to consider the path-delay-disparity problem that different input-to-output delays of the output voter could be different [19].

Our mutual synchronization is conducted based on the following "**3-way alignment**" principles:

Each of the 3 Mesh-DLL takes all the 3 dclk signals {dclk1, dclk2, dclk3}, and then elects the **middle signal** out of them. The middle signal is referred to as the one with the most compromised rising edge (i.e., the one with the 2nd rising edge in time). Once the middle signal is decided, the other dclk signals attempt to align to this elected middle signal. By doing so, the 3 dclk signals form an "ensemble

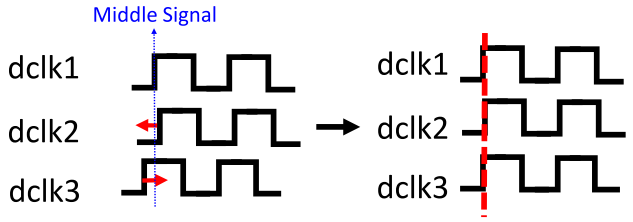

Fig. 5 The principle of our 3-way alignment during the tracking stage

group" with very small phase differences among their rising edges.

(Example 1): Consider the 3 dclk signals {dclk1, dclk2, dclk3} during the mutual synchronization stage in our FET-PLL design, as shown in Fig. 5. At a moment, dclk1 is the middle signal among them, with dclk2 lagging and dclk3 leading slightly. The 3 Mesh-DLL instances {MDLL-1, MDLL-2, MDLL-3} will recognize this situation and react independently. For example, MDLL-1 will make no change, and MDLL-2 will decrease the delay of its delay line to catch up, while MDLL-3 will increase the delay of its delay line to slow down. After the slight adjustment, the phases of 3 dclk signals {dclk1, dclk2, dclk3} will remain locked.

3.5 Adaptive Mutual Synchronization Scheme

The above 3-way alignment can operate correctly even when any of the 3 dclk signals {dclk1, dclk2, dclk3} experiences a soft-error attack momentarily. However, when one of them is faulty, e.g., stuck-at-0 fault triggered at a certain time, this primitive scheme may fail and thus we need an **adaptive version** to cope with it. In the following, we demonstrate the potential incorrect result first when there is a faulty signal among {dclk1, dclk2, dclk3} with an example.

(Example 2): Consider the 3 dclk signals {dclk1, dclk2, dclk3} during the phase tracking phase of our FET-PLL, as shown in Fig. 6. A stuck-at-1 fault is assumed to have occurred to signal dclk3. In this situation, signal dclk1 is elected as the middle signal. Then, dclk2 will try to align itself to dclk1 by moving its next few rising edges earlier (to the left) gradually. After some time, this movement will become too excessive, resulting in a situation in that dclk2 has a rising edge slightly earlier than that of dclk1 (as shown in the middle of the figure). From that moment on, dclk2 becomes the new middle signal, and dclk1 will try to align itself to dclk2 by moving its next few rising edges earlier (to the left). As time goes on, this becomes a race to the left for the two normal dclk signals. The output clock voted by these two dclk signals will have shorter and shorter clock cycle times and thus larger jitters.

This can be fixed by an **adaptive mutual synchronization scheme** with the following 3 steps.

(Step 1): We check if there are any abnormal dclk signals. The circuit for this purpose will be discussed in detail later.

(Step 2): In a situation when there is no abnormal dclk signal, we use the original 3-way alignment.

(Step 3): In a situation when there is an abnormal dclk signal, then we exclude the abnormal dclk signal. Only the 2 normal dclk signals are involved in a two-way alignment, in which each of the 2 normal dclk signals tries to align with the other.

(Example 3): Consider the 3 dclk signals {dclk1, dclk2, dclk3} during the phase tracking phase of our FET-PLL, as shown in Fig. 7. A stuck-at-1 fault is assumed to have occurred to signal dclk3. In this situation, since dclk3 has been identified as an abnormal one, the other dclk signals, i.e., dclk2 and dclk3 perform the two-way alignment. At the very beginning (i.e., snapshot 1), dclk1 is leading in time and dclk2 is lagging. Therefore, MDLL-1 will try to move dclk1's rising edges earlier (to the left), and MDLL-2 will try to move dclk2's rising edges later (to the right). Soon, we will enter snapshot 2, during which the timing relationship of dclk1 and dcl2 is reversed, with dclk1 becoming the leading and dclk2 the lagging. Again, the two dclk signals try to align with each other. Sometime later, we enter snapshot 3, and so on. *During this two-way alignment process, both normal dclk signals dance around their original timing positions and thus they do not exhibit the run-away phenomenon as demonstrated in the original 3-way mutual alignment.*

3.6 Operational Flow of Proposed FET-PLL

Figure 8 depicts the overall operational flow of our FET-PLL using adaptive mutual synchronization, divided into 3 stages – (1) individual PLL locking, (2) pivot-based synchronization, and (3) adaptive mutual synchronization.

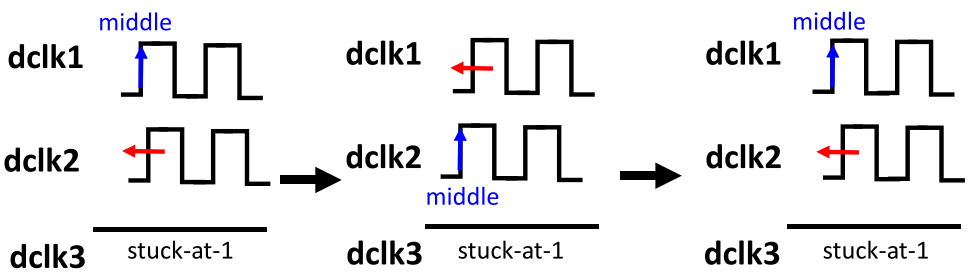

Fig. 6 The incorrect result of a 3-way alignment scheme in the presence of a faulty *dclk* signal, which is *dclk3* in this example

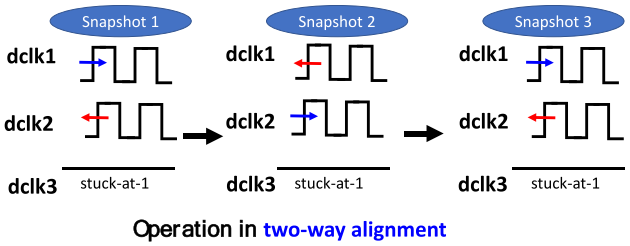

Fig. 7 The two-way alignment in our adaptive mutual synchronization scheme in the presence of a faulty *dclk* signal

(Stage 1): After the PLL is powered up, the 3 PLL instances {PLL-1, PLL-2, PLL-3} operate independently to achieve their locking. When a PLL instance is locked, its status flag "locked" will go high and its output pclk signal will have the correct frequency. Then the 3 MDLL instances {MDLL-1, MDLL-2, MDLL-3} operate independently to achieve its locking. Afterward, dclk1 is aligned with pclk1, dclk2 is aligned with pclk2, and dclk3 is aligned with pclk3, respectively. However, the phases of {dclk1, dclk2, dclk3} are not mutually aligned yet.

(Stage 2): After at least two PLL instances are locked, the 3 MDLL instances {MDLL-1, MDLL-2, MDLL-3} will conduct the pivot-based synchronization. *When this is complete, {dclk1, dclk2, dclk3} are finally mutually aligned and our FET-PLL is ready to produce a stable output clock signal.*

(Stage 3): After our FET-PLL is ready and stable, it starts the adaptive mutual synchronization to respond to environmental fluctuations so our FET-PLL can continue to maintain the locked condition.

Table 1 further outlines the status of our FET-PLL at the end of the above 3 stages. For example, after Stage 1, the 3 pclk signals {pclk1, pclk2, pclk3} become valid and **locked** to the right output frequency. After Stage 2, the 3 dclk signals {dclk1, dclk2, dclk3} are **aligned**. At the same time, the output clock clk_out becomes **ready**. After stage 3, {dclk1, dclk2, dclk3} continue to remain aligned and clk_out remain stable with the correct clock frequency and small jitter.

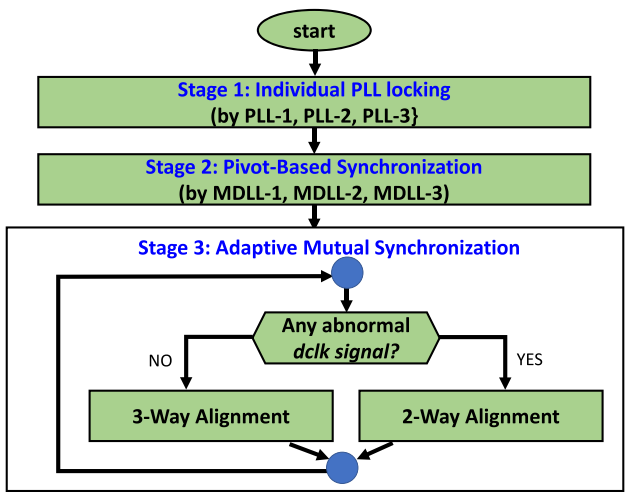

Fig. 8 The operational flow of our FET-PLL

4 Details of Our FET-PLL Design

In this section, we discuss the details of our FET-PLL design. The issues include the Mesh-DLL (MDLL) design, the robust pivot-signal selection, the timing correction to compensate for the voter circuit delay, and the nullification of the wiring effects.

4.1 The Architecture of The Mesh DLLs

In general, our FET-PLL can use any given PLL without modification. However, we need to enhance a given DLL to become our Mesh-DLL (MDLL).

Recall that a DLL contains 3 different circuit blocks – Tunable Delay Line (TDL), Phase Detector (PD), and the controller. The architecture of our Mesh-DLL is shown in Fig. 9**.** In our Mesh-DLL, we can reuse the TDL, and use 5 copies of the PD, while modifying the controller to support the aforementioned synchronization scheme (including the pivot-based synchronization and the adaptive mutual synchronization). In addition, our MDLL also includes a dummy voter circuit as will be explained later.

In this example, we consider MDLL-1, and so it produces output signal dclk1. The input signals include {pclk1, pclk2, pclk3} and {dclk2, dclk3}. Some logistic signals such as *Reset* and *clk_ref* are omitted in the figure. This MDLL circuit conducts the following basic operations:

(1) For the pivot-based synchronization, we use signal pivot_sel[1:0] to record which one of {pclk1, pclk2, pclk3} has been selected as the pivot signal. Then, that selected pivot signal is used as the "main input" driving

Table 1 The status of key signals in our FET-PLL at different stages of the entire operational flow

Stage	{pclk1, pclk2, pclk3}	{dclk1, dclk2, dclk3}	clk_out
Stage 1 Initial Locking	Locked (in frequency)	Not Valid Yet	Not Valid Yet
Stage 2 Pivot-Based Synchronization	Remain Locked	Aligned (in phase)	Ready (right freq.) (small jitter)
Stage 3 Adaptive Mutual Synchronization	Remain Locked	Remain Aligned	Remain Ready

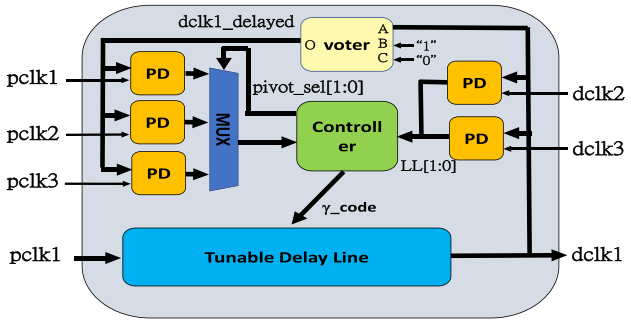

Fig. 9 Basic architecture of our MDLL-1

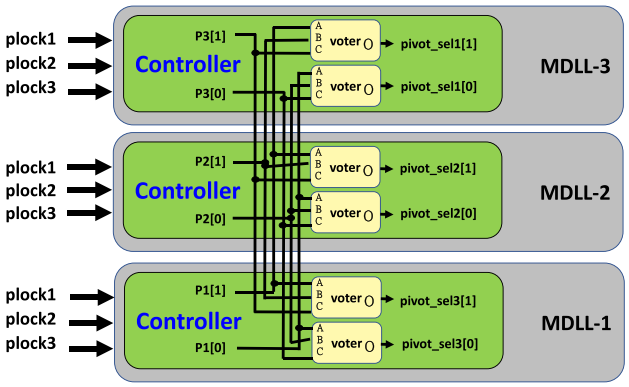

Fig. 10 Two voter circuits added in the controller of each Mesh-DLL ensure the correct "pivot signal election" for subsequent pivot-based synchronization

our MDLL operation in the sense that our MDLL tries to lock the phase of its output clock signal (i.e., dclk1 in MDLL-1) to that of this pivot signal.

(2) Let us assume that "pclk2" has been elected as the pivot signal. Then, the subsequent MDLL-1 operates as follows. The data path is mainly from pclk1 to dclk1 (since we are in MDLL-1), passing through the Tunable Delay Line (TDL). The delay of this TDL is controlled by a γ-code issued by the controller. For simplicity without losing generality, we assume a zero delay for the Voter Circuit at this moment. The goal now is to align the phase of MDLL-1's output, i.e., dclk1, with the current elected pivot signal, i.e., pclk2. At each iteration, the phase detection is conducted on signals dclk1 and pclk2, producing a binary leading-or-lagging signal to the controller. The sequence of this leading-or-lagging signal will pass through the MUX (controlled by 2-bit signal pivot_sel[1:0]) to feed the controller and then a tracking algorithm will be used by the controller to decide how to change the γ-code gradually to achieve the locking – that aligns the phases of dclk1 and the pivot signal, pclk2. The other 2 MDLLs perform the same locking, attempting to align their respective dclk signal with the pivot signal, pclk2.

(3) For the mutual synchronization, we need 2 PDs on the right-hand side of the figure to compare the phase of dclk1 with those of the other 2 dclk signals {dclk2, dclk3}. It produces 2 binary "Leading or Lagging" signals jointly denoted as LL[1:0]. Our controller will interpret this two-bit signal to decide which one of {dclk1, dclk2, dclk3}is the middle signal when necessary.

4.2 The Details of Pivot Selection

The **pivot selection** conducted by the controllers in our Mesh-DLLs is not trivial as it is to be conducted by each of {MDLL-1, MDLL-2, MDLL-3} independently while leading to a consistent result.

A PLL instance produces a flag signal, named *plock*, in our design. This flag signal when set to '1' indicates its corresponding pclk signal has been locked with the desired frequency. Otherwise, it is '0'. With the aid of 3 plock signals, we can use the following basic principle for the pivot signal selection:

In principle, we select the first pclk signal that has been locked first among the 3 PLL instances as our pivot signal. However, more than one PLL instance could complete the locking process almost at the same time. The controllers of the 3 MDLLs could elect different pivot signals. To resolve this kind of dispute, two voter circuits are added in each MDLL's controller as illustrated in Fig. 10*.* ***How the voting resolves a dispute can be proved by the following proposition.***

(Proposition 1): The above dispute-resolving scheme by adding voter circuits can guarantee "consistent" pivot-signal election results among all 3 MDLLs.

(Proof): The controller in each MDLL independently checks the 3 plock signals, i.e., {plock1, plock2, plock3}, to decide *which pclk signal is locked the first*. The results are encoded into **P1[1:0], P2[1:0], P3[1:0]**. These 2-bit signals are further distributed to each controller, where, the final pivot signal ID is produced and encoded in another 2-bit signal, **pivot_sel[1:0]**. We need to consider the following two cases when converting the {P1[1:0], P2[1:0], P3[1:0]} signals to the pivot_sel[1:0] signals, one in each MDLL-controller.

(Case 1): In this case, there is consensus among at least 2 P signals. For example, P1[1:0] = 2'b01, P2[1:0] = 2'b01, P3[1:0] = 2'b00. It follows that both P1 and P2 agree on 2'b01, implying that PLL-1 is the first to lock. This consensus will dominate the voting process and thus "pclk1" will be elected as the pivot signal, regardless of the other rebellious P3 signal.

(Case 2): In this case, all 3 P signals differ. For example, P1[1:0] = 2'b01, P2[1:0] = 2'b10, P3[1:0] = 2'b11. In other words, P1 elects pclk1, P2 elects pclk2, and P3 elects pclk3, as the pivot signal. After bitwise voting of {P1[1:0], P2[1:0], P3[1:0]}, the result is bitwise-voting(2'b01, 2'b10, 2'b11) = 2'b11.

In other words, pclk3 will be set as the default pivot in our encoding when all three P signals differ. Based on the above derivations, we conclude that this proposition holds and our pivot selection is robust. ■

4.3 Detecting Abnormal dclk signal

As previously mentioned, during the adaptive mutual synchronization, we have to check if there is any abnormal dclk signal. If there is no abnormal dclk signal, our MDLL performs the 3-way alignment. Otherwise, our MDLL performs 2-way alignment. We use counter-based circuits for this purpose. Notably, the 3 dclk signals have been largely locked in their phases in this stage. Each of our detection circuits counts the rising edges of the 3 dclk signals {dclk1, dclk2, dclk3} after the reset and checks if they are consistent.

4.4 Compensation for the Voter Circuit Delay

Previously we have assumed a zero delay for the voter circuit. In reality, this is not true and thus we need to consider it. Fortunately, as discussed in [19, 20], this can be done by inserting a dummy voter circuit in each MDLL instance on the feedback path from its output signal, dclk, back to the inputs of the 3 Phase Detectors (PD) for phase detections with {pclk1, pclk2, pclk3}. The details of this technique can be found in the literature [19, 20].

4.5 Impact of Wiring Effects

The overall architecture of our FET-PLL has not carefully considered the wiring delay mismatch of the two signals driving a PD. This may degrade our phase detection accuracy and lead to larger jitters in our output clock signal, clk_out. Recall that there are 5 Phase Detectors (PDs) in each MDLL instance. Ideally, a PD should be placed in a location of roughly equal distance to the two clock signals it compares. As a result, we take out the $5 \times 3 = 15$ PDs that are originally part of the 3 MDLLs and distribute them at proper positions as shown in Fig. 11. In this figure, PD1.1 to PD1.5 belong to MDLL-1, PD2.1 to PD2.5 belong to MDLL-2, and PD3.1 to PD3.5 belong to MDLL-3. Take PD2.3 on the left-hand-side column for example, it is the 3rd PD in MDLL-2 and receives dclk2 and pclk3 as the inputs. So, it is located in the middle region between MDLL-2 and PLL-3. On the other hand, PD2.4 on the right-hand-side column is the 4th PD in MDLL-2 and receives dclk2 and dclk1 as the inputs and so it is located in the middle region between MDLL-2 and MDLL-1.

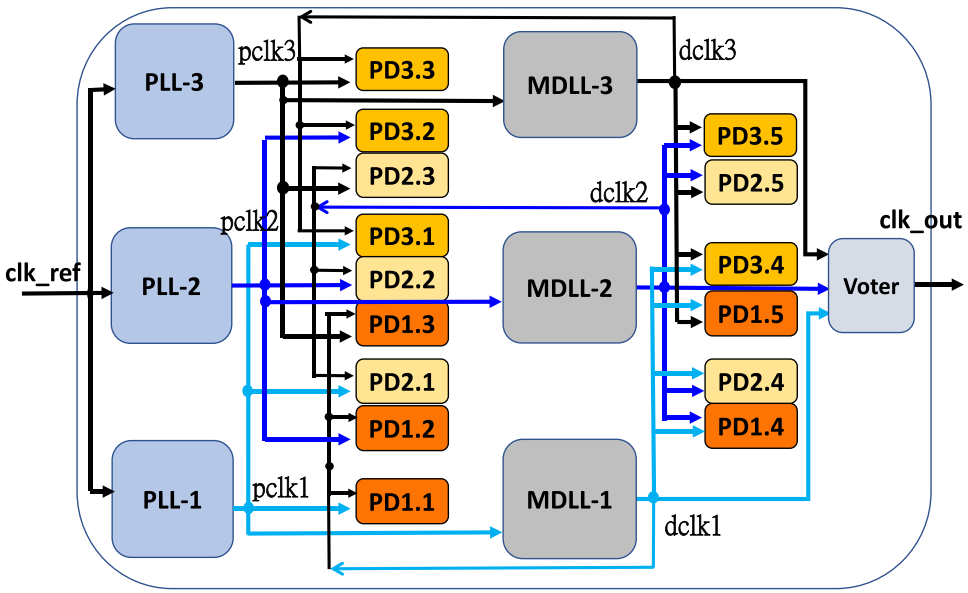

Fig. 11 Placement of the Phase Detectors (PDs) originally in our 3 MDLLs to reduce the wiring effects on the accuracy of our phase detection. In turn, the jitter of *clk_out* can be reduced

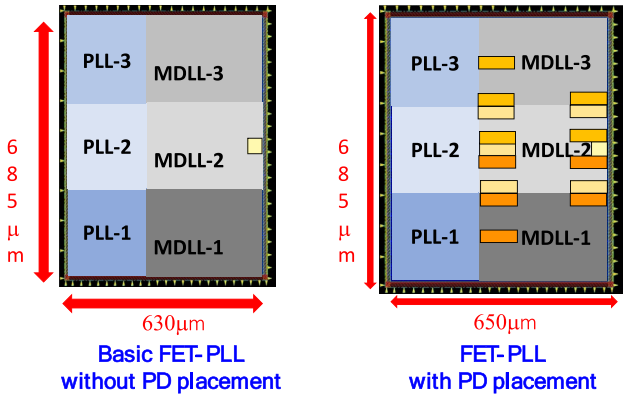

Fig. 12 Layout of the proposed FET-PLL

Table 2 Area overhead of our FET-PLL

Area \ PLL version	Primitive PLL	Basic FET-PLL	Careful-PD-Placement FET-PLL
Area (um^2)	56,592	431,843	445,546
Area comparison	100%	763%	787%

5 Simulation Results

5.1 Layout and Post-Layout Transistor-Level Simulation

We have implemented the proposed FET-PLL using a 90 nm CMOS process [5]. The underlying primitive PLL design is based on the work reported in [18] and the underlying MDLL is based on the work reported in [21]. Figure 12 shows the layouts of two versions of our FET-PLL design. The first version (on the left) is our basic version without special treatment of the PDs, while the 2nd version (on the right) places the PDs carefully to balance the wiring effects of the two input signals to each PDs. Their areas are 0.431mm^2 and 0.445mm^2, respectively. As summarized in Table 2, the area of the primitive PLL is treated as the reference (i.e., 100%). Then, the area of our basic FET-PLL is 763% and that of the careful-PD-placement version is 787%.

Post-layout transistor-level simulation has been performed to extract the waveforms of some key signals as shown in Fig. 13. In this simulation, a 32 MHz *clk_ref* is applied, and a 1 GHz *clk_out* is produced. It is worth mentioning that the proposed method is general and not limited to a specific clock frequency. For example, a higher clock frequency can be operated as long as the underlying PLL instances and the DLL instances can support it.

As can be seen from the waveforms, the three PLL instances are successfully locked after a while but at different times. Then, the design enters the pivot-based synchronization stage, in this case, pclk1 is elected as the pivot signal because PLL-1 is locked first. After two MDLLs are locked, our FET-PLL enters the mutual synchronization stage. Also, the flag signal "ready" is raised. Figure 14 shows the zoom-in view. The output clock signals of the 3 MDLLs, i.e., {dclk1, dclk2, dclk3} are approximately aligned as expected.

5.2 Jitter performance

For jitter performance assessment, we record the waveform of the output clock, *clk_out*, for roughly 1000 clock cycles after it is stable. By a *perl* script, we analyze the 1000 clock cycle time samples to derive its RMS jitter and Peak-to-Peak Jitter. The results are shown in Table 3. In terms of the

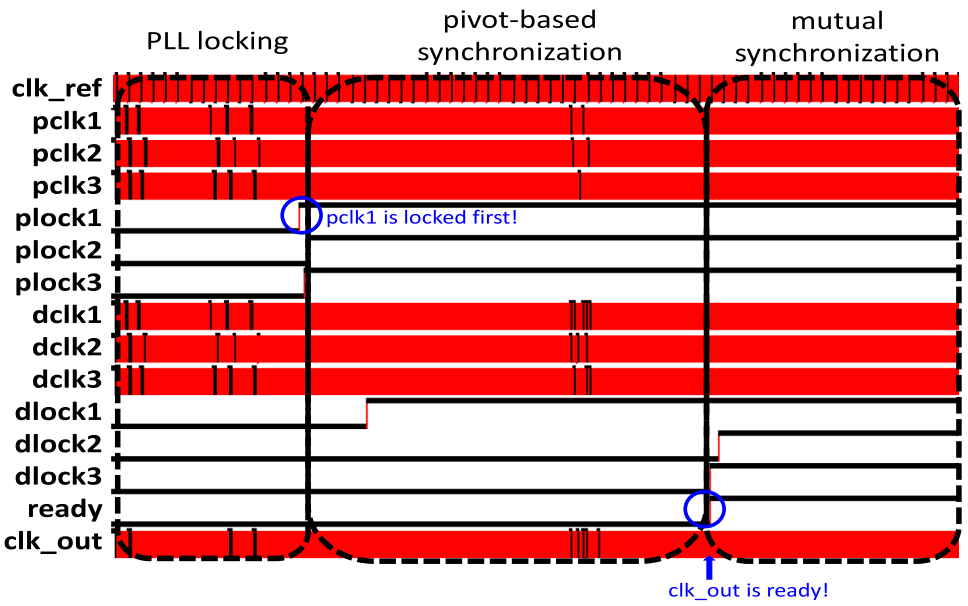

Fig. 13 Post-layout transistor-level simulation waveforms of our FET-PLL

peak-to-peak jitter, the original primitive PLL is 9.4 ps. Our basic FET-PLL is 24.8 ps. However, through the careful-PD-placement, the peak-to-peak jitter can be reduced to 11 ps, which is quite similar to that of the primitive PLL in the typical process corners.

We have also simulated our 2nd version of FET-PLL under various 5 process corners, {TT, SS, FF, SNFP, FNSP} to record their impacts on the jitters. The results are shown in Table 4. It shows a peak-to-peak jitter range from 7.4 ps to 20.9 ps. Furthermore, we conducted the Monte Carlo simulation. Even though the simulation is quite time-consuming, we managed to have collected the data from 100 Monte Carlo points. It shows that the peak-to-peak jitter of our output clock signal ranges from 6 to 28 ps.

5.3 Simulation of Faulty Conditions

Our FEL-PLL can be viewed as 3 modules producing 3 dclk signals that are further voted to produce the final output clock signal, *clk_out*. When there is a fault or soft error within a module, its adverse effect needs to be propagated to its dclk signal in order to impact *clk_out*. As a result, for simplicity without losing generality, our fault simulation will consider only faults or soft errors that directly affect the dclk signals.

To verify the fault soft-error tolerance of our FET-PLL, we consider the following two cases:

(1) Case 1: We verify if our output clock signal is still valid when there is one stuck-at-1 fault at a *dclk* signal.
(2) Case 2: We verify if our output clock signal is still valid when there is one soft error attacking a *pclk* signal.

To perform the fault injection, we modify our FET-PLL by inserting 3 MUXes, with the control signals of {F1, F2, F3} and the side-input signals of {E1, E2, E3}, as shown in Fig. 15. For example, if we attempt to inject a stuck-at-1 fault to pclk3, we only need to set {F1, F2, F3} to (0, 0, 0) in the beginning and then turn F3 to '1' at the fault activation time. Also, we apply '1' to signal E3. Overall, it will activate a stuck-at-1 fault at pclk3. On the other hand, if we attempt to inject a soft error to pclk1, we change F3 to '1' for a short time interval to connect stuck-at-1 faulty signal E1 to pclk1. Out of this short time interval, pclk1 will remain error-free.

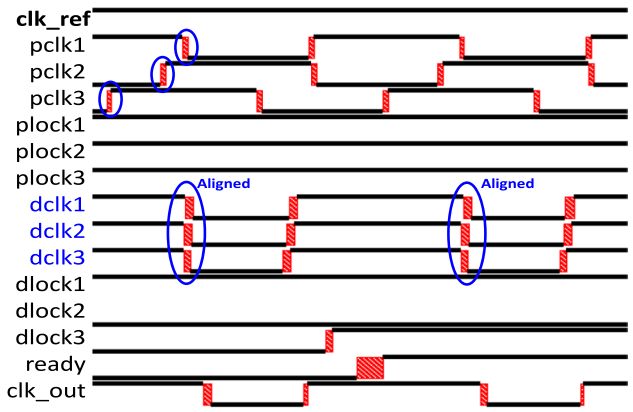

Fig. 14 Waveforms of key signals of our FET-PLL after "clk_out" is ready

Table 3 Impact of various Phase detectors

Jitter Type \ PLL version	Primitive PLL	Baseline FET-PLL	Careful-PD-Placement FET-PLL
RMS jitter(ps)	1.1	1.8	1.4
Peak to peak jitter(ps)	9.4	24.8	11.0

(Faulty Case 1): Stuck-at-1 fault at pclk3 The simulation results are shown in Fig. 16. Notably, our adaptive mutual synchronization will automatically switch to the 2-way alignment scheme and therefore the peak-to-peak jitter of our output clock signal remains reasonably small at 12.2 ps in the presence of the stuck-at fault. If we do not have an adaptive scheme and still use the default 3-way scheme, then the peak-to-peak jitter will become much larger at 63.2 ps.

(Faulty Case 2): A soft error occurring to pclk1 The simulation results are shown in Fig. 17. The soft error is represented by a "short pulse" imposed on the waveform of pclk1 for a short time interval. Even though this error is transient, it still propagates through its subsequent MDLL-1 down the stream and affects signal dclk1 as highlighted in the figure. Again, this transient error on dclk1 is masked after it is voted with the normal {dclk2, dclk3} signals. Since this soft error has only a short-time effect, our adaptive mutual synchronization still uses the default 3-way scheme. Even so, the analysis of the waveform of the output clock signal, clk_out, indicates that the peak-to-peak jitter remains reasonably small at 19.3 ps.

Table 4 The jitter performance under 5 process corners

Metric	Process Corners				
	TT	SS	FF	SNFP	FNSP
ref_clk	32MHz	32MHz	32MHz	32MHz	32MHz
clk_out	1GHz	1GHz	1GHz	1GHz	1GHz
Jitter range(ps)	[-5.7, 5.3]	[-11.3, 9.5]	[-4.3,4.5]	[-3.5, 3.9]	[-5.7, 9.1]
RMS jitter(ps)	1.4	2.5	1.3	1.1	1.6
Peak to peak jitter(ps)	11.0	20.9	8.8	7.4	14.8

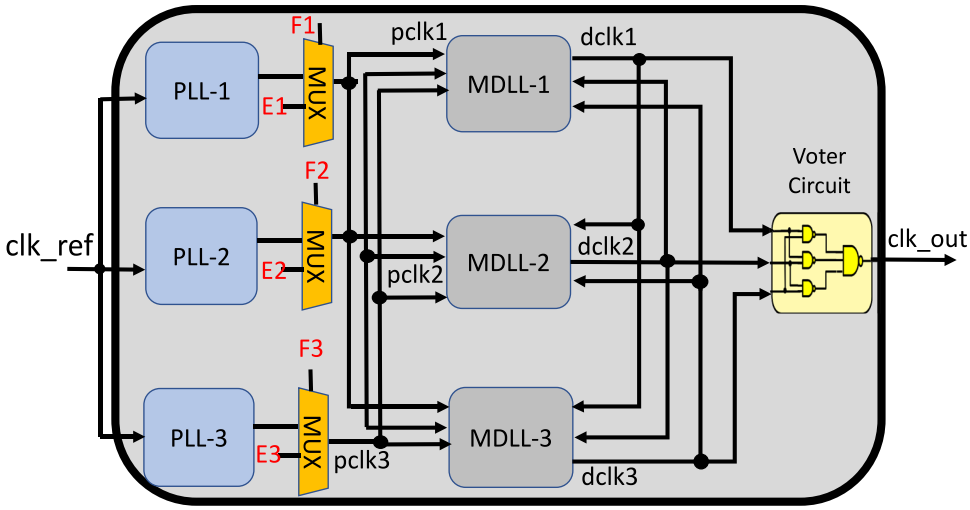

Fig. 15 A modified FET-PLL with the fault injection ability through 3 inserted MUXes

5.4 Miscellaneous Discussions

In a PLL or DLL, a loop filter with a proper loop bandwidth is often solicited to achieve a good balance between the PLL's sensitivity to environmental changes and the reduced clock jitter. In our designs, we use suppressive digital filter techniques proposed in [6] for this purpose.

In a very severe situation when the faults or soft errors affect more than one dclk signal driving the voter circuit, then the output clock could still fail. In that case, we need to rely on a PLL monitor to detect this kind of anomaly to raise an alert signal. For example, the PLL monitor reported in [4] can detect a transient error occurring to a clock signal.

It is worth mentioning that the output clock of a PLL is very vulnerable. Even a very short-time transient glitch caused by a particle strike will make it completely useless and will cause severe computational errors in the downstream logic circuits it drives. This work may incur a larger area overhead than the prior Radiation-Hardening-By-Design (RHBD) techniques. However, we also provide much stronger protection that covers almost the entire circuit except the tiny output voter against the attack of a single-module fault or soft error. The benefit of this kind of protection is not just the improvement of the jitter performance but to saving the entire IC from catastrophic failure in a dangerous situation. Furthermore, the area of our mostly cell-based PLL can shrink with the advanced process technology and therefore it will become an even smaller portion of the entire chip area in the future.

Our area overhead may increase the probability of the Single-Event-Transient (SET). However, the resulting failing probability due to soft errors of our FET-PLL is still

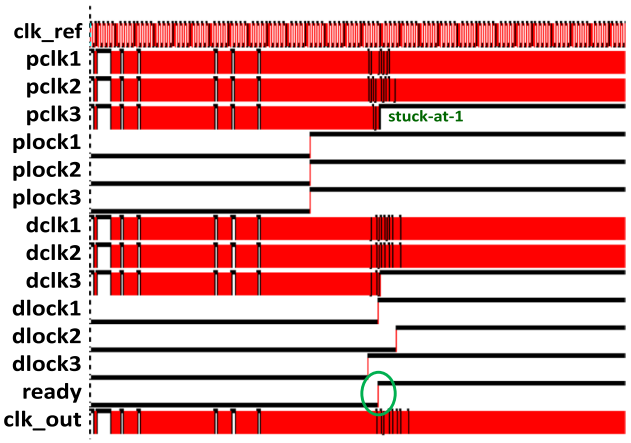

Fig. 16 Simulation waveforms of our FET-PLL with a stuck-at-1 fault occurring to pclk3 activated at a certain time

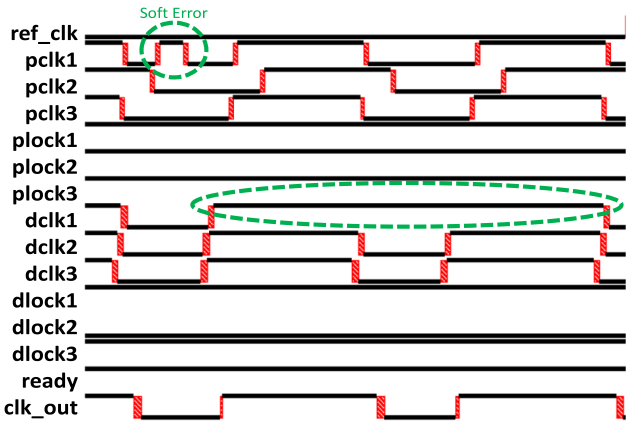

Fig. 17 Simulation waveforms of our FET-PLL with a soft error occurring to signal pclk1

much lower. Let the SET rate in a primitive PLL be denoted as p (per clock cycle). Then, the SET rate in our FET-PLL is about $(7.87 \cdot p)$. Now let us estimate the failing probability. A primitive PLL is very likely to fail at an SET so its failing probability is p. However, our FET-PLL fails only when two events strike the same module at the same clock cycle, which is roughly $(7.87/3)^2 \cdot p^2 \sim 6.88 \cdot p^2$. Consider one SET per minute for a system at 1 GHz. It is equivalent to one strike every $6 \cdot 10^{10}$ clock cycles. Then p is around $(1.67 \cdot 10^{-11})$, and $6.88 \cdot p^2$ is around $(1.92 \cdot 10^{-21})$. Relatively, the failing rate of our FET-PLL to that of a primitive PLL is thus $(1.92 \cdot 10^{-21})/(1.67 \cdot 10^{-11}) = 1.14 \cdot 10^{-10}$, indicating that our FET-PLL achieves a 10 orders of magnitude reduction in the failing probability.

6 Conclusion

In this work, we have reported the first truly fault and soft-error-tolerant PLL design. As compared to the previous Radiation-Hardening-By-Design (RHBD) techniques, our design provides stronger and more complete protection against radiation attacks as well as permanent faults triggered in the field. We demonstrated that naïve Triple-Modular Redundancy (TMR) will not achieve this goal unless enhanced by a synchronization scheme before the voting operation that produces the final output clock signal. Post-layout transistor-level simulation based on a 90 nm CMOS process has been used to demonstrate its ability to tolerate soft errors or stuck-at faults affecting only one module. In addition to the basic pivot-based synchronization, we also developed an adaptive mutual synchronization scheme, which is effective in reducing the peak-to-peak jitter from 63.2 ps to 12.2 ps in a faulty situation.

Funding National Science and Technology Council, 111-2221-E-007-115-MY3, Shi-Yu Huang

Data Availability The datasets generated during and/or analyzed during the current study are available from the corresponding author upon reasonable request.

Declarations

Competing Interest This work was sponsored in part by the Ministry of Science and Technology (MOST) of Taiwan under research grants 111–2221-E-007–115-MY3, and in part, by Power Semiconductor Manufacturing Corp. We also acknowledge the help of Taiwan Semiconductor Research Institute (TSRI) for providing the access to the EDA tools.

References

1. Boulghassoul Y, Massengill LW, Sternberg AL, Bhuva BL, Holman WT (2006) Towards SET mitigation in RF digital PLLs: from error characterization to radiation hardening considerations. IEEE Trans Nuclear Science 53(4):2047–2053
2. Chen P-L, Chung C-C, Lee C-Y (2005) A portable digitally controlled oscillator using novel varactors. IEEE Trans Circuits Syst II, Exp Briefs 52(5):233–237
3. Chen Z, Lin M, Zheng Y, Wei Z, Huang S, Zou S (2016) Single-event transient characterization of a radiation-tolerant charge-pump phase-locked loop fabricated in 130 nm PD-SOI technology. IEEE Trans Nuclear Science 63(4):2402–2408
4. Chen W-H, Hsu C-C, Huang S-Y (2020) Rapid PLL monitoring by a novel min-MAX time-to-digital converter. Proc IEEE Int'l Test Conf (ITC), pp. 1–8
5. CIC Reference Flow for Cell-based IC Design (2008) Chip implementation center, CIC, Taiwan, Document no. CIC-DSD-RD-08–01
6. Hsu H-J, Huang S.Y (2011) A low-jitter ADPLL via a suppressive loop filter and an interpolation-based locking scheme. IEEE Trans Very Large Scale Integr Syst 19(1):165–170
7. Kim D-S, Song H, Kim T, Kim S, Jeong D-K (2010) A 0.3-1.4 GHz all-digital fractional-N PLL with suppressive loop gain controller. IEEE J Solid-State Circuits 45(11):2300–2311
8. Lin DY-W, Wen CH-P (2022) Rad-hard designs by automated latching-delay assignment and time-borrowable d-flip-flop. IEEE Trans Comput (TC) 71(5):1008–1020
9. Loveless TD, Massengill LW, Bhuva BL, Holman WT, Witulski AF, Boulghassoul Y (2006) A hardened-by-design technique for RF digital phase-locked loops. IEEE Trans Nuclear Science 53(6):3432–3438
10. Loveless TD, Massengill LW, Bhuva BL, Holman WT, Reed RA, McMorrow D, Melinger JS, Jenkins P (2007) A single-event-hardened phase-locked loop fabricated in 130 nm CMOS. IEEE Trans Nucl Sci 54(6):2012–2020
11. Loveless TD, Massengill LW, Bhuva BL, Holman WT, Casey MC, Reed RA (2008) A probabilistic analysis technique applied to a radiation-hardened-by-design voltage-controlled oscillator for mixed-signal phase-locked loops. IEEE Trans Nucl Sci 55(6):3447–3455
12. Nemmani AN (2005) Design techniques for radiation hardened phase-locked loops," master's thesis, Oregon State University, A874174
13. Olsson T, Nilsson P (2004) A digitally controlled PLL for SoC applications. IEEE J Solid-State Circuits 39(5):751–760
14. Richards EW, Loveless TD, Kauppila JS, Haeffner TD, Holman WT, Massengill LW. IEEE Trans Nucl Sci 67(6):1144–1151
15. Riley TA, Copeland MA, Kwasniewski TA (1993) Delta-sigma modulation in fractional-n frequency synthesis. IEEE J Solid-State Circuits 28(5):553–559
16. Shuler RL (2010) "SEU/SET tolerant phase-locked loops", Chapter 12 of radiation effects in semiconductors edited by K. Imprint-CRC Press, Iniewski
17. Shuler RL, Kouba C, O'Neill PM (2005) SEU performance of TAG based flip-flops. IEEE Trans Nucl Sci 52(6):2550–2553
18. Tzeng C-W, Huang S-Y (2014) Parameterized all-digital PLL architecture and its compiler to support easy process migration. IEEE Trans on VLSI Systems 22(3):621–630

19. Yang J-Y, Huang S-Y (2020) Fault and soft error tolerant delay-locked loop. Proc of IEEE Asian Test Symp (ATS), pp. 1–6
20. Yang J-Y, Huang S-Y (2022) Process resilient fault-tolerant delay locked loop using TMR with dynamic timing correction. IEEE Trans. Comput-Aided Design Integr Circuits Syst (TCAD) 41(5):1563–1572
21. Zhang Z-H, Chu W, Huang S-Y (2019) A ping-pong methodology for boosting the resilience of cell-based delay-locked loop. IEEE Access 7:97928–97937

Shun-Hua Yang received his B.S. degree in Electrical Engineering from National Sun Chung Cheng University, Taiwan, in 2019, and his M.S. degree in Electrical Engineering from National Tsing Hua University, Taiwan in 2023. His research interests include VLSI design and All-Digital Phase-Locked Loop (ADPLL).

Shi-Yu Huang received his B.S. and M.S. degrees in Electrical Engineering from National Taiwan University in 1988, and 1992, and a Ph.D. degree in Electrical and Computer Engineering from the University of California, Santa Barbara in 1997, respectively. He has been on the faculty of the EE Dept., National Tsing Hua University, Taiwan, since 1999. His research interests include VLSI design, automation, and testing, with a current emphasis on All-Digital Phase-Locked Loop (ADPLL) design and its application in parametric fault testing and monitoring in 3D ICs. He ever served as an Associate Editor for IEEE Trans. on Computers from 2015 to 2018.