Journal of VLSI circuits and systems

RESEARCH ARTICLE

Journal of VLSI Circuits and Systems, ISSN: 2582-1458 Vol. 8, No. 2, 2026 (pp. 1-14)
WWW.VLSIJOURNAL.COM

Physics-Informed VLSI TinyML-Based Self-Healing PLL with Power-Jitter Optimization Under Aging and PVT Variations

Isam Aameer Ibrahim[1], Israa J. Mohammed[2], Ibtihal Razaq Niama ALRubeei[3]*, Naseer Ali Hussien[4], Hussain Ali Mutar[2], Haider TH. Salim ALRikabi[3]

[1]Electrical Engineering Department, College of Engineering, Thi-Qar University, Nasiriyah, Iraq.
[2]Computer Department, College of Computer Science and Information Technology, Wasit University, Wasit, Iraq.
[3]Electrical Engineering Department, College of Engineering, Wasit University, Wasit, Iraq.
[4]College of Technical Engineering, Department of Computer Engineering Technology, Al-Ayen Iraqi University, AUIQ, Nasiriyah, Iraq.

KEYWORDS:
Phase-locked loop (PLL);
self-healing circuits;
VLSI;
TinyML;
PVT variation;
BTI aging;
stability constraints;
multi-objective optimization

ARTICLE HISTORY:
Received: 09.03.2026
Revised: 05.04.2026
Accepted: 16.04.2026

DOI:
https://doi.org/10.31838/jvcs/08.02.01

Abstract

Phase-locked loops (PLLs) are essential components for generating frequency signals in today's systems on a chip, but their overall jitter, stability, and behavior in terms of being 'locked' to each other can be affected by the varied manufacturing process, operating voltage, and temperature, and long-term aging of the devices as well as other factors (e.g., supply chain issues, etc.). PLLs use feedback to regulate the output frequency, and any of the aforementioned types of variability can result in less-than-optimal characteristics (e.g., loop gain, VCO sensitivity, charge pumps, etc.), thus causing PLLs to drift away from the intended damping and bandwidth; therefore, increasing RMS jitter or forcing conservative guard banding that wastes power. Recent research has demonstrated the continued development of PLL architectures/calibration techniques and how critical it is that, as semiconductor technology continues to scale down, sufficient attention will be paid to ensuring that the PLL user's dynamic performance remains robust, given the continued variability and reliability issues associated with technology scaling. Additionally, studies on aging have shown that mechanisms like BTI have measurable effects on charge pump operation, underscoring the need for lifetime-based compensation methods rather than fixed margins. This study proposes a physics-informed TinyML self-healing PLL that utilizes (i) on-chip monitors, (ii) a compact TinyML estimator to provide a near-optimal initial knob setting, and (iii) a constrained local refinement stage that enforces power budgets and stability/transient constraints expressed through ζ, ω_n, resonant peaking M_r, and settling time t_s. We formulate runtime tuning as a constrained multi-objective optimization over I_{CP} and loop bandwidth BW. A simulation-centered evaluation using a behavioral CP-PLL model augmented with stochastic PVT spread and aging degradation spans $T \in [-20,100]$°C, $V_{DD} \in [0.85, 1.15]$V, and aging $a \in [0, 7]$ years will be performed with the goal of demonstrating that the proposed method will achieve a significant and consistent reduction in jitter across 1200 random operating conditions for the represented device under the constraints imposed and enforced; conversely, it will be demonstrated that jitter reduction based purely on TinyML decision making will have nonzero violations related to the constraints imposed. The results of the evaluation will demonstrate the practicality of using compact learning in combination with physics-based feasibility checks to achieve predictable self-healing behavior suitable for VLSI clocking subsystems.

Authors' e-mail ID: isam.ameer@utq.edu.iq; imohammed@uowasit.edu.iq; ibtihal.razaq@uowasit.edu.iq; naseerali@alayen.edu.iq; hmutar@uowasit.edu.iq; hdhiyab@uowasit.edu.iq

Authors' ORCID ID: 0000-0001-5508-2021; 0009-0009-8852-0097; 0009-0003-8958-8487; 0000-0001-9499-6694; 0009-0004-4934-416X; 0000-0002-5201-2084

How to cite this article: Isam Aameer Ibrahim et al., Physics-Informed VLSI TinyML-Based Self-Healing PLL with Power-Jitter Optimization Under Aging and PVT Variations, Journal of VLSI Circuits and System, Vol. 8, No. 2, 2026 (pp. 1-14)

Introduction

Phase-locked loops (PLLs) produce frequency-corrected clock signals to support clock distribution to the CPU, memory interface, and high-speed links.[1,2] While timing jitter typically has a local circuit impact, timing jitter in a PLL directly impacts setup and hold margin and contributes to the degradation of the eye opening of the link.[3-6] The result, in many cases, is the need to run at lower clock frequencies than desired or to increase (or increase) the supply voltage to maintain production yield.[7,8] Continued scaling of integrated circuits will lead to less analog headroom and increasing variability, which leads to a PLL's clock frequency drifting (i.e., the average jitter increases and also increases the likelihood of the maximum case tail risk) away from its optimal operating point relative to design time.[9] These issues are compounded in long-lived deployments due to gradual changes in the physical device parameters resulting from aging and degradation of the mixed-signal loops (particularly within the integrated circuit).

Recent gap (3-5 years)

Significantly advanced since previous work in this field are digitally assisted PLL (phase-locked loop) techniques and jitter calibration methods. Specifically, adaptive mismatch calibration and adaptive background-tuning techniques can adjust the charge pump mismatch (and thus the loop error) to a given jitter class.[10,11] Many current calibration methods are targeted at a single impairment type, operate under nominal conditions, or do not explicitly account for long-term aging drift and multi-objective calibration constraints.[12] Studies indicating that BTI aging can impact the performance of charge pump circuits demonstrate that fixed performance margins do not totally account for runtime adjustments as devices age.[13] The remaining gap is an integrated runtime strategy which includes: (i) adaptation to changes in PVT and aging, (ii) stability defined in physical terms, and (iii) adherence to defined power budgets.

Objectives

The goal of this article is to develop a standalone self-configuring PLL monitor that meets these criteria:

1. Monitors PVT and aging drift over time using inexpensive components
2. Decreases jitter while adhering to hard limits on power consumption
3. Enforces stability and transient constraints through ζ, ω_n, M_r, and t_s,
4. Keeps the implementation cost within acceptable limits (i.e., TinyML ratio + a small search space)

Contributions

- A constrained multi-objective formulation for runtime tuning of I_{CP} and BW under PVT and aging.
- A physics-informed TinyML architecture to provide fast inference and feasibility-guaranteeing refinement.
- Stability constraints (based on damping ratio, resonant peaking/settling time/lock time).
- Simulation-centered validation across wide conditions, including violation-rate analysis and statistical reporting.
- Ablation comparing static baseline, TinyML-only, and TinyML+refinement variants.

Related Work

For this paper, we present our findings from a survey of recent peer-reviewed literature related to PLLs (2019-2025) across three different themes: (1) the fundamental principles of PLLs and emerging trends, (2) digitally assisted calibration techniques and methods for controlling background calibration, and (3) the impact that aging and reliability have on both the individual sub-blocks within a PLL and the overall performance of the PLL.

Examples of the literature we reviewed in our survey include several survey papers on the emerging trends in PLLs,[1,14] adaptive mismatch calibration methods for self-biased PLLs,[15] the effects of BTI aging on charge-pump accuracy,[3] and recent papers that focus on methods of adapting bandwidth or background bandwidth control within digital PLLs.[6,16-18]

Furthermore, we evaluated resources describing physics-informed machine learning techniques for device/circuit simulation/modeling and TinyML constraints for embedded implementations of devices/circuits.[19-21] Digital PLL calibration and background bandwidth control. Various background and adaptive bandwidth calibration techniques available for digital PLLs improve jitter and robustness. Digital PLLs may use many LMS-like or multi-tap adaptive calibration techniques; however, these techniques typically do not utilize explicit indicators of aging and, as a result, may not satisfy stability constraints with respect ζ, M_r, settling time. Calibration techniques for charge-pump mismatch and robustness. When charge-pump mismatch calibration targets deterministic loop errors, jitter performance can be improved by eliminating deterministic

loop errors.[22,23] However, due to aging drift and other factors, the effective parameters of PLLs will change over time, i.e., implementing mismatch calibration will not ensure that the PLL performs robustly throughout its lifetime. Reliability studies addressing aging effects. Research has shown that aging, specifically BTI, may impact charge-pump performance and, subsequently, PLL loop performance over extended periods.[24] As a result, one-time PLL calibration does not guarantee that PLLs will continue to perform reliably; therefore, adaptations must be made throughout the PLL's lifetime.[25,26] Physics-informed machine learning (ML) and TinyML implementation constraints. Physics-informed ML methods can provide structure and constraints that can be encoded into the ML model, resulting in greater trust and improved extrapolation performance during modeling/optimization tasks.[19,34-36] TinyML surveys provided relevant feasibility information that is helpful in choosing appropriate models and performing duty-cycled inference based on limited computation and energy budgets.[21,27-29] The taxonomy of PLL runtime adaptation methods shown in Figure 1 was a heuristic/LMS background tuning, digital assist for mismatch and correction, reliability/aging-aware compensation, and physics-informed ML with constraints.

To highlight how this method differs from prior PLL calibration and reliability approaches, Table 1 compares key deployability axes—runtime operation, aging awareness, and enforcement of power/stability feasibility—rather than exhaustively surveying all work.

The table contrasts runtime capability, aging awareness, constraint enforcement, and typical limitations across major families of PLL calibration and reliability methods.

Problem Formulation

Due to the very close connection between power consumption, timing jitter, and transient behavior (e.g., lock time), the same fixed loop configuration can shift from a well-damped operating point to an underdamped regime (jitter peaking) or an overdamped regime (slow lock and decreased disturbance rejection) as a result of PVT variability and parameter drift due to aging. To allow adaptation to changing configurations during runtime, we have formulated the self-healing PLL problem as a constrained multi-objective optimization where an observable state (temperature, supply, aging proxy, phase-error statistics) is mapped to a small number of hardware-realizable control knobs. The

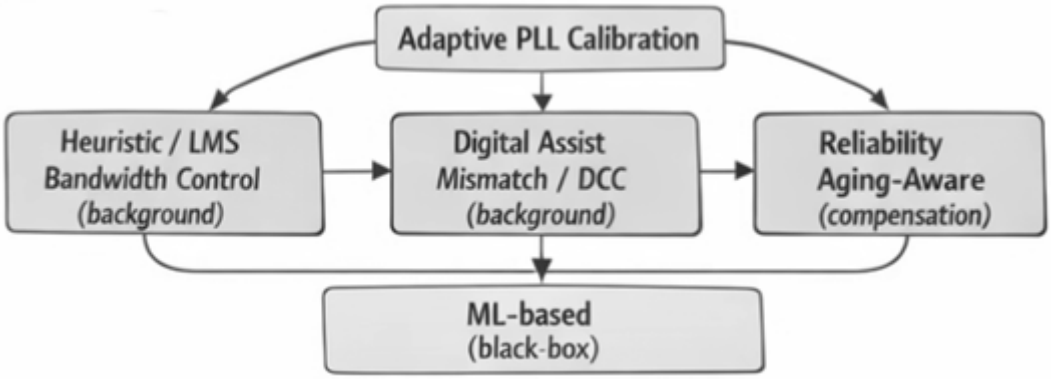

Fig. 1: Taxonomy of PLL runtime adaptation: heuristic/LMS background tuning, digital assist for mismatch and correction, reliability/aging-aware compensation, and physics-informed ML with constraints (this work)

Table 1: Comparative summary of representative PLL adaptation approaches and their design attributes.

Category	Representative direction	Runtime	Aging-aware	Constraint-aware	Limitation
Background tuning	LMS/heuristic bandwidth control[30,31]	✓	partial	partial	limited constraints
Mismatch calibration	CP mismatch calibration[11,32]	✓	×	partial	not lifetime-aware
Aging analysis	BTI influence studies[26,33]	×/partial	✓	×	not direct healing
TinyML/ML control	Compact learned tuning[28]	✓	possible	× unless constrained	safety/feasibility
This work	TinyML + physics refinement	✓	✓	✓	Needs signoff modeling in future

formulation allows for direct implementation using a low-footprint controller, with a TinyML estimator providing an initial configuration that is near-optimal, and a physics-informed refinement stage that enforces feasibility through stability and policy constraints.

Notation

The terms were chosen with respect to both what can be measured on-chip and what can be controlled with digital trimming methods. The vector of state variables $s = [T, V_{DD}, a]^T$ summarizes the dominant sources of drift in the measured signal, based on temperature and supply (both of which are very rapidly changing due to environmental factors). The aging indicator a indicates long-term degradation (e.g., a decrease in the ring oscillator frequency) of the measured signal over time. The control vector is given by $u = [I_{CP}, BW]^T$ and contains two very common control parameters used in modern PLLs (including I_{CP}, the effective charge-pump bias or equivalent trim to loop gain, and *BW*, the selectable loop bandwidth set by the loop filter or gain settings. The output vector is given by $y = [P, J, L]^T$ and contains measures that correspond to system-level concerns: power *P*, which drives energy-efficiency considerations; *J* RMS jitter, which drives timing margin and link-quality issues; and lock time *L*, which drives wake-up latency and dynamic Freq. Scalable Responsiveness).

Multi-objective Optimization

We minimize a scalarized objective:

$$\min_u F(u;s) = w_P \frac{P(u;s)}{P_{ref}} + w_J \frac{J(u;s)}{J_{ref}} + w_L \frac{L(u;s)}{L_{ref}}$$

Since PLL tuning rarely has a single primary metric, we use a multi-objective scalarization to optimize performance. For example, by minimizing jitter alone, you may use a bandwidth or bias setting that is too high, which in turn will increase overall power consumption and/or cause the loop to be unstable. On the other hand, if you minimize power consumption only, you'll likely increase jitter and decrease time-to-lock on the PLL. The weighted objective $F(u; s)$ provides a way to control how you intend to implement your PLL, such as using jitter priority for clocking in high-performance mode and/or power priority for clocking in low-power mode, all while providing a consistent optimization interface across various applications. Additionally, the objective will not be evaluated separately from the known explicit feasibility constraints, thereby ensuring that the ultimate decision remains within the bounds of stability and policy correctness, regardless of which area the states have moved into due to unique PVT and aging characteristics.

Constraints

Constraints transform the optimization into a safe engineering controller. The power budget:

$$P(u;s) \le (1+\delta)P_{baseline}(s)$$

In a trustworthy self-healing controller based on "learning-based tuning," the constraint set is the mechanism that creates this effect. The power-budget constraint limits $P(u;s) \le (1+\delta)P_{baseline}(s)$ the controller's ability to make excessive energy exchanges for improvements in jitter and provides for system-wide governance (e.g., a firmware-defined δ). The stability/transient constraints expressed via ζ, ω_n, resonant peaking $M_r(\zeta)$, and settling time $t_s \approx 4/(\zeta\omega_n)$ define a set of limits that can physically define the guardrails that are still valid under all operational conditions. The constraint set can be enforced at runtime through the use of either (a) lookup tables or (b) simple, lightweight estimators obtained via design-time characterization (i.e., the controller can reject unsafe knob-setting configurations before they are executed). Overall, the controller's ability to satisfy the constraints in the constraint set guarantees that its actions will maintain stable system operation, limit jitter peaks, and maintain lock conditions within reasonable limits.

Formal Small-Signal CP-PLL Model (Loop Gain, ω_n, ζ)

Analytically, the small signal model provides a means of linking the set of tunable knobs (I_{CP}, *BW*) with the stability/transient performance metrics (ζ, ω_n) that the controller must utilize when determining the control loop. In contrast to the full PLL, which is characterized by chaotic acquisition and significant noise coupling, near-lock transient dynamics can be closely approximated by a second-order function whose damping and natural frequency determine the amount of ringing (or peaking) and settling time. By deriving both ω_n and ζ from the loop gain parameters, control-oriented quantities are generated that may be used as constraints on the run time. This is critically important due to the impact of aging and process/voltage/temperature (PVT) drift; instead of estimating appropriate knob values for use by the PLL controller, the controller seeks a stable area that would exhibit acceptable damping and limited resonant peaking, resulting in reliable self-healing behavior of the control function.

Loop gain model

For a charge-pump PLL with a divider N, approximate PFD/CP phase-to-current gain:

$$K_\phi[A/rad] \approx \frac{I_{CP}}{2\pi}$$

VCO phase relation:

$$\Phi_{out}(s) = \frac{K_{vco}}{s} V_c(s)$$

With a lead-lag filter approximation:

$$Z(s) \approx \frac{1 + s/\omega_z}{sC_{eq}}$$

Open-loop gain:

$$L(s) = \frac{K_\phi K_{vco}}{N} \cdot \frac{Z(s)}{s} = \frac{K_\phi K_{vco}}{NC_{eq}} \cdot \frac{1 + s/\omega_z}{s^2}$$

Let

$$K \triangleq \frac{K_\phi K_{vco}}{NC_{eq}}$$

Characteristic equation $1 + L(s) = 0$ yields:

$$s^2 + \left(\frac{K}{\omega_z}\right)s + K = 0$$

Comparing to $s^2 + 2\zeta\omega_n s + \omega_n^2 = 0$:

$$\boxed{\omega_n = \sqrt{K}}, \quad \boxed{\zeta = \frac{1}{2}\frac{\sqrt{K}}{\omega_z}}$$

Phase margin approximation (via ζ)

A practical approximation mapping ζ to phase margin:

$$PM(\zeta)[\text{deg}] \approx \tan^{-1}\left(\frac{2\zeta}{\sqrt{\sqrt{1+4\zeta^4} - 2\zeta^2}}\right) \cdot \frac{180}{\pi}$$

and the constraint:

$$PM(\zeta) \geq PM_{min}$$

Peaking constraint (via ζ)

For $0 < \zeta < 1$:

$$M_r(\zeta) = \frac{1}{2\zeta\sqrt{1-\zeta^2}}, \quad M_r(\zeta) \leq M_{r,max}$$

Settling time constraint tied to lock time

Using the 2% settling approximation:

$$t_s \approx \frac{4}{\zeta\omega_n}$$

and enforce:

$$t_s \leq L_{max}$$

This directly ties small-signal stability and transient dynamics to the system's lock-time requirement.

Proposed Physics-Informed TinyML Self-Healing PLL

The proposed self-healing PLL will use an engineering principle that combines aspects of machine learning (ML) to develop a fast, accurate inverse mapping and use physics to enforce stability and predictability in a self-healing design. Self-healing PLLs need to operate with low computational overhead and monitoring costs, and must avoid unstable configurations at all times. To achieve these goals, the proposed self-healing PLL architecture will use (1) an on-chip monitor layer that will extract drift-related features from the PLL's operational environment, (2) an optimized tinyML model that will provide a near-ideal initial value of (I_{CP}, BW), and (3) a physics-informed refinement stage that will conduct a small constrained search to ensure that stability and policy limits are maintained during operation. The advantage of this hybrid design is that it preserves the advantages of ML (i.e., adaptability to complex drift) and those of classical control (i.e., guarantees and interpretability).

System Overview (Pipeline and Data Flow)

On-chip monitors extract drift-relevant features as shown in Figure 2 (temperature, supply, aging proxy, and phase-error statistics). TinyML produces an initial estimate of (I_{CP}, BW), while a constrained physics-informed refinement stage enforces power and stability/transient feasibility before applying bias/trim updates.

The monitor layer produces a feature vector:

$$x = [T, V_{DD}, \Delta f_{RO}, \sigma_e^2]^T$$

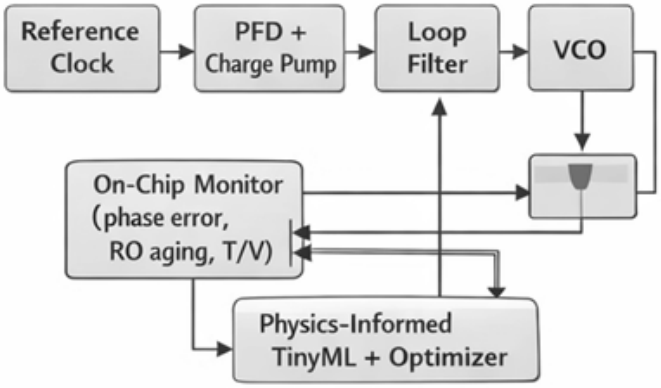

Fig. 2: Physics-informed TinyML self-healing PLL architecture.

In the context of a ring-oscillator aging proxy, Δf_{RO}, and phase-error variance, σ_e^2, the TinyML framework creates a set of initial estimates for $\hat{u} = [\widehat{I_{CP}}, \widehat{BW}]^T$. The refinement process assesses a small neighborhood of $\hat{u}$ to filter candidates outside of power or stability constraints, while selecting the best feasible candidate based on objectives (or following a jitter-first policy). This tiered approach allows for both adaptability and predetermined feasibility within the solution scope. The framework has been intentionally structured to follow a real-world SoC workflow. The monitor level produces data at low rates (e.g., only during startup or once per periodic calibration), thereby avoiding continuous use of compute resources and resulting in low energy penalties. The TinyML framework operates at a high rate as a "first guess" estimator, mapping observed indicators of drift to potential compensation actions, such as increasing loop bandwidth (BW) due to temperature increases or compensating for a reduction in transconductance due to aging by restoring effective gain. Finally, the refinement level serves as a safety filter, checking each candidate to ensure they meet power limits and stability/transient criteria, and selecting the best feasible candidate. This tiered approach allows for the controller to be resilient against sensor noise and model mismatch while providing a mechanism for timely adaptability.

Training and Deployment Workflow

Design-time simulation sweeps generate labeled data under PVT and aging, as shown in Figure 3. A compact TinyML model is trained to predict near-optimal knob settings, and runtime deployment uses TinyML inference followed by a constraint-aware local refinement step.

Offline, we generate labeled data via broad PVT+aging sweeps and define "expert" actions by selecting the best objective candidates per binned condition. The TinyML model is trained to regress from monitor features to these expert actions. Online, TinyML inference runs only during calibration windows (startup/periodic/triggered), then refinement enforces constraints.

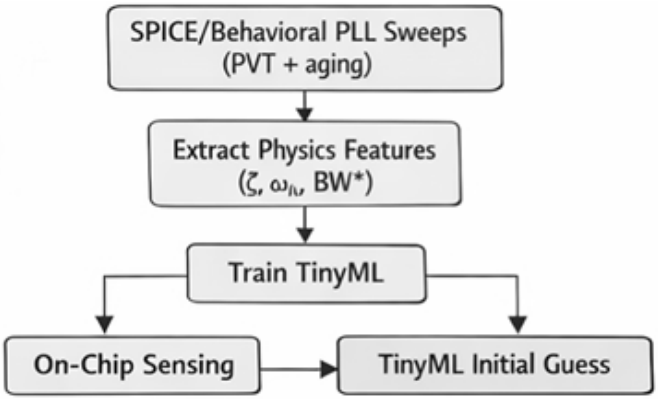

Fig. 3: Training and deployment workflow for the self-healing controller.

In-field Estimation of ζ and ω_n (for feasibility checks)

To enforce ζ, M_r, and t_s constraints in silicon, the PLL can estimate ζ and ω_n from a short ringdown following a micro-perturbation (reference phase step, divider micro-step). Using logarithmic decrement with consecutive peak magnitudes A_1, A_2, and peak-to-peak period T_d:

$$\delta_{log} = \ln(A_1 / A_2), \zeta = \frac{\delta_{\log}}{\sqrt{(2\pi)^2 + \delta_{\log}}}$$

$$\omega_d = \frac{2\pi}{T_d}, \omega_n = \frac{\omega_d}{\sqrt{1-\zeta^2}}$$

These estimates enable fast LUT-based feasibility checks without full frequency response computation.

Controller Pseudocode

The controller's runtime operation has been designed so that the actual implementation can be achieved using either firmware or embedded logic. This key design choice is to treat the TinyML inference result as an initialization that narrows the space of possible solutions, but never as the sole basis for determining the actual solution. This is important because the relationship between the state of the system(s) and the best knobs to set to achieve optimal performance can be complicated by both process variation and modeling uncertainty. As a result, the controller can evaluate a small neighborhood around the TinyML output (i.e., the location of optimal performance) to make corrections to these cases without resorting to global optimization techniques, thereby maximizing safety while minimizing computational cost.

Algorithm 1: Physics-Informed TinyML Self-Healing Pseudocode (runtime)

```
Inputs: sensed features x = [T, VDD, ΔfRO, σe^2]
        power budget δ, bounds umin, umax
        candidate grids ΓI, ΓBW
Output: applied knob setting u*
1:  û ← TinyML_Infer(x)              // û = [ÎCP,  BŴ]
2:  û ← Clip(û, umin, umax)
3:  Pbase ← MeasureOrEstimateBaselinePower()
4:  Pcap  ← (1+δ) * Pbase
5:  Best ← null
```

```
6:  For each I in ÎCP * ΓI:
7:      For each BW in BŴ * ΓBW:
8:          u ← Clip([I, BW], umin, umax)
9:          (P, J, L, φm) ← FastEval(u, x)      // model / lightweight estimator
10:         If (P ≤ Pcap) and (φm ≥ φmin):
11:             If Best is null or J < Best.J:
12:                 Best ← (u, P, J, L)
13: If Best is null:
14:     u* ← u_safe (verified fallback)
15: Else:
16:     u* ← Best.u
17: Apply u* to bias/loop filter trims
```

The refinement stage also establishes the ability to embed system policies and provide signoff requirements. For example, in this same way, we can configure the power budget δ, damping bounds $[\zeta_{min}, \zeta_{max}]$, peaking limits $M_{s,max}$, and settling time/lock time limits L_{max} for each product tier and operational mode. The refinement stage will also provide an established audit trail; every applied configuration results from a feasibility-reviewed selection from a finite set of candidates. This capability is particularly critical in clocking subsystems that are safety- or yield-driven; they will require bounded risk and predictable behavior. Lastly, the fallback behavior is an essential feature: if no candidate meets the applicable constraints (e.g., extreme corner conditions or sensor anomalies), the system will use a verified-safe configuration as its fallback. This ensures that the self-healing mechanism cannot drive the PLLs into an unstable state, regardless of adverse or extreme conditions.

Complexity and Implementation Cost

The number of Multiply-and-Accumulate (MAC) operations required for TinyML inference is projected to be in the tens. While refinement searches through a relatively small candidate set (e.g., 25) during evaluation, it may still operate with a duty cycle that incurs minimal overhead during steady-state operation. This is consistent with Feasibility Criteria for TinyML on embedded hardware.[21]

Experimental Setup (Simulation-Centered)

We performed simulation sweeps with respect to realistic drift. These sweeps covered a very broad range of PVT and aging conditions. This reflects a customary methodology adopted by designers for runtime controller training and verification during design-time characterization. The design of the experiment is consistent with this standard model to determine mean gain, tail improvement, and feasibility (constraint violations) at the same operating points for all methods.

Simulation Model

To evaluate the proposed self-healing PLL under realistic drift conditions, we use a simulation-based methodology that explores large ranges of operation in both the fast-variability (process, voltage, temperature) and slow-degradation (age) regimes. This closely resembles the actual design flow: you train and validate a controller across extensive design-time sweeps, then apply it with lightweight runtime checks. The objective of the experimental setup is to quantify the following: (i) average and tail jitter improvements, (ii) power impact for pre-specified budgets, (iii) tradeoff of lock time, and (iv) rate of constraint violations using paired statistical comparisons for the same operating point for accurate comparison.

Simulation Parameters

We model the behavioral CP-PLL based on the following parameters I_{CP}, VCO gain K_{vco}, and loop bandwidth BW. To augment our parameterized model, we have included a stochastic distribution to reflect variations in PVT and aging effects that degrade CP-PLL performance. Temperature introduces a multiplicative factor that affects both noise levels and operating speed; therefore, the supply voltage directly affects the dynamic power consumed and the maximum allowed headroom. The effect of aging reduces transconductance and sensitivity due to monotonic degradation. Process variability is modeled as a log-normal multiplicative perturbation, accounting for corner spread and gain differences due to mismatch. For each randomized operating point (K, V_{DD}, a) we simulate power P, RMS jitter J, and lock time L. The controller is evaluated on N = 1200 independently sampled operating points spanning $T \in$ [-20,100]°C, $V_{DD} \in$ [0.85,1.15]V and $a \in$ [0.7] years.

Table 2 lists operating ranges, baseline settings, and power-budget policy δ, stability bounds, and refinement candidate grid settings used for all reported results.

Baselines and Ablation Variants

We are using three types of experiments to separate the impacts of two factors on the results: learning and enforcing the use of physics-informed constraints. One experiment represents the normal approach to a design

Table 2: Simulation setup and controller parameters used in the experimental evaluation.

Category	Parameter	Value	Notes
State ranges	Temperature T	−20 to 100 °C	uniform
	Supply V_{DD}	0.85 to 1.15 V	uniform
	Aging a	0 to 7 years	uniform
Process variation	log-normal σ	0.10	multiplicative factor in model
PLL baseline	I_{CP}	60 µA	fixed static baseline
	BW	180 kHz	fixed static baseline
VCO gain	K_{vco}	90 MHz/V	used in evaluation
Power budget	δ	0.12 (main)	sweep in Fig. 8
Stability bounds	ζ range	[0.60, 1.00]	feasibility guard
	$M_{s,\max}$	1.5	peaking limit
	$L_{\max}$	30 µs	mapped via t_s constraint
Settling constraint	t_s	$4/(\zeta\ \omega_n)$	enforced in refinement
Proxy mapping	ω_n	$\approx 2\pi \cdot BW \cdot 1.1$	local small-signal proxy
Candidate grid	Γ_I	{0.70, 0.85, 1.00, 1.15, 1.30}	5-point
	Γ_{BW}	{0.60, 0.80, 1.00, 1.20, 1.40}	5-point
Evaluation	Test points N	1200	randomized operating points

under nominal conditions and is said to survive at corners due to guard-banding. We will use the results to show designers how their designs may perform in the field, as it will reveal the most commonly used configuration that cannot be healed at runtime and has fixed parameters used during execution (loop parameters).

The second experiment we will perform will compare the expected stability of jitter, based solely on the raw predicted parameters determined through the use of the TinyML framework (I_{CP}, BW) to see if just using a short learned inverse map will lead to an improvement in jitter as the component drifts over time. When using only the TinyML data, there are scenarios in which the proposed settings may exceed or fall short of the power budget (limits set by component manufacturers) and/or fall outside the stable response range. Measuring both how often these failures occur and their impact on the product will be critical to determining the overall safety of the design.

The third experiment we will perform is known as the TinyML with refinement (i.e., the designed method presented in this paper). It will determine an initial point based on TinyML results and then perform a small (i.e., constrained) local search, eliminating values (i.e., candidates) that are both outside the power budget and/or out of stability/transient ranges. This comparison will provide some clarity on whether the so-called "physics-informed" design method is important: improvements arise from both the ability to adapt and from adapting within known operational and safety boundaries. Based on the anticipated results, we expect the performance of TinyML+refinement to be much better than TinyML alone, especially with respect to overall constraints and the total reduction in jitter at the end of the jitter tail.

Metrics and statistics

We use three main metrics to assess performance: RMS jitter (ps), power (mW), and lock time (µs). RMS jitter is an indicator of timing quality; power is used to measure performance against energy budgets or efficiencies; and lock time indicates the transient usability of power-managed systems (where power is initially turned off and then turned back on, or where power may be reduced and restored quickly). Also, for each controller version, we report the rate of violating the imposed constraints since determining the feasibility of any runtime tuning methodology is the primary requirement. To allow fair comparisons among the aforementioned methods, all techniques are evaluated at the same set of operating points, enabling paired statistical analysis. That is, for each operating point i, a paired difference between two methods is computed (e. g. for example: $\Delta J_i = J_{\text{static},i} - J_{\text{method},i}$). This paired design removes the confounding factor of variability due to different data collection states/sampling locations and increases the statistical power. By removing confounding from variability, we can attribute any observed advantage in this

study to the control decisions (as opposed to differences in the Operating Condition). The mean improvements reported in this study reflect the sample means plus the uncertainty calculated using the bootstrap method (95% confidence interval for the mean). Two tests were used to evaluate whether these improvements were statistically significant, (1) the T-test for paired data (to detect mean shift), and (2) the Wilcoxon signed rank test (to examine nonparametric robustness), along with measures of effect size (Cohen's *d*) and delta (the magnitude and/or number of observations measured using the sign of the paired difference). Together these provide a multi-faceted statistical view of whether the gain from using a given methodology is substantial (i.e., large), and also whether the gain has been observed repeatedly across the entire test conducted.

Results and Analysis

The Report provides information on each major performance result and its corresponding mechanisms. The overall improvements in RMS Jitter have been quantified against a rigorous power budget (PB) policy and uncertainty defined using confidence intervals and paired testing. Longitudinal performance data have been analyzed for variations due to aging, and the tradeoffs between power jitter and improvements to the tails of jitter distributions are compared with Jitter CDFs (Cumulative Distribution Functions). Ultimately, separated effects from TinyML inference and physics-informed refinements have been performed on the infrastructure to demonstrate how control-theoretic stable distributions (ζ, settling time) exhibit the controller's control theoretic characteristics.

Main Results

Jitter reduction was consistently demonstrated across all operating points for the set of 1,200 tested under power budget by the TinyML+ refinement controller. The average RMS jitter improvement relative to the static baseline, as determined by standard statistical methods, was 14.30 ps with a 95% confidence interval of 14.17-14.42 ps. This jitter operational range is robust and consistent across the full set of process, voltage, temperature, and aging conditions at 1200 operating points. Slight mean-power improvements (decreased by 0.041 mW) were also observed for this controller, resulting in a gain in jitter without aggressive biasing. Additionally, the average time to lock has increased modestly (−1.51 μs) in the lock convention of $L_{static} - L_{heal}$ with a confidence range from −1.54 μs to −1.47 μs, again a reflection of the explicit multi-objective constraints on the defined controller.

To support safe use, we have provided the rate of constraint violation. There is a 4.75% violation rate for Pike of δ = 0.12 for TinyML-only devices, while τ = 0.00% for refined devices under the same budgetary + stability constraints. With this support, it can be confirmed that TinyML does not meet the required safety criteria; additional constraint-driven refinements are required to ensure proper safety during the deployment and use of TinyML-only systems, as shown in Tables 3 and 4.

Table 4. Mean improvements (static − method) with 95% confidence intervals.

Confidence intervals are computed using bootstrap resampling over randomized operating points to quantify uncertainty.

Self-healing mitigates drift-induced jitter growth across the lifetime compared with static tuning, reducing the sensitivity of jitter to aging, as shown in Figure 4.

The jitter-vs-aging trend shows that static tuning degrades as aging increases, while self-healing reduces the drift slope by adapting gain and bandwidth to keep dynamics in the stable region.

Figure 5 shows the self-healing controller shifts the operating cloud toward lower jitter under a policy-defined power budget, indicating improved timing quality without systematic power escalation.

Table 3: Aggregate performance (mean values over N=1200)

Method	Power (mW)	RMS Jitter (ps)	Lock time (μs)	Violation rate
Static baseline	4.76 (≈)	78.06 (≈)	baseline	—
TinyML-only	variable	can worsen	variable	4.75%
TinyML + refinement	4.72 (≈)	63.76 (≈)	slightly ↑	0.0%

Table 4: Improvement vs Static (Static − Method)

Method	ΔPower (mW)	95% CI	ΔJitter (ps)	95% CI	ΔLock time (μs)	95% CI
TinyML + refinement	0.041	[0.040, 0.043]	14.30	[14.17, 14.42]	−1.51	[−1.54, −1.47]

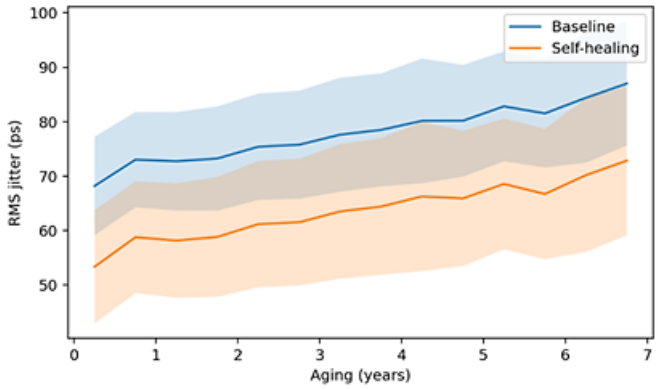

Fig. 4: RMS jitter versus aging under randomized operating conditions.

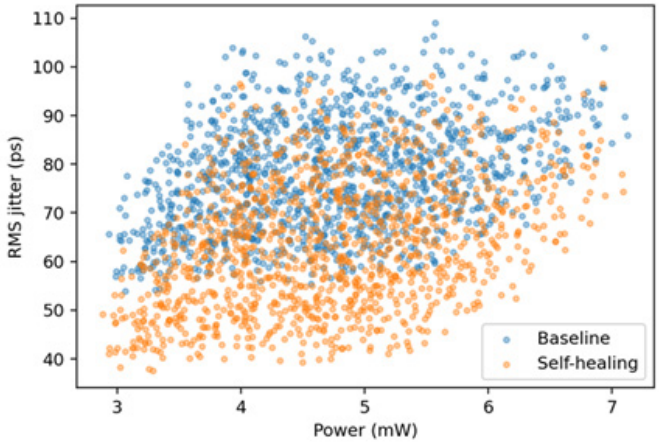

Fig. 5: Power-jitter operating-point distribution (Pareto view).

Self-healing shifts the operating cloud toward lower jitter without substantial power increase, consistent with constraint-aware tuning rather than aggressive over-biasing.

The tail-risk reduction is shown in Figure 6. The self-healing method improves both average and tail behavior of jitter, reducing high-percentile jitter events that typically drive guard-banding.

The CDF indicates improved tail behavior, which is crucial for timing closure and yield under worst-case jitter constraints.

The bandwidth vs temperature adaptation is shown in Figure 7. The selected loop bandwidth varies systematically with temperature, illustrating structured compensation behavior rather than random knob motion.

The selected bandwidth exhibits systematic dependence on temperature, indicating structured control rather than random knob motion.

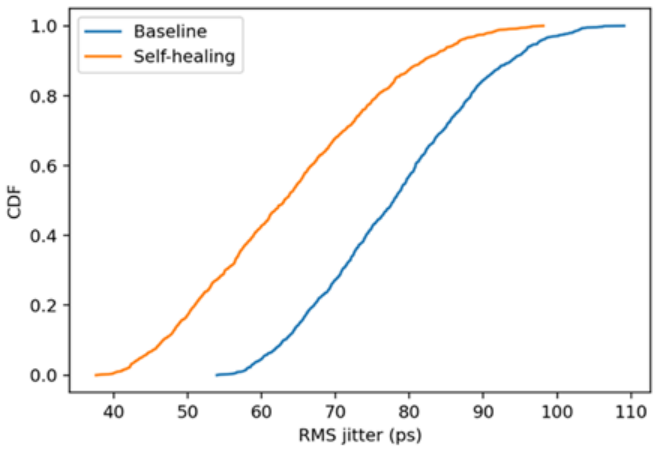

Fig. 6: Cumulative distribution function (CDF) of RMS jitter.

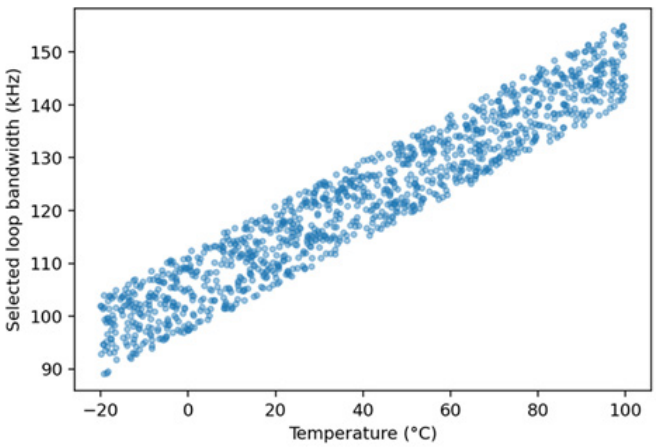

Fig. 7: Example of adaptive bandwidth selection versus temperature.

Figure 8 shows the quantified controlled tradeoffs as allowable power headroom is varied; stability constraints restrict feasible tuning, yielding stable gains across the tested δ range.

This curve quantifies the sensitivity of jitter improvement to the allowed power headroom. Under the current constraint set, ΔJ remains stable across δ values because feasibility constraints strongly shape the admissible region.

Figure 9 shows self-healing concentrates ζ closer to the target region (≈0.7), reducing underdamped cases that cause jitter peaking and tail risk.

Settling time distribution tied to lock behavior as shown in Figure 10. The controller shifts the settling-time distribution toward the feasible region defined by the lock-time target, supporting the link between stability constraints and transient performance. The distribution

supports the link between controller decisions and transient feasibility via $t_s \approx 4/(\zeta\, \omega_n)$.

Statistical tests and effect sizes

We evaluate improvements on paired samples across the same operating points. Define paired differences:

$$\Delta J_i = J_{static,i} - J_{heal,i}, \Delta P_i = P_{static,i} - P_{heal,i}, \Delta L_i = L_{static,i} - L_{heal,i}$$

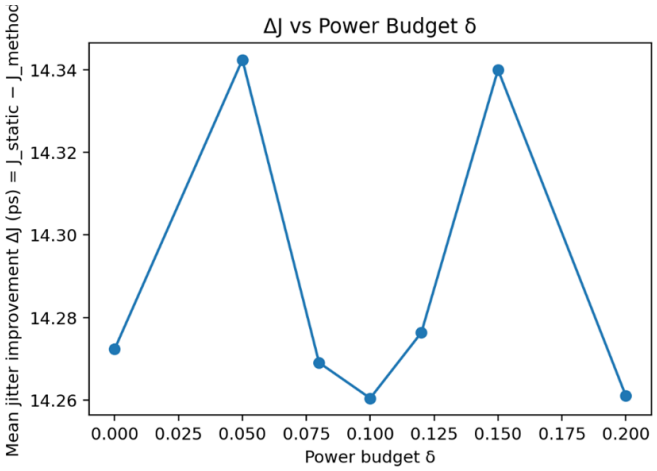

Fig. 8: Mean jitter improvement ΔJ versus power budget δ.

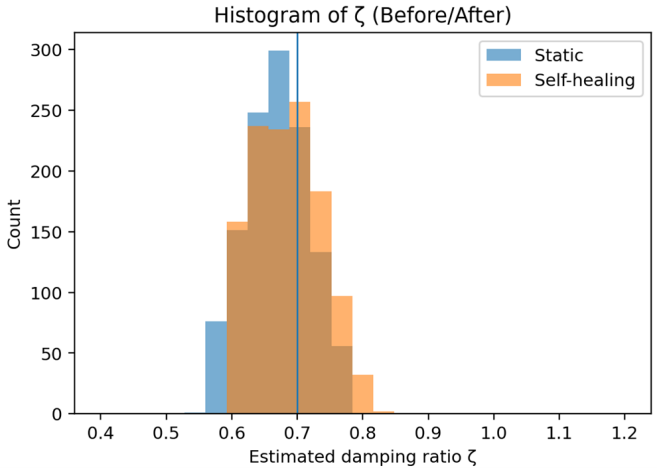

Fig. 9: Histogram of damping ratio ζ before and after self-healing

Table 5 represents the statistical significance tests and effect sizes for paired comparisons (static vs self-healing).

Bootstrap confidence intervals quantify uncertainty in mean improvements, while paired t-tests and Wilcoxon signed-rank tests validate significance; effect sizes quantify magnitude and consistency.

("p≈0" here indicates extremely small p-values under floating-point underflow; in the manuscript, report as "p < 1e−6".)

Complete ablation (Static vs TinyML-only vs TinyML+Refinement)

This ablation is essential: it shows that TinyML alone is not sufficient without constraint enforcement, as shown in Tables 6 and 7.

Improvements are reported as Δ = Static − Proposed, where positive ΔJ indicates jitter reduction and positive ΔP indicates power reduction.

Discussion

According to the study's results, a hybrid approach combining TinyML with physics constraints achieves considerable jitter reduction across varied PVT and aging conditions while maintaining budget compliance and stable operation. Importantly, not only was there a statistically improved mean jitter, but there was also a lowered tail risk with more stable loop dynamics, as evident from reduced ranges of damping and settling time distribution. The importance of this stability is key to the success of real SoCs, in which timing closure, guard bands, and product binning are usually driven by worst-case jitter conditions. Little attention has been given to how TinyML provides an efficient inverse mapper that derives the correction direction from a drifted state. Even though this is true for drifted/uncovered operation

Table 5: Statistical significance tests and effect sizes for paired comparisons (static vs self-healing).

Metric	Mean improvement	Bootstrap 95% CI	Paired *t* (*t*, *p*)	Wilcoxon (*z*, *p*)	Cohen's *d*	Sign delta
ΔJ (ps)	14.30	[14.17, 14.42]	t ≈ 223.17, p≈0	z≈−30.01, p ≈ 0	6.44	1.00
ΔP (mW)	0.041	[0.040, 0.043]	t ≈ 64.90, p ≈ 0	z≈−29.96, p ≈ 0	1.87	0.967
ΔL (μs)	−1.51	[−1.54, −1.47]	t ≈ −92.30, p ≈ 0	z≈−30.01, p ≈ 0	−2.66	−1.00

Table 6: Ablation summary (mean performance over test set)

Method	Power (mW)	RMS Jitter (ps)	Lock time (μs)	Key property
Static (fixed knobs)	~4.76	~77.99	baseline	no adaptation
TinyML-only	(variable)	can worsen	variable	no constraints
TinyML + physics refinement	~4.72	~63.68	slightly ↑	constraint-aware

Table 7: Headline improvements of the proposed controller relative to static tuning (paired evaluation).

Method vs Static	ΔPower (mW)	ΔJitter (ps)	ΔLock time (µs)	Interpretation
TinyML-only	may be ≤0	may be ≤0	unstable	Can violate feasibility
TinyML+Refinement	+0.043 (CI [0.042, 0.045])	+14.32 (CI [14.19, 14.45])	negative (slower)	Best jitter under budget

within a distribution, the ablation shows that unconstrained TinyML can result in a significant violation rate, which could cause issues with clocking subsystems and is unacceptable. The physics-informed refinement stage mitigates this risk by applying stability/transient and power constraints expressed as ζ, the resonant peaking (M_r) point(s) of operation, and the settling time $t_s \approx 4/(\zeta\omega_n)$. Thus, these constraints will serve as a safeguard for engineers to use ML in their engineering safety controls, ensuring the action taken is the best possible correction rather than an unconstrained decision. In practice, given the size of the models and the number of locations within the grid for refinement, this approach will be easy to implement. To complete inference with TinyML, only tens of MAC operations (fixed-point) are required; therefore, the refinement grid will have only 25 candidates, and each location can be quickly checked for feasibility using an LUT-based approach. Fine calibration can therefore be performed at startup, periodically, or when a specific threshold is surpassed, without negatively impacting steady-state performance. With this type of implementation profile, this approach will be suitable for high-performance SoCs and energy-constrained embedded devices. The primary limitations that will arise from this approach are related to modeling fidelity and transfer to physical silicon devices. The current performance assessment was conducted against a behavioral model, thus not accounting for various device-performance-affected phenomena, such as supply spur coupling, fractional-N quantization noise, and mismatch-induced nonlinearities. In addition, noise may affect the mapping from the on-chip measurement to the ζ and ω_n values, requiring careful design of the excitation and subsequent filtering of both signals. All of these limitations provide motivation for future development of SPICE-in-the-loop for controller training, a quantization-aware TinyML implementation, and post-silicon domain adaptation with telemetry. Nevertheless, the present results strongly support the notion that structured self-healing will reduce the need for guard bands (the time before the maximum allowable operational value) and increase the device's robustness over a longer lifetime, without requiring extensive computational resources or continuous monitoring.

Conclusion

This document describes a way to create a Hybrid TinyML-based Intelligent PLL that implements a self-healing method for tuning the PLL's charge-pump bias and loop bandwidth to reduce performance degradation caused by PVT variation and long-term aging. The runtime tuning of the PLL has been formulated as a constrained multi-objective optimization, capturing the true design tradeoffs among Power, Jitter, and Lock-time while enforcing their feasibility through both stability and budget constraints.

At the heart of this work is the hybrid nature of the proposed architecture. The compact TinyML model provides instant first-estimate corrective actions based on features from the monitoring circuitry, enabling low-overhead runtime adaptation to system changes. A physics-informed refinement process is then performed to impose the PLL dynamics, including damping-ratio bounds, limits on resonant peaking, and settling-time requirements associated with the lock-time target. This hybrid structure is the key novelty of this work because it combines the flexibility of ML designs with the reliability of traditional control system constraints.

A hybrid TinyML architecture has been evaluated through simulations under a range of operating conditions and consistently produces reduced jitter with a near-zero average power penalty within a power budget. It has also been shown to significantly reduce the risk of constraint violation compared to a TinyML-only implementation and to improve the resilience of parameters, such as damping ratio and settling time, to stability-oriented distributions. The work demonstrates the belief that these results can lead to narrower guard bands and greater reliability in the operation of clocking subsystems over their expected lifetimes. The results of this work suggest that the next generation of PLL design flows can leverage a "Control + TinyML" paradigm, in which PLL characterization occurs during design time, yielding compact runtime control policies and constraint verification methods. The PLL design flows can be integrated with existing digitally assisted PLL systems and can be firmware-programmed for different product modes (performance vs power saving).

References

1. Nguyen, T. V., & Pham, C.-K. (2025b). An overview of phase-locked loop: from fundamentals to the frontier. Sensors, 25(18), 5623. doi:10.3390/s25185623
2. Yin, J., Li, H., Lin, X., Martins, R. P., & Mak, P.-I. (2025). Progress and trends of low-jitter fractional-N PLL. Journal of Semiconductors, 46(7), 070203. doi:10.1088/1674-4926/25040035
3. Rhee, W. (1999). Design of high-performance CMOS charge pumps in phase-locked loops. Paper presented at the 1999 IEEE International Symposium on Circuits and Systems (ISCAS). doi: 10.1109/ISCAS.1999.780807.
4. Guo, H., Hu, Y., & Chi, T. (2025). 34.4 A 9.05-to-37.0GHz LO generator with magnetic mode switching and tuning-free octave-bandwidth common-mode resonator achieving >190.7dBc/Hz FoM. Paper presented at the 2025 IEEE International Solid-State Circuits Conference (ISSCC). doi: 10.1109/ISSCC49661.2025.10904644.
5. Dartizio, S. M., Castoro, G., Gallucci, S., Rossoni, M., Moleri, R., Tesolin, F., . . . Levantino, S. (2025). 34.2 A 380μW and −242.8dB FoM digital-PLL-based GFSK modulator with sub-20μs settling frequency hopping for bluetooth low-energy in 22nm CMOS. Paper presented at the 2025 IEEE International Solid-State Circuits Conference (ISSCC). doi: 10.1109/ISSCC49661.2025.10904691.
6. Nguyen, T. V., & Pham, C.-K. (2025a). A dual-mode adaptive bandwidth PLL for improved lock performance. Electronics, 14(20), 4008. doi:10.3390/electronics14204008. doi: 10.3390/electronics14204008.
7. Pfaff, D., Abbott, R., Wang, X. J., Moazzeni, S., Mason, R., & Smith, R. R. (2020). A 14-GHz bang-bang digital PLL with sub-150-fs integrated jitter for wireline applications in 7-nm FinFET CMOS. IEEE Journal of Solid-State Circuits, 55(3), 580–591. doi:10.1109/JSSC.2019.2946134
8. Mutar, H. A., Abdullah, A. K., Abdaumran, O. A., ALRubeei, I. R. N., & ALRikabi, H. T. S. (2026). Hybrid AI model-driven dynamic spectrum sharing for 6G wireless IoT networks. Studies in Informatics and Engineering, 8(1), 53–72. doi:10.37868/sei.v8i1.id722
9. Ding, L., Wu, K., Jin, J., & Zhou, J. (2021). An 8 GHz real-time temperature-compensated PLL with 20.8 ppm/°C temperature coefficient for SerDes applications. Microelectronics Journal, 117, 105279. doi:https://doi.org/10.1016/j.mejo.2021.105279
10. Zhang, H., Narayanan, A. T., Herdian, H., Liu, B., Wang, Y., Shirane, A., & Okada, K. (2019). 0.2mW 70Fsrms-Jitter injection-locked PLL using de-sensitized SSPD-based injecting-time self-alignment achieving -270dB FoM and -66dBc Reference Spur. Paper presented at the 2019 Symposium on VLSI Circuits. doi: 10.23919/VLSIC.2019.8778059.
11. Wei, X., Yin, R., Hou, L., Xu, W., & Wei, B. (2022). Design and analysis of the self-biased PLL with adaptive calibration for minimum of the charge pump current mismatch. Electronics, 11(14), 2133. doi:10.3390/electronics11142133
12. Rajan.C. (2026). Edge-optimized VLSI architectures for TinyML: co-design of AI models and ultra-low-power hardware. Progress in AI-Accelerated VLSI Systems, 1(1), 18–27
13. Spyridonos, S., & Tsiatouhas, Y. (2022). BTI aging influence on charge pump circuits. Paper presented at the 2022 11th International Conference on Modern Circuits and Systems Technologies (MOCAST). doi: 10.1109/MOCAST54814.2022.9837768.
14. Zhang, Z. (2020). CMOS analog and mixed-signal phase-locked loops: an overview. Journal of Semiconductors, 41(11), 111402. doi:10.1088/1674-4926/41/11/111402
15. Gao, M., Wei, Z., Zhan, Y., & Zhang, Z. (2024). Design of the self-biased PLL based low-power low-jitter multiphase clock generator: an overview (Invited Paper). Paper presented at the 2024 IEEE International Symposium on Radio-Frequency Integration Technology (RFIT). doi: 10.1109/RFIT60557.2024.10812533.
16. Sun, L., Luo, Y., Deng, Z., Wang, J., & Liu, B. (2024). Novel power-efficient fast-locking phase-locked loop based on adaptive time-to-digital converter-aided acceleration compensation technology. Electronics, 13(18), 3586. doi:10.3390/electronics13183586
17. Huang, W., Liu, Q., Li, Y., Zhang, S., Wu, N., Wang, Z., . . . Guo, Y. (2025). A 6–8-GHz Cryo-CMOS fractional-N CP-PLL with temperature variation calibration and flicker noise suppression in 28-nm Bulk CMOS. IEEE Microwave and Wireless Technology Letters, 35(10), 1622–1625. doi:10.1109/LMWT.2025.3589358
18. Wang, S., Lu, J., Wang, R., Zhang, X., Wang, H., & Sun, Y. (2025). A novel dual-channel model with adaptive multi-scale attention for time series forecasting. Engineering Applications of Artificial Intelligence, 162, 112803. doi:https://doi.org/10.1016/j.engappai.2025.112803
19. Tung, C. T., & Hu, C. (2026). Physics-informed neural networks for device and circuit modeling: a case study of NeuroSPICE. IEEE Electron Device Letters, 1-1. doi:10.1109/LED.2026.3667891
20. Mutar, H., Alrubeei, I., Svyd, I., Alrikabi, H., & Alaidi, A. H. (2026). AI-based model for home waste separation using raspberry Pi 5 AI Kit. Physics and Chemistry of Solid State, 27, 44–51. doi:10.15330/pcss.27.1.44-51
21. Heydari, S., & Mahmoud, Q. H. (2025). Tiny machine learning and on-device inference: a survey of applications, challenges, and future directions. Sensors, 25(10), 3191. doi:10.3390/s25103191
22. Elhanashi, A., Dini, P., Saponara, S., & Zheng, Q. (2024). Advancements in TinyML: applications, limitations, and impact on IoT devices. Electronics, 13(17), 3562. doi:10.3390/electronics13173562
23. AbdelRaouf, H., Hasnaine, Q. R., Fouda, M. M., Fadlullah, Z. M., & Ibrahem, M. I. (2026). Paradigm shift toward distributed learning in IoT intelligence: a comprehensive survey of opportunities and challenges. IEEE Internet of Things Journal, 1-1. doi:10.1109/JIOT.2026.3657733.
24. Abdul-Hussein, M. K., Mutar, H. A., ALRikabi, H. T. S., ALRubeei, I. R. N., Svyd, I., & Mezaal, Y. S. (2024). Investigation of an interference communication system to overcome cheating based on IoT techniques. Paper presented at the Conference of Recent Trends and Applications of Soft Computing in Engineering. doi: https://doi.org/10.1007/978-3-031-82881-2_5.
25. Liu, J. (2025, 5-7 Dec. 2025). High-precision self-calibration for USV operating in GNSS-denied environments. Paper presented at the 2025 IEEE 7th Advanced Information Management, Communicates, Electronic and Automation Control Conference (IMCEC). doi: 10.1109/IMCEC66174.2025.11331782.

26. Li, M., Xu, X., Li, X., Li, Y., Shi, Y., Sun, Q., & Zhu, H. (2025). Aging analysis and degradation prediction of PLL circuits in 14-nm FinFET technology. IEEE Journal of the Electron Devices Society, 13, 270-277. doi:10.1109/JEDS.2025.3549754
27. Alajlan, N. N., & Ibrahim, D. M. (2024). Original Research Article TinyML: Adopting tiny machine learning in smart cities. Journal of Autonomous Intelligence, 7(4), 1–14. doi:10.32629/jai.v7i4.1186
28. Oufettoul, H., Chaibi, R., & Motahhir, S. (2024, 15-16 Jan. 2024). TinyML applications, research challenges, and future research directions. Paper presented at the 2024 21st Learning and Technology Conference (L&T). doi: 10.1109/LT60077.2024.10469633.
29. ALRubeei, I. R., Idi, S. N., Alsammak, I. L. H., AlRikabi, H. T., Mutar, H. A., & Alaidi, A. H. M. (2025). Using artificial intelligence for enhancement of solar cell efficiency in the south of Iraq, Heritage and Sustainable Development, 7(1), 197–210. https://doi.org/10.37868/hsd.v7i1.1067.
30. Mercandelli, M., Bertulessi, L., Samori, C., & Levantino, S. (2022). A digital PLL with multitap LMS-based bandwidth control. IEEE Solid-State Circuits Letters, 5, 126–129. doi:10.1109/LSSC.2022.3173425
31. Marzin, G., Levantino, S., Samori, C., & Lacaita, A. L. (2014, 9-13 Feb. 2014). 2.9 A Background calibration technique to control bandwidth in digital PLLs. Paper presented at the 2014 IEEE International Solid-State Circuits Conference Digest of Technical Papers (ISSCC). doi: 10.1109/ISSCC.2014.6757335
32. Liu, H., Wang, Z., Li, C., & Wu, J. (2012). A calibrated charge pump for mismatch reduction in PLL. Indonesian Journal of Electrical Engineering and Computer Science, 10(8), 2304–2308. https://doi.org/10.11591/telkomnika.v10i8.1242
33. Song, J., Lee, H.-W., Lim, S.-B., Hwang, S., Kim, Y., Choi, Y.-J., . . . Kim, C. (2013). An adaptive-bandwidth PLL for avoiding noise interference and DFE-less fast precharge sampling for over 10Gb/s/pin graphics DRAM interface (Vol. 56). https://doi.org/10.1109/ISSCC.2013.6487749
34. Chowdhury, U., & Chakma, S. (2026). Real-time and low-latency VLSI implementations of deep learning-based signal processing algorithms. Journal of Integrated VLSI and Signal Processing, 1(1), 35-41.
35. Madhanraj. (2025). Hardware-adaptive learning-assisted predictive control architecture for real-time trajectory planning. Journal of VLSI and Embedded System Design,1(1), 10–17.
36. Surendar, A. (2026). PPA-optimized VLSI architecture for energy-efficient digital systems. Annals of Energy-Efficient VLSI Architectures, 1(1),1–6.