On-Chip Runtime Frequency Feedback Closed-Loop Scheme for Low-Power Management

JIANPENG SUN & ZHAN TONG
MASTER´S THESIS
DEPARTMENT OF ELECTRICAL AND INFORMATION TECHNOLOGY
FACULTY OF ENGINEERING | LTH | LUND UNIVERSITY

On-Chip Runtime Frequency Feedback Closed-Loop Scheme for Low-Power Management

Jianpeng Sun
Zhan Tong

Department of Electrical and Information Technology
Lund University

Supervisor: Baktash Behmanesh (academic supervisor)
Forest Yu (industrial supervisor)

Examiner: Pietro Andreani

June 4, 2026

Printed in Sweden
Tryckeriet i E-huset, Lund

Abstract

This thesis studies a runtime frequency feedback method for low-power digital systems. Conventional low-power designs usually reserve extra voltage and timing margins to ensure stable chip operation under process, voltage, temperature, and aging variations. However, fixed margins can also lead to unnecessary power consumption. To address this problem, this thesis proposes and implements a Runtime Frequency Feedback system. The system observes the current silicon speed through a VCRO-based frequency sensing path and provides this information to later low-power feedback control logic. The thesis covers behavioral modeling, system design, RTL circuit implementation, and functional verification. The results show that the proposed design can provide a practical hardware basis for sensing runtime timing capability, reducing conservative design margins, and supporting more adaptive low-power management.

Popular Science Summary

Chips are used in phones, computers, cameras, cars, and many smart devices. People expect chips to be fast, stable, and energy efficient. To prevent errors caused by temperature changes, manufacturing differences, or aging, engineers usually keep a large safety margin in chip design. This is similar to keeping a long safety distance when driving. It is safer, but it can also waste energy when the margin is larger than needed.

This thesis asks whether a chip can observe its own condition while it is running, instead of always working under the most conservative assumption. To do this, the thesis designs a Runtime Frequency Feedback system. In simple terms, it works like a small speed sensor inside the chip. It helps the system estimate how fast the chip can currently operate. If the chip is in a good condition, there may be room to reduce unnecessary power use. If the condition becomes worse, the system can detect the risk earlier.

This thesis includes modeling, hardware design, and simulation-based verification. The work studies how to calibrate this “speed sensor,” how to measure its output, and how to provide the measured result to later low-power control logic. Overall, this thesis aims to provide a practical hardware basis for smarter and more adaptive low-power management in future chip systems.

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Power Challenges in Modern SoCs

For decades, semiconductor scaling following Moore's Law delivered exponential growth in transistor density, while Dennard scaling ensured that power density remained approximately constant as feature sizes shrank. This favorable trend ended around the 130–90 nm technology node in the early 2000s: as gate oxide thickness approached atomic dimensions and threshold voltages could no longer be reduced proportionally, supply voltage scaling stalled and power density began to rise with each new generation [1]. The breakdown of Dennard scaling fundamentally changed the landscape of integrated circuit design, making power dissipation rather than transistor count the primary constraint on system performance. As illustrated in Figure 1.1, although transistor counts have continued to grow exponentially, clock frequency and typical power dissipation have plateaued since the mid-2000s, forcing the industry to shift toward multi-core architectures and power-aware design methodologies.

The practical consequences of this power challenge are visible across all market segments. In data centers, electricity for computing and cooling accounts for a substantial fraction of total operating cost; global data center energy consumption was estimated at 200–250 TWh per year in 2020, with projections of continued rapid growth driven by cloud computing and artificial intelligence workloads [3]. In mobile devices, the thermal design power of an application processor is constrained to 3–5 W by the absence of active cooling, forcing designers to trade peak performance for energy efficiency. Even in high-performance processors, power budgets have plateaued: modern server CPUs operate at TDPs of 200–350 W, a range that has remained

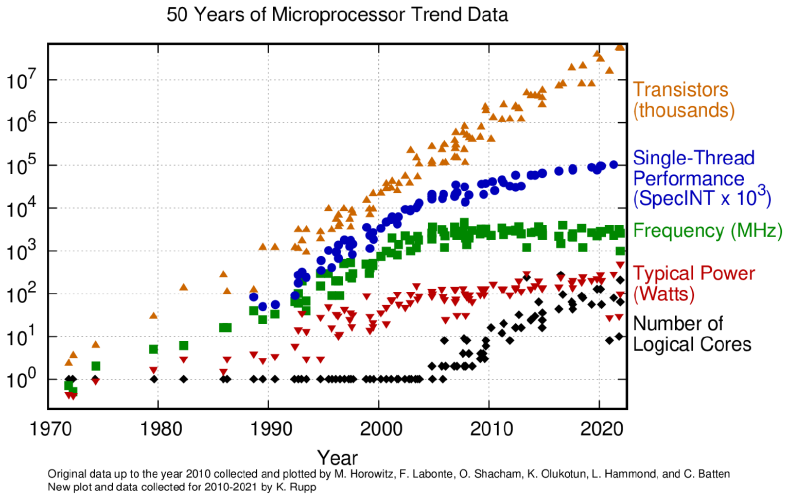


Figure 1.1: 50 years of microprocessor trend data. Transistor count continues to grow exponentially, while clock frequency and typical power have stagnated since the mid-2000s due to the breakdown of Dennard scaling, driving the shift to multi-core designs. Data compiled by K. Rupp [2].

largely flat over recent generations despite significant increases in core count and transistor budget. In all these domains, the ability to deliver higher performance within a fixed power envelope depends critically on the efficiency of on-chip power management.

1.1.1 CMOS Power Dissipation

The total power consumption P_{total} of a CMOS circuit can be decomposed into three components [4]:

$$P_{\text{total}} = P_{\text{dynamic}} + P_{\text{short-circuit}} + P_{\text{leakage}}. \tag{1.1}$$

The dominant term in most digital circuits is the *dynamic* (switching) power, consumed when load capacitances are charged and discharged during logic transitions:

$$P_{\text{dynamic}} = \alpha C V_{DD}^2 f, \tag{1.2}$$

where α is the switching activity factor, C is the total switched capacitance, V_{DD} is the supply voltage, and f is the clock frequency. The

quadratic dependence on V_{DD} makes voltage reduction the single most effective lever for power savings.

However, reducing V_{DD} has a direct impact on circuit speed. The propagation delay of a CMOS gate is approximated by

$$t_d \propto \frac{C \cdot V_{DD}}{(V_{DD} - V_{th})^{\gamma}}, \tag{1.3}$$

where V_{th} is the threshold voltage and γ is a technology-dependent parameter typically between 1 and 2 [4]. As V_{DD} approaches V_{th}, the delay increases sharply, placing a practical lower bound on the supply voltage for a given target frequency.

Short-circuit power arises during input transitions when both PMOS and NMOS transistors conduct simultaneously, creating a momentary path from V_{DD} to ground. It can be approximated as [4]:

$$P_{\text{short-circuit}} = \beta \cdot t_{sc} \cdot V_{DD} \cdot f, \tag{1.4}$$

where β is a factor related to transistor sizing, and t_{sc} is the duration of the short-circuit current pulse during each transition. This component is typically 10–15% of P_{dynamic} in well-designed circuits and is minimized by matching input and output transition times.

Leakage power is the static dissipation that flows even when a circuit is idle and no switching occurs. It is modeled as:

$$P_{\text{leakage}} = V_{DD} \cdot (I_{\text{sub}} + I_{\text{gate}}), \tag{1.5}$$

where I_{sub} is the subthreshold conduction current that flows between source and drain when the transistor is nominally off, and I_{gate} is the gate oxide tunneling current that penetrates the thin insulator between gate and channel.

1.1.2 The Dark Silicon Challenge

Together, the combined pressure of dynamic and static power has led to the phenomenon known as *dark silicon*: at each new technology node, the fraction of transistors that can be simultaneously active within the thermal design power envelope decreases [5]. Projections have indicated that at 8 nm, more than 50% of the chip area may need to remain powered off at any given time. This fundamental scaling barrier has motivated a broad range of power management techniques, from circuit-level optimizations to system-level adaptive strategies.

1.2 Power Reduction Techniques

The power equations introduced above reveal several avenues for reducing consumption. From Equation 1.2, the dynamic power $P = \alpha C V_{DD}^2 f$ can be lowered by reducing any of its four factors. Each approach operates at a different level of the design hierarchy and carries distinct tradeoffs:

- **Reducing switching activity (α) — Clock gating:** Disabling the clock to idle functional units eliminates their switching power entirely. This is the most widely used power optimization in synchronous digital design: modern synthesis tools can insert clock gating automatically, achieving significant dynamic power reduction with minimal area overhead [6]. However, clock gating does not address leakage power, since all transistors remain powered on.
- **Reducing capacitance (C) — Architecture and physical design:** Choosing smaller transistors, shorter interconnects, and more efficient architectures reduces the total switched capacitance. While fundamental, this approach is largely determined at design time and offers limited runtime flexibility.
- **Reducing voltage (V_{DD}) — Voltage scaling:** Voltage reduction benefits all three power components. Dynamic power decreases quadratically ($P_{\text{dynamic}} \propto V_{DD}^2$), so a 10% reduction in V_{DD} translates to approximately 19% less dynamic power. Short-circuit power and leakage power also decreases according to Equation 1.4 and Equation 1.5. This makes voltage scaling the single most effective lever for total power reduction. Modern SoC design exploits this method: multi-voltage domain is designed to split the SoC into independent voltage island, each operating at a supply matched to its performance requirement [7].
- **Reducing frequency (f) — Frequency scaling:** Lowering the clock frequency reduces dynamic power linearly. In practice, frequency reduction is almost always combined with voltage reduction, since a lower frequency relaxes timing constraints and permits a lower supply voltage.

For leakage power, the primary technique is **power gating**: completely disconnecting idle blocks from the supply rail using high-V_{th} header or footer switches [8]. This can reduce leakage by orders of magnitude, but introduces wake-up latency and requires isolation and

state-retention logic, adding design complexity.

Nowadays, **Dynamic Voltage and Frequency Scaling (DVFS)** has emerged as the most impactful runtime power management technique. By adjusting both V_{DD} and f together based on workload demand, DVFS exploits the cubic relationship between power and voltage-frequency to deliver significant energy savings while maintaining throughput when needed. However, as will be discussed in the following section, conventional DVFS requires substantial voltage guardbands that limit the achievable savings, motivating the adaptive closed-loop approaches that are the focus of this thesis.

1.3 From DVFS to Adaptive Voltage Scaling

1.3.1 DVFS Mechanism

As established in the previous section, simultaneously reducing the supply voltage and clock frequency is the most effective runtime strategy for power savings. Dynamic Voltage and Frequency Scaling (DVFS) formalizes this idea by defining a set of discrete *Operating Performance Points* (OPPs), each specifying a validated (V_{DD}, f) pair. A hardware or software controller monitors the current workload and thermal conditions and selects an appropriate OPP: high-performance points for demanding tasks and low-power points during idle or light-load periods.

The transition between OPPs must follow a strict protocol to avoid timing violations. When scaling up to a higher performance point, the voltage must be raised *before* the frequency is increased, ensuring that the supply is sufficient for the faster clock. Conversely, when scaling down, the frequency is reduced first, and the voltage is lowered afterward. The cooperation between the clock generator and the voltage regulator, is the fundamental of DVFS implementation, which makes fast and accurate observation of the circuit state essential.

DVFS is widely deployed in modern processors and system. For example, Intel's Enhanced SpeedStep and Turbo Boost technologies rely on DVFS to balance performance and power across cores [9]. At the system level, operating systems such as Linux implement DVFS governors (e.g., ondemand, performance, and powersave) that adjust CPU frequency, or equivalently the operating performance point (OPP), in response to CPU utilization [10].

1.3.2 The Voltage Guardband Problem

Despite its effectiveness, conventional DVFS has a fundamental limitation: the voltage assigned to each OPP must guarantee correct operation under *worst-case* conditions. Chip designers insert a *voltage guardband* — an additional margin above the minimum V_{DD} required at the target frequency — to account for multiple sources of uncertainty:

- **Process variation:** Die-to-die and within-die variations in threshold voltage, channel length, and interconnect resistance lead to variations in circuit delay. Consequently, chips from different process corners may require different supply voltages to sustain the same operating frequency.
- **Voltage droop:** Transient current demand from switching logic can cause voltage fluctuations in the power delivery network, temporarily lowering the effective supply voltage at the logic gates.
- **Temperature variation:** Changes in junction temperature affect circuit delay and therefore the minimum voltage needed to sustain a target frequency. Because thermal conditions vary during operation, extra voltage margin is often required to maintain timing safety.
- **Aging:** Bias Temperature Instability (BTI) and Hot Carrier Injection (HCI) progressively degrade transistor performance over the product lifetime, requiring additional margin to guarantee end-of-life operation [11].

Conventional worst-case design often requires a substantial cumulative voltage guardband to ensure correct operation under unfavorable conditions [12]. In practice, the cumulative guardband can reach 10–20% of the nominal supply voltage [13]. Since power scales quadratically with voltage, this margin alone accounts for 20–35% of excess dynamic power consumption. Critically, this guardband is static: it is designed for the worst-case combination of all the above factors, which rarely occurs simultaneously during normal operation. A typical chip under nominal conditions has far more timing margin than the guardband assumes, representing a substantial waste of energy.

Furthermore, because DVFS operates on discrete OPPs, it cannot track continuous variations within a single operating point. For instance, if the junction temperature drops during a lightly loaded period, the actual silicon speed increases, and the current OPP voltage

becomes unnecessarily high — but a conventional DVFS governor has no mechanism to exploit this opportunity.

1.3.3 Adaptive Voltage and Frequency Scaling

The limitations of static guardbands and discrete OPPs motivate a closed-loop approach. Adaptive Voltage Scaling (AVS) incorporates on-chip sensors that continuously estimate the actual timing margin or silicon speed, allowing the system to reduce the supply voltage until the margin reaches a target threshold. The guardband is no longer a fixed worst-case value but a dynamically tracked quantity that adapts to the current process, voltage, and temperature (PVT) conditions [13].

Adaptive Voltage and Frequency Scaling (AVFS) extends AVS further by simultaneously adjusting both voltage and frequency in the closed loop. Instead of merely trimming the voltage at a fixed OPP, an AVFS system continuously tracks the voltage-frequency relationship and makes fine-grained adjustments within an operating point, as well as informing OPP transitions [14, 15]. This enables the system to reclaim the majority of the static guardband, which translates directly to significant power savings across the chip.

The key enabler of any AVFS system is a reliable on-chip mechanism that can observe the actual silicon speed in real time and feed it back into the operating-point decision. This is the role of the on-chip Runtime Frequency Feedback (RFF) closed-loop scheme studied in this thesis: by combining a Voltage-Controlled Ring Oscillator (VCRO), a digital frequency measurement engine, and dedicated configuration and reporting paths, the RFF scheme produces a continuous feedback signal that closes the AVFS control loop.

1.4 The Runtime Frequency Feedback Scheme

To accurately observe the relationship between supply voltage and circuit speed in a given voltage domain, several monitoring approaches have been proposed in the previous study:

- **Critical path replicas and Razor-style monitors** detect timing errors or near-errors on actual or synthetic critical paths, providing direct feedback on timing margin [12]. While highly accurate, these approaches introduce design complexity and potential

metastability issues.

- **Ring oscillator-based monitors** use a VCRO whose frequency is a direct physical function of the local PVT conditions. By measuring the oscillation frequency, the system obtains a digital proxy for the current silicon speed in the monitored voltage domain [16]. This approach is simpler to integrate and provides a continuous, averaged measurement over the local region.

The Runtime Frequency Feedback (RFF) closed-loop scheme studied in this thesis adopts the second approach. Rather than replicating an existing critical path, the RFF scheme converts the local silicon speed into a digital code through a VCRO and a windowed frequency meter, and feeds this code into a programmable feedback path that adjusts the operating point. Within an SoC voltage domain, the scheme is intentionally scoped to four digital, register-mapped, and verifiable paths that together constitute the on-chip portion of the feedback loop:

1. **Configuration path:** Through the Advanced Peripheral Bus (APB), the host CPU programs the measurement window, the target frequency, the calibration policy, and the reporting content, and reads back the run-time status of the subsystem.
2. **Calibration path:** Before the VCRO output can be monitored, the VCRO frequency must be locked to the target frequency. This is accomplished through a feedback loop that feeds the VCRO output back to the control circuitry.
3. **Observation path:** the VCRO together with the windowed frequency meter, which counts edges over a programmable reference window and produces a frequency code with a quantifiable error budget.
4. **Reporting path:** the State-Interface-Communication (SIC) transmitter that exports the latched frequency code and run-time status to an upstream consumer through a back-pressure-tolerant handshake, isolating the inner control loop from external timing.

It is also important to clarify the scope of this thesis. This work does not include a complete on-chip power management unit, does not implement the full DVFS or AVFS policy stack, and does not by itself produce a post-silicon power figure. Instead, within a larger AVFS system, the RFF scheme functions as an observation-and-reporting node that provides a calibrated frequency code to an upstream adaptive controller. The controller combines this frequency information with

readings from other on-chip sensors and generates a unified voltage-adjustment request. This request is then processed by upstream circuitry, which analyzes all incoming requests and converts them into an analog control signal to change the power supply.

1.5 Thesis Scope

This thesis focuses on the behavior-level pre-evaluation, digital RTL implementation, and functional verification of the on-chip RFF closed-loop scheme:

1. **Scheme definition and behavior-level pre-evaluation:** A unified Baseline-versus-Feedback comparison framework is constructed at the behavior level under a single set of disturbances, statistical windows, and modeling assumptions. The pre-evaluation analyses error convergence, cumulative energy trend, and the parameter window in which the feedback scheme remains stable, and is used to decide whether the scheme is worth carrying into RTL implementation.
2. **Hardware design and implementation:** A practical hardware implementation of the on-chip digital paths of the RFF scheme, comprising the VCRO calibration engine, the windowed frequency meter, the SIC transmitter, the APB slave/master pair that exposes the register file and exports the latched frequency code, and the clock-domain-crossing synchronizers between the high-speed measurement domains (`wclk` / `vcro_clk`) and the bus domain (`pclk`).
3. **UVM functional verification:** A Universal Verification Methodology (UVM) testbench environment for the integrated RFF subsystem, including stimulus generation, reference-model-based checking, scenario-based coverage of the configuration, measurement, calibration, reporting, CDC, and reset paths, and explicit pass/fail criteria expressed as waveform-level assertions.

1.6 Main Contributions

The main contributions of this thesis are summarized as follows:

1. It defines the role of Runtime Frequency Feedback in a low-power management context. RFF is treated as a runtime frequency-capability observation and reporting node. Here, frequency capa-

bility refers to the maximum safe operating speed that the chip can support under the current runtime condition. RFF serves in this role rather than as the workload operating clock itself.

2. It builds a controlled Baseline-versus-Feedback behavioral model. The comparison uses the same target, disturbance profile, evaluation window, metric, and initial condition, so that the energy trend can be traced to the feedback mechanism.
3. It implements the digital RTL paths of the RFF subsystem, including APB configuration, VCRO calibration support, frequency measurement, status readback, and SIC payload reporting.
4. It verifies the implemented RTL at module level through directed cases, monitors, checkers, waveform evidence, coverage analysis, and explicit waiver reasoning.

1.7 Thesis Structure

The remainder of this thesis is organized as follows:

- **Chapter 2, System-Level Development and Behavioral Modeling of RFF:** Establishes the behavioral-level model used to project the power reduction contributed by the proposed system, and describes the top-level composition of the RFF subsystem and the data and control flow between its blocks.
- **Chapter 3, Hardware Design and RTL Implementation of RFF:** Presents the design of each sub-module of the RFF subsystem in turn: the VCRO, the calibration engine that locks it to a programmable reference, the windowed frequency meter, the SIC transmitter responsible for reporting, and the APB slave/master pair that connects the subsystem to the host bus.
- **Chapter 4, Functional Verification of RFF:** Details the UVM testbench architecture, test plan, coverage strategy, and simulation results obtained for the top-level RFF subsystem.
- **Chapter 5, Conclusion:** Summarizes the contributions of this work and identifies directions for future work.

Chapter 2

System-Level Development and Behavioral Modeling of RFF

2.1 Introduction to RFF System

As established in Chapter 1, the core idea behind AVFS/AVS power reduction is to dynamically tighten the voltage guardband at run time rather than fixing it at the worst-case value, thereby lowering power consumption. To achieve this, three elements are required: an on-chip proxy quantity that reflects the current silicon speed, a system that analyses the proxy and gives voltage adjustment command, and a closed loop that converts the command into an actual voltage adjustment action. The RFF subsystem is the component in the AVFS control loop responsible for the first of these three elements. It does not adjust the voltage itself; instead, it observes silicon speed in real time and provides the basis on which voltage regulation decisions are made.

VCRO as the silicon-speed proxy. At the heart of the RFF subsystem is a VCRO. The oscillation frequency of the VCRO is a composite physical function of V_{DD}, temperature, and process corner of the chip. First, the VCRO is calibrated to a reference frequency. As environmental conditions vary, the VCRO frequency may drift from its calibrated value, and this drift can be used to estimate the available guardband in the current voltage domain.

- high voltage, low temperature, and a fast process corner cause the VCRO to oscillate faster, indicating sufficient guardband margin for the monitored voltage domain;
- low voltage, high temperature, and a slow process corner cause the

VCRO to oscillate slower, indicating insufficient guardband margin for the monitored voltage domain.

Therefore, the VCRO frequency itself serves as the physical proxy for silicon speed. However, the AVSC controller cannot consume a raw high-speed clock directly. It requires a digital value that can be stored in a register and transmitted over a bus.

From analog frequency to digital code. The RFF subsystem is precisely the mechanism that converts the VCRO frequency into such a digital quantity. This conversion proceeds in three stages:

1. *Calibration.* At SoC initial power-up, the VCRO control word is calibrated to a software-specified target, establishing a known operating point for all subsequent measurements. Without this step the VCRO frequency would be an arbitrary number mainly reflecting uncontrolled PVT variation.
2. *Measurement.* Within each measurement window controlled by the CPU, edges of `vcro_clk` are counted, producing a 32-bit `freq_code`.
3. *Reporting.* The `freq_code` together with run-time status information is packed and delivered to the AVSC through the reporting interface.

Upon receiving `freq_code`, the AVSC compares it against the target silicon speed required by the current OPP:

- if `freq_code` is above the target, the silicon is running faster than necessary and the guardband margin is comfortable; the AVSC requests a V_{DD} decrease;
- if `freq_code` is below the target, the silicon speed is tight and the guardband margin is insufficient; the AVSC requests a V_{DD} increase.

The adjustment request is consumed by other units in the AVFS system, which physically change the chip supply voltage. The new voltage feeds back to the VCRO, causing its frequency to change accordingly. On the next measurement window the RFF subsystem produces a fresh `freq_code`, and the closed loop continues to operate in this manner.

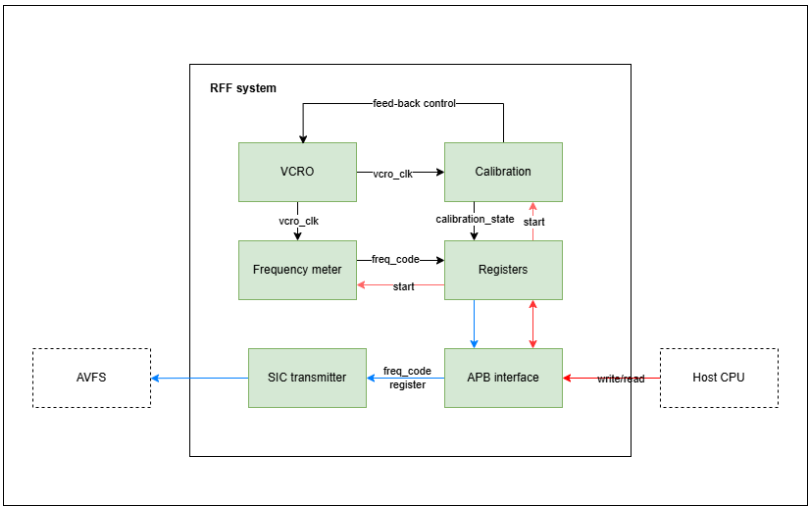


Figure 2.1: Top-level block diagram of the RFF subsystem showing the five sub-modules, and data/control paths.

2.2 Architecture of RFF System

The on-chip RFF subsystem consists of five modules and registers, as illustrated in Figure 2.1: a VCRO, a calibration engine, a frequency meter, a SIC transmitter, and an APB interface. They cooperate across three asynchronous clock domains and exchange information through dedicated cross-domain synchronizers. This section introduces all the modules and the data/control flows between them; Chapter 3 then revisits each block in implementation detail.

2.2.1 Sub-modules

- **VCRO.** Although the VCRO is a mixed-signal circuit in practical implementations, the focus of this work is on the implementation of the digital path of RFF rather than on the influence of PVT variations on VCRO performance. So the VCRO is modeled as a digitally programmable voltage-sensitive clock source that provides the high-speed clock `vcro_clk`.
- **Calibration engine (`vcro_calib`).** A digital frequency-locked loop that compares the edge count of `vcro_clk` over a programmable measurement window against a software-supplied target and adjusts the VCRO control word until convergence.

- **Frequency meter (`freqm`).** A windowed-measurement engine that measures the calibrated `vcro_clk` in a stable clock domain `pclk`, counting `vcro_clk` edges over a programmable number of `pclk` cycles to produce a 32-bit frequency code together with a valid flag.
- **APB slave / master (`rff_apb_slv` / `apb_mst`).** Two cooperating APB3 modules sharing the `pclk` domain. The slave exposes the complete register file of the subsystem to the host CPU and services both writes (configuration) and reads (status). The master, under firmware-configured policy, autonomously issues APB3 write transactions carrying the latched frequency code or inner states towards SIC transmitter.
- **SIC transmitter.**The reporting block. It receives the latched frequency code or inner states from APB master, exports it to an external consumer through a valid–ready handshake; The reporting policy can be modified by host CPU.

The mapping from these five blocks to the four functional paths is direct: the configuration path is the APB slave together with the register file; the observation path is the VCRO and the frequency meter; the feedback path is the calibration engine; and the reporting path is the SIC transmitter.

2.2.2 Clock domains and cross-domain handling

Three mutually asynchronous clock domains are present:

- `pclk` — the APB bus clock, slow and stable. Firmware configuration, the register file, the APB master state machine, and the SIC transmitter handshake all live in this domain.
- `wclk` — a high-speed clock used by the calibration engine for edge counting and lock-completion sequencing.
- `vcro_clk` — the VCRO output itself, the measurand of the frequency meter, asynchronous to both of the above.

Because the high-speed domains are mutually asynchronous to `pclk`, two cross-domain crossings are required, and they are handled in dedicated synchronizers. Single-cycle completion events — the lock pulse from the calibration engine and the measurement-done pulse from the frequency meter — cross from `wclk` or `vcro_clk` into `pclk` through a toggle-based pulse synchronizer. Multi-bit payloads such

as `freq_code` and the captured calibration registers are sampled in the destination domain only *after* the corresponding completion pulse has been observed there, so that the data lines are guaranteed to be static when they are read. In the opposite direction, level-stable software-driven controls (target, window length, enables) are picked up by two-flop synchronizers; the small number of edge-sensitive software actions (manual start, auto-recalibration request) are converted into single-cycle pulses on the destination side.

2.2.3 Data and control flow

Figure 2.1 also shows the data and control flow of RFF system. In the steady state the system operates as a single closed-loop pipeline:

1. The host CPU writes the calibration/measurement start, target reference, calibration limits, and reporting policy into register file (configuration path) by APB interface;

2. the calibration engine drives the VCRO control word to the programmed target through the feedback loop and asserts the write the calibration state to the register (calibration path);

3. After VCRO frequency is locked, CPU can send start command to APB interface, then write it to register. Then the frequency meter starts to measure the `vcro_clk` and produces a fresh `freq_code` together with a measurement-completion flag (observation path);

4. The `freq_code` is also stored in register. The APB interface, based on the configuration field in the register, autonomously send `freq_code` or other `register` bits to the SIC transmitter, the SIC transmitter sends the information to the upstream AVFS block and waits for the external consumer to acknowledge it (report path).

The inner control loop never depends on the timing of the external consumer. If upstream consumer (AVFS) is busy and cannot receive information from RFF, SIC transmitter stalls the reporting path: the calibration engine and the frequency meter continue and send the data out. SIC transmitter handles the back-pressure and acts as a barrier between RFF and AVFS. Chapter 3 gives more detailed implementation of the modules above.

2.3 Modeling Purpose and Analysis Object

A fixed-margin strategy in low-power management usually sets the voltage or frequency safety margin according to the worst operating condition. This method is simple to implement. It can cover a certain range of PVT variation, temperature disturbance, and aging effect. However, in most discrete time steps, it keeps more margin than the actual requirement. For a large SoC voltage/frequency domain, this excessive margin leads to extra power consumption. This chapter builds a behavioral model to evaluate runtime frequency feedback under controlled conditions. The main question is whether feedback can release the fixed margin and reduce system-level energy after feedback hardware overhead is included.

The explicit inputs in this chapter—including the disturbance profile in Equation (2.1), the parameter values in Table 2.1, and the accompanying simulation waveforms—are behavioral constructs for controlled comparison. They are *not* fitted from measured silicon data, and they are not used to claim post-layout sign-off or absolute power numbers for a specific technology node. The purpose is to connect frequency feedback, frequency capability, disturbance degradation, power variation, and hardware overhead in a reproducible form. To avoid a gap between simulation figures and equations, this chapter follows a clear sequence. It first defines the disturbance and frequency capability model. It then gives the baseline fixed-margin strategy and the behavioral feedback margin trajectory. After that, it introduces the power and cumulative energy models. Finally, it uses a hardware-overhead sweep to show the boundary under which net energy saving is achieved.

The analysis object in this chapter is a SoC voltage/frequency domain with an RFF loop. The baseline represents a fixed-margin control method. The feedback case uses a time-varying equivalent margin sequence $u_{\mathrm{fb},k}$ taken from the same behavioral time-domain simulation as the figures. The two schemes use the same disturbance profile, target frequency, time-step horizon, and initial condition. The only difference is the control margin trajectory. This setting makes the simulation a controlled comparison. It also avoids treating differences caused by different disturbance conditions or different workloads as feedback benefits.

2.4 Parameter Setting and Value Justification

This chapter uses normalized parameters for behavioral simulation. The parameters are selected to preserve basic circuit-level monotonic relationships. When the equivalent control margin increases, the frequency capability improves. When the disturbance increases, the frequency capability decreases. When the control margin is reduced, the controlled-domain power decreases. The added feedback circuit also introduces extra overhead. The values in Table 2.1 should be read as known inputs, engineering estimates, and sweep variables in this behavioral model.

Table 2.1: Parameter settings and value justification in the behavioral model

Parameter	Value	Justification
k	0-1000	This represents simulation time steps. This length is enough to cover initial margin release, mid-stage disturbance increase, and late-stage compensation.
f_{target}	1.000	This is the normalized target frequency boundary. All frequency capabilities are expressed as relative values around this target.
K_{vf}	0.75	This is the equivalent gain from control input to frequency capability. It represents the positive effect of voltage or margin adjustment on frequency capability. It is a normalized sensitivity parameter.
K_d	1.00	This is the degradation coefficient from disturbance to frequency capability. The value 1.00 allows the disturbance value to directly represent frequency capability loss. This makes the equations and figure trends easier to explain.

Parameter	Value	Justification
u_{fixed}	0.17	This is the fixed conservative margin used by the baseline. The value is selected to cover the worst stage of the disturbance profile in this chapter. Therefore, the baseline can keep frequency safety, but it also keeps excessive margin in most time steps.
$u_{\text{min}}, u_{\text{max}}$	0.00, 0.20	These are the allowed limits of the control input. They model the upper and lower bounds of configurable margin in real hardware. u_{max} is slightly higher than u_{fixed}, so late-stage disturbance compensation is allowed.
V_{nom}	0.80	This is the normalized nominal voltage reference. The power model only uses relative voltage variation, so no specific voltage unit is introduced.
$K_{\text{oh,typ}}$	0.020	This is an engineering estimate for the feedback hardware overhead. It means that the added control logic consumes about 2% of the reference controlled-domain power. This value represents a small control IP in a larger SoC domain. The conclusion is checked by sweeping K_{oh}, not by this single value.

The parameters above can be divided into three groups. The first group contains normalization parameters, such as f_{target} and V_{nom}. They build a common scale. The second group contains behavioral sensitivity and margin limits, such as K_{vf}, K_d, u_{fixed}, u_{min}, and u_{max}. They keep the frequency and power trends easy to interpret. The third group is the hardware overhead parameter K_{oh}. This chapter does not use one fixed absolute overhead value. The value $K_{\text{oh,typ}} = 2\%$ is used as an engineering estimate for a small monitoring and control IP in a larger controlled domain. The conclusion is not based only on this value. A K_{oh} sweep is used to find the maximum overhead that the feedback benefit can tolerate. The equation structure and break-even judgment remain unchanged.

It is important to distinguish K_{vf} from K_{oh}. K_{vf} enters the frequency model. It represents the gain from control input to frequency capability. K_{oh} enters the power model. It represents the additional power ratio introduced by the feedback hardware. They cannot be mixed. Otherwise, one parameter would carry two meanings: frequency gain and hardware overhead.

2.5 Disturbance Profile and Frequency Capability Model

The runtime disturbance is denoted by d_k, where k is the discrete time step index. This chapter uses a deterministic and monotonically increasing disturbance profile to represent gradually increasing timing degradation:

$$d_k = 0.018 + 0.000035k + \frac{0.018}{1+\exp[-(k-250)/20]} + \frac{0.030}{1+\exp[-(k-620)/25]}. \tag{2.1}$$

Equation (2.1) defines a reproducible two-stage degradation input. The first part represents slow drift. The two logistic terms create smooth changes in mild and strong disturbance regions. This setting helps show how excess timing margin is consumed as d_k grows. Figure 2.2 shows this disturbance profile. Since d_k increases monotonically, the theoretical required control input that is directly derived from the disturbance should also increase monotonically.

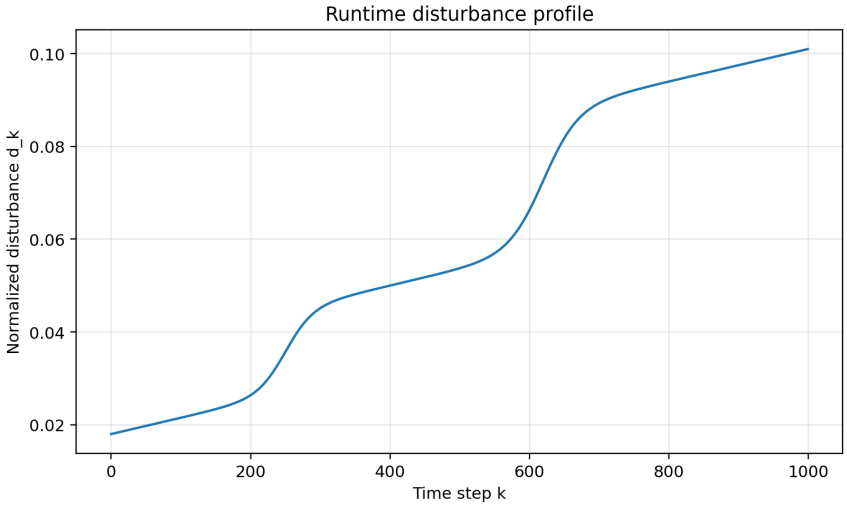


Figure 2.2: Normalized runtime disturbance profile. d_k represents timing degradation that increases with the time step index k.

The frequency capability model is defined as:

$$\hat{f}_k = 1 + K_{\text{vf}} u_k - K_d d_k. \tag{2.2}$$

Here, the constant 1 denotes the normalized reference frequency capability when $u_k = 0$ and $d_k = 0$. $\hat{f}_k$ denotes the normalized frequency capability under the current runtime condition. It does not represent the workload operating clock frequency. Instead, it represents the maximum normalized frequency that the current runtime condition can safely support in this model. The workload operating frequency f_{op} is treated as fixed in this behavioral model. When $\hat{f}_k > f_{\text{op}}$, timing margin still exists. When $\hat{f}_k < f_{\text{op}}$, the current condition can no longer safely support the workload frequency.

The variable u_k is the normalized equivalent control margin. It is not the absolute supply voltage. This chapter defines an effective margin variable as:

$$V_{\text{eff},k} = V_{nom} + u_k. \tag{2.3}$$

A smaller u_k means that more conservative margin is released. A larger u_k means that more margin is added to compensate for disturbance. Equation (2.2) expresses two opposite effects:

$$u_k \uparrow \Rightarrow \hat{f}_k \uparrow, \qquad d_k \uparrow \Rightarrow \hat{f}_k \downarrow . \tag{2.4}$$

This model allows the effects of control input and disturbance to be observed separately. A larger disturbance reduces the frequency capability. The feedback branch offsets part of this degradation through a different margin sequence u_k. The difference between the baseline and feedback schemes comes from the way u_k is generated.

2.6 Baseline and Feedback Control Strategies

2.6.1 Baseline Fixed-Margin Strategy

The baseline strategy uses a fixed control input:

$$u_k^{base} = u_{\text{fixed}}. \tag{2.5}$$

Substituting it into Equation (2.2) gives the baseline frequency capability:

$$\hat{f}_k^{base} = 1 + K_{\text{vf}} u_{\text{fixed}} - K_d d_k. \tag{2.6}$$

Since u_{fixed} does not change with runtime status, the baseline frequency capability is mainly determined by the disturbance profile. As d_k increases, $\hat{f}_k^{base}$ gradually decreases. The fixed-margin strategy is simple and can cover the preset disturbance range. However, when the disturbance has not reached the worst condition, the system still keeps a high margin. This causes unnecessary power consumption.

2.6.2 Theoretical Required Control Input

To avoid confusion between different control quantities, this chapter first defines the theoretical required control input $u_{\text{req},k}$. The frequency target used for this derivation is f_{target}. From Equation (2.2), the required control input can be derived as:

$$u_{\text{req},k} = \frac{f_{\text{target}} - 1 + K_d d_k}{K_{\text{vf}}}. \tag{2.7}$$

$u_{\text{req},k}$ means the control margin that is theoretically needed at time step k if the frequency capability is directly kept at f_{target}. Since d_k increases monotonically, and both K_d and K_{vf} are positive constants, $u_{\text{req},k}$ should also increase monotonically. Therefore, if a curve repre-

sents the required control input, its trend should follow the disturbance trend.

2.6.3 Feedback Closed-Loop Control Strategy

In the feedback strategy, the control input is not given directly by Equation (2.7). It is updated step by step by a behavioral controller according to the frequency error. The error is defined as

$$e_k = f_{\text{ctrl}} - \hat{f}_k^{fb}, \tag{2.8}$$

where f_{ctrl} is the feedback regulation set-point. In the simulation behind Figure 2.3, f_{ctrl} is set slightly above the nominal target f_{target} in Table 2.1 so that the loop does not dwell below the plotted target line. This offset is a behavioral safety margin for trend comparison, not a post-silicon sign-off statement.

The control input uses a discrete feedback update with saturation:

$$u_{\text{fb},k+1} = \text{sat}\big[u_{\text{fb},k} + K_p(e_k - e_{k-1}) + K_i T_s e_k\big], \tag{2.9}$$

where sat$[\cdot]$ keeps $u_{\text{fb},k}$ within $[u_{\min}, u_{\max}]$ from Table 2.1, T_s is the normalized control-window period (one step in the k axis), and K_p and K_i are the proportional and integral gains used in the accompanying simulation ($K_p = 0.32$, $K_i = 0.03$, $T_s = 1$).

The frequency capability under feedback remains the affine plant model

$$\hat{f}_k^{fb} = 1 + K_{\text{vf}} u_{\text{fb},k} - K_d d_k. \tag{2.10}$$

When the initial fixed margin is higher than the actual requirement, $\hat{f}_k^{fb} > f_{\text{ctrl}}$ and $e_k < 0$, so the controller reduces $u_{\text{fb},k}$ and releases the redundant conservative margin. As the disturbance d_k keeps increasing, the frequency capability approaches the control target and the controller increases $u_{\text{fb},k}$ to compensate for degradation. Therefore $u_{\text{fb},k}$ does not increase monotonically with d_k. The non-monotonic trend is not caused by a decrease in disturbance. It reflects closed-loop behavior: the controller first gives back excess margin and then actively compensates for external stress.

Figure 2.3 compares three control inputs under the same disturbance profile. The baseline stays at u_{fixed}. The required input $u_{\text{req},k}$ from Equation (2.7) increases monotonically with d_k because it only reflects how much margin is needed to hold f_{target}. The feedback output $u_{\text{fb},k}$ is the actual feedback controller trajectory from Equations (2.8)–

(2.9). It decreases in the early stage and rises in the later stage. Required control and feedback output must therefore be separated when the control-input trend is explained: the former tracks disturbance impact on margin demand, whereas the latter tracks the dynamic process of margin release and compensation.

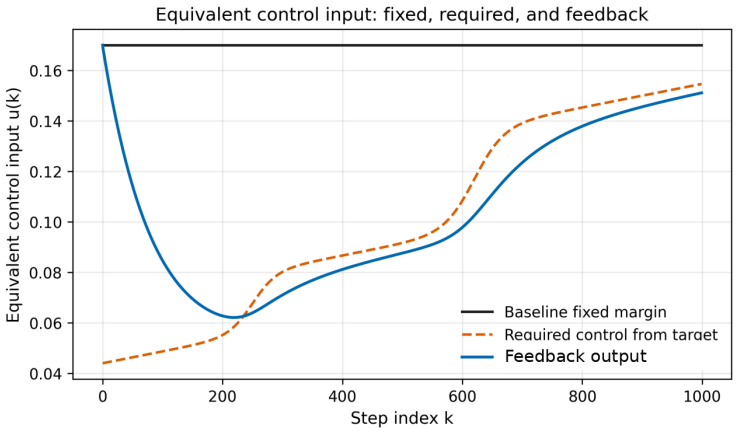


Figure 2.3: Comparison of fixed margin, theoretical required control input, and actual feedback control output. $u_{\mathrm{req},k}$ increases monotonically with the disturbance, while $u_{\mathrm{fb},k}$ shows closed-loop behavior that first releases the conservative margin and then compensates for the disturbance.

2.7 Power and Cumulative Energy Model

The controlled SoC-domain power uses a normalized voltage-power relationship:

$$P_{\mathrm{soc},k} = A(V_{nom} + u_k)^2 f_{\mathrm{op}}. \tag{2.11}$$

Here, A is a scaling coefficient. f_{op} is the workload operating factor. It is the same in the baseline and feedback schemes. The feedback loop changes the available frequency capability and the equivalent control margin. It does not change the workload in this behavioral comparison. Therefore, $\hat{f}_k$ and f_{op} have different meanings. $\hat{f}_k$ is the available frequency capability. f_{op} is the same workload factor used in the power model.

The baseline power is:

$$P_{\text{base},k} = A(V_{nom} + u_{\text{fixed}})^2 f_{\text{op}}. \tag{2.12}$$

The feedback controlled-domain power is:

$$P_{\text{fb,domain},k} = A(V_{nom} + u_{\text{fb},k})^2 f_{\text{op}}. \tag{2.13}$$

It is not enough to compare only Equation (2.12) and Equation (2.13). The feedback closed loop itself needs additional hardware, including frequency monitoring, calibration logic, control registers, and a state machine. Therefore, system-level power must include the feedback hardware overhead. This chapter defines:

$$K_{\text{oh}} = \frac{P_{\text{RFF}}}{P_{\text{ref}}}, \tag{2.14}$$

where P_{RFF} is the average power of the added feedback hardware, and P_{ref} is the reference power of the controlled domain under the baseline fixed margin:

$$P_{\text{ref}} = A(V_{nom} + u_{\text{fixed}})^2 f_{\text{op}}. \tag{2.15}$$

After feedback hardware overhead is included, the feedback net power becomes:

$$P_{\text{fb,net},k} = A(V_{nom} + u_{\text{fb},k})^2 f_{\text{op}} + K_{\text{oh}} P_{\text{ref}}. \tag{2.16}$$

Equation (2.16) avoids the idealized assumption of zero-cost feedback hardware. If $K_{\text{oh}} P_{\text{ref}}$ is ignored, then power reduction almost necessarily holds as long as feedback reduces u_k. After K_{oh} is included, the model can further judge whether the power of the added control circuit cancels the benefit from voltage-margin reduction in the controlled domain.

The cumulative energy is defined as:

$$E(N) = \sum_{k=0}^{N-1} P_k. \tag{2.17}$$

Each time step is assigned unit normalized duration, so cumulative energy is the sum of per-step power.

The energy saving ratio is defined as:

$$\eta_E = \frac{E_{\text{base}} - E_{\text{fb}}}{E_{\text{base}}}. \tag{2.18}$$

By setting $E_{\text{fb}} = E_{\text{base}}$, the critical value of the feedback hardware

overhead can be obtained:

$$K_{\text{oh}}^{\text{crit}} = \frac{\sum_{k=0}^{N-1} \left[P_{\text{base},k} - P_{\text{fb,domain},k} \right]}{N P_{\text{ref}}}. \tag{2.19}$$

When $K_{\text{oh}} < K_{\text{oh}}^{\text{crit}}$, the feedback net energy is lower than the baseline energy. When $K_{\text{oh}} = K_{\text{oh}}^{\text{crit}}$, the energy saved in the controlled domain is exactly cancelled by the added hardware overhead. When $K_{\text{oh}} > K_{\text{oh}}^{\text{crit}}$, the feedback scheme no longer provides net energy benefit under this set of conditions.

2.8 Simulation Result Analysis

2.8.1 Frequency Response

Figure 2.4 shows the normalized frequency capability of the baseline and feedback schemes. The baseline uses a fixed margin, so its curve is mainly determined by Equation (2.6). As d_k increases, the baseline frequency capability gradually decreases, but it remains above the target value. This means that the fixed margin can cover the disturbance range used in this chapter. It also means that the system keeps more frequency capability than required in most time steps.

The feedback curve is obtained by inserting $u_{\text{fb},k}$ into Equation (2.10). Relative to the baseline, the feedback trajectory spends more time with a smaller equivalent margin when the disturbance is mild, and it increases the margin when the disturbance strengthens. The same static power map is then applied pointwise in time.

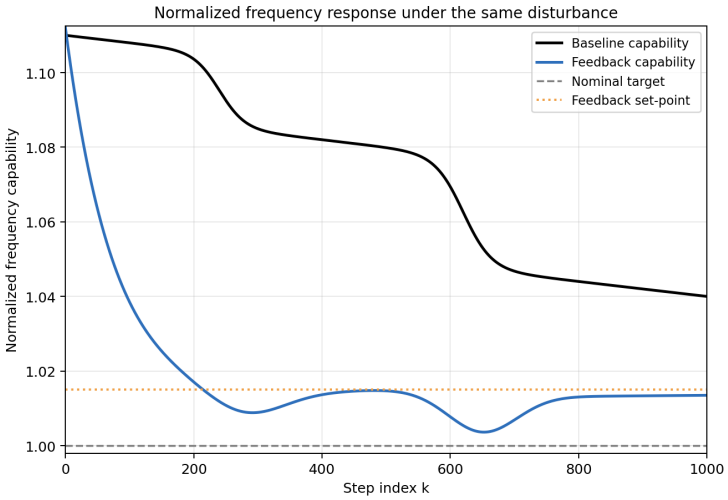


Figure 2.4: Normalized frequency capability response of the baseline and feedback schemes under the same disturbance. The dashed gray line marks the nominal target, and the dotted orange line marks the feedback set-point. The baseline keeps a fixed margin. The feedback curve uses $u_{\mathrm{fb},k}$ in Equation (2.10).

2.8.2 Instantaneous Power

Figure 2.5 shows the instantaneous power comparison. Since u_{fixed} does not change, the baseline power in Equation (2.12) stays essentially constant. The feedback controlled-domain power in Equation (2.13) follows $u_{\mathrm{fb},k}$ from Section 2.6.3. In the initial stage, $u_{\mathrm{fb},k}$ decreases and the controlled-domain power drops. In the later stage, as the disturbance strengthens, $u_{\mathrm{fb},k}$ rises again and the controlled-domain power gradually recovers. After the typical feedback hardware overhead $K_{\mathrm{oh,typ}} = 2\%$ from Table 2.1 is added through Equation (2.16), the feedback net power is higher than the controlled-domain power alone, but it remains lower than the baseline fixed-margin power.

The power benefit of feedback comes from releasing the fixed margin, not from changing the workload f_{op}. The added feedback circuit consumes extra power, so the system-level benefit depends on whether the controlled-domain power reduction is larger than the overhead term $K_{\mathrm{oh}}P_{\mathrm{ref}}$. For a large SoC voltage/frequency domain, the relative overhead of a small control IP is usually low, and the gain from margin release can more easily cover $K_{\mathrm{oh,typ}}$. If the controlled object itself is very small, K_{oh} must be re-evaluated because it may become too large

relative to the domain being controlled.

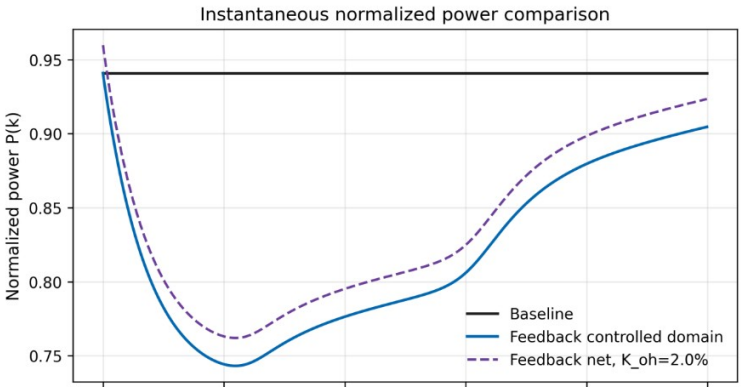


Figure 2.5: Baseline power, feedback controlled-domain power, and feedback net power after feedback hardware overhead is considered ($K_{\mathrm{oh,typ}} = 2\%$).

2.8.3 Cumulative Energy

Figure 2.6 shows the cumulative energy result. Under the current normalized parameters, the final cumulative energy of the baseline is 940.90. The final cumulative energy of the feedback scheme is 818.13 if only the controlled domain is considered. After $K_{\mathrm{oh,typ}} = 2\%$ is added, the feedback net cumulative energy is 836.94. The controlled-domain energy reduction is about 13.05%. The net energy reduction after feedback hardware overhead is considered is about 11.05%.

This result should not be interpreted as the final power reduction ratio of a specific chip implementation. It should be interpreted as a behavioral trend under this set of normalized parameters. Its meaning is that, under the same disturbance and the same time-step horizon, feedback can reduce the controlled-domain energy by lowering excessive control margin. When the added feedback hardware overhead remains at a low ratio, the system-level net energy is still lower than the fixed-margin baseline.

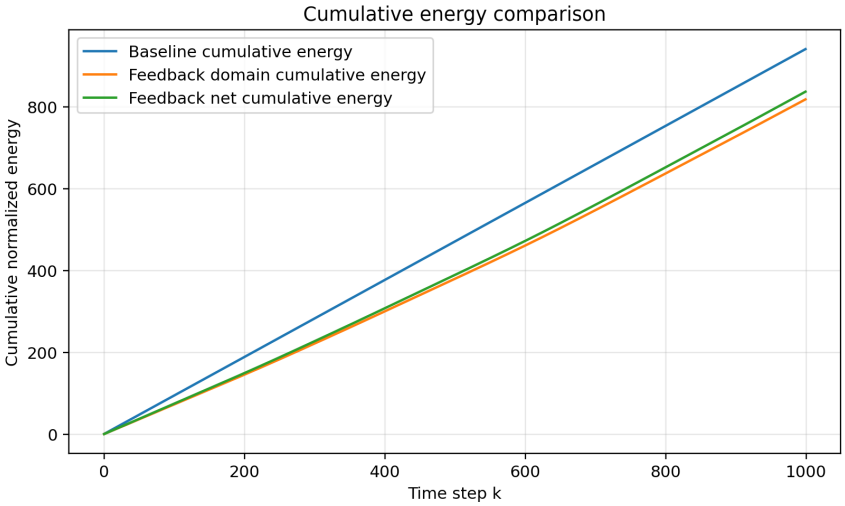


Figure 2.6: Cumulative energy comparison between the baseline and feedback schemes. The feedback net energy includes the typical feedback hardware overhead.

2.8.4 Hardware Overhead Sensitivity

Figure 2.7 shows the variation of the net energy saving ratio η_E with the feedback hardware overhead ratio K_{oh}. As K_{oh} increases, the feedback net energy benefit gradually decreases. When K_{oh} reaches the critical value, the energy saved in the controlled domain is exactly cancelled by the added hardware overhead, and the net benefit becomes zero. From Equation (2.19), the critical value under this set of parameters is:

$$K_{\mathrm{oh}}^{\mathrm{crit}} \approx 0.1305. \tag{2.20}$$

This means that, under the current disturbance profile, behavioral margin trajectory, and power model, feedback still provides net energy benefit as long as the average feedback hardware power is lower than about 13.05% of the baseline reference controlled-domain power. If the overhead is larger than this ratio, the voltage-reduction benefit in the controlled domain is cancelled by the additional hardware power.

This figure gives the applicable range of the scheme. The behavioral comparison is more representative for a large SoC voltage/frequency domain. In this case, the added control logic usually

takes a low power ratio compared with the whole controlled domain. If the same control logic is used for a very small module, K_{oh} may become large, and the net energy-saving conclusion cannot be directly applied. Therefore, sweeping K_{oh} is more meaningful than giving only one energy-saving percentage. It shows the benefit boundary of the feedback scheme under different hardware overhead conditions.

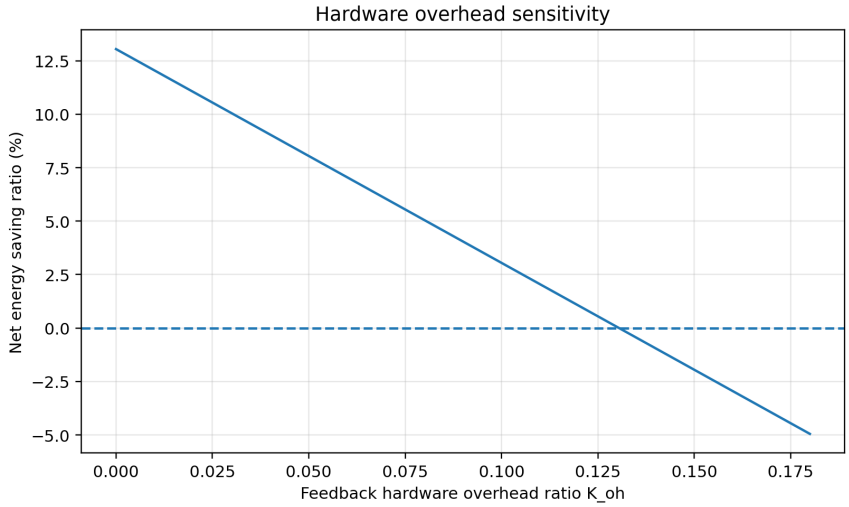


Figure 2.7: Sensitivity of the net energy saving ratio to the feedback hardware overhead ratio K_{oh}. The crossing point corresponds to the break-even overhead.

2.9 Model Scope and Summary

The conclusion of this behavioral model is based on three assumptions. First, the baseline and feedback schemes use the same disturbance, target, time-step horizon, and initial condition. Therefore, the result reflects the difference between control strategies, not the difference between simulation conditions. Second, the parameters are normalized. They are used to describe the trend relationship among frequency capability, disturbance, control input, and power. Third, the feedback hardware overhead is not ignored. It is included in the net power model through the parameter K_{oh}. A sensitivity analysis is used to give the break-even condition.

The simulation results show that the fixed-margin baseline can keep the frequency capability above the target. However, it also keeps

a high control margin in most time steps. The feedback trajectory lowers the average equivalent margin relative to the baseline under the same d_k, which reduces the controlled-domain energy through the static power map. The non-monotonic gap between $u_{\mathrm{fb},k}$ and $u_{\mathrm{req},k}$ is a bookkeeping consequence of comparing a fixed conservative margin with a time-varying behavioral margin trace, not a claim about a specific inner hardware implementation.

Under the current parameters, after the feedback hardware overhead $K_{\mathrm{oh,typ}} = 2\%$ is considered, the feedback scheme shows a net energy reduction trend of about 11.05%. The hardware-overhead sensitivity analysis gives a break-even overhead of about 13.05%. The RFF closed loop does not reduce power under every scale and every overhead condition. It is more suitable for a large SoC voltage/frequency domain. When the relative overhead of the added feedback hardware is lower than the critical value in Equation (2.19), the energy saved by fixed-margin release can cover the extra power of the control circuit. In this case, the system-level net energy is lower than the fixed-margin baseline.

Chapter 3

Hardware Design and RTL Implementation of RFF

3.1 RFF Hardware Architecture Overview

The RFF RTL implementation is organized around a configurable sensing and reporting path. The top-level module contains an APB register interface, configuration registers, VCRO control outputs, calibration logic, a frequency measurement unit, status generation logic, and a payload output path. These blocks allow the system to configure the RFF module, start calibration, observe frequency-related information, and read back the result.

The design separates system configuration from runtime observation. The APB path programs mode and control fields. The VCRO-related path produces observable timing or frequency behavior. The calibration path searches and locks a delay code. The frequency meter converts a clock-like observation into a digital count. The status and payload paths make the result visible to software or an upper monitoring block.

3.2 Register Interface and Configuration Path

The register interface provides the control boundary of the RFF module. The main configuration fields include module enable, mode selection, calibration start, measurement period, VCRO range/coarse/fine control, threshold or reference value, soft-read address, payload send

period, and status clear. The status fields include busy, done, valid, calibration counter result, frequency measurement result, error indication, and selected debug readback values.

Reserved fields are kept stable and are not used for functional decisions. Undefined address holes must not corrupt valid registers. This rule is also checked in the verification chapter. The configuration path therefore has two purposes. It drives the functional blocks during operation, and it provides a clear verification boundary for register read/write behavior.

3.3 APB Interface

3.3.1 General introduction to APB protocol

APB is a low-bandwidth, low-complexity control bus introduced by ARM as part of the AMBA family of on-chip interconnect specifications [17]. It is intended for register-level access to peripheral blocks — timers, UARTs, GPIO controllers, configuration spaces of larger accelerators. Although the throughput of APB is modest, its simple protocol keeps the control-plane area and power overhead low. APB transactions are unpipelined and follow a simple two-phase access pattern (a setup phase followed by an access phase), with no burst, no out-of-order completion, and no transaction identifiers. A single master drives `paddr`, `pwrite`, `pwdata`, `psel` and `penable` towards a slave, and the slave responds with `prdata`, `pready` (which may extend the access phase into wait states) and the optional `pslverr`.

3.3.2 Function in the RFF system

The preceding sections describe modules that produce or consume information internally to the RFF subsystem, but none of these modules is addressable from the host CPU. The APB interface described in this section is the missing link: it is the register-level control plane through which CPU configures the subsystem and through which run-time inner state is returned to the host CPU and upstream module.

In this work the APB interface carries traffic in both directions and is therefore split into two cooperating modules rather than a single slave. When RFF acts as the slave, the register files of the RFF subsystem must be reachable for writing and status readback by the CPU; When acting as the master, the latched frequency code and inner state

produced by the frequency meter and calibration module must be exported to the upstream AVFS consumer. So the APB interface is split into apb_slv and apb_mst sharing the same pclk domain:

- apb_slv — an APB3 slave that exposes the complete register file of the RFF subsystem to a host bus, servicing both writes and reads (status / result observation).
- apb_mst — a small APB3 master that, under firmware-configured policy, autonomously issues write transactions carrying freqm_code towards an upstream target on the same bus fabric.

3.3.3 Bus protocol

Master: apb_mst

The master apb_mst is responsible for issuing APB write transactions whose payload is freqm_code and inner state produced by RFF. The master exposes the standard set of APB3 master-side signals — m_paddr, m_psel, m_penable, m_pwrite and m_pwdata as outputs, and m_pready as the only input from the upstream target. The write-only nature of the path means m_pwrite is tied to logic one for every transaction, and the read-data line m_prdata is not consumed. First, the CPU configures the reporting policy by writing values in some configuration registers: periodic reporting or event-triggered reporting. Periodic reporting refers to reporting at set intervals, while event-triggered reporting refers to reporting immediately when a new value is written to a specific register (e.g., freqm_code).When reporting starts, necessary internal states of the RFF (also configured by CPU) is sent to the SIC transmitter via the apb_mst.

As illustrated in Figure 3.1: IDLE is the default state. Upon a new request (fire_req), the FSM transitions to SETUP, driving the data. One clock cycle later, it moves into the ACCESS state. In this state, the FSM stays in ACCESS until m_pready is asserted, signaling the completion of the transaction and returning to IDLE.

A representative write transaction is shown in Figure 3.2. In the T1 cycle the master drives m_paddr and m_pwdata with the destination address and the latched freqm_code, holds m_pwrite high (write-only path), and asserts m_psel; m_penable is still low. On the next edge the FSM enters ACCESS and the master raises m_penable; if the upstream target accepts the transaction on the same edge by sampling m_pready high, the access phase ends and the FSM returns to IDLE on the follow-

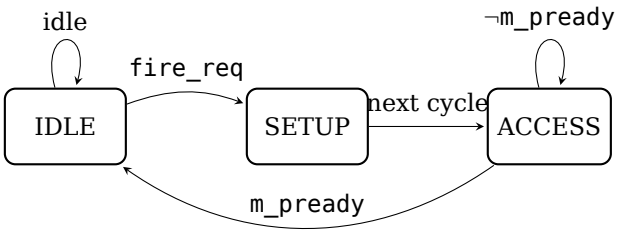


Figure 3.1: Master finite-state machine.

ing edge, but the waveform illustrates the wait-state case: `m_pready` stays low for one extra cycle, during which `m_paddr`, `m_pwdata`, `m_psel` and `m_penable` remain bit-identical, and the transaction commits as soon as `m_pready` rises.

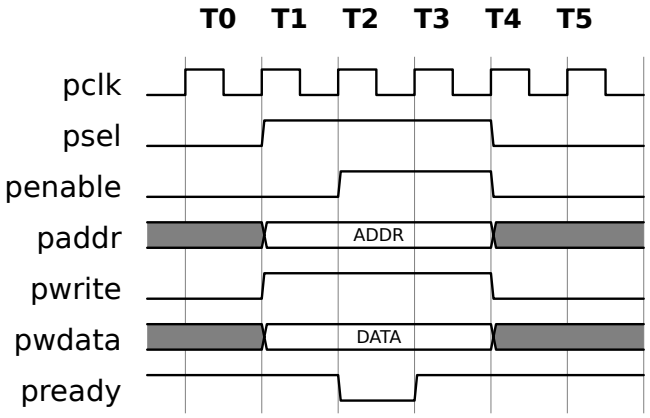


Figure 3.2: APB3 write transaction

Slave: `apb_slv`

The `apb_slv` in the RFF system is designed to enable the host CPU to read from and write to the internal register file. Through the APB bus, the CPU can perform read operations to monitor the internal status of the RFF system, while also controlling its operation by writing values to specific registers; for example, initiating the frequency measurement module is achieved by writing a 1 to the `freqm_start` register. There are two simplifications:

- `pready` is tied to logic one, declaring zero wait states for both reads and writes;

- `pslverr` is tied to logic zero, so no transaction is ever reported as an error to the host.

The register file is implemented as a flat array of 32 bit registers, and there is no shared resource for which a host might have to wait. Returning a wait state would only introduce a path that could mask a downstream stall; returning an error response would require the host to implement a recovery sequence that is unnecessary at this level of the design. The address space is sparsely populated, but unmapped reads return zero and unmapped writes are silently dropped, which keeps the protocol predictable without needing a slave-error path.

For a write transaction, the waveform is the same as Figure 3.2.For a read transaction, a typical waveform is shown in Figure 3.3. In T0, the host drives `paddr` with the target offset, deasserts `pwrite` to indicate a read, and asserts `psel`; `penable` is still low, so `rd_en` stays low and `prdata` reads as zero. On the next edge the host raises `penable`, entering the access phase: `rd_en` is now high, the address decoder selects the addressed register, and its contents appear combinationally on `prdata`. Because the slave ties `pready` to one, the access phase lasts exactly one cycle and the host samples the data on the same edge that ends the transaction; on the following cycle `psel` and `penable` return low and `prdata` falls back to zero.

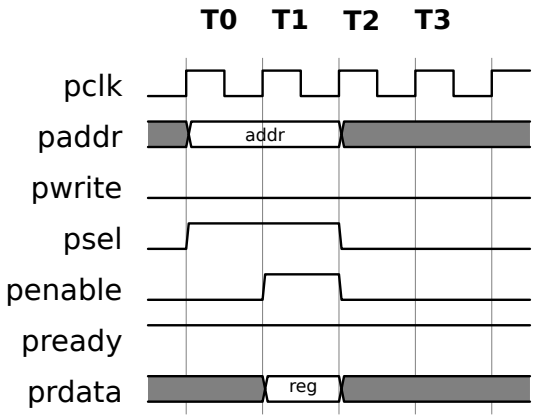


Figure 3.3: APB3 read transaction

3.4 VCRO Behavioral Model and Interface

The VCRO is the tunable timing element in the RFF system. Its main function is to provide a controllable delay or clock path for calibra-

tion and frequency observation. In calibration mode, WCLK is injected into the VCRO delay path. The digital control fields, including Range, Coarse, and Fine, are adjusted until the delay of this path is close to half of the WCLK period. This gives the RFF system a locked timing configuration. In normal mode, the locked configuration is used to generate the VCRO output clock. This output is then measured by the frequency meter and used by the later RFF feedback path.

This chapter defines the digital behavioral model of the VCRO structure used in the RFF system. It also defines the interface boundary between this model and the later calibration module, frequency meter, and CDC/report path. The chapter does not sign off the transistor-level analog VCRO macro. Its focus is narrower. It explains how the VCRO is represented in RTL as a configurable, observable, and calibratable timing object.

From the RTL modeling view, the VCRO structure in RFF can be written as an abstract mapping:

$$M_{\text{VCRO}} : (x_w(t), \boldsymbol{u}, \xi, m) \mapsto (x_d(t), y_{cal}, v_{clk}, s), \tag{3.1}$$

where $x_w(t)$ is the input reference clock waveform, and $\boldsymbol{u} = [R, C, F]^T$ is the control vector. The three entries represent the range, coarse, and fine control fields. The variable ξ denotes an equivalent environmental disturbance in the behavioral model, such as PVT shift, temperature change, or aging. The variable m denotes the operating mode.

The outputs also have fixed meanings. The signal $x_d(t)$ is the delay-line observation output in calibration mode. The signal y_{cal} is the calibration decision signal. The signal v_{clk} is the output clock observed by the frequency meter in normal mode. The status variable s includes done, error, locked code, and other reportable states.

This chapter only defines the external behavior needed by the digital model. The detailed window statistics, search rule, lock condition, and failure handling belong to the calibration chapter.

3.4.1 Analog VCRO and Digital Modeling Boundary

The VCRO macro is an analog or mixed-signal clock block. Its real output frequency, edge jitter, phase noise, PVT sensitivity, aging response, and delay-cell behavior depend on the circuit design, layout parasitics, and process conditions. These properties must be checked by analog simulation, mixed-signal verification, and silicon test.

The model used in this chapter is not a replacement for that analog

sign-off. It is a behavioral abstraction for the digital RFF control path. It lets the calibration module, frequency meter, CDC capture logic, and report/readback path run in an RTL simulation. The supported conclusion is therefore limited: under the defined behavioral model, the digital logic can be checked for correct configuration, observation, locking, and reporting.

3.4.2 VCRO Circuit Structure and Delay-Line Abstraction in Calibration Mode

Figure 3.4 shows the VCRO circuit structure. The figure contains input selection logic, Range Delay, Coarse Delay, Fine Delay, calibration decision logic, and configuration control logic. The main timing path has a clear staged structure. This is why the VCRO structure can be abstracted as a configurable delay line in calibration mode.

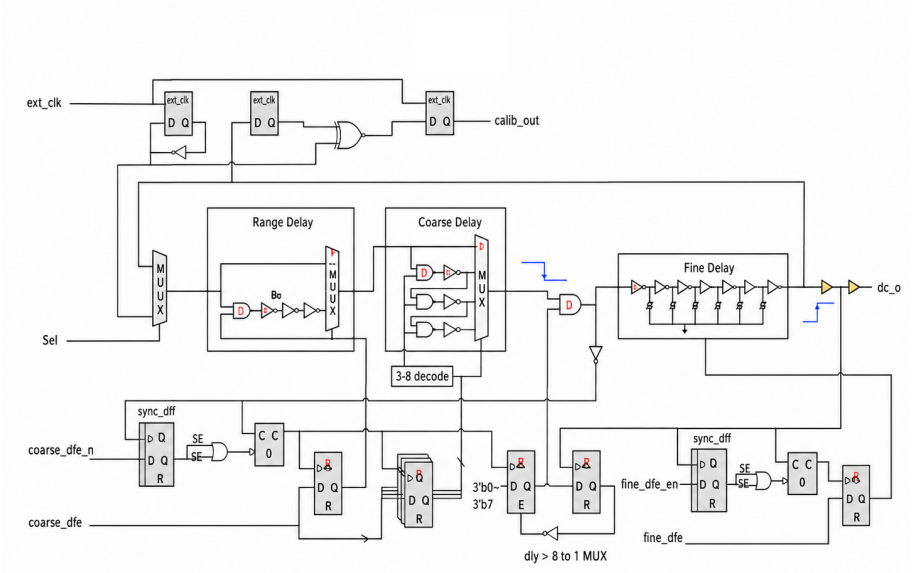


Figure 3.4: VCRO circuit structure

A delay line is a signal propagation path made of delay cells, buffers, inverters, or equivalent gate-level elements. It does not generate a new clock by itself. It delays an input transition. After the input signal passes through several delay stages, the output becomes a delayed version of the input.

In calibration mode, the external reference clock $x_w(t)$ enters the

internal delay path through the MUX. This clock is the `ext_clk` in the figure, or WCLK in the system description. In this mode, the ring oscillator feedback loop is not closed. The VCRO is therefore not used as a free-running oscillator. Still, an output can be observed. The external clock transitions travel through the Range Delay, Coarse Delay, and Fine Delay stages. They then appear at the observation node after a controlled delay.

The calibration-mode behavior can be written as

$$x_d(t; \boldsymbol{u}, \xi) = x_w\big(t - D_{sum}(\boldsymbol{u}, \xi)\big), \tag{3.2}$$

where D_{sum} is the equivalent total delay under the current control vector and environmental condition.

The decision logic at the top of Figure 3.4 generates `calib_out`. This signal does not give a continuous-time delay value. It gives binary information about the delay state relative to the WCLK half-period target. The later calibration counter counts the high and low values of `calib_out` in a measurement window. The counter result is then used to decide whether the delay should be increased, decreased, or locked.

3.4.3 R/C/F Total Delay Model

The main path in Figure 3.4 passes through Range Delay, Coarse Delay, and Fine Delay. The total delay is therefore controlled by three fields and a fixed path delay. The control vector is defined as

$$\boldsymbol{u} = [R, C, F]^T, \quad R \in \{0, 1\}, \quad C \in \{1, 2, \ldots, 15\}, \quad F \in \{1, 2, \ldots, 36\}. \tag{3.3}$$

Let D_R be the range delay unit, D_C be the coarse-related delay unit, D_F be the fine delay unit, and D_{fix} be the fixed path delay. The digital behavioral model uses

$$D_{sum} = (7R + C)D_R + (R + C)D_C + FD_F + D_{fix}. \tag{3.4}$$

This equation captures the three-level adjustment. Range Delay selects a large delay region. Coarse Delay moves the delay closer to the target with a medium step. Fine Delay makes small corrections near the target. The term D_{fix} covers the non-tunable delay from MUXes, buffers, and fixed gate paths.

When environmental disturbance is included, the delay can be writ-

ten as

$$D_{sum}(\xi) = (7R + C)D_R(\xi) + (R + C)D_C(\xi) + FD_F(\xi) + D_{fix}(\xi). \tag{3.5}$$

For RTL trend simulation, a first-order approximation is enough:

$$D_{sum}(\xi) \approx D_{sum}(0) + \Delta D_\xi, \qquad \Delta D_\xi \approx \alpha_D \xi D_{sum}(0). \tag{3.6}$$

This approximation is only used for behavioral trend modeling. It is not used as a replacement for an analog PVT frequency curve.

3.4.4 Output Clock Abstraction in Normal Mode

This section defines the output clock meaning in normal mode. It does not repeat the calibration search algorithm. The direct result of calibration is not the final working clock. The direct result is a locked control vector:

$$\boldsymbol{u}_{lock} = [R_{lock}, C_{lock}, F_{lock}]^T. \tag{3.7}$$

This vector makes the configurable delay in calibration mode close to one half of the WCLK period:

$$D_{lock} = D_{sum}(\boldsymbol{u}_{lock}, \xi), \qquad D_{lock} \approx \frac{1}{2} T_{wclk}. \tag{3.8}$$

The remaining half-period error is

$$e_D = D_{lock} - \frac{1}{2} T_{wclk}. \tag{3.9}$$

When $|e_D|$ is within the allowed range, the current control vector is stored as the locked configuration.

After the system enters normal mode, $\boldsymbol{u}_{lock}$ becomes the active configuration. The delay path that was observed in open-loop form during calibration is now used in the normal output path. In behavioral terms, calibration first tunes one propagation delay to about $\frac{1}{2} T_{wclk}$. In normal mode, one full output period is formed by two equivalent propagation intervals, one related to the rising edge and one related to the falling edge:

$$T_{vclk} \approx D_{\uparrow,lock} + D_{\downarrow,lock}. \tag{3.10}$$

If the two intervals are treated as approximately symmetric,

$$D_{\uparrow,lock} \approx D_{\downarrow,lock} \approx D_{lock}, \tag{3.11}$$

then

$$T_{vclk} \approx 2D_{lock} \approx T_{wclk}, \qquad f_{vclk} \approx f_{wclk}. \tag{3.12}$$

This is the key meaning of the normal-mode abstraction. Calibration locks a half-period delay. Normal mode uses that locked delay to form a full output clock period. As a result, the internal output clock is close to WCLK in period and frequency at the behavioral level. Here, "locked" means that the R/C/F control vector is held and used by the normal output path. It does not prove the dynamic lock behavior of an analog PLL.

If the rising and falling paths are not fully symmetric, or if the normal output path adds fixed loop delay, the period error can be written as

$$e_T = T_{vclk} - T_{wclk} \approx 2e_D + e_{asym} + e_{loop}, \tag{3.13}$$

where e_{asym} is the residual error from rising/falling path mismatch, and e_{loop} is the residual error from extra fixed loop delay or model mismatch.

The active control vector in different stages is

$$\boldsymbol{u}_{active} = \begin{cases} \boldsymbol{u}_{seed}, & \text{pre-calibration}, \\ \boldsymbol{u}_{search}, & \text{calibration running}, \\ \boldsymbol{u}_{lock}, & \text{post-lock / normal mode}, \\ \boldsymbol{u}_{safe}, & \text{error state}. \end{cases} \tag{3.14}$$

Here, $\boldsymbol{u}_{seed}$ is the initial configuration. $\boldsymbol{u}_{search}$ is the temporary configuration during search. $\boldsymbol{u}_{safe}$ is the safe configuration used in the error state. The error state means that calibration did not produce a reliable locked code under the current condition. For example, the search may hit the maximum iteration count, the control code may reach its boundary, or WCLK may be abnormal. In that case, the system should not use the temporary search code. It should switch to $\boldsymbol{u}_{safe}$ and report `calib_error`, `max_iter_hit`, or an equivalent status.

3.4.5 Model Assumptions, Interface Boundary, and Limitations

The model uses five assumptions for RTL verification.

First, a change in the control vector can cause an observable delay change:

$$\Delta \boldsymbol{u} \neq 0 \Rightarrow \Delta D_{sum} \neq 0. \tag{3.15}$$

This follows the three tunable stages in Figure 3.4. It is the basic condition for calibration search.

Second, within the valid code range, the delay control has a clear direction:

$$\text{increase delay} \Rightarrow D_{sum,n+1} > D_{sum,n}, \tag{3.16}$$

$$\text{decrease delay} \Rightarrow D_{sum,n+1} < D_{sum,n}. \tag{3.17}$$

This supports the search direction. It does not require the real analog VCRO to be perfectly linear under all conditions.

Third, the decision output in calibration mode reflects the relative position between the current delay and the WCLK half-period target:

$$y_{cal} = \Phi\left(D_{sum} - \frac{1}{2}T_{wclk}\right), \tag{3.18}$$

where $\Phi(\cdot)$ is the binary decision relation formed by sampling and combinational logic. The RTL does not calculate a continuous-time delay error. It uses the statistics of `calib_out` to judge the delay state.

Fourth, the input clock, control vector, and equivalent environmental disturbance are assumed to be nearly stable within one measurement window:

$$\boldsymbol{u}[k] \approx \boldsymbol{u}_n, \quad T_{wclk}[k] \approx T_{wclk}, \quad \xi[k] \approx \xi_n, \quad k = 0, 1, \ldots, N-1. \tag{3.19}$$

This assumption makes the high/low count in one window correspond to one working condition.

Fifth, the model is a behavioral abstraction. It keeps only the input and output relations needed by the digital system:

$$M_{RTL} = \{\boldsymbol{u}, x_w, y_{cal}, v_{clk}, s\}. \tag{3.20}$$

Jitter, phase noise, layout parasitics, transistor-level nonlinearity, and real PVT frequency curves are not included in this RTL behavioral model.

Based on these assumptions, the interface boundary between the VCRO model in RFF and the downstream modules is shown in Table 3.1.

Table 3.1: Interface boundary between the VCRO model and downstream modules

Interface	Model responsibility	Downstream responsibility
VCRO model → Calibration	Provide the delay-line observation output controlled by $\boldsymbol{u}$ and the `calib_out` decision signal.	Count the window statistics, choose the search direction, check the lock condition, and handle failure.
VCRO model → Frequency meter	Provide the measurable output clock v_{clk} in normal mode under $\boldsymbol{u}_{lock}$.	Count the output clock in a measurement window and generate `freq_code`.
VCRO model → CDC/report	Provide the locked code, done flag, error flag, and related status information.	Capture cross-domain status and generate the software-visible report.

Thus, this chapter can verify one point only: under the defined behavioral abstraction, the RFF digital control path can be checked for configuration, observation, locking, frequency measurement, and reporting. The real VCRO jitter, PVT-corner behavior, silicon frequency accuracy, and aging response remain part of analog verification and silicon test.

3.5 RFF Calibration Module

The main purpose of the RFF calibration module is to find a proper control code before the RFF enters normal operation. This control code is used to align the VCRO delay line inside the RFF system with the external reference clock `wclk`.

The VCRO delay line inside the RFF system is controlled by three fields: range, coarse, and fine. Different code combinations produce different delay values. During calibration, the module searches for a suitable range, coarse, and fine code. The target is to make the delay of the VCRO delay line close to one half of the `wclk` period. When this condition is met, the delay path is considered to be matched with the reference clock.

This calibration code is not only a normal configuration value. It de-

fines the working point of the VCRO delay path inside the RFF system. After calibration is finished, the locked range, coarse, and fine code is used as the active RFF configuration in normal mode. With this setting, the VCRO can generate an output clock that is close to the reference clock period, and the following frequency measurement and runtime feedback logic can work from a stable starting point.

In simple terms, the calibration module first uses `wclk` as a reference. It then adjusts the VCRO delay line inside the RFF system to a proper position. Finally, it locks the selected control code and provides this code to the normal RFF operation.

RTL port names in Figure 3.5 keep the implementation prefix `AVTS_VCRO_*`. In this thesis, this path is discussed as the RFF calibration path.

3.5.1 Inputs, Outputs, and Operating Modes

At hardware level, the calibration path contains input selection, a configurable delay line, combinational decision logic, level counters, lock registers, and readback registers. The same delay-line hardware is used in two modes. In calibration mode, the MUX selects the external calibration clock path. In normal mode, the MUX selects the feedback path, and the delay line forms the oscillation loop.

Table 3.2: Main inputs and outputs of the RFF calibration module

Category	Signal / field	Function
Input	`ext_clk` / `wclk`	Reference clock for calibration and half-period matching.
Input	`calib_start` / `cal_sel`	Starts calibration and selects the calibration path or the feedback path.
Input	R/C/F seed code	Initial delay code for the search.
Input	`window_size`, `match_threshold`	Set the observation-window length and the allowed count imbalance.
Output	`rff_calib_out`	Binary decision signal generated by combinational logic.
Output	`R_lock`, `C_lock`, `F_lock`	R/C/F delay code locked after calibration.
Output	`high_count`, `low_count`	Counts of `rff_calib_out = 1` or `rff_calib_out = 0` inside the observation window.
Output	`calib_done`, `calib_error`	Status signals for completion, failure, or out-of-range search.

3.5.2 Calibration Path and Delay-Line Reuse

Figure 3.5 shows the main calibration path. `AVTS_VCRO_EXT_CLK` is the external reference clock. Node *a* is the main reference-clock path. Node *b* is the sampled node after `CA`. Node *d* is the retimed node before the combinational gate. Node *e* is the combinational decision node before `CB`. The output of `CB` is `AVTS_VCRO_CALIB_OUT`.

Node *c* is the reference path sent to the MUX in calibration mode. The MUX selects whether the delay line receives the calibration clock path or the feedback path. After the MUX, the signal passes through `Range Delay`, `Coarse Delay`, and `Fine Delay`. `Range Delay` selects a large delay region. `Coarse Delay` moves the delay with a larger step. `Fine Delay` gives the final tuning step. The output after `Fine Delay` is buffered as `AVTS_VCRO_OUT`. The same output also returns to the MUX and closes the loop in normal mode.

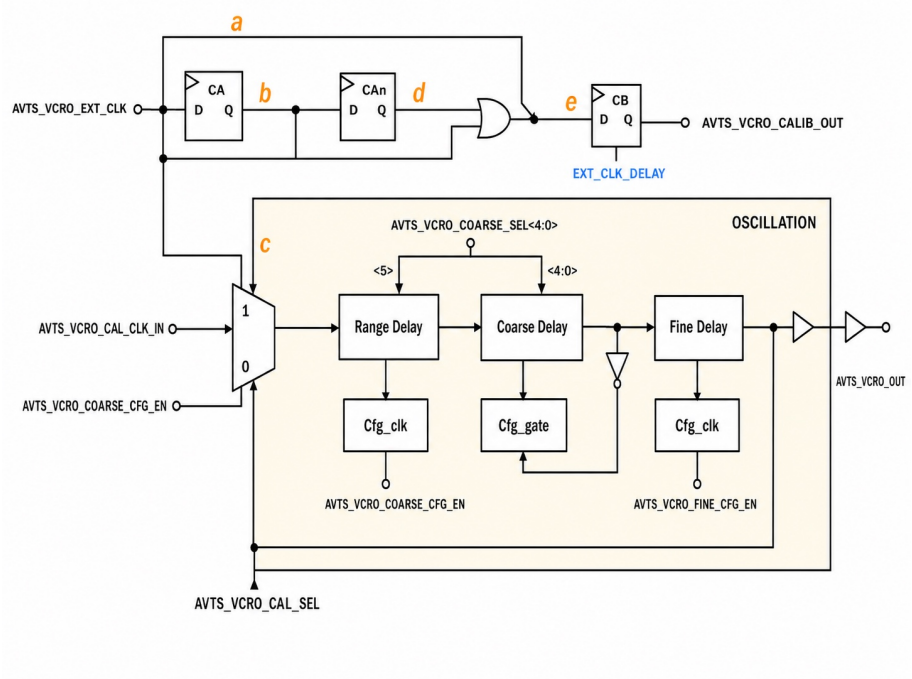


Figure 3.5: RFF calibration path and delay-line reuse structure

This structure does not measure d_{sum} as a continuous time value. It converts the delay-line phase relation into a digital decision signal. The search logic only needs to know whether `rff_calib_out` is more often high or more often low inside the observation window.

3.5.3 Combinational Meaning of `rff_calib_out`

`rff_calib_out` is generated by combinational logic. It is not produced by separately counting rising edges and falling edges. The logic uses the relation among the reference path, the retimed path, and the delayed observation path. The result is a binary signal that indicates which side of the half-period target the current delay is closer to.

Figure 3.6 shows two cases. In the left case, d_{sum} is smaller than half of $T_{\text{ext\_clk}}$, so the delay path is short. In the right case, d_{sum} is larger than half of $T_{\text{ext\_clk}}$, so the delay path is long. Signals *a–f* correspond to the key nodes shown in Figure 3.5 and to their derived timing relations. The red and blue dashed markers show the timing positions used by the combinational decision path.

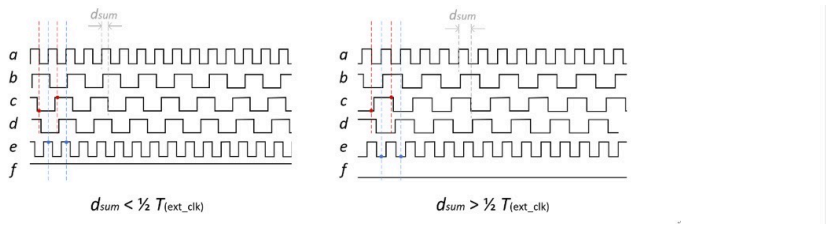


Figure 3.6: Calibration decision timing for different d_{sum} values

The polarity is defined as follows. When $d_{sum} < \frac{1}{2}T_{\text{ext\_clk}}$, `rff_calib_out` is more likely to be low in the observation window. When $d_{sum} > \frac{1}{2}T_{\text{ext\_clk}}$, `rff_calib_out` is more likely to be high. When d_{sum} is close to $\frac{1}{2}T_{\text{ext\_clk}}$, high and low samples become close in number.

3.5.4 Definition of `high_count` and `low_count`

The counters sample the level of `rff_calib_out` during an observation window with N sample cycles. `high_count` is the number of samples with `rff_calib_out = 1`. `low_count` is the number of samples with `rff_calib_out = 0`.

$$H_N = \text{count}(\texttt{rff\_calib\_out} = 1), \qquad L_N = \text{count}(\texttt{rff\_calib\_out} = 0), \qquad H_N + L_N = N. \tag{3.21}$$

`high_count` and `low_count` are level counts. `high_count` increments when `sample_en` is active and `rff_calib_out` is 1. `low_count` increments when `sample_en` is active and `rff_calib_out` is 0. They are not transition counters.

Table 3.3: Hardware actions for `high_count` and `low_count` results

Window result	Delay state	Hardware action
`low_count` is clearly larger than `high_count`	$d_{sum} < \frac{1}{2}T_{\text{ext\_clk}}$; delay is short	Increase the total delay of the R/C/F code.
`high_count` is clearly larger than `low_count`	$d_{sum} > \frac{1}{2}T_{\text{ext\_clk}}$; delay is long	Decrease the total delay of the R/C/F code.
$\lvert H_N - L_N \rvert \leq$ `match_threshold`	Delay is close to the half-period target	Lock the current R/C/F code.

3.5.5 Search, Lock, and Status Readback

The search process follows a fixed hardware sequence. It loads the seed code, applies the code to the delay line, waits for settling, counts one observation window, compares `high_count` and `low_count`, and then updates or locks the code.

If `low_count` is dominant, the delay is short and the search logic increases the delay code. If `high_count` is dominant, the delay is long and the search logic decreases the delay code. If the count difference is within `match_threshold`, the current R/C/F code is written into the lock register.

After lock, `R_lock`, `C_lock`, and `F_lock` become the active configuration for normal mode. The MUX then selects the feedback path. The delay line and output buffers form the oscillation loop. The frequency meter observes the VCRO output clock generated by the locked configuration. It does not use the intermediate calibration nodes.

If the search reaches the maximum iteration count, or if the R/C/F code reaches the valid range boundary before a match is found, the module asserts `calib_error` or `max_iter_hit`. `high_count`, `low_count`, the locked code, and the state bits are kept in readback registers. They are used by software and by the verification environment.

3.6 Frequency Meter

3.6.1 Module function within the RFF subsystem

In RFF closed loop scheme, VCRO serves as a high-frequency clock source, the frequency of which drifts dynamically with control voltage and environmental PVT (process, voltage, temperature) variations. To apply closed-loop control to the VCRO, the system requires a mechanism to observe its physical frequency in real time.

The frequency meter serves precisely this role. Its main function is to use the low-frequency, highly stable Advanced Peripheral Bus (APB) clock f_{pclk}, as a reference to measure the highly variable, asynchronous high-frequency VCRO clock, f_{vcro}. The module subsequently converts this measurement into a digital frequency control word `freq_code`, which is exported to upstream via the APB interface.

To simultaneously achieve low power consumption, minimal area overhead and safe multi-bit clock-domain crossing (CDC), the design of the frequency meter is governed by a windowed-measurement architecture coupled with a static-readout CDC mechanism: the observation window is generated in the slow domain, the edges are counted in the fast domain, and the multi-bit result is sampled by the bus only after the counter has demonstrably stopped.

3.6.2 Hardware Architecture

The hardware architecture spans the low-frequency APB control domain (`pclk`) and the high-frequency target domain (`vcro_clk`). The complete operational flow is partitioned into four distinct phases: window generation, clock-domain synchronization, high-speed counting and static readout. A diagram is given in Figure 3.7.

Window Generation in the `pclk` Domain

The measurement sequence is initiated by firmware via the APB interface. The host first writes a reference target value N_{ref} to the configuration register, defining the length of the measurement window in `pclk` cycles. A subsequent write to a start bit triggers the state machine. Within the stable `pclk` domain, a reference counter increments from 0 to N_{ref}; while the counter is active, the logic asserts a window enable signal `en_pclk`. The absolute duration of the reference window

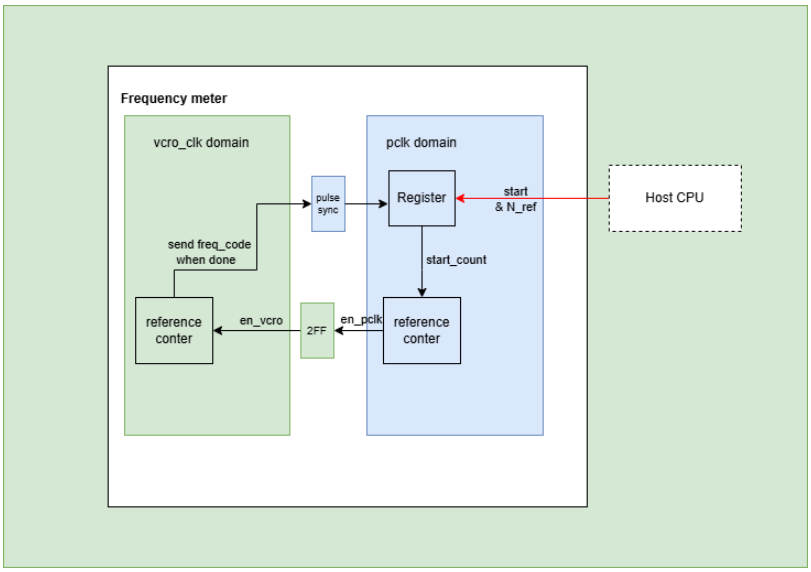


Figure 3.7: Block view of the frequency meter. The reference window is generated in the `pclk` domain; the edge counter lives entirely in the `vcro_clk` domain; only a single-bit gate crosses slow-to-fast and a single-bit completion pulse crosses fast-to-slow.

is therefore

$$T_{\text{window}} = \frac{N_{\text{ref}}}{f_{\text{pclk}}}.$$

Clock-Domain Crossing (Slow-to-Fast)

Because f_{pclk} and f_{vcro} are mutually asynchronous, `en_pclk` cannot directly drive the high-frequency counter without risking metastability at its transitions. The module therefore propagates this enable signal into the `vcro_clk` domain through a standard two-flip-flop(2FF) synchronizer, producing `en_vcro`. Crucially, because $f_{\text{pclk}} \ll f_{\text{vcro}}$, the enable level generated in the slow domain persists for many `vcro_clk` cycles relative to the fast domain; the slow-to-fast single-bit crossing is therefore inherently safe and guarantees no pulse loss. The two synchronizer flops do introduce one or two `vcro_clk` cycles of latency at each edge of the enable, so the actual counting interval differs from T_{window} by at most ± 2 `vcro_clk` cycles.

High-Speed Counting

The high-speed counter is entirely confined to the `vcro_clk` domain. While `en_vcro` is sampled high, the counter increments on every rising edge of `vcro_clk`; once the window closes and the signal is deasserted, the counter immediately freezes. The accumulated value constitutes the preliminary `freq_code`. To accommodate the worst-case operating conditions without overflow during an extended measurement window, the counter is implemented with a 32-bit width.

Static readout via toggle synchronization

Upon completion of the measurement, the system must notify the APB control domain to read the 32-bit `freq_code`. However, transferring a single-cycle completion pulse from the gigahertz-range `vcro_clk` domain back to the megahertz-range `pclk` domain poses a fundamental sampling hazard: a direct nanosecond-scale pulse would inevitably fall between two sampling edges of the slower clock and be missed entirely.

To reliably readout from this fast-to-slow asynchronous boundary, the module employs a toggle-synchronizer (pulse synchronizer) mechanism, as illustrated in Figure 3.8, executing in three steps:

1. **Pulse-to-toggle (fast domain).** In the `vcro_clk` domain, the instantaneous completion pulse generated by the falling edge of `en_vcro` is used to invert the state of a dedicated toggle register. This action effectively stretches a transient, single-cycle pulse into a stable, long-duration level signal.
2. **Two-flop synchronization (CDC).** The level-stable toggle signal is then safely propagated into the `pclk` domain through a standard two-flop synchronizer, completely bypassing the pulse-skipping hazard.
3. **Edge detection (slow domain).** Within the `pclk` domain, the synchronized signal is delayed by one additional cycle. An XOR gate compares the current and delayed states to detect transitions (both rising and falling edges); its output reconstructs a perfectly timed, single-cycle `measure_done` pulse native to the `pclk` domain.

Once the APB-side state machine registers this reconstructed `measure_done` flag, it is guaranteed that the high-frequency counter has been stalled entirely, rendering the multi-bit `freq_code` data lines

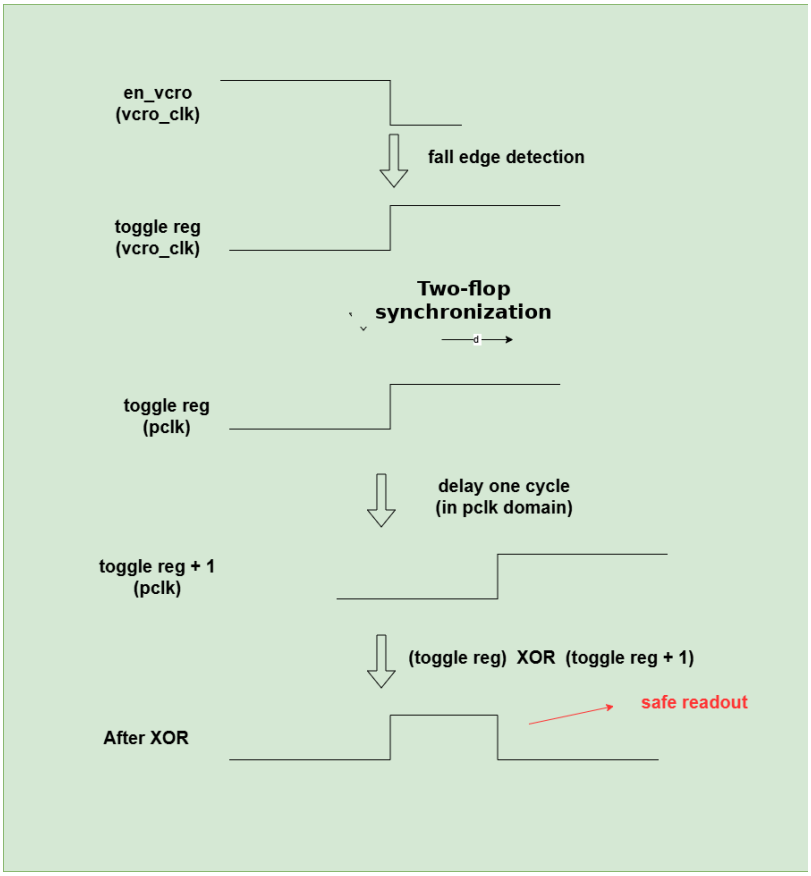


Figure 3.8: Toggle-based pulse synchronizer for fast-to-slow clock domain crossing.

strictly static. The APB interface is then free to safely sample and report the full-width payload without requiring an area-intensive asynchronous FIFO or Gray-code conversion logic.

3.6.3 Mathematical Model and Precision Analysis

Following a completed static readout, the relationship between the captured `freq_code` and the physical frequency f_{vcro} is

$$f_{\text{vcro}} = f_{\text{pclk}} \cdot \frac{\texttt{freq\_code}}{N_{\text{ref}}}.$$

Rearranging this expression yields the theoretical hardware cycle count

$$\texttt{freq\_code} \;=\; N_{\mathrm{ref}} \cdot \frac{f_{\mathrm{vcro}}}{f_{\mathrm{pclk}}}.$$

Quantisation error and precision. The absolute error of the CDC sampling process originates from the asynchronous sampling of the slow-domain window boundaries by the fast-domain clock. Because the two domains are not phase-aligned, the exact open and close instants of the window cannot be observed simultaneously by both counters; this introduces a fundamental quantisation error of ± 1 `vcro_clk` cycle. The relative measurement error of the frequency meter is therefore

$$E \;=\; \frac{\pm 1}{\texttt{freq\_code}}.$$

This architectural choice — using the low-frequency `pclk` to generate a long observation window while accumulating counts at the high-frequency `vcro_clk` — is the central pillar of the meter's precision. Even within a modest absolute timeframe of, e.g., $1\,\mathrm{ms}$, the extremely high f_{vcro} accumulates a `freq_code` of order 10^6. This large denominator aggressively suppresses the fractional impact of the ± 1 quantisation error, allowing the frequency meter to consistently achieve part-per-million (ppm) accuracy.

3.7 SIC Transmitter

3.7.1 Module function within the RFF system

The calibration block of Section 3.5 establishes a known reference for the VCRO, and the frequency meter of Section 3.6 converts the calibrated clock into a digital code consumed by AVFS. Both blocks are entirely confined to the RFF subsystem and produce no output externally on their own. The System-Interface-Communication transmitter (`sic_tx`) completes the observation chain by exporting the runtime state of the RFF subsystem so that it can be consumed by the upstream AVFS.

The design of the SIC is governed by two fundamental principles:

- **Resource Arbitration:** As illustrated in Figure 3.9, in the AVFS system, the `freq_code` generated by the RFF is not the only data requiring transmission. Other monitoring systems also report data

via the SIC. Due to limited on-chip pin resources, the SIC can only transmit data from one monitoring system at a time. Therefore, the SIC must resolve reporting conflicts by arbitrating simultaneous requests.

- **Timing Isolation:** The reporting path must not introduce timing dependence into the main control loop. The external receiver may be slow, temporarily absent, or apply arbitrary back-pressure; none of these conditions are permitted to delay the calibration finite-state machine or the frequency meter. Consequently, the module acts as both a transmitter and an isolation barrier, protecting the internal feedback path from unreliable external consumers by managing timing mismatches locally.

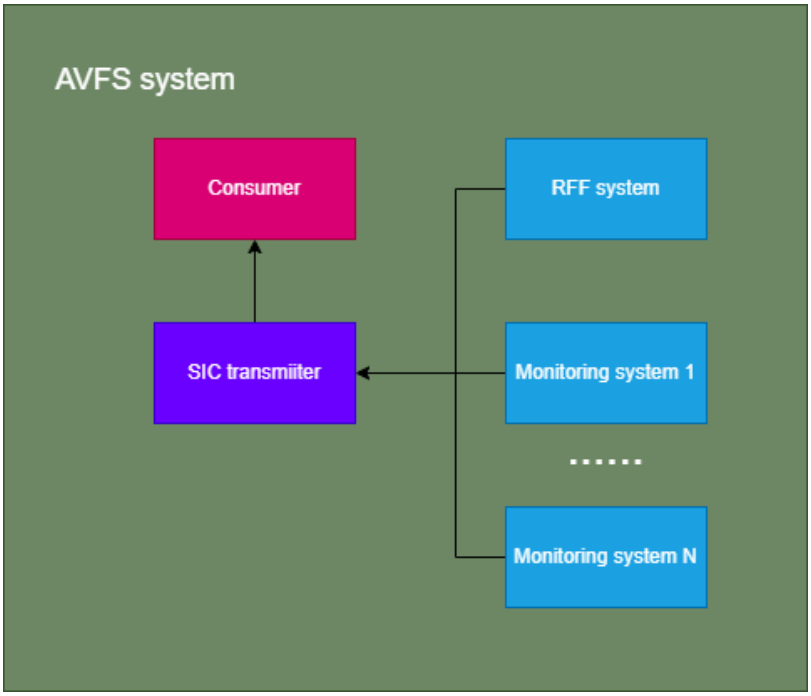


Figure 3.9: SIC Reporting path in AVFS system

3.7.2 Valid–ready handshake and back-pressure isolation

The unit of communication exported by `sic_tx` is referred as a *beat*. A *beat* is delivered to the external receiver through a standard valid–

ready handshake. the transmitter `sic_tx` drives `report_valid` together with the data lines `report_payload`, while the external receiver drives `report_ready` back into the module. A beat is regarded as committed only on a clock edge at which both `report_valid` and `report_ready` are sampled high simultaneously.

Protocol

In the present module the handshake operates at the granularity of a single beat and is summarised by the following three contracts.

1. *Launch.* On the cycle in which a trigger is granted, the module captures the current internal state into the register `report_payload` and asserts `report_valid`.
2. *Hold.* While `report_valid` is asserted and `report_ready` has not yet been observed high, the contents of `report_payload` remain bit-identical; no field is permitted to change. The receiver is therefore free to delay sampling for an arbitrary number of cycles without risking data tearing.
3. *Commit.* On the first cycle on which both `report_valid` and `report_ready` are sampled high simultaneously, the beat is considered committed. `report_valid` is deasserted on the following cycle and the module returns to its idle state.

Back-pressure isolation and Arbitration

The case in which `report_ready` is held low for an extended interval is the concern of the design. The module remains in the *Hold* phase indefinitely, but the monitoring system may continue to send multiple reporting requests, which are stored in an internal First-In-First-Out (FIFO) buffer. When the HOLD state is released, SIC resumes transmission starting from the oldest request in the FIFO. From the perspective of the monitoring system, the behavior of the system is therefore independent of whether the external receiver is operational, busy, or absent.

Another case arises when multiple monitoring systems issue requests that reach the SIC simultaneously, causing a conflict. In this situation, the SIC must perform arbitration to decide which request is transmitted first and which is stored in the FIFO. The arbitration depends on two factors:

(1) Type of request — as mentioned in Section 3.4, there are two reporting policies, namely periodic reporting and event-triggered reporting, and within the SIC, event-triggered reporting has higher priority than periodic reporting;

- periodic reporting, derived from a free-running counter that overflows every `report_period` `pclk` cycles. The periodic trigger emits report beats at a low rate on a fixed interval, so that the receiver can confirm that the RFF subsystem remains operational and that its internal state continues to advance.

- event-triggered reporting, derived from rising-edge detection on the internal indicators, for example:`calib_done`, `freqm_valid`, The event trigger causes a beat to be emitted as soon as a meaningful change in the internal state occurs.

(2) Configuration — when the conflicting requests are of the same type, the CPU-defined configuration determines which monitoring system's request has higher priority.

Representative scenarios

The operation of the arbitration logic is most easily illustrated through a small number of representative scenarios. In each of the timing diagrams below, the signals shown are `pclk` as the reference clock. For simplicity, the SIC takes inputs from only two monitoring systems, so there are only two pending_req signals: `pending_req1` and `pending_req2`. `event_tick1` and `event_tick2` means the request is event-triggered. `report_valid` and `report_ready` for the external handshake.

Scenario A — isolated request, no back-pressure. A single event-triggered req arrives while the module is idle and `report_ready` stays high (Figure 3.10). The beat is launched on the next cycle, committed immediately, and `seq` advances once. No arbitration occurs.

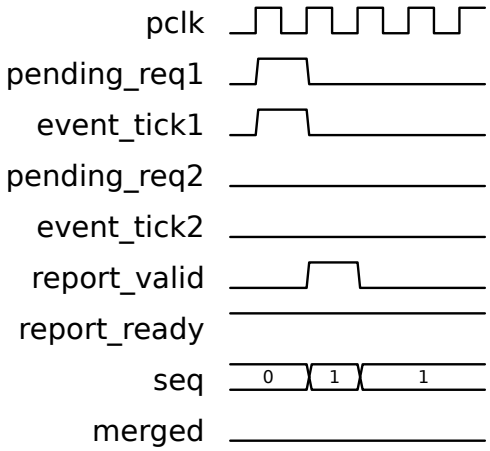


Figure 3.10: Scenario A: an isolated trigger is dispatched and committed in successive cycles.

Scenario B — event and periodic triggers on the same cycle. Both sources assert requests in the same cycle: `pending_req1` is event-triggered, while `pending_req2` is periodic. The event-triggered wins dispatch ; the periodic tick is folded into FIFO and emitted in the next idle cycle. This illustrates the first priority rule.

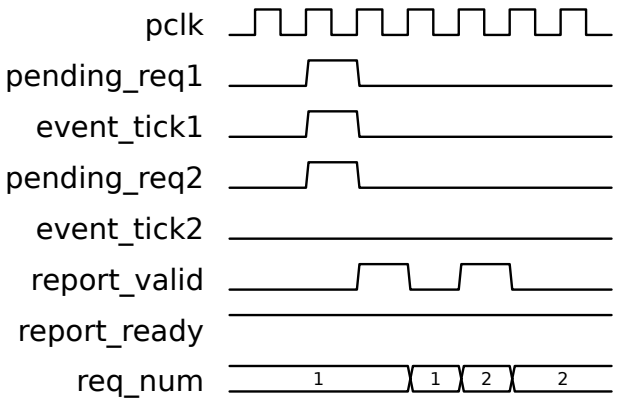


Figure 3.11: Scenario B: conflict happens. event wins, periodic is deferred to the FIFO.

Scenario C — two event-triggered requests conflict. Both sources assert request together while `pending_req1` and `pending_req1` are both event-triggered. The priority is decided by configuration. If the CPU has previously configured the SIC such that monitoring system 1 has higher priority than monitoring system 2, then

`pending_req1` wins the dispatch. The waveform is shown in the Figure 3.12 and is nearly identical to Scenario B.

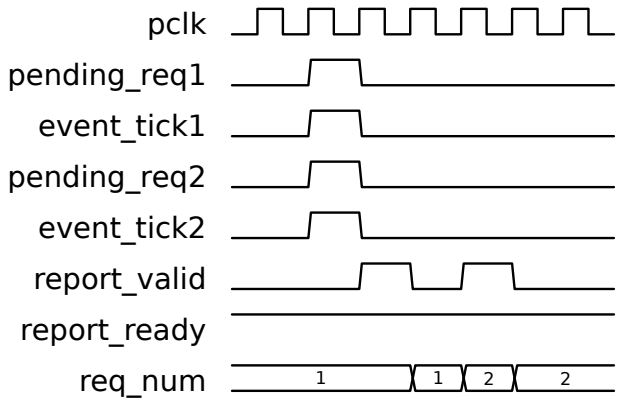


Figure 3.12: Scenario C: a first event launches under back-pressure; two further events fold into the single pending slot; only two beats are committed.

3.8 CDC and Status Synchronization

3.8.1 CDC Role, Clock Domains, and Top-Level Paths

The CDC synchronization layer connects the software-visible APB domain, the work-clock domain, the VCRO/VCLK observation boundary, and the report interface. Its purpose is *not* to perform frequency measurement or calibration by itself. Instead, it makes configuration events, completion pulses, status snapshots, and report payloads cross clock boundaries without losing events or tearing multi-bit data.

The layer is therefore not a standalone algorithm block in the functional sense. It is the infrastructure that lets APB configuration, calibration completion, frequency-meter results, and SIC/report traffic remain coherent across domains. The main risks addressed are short-pulse loss, multi-bit tearing, stale status, reset-induced false events, and back-pressure-related payload overwrite.

Table 3.4 summarizes the principal clock and reset domains. The important point is not merely that several clocks exist, but that each clock *owns different state*. A start bit written in `pclk` is not a start event in `wclk` until the CDC protocol has accepted it. Similarly, a result produced in `wclk` is not software-visible in `pclk` until it has been captured coherently.

Table 3.4: Clock and reset domains in the RFF subsystem.

Domain	Clock / reset	Main responsibility	Typical boundary signals
APB domain	`pclk`, `prst_n`	Register write/read; software control and status	`calib_start`, cfg update, `calib_done_p`, `freq_code` readout
Work domain	`wclk`, `wrst_n`	Calibration FSM, delay update, window control	`calib_done_w`, `high_cnt`, `low_cnt`, tuned result
Observation boundary	`vclk` / `vcro_clk`	Measured or observed timing source	Edge events, sampled clock state, meter count
Report path	`pclk` or dedicated report clock	Payload transfer to system interface	`tx_valid`, `tx_ready`, payload
Reset boundary	Per-domain release	Clears pending events, toggles, sticky flags	`prst_n`, `wrst_n`, system reset tree

When the report path is clocked identically to `pclk` in a given implementation, it is still treated here as a logically separated egress path with valid-ready semantics and payload hold requirements.

Functionally, the CDC layer has three coupled responsibilities:

1. carrying software control from `pclk` into the `wclk` work domain;

2. returning completed-window snapshots from `wclk` to `pclk` on synchronized done events;

3. protecting multi-bit report payloads while an external consumer applies back-pressure.

Figure 3.13 gives a top-level view. Table 3.5 maps typical paths to the CDC method and the protected failure mode.

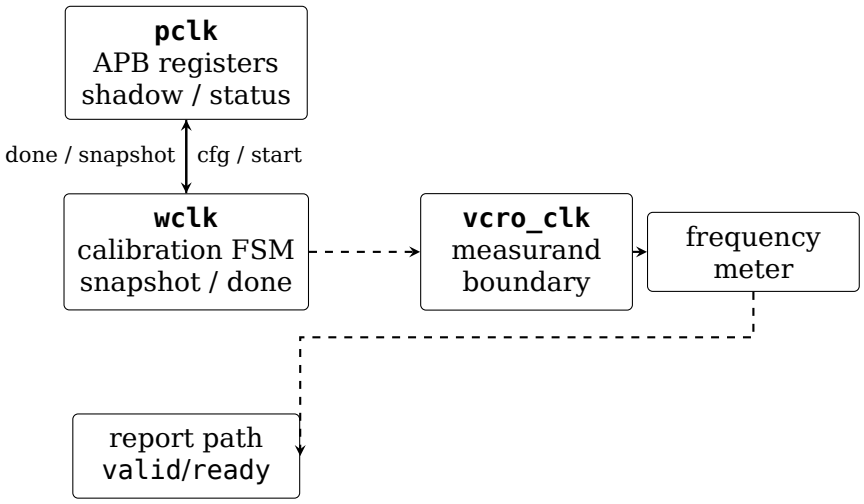


Figure 3.13: Top-level CDC paths in the RFF subsystem: control into `wclk`, status and results back to `pclk`, measured-clock boundary to the frequency meter, and report egress with handshake hold.

Table 3.5: CDC path mapping: typical signals, methods, and protected failures.

CDC path	Typical signals	Method	Protected failure
`pclk` → `wclk` config	`calib_start`, thresholds, window	toggle-based event synchronization; stable shadow capture	lost start; half-updated multi-bit fields
`wclk` → `pclk` status	`calib_done`, `calib_hcnt`	pulse/toggle synchronizer + snapshot registers	missed pulse; incoherent counter pair
measurement result → `pclk` readout	`freq_code`	static readout after counter stopped	multi-bit tearing
internal → report	`report_payload`	valid-ready hold	payload overwrite under backpressure
reset boundary	toggles, sticky flags, `valid`	clear + per-domain release	false done / false valid

3.8.2 Control, Status, and Result Crossing

Configuration fields are written by firmware in the `pclk` domain, but they are consumed by the calibration and measurement logic in the

wclk domain. A direct combinational connection would let the target FSM observe partially updated fields or miss a one-cycle start pulse.

The start event is treated as an *event crossing* rather than as a level. A write to the APB start bit creates a source-domain event, for example a toggle. The event is synchronized into wclk and edge-detected there, producing exactly one accepted calib_start_w or equivalent event per qualifying software action.

Multi-bit fields such as delay code, calib_period, max_iters, and imbalance tolerance are first held stable in pclk shadow registers. The wclk domain captures them only when the update protocol has been accepted and the local FSM is in a safe state. This avoids visible stitching across bit slices.

Table 3.6: Configuration crossing from pclk to wclk (representative fields).

Source field	From	To	CDC method	Meaning in wclk
calib_start	pclk	wclk	toggle-based event synchronization	one accepted calibration start
calib_period	pclk	wclk	shadow capture	measurement window length
match_threshold	pclk	wclk	shadow capture	high/low imbalance tolerance
max_iters	pclk	wclk	shadow capture	iteration ceiling
auto_recalib_en	pclk	wclk	level synchronization or captured config	periodic re-lock enable

The most important return path is the *completed-window snapshot*. The calibration FSM produces high_cnt, low_cnt, tuned result, calib_status, and imbalance metrics in the wclk domain. These values must be read by APB software in pclk, but they must not be sampled as unrelated live wires. The wclk domain therefore freezes a coherent snapshot before launching the done event.

The sequence is fixed in three steps:

1. wclk: freeze high_cnt, low_cnt, status, and tuned word for the completed window;
2. wclk: assert a short calib_done_w pulse, or use a toggle encoding, into a pulse/toggle synchronizer toward pclk;
3. pclk: on the synchronized done event, capture the full payload into shadow registers visible to software.

high_cnt and low_cnt are coherent only as a *completed-window pair*. Arbitrary APB reads must not reconstruct a window by sampling live counters on the two sides of the domain boundary.

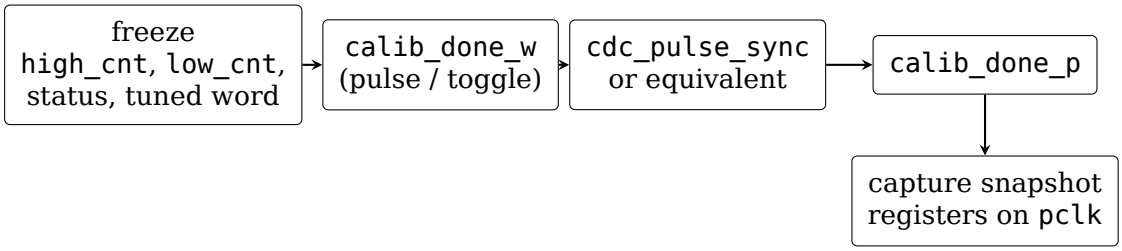


Figure 3.14: Completed-window snapshot capture across wclk and pclk. The counters and status are frozen before the done event; the synchronized pulse in pclk triggers coherent sampling of the full payload.

Table 3.7: Status and result capture from wclk to pclk.

Returned item	Produced in	Visible in	Capture rule
calib_done	wclk	pclk	pulse/toggle synchronizer or sticky status bit
calib_status	wclk	pclk	sampled with done snapshot
calib_hcnt	wclk	pclk	completed-window counter, frozen with calib_lcnt
calib_lcnt	wclk	pclk	completed-window counter, frozen with calib_hcnt
calib_result / tuned	wclk	pclk	stable code sampled after done
calib_err_abs	wclk	pclk	imbalance metric with the same snapshot

The measured clock is not a conventional control signal. It is the object being counted. Its relationship to the frequency meter therefore cannot be reduced to a two-flop synchronizer on a data bit while preserving measurement meaning.

Typical contracts are:

- a window enable or run command crosses from a slower domain into the meter domain with a level synchronizer, because the enable persists for many cycles of the measurand clock;

- a completion event crosses back with a toggle or pulse synchronizer, because a naive pulse may be shorter than a destination period;
- the multi-bit `freq_code` is read in `pclk` only after the counter has stopped or latched, giving *static readout* and avoiding bit-tearing;
- `vcro_clk` itself remains local to the counting structure: it is counted, not synchronized as ordinary data.

Table 3.8: Measured-clock boundary, aligned with the frequency meter.

Boundary	Signal example	Direction	CDC strategy	Reason
Window enable	`window_en_pclk`	`pclk` → meas.	two-flop level synchronization	slow level, long compared with fast clock
Done event	`measurement_done`	meas. → `pclk`	toggle-based event synchronization	short source pulse
Result code	`freq_code`	meas. → `pclk`	static readout after stop	avoid multi-bit tear
Measured clock	`vcro_clk`	local to counter	counted, not data-synchronized	it is the measurand

The detailed timing diagram and counter gating belong in Section 3.6. Here only the CDC contract is fixed. This mechanism documents the digital readout protocol and static result capture. It does not by itself prove analog VCRO jitter, silicon-level frequency accuracy, or metastability MTBF.

3.8.3 Report Path and Reset Handling

Even when the report path shares `pclk`, it behaves like a boundary to a potentially slower or asynchronous consumer. The main boundary requirement is *payload stability under back-pressure*.

A report beat is formed from internal status and frequency information only after the relevant sources have been latched. Once `report_valid` is asserted, `report_payload` remains stable until `report_ready` is sampled high. This isolates calibration and measurement from external receiver timing and prevents tearing at the receiver.

Table 3.9: Report path handshake rules.

Signal	Rule	Failure avoided
`report_valid`	remains asserted until the beat is accepted	lost beat
`report_ready`	may be de-asserted by the receiver	uncontrolled sink timing
`report_payload`	stable while valid and not ready	payload tearing
`pending_req` if used	merges deferred triggers	unbounded queue growth
`trig_merged_flag` if used	records collapsed events	silent loss of observability

Reset is also treated as a boundary condition because each domain releases independently. The design uses domain-local reset synchronization so that event toggles, sticky completion flags, and pending report state restart from a known relation. After reset, no done pulse, valid frequency code, or report beat may be produced until a new qualified start sequence is accepted.

Table 3.10: Reset clearing expectations.

Reset item	Expected cleared state	Reason
event toggle / pulse FSM	source and target aligned	avoid false start after reset
`calib_done` sticky	de-asserted	avoid false lock indication
`freq_valid`	de-asserted	avoid consuming invalid result
`pending_req`	cleared	avoid stale report merge
`report_valid`	de-asserted	avoid uninitialized payload
snapshot / shadow registers	benign default until valid	avoid stale APB read

3.8.4 CDC Layer Summary

This section defines the CDC synchronization layer of the RFF subsystem. Its role is to connect the APB configuration domain, the work-clock calibration domain, the measured-clock boundary, and the report path without losing events or tearing multi-bit state.

Three design choices structure the layer. *First*, `pclk`-to- `wclk` control uses event synchronization and stable configuration capture rather than naive single-cycle pulse stretching. *Second*, `wclk`-to-`pclk` status uses completed-window snapshots and synchronized done events so high/low counters and tuned codes are captured coherently. *Third*, the report path holds payloads under valid-ready back-pressure, isolating the feedback path from external receiver timing.

Together these choices give the rest of the RFF system software-visible, clock-domain-safe control and observation, without replacing the functional sections dedicated to calibration, the VCRO model, the frequency meter, or the SIC transmitter.

3.9 RTL Implementation Summary

The RFF RTL implements the digital part of the runtime frequency feedback path. It provides register configuration, VCRO control mapping, calibration search support, frequency measurement, clock-domain crossing and status synchronization, and payload output. The analog behavior of the VCRO macro is abstracted for RTL simulation, while the digital control path is implemented and verified as a module-level design.

The hardware implementation follows the system requirements from Chapter 2. The calibration path provides an initial locked configuration. The frequency meter provides a digital observation result. The status and payload paths make the result usable by the system. The next chapter verifies whether these implemented RTL functions behave correctly under the defined unit-test scope.

Chapter 4

Functional Verification of RFF

4.1 Verification Target and Scope

This chapter verifies the RTL implementation of the Runtime Frequency Feedback (RFF) module. The goal is not to repeat the behavioral power analysis from the previous chapter. The goal is to check that the RTL can complete the functions defined for the current unit-test scope.

The verification target is the RFF top module, `rff_top`. The main checks cover reset behavior, APB register access, frequency measurement, VCRO control, calibration control, status readback, and SIC_TX payload output. These functions form the main RTL path of the RFF module. A valid test must show that configuration data can enter the module, internal control and measurement logic can react to the configuration, and the result can be read back or sent out through the defined output path.

The scope is limited to module-level RTL unit testing. DFT mode, AVFS VCRO analog pins, system-level integration, post-layout timing, and real PVT behavior are outside this chapter. These boundaries are important for the later waiver analysis. A signal that belongs to DFT mode, analog power pins, or reserved system integration paths is not treated as a failure of the current RFF functional verification.

4.2 Verification Platform and Method

The verification platform uses `rff_top` as the DUT. Test sequences generate APB accesses, frequency inputs, calibration triggers, and payload sending scenarios. Drivers apply these stimuli to the DUT. Moni-

tors sample key interface and internal signals. Checkers compare the observed behavior with the expected result. A scoreboard keeps the register mirror and the expected frequency and payload models. The coverage collector records both code coverage and functional coverage.

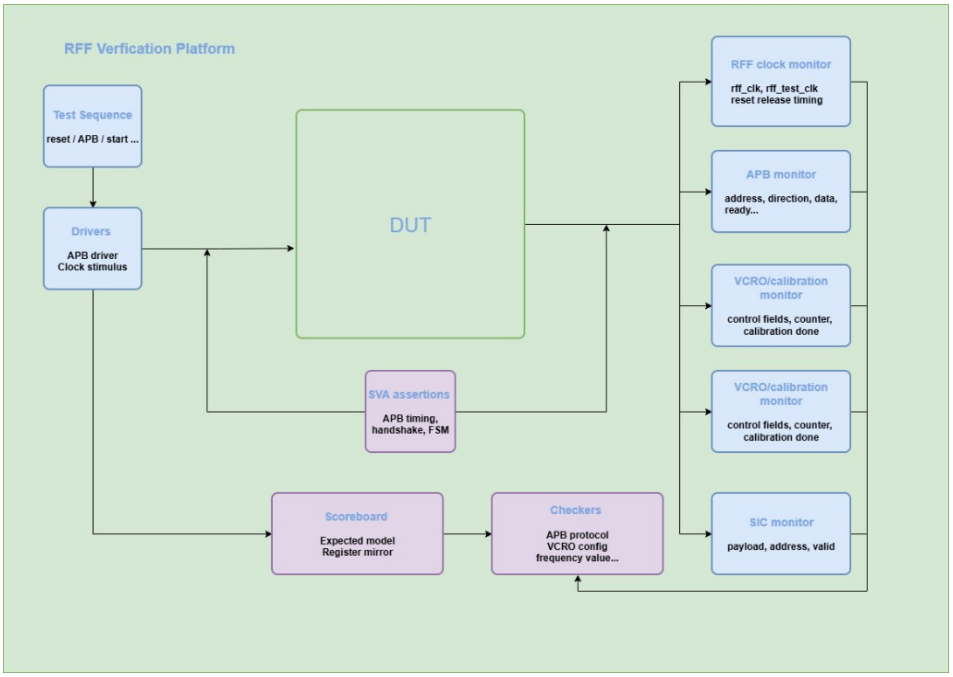


Figure 4.1: RFF unit-test verification platform

The RFF clock monitor observes `rff_clk`, `rff_test_clk`, and the reset release timing. It confirms that the clock environment is valid before other checks start. This is needed because the frequency meter and payload sending logic both depend on a correct time base.

The APB monitor samples `psel`, `penable`, `pwrite`, `paddr`, `pwdata`, `prdata`, `pready`, and `pslverr`. It records each register access as a transaction. It also marks whether the access targets a valid register, a reserved field, or a hole address. The APB checker then checks the protocol timing and the data result. It compares register writes with the scoreboard mirror and compares readback data with the expected value.

The frequency meter monitor samples `freqm_raw_code`, `freqm_code`, the measurement enable signal, and the measurement window. The frequency meter checker calculates the expected

count range from the input frequency and the measurement window. It compares only the valid range defined by the RTL data format. This avoids treating unused counter headroom as a functional error.

The VCRO and calibration monitor observes control fields, calibration enable, calibration start, calibration counters, and the done flag. The VCRO configuration checker compares APB-written fields with the control outputs. The calibration checker checks the order of start, count, and done. It also checks that the status readback agrees with the internal calibration state.

The SIC_TX payload monitor samples the payload data, payload address, and valid signal. The payload checker builds the expected payload from the current send mode, frequency result, soft-read data, and configured address. It checks that payload valid is asserted only when the send condition is met.

The scoreboard connects these local checks. It stores a register mirror for APB writes. It updates the expected frequency result after a frequency-meter transaction. It builds the expected payload after the payload source is selected. This makes the check path continuous from configuration input to final output. SVA assertions are also used for reset, APB timing, handshake behavior, and legal FSM transitions.

4.3 Functional Testcase Design

The testcase design follows the RFF functional path. Each testcase has a clear target. It either checks reset and clock behavior, APB access, frequency measurement, VCRO control, calibration, payload output, status reporting, or coverage closure.

Reset and clock cases check the initial state and reset recovery. `TC_RFF_RESET_DEFAULT_STATE` checks that registers, valid signals, done flags, and FSM states return to their default values during reset. `TC_RFF_RESET_DURING_ACTIVE_OP` applies reset during an active frequency count or payload send. The expected result is that the current operation is cleared and the module can restart after reconfiguration. `TC_RFF_CLOCK_STABLE_AFTER_RESET` checks that the RFF clocks are stable after reset release.

APB cases check register access. `TC_RFF_APB_SINGLE_WRITE_READ` writes and reads one valid register. `TC_RFF_APB_REG_SWEEP` traverses all defined register addresses. `TC_RFF_APB_HOLE_ACCESS` accesses undefined address space and checks that valid registers are not corrupted. `TC_RFF_APB_RESERVED_BITS` writes patterns to reserved bits

and checks that valid fields are not affected.

Frequency meter cases check the measurement path. TC_RFF_FREQM_BASIC_COUNT checks one fixed input frequency and one fixed measurement window. TC_RFF_FREQM_MULTI_FREQ_SWEEP applies low, middle, and high frequency inputs. TC_RFF_FREQM_PERIOD_SWEEP changes the measurement window. TC_RFF_FREQM_READBACK reads the frequency result through APB and compares it with the monitored internal result.

VCRO and calibration cases check the control path. TC_RFF_VCRO_LVT_CONFIG and TC_RFF_VCRO_LVTLL_CONFIG verify the mapping from register fields to VCRO control outputs. TC_RFF_CALIB_START_DONE checks the start, count, and done sequence. TC_RFF_CALIB_COUNTER_READBACK checks that the calibration status can be read back.

Payload cases check the output path. TC_RFF_PAYLOAD_FREQM_MODE sends the frequency-meter result. TC_RFF_PAYLOAD_SOFT_READ_MODE sends soft-read data. TC_RFF_PAYLOAD_PERIODIC_SEND checks the configured send period. TC_RFF_PAYLOAD_NO_SPURIOUS_VALID checks that no false valid pulse is generated when the send condition is not met.

Status and closure cases complete the plan. TC_RFF_STATUS_IDLE_BUSY_DONE checks the status sequence from configuration to completion. TC_RFF_SOFT_READ_ADDR_ALIGN checks the address alignment rule. TC_RFF_COVERAGE_BOUNDARY applies boundary values to major fields and supports coverage closure.

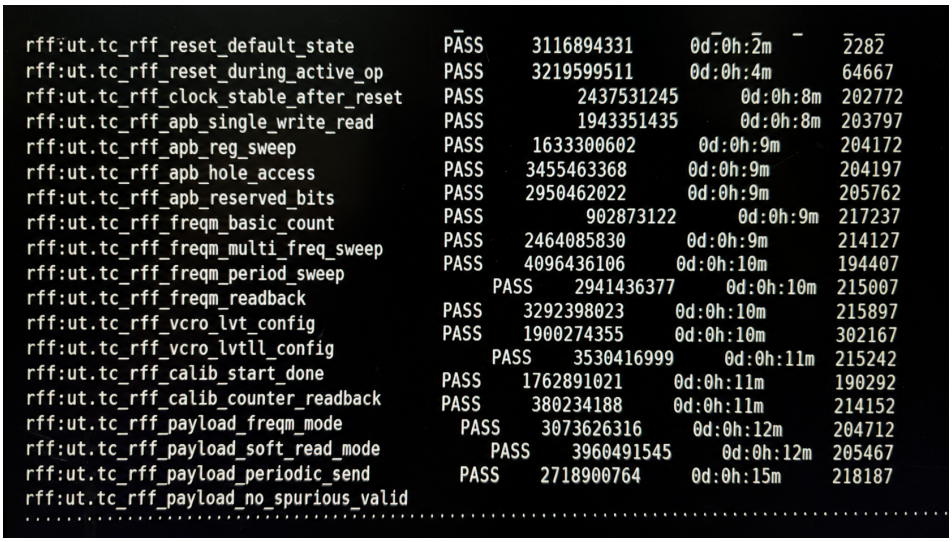


Figure 4.2: RFF testcase regression result

The regression result shows that the selected unit-test cases pass. These cases cover reset, APB access, frequency measurement, VCRO control, calibration, payload output, status readback, and coverage closure.

4.4 Code Coverage Results and Waiver Analysis

4.4.1 Overall Code Coverage

Code coverage is used to check whether RTL statements, FSM states, conditions, branches, and signal toggles are exercised by the regression. The overall RFF unit-test code coverage score is 97.70%. Line coverage is 99.00%. FSM coverage is 100.00%. Condition coverage is 98.72%. Branch coverage is 99.47%. Toggle coverage is 91.31%.

Table 4.1: RFF unit-test code coverage summary

Metric	Result	Interpretation
Overall score	97.70%	Executable RTL paths are highly covered at unit-test level.
Line coverage	99.00%	Most reachable statements are executed by the regression.
Toggle coverage	91.31%	Residual gaps are mainly caused by DFT signals, reserved interfaces, aligned address bits, and range-unreachable bits.
FSM coverage	100.00%	Legal states and legal state transitions are covered.
Condition coverage	98.72%	Most conditional expressions observe the required truth combinations.
Branch coverage	99.47%	Normal functional branches are covered. Residual items are waiver candidates.

The remaining analysis focuses on the residual toggle and branch gaps. These gaps are grouped by design reason. This keeps the waiver discussion tied to the RTL structure rather than treating every uncovered bit as a separate problem.

4.4.2 Top-Level Toggle Gaps

At the `u_rff_top` level, the toggle gaps mainly come from DFT ports, reserved input interfaces, and unused APB address or data bits. The used APB control signals, RFF clock outputs, and payload output path are covered. The uncovered items are concentrated in `dft_mode`, `dft_rst_n`, `tx_data_i`, `tx_valid_i`, and selected address/data bits.

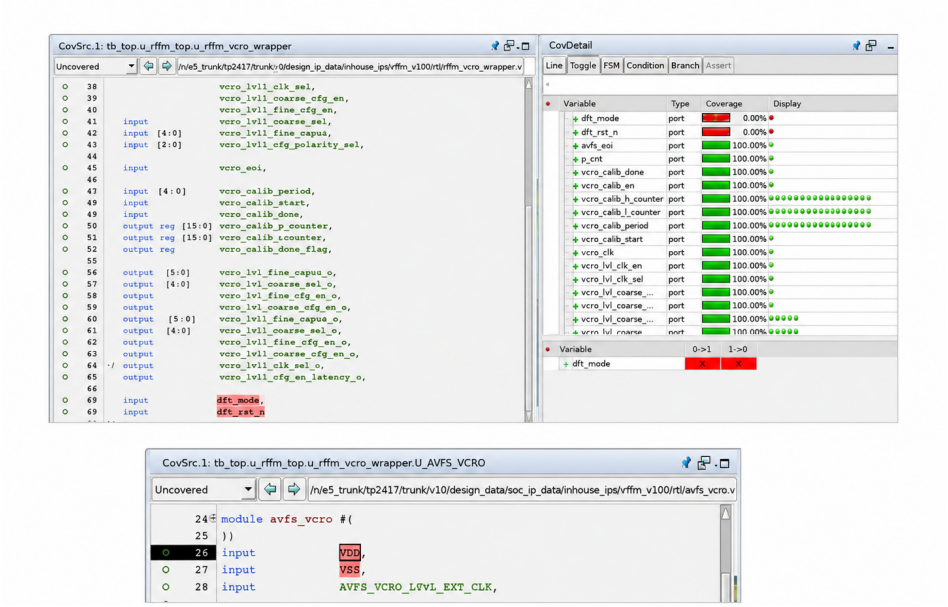


Figure 4.3: Top-level toggle coverage gaps

The DFT signals stay fixed because DFT mode is not part of the current normal-function unit test. The `tx_data_i` and `tx_valid_i` inputs are reserved upstream inputs in this version. The address and data gaps are explained by the register map and APB alignment rule.

4.4.3 APB Address and Data Bit Waiver

Some APB address and data bits do not toggle in the current regression. This is expected from the register map. The RFF register space uses a limited address range, so high address bits may remain fixed. APB accesses are 4-byte aligned, so the low address bits do not need to toggle.

For data signals, the unused high bits of `pwdata`, `prdata`, and `soft_read_data` are reserved or not defined in the current map. The checkers compare the defined fields. Reserved high bits are not used to decide the result of configuration, measurement, readback, or payload sending.

4.4.4 Frequency Meter Range-Related Gaps

The frequency meter has wider internal counters than the range reached by the current test settings. The input frequency range and

the measurement window do not drive all high counter bits. The lower and middle bits still show the count variation in the valid measurement range.

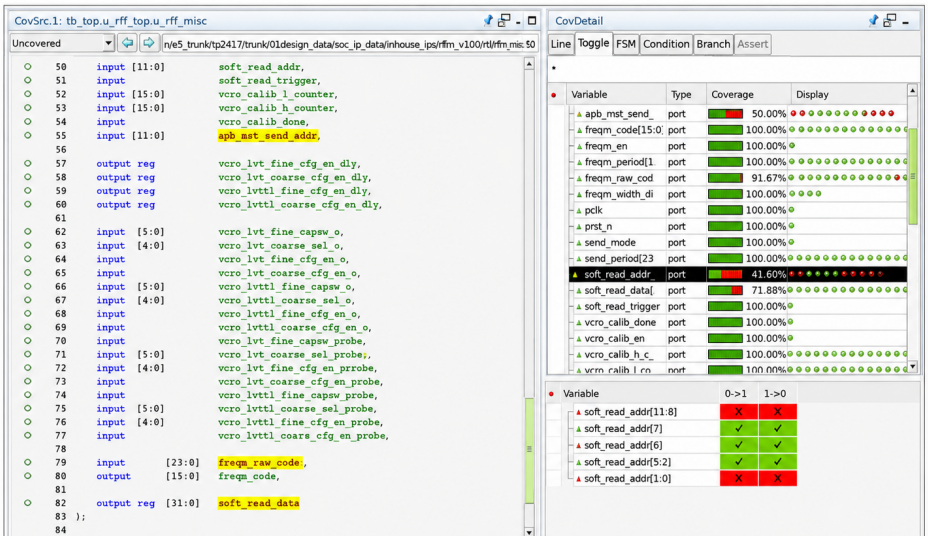


Figure 4.4: Frequency-meter and soft-read related toggle gaps

The frequency meter checker compares the expected count range with `freqm_raw_code` and `freqm_code`. It follows the RTL output width and data format. High bits that cannot be reached by the current frequency range are treated as range-unreachable bits during coverage closure.

4.4.5 VCRO Wrapper and DFT/AVFS Gaps

The VCRO wrapper contains DFT signals and AVFS VCRO boundary signals. The DFT signals include `dft_mode` and `dft_rst_n`. These signals are fixed during normal-function unit tests. The AVFS VCRO power or analog boundary pins, such as VDD and VSS, are not checked as normal digital function ports in this RTL unit test.

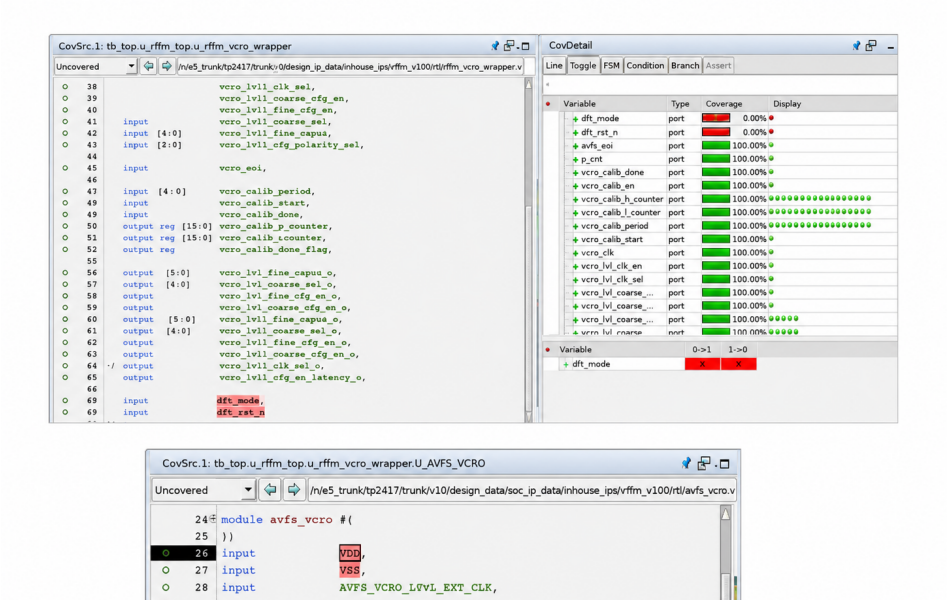


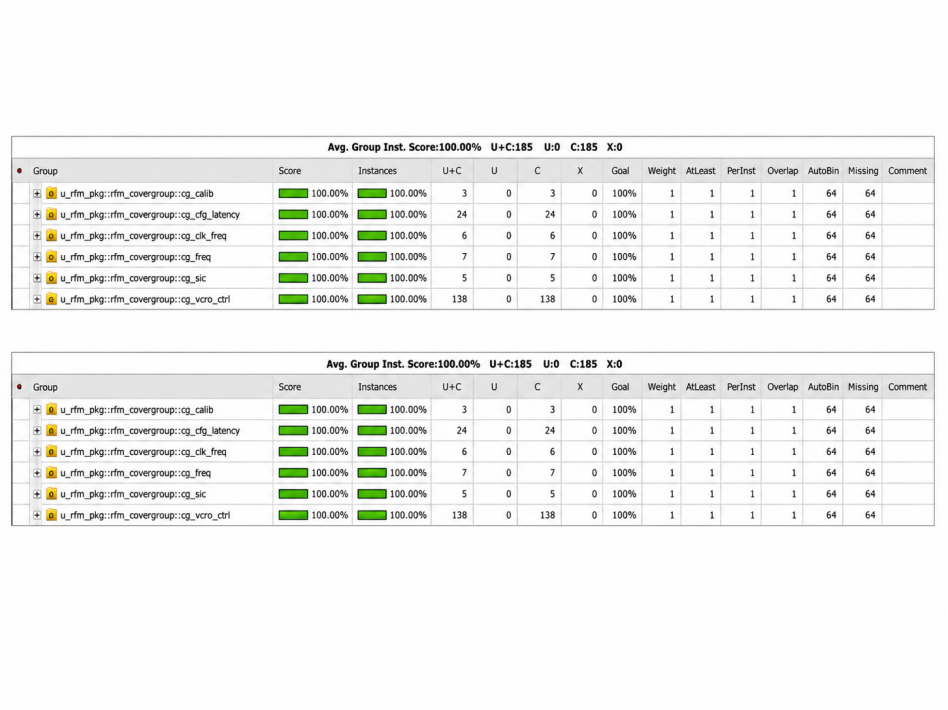
Figure 4.5: VCRO wrapper DFT and AVFS related coverage gaps

These items define the verification boundary. They are not missing functional behavior in the current RFF unit-test scope.

4.5 Functional Coverage Results

Functional coverage checks whether the planned RFF function combinations have been triggered by the testcases. It does not only ask whether RTL statements are executed. It checks whether the intended scenarios are covered.

The functional coverage model includes six main covergroups. `cg_calib` covers calibration enable, start, done, calibration period, and calibration counter/status bins. `cg_cfg_latency` covers configuration latency, capture latency, and related cross combinations. `cg_clk_freq` covers reference clock bins, measured frequency bins, and low, nominal, and high frequency ranges. `cg_freq` covers `freqm_en`, `freqm_period`, `freqm_raw_code`, and `freqm_code` bins. `cg_sic` covers send mode, send period, soft-read trigger, and send address bins. `cg_vcro_ctrl` covers VCRO coarse, fine, cap, clock select, enable, and control cross combinations.



Avg. Group Inst. Score:100.00% U+C:185 U:0 C:185 X:0

Group	Score	Instances	U+C	U	C	X	Goal	Weight	AtLeast	PerInst	Overlap	AutoBin	Missing	Comment
u_rfm_pkg::rfm_covergroup::cg_calib	100.00%	100.00%	3	0	3	0	100%	1	1	1	1	64	64	
u_rfm_pkg::rfm_covergroup::cg_cfg_latency	100.00%	100.00%	24	0	24	0	100%	1	1	1	1	64	64	
u_rfm_pkg::rfm_covergroup::cg_clk_freq	100.00%	100.00%	6	0	6	0	100%	1	1	1	1	64	64	
u_rfm_pkg::rfm_covergroup::cg_freq	100.00%	100.00%	7	0	7	0	100%	1	1	1	1	64	64	
u_rfm_pkg::rfm_covergroup::cg_sic	100.00%	100.00%	5	0	5	0	100%	1	1	1	1	64	64	
u_rfm_pkg::rfm_covergroup::cg_vcro_ctrl	100.00%	100.00%	138	0	138	0	100%	1	1	1	1	64	64	

Avg. Group Inst. Score:100.00% U+C:185 U:0 C:185 X:0

Group	Score	Instances	U+C	U	C	X	Goal	Weight	AtLeast	PerInst	Overlap	AutoBin	Missing	Comment
u_rfm_pkg::rfm_covergroup::cg_calib	100.00%	100.00%	3	0	3	0	100%	1	1	1	1	64	64	
u_rfm_pkg::rfm_covergroup::cg_cfg_latency	100.00%	100.00%	24	0	24	0	100%	1	1	1	1	64	64	
u_rfm_pkg::rfm_covergroup::cg_clk_freq	100.00%	100.00%	6	0	6	0	100%	1	1	1	1	64	64	
u_rfm_pkg::rfm_covergroup::cg_freq	100.00%	100.00%	7	0	7	0	100%	1	1	1	1	64	64	
u_rfm_pkg::rfm_covergroup::cg_sic	100.00%	100.00%	5	0	5	0	100%	1	1	1	1	64	64	
u_rfm_pkg::rfm_covergroup::cg_vcro_ctrl	100.00%	100.00%	138	0	138	0	100%	1	1	1	1	64	64	

Figure 4.6: RFF functional coverage result

The reported functional coverage reaches 100% for the listed covergroups. This means the defined calibration, configuration latency, clock/frequency, frequency meter, SIC_TX payload, and VCRO control scenarios are all triggered by the regression. Together with the checker results, this supports the functional completeness of the current unit-test scope.

4.6 Chapter Summary

This chapter verified the RFF RTL at module level. The unit-test environment covers reset, APB configuration and readback, frequency measurement, VCRO control, calibration, SIC_TX payload output, and status reporting. The monitors, checkers, scoreboard, and assertions build a continuous verification path from input configuration to observed output.

The code coverage result is high. FSM coverage reaches 100%, and the overall score reaches 97.70%. The remaining toggle gaps are explained by DFT/AVFS boundary signals, reserved interfaces, address

alignment, register reserved fields, and range-unreachable counter bits. These gaps are handled as waiver items with explicit design reasons.

The functional coverage result reaches 100% for the planned covergroups. This shows that the main RFF functional scenarios have been exercised. Under the defined RTL unit-test scope, the verification result supports RFF module functional signoff for the current implementation.

Chapter 5

Conclusion and Future Work

5.1 Thesis Summary

This thesis studied Runtime Frequency Feedback for low-power management support. The work started from the fixed-margin problem. A conservative guardband can protect timing safety, but it can also keep more margin than needed under typical runtime conditions. RFF was introduced as a feedback source that reports runtime frequency capability through a VCRO-based observation path.

Chapter 2 developed the system-level concept and behavioral model. The Baseline-vs-Feedback comparison used the same disturbance profile, target, time horizon, and initial condition. The behavioral results showed that feedback can lower the average equivalent margin and reduce cumulative energy under the normalized model. The model also included feedback hardware overhead, so the benefit was not treated as free. Under the selected parameters, the feedback net energy remained lower than the fixed-margin baseline when the overhead stayed below the break-even value.

Chapter 3 described the RTL-oriented hardware implementation. The RFF design includes a register interface, VCRO digital abstraction, calibration module, frequency measurement unit, normal-mode control, status logic, and payload output path. The chapter also clarified the boundary between RTL behavior and analog VCRO signoff.

Chapter 4 verified the RFF RTL at module level. The verification environment checked reset, APB register access, VCRO control, calibration, frequency measurement, status readback, payload output, and coverage closure. The reported code coverage and functional coverage support the completeness of the current unit-test scope. Remaining coverage gaps were explained by DFT/AVFS boundary signals,

reserved interfaces, address alignment, reserved fields, and range-unreachable counter bits.

5.2 Main Contributions

The first contribution is the RFF system definition. The thesis separates workload operating frequency from runtime frequency capability. This distinction makes the feedback object clear and avoids treating VCRO output as the workload clock itself.

The second contribution is the controlled behavioral comparison. The Baseline and Feedback cases use the same input conditions, and only the margin trajectory is changed. This setup makes the energy trend traceable and reduces the risk of over-claiming.

The third contribution is the RTL implementation of the RFF digital path. The design connects configuration, calibration, frequency measurement, status reporting, and payload output into one module-level structure.

The fourth contribution is the verification evidence. Directed test cases, checkers, waveform analysis, code coverage, functional coverage, and waiver analysis are used to support the functional correctness of the implemented RFF RTL under the defined scope.

5.3 Limitations

The results have clear boundaries. The behavioral model uses normalized parameters. It shows a trend, not a measured silicon power number. The VCRO behavior in RTL simulation is an abstraction. It does not replace analog simulation, mixed-signal verification, layout-aware analysis, or silicon characterization.

The RTL verification is also limited to module-level functional verification. It does not cover full-chip power-management policy, complete DVFS/AVFS integration, post-layout timing, or physical implementation overhead. The coverage waiver items are reasonable for the current unit-test scope, but they do not remove the need for later integration-level checks.

The energy-saving conclusion should therefore be read carefully. RFF can reduce fixed-margin waste when the controlled domain is large enough and when feedback hardware overhead remains low. The

thesis does not claim that RFF always saves power under every design scale, every workload, or every PVT condition.

5.4 Future Work

Future work can extend this thesis in four directions. First, the VCRO model can be improved with circuit-level simulation data and PVT characterization. This would make the frequency capability model closer to real silicon behavior. Second, the RFF module can be integrated with a full DVFS or AVFS controller, so that the feedback result can drive a complete voltage/frequency policy.

Third, the implementation can be evaluated after synthesis and physical design. Area, timing, and power overhead should be measured rather than estimated. Gate-level simulation and post-layout timing analysis can further check implementation robustness. Fourth, verification can be extended with constrained-random tests, formal checks for protocol properties, CDC analysis, and system-level regression. These steps would move RFF from module-level proof toward stronger integration-level signoff.

References

[1] Mark Bohr. “A 30 Year Retrospective on Dennard’s MOSFET Scaling Paper”. In: *IEEE Solid-State Circuits Society Newsletter*. Vol. 12. 1. 2007, pp. 11–13.

[2] Karl Rupp. *50 Years of Microprocessor Trend Data*. 2024. URL: `https://github.com/karlrupp/microprocessor-trend-data` (visited on 04/24/2026).

[3] Eric Masanet et al. “Recalibrating Global Data Center Energy-Use Estimates”. In: *Science* 367.6481 (2020), pp. 984–986.

[4] Jan M. Rabaey, Anantha Chandrakasan, and Borivoje Nikolić. *Digital Integrated Circuits: A Design Perspective*. 2nd. Prentice Hall, 2003.

[5] Hadi Esmaeilzadeh et al. “Dark Silicon and the End of Multicore Scaling”. In: *Proceedings of the 38th Annual International Symposium on Computer Architecture (ISCA)*. 2011, pp. 365–376.

[6] Synopsys, Inc. *What is Low Power Design?* 2024. URL: `https://www.synopsys.com/glossary/what-is-low-power-design.html` (visited on 05/09/2026).

[7] Michael Keating et al. *Low Power Methodology Manual: For System-on-Chip Design*. Springer, 2007.

[8] Shin'ichiro Mutoh, Takakuni Douseki, Yasuyuki Matsuya, et al. "1-V Power Supply High-Speed Digital Circuit Technology with Multithreshold-Voltage CMOS". In: *IEEE Journal of Solid-State Circuits* 30.8 (1995), pp. 847–854.

[9] Intel Corporation. *Enhanced Intel SpeedStep Technology for the Intel Pentium M Processor*. 301170-001. White Paper. Intel Corporation. 2004.

[10] Linux Kernel Community. *CPU Performance Scaling*. Linux Kernel Documentation. 2024. URL: `https://docs.kernel.org/admin-guide/pm/cpufreq.html` (visited on 05/09/2026).

[11] Dieter K. Schroder and Jeff A. Babcock. "Negative Bias Temperature Instability: Road to Cross in Deep Submicron Silicon Semiconductor Manufacturing". In: *Journal of Applied Physics* 94.1 (2003), pp. 1–18.

[12] Dan Ernst, Nam Sung Kim, Shidhartha Das, et al. "Razor: A Low-Power Pipeline Based on Circuit-Level Timing Speculation". In: *Proceedings of the 36th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*. 2003, pp. 7–18.

[13] Minsoo Cho, Seung T. Kim, Carlos Tokunaga, et al. "Post-silicon Voltage Guard-Band Reduction in a 22 nm Graphics Execution Core Using Adaptive Voltage Scaling and Dynamic Power Gating". In: *IEEE Journal of Solid-State Circuits* 52.1 (2017), pp. 50–63.

[14] Ivan Miro-Panades, Edith Beigné, Yvain Thonnart, et al. "A Fine-Grain Variation-Aware Dynamic V_{DD}-Hopping AVFS Architecture on a 32 nm GALS MPSoC". In: *IEEE Journal of Solid-State Circuits* 49.7 (2014), pp. 1475–1486.

[15] Yazhou Zu et al. "Adaptive Guardband Scheduling to Improve System-Level Efficiency of the POWER7+". In: *Proceedings of the 48th International Symposium on Microarchitecture (MICRO)*. 2015, pp. 308–321.

[16] Yuta Miyake, Yasuo Sato, Seiji Kajihara, et al. “Temperature and Voltage Estimation Using Ring-Oscillator-Based Monitor for Field Test”. In: *Proceedings of the 23rd IEEE Asian Test Symposium (ATS)*. 2014, pp. 133–138.

[17] ARM Limited. *AMBA 3 APB Protocol Specification*. IHI 0024B. ARM Limited. Cambridge, UK, 2004.

Printed by Tryckeriet i E-huset, Lund 2026

LUND
UNIVERSITY

Series of Master's theses
Department of Electrical and Information Technology
LU/LTH-EIT 2026-1125
http://www.eit.lth.se