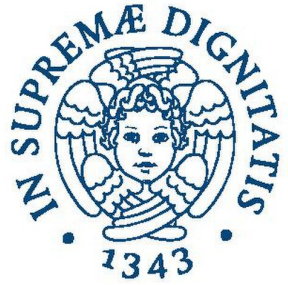

UNIVERSITÀ DI PISA

Department of Information Engineering

Master's Degree in Electronic Engineering

Digital Systems Design Project

Design of an All-Digital Phase-Locked Loop (ADPLL)

Professors:
Prof: Roberto Saletti
Prof: Pietro Nannipieri

Students:
Samuele De Carlo
Andrea Pellegrinotti

Contents

List of Figures

Chapter 1

Introduction

1.1 Description

A Phase-Locked Loop (PLL) is a fundamental building block in modern electronic systems, widely used for frequency synthesis, clock generation, synchronization, and signal recovery. Its operating principle is based on the continuous comparison between the phase and the frequency of a **reference signal** and that of a **generated signal**, allowing the system to adjust and lock the output frequency to the desired value. Since their introduction in analog form, PLLs have played a crucial role in communication systems, control electronics, and digital circuits.

With the progressive scaling of CMOS technologies and the increasing demand for flexibility and portability, the traditional analog PLL has gradually evolved into the **All-Digital Phase-Locked Loop** (ADPLL). In an ADPLL, all the main functional blocks are implemented in the **digital domain**, replacing analog components with digital signal processing techniques. This fully digital approach offers significant advantages, such as improved robustness to process, voltage, and temperature variations, as well as enhanced scalability and ease of integration in modern digital systems.

In recent years, ADPLLs have become particularly attractive for implementation on **Field-Programmable Gate Arrays** (FPGAs), where their digital nature allows efficient realization using hardware description languages. In this work, the ADPLL is entirely implemented on FPGA and described using the **Verilog** hardware description language.

A key focus of this project is the comparative analysis of two different ADPLL architectures. By evaluating their performance in terms of locking behavior, frequency stability, and implementation complexity, this study aims to highlight the strengths and limitations of each approach, providing insight into the design trade-offs involved in fully digital phase-locked loop systems.

1.2 Materials

For the development of the project we have used:

- the FPGA Altera DE2 Cyclone II
- an oscilloscope
- a function generator

For the reference input signal to the ADPLL, we used a **function generator** that produced a square wave, allowing us to vary the frequency and test the PLL's locking range. The output was acquired using the two **GPIO** banks of the board, GPIO0 and GPIO1. The reset was controlled via a switch on the board, and the internal logic was clocked by the 50MHz **clock** of the board itself.

Chapter 2

Fundamentals of Phase-Locked Loops

2.1 Overview and Working Principle

Before introducing our specific ADPLL architectures, it is essential to understand the basic operation of a Phase-Locked Loop (PLL). A PLL is a feedback control system that generates an output signal whose phase is locked to the phase of an input reference signal.

Figure 2.1 shows the basic block diagram of a conventional PLL. The main components are:

- **Reference signal (REF)**: the input signal to which the PLL locks its output.
- **Phase Detector (PD)**: compares the phase of the reference signal with the feedback signal (O.L) coming from the output of the VCO. It generates an error signal V_e proportional to the phase difference. It is worth noting that, in steady-state conditions, the output and reference phases are not necessarily aligned. The phase relationship at lock depends on the specific type of phase detector employed and on its characteristic. A detailed discussion of the phase detector types considered in this work is therefore postponed to the description of the proposed ADPLL architectures.
- **Loop Filter** $F(s)$: processes the error signal V_e to produce a control voltage V_d for the Voltage-Controlled Oscillator (VCO). The filter determines the dynamic response and stability of the PLL. It is typically designed as a low-pass filter, since its main purpose is to extract the low-frequency component of the phase error while attenuating high-frequency components, such as noise and spurious terms generated by the phase detector. In most practical implementations, it includes at least one pole at the origin to ensure zero steady-state phase error, and may incorporate additional poles and zeros to improve stability and transient performance.
- **Voltage-Controlled Oscillator (VCO)**: generates an output signal whose frequency is directly proportional to the control voltage V_d. The VCO output is fed back to the phase detector as the open-loop signal (O.L).

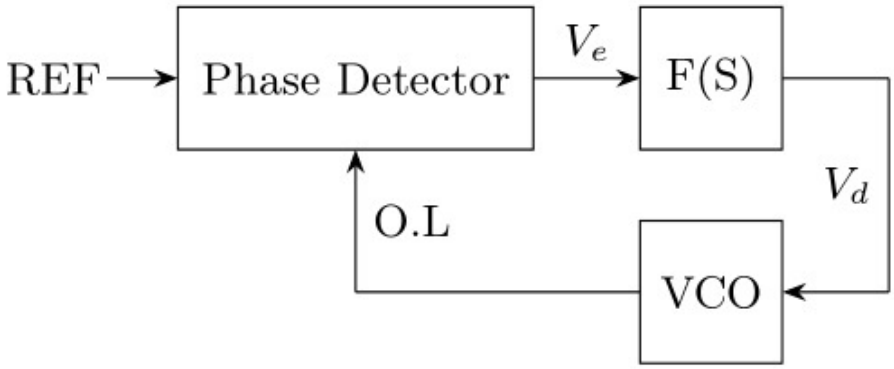

Figure 2.1: Basic block diagram of a Phase-Locked Loop.

2.1.1 Mathematical Description

The operation of a PLL can be described mathematically using the signals shown in the block diagram:

$$V_e(t) = K_d \left[\phi_{\text{REF}}(t) - \phi_{\text{O.L}}(t)\right] \tag{2.1}$$

where K_d is the gain of the phase detector, ϕ_{REF} is the phase of the reference signal, and $\phi_{\text{O.L}}$ is the phase of the VCO feedback signal. After passing through the loop filter $F(s)$, the control voltage $V_d(t)$ applied to the VCO is:

$$V_d(t) = V_e(t) \circledast F(t) \tag{2.2}$$

The VCO converts this control voltage into an output frequency:

$$\omega_{\text{out}}(t) = \omega_{\text{free}} + K_v V_d(t) \tag{2.3}$$

where K_v is the VCO gain (rad/s per volt) and ω_{free} is the free-running frequency of the VCO.

In the Laplace domain, the closed-loop transfer function of the PLL can be written as:

$$\frac{\Phi_{\text{out}}(s)}{\Phi_{\text{REF}}(s)} = \frac{K_d K_v F(s)}{s + K_d K_v F(s)} \tag{2.4}$$

This expression highlights that the PLL acts as a feedback system that forces the output phase Φ_{out} to follow the reference phase Φ_{REF}.
It can be demonstrated that the closed-loop transfer function derived for the phase domain is equally valid when considering the VCO output frequency and the reference frequency.

2.1.2 Key Properties and Applications

Some important characteristics of a PLL include:

- **Lock Range**: the range of reference frequencies over which the PLL can achieve phase lock.
- **Capture Range**: the frequency range over which the PLL can initially acquire lock starting from an unlocked condition.

- **Phase Noise and Stability**: determined primarily by the loop filter $F(s)$ and VCO characteristics.

These introductory concepts will later prove useful for understanding our ADPLL architecture.

2.1.3 From Analog PLLs to All-Digital PLLs

The PLL model introduced so far is based on a continuous-time, analog representation, where signals are described in terms of voltages, phases, and transfer functions in the Laplace domain. This formulation serves as a valuable reference framework even when moving toward digital implementations.

In an All-Digital Phase-Locked Loop, the same feedback principle is preserved, but all the main building blocks are implemented in the digital domain. As a result, signals are processed in discrete time and represented by digital quantities rather than continuous voltages. Consequently, the loop filter is replaced by a digital filter, the phase detector is implemented using digital logic, and the voltage-controlled oscillator is substituted by a digitally controlled oscillator (DCO).

In the following chapter, these principles are applied to the description and analysis of the proposed ADPLL architectures implemented on FPGA.

Chapter 3

First Architecture

In this section, the first of the two architectures considered for the implementation of an ADPLL is presented. First, the operation of the individual building blocks is described; subsequently, the selected values of the main design parameters are discussed. All blocks are designed with asynchronous reset capability. The proposed architecture is taken from the fifth edition of Phase-Locked Loops: Design, Simulation, and Applications, published by McGraw-Hill in 2003.

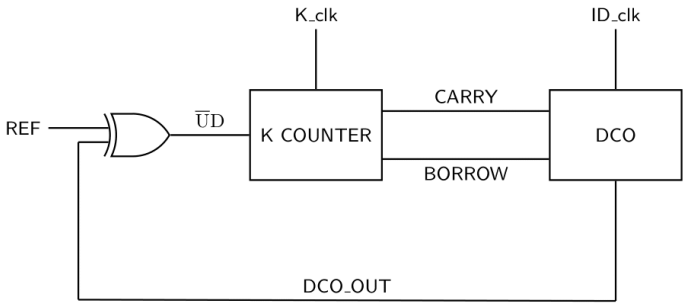

Figure 3.1: First architecture block diagram

3.1 Phase Detector

An *XOR* gate is employed as the phase detector. The input signals of the block are the reference signal and the signal generated by the DCO, which is frequency-divided by a factor N. Specifically, the input frequencies are f_{ref} and f_{DCO}, where f_{DCO} denotes the frequency of the DCO output signal.

The phase detector output signal exhibits a **variable duty cycle**, which depends on the phase difference between the two input signals. Once the PLL is locked, two operating conditions can be identified:

- $f_{\text{ref}} = f_0$: under this condition, the steady-state phase difference between the two signals is $\pi/2$, resulting in a 50% duty cycle at the XOR output.
- $f_{\text{ref}} \neq f_0$: in this case, the steady-state phase difference lies in the range between 0 and π.

As will be discussed in a subsequent section, the PLL may fail to achieve lock if the reference frequency deviates excessively from the nominal frequency f_0.

3.2 Digital Loop Filter

A **K-counter** is implemented as the digital loop filter, consisting of two independent counters, referred to as *up_ counter* and *down_ counter*. This block acts as a digital low-pass filter with integrative behavior, averaging the decisions of the phase detector over time.

3.2.1 Architecture and timing

The module inputs include the clock signal k_{clk}, operating at a frequency

$$f_k = M \cdot f_0,$$

where M denotes the oversampling factor, the reset signal, and the phase detector output u_d.

Rationale for Oversampling in K-counter Loop Filters

In the architecture of an All-Digital Phase-Locked Loop (ADPLL), the K-counter loop filter operates at a clock frequency significantly higher than the reference frequency, defined as $f_{\text{k}} = M \cdot f_0$, where M is the oversampling ratio. This design choice is fundamental for two primary reasons:

1. **Enhancement of Phase Resolution and Jitter Reduction:** Since the phase error is sampled in the digital domain, the temporal resolution of the system is limited by the system clock period. By employing oversampling ($M > 1$), the reference period is divided into M discrete intervals, effectively reducing the phase quantization step to $2\pi/M$ radians. This finer resolution mitigates limit-cycle oscillations and reduces intrinsic phase jitter, allowing the K-counter to integrate the phase error with higher precision before generating correction pulses (*carry* or *borrow*).

2. **Extension of Tracking and Hold Range:** The dynamic behavior of the ADPLL is tightly related to the system clock frequency. The maximum frequency deviation that the loop can sustain while maintaining lock, referred to as the *Hold Range* (Δf_H), is directly proportional to the oversampling factor M and can be expressed as:

$$\Delta f_H = \frac{f_0 \cdot M}{2KN} \tag{3.1}$$

   where K is the counter modulus and N is the feedback division ratio. Increasing M enhances the correction capability of the loop per unit time, allowing the ADPLL to remain locked even in the presence of significant frequency offsets between the reference signal and the digitally controlled oscillator.

3.2.2 Counter architecture

The filter is implemented by means of two independent counters that count from 0 to $K-1$ when their corresponding enable signal is asserted. The output of each counter is given by its **most significant bit** (MSB), which is used to generate the *carry* and *borrow* signals. The enable logic of the two counters is defined as follows:

- *up_counter*: enabled when $u_d = 0$;
- *down_counter*: enabled when $u_d = 1$.

```
module counter#(
    parameter k=4
)
(
    input in, clk, rst,
    output reg out
);
    integer count=k-1;
    always @(posedge clk or posedge rst)
        if (rst==1)
            out<=0;
        else if(in==0)
        begin
            if (count==((k/2)-1))
                begin
                    count<=count+1;
                    out<=1;
                end
            else if(count==(k-1))
                begin
                    count<=0;
                    out<=0;
                end
            else
                count <=count+1;
        end
endmodule
```

Figure 3.2: Counter verilog code

3.2.3 Operating conditions analysis

The behavior of the filter depends on the relationship between the reference frequency f_{ref} and the nominal oscillator frequency f_0:

- $f_{\text{ref}} = f_0$: the signals are in quadrature and the signal u_d exhibits a 50% duty cycle. Under this condition, *carry* and *borrow* pulses are balanced, maintaining the DCO frequency stable.
- $f_{\text{ref}} > f_0$: *carry* pulses dominate, causing an increase in the DCO frequency, in this case the signal u_d exhibits a duty cycle lower than 50%.
- $f_{\text{ref}} < f_0$: *borrow* pulses dominate, resulting in a reduction of the DCO frequency, in this case the signal u_d exhibits a duty cycle greater than 50%.

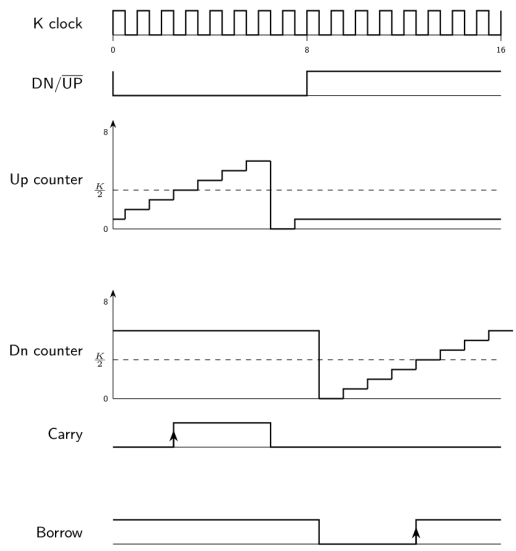

Figure 3.3: $f_{\text{ref}} = f_0$

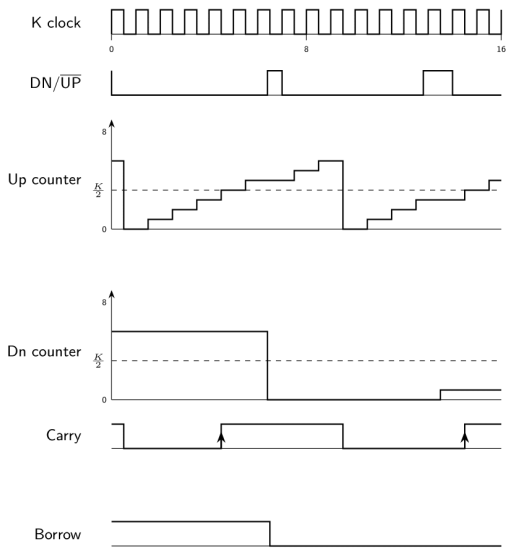

Figure 3.4: $f_{\text{ref}} > f_0$

3.3 Digitally Controlled Oscillator (DCO)

The Digitally Controlled Oscillator (DCO) is implemented using a module composed of an *ID_counter* and a frequency divider. The input signals of the DCO are the *carry* and *borrow* signals generated by the loop filter described in the previous section, an asynchronous reset signal, and the clock signal ID_{clk} with frequency

$$f_{ID} = 2Nf_0.$$

The output signal is then forwarded to the phase detector.

3.3.1 ID_Counter Architecture

The *ID_counter* is implemented using a *toggle flip-flop* controlled by the *carry* and *borrow* signals, followed by a NOR logic gate.

Synchronization and Edge Detection

Before being processed by the internal logic, the input signals undergo a fundamental conditioning stage to ensure system stability. This stage consists of two main blocks:

- **Synchronization (Metastability Reduction):** Since the *carry* and *borrow* signals are asynchronous with respect to ID_{clk}, there is a risk of setup and hold time violations. A **double flip-flop synchronization chain** is employed to confine any metastability to the first stage, ensuring that the second flip-flop delivers a stable and clock-synchronous signal to the FSM.
- **Edge Detection:** To prevent the FSM from interpreting a prolonged input pulse as multiple commands, a rising edge detector (*PosEdge Detector*) is implemented. By storing the previous signal state in an additional register, the logic generates an internal trigger only during the transition from logic '0' to logic '1'. This guarantees that each phase correction is applied exactly once for every pulse generated by the loop filter.

```
always @(posedge ID_clk or posedge rst) begin
        if (rst) begin
            {carry_f1, carry_f2, past_carry} <= 3'b0;
            {borrow_f1, borrow_f2, past_borrow} <= 3'b0;
        end
        else
        begin
            carry_f1    <= carry;
            carry_f2    <= carry_f1;
            past_carry  <= carry_f2;
            borrow_f1   <= borrow;
            borrow_f2   <= borrow_f1;
            past_borrow <= borrow_f2;
        end
    end

    wire carry_rise  = (carry_f2 == 1'b1) && (past_carry ==
        1'b0);
    wire borrow_rise = (borrow_f2 == 1'b1) && (past_borrow
        == 1'b0);
```

Figure 3.5: Synchronization and Edge Detection verilog code

Toggle Flip-Flop

The toggle flip-flop is realized through a finite state machine (FSM), whose state diagram is shown in Fig. 3.6.

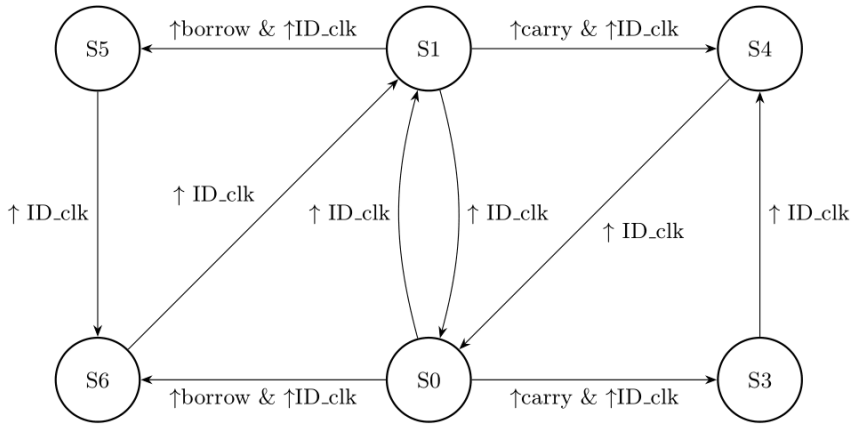

Figure 3.6: FSM state diagram

3.3.2 ID_Counter Architecture

The FSM behavior can be summarized as follows:

- If neither *carry* nor *borrow* signals are asserted, the FSM behaves as a standard toggle flip-flop, generating a square wave with frequency equal to half of ID_{clk}.
- If a *carry* signal is received, two cases are possible:
  1. If the output is low at the arrival of the *carry* signal, at the next rising edge of the clock the output goes high for one clock cycle and then remains low for the following two clock cycles.
  2. If the output is high at the arrival of the *carry* signal, at the next rising edge of the clock the output goes low and remains low for two clock cycles.
- If a *borrow* signal is received, two cases are possible:
  1. If the output is low at the arrival of the *borrow* signal, at the next rising edge of the clock the output goes high and remains high for two clock cycles.
  2. If the output is high at the arrival of the *borrow* signal, at the next rising edge of the clock the output goes low for one clock cycle and then remains high for the following two clock cycles.

Since, in the worst-case scenario, the processing of a *carry* or *borrow* pulse requires three clock cycles of ID_{clk}, the maximum allowable frequency of the *carry* and *borrow* signals must be limited to

$$\frac{f_{ID}}{3}$$

otherwise some pulses might not be correctly detected.

FSM Output Characteristics

As shown in the timing diagrams reported below,

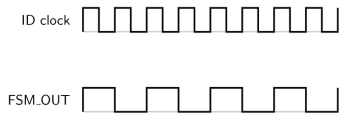

Figure 3.7: No carry or borrow

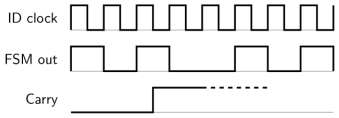

Figure 3.8: Carry received

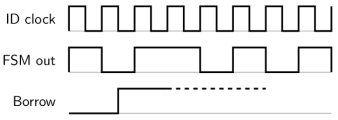

Figure 3.9: Borrow received

the FSM generates an output signal fsm_{out} whose duty cycle δ depends on the input conditions:

- $\delta = 50\%$ if neither *carry* nor *borrow* signals are present;
- $\delta = 33\%$ if a *carry* signal is received;
- $\delta = 66\%$ if a *borrow* signal is received.

The output of the FSM is then combined with the ID_{clk} signal to generate the output of the *ID_ counter* according to

$$ID_{\text{out}} = \overline{fsm_{\text{out}} + ID_{\text{clk}}}$$

This operation produces a pulse train whose number of pulses depends on the type of input signal received.

It is worth noting that when $f_{\text{ref}} = f_0$, the number of *carry* and *borrow* pulses is identical, and their effects cancel out. As a result, the average output frequency remains constant and equal to

$$f_{ID\_\text{out}} = Nf_0.$$

Conversely, when $f_{\text{ref}} \neq f_0$, the imbalance between *carry* and *borrow* pulses causes a variation in the average output frequency of the *ID_ counter*, which increases or decreases until frequency lock with the reference signal is achieved.

3.3.3 N divider

The output signal of the *ID_ counter* module is characterized by a time-varying instantaneous frequency, whose average value is equal to Nf_x. This variability originates from the phase corrections applied by the control loop, which advance or delay individual switching edges in order to minimize the phase error with respect to the reference.

After frequency division by a factor N, the resulting signal is a rectangular waveform at frequency f_x, which is fed back to the *Phase Detector* (PD).

The frequency divider performs a function that extends beyond simple frequency scaling, effectively acting as a **phase accumulator** with an inherent **jitter attenuation** capability:

- **Temporal correction averaging:** Phase adjustments introduced by the ID-counter through *carry* and *borrow* signals alter the timing of individual edges; however, the divider toggles its output only after counting a fixed number of events. As a result, instantaneous timing errors are averaged over a longer observation interval.
- ***Phase ripple* reduction:** High-frequency phase fluctuations present in the ID_{out} signal are significantly attenuated in the divided signal u_2'. The phase asymmetries introduced by control actions tend to cancel out over a complete period of the feedback signal.
- **Loop stability improvement:** Reducing the frequency from Nf_x to f_x allows the Phase Detector to operate on signals with compatible dynamics, thereby limiting the impact of high-frequency spurious components and promoting stable ADPLL lock behavior.

In conclusion, the frequency divider does not merely scale the ID-counter output frequency, but plays a critical role in generating a stable and regular feedback signal, which is essential to ensure system accuracy under locked conditions.

```
module N_div(
    input clk, rst,
    output reg clk_d
);
integer cnt=0;
    always@(posedge clk or posedge rst)
    begin
        if(rst==1)
            clk_d <= 1'b0;
        else
        begin
            if(cnt==3)begin
                clk_d <= ~clk_d;
                cnt = 0;
            end
            else
                cnt = cnt+1;
        end
    end
endmodule
```

Figure 3.10: N divider verilog code

3.4 Parameter Selection

This section analyzes how the choice of the **design parameters** affects the presence of *phase ripple* and the hold frequency of the proposed ADPLL.

3.4.1 Phase ripple cancellation

During the parameter selection phase, several design aspects must be considered, including the possible occurrence of **phase ripple** at the DCO output and the achievable system hold range.

To prevent ripple on the output signal, the *K-counter* parameter K is selected according to:

$$K = \frac{M}{4}$$

When the reference frequency equals the nominal design frequency ($f_{\text{ref}} = f_0$), the selected parameter set plays a crucial role in the loop behavior. In the considered design, setting $M = 16$ yields $K = 4$. With this configuration, the up and down counters reach their terminal count within each quarter period of the reference signal, generating *carry* and *borrow* events that are uniformly distributed over time. As a result, their effects on the DCO frequency cancel out over each reference period, leading to a constant average output frequency and ideally zero phase ripple.

In addition to the condition on K, proper ripple cancellation requires the increment/decrement (ID) counter to process all *carry* and *borrow* events without temporal overlap. Since the duration of one ID clock cycle is $1/(2Nf_0)$, the feedback division factor N must be sufficiently large to avoid missed events. This requirement imposes the following lower bound:

$$N > \frac{3M}{2K}$$

which guarantees correct carry/borrow handling and preserves the ripple cancellation mechanism.

When the reference frequency differs from the nominal value ($f_{\text{ref}} \neq f_0$), the symmetry between *carry* and *borrow* events is lost. Although the counters still operate according to the same timing constraints, one type of event statistically dominates the other, producing a net frequency correction. In this condition, phase ripple is no longer fully canceled and a residual ripple appears at the DCO output, whose amplitude depends on the magnitude of the frequency offset.

3.4.2 Hold range considerations

In addition to phase ripple suppression, the selection of M, K, and the feedback division factor N directly impacts the maximum frequency deviation that the ADPLL can track while remaining locked, commonly referred to as the **hold range**.

Assuming a worst-case operating condition in which the *K-counter* continuously generates *carry* events, the maximum carry frequency is given by:

$$f_{\text{carry,max}} = f_0 \frac{M}{K}$$

Each *carry* pulse applied to the increment/decrement (ID) counter introduces an effective half-period correction to the DCO output. As a result, the maximum achievable frequency increment at the ID output is:

$$\Delta f_{\text{IDout}} = f_0 \frac{M}{2K}$$

Considering the presence of a frequency divider with ratio N in the feedback loop, the maximum hold range of the ADPLL can therefore be expressed as:

$$\Delta f_H = f_0 \frac{M}{2KN}$$

This expression highlights the inherent trade-off between phase ripple suppression and frequency tracking capability. While increasing K improves phase ripple attenuation by averaging the phase detector output over longer intervals, it simultaneously reduces the achievable hold range and increases the minimum required value of N for correct ID counter operation. The selected values $M = 16$, $K = 4$, and $N = 8$ represent a balanced compromise between output phase stability, correct carry/borrow processing, and frequency acquisition capability.

Chapter 4

Second Architecture

4.1 Block Diagram

The second architecture, as shown in figure 4.1, is composed of three main blocks: a Phase-Frequency Detector (PFD), a digital Proportional-Integral (PI) filter, and a threshold-based / comparator-based DCO.

The **Phase Frequency Detector**, as shown in figure 4.3 is implemented using two D flip-flops with their D inputs permanently tied to logic '1'. The clock inputs of the two flip-flops are driven by the reference clock (ref) and by the output of the DCO (out), respectively. The outputs of the PFD are the up and $down$ signals, which encode the relative phase and frequency difference between the reference and the oscillator signals.

The PFD outputs are processed by a digital **PI Filter**, whose task is to generate a control word proportional to the accumulated phase error. This control word adjusts the behavior of the DCO.

The **Digital Controlled Oscillator** is implemented as a digital timing system based on a counter. The oscillator output toggles whenever the counter reaches a programmable threshold value. This threshold is dynamically increased or decreased by the PI filter output, effectively controlling the oscillation period and therefore the output frequency. This approach allows a fully digital realization of the oscillator while preserving fine frequency resolution.

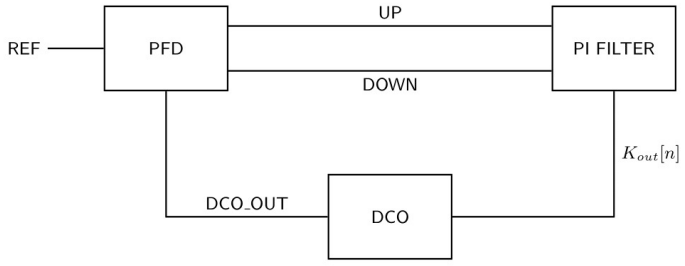

Figure 4.1: Second architecture block diagram

4.2 Asynchronous Reset with Synchronous Release

While an asynchronous reset ensures that the system enters a known state immediately regardless of the clock, its *release* (de-assertion) occurs independently of the system clock.

If the reset signal transitions from active to inactive too close to the rising edge of the clock, it can violate the **recovery time** (t_{rec}) or **removal time** (t_{rem}) of the flip-flops.

This violation can trigger **metastability**, where the internal nodes of the flip-flops fail to settle into a stable logical state within the required time, leading to unpredictable system behavior or functional failure. To mitigate this, a **reset synchronizer** circuit is implemented, as shown in figure 4.2. This configuration allows for an immediate global reset while ensuring that the release of the reset signal is synchronized with the clock domain. By forcing the reset de-assertion to occur on a valid clock edge, the circuit guarantees sufficient timing margins, effectively eliminating metastability risks and ensuring a deterministic start-up for all downstream logic.

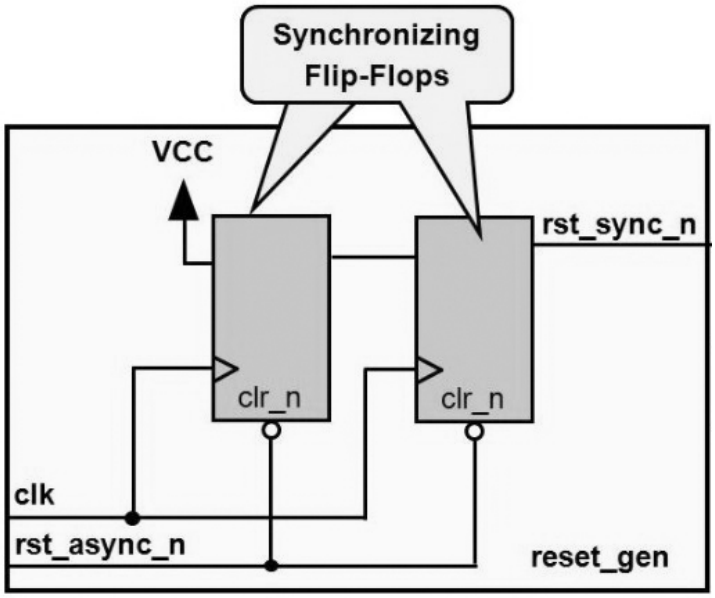

Figure 4.2: Asynchronous Reset with Synchronous De-assertion

4.3 Phase-Frequency Detector

The Phase-Frequency Detector is one of the most widely used phase detectors in PLL architectures due to its robustness and its capability to guarantee frequency acquisition over a wide operating range. Unlike simpler phase detectors such as XOR-based or JK flip-flop-based detectors, the PFD is **able to detect** both phase and frequency differences and does not suffer from false locking or dead zones. This characteristic ensures reliable lock acquisition under all operating conditions.

The phase detector exhibits a linear phase-to-output characteristic over a range from -2π to $+2\pi$, which is particularly advantageous for loop stability and linear analysis. This linear behavior allows the PFD to be modeled with a constant gain K_d over a wide phase error range.

The operation of the PFD can be described by the finite state machine shown in Fig. 4.4. The detector reacts to the rising edges of the reference signal (ref) and the DCO output signal (out), which act as the clock inputs of the two D flip-flops. When a rising edge occurs on the reference signal before the oscillator output, the *up* signal is asserted. Conversely, when the oscillator output edge occurs first, the *down* signal is asserted.

Thanks to this edge-triggered operation, the PFD remains fully functional even when

the reference and feedback signals have different **duty cycles**. This represents a significant advantage compared to level-sensitive phase detectors, whose operation can be degraded by duty cycle mismatches.

The logical implementation of the PFD includes a reset path generated by an AND gate that detects the condition $up = 1$ and $down = 1$. This condition does not represent a valid operational state and is therefore inhibited. When both outputs are simultaneously asserted, the reset signal is activated and asynchronously clears both flip-flops, forcing *up* and *down* back to zero.

Due to the finite propagation delay of the reset logic, very short pulses or spikes may appear on the *up* and *down* signals. These glitches are inherent to the PFD structure but do not affect the overall system performance.

Overall, the PFD structure ensures wide capture range, monotonic phase detection, immunity to duty cycle variations, and reliable lock behavior, making it particularly suitable for all-digital PLL architectures.

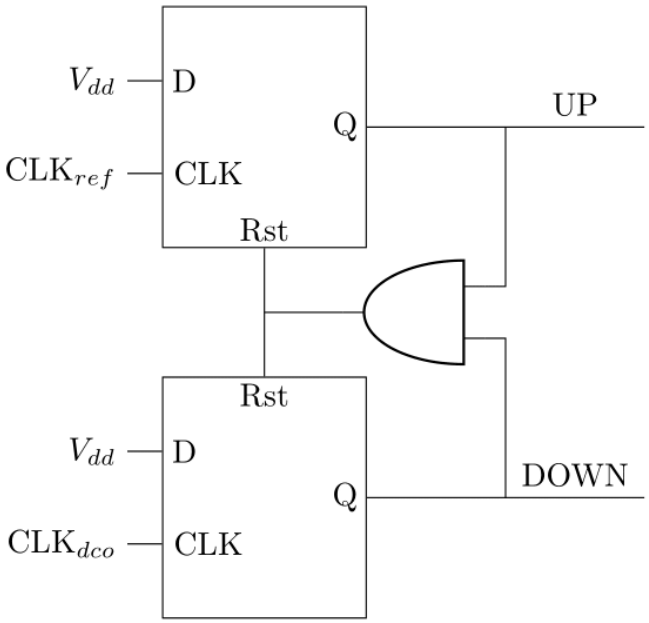

Figure 4.3: Logical schematic of the Phase-Frequency Detector

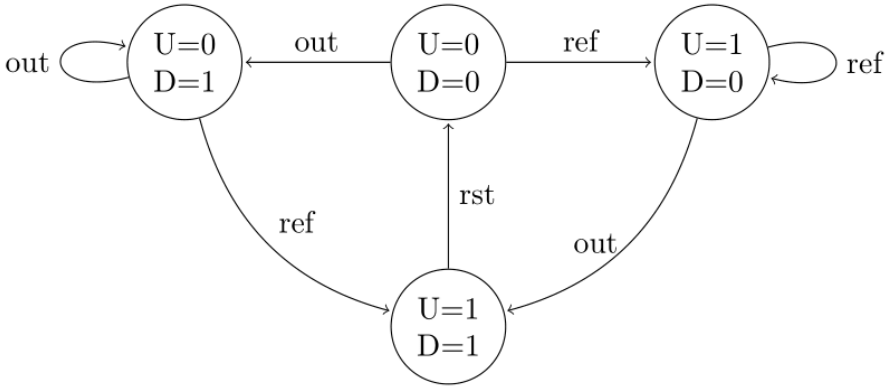

Figure 4.4: Finite State Machine (FSM) describing the operation of the PFD

4.4 Pulse Synchronization

The Phase-Frequency Detector generates asynchronous *up* and *down* pulses. The widths of these pulses are proportional to the **phase error** between the reference clock (f_{ref}) and the DCO clock (f_{dco}). Since the subsequent PI (Proportional-Integral) filter operates in a **synchronous clock domain** (f_{clk} = 50 MHz), direct sampling of these pulses is impossible due to the risk of **metastability** and the high probability of "missing" narrow pulses that occur between system clock edges.

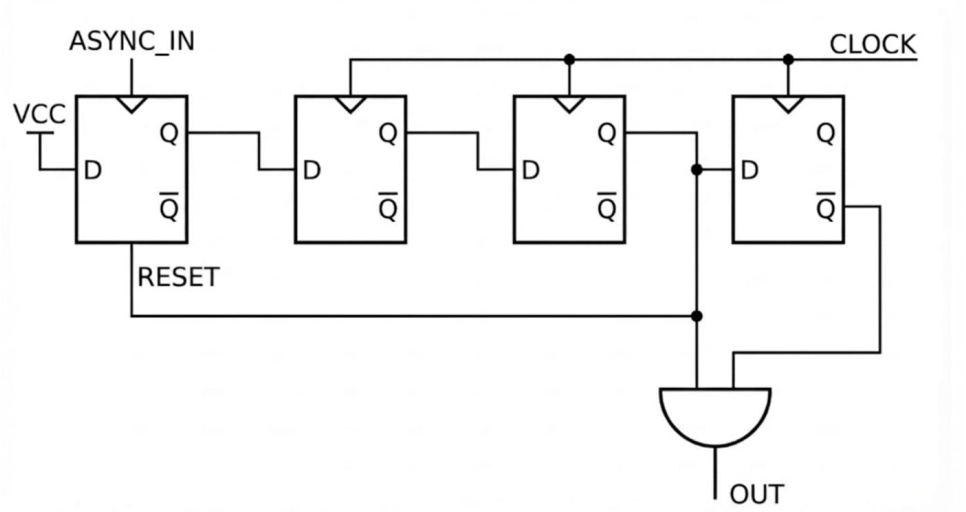

Figure 4.5: Pulse Synchronization

4.4.1 Architectural Implementation

The implemented interface, as shown in figure 4.5 utilizes a specialized Pulse Synchronizer architecture for each channel. The design consists of a **Request Flip-Flop** (acting as an asynchronous flag) followed by a multi-stage synchronization chain. The synchronization process follows these steps:

1. **Asynchronous Event Capture:** The `flag_async` register acts as an edge-triggered

latch. It is set to '1' by the rising edge of the `async_in` signal. This ensures that even an extremely narrow pulse (shorter than the 50 MHz clock period) is registered.

2. **Clock Domain Crossing (CDC):** The output of the flag is sampled by the 50 MHz clock domain through a two-stage synchronizer (`sync_q1`, `sync_q2`). This suppresses metastability and aligns the asynchronous event to the system timebase.

3. **Feedback Reset (Acknowledge):** Once the signal is successfully captured in the synchronous domain (`sync_q2` becomes high), the `flag_clear` signal is asserted. This resets the `flag_async` register, preparing the circuit for the next PFD event.

4. **Edge Detection:** A third register (`sync_q3`) is used to perform a rising-edge detection. The output `pulse_out` is high for exactly one clock cycle when:

$$pulse_{out} = sync\_q2 \cdot \overline{sync\_q3} \tag{4.1}$$

4.4.2 Technical Advantages and Necessity

This architecture is superior to simple double-registering for the following reasons:

- **Pulse Preservation:** A simple synchronizer requires the input pulse to be at least 1.5× to 2× longer than the system clock period to be detected.
- **Metastability Mitigation:** By using a dedicated synchronization chain, the Mean Time Between Failures (MTBF) is exponentially increased, ensuring the PI filter does not receive "glitched" or intermediate logic levels.
- **Uniform Weighting:** Every PFD event, regardless of its original duration, is converted into a calibrated 1-cycle pulse. In a digital PI filter, this allows the integrator to treat each pulse as a discrete "unit" of phase error, effectively digitizing the phase error magnitude through pulse density or frequency.

4.4.3 Design Constraints and Considerations

For correct operation, it is important to note that this synchronizer has a **dead time**. The PFD cannot fire a second pulse until the first has been cleared by the `flag_clear` signal. Given $f_{clk} = 50$ MHz ($T_{clk} = 20$ ns), the maximum frequency of the PFD pulses must satisfy:

$$f_{PFD,max} < \frac{1}{4 \cdot T_{clk}} = 12.5 \text{ MHz} \tag{4.2}$$

In most ADPLL applications where the reference clock is lower than the system clock, this condition is naturally satisfied.

In summary, the implemented interface acts as a robust bridge between two fundamentally different physical domains.

4.5 Digital Loop Filter

A Proportional-Integral (PI) filter is a key component in control systems and digital phase-locked loops, providing both immediate correction of errors and long-term stability. Its function is to generate a control signal that depends on the current error as well as the accumulated past errors.

Continuous-time formulation The continuous-time PI filter can be expressed as:

$$H(s) = K_P + \frac{K_I}{s} \tag{4.3}$$

where K_P is the proportional gain, providing an immediate response to the error, and K_I is the integral gain, accumulating the error over time. The integral term ensures zero steady-state error for a constant input.

Discrete-time formulation In digital systems, the filter is implemented in discrete time using the z-transform. Assuming a sampling period T_s, the discrete-time transfer function is:

$$H(z) = K_P + K_I \frac{T_s}{1 - z^{-1}} \tag{4.4}$$

which corresponds to the difference equation:

$$y[n] = y[n-1] + q_0 e[n] + q_1 e[n-1] \tag{4.5}$$

where $e[n]$ is the error at the n-th sample, $y[n]$ is the output of the filter and the coefficients q_0 and q_1 represent the tuning parameters of the discrete-time PI controller in its recursive form. They encapsulate the proportional gain K_P, the integral gain K_I, and the sampling period T_s.

Frequency response The Bode diagram of a PI filter typically shows:

- At low frequencies, the magnitude decreases at +20 dB/decade due to the integral term, ensuring elimination of steady-state error.
- At high frequencies, the proportional term dominates, providing fast response without excessive overshoot.

This combination ensures both stability and responsiveness of the control loop.

Implementation in the current design In this project, the PI filter is implemented digitally in Verilog as the loop filter for the ADPLL. The filter processes the *up* and *down* pulses from the PFD and generates the output control signal K_{out} for the Digitally Controlled Oscillator (DCO).

Key features of the implementation include:

- **Proportional term:** generates short pulses corresponding to the immediate phase error. Mathematically, it can be expressed as:

$$P[n] = K_P \cdot \text{pulse}[n] \tag{4.6}$$

- **Integral term:** accumulates error over time, with saturation to prevent overflow:

$$I[n] = \min(I_{\max},\, I[n-1] + K_I \cdot \text{pulse}[n]) \tag{4.7}$$

- **Lock mode:** the gains K_P and K_I can be reduced when the loop is locked, improving stability and avoiding overshoot.

- **Pulse scaling:** the filter accumulates multiple pulses before applying the proportional correction, allowing fine resolution of the control output.

The resulting control signal is:

$$K_{\text{out}}[n] = P[n] + I[n] \tag{4.8}$$

Role of K_{out} The output of the PI filter, K_{out}, serves as the control signal for the DCO. Specifically, it is used to adjust the threshold of a comparator within the DCO, effectively increasing or decreasing the oscillation frequency according to the loop's error.

In this way, K_{out} directly influences the DCO to correct phase and frequency deviations detected by the PFD. A detailed description of the DCO architecture and how K_{out} interacts with it will be presented in a subsequent section.

4.5.1 PI Filter Instance and Parameterization

The architecture, as shown in Fig. 4.6, allows for fine-tuning through several key parameters:

- **Gains (`P_GAIN`, `I_GAIN`):** These parameters define the magnitude of the proportional and integral pulses applied to the control word, directly influencing the loop's tracking speed and stability.
- **Thresholds (`p_threshold`, `i_threshold`):** These values define the sensitivity of the filter. Internal counters must reach these specific thresholds before the respective proportional or integral correction is actually applied to the output. This mechanism acts as a digital prescaler, filtering out high-frequency noise and preventing excessive jitter in the K_{out} signal.

```
module PI_filter #(
    parameter P_GAIN     = 1,
    parameter I_GAIN     = 1,
    parameter p_threshold = 5,
    parameter i_threshold = 5
)(
    input rst,
    input clk,
    input up,
    input down,
    output reg signed [12:0] K_out
);
```

Figure 4.6: PI filter verilog code

4.6 Digitally Controlled Oscillator (DCO)

The Digitally Controlled Oscillator in this project is implemented as a counter-based threshold oscillator. Its output frequency is controlled by a digital signal, K_{out}, generated by the PI filter described in the previous section. The DCO operates by counting clock cycles from a reference clock and toggling its output when a programmable threshold is reached.

DCO operation principle Let the reference clock frequency of the system be f_{clk}. The DCO uses a counter that increments on each clock cycle. When the counter reaches a threshold value, the DCO output toggles and the counter resets. Mathematically, if the effective threshold is T_{eff}, the DCO period T_{DCO} is:

$$T_{\text{DCO}} = 2 \cdot T_{\text{clk}} \cdot T_{\text{eff}} \tag{4.9}$$

where $T_{\text{clk}} = 1/f_{\text{clk}}$ is the period of the reference clock, and the factor of 2 accounts for the toggle behavior (two threshold crossings per full output period).

The effective threshold is calculated from the base threshold and the control signal:

$$T_{\text{eff}} = \text{clamp}\big(T_{\text{base}} - K_{\text{out}},\, T_{\text{min}},\, T_{\text{max}}\big) \tag{4.10}$$

where clamp$(\cdot)$ limits the threshold within a minimum and maximum range to prevent overflow or underflow.

DCO frequency as a function of K_{out} From the period equation, the DCO output frequency is:

$$f_{\text{DCO}} = \frac{1}{T_{\text{DCO}}} = \frac{f_{\text{clk}}}{2 \cdot T_{\text{eff}}} = \frac{f_{\text{clk}}}{2 \cdot \text{clamp}(T_{\text{base}} - K_{\text{out}}, T_{\text{min}}, T_{\text{max}})} \tag{4.11}$$

- **At rest ($K_{\text{out}} = 0$):** the DCO frequency is:

$$f_{\text{rest}} = \frac{f_{\text{clk}}}{2 \cdot T_{\text{base}}} \tag{4.12}$$

- **For a generic K_{out}:** the frequency increases as K_{out} grows, since a larger K_{out} reduces T_{eff}:

$$f_{\text{DCO}}(K_{\text{out}}) = \frac{f_{\text{clk}}}{2 \cdot (T_{\text{base}} - K_{\text{out}})} \quad \text{subject to } T_{\text{min}} \leq T_{\text{base}} - K_{\text{out}} \leq T_{\text{max}} \tag{4.13}$$

Frequency resolution and jitter The smallest change in output frequency occurs when K_{out} changes by one unit. Denoting this step as ΔK, the resulting frequency step is:

$$\Delta f_{\text{DCO}} \approx \frac{f_{\text{clk}}}{2} \left(\frac{1}{T_{\text{base}} - K_{\text{out}} - \Delta K} - \frac{1}{T_{\text{base}} - K_{\text{out}}} \right) \tag{4.14}$$

This step size determines the **phase jitter** of the DCO when compared to a reference signal, since any quantization in K_{out} translates directly into discrete frequency increments.

Numerical example (formulas only) Given a system clock $f_{\text{clk}} = 50$ MHz, base threshold $T_{\text{base}} = 2500$, and allowed threshold range $T_{\text{min}} = 10$, $T_{\text{max}} = 4990$, the DCO frequency at rest is:

$$f_{\text{rest}} = \frac{50 \text{ MHz}}{2 \cdot 2500} = 10 \text{ kHz} \tag{4.15}$$

For a non-zero K_{out}:

$$f_{\text{DCO}}(K_{\text{out}}) = \frac{50\ \text{MHz}}{2 \cdot (2500 - K_{\text{out}})} \tag{4.16}$$

The frequency step, corresponding to a change of one unit in K_{out}, is:

$$\Delta f_{\text{DCO}} = \frac{50\ \text{MHz}}{2} \left(\frac{1}{2499} - \frac{1}{2500} \right) \approx 4\ \text{Hz} \tag{4.17}$$

which shows the quantization limit of the DCO and its contribution to jitter.

Operating range considerations In practice, the DCO performs well in a frequency range where K_{out} can adjust the threshold without reaching $T_{\min}$ or $T_{\max}$.

$$f_{\text{DCO}} \in [f_{\text{clk}}/(2 \cdot T_{\max}),\ f_{\text{clk}}/(2 \cdot T_{\min})] \tag{4.18}$$

However, at higher frequencies, the discrete threshold steps become a significant fraction of the period, causing poorer phase tracking and increased jitter.

4.6.1 Frequency Lock Accuracy and DCO Operating Range

The frequency resolution of the digitally controlled oscillator (DCO) directly determines the achievable lock accuracy of the ADPLL. In this work, the **relative frequency lock error** is defined as the ratio between the smallest achievable frequency step of the DCO, Δf_{DCO}, and the target oscillation frequency f_{DCO}:

$$\varepsilon_f \triangleq \frac{\Delta f_{\text{DCO}}}{f_{\text{DCO}}} \tag{4.19}$$

The minimum frequency step occurs when the output control word K_{out} changes by one unit. The corresponding expression is:

$$\Delta f_{\text{DCO}} \approx f_{\text{clk}} \left(\frac{1}{2\,(T_{\text{eff}} - 1)} - \frac{1}{2T_{\text{eff}}} \right) \tag{4.20}$$

For sufficiently large values of the effective threshold T_{eff}, the relative frequency error can be approximated as:

$$\varepsilon_f \approx \frac{1}{T_{\text{eff}}} \tag{4.21}$$

Imposing a maximum admissible relative frequency error $\varepsilon_f^{\max}$ leads to the following condition on the effective threshold:

$$T_{\text{eff}} \geq \frac{1}{\varepsilon_f^{\max}} \tag{4.22}$$

As a consequence, the maximum DCO frequency at which frequency lock can be maintained within the desired accuracy is:

$$f_{\text{DCO,max}} = \frac{f_{\text{clk}}}{2}\, \varepsilon_f^{\max} \tag{4.23}$$

Conversely, for lower oscillation frequencies the relative frequency error further decreases, and the minimum achievable frequency is limited by the maximum representable threshold of the counter:

$$f_{\mathrm{DCO,min}} = \frac{f_{\mathrm{clk}}}{2T_{\max}}, \qquad T_{\max} = 2^N - 1 \tag{4.24}$$

It is worth noting that increasing the system clock frequency f_{clk} allows the same relative frequency accuracy to be preserved at higher output frequencies. Indeed, Δf_{DCO} scales linearly with f_{clk} (see Eq. 4.20), thereby extending the DCO lock range toward higher frequencies without degrading the relative frequency resolution.

To better highlight the trade-off between frequency lock accuracy and the achievable DCO operating range, Table 4.1 reports the **maximum lockable** DCO frequency as a function of the admissible relative frequency error. For each target accuracy, the corresponding minimum effective threshold T_{eff} is derived from Eq. 4.22, and the resulting maximum output frequency is computed according to Eq. 4.23, assuming a system clock frequency of $f_{\mathrm{clk}} = 50\,\mathrm{MHz}$. This comparison provides a clear quantitative insight into how relaxing the frequency accuracy requirement allows the ADPLL to operate over a wider frequency range.

Table 4.1: Maximum DCO lock frequency as a function of the admissible relative frequency error, assuming $f_{\mathrm{clk}} = 50\,\mathrm{MHz}$.

Relative frequency error $\varepsilon_f^{\max}$	Minimum T_{eff}	$f_{\mathrm{DCO,max}}$
0.01%	10 000	2.5 kHz
0.1%	1 000	25 kHz
1%	100	250 kHz
5%	20	1.25 MHz

4.7 Summary and Design Considerations

This chapter presented the second ADPLL architecture, detailing the structure and operation of each functional block and highlighting the design choices adopted to achieve a fully digital and robust frequency locking system. The use of a PFD ensures wide capture range and reliable frequency acquisition, while the introduction of an asynchronous digital spike filter effectively suppresses spurious pulses, improving loop stability near lock conditions.

The digital PI filter provides a flexible control mechanism, combining fast transient response through the proportional term with zero steady-state error ensured by the integral action. Its digital implementation allows programmable gains, saturation control, and lock-dependent behavior, enabling a trade-off between acquisition speed and steady-state stability.

The counter-based DCO represents a simple yet effective digitally controlled oscillator implementation, offering fine frequency resolution and straightforward integration with the digital loop filter. The analysis of frequency resolution and relative lock accuracy shows that the achievable operating range is fundamentally limited by the **quantization** of the DCO threshold. In particular, a clear trade-off emerges between frequency accuracy and maximum lockable output frequency, as quantified by the analytical expressions and summarized in Table 4.1.

4.8 ModelSim Functional Verification

4.8.1 First architecture

The functional verification of the proposed ADPLL architectures was performed using *ModelSim Altera* through *gate-level* simulations. The purpose of these simulations was to demonstrate the ability of the system to achieve frequency lock and to show that, in steady-state conditions, the phase offset between the reference signal and the feedback signal depends on the relationship between the reference frequency f_{ref} and the nominal design frequency $f_0 = 781.25$ kHz.

Case $f_{\text{ref}} = f_0$

When the reference frequency matches the nominal design frequency, the system achieves perfect frequency lock. Under these conditions, the steady-state phase difference between the signals is equal to $\pi/2$, as expected for an XOR-based phase detector.

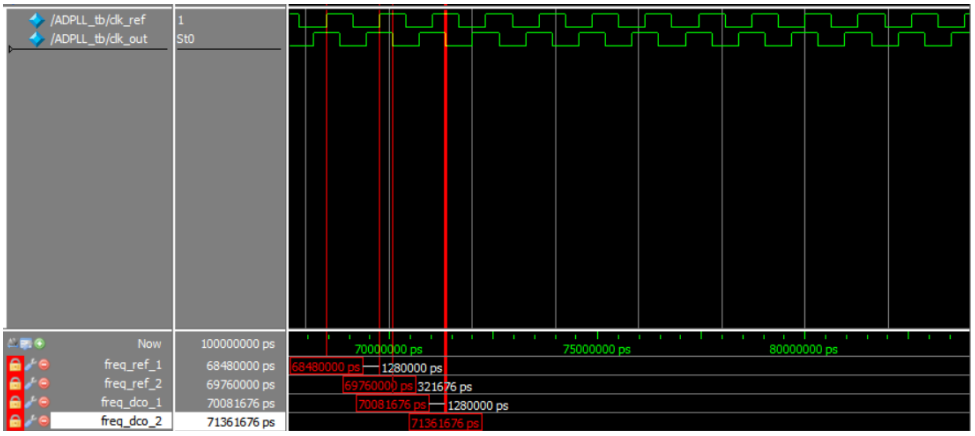

Figure 4.7: $f_{\text{ref}} = f_0$

Case $f_{\text{ref}} > f_0$

When the reference frequency is higher than the nominal frequency, the system settles with a phase difference greater than $\pi/2$. In the case shown in the figure 4.8, the reference frequency is $f_1 = 905$ kHz and the measured mean frequency is $f_2 = 914.63$ kHz, obtained by averaging over six periods. The corresponding relative error is $E = 1.064\%$.

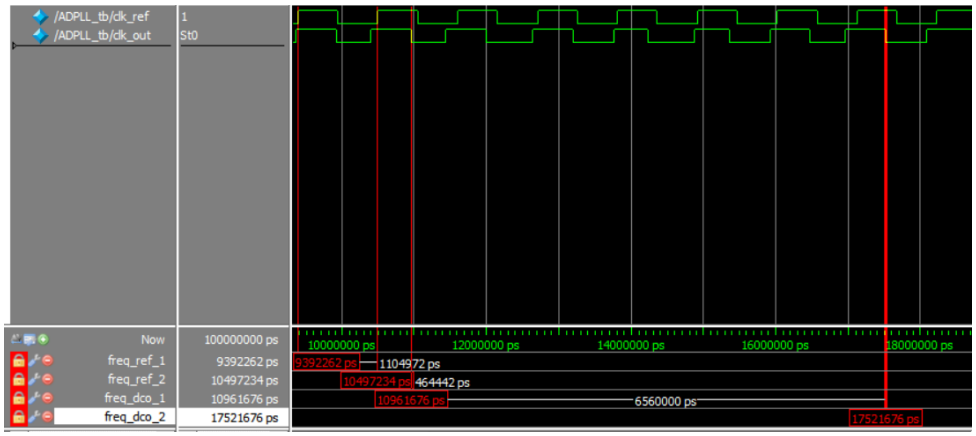

Figure 4.8: $f_{\text{ref}} > f_0$

For the same operating condition, an experimental laboratory test was also carried out using a *DE2* development board. The measured output signal exhibits a mean frequency of $f_{meas} = 909.09$ kHz, corresponding to a relative percentage error of $E = 0.45\%$. The comparison between *gate-level* simulations and experimental results confirms a good agreement between the theoretical model and the physical implementation of the ADPLL.

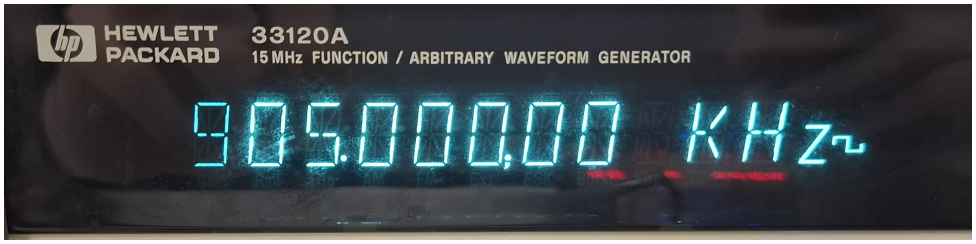

Figure 4.9: Value of f_{ref}

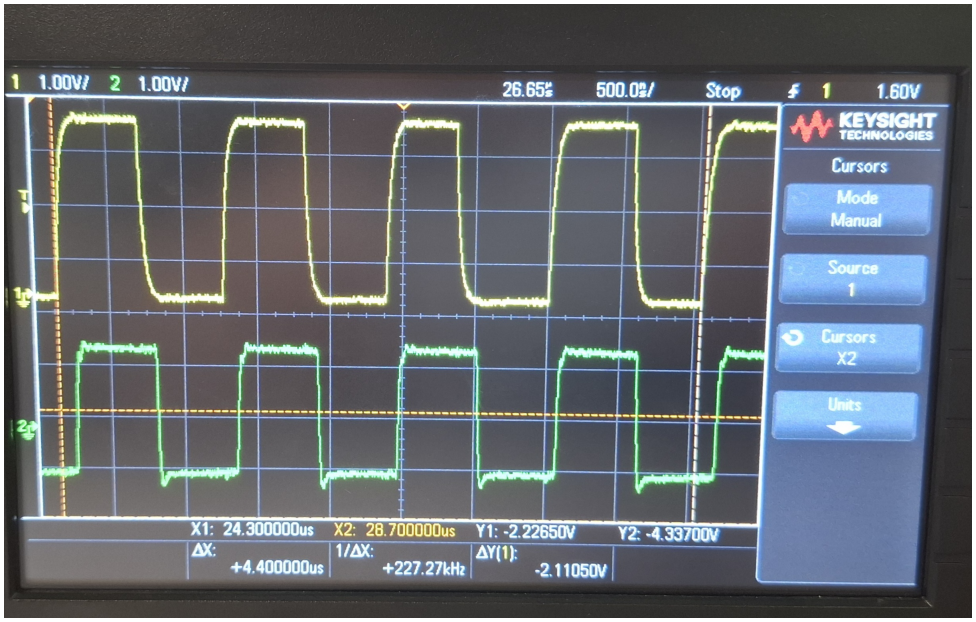

Figure 4.10: Measured value of f_{DCO}

Case $f_{\mathbf{ref}} < f_0$

When the reference frequency is lower than the nominal frequency, the system settles with a steady-state phase difference smaller than $\pi/2$. In the case shown in the figure, the measured mean frequencies are $f_1 = 714.286$ kHz and $f_2 = 710.227$ kHz, where f_2 is computed by averaging over ten periods. The corresponding relative percentage error is $E = 0.568\%$.

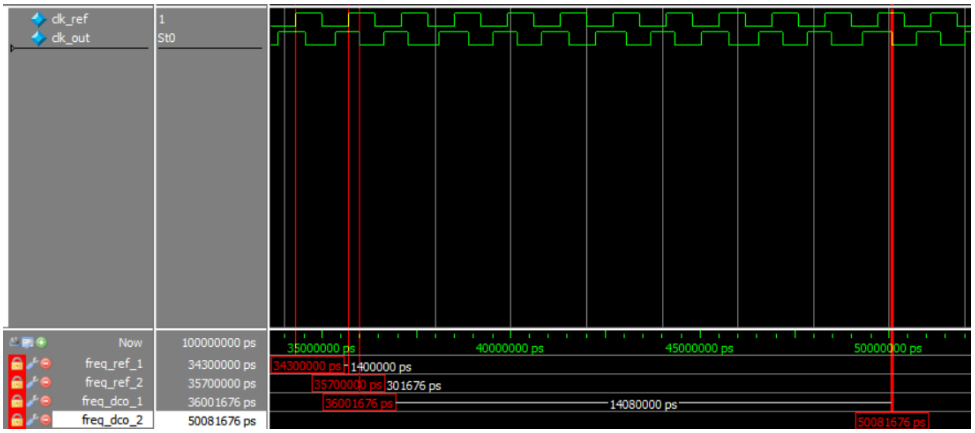

Figure 4.11: $f_{\text{ref}} < f_0$

4.8.2 Second architecture

The functional behavior of the ADPLL was verified through ModelSim simulations, focusing on the transition from the frequency acquisition phase to the steady-state lock. The system operates with a frequency system clock ($f_{clk} = 50$ MHz) and a reference signal (f_{ref}) with a period of 60 μs, which corresponds to:

$$f_{ref} = \frac{1}{60 \times 10^{-6}\ \text{s}} \approx 16.67\ \text{kHz}$$

4.8.3 Frequency Acquisition (Out-of-Lock)

As shown in the simulation traces (Figure 4.12), the system initially starts in an "out-of-lock" condition. In this phase, the Digitally Controlled Oscillator (DCO) has not yet matched the reference frequency.

- **Control Word (K_{out}):** The loop filter output K_{out} shows significant variation, representing the filter's effort to drive the DCO toward the target frequency.
- **Signal Alignment:** The `clk_dco` and `clk_ref` edges are not yet aligned, indicating a drifting phase relationship.

4.8.4 Steady-State Tracking (Locked)

Figure 4.13 illustrates the ADPLL once it has successfully "hooked" the reference signal.

- **Filter Command Logic:** The signal `K_out` represents the control word sent from the digital loop filter to the DCO. This value effectively dictates the DCO operation, instructing it to increase or decrease its frequency to maintain alignment with the reference.

- **Lock Stability and LSB Oscillation:** Once in steady-state, K_{out} stabilizes around a mean value, showing slight oscillations of a few LSBs (Least Significant Bits), for instance between 999 and 1002.
- **Influence of Filter Parameters:** This residual oscillation at steady-state is expected in digital loops; its magnitude and dynamics are strictly dependent on the specific filter parameters (such as proportional and integral gains).
- **Pulse Balancing:** The `up_sync_pulse` and `down_sync_pulse` signals become periodic and sparse, occurring only to compensate for minimal phase drifts.
- **Frequency Matching:** The DCO frequency now matches 16.67 kHz, perfectly synchronized with the external reference.

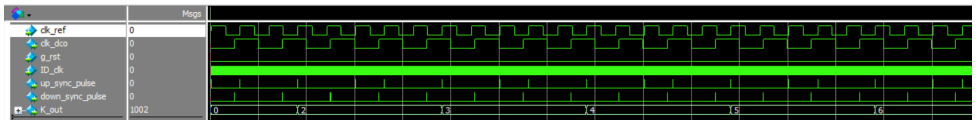

Figure 4.12: Out-of-Lock State

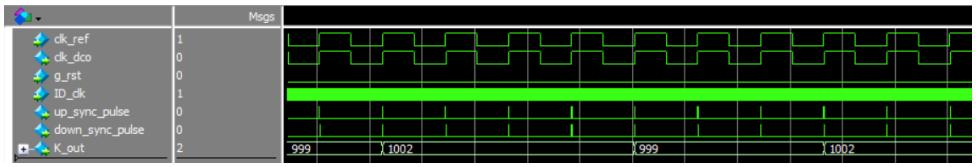

Figure 4.13: Locked State

The experimental validation of the ADPLL was performed using a benchtop function generator to provide the reference signal. Figures 4.14 and 4.15 illustrate the oscilloscope captures for two different reference frequencies. In both captures, Channel 1 represents the reference signal (f_{ref}), while Channel 2 displays the DCO output (f_{dco}). At a lower frequency of 17.99 kHz, the system achieves a near-perfect lock. However, as the frequency increases, a measurable steady-state error emerges.

In the second test, the oscilloscope shows $f_{ref} = 49.14$ kHz and a DCO output $f_{dco} = 49.02$ kHz. The relative frequency error, ϵ_{exp}, is defined as:

$$\epsilon_{exp} = \frac{|f_{ref} - f_{dco}|}{f_{ref}} \times 100 = \frac{|49.14 - 49.02|}{49.14} \times 100 \approx 0.244\% \tag{4.25}$$

To evaluate if this result aligns with the system's design specifications, a comparison with the data previously reported in Table 4.1 is necessary. The table provides two boundary points for the locking error: an error of 0.1% at 25 kHz and 1% at 250 kHz. Assuming a linear relationship between the frequency and the locking error, the theoretical error $\epsilon_{theo}(f)$ can be modeled using the point-slope form:

$$\epsilon_{theo}(f) = \epsilon_1 + \frac{\epsilon_2 - \epsilon_1}{f_2 - f_1} \cdot (f - f_1) \tag{4.26}$$

Substituting the values from Table 4.1 ($f_1 = 25, \epsilon_1 = 0.1$ and $f_2 = 250, \epsilon_2 = 1$), with f expressed in kHz and ϵ as a percentage:

$$\epsilon_{theo}(f) = 0.1 + \frac{1 - 0.1}{250 - 25} \cdot (f - 25) = 0.1 + 0.004 \cdot (f - 25) \tag{4.27}$$

By applying this linear approximation to the experimental reference frequency of 49.14 kHz, we obtain the expected theoretical error:

$$\epsilon_{theo}(49.14) = 0.1 + 0.004 \cdot (49.14 - 25) = 0.1 + 0.004 \cdot (24.14) \approx 0.197\% \tag{4.28}$$

Comparing the results, the experimental error ($\epsilon_{exp} \approx 0.244\%$) and the theoretical prediction ($\epsilon_{theo} \approx 0.197\%$) are quite close. These slight deviations depend on several factors, such as the oscilloscope's measurement resolution and the inherent nature of the linear approximation. Nevertheless, the results confirm that the theoretical trend is experimentally verified.

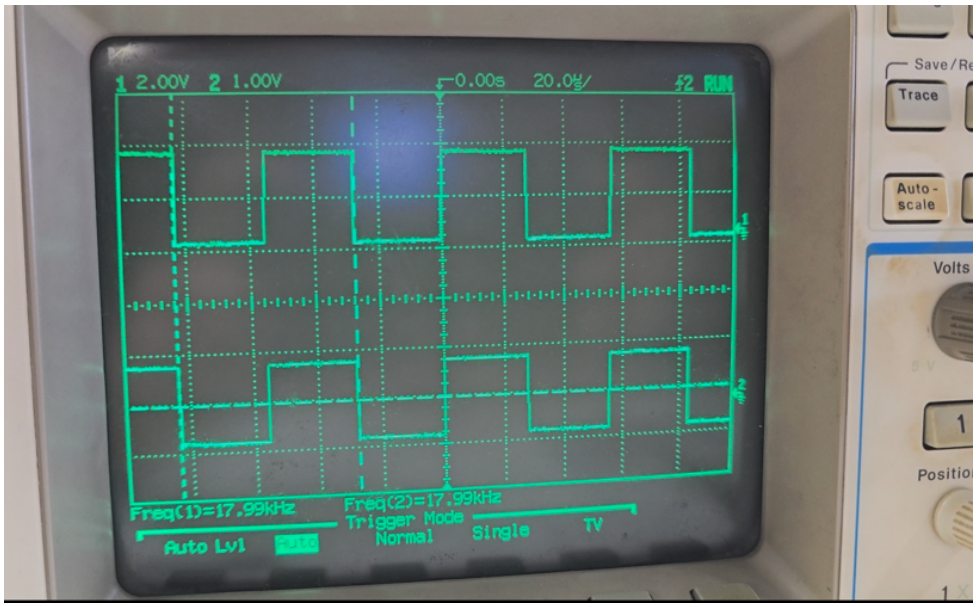

Figure 4.14: Test 1, frequency lock verification

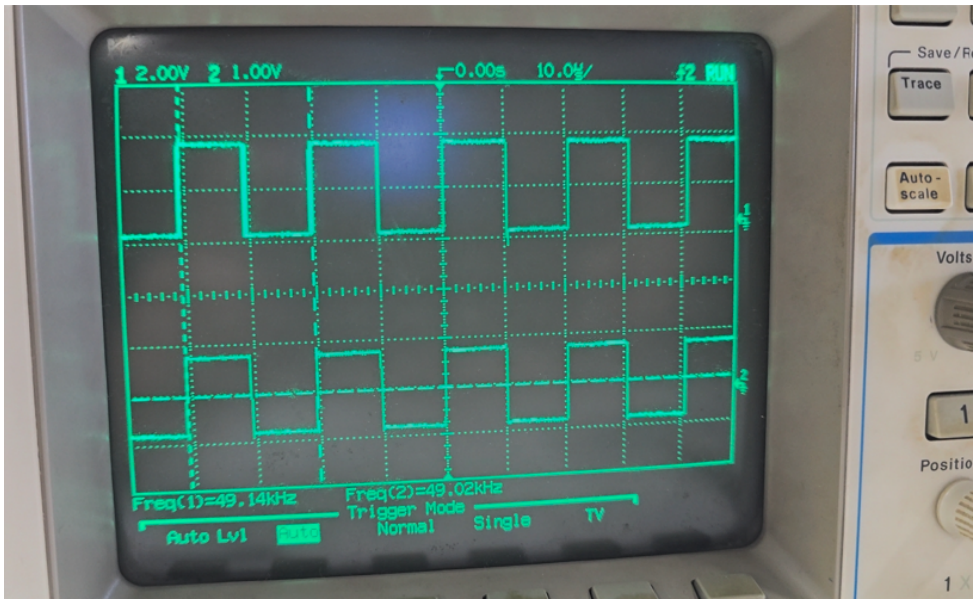

Figure 4.15: Test 2, frequency lock verification

4.9 Quartus Implementation Results

In this section, the results obtained from the implementation of the proposed architectures using the *Quartus* software are presented. Specifically, the maximum achievable operating frequency, the clock signals considered in the timing analysis, the hardware resource utilization, and the main synthesis warnings reported by Quartus are discussed.

4.9.1 First Architecture

	Clock Name	Type	Period	Frequency	Rise	Fall
1	INTERNAL_CLOCK	Base	20.000	50.0 MHz	0.000	10.000
2	K_ID_CLK	Generated	80.000	12.5 MHz	0.000	40.000

Figure 4.16: Timing Clock

	Fmax	Restricted Fmax	Clock Name
1	176.62 MHz	176.62 MHz	K_ID_CLK
2	239.64 MHz	239.64 MHz	INTERNAL_CLOCK

Figure 4.17: Maximum frequency

Since, in the proposed architecture, the $K_{\mathrm{ID\_CLK}}$ signal is derived by dividing the internal clock CLK_{INT} by a factor of 4, the maximum achievable operating frequency of $K_{\mathrm{ID\_CLK}}$ is equal to 59.91 MHz. It should be noted that this value represents a theoretical upper bound, which can only be achieved when the internal clock CLK_{INT} operates at its maximum allowable frequency. Consequently, this condition cannot be met when using

the on-board oscillator of the DE2 development board and would require an external high-performance clock source.

Family	Cyclone II
Device	EP2C35F672C6
Timing Models	Final
Total logic elements	197 / 33,216 (< 1 %)
Total combinational functions	193 / 33,216 (< 1 %)
Dedicated logic registers	145 / 33,216 (< 1 %)
Total registers	145
Total pins	4 / 475 (< 1 %)
Total virtual pins	0
Total memory bits	0 / 483,840 (0 %)
Embedded Multiplier 9-bit elements	0 / 70 (0 %)
Total PLLs	0 / 4 (0 %)

Figure 4.18: Fitter

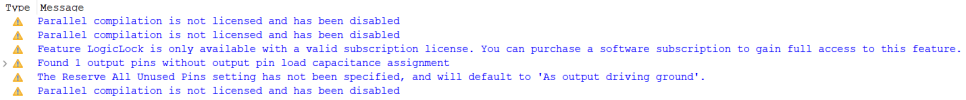

Figure 4.19: Warning

4.9.2 Second Architecture

	Clock Name	Type	Period	Frequency	Rise	Fall
1	CLK_OUT_DCO	Generated	100000.000	0.01 MHz	0.000	50000.000
2	CLK_PFD_DOWN	Generated	100000.000	0.01 MHz	0.000	50000.000
3	CLK_PFD_UP	Generated	20000.000	0.05 MHz	0.000	10000.000
4	CLK_REF	Base	20000.000	0.05 MHz	0.000	10000.000
5	ID_CLK	Base	20.000	50.0 MHz	0.000	10.000

Figure 4.20: Timing Clock

	Fmax	Restricted Fmax	Clock Name
1	179.37 MHz	179.37 MHz	ID_CLK

Figure 4.21: Maximum frequency

Family	Cyclone II
Device	EP2C35F672C6
Timing Models	Final
Total logic elements	147 / 33,216 (< 1 %)
Total combinational functions	145 / 33,216 (< 1 %)
Dedicated logic registers	71 / 33,216 (< 1 %)
Total registers	71
Total pins	4 / 475 (< 1 %)
Total virtual pins	0
Total memory bits	0 / 483,840 (0 %)
Embedded Multiplier 9-bit elements	0 / 70 (0 %)
Total PLLs	0 / 4 (0 %)

Figure 4.22: Fitter

Type	ID	Message
⚠	20028	Parallel compilation is not licensed and has been disabled
⚠	20028	Parallel compilation is not licensed and has been disabled
⚠	292013	Feature LogicLock is only available with a valid subscription license. You can purchase a software subscription to gain full access to this feature
⚠	332153	Family doesn't support jitter analysis.
⚠	169174	The Reserve All Unused Pins setting has not been specified, and will default to 'As output driving ground'.
⚠	20028	Parallel compilation is not licensed and has been disabled
⚠	332153	Family doesn't support jitter analysis.

Figure 4.23: Warning

Chapter 5

Alternative Architectures in ADPLLs

After the presentation of the two complete ADPLL architectures considered in this work, it is useful to briefly review additional architectural solutions that are commonly found in the literature. Over the years, a wide variety of digital phase detectors, loop filters, and digitally controlled oscillators have been proposed, each targeting specific performance objectives and application requirements.

These alternative architectures are not implemented in the current design, but their discussion helps to place the proposed solutions within a broader design context. Moreover, it highlights that the architecture of an ADPLL is not unique, and that different combinations of building blocks can be employed depending on constraints such as frequency range, resolution, power consumption, implementation complexity, and robustness.

5.1 Ring Oscillator

Among the possible architectures for the DCO, the ring oscillator represents a widely adopted solution due to its simplicity, full digital compatibility, and ease of frequency tuning. Figure 5.1 illustrates a basic ring oscillator architecture composed of an odd number of inverting stages connected in a closed feedback loop.

5.1.1 Principle of Operation

A ring oscillator consists of N cascaded inverter stages, where N must be an odd number to satisfy the oscillation condition. Each inverter introduces a propagation delay t_d, and the total loop delay determines the oscillation frequency.

Let t_d be the average propagation delay of a single inverter stage. The oscillation period T_{osc} is given by

$$T_{osc} = 2 \cdot N \cdot t_d \tag{5.1}$$

and therefore the oscillation frequency is

$$f_{osc} = \frac{1}{2 \cdot N \cdot t_d} \tag{5.2}$$

The factor of 2 accounts for the fact that a full oscillation period requires a rising and a falling transition to propagate through all stages.

5.1.2 Ring Oscillator as a Digitally Controlled Oscillator

In an ADPLL, the ring oscillator can be transformed into a Digitally Controlled Oscillator by making its effective delay digitally programmable. This is typically achieved by modifying either the number of active delay stages or the delay of individual stages using digital control signals.

A common approach consists in implementing multiple selectable delay paths; these paths may include:

- Additional inverter stages that increase the total loop delay.
- Bypass paths that reduce the number of active inverters.
- Digitally enabled load elements that modify the propagation delay of individual stages.

If M selectable delay elements are available, the effective delay can be expressed as

$$t_{d,\text{eff}} = t_{d,0} + \sum_{i=1}^{M} b_i \cdot \Delta t_{d,i} \tag{5.3}$$

where $b_i \in \{0, 1\}$ are digital control bits and $\Delta t_{d,i}$ represents the incremental delay associated with the i-th element. Consequently, the DCO frequency is given by

$$f_{DCO} = \frac{1}{2 \cdot N \cdot t_{d,\text{eff}}} \tag{5.4}$$

This structure allows for a digitally quantized frequency tuning characteristic, which is well suited for all-digital control loops.

5.1.3 Coarse and Fine Tuning in ADPLL

To achieve both wide tuning range and high frequency resolution, ring oscillator based DCOs are often designed with hierarchical tuning mechanisms:

- **Coarse tuning**, implemented by selecting different delay paths or enabling additional inverter stages.
- **Fine tuning**, achieved by digitally adjusting small capacitive or resistive loads within each inverter stage.

This separation allows the ADPLL to first lock using coarse tuning and then minimize the steady-state phase error through fine tuning.

5.1.4 Advantages and Limitations

Ring oscillator based DCOs offer several advantages in ADPLL applications:

- Fully digital implementation.
- Small silicon area and simple architecture.
- Wide tuning range.

5.1.5 Implementation Constraints on FPGA

When implemented on an FPGA, the ring oscillator based DCO is constrained by the purely digital nature of the available resources. Unlike ASIC implementations, it is not possible to explicitly add tunable capacitive or resistive loads to control the inverter delay. Therefore, frequency tuning is achieved by exploiting FPGA-specific resources such as logic buffers, LUT-based delay elements, multiplexed paths, and routing-induced delays. As a result, the achievable frequency resolution and phase noise performance are inherently limited by the granularity and variability of the FPGA fabric.

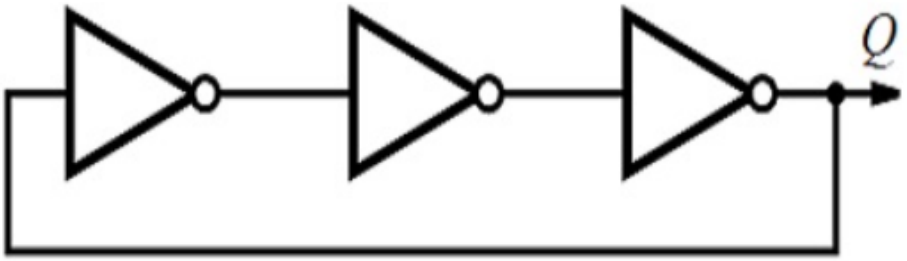

Figure 5.1: Ring oscillator

Chapter 6

Conclusions

This report has presented the design and verification of two distinct architectural approaches. The two ADPLL architectures demonstrated differences in terms of design complexity and tracking precision. Ultimately, the choice between these configurations depends strictly on the specific target application requirements and the available hardware resources.

6.0.1 Comparison of the two architectures

Architecture 1

The first architecture employs a simple XOR as the phase detector (PD). As is well-known, the XOR PD allows for phase detection but only guarantees locking in frequency, not in phase. Consequently, at steady state, the frequency difference between the reference and the DCO output remains non-zero, resulting in a quadrature condition rather than exact phase alignment.

The loop filter is implemented using a counter that acts as an integrator. This effectively accumulates the PD output to produce a control signal for the DCO. The DCO itself is designed as a finite state machine (FSM), which adjusts its output based on the carry and borrow signals generated by the integrator counter.

Architecture 2

The second architecture uses a phase-frequency detector (PFD) as the phase detector. This type of PD provides both phase and frequency detection, enabling a faster and more reliable lock. The PFD is widely used in modern ADPLL designs due to its superior precision.

The loop filter is a proportional-integral (PI) filter, which provides both immediate and accumulated corrections to the DCO. The DCO is implemented as an oscillator whose output is modulated based on a control threshold, which is dynamically adjusted according to the word received from the PI filter.

Comparison and Discussion

Comparing the two architectures, several key differences emerge:

- **Phase Detector:** XOR PD (Arch. 1) vs PFD (Arch. 2): the PFD enables both phase and frequency locking, improving steady-state accuracy.

- **Loop Filter:** Counter integrator (Arch. 1) vs PI filter (Arch. 2): the PI filter allows simultaneous proportional and integral corrections, improving dynamic response and stability.
- **Resource Utilization:** Arch. 2 uses fewer resources (147 LEs, 71 registers) than Arch. 1 (197 LEs, 145 registers).

Overall, the second architecture demonstrates superior performance in terms of phase accuracy, lock range, and resource efficiency, while the first architecture, although simpler, suffers from steady-state phase error and higher resource consumption.

6.0.2 State-of-the-Art ADPLL Trends

Looking at the current state-of-the-art, ADPLLs have largely superseded traditional Analog PLLs in SoC (System-on-Chip) design. Modern industry trends are moving towards **Bang-Bang Phase Detectors** (BBPD) and **Time-to-Digital Converters** (TDC) with sub-picosecond resolution to further reduce jitter.

Furthermore, today's "smart" ADPLLs often incorporate **machine learning-based calibration** to dynamically adjust filter parameters (K_p, K_i) in response to PVT (Process, Voltage, Temperature) variations. This ensures that the "LSB ripple" observed in our K_{out} signal remains minimized across all operating conditions.