

# Effektive Systemsimulation einer 4 GHz $\Delta\Sigma$ Fractional-N PLL durch pseudo-analoge Verhaltensmodellierung in VHDL

**Christian Munker**

C. Munker

SMS HW RFE

**RF**  
**ICs**

stop thinking  
never

# Übersicht

---

- **Verifikation von Mixed-Signal (MS) Systemen**
- **Fallbeispiel: GSM Transceiver**
- **Mixed-Signal Simulation mit VHDL**
- **Rauschsimulation mit VHDL**
- **Alternativen zur MS – Simulation mit VHDL**
- **Zusammenfassung und Ausblick**

## Mixed-Signal Verifikationsflow (1)

---

- Moore's Law: „Alle 18 ... 24 Monate verdoppelt sich Anzahl der möglichen Transistoren auf Chip“
  - Wettbewerbsdruck erzwingt, Integrationsdichte zu erhöhen, immer komplexere Algorithmen on-Chip
  - Aber: für MS-Chips kein vergleichbarer hierarchisch strukturierter Flow wie für Digitalchips
  - Lücke zwischen Konzept und Blocklevel
- NEU:** nutze Techniken der digitalen Signalverarbeitung, um die Methodik des Digitaldesigns für Analogblöcke zugänglich zu machen!

# Mixed-Signal Verifikationsflow (2)

<i>Level</i>	<i>Beispiel</i>	<i>Tools</i>
<b>Konzept / Algorithmus</b>	Direct Conversion	Matlab, CoCentric, Excel, C++
<b>System</b>	GSM Transceiver	
<b>Block / RTL</b>	VCO (Modell) μC (Modell, RTL)	VHDL / Verilog / Mixed-Signal Simulatoren
<b>Gatelevel</b>	Synthetisierte Netzliste	
<b>Transistor-Netzliste</b>	VCO (detailliert)	SPICE
<b>Layout</b>		Layout-Tools
<b>Device</b>	RF-Transistor	Feldsimulator

stop thinking  
never

# Analog- vs. Digitalsimulator

---

## Analogsimulator

- Löst DGL - Systeme
- Löst implizite Funktionen
- Orientiert an Struktur
  
- Zeit- und wertkontinuierlich
  
- Zeitschritt-Automatik
- Analoges Postprocessing

## Digitalsimulator

- Event getrieben
- Nur explizite Funktionen
- Orientiert an Datenfluss / Algorithmen
  
- Zeitdiskret und wertkontinuierlich
  
- Zeitschritt vorgegeben
- Digitales Postprocessing

# VHDL für Analo­gsimulationen?

---

## Features

- Wertkontinuierlich durch Real – Darstellung
- Umfangreiche Bibliotheken mit math. Funktionen

## Am besten geeignet für

- Systeme mit klar getrennten Funktionsblöcken, die sich algorithmisch beschreiben lassen
- Systeme ohne (oder nur mit langsamen) Schleifen im analogen Teil
- Überwiegend zeitdiskrete Systeme

## Beispiele

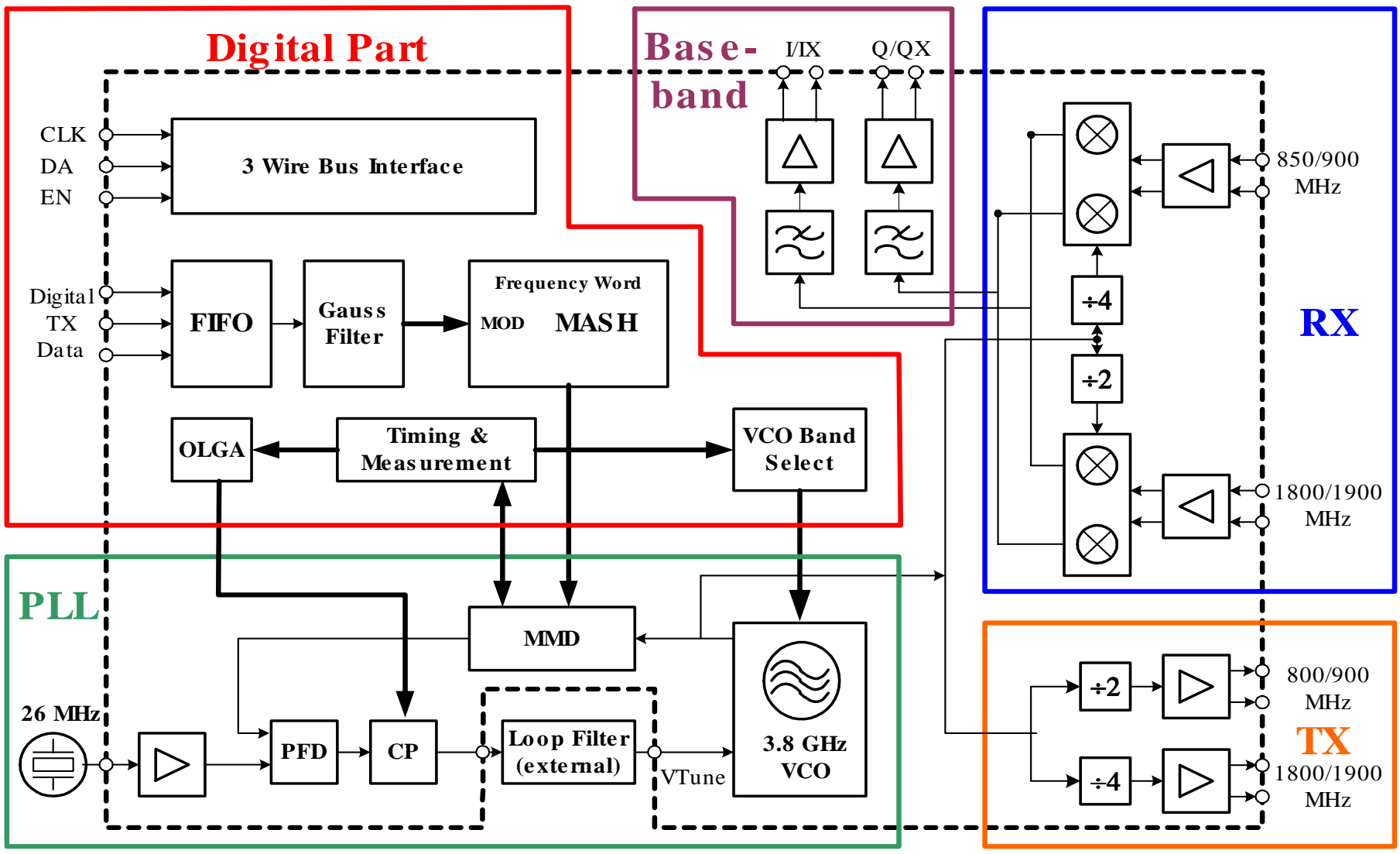
- PLLs, digitale Kalibrierungen, digitale Signalverarbeitung mit ADC / DAC

# Numerische Grenzen von VHDL

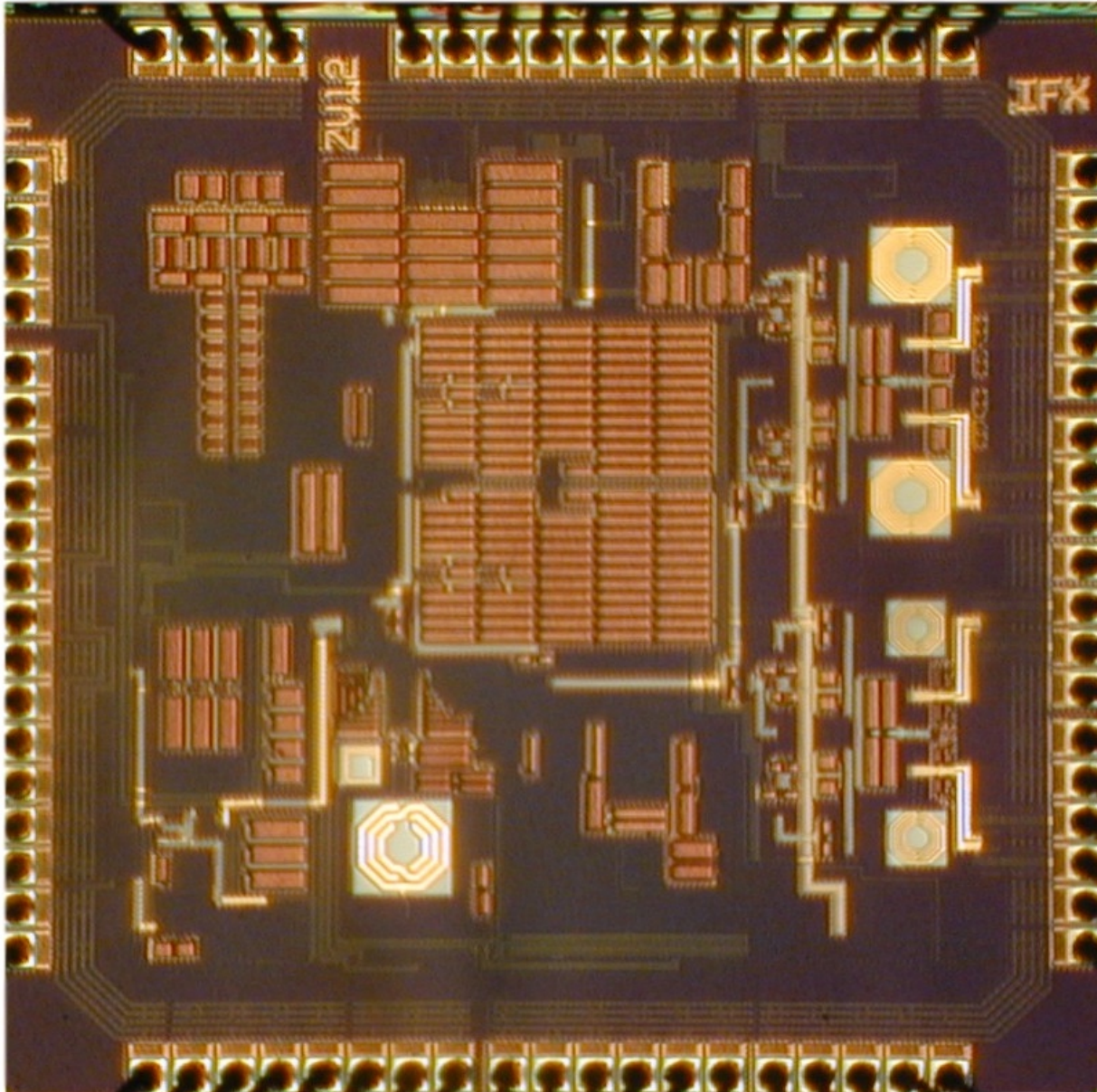
---

- Typ **integer**: üblicherweise 32 Bit  
⇒  $0 \dots 2.1 \cdot 10^9$
- Typ **time** repräsentiert Zeit / Events  
Min. Zeitschritt: 1 fs, 64 Bit Bereich  
⇒  $1 \text{ fs} \dots 2^{63} \text{ fs} \approx 2 \frac{1}{2} \text{ Stunden}$
- Typ **real** (Double Precision ANSI / IEEE Std.):  
11 Bits Exponent, 53 Bits Mantisse  
⇒  $\pm 2.2 \cdot 10^{-308} \dots 1.8 \cdot 10^{308}$  mit  $\epsilon = 2.2 \cdot 10^{-16}$

# Fallbeispiel: Quad Band GSM Transceiver



# Chipfoto GSM Transceiver



**Fläche: 5 mm<sup>2</sup>**

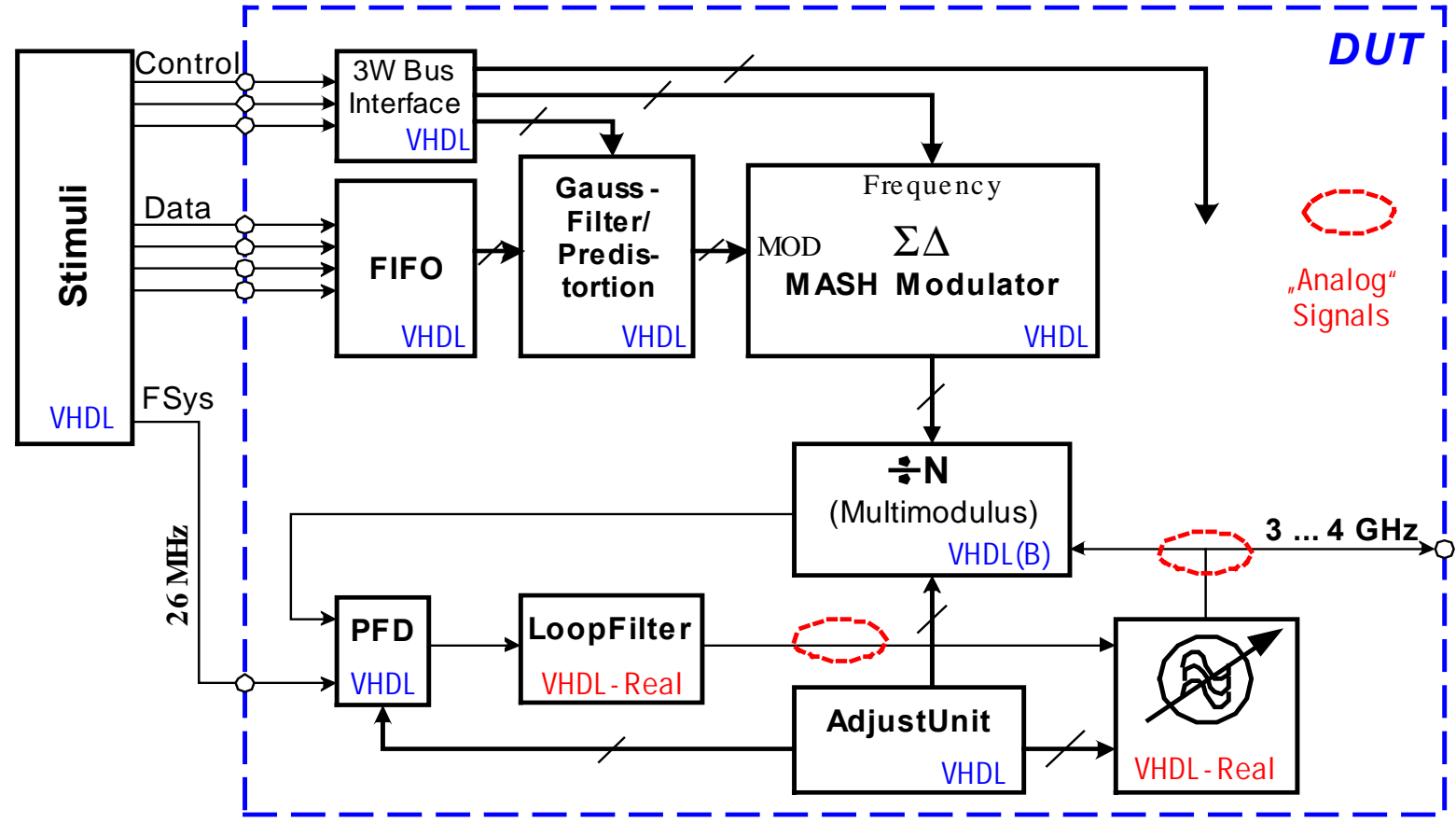
**Technologie:  
0.13 µm CMOS**

**RX: 250 mW**

**TX: 210 mW**

**Gehäuse:  
48 Pin VQFN**

# Simulationssetup zur Verifikation des TX Pfads

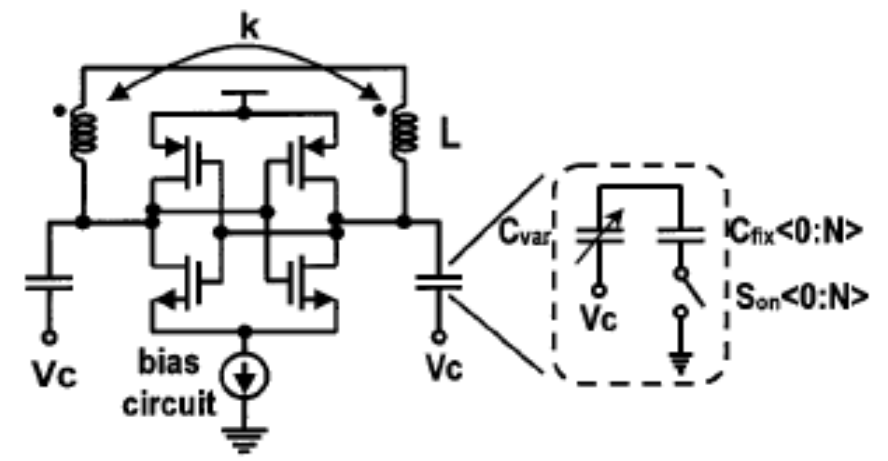
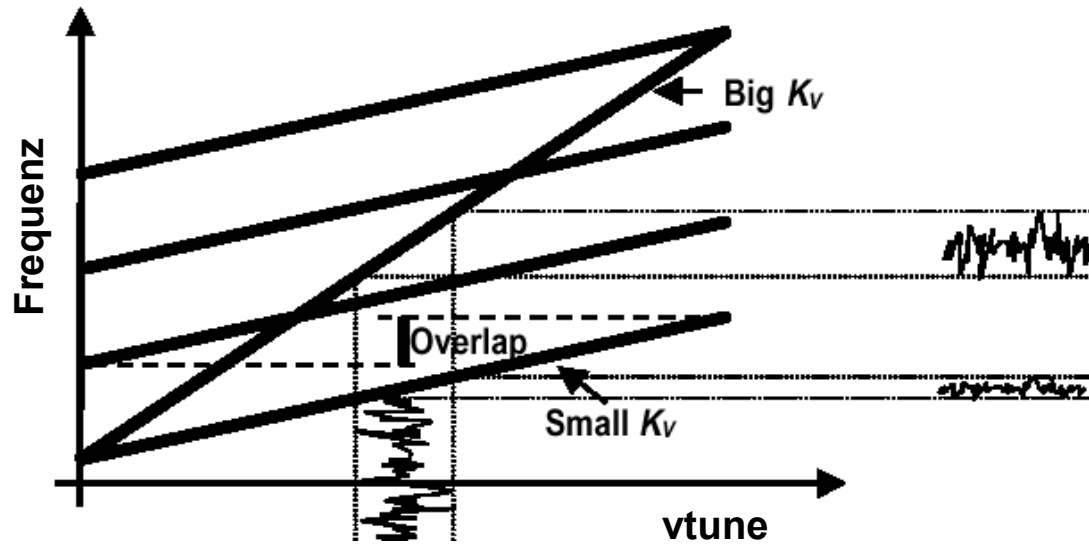


**Verifikationsprobleme: Digitale Kalibrierung, MASH-Modulator, Closed Loop Betrieb, Digitale Modulation, ...**

# VCO mit digitalem Pretuning

## Mehrere Bänder

- Großer Abstimmbereich mit kleinem  $K_V$   
 $\Rightarrow$  unempfindlich gegen Störungen
- Overlap: keine „Löcher“ im Band,  
 muß größer sein als Modulationshub
- $K_V$  soll konstant über Bänder bleiben  
 $\Rightarrow$  zwei Kalibrierungen:  
 Bandauswahl und Steilheit



# Modellierung des VCOs (1)

---

- **VCO kann auf Systemebene als zeit- und wertdiskret betrachtet werden**
- **Einschränkung: Zeitdiskretisierung der Frequenzänderung**
  - ⇒ Steuerspannung beeinflusst nur einmal pro Periode die Periodendauer
- **Realisierung als Verhaltensmodell, angelehnt an Schaltung:**
  - Frequenzberechnung aus Schaltungsparametern
  - Umsetzung Periode → Event

## Modellierung des VCOs (2)

---

```
VCOLoop : loop
    vco_out  <= transport '1' after delay_t,
                '0' after period_t/2;

    wait for period_t;

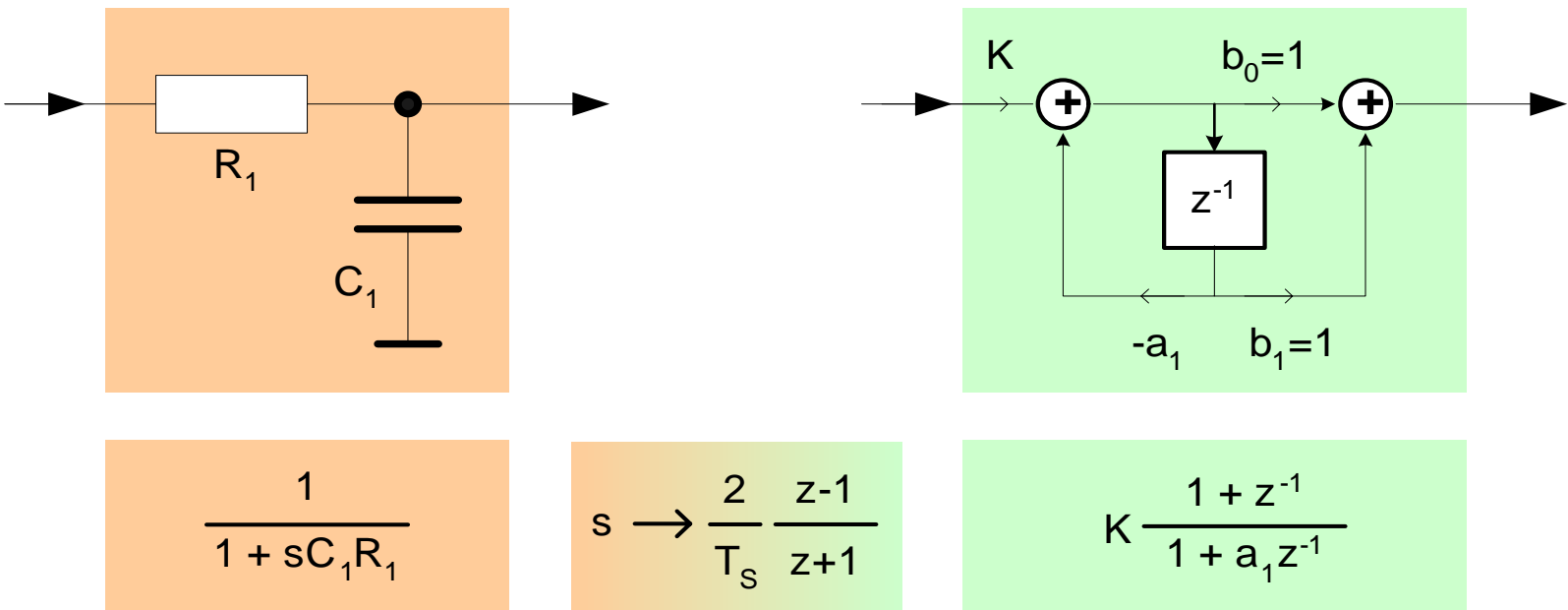
    C_tot    := (vt_max - vt) * C_var + C_fix;
    period   <= 2.0 * math_pi * sqrt(L_0 *
                C_tot) * 1.0e3;

    period_t <= period * fs;  -- period in fs
    delta_f  <= (1.0e6/period - F_TARGET)
                * 1.0e9;

end loop VCOLoop;
```

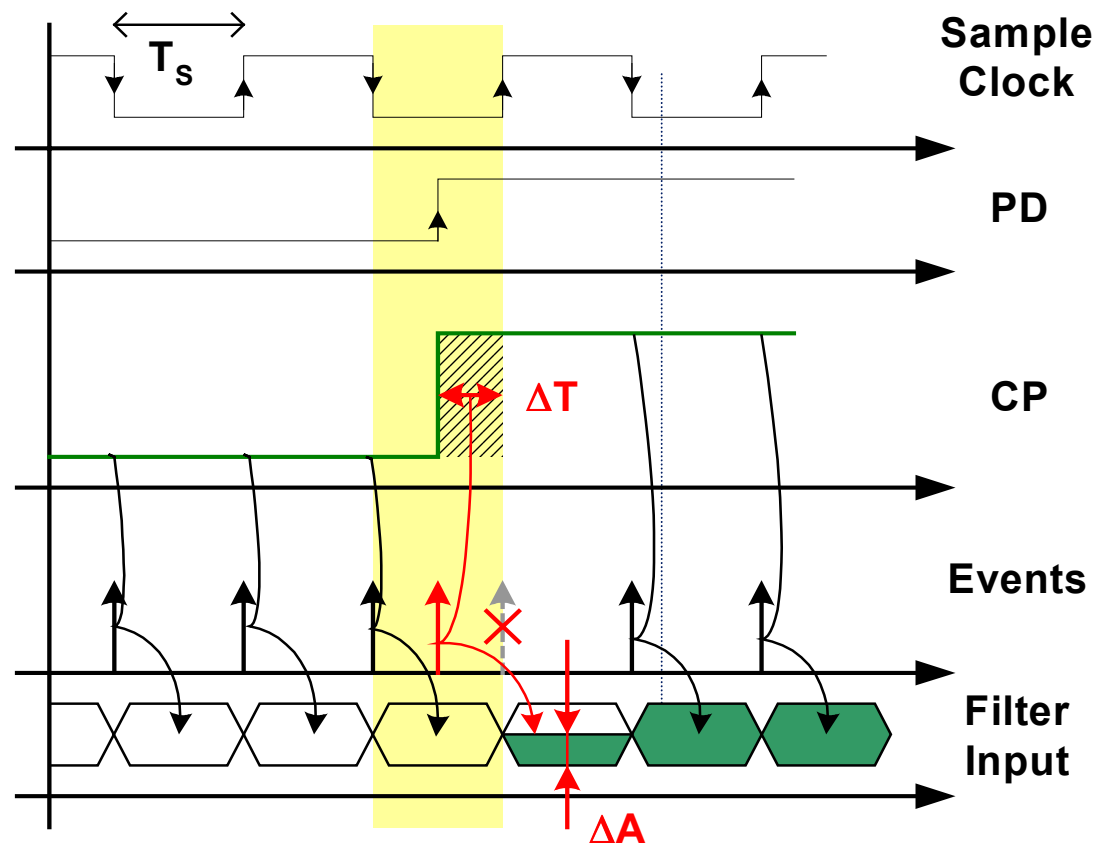
# Modellierung des Loop Filters

- Problem: zeit- und wertkontinuierlich
- Lösung: Transformation s-Ebene -> z-Ebene



- Abtastrate  $T_s \Rightarrow$  Mittlerer Timing Fehler  $T_s/2$

# Fractional Sampling im Filter

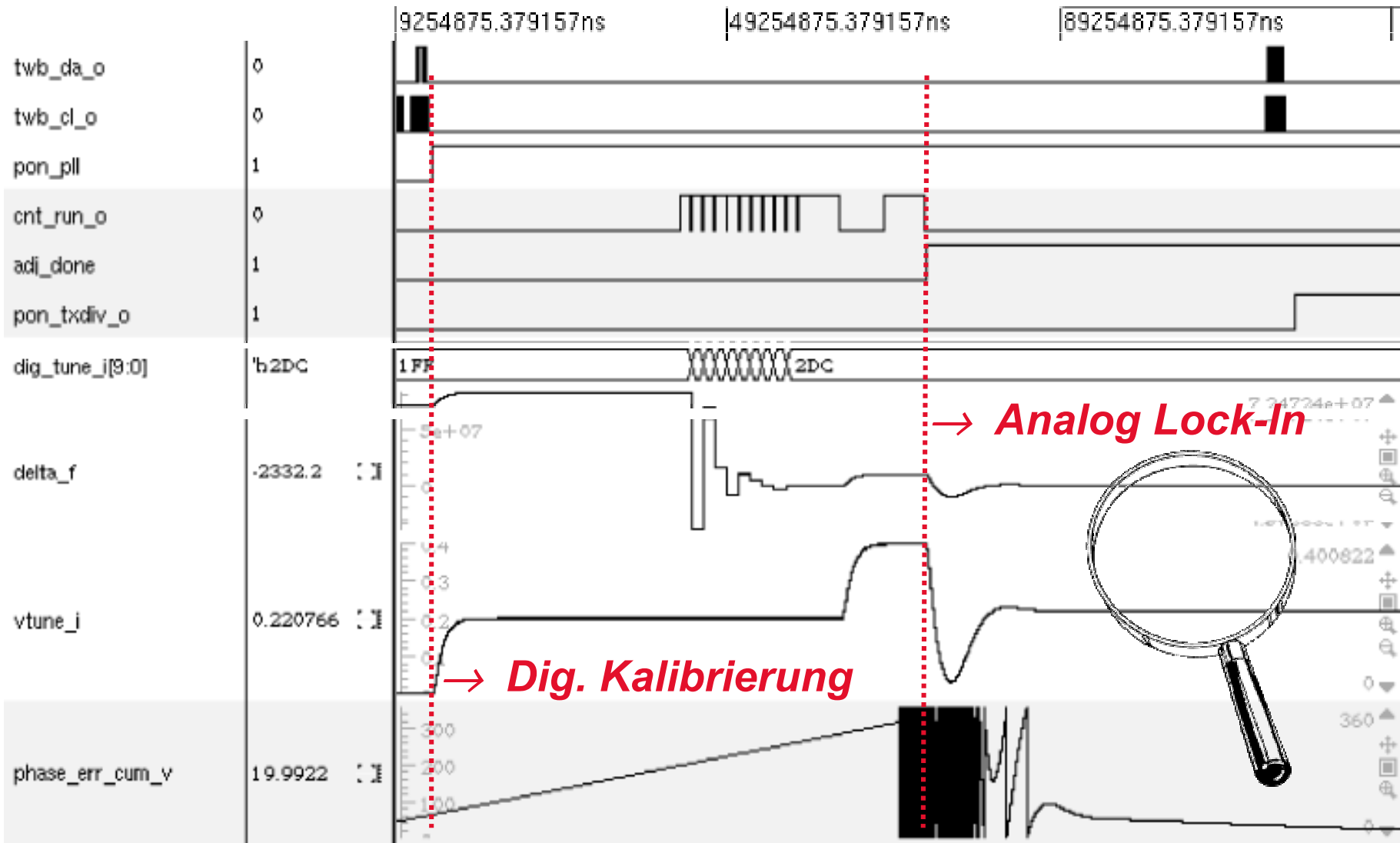


**Variation im Zeitbereich  $\Delta T$  wird dargestellt durch Variation der Amplitude  $\Delta A$**

**$\Rightarrow$  Sampling Intervall  $T_s$  im Filter bleibt konstant,  $\Delta T$  wird durch Interpolation trotzdem berücksichtigt**

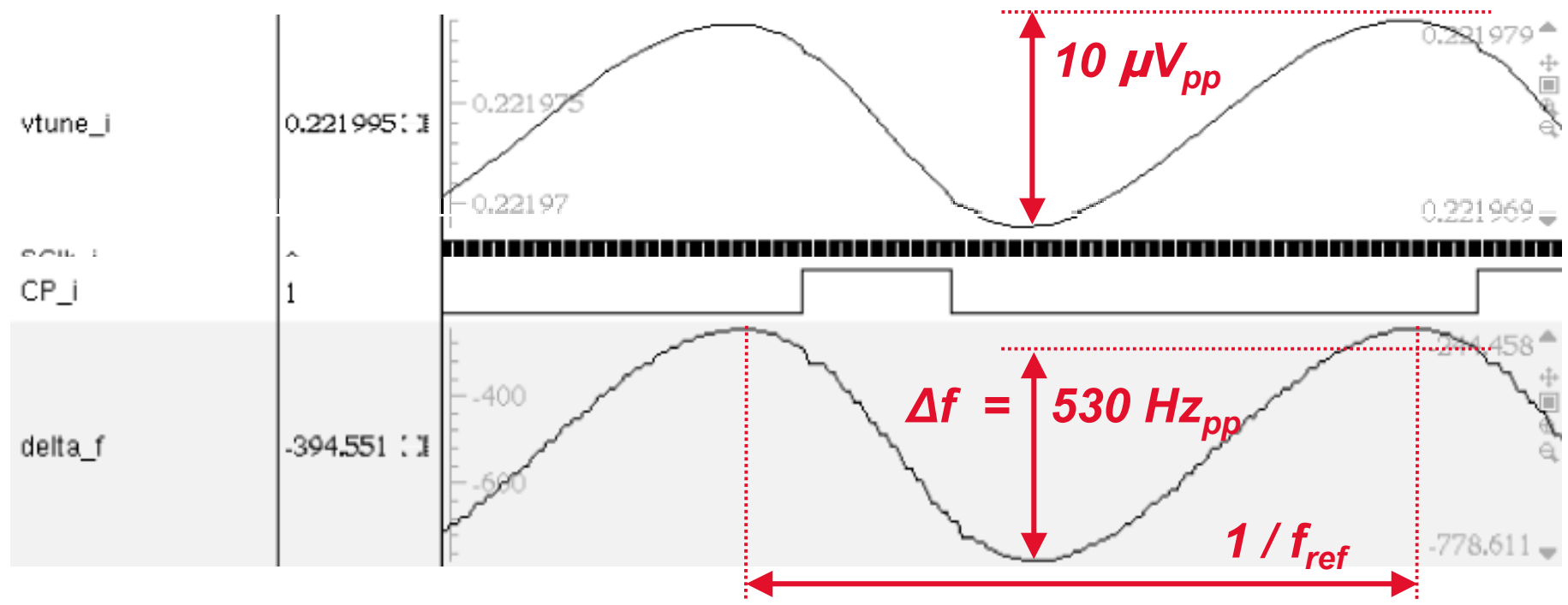
# Simulation: Einrasten der PLL

stop thinking  
never



**Verifikation: Timing, Settling, Phasenfehler**

# Simulation: Restwelligkeit nach Loop Filter

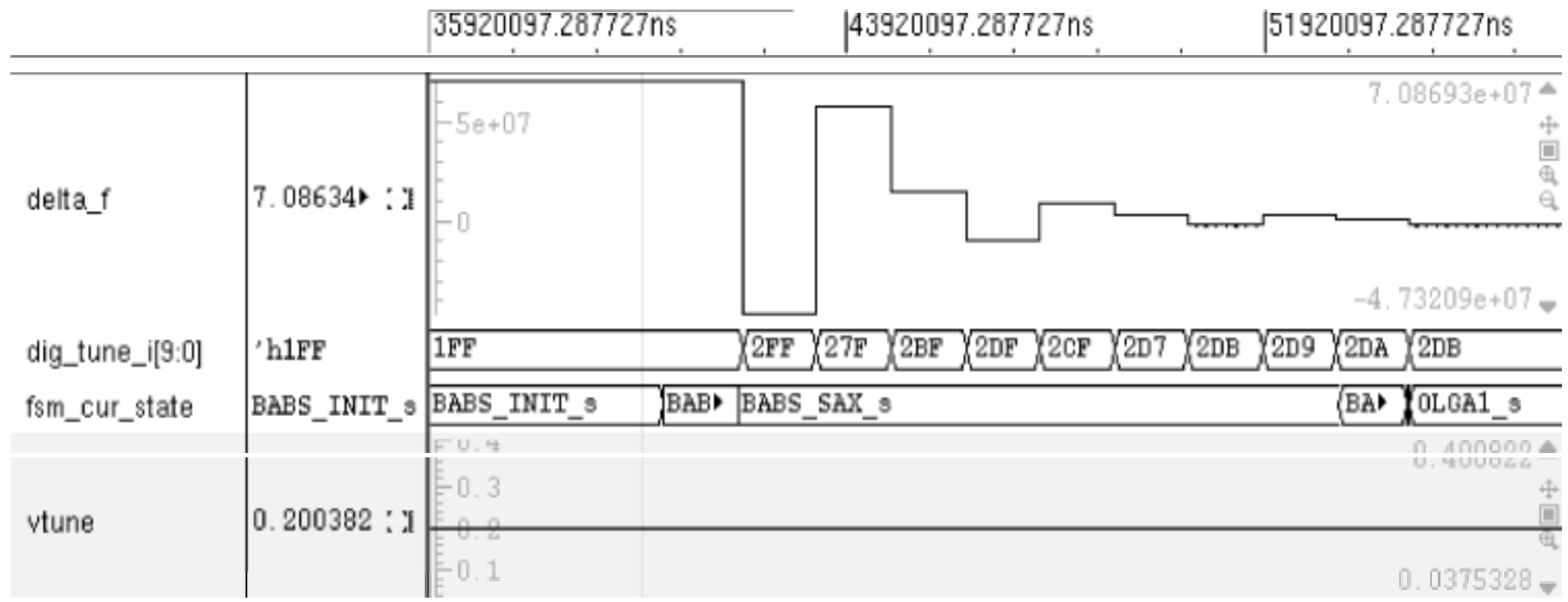


**Restwelligkeit nach Loopfilter verursacht periodischen Frequenzfehler, messbar als Spurious Sideband:**

$$spur = \frac{1}{2} \frac{\Delta f}{f_{ref}} = \frac{265 \text{ Hz}}{2 \cdot 26 \text{ MHz}} = 5.1 \cdot 10^{-6} \equiv -106 \text{ dBc}$$

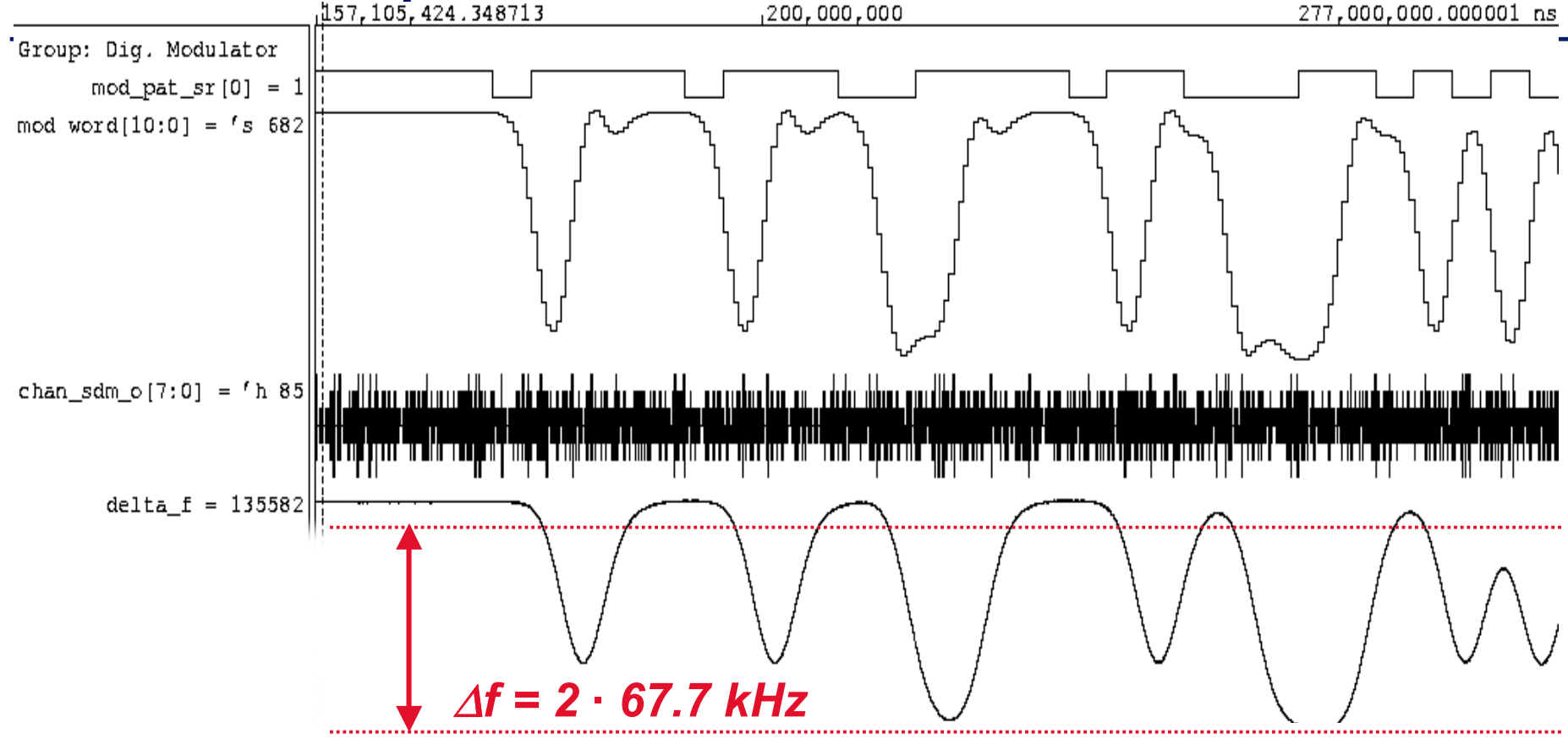
stop thinking  
never

# Simulation: Digitale VCO Kalibrierung



**Verifikation: digitaler Abgleich der VCO Bänder  
(Algorithmus, Timing, Restfehler etc.)**

# Simulation: Digitale Modulation



**Verifikation: Gauss-Filter mit Vorverzerrung,  
Modulationshub am TX-Ausgang,  
Modulationsspektrum**

stop thinking  
Never

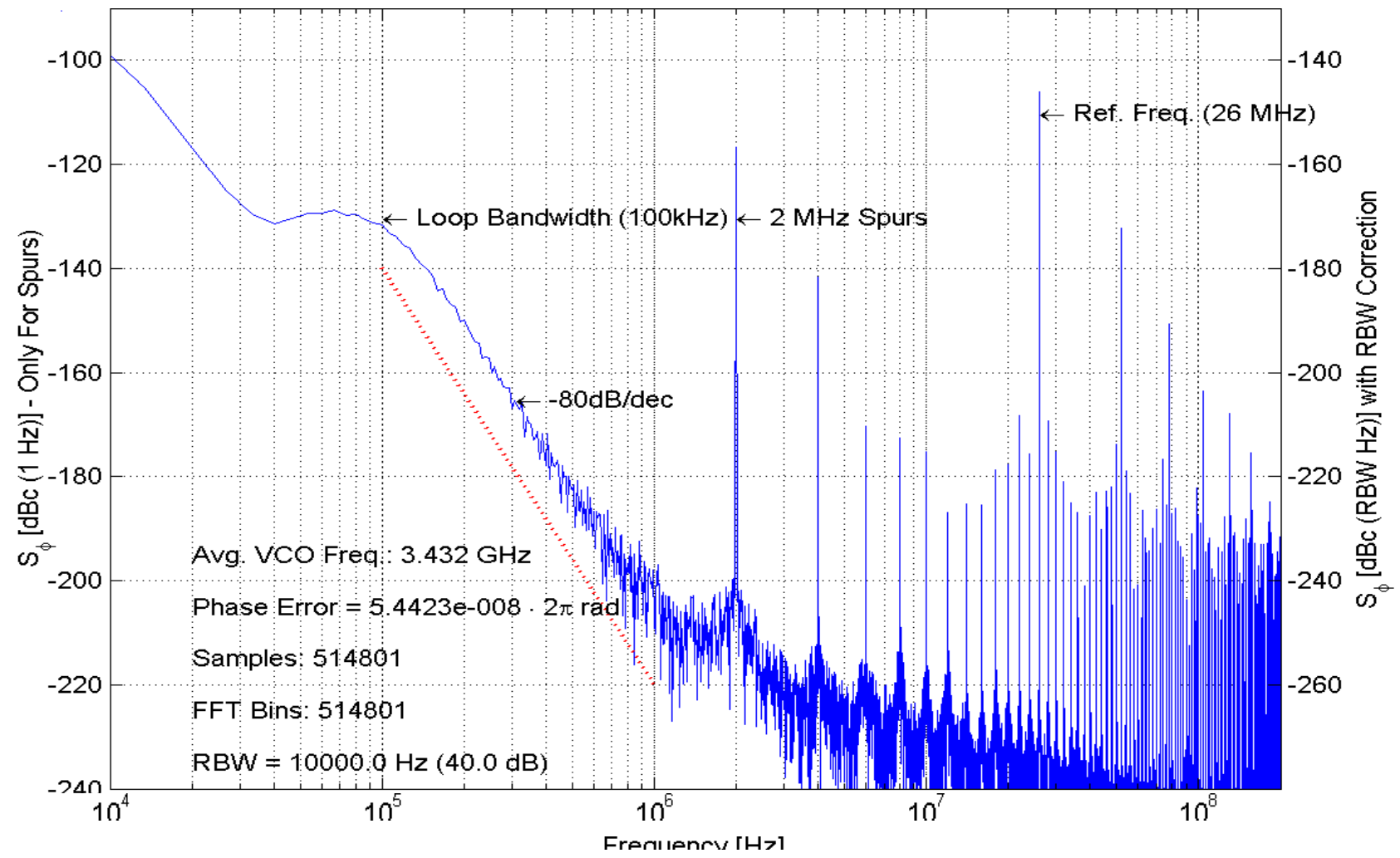
# Rauschsimulation mit VHDL

---

- **VHDL als zeitschrittgesteuerter Simulator kann Rauschen nur als Jitter / Zufallsprozess darstellen**
- **Generiere Zufallsprozess mit Gauss'scher Verteilung**
- **Rechne Phase Noise in Jitter um**
- **Modelliere analoge / digitale Blöcke mit Jitter**
- **Ext. Postprocessing (z.B. Matlab) für spektrale Darstellung und weitere Analysen**

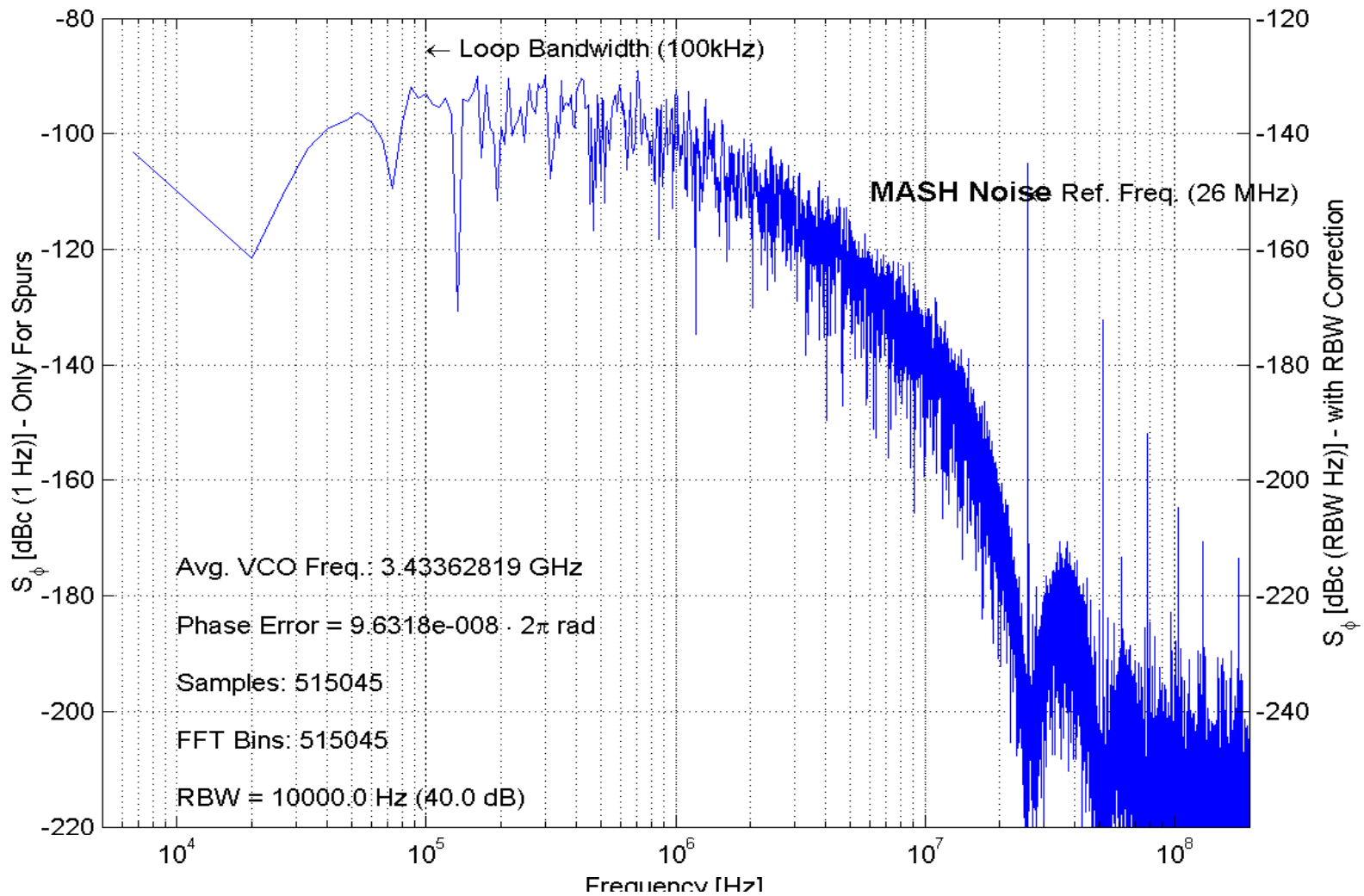
# Rauschsimulation: Grenzen der Genauigkeit

Power Spectral Density of PLL Output Phase (Integer Frequency, No Added Jitter)



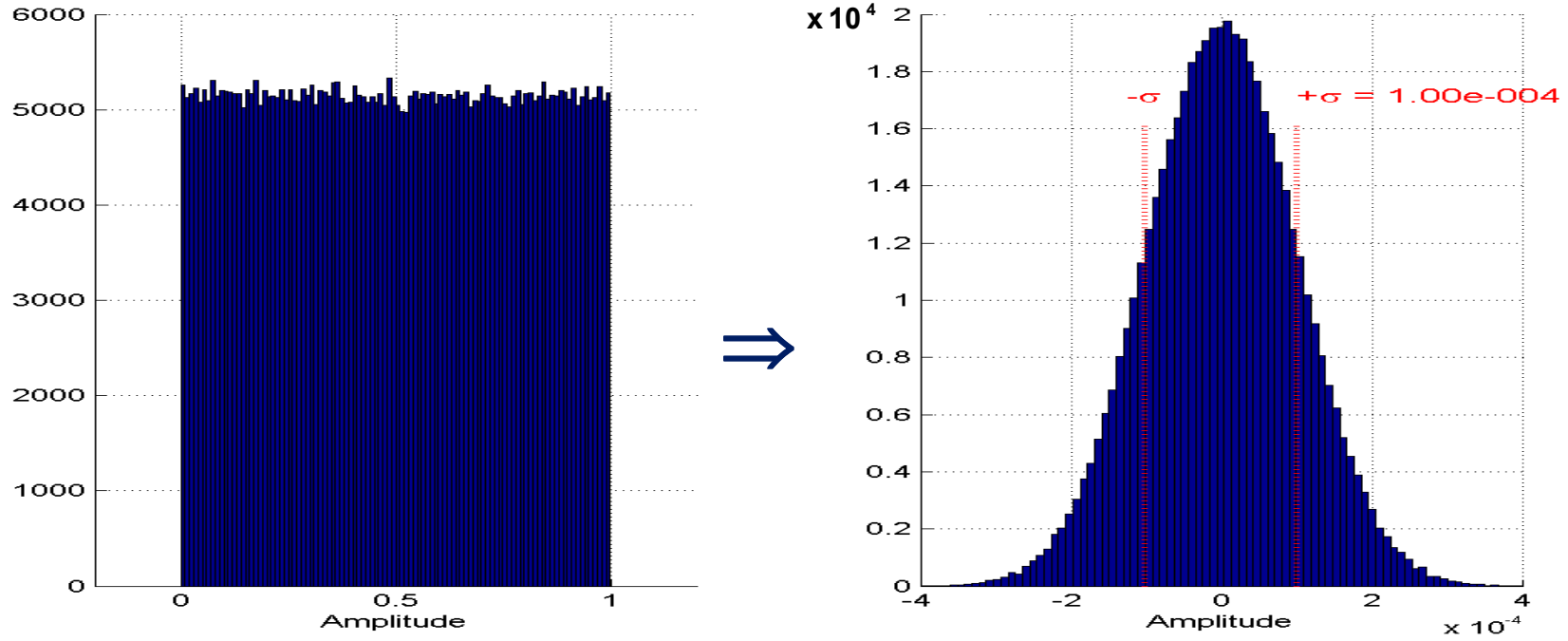
# Rauschsimulation: MASH Rauschen

Power Spectral Density of PLL Output Phase (Fractional Frequency, No Added Jitter)



stop thinking  
never

# Jitter Generierung in VHDL



**Umwandlung von gleichförmig verteilten Zufallsfolgen in gaussverteilte:**

$$G_{n,1} = m_1 + \sigma_1 \sqrt{2 \ln \frac{1}{1 - U_{n,1}}} \cos 2\pi U_{n,2}$$

$$G_{n,2} = m_2 + \sigma_2 \sqrt{2 \ln \frac{1}{1 - U_{n,1}}} \sin 2\pi U_{n,2}$$

stop thinking  
Never

# Umrechnung von Phase Noise in Jitter (1)

Jitter  $J_{UI}$  ist die relative Störung pro Periode (Unit Intervall, UI):

$$J_{UI}(t) = \frac{\Delta T(t)}{T_0} = \frac{\Delta \phi(t)}{2\pi}$$

$S_\phi(f)$  ist die spektrale Leistungsdichte der Phasenänderung:

$$\int_{f_1}^{f_2} S_\phi(f) df = \Delta \phi_{RMS}^2$$

$$\Rightarrow J_{UI,RMS} = \frac{\Delta \phi_{RMS}}{2\pi} = \frac{1}{2\pi} \sqrt{\int_{f_1}^{f_2} S_\phi(f) df}$$

Phasenrauschen  $L(f)$  ist  $S_\phi(f)$  um Träger  $f_0$  gefaltet:

$$L(f) \approx \frac{1}{2} S_\phi(f - f_0)$$

## Umrechnung von Phase Noise in Jitter (2)

**PM mit weissem Rauschen:  $S_{\phi}(f) = S_{PM} = \text{const.}$**

$$\Rightarrow J_{UI,RMS,PM} = \frac{1}{2\pi} \sqrt{S_{PM} (f_2 - f_1)}$$

$$f_1 = 1 \text{ MHz}, f_2 = 1.7 \text{ GHz}, S_{PM} = -150 \text{ dBc} \Rightarrow J_{UI,RMS,PM} = 1.41 \cdot 10^{-4}$$

**FM mit weissem Rauschen  $\Rightarrow S_{\phi}(f) = S_{FM} (1 \text{ Hz}) / f^2$**

$$\Rightarrow J_{UI,RMS,FM} = \frac{1}{2\pi} \cdot \sqrt{S_{FM} (f_1^{-1} - f_2^{-1})}$$

$$f_1 = 100 \text{ kHz}, f_2 = 1 \text{ GHz}, S_{FM} = 0 \text{ dBc} \Rightarrow J_{UI,RMS,FM} = 5.0 \cdot 10^{-4}$$

# Rauschmodellierung des VCOs

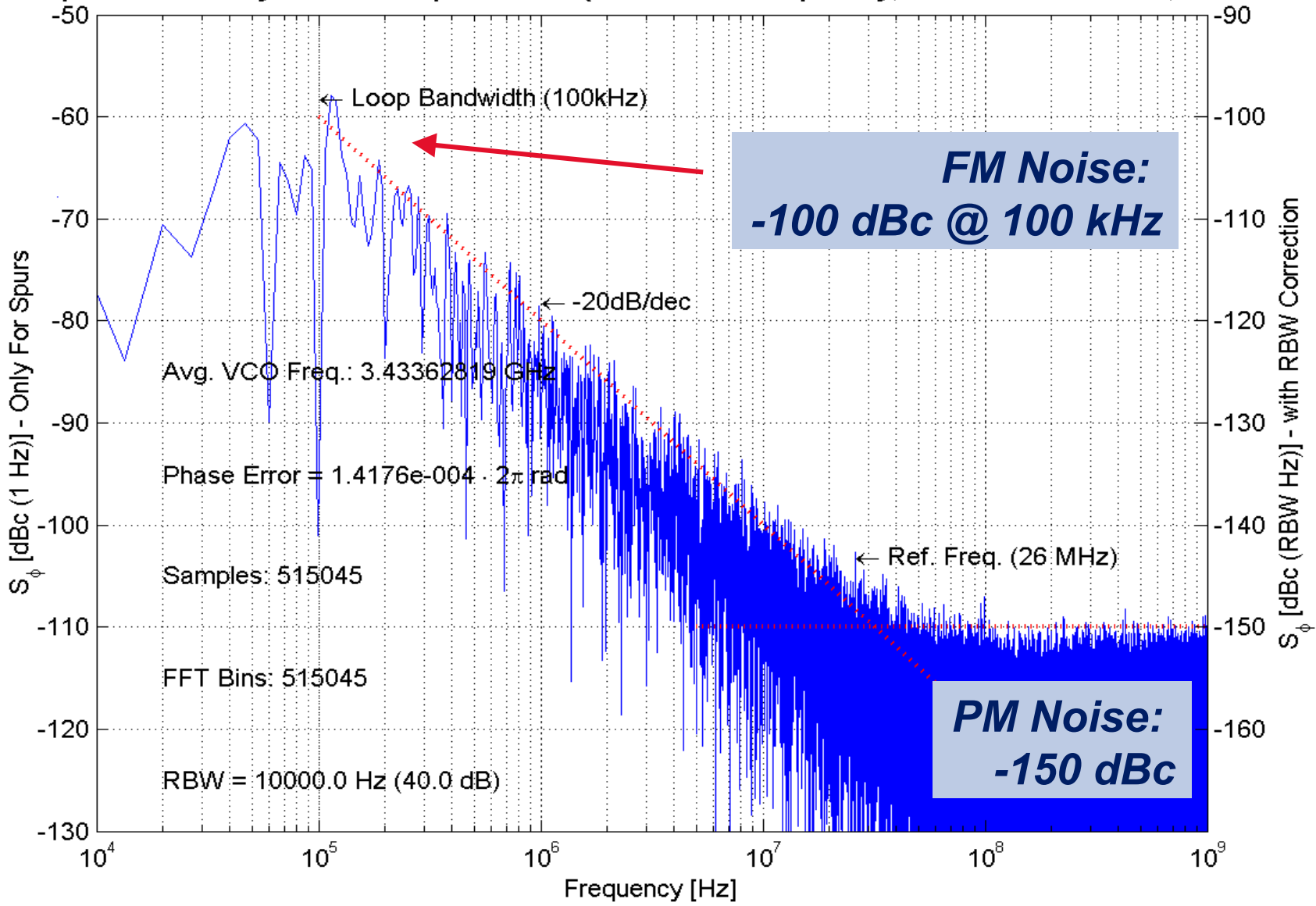
---

- **FM Jitter: moduliere Periodendauer**
- **PM Jitter: Addiere Jitter zu Periodenlänge, subtrahiere Jitter im nächsten Schritt  
⇒ keine Änderung der Periodendauer**

```
period  <=  2.0 * math_pi * sqrt(L_0 *  
            C_tot) * 1.0e3  
            * (1.0 + fm_jitter)  
            + pm_jitter - pm_jitter_last;
```

# Rauschsimulation: PLL mit PM und FM Jitter im VCO

Power Spectral Density of PLL Output Phase (Fractional Frequency, PM:  $\sigma = 1.41 \cdot 10^{-4}$ , FM:  $\sigma = 10^{-4}$ )



stop thinking never

# Alternativen zur Systemsimulation mit VHDL

---

## ■ Matlab zur Modellierung analoger Blöcke + VHDL

⇒ Schneller Transfer von der Konzeptebene, aber keine Verifikation von Connectivity, Schleifen nur schwierig implementierbar

## ■ ADS System (Ptolemy) + VHDL + Matlab

⇒ Sehr mächtiges Tool, aber steile Lernkurve

## ■ Mixed-Signal Simulatoren

⇒ Zu ressourcenhungrig für komplexe Systeme

# Zusammenfassung

---

- Durch algorithmische Modellierung können analoge Blöcke mit Standard VHDL Simulatoren simuliert werden
  - Zeitkontinuierliche Systeme lassen sich durch Transformation  $s \rightarrow z$ -Ebene modellieren
  - Durch „Fractional“-Sampling läßt sich Simulationsgenauigkeit stark verbessern
  - Damit können auch analoge Blöcke effektiv in einer digitalen Designumgebung verifiziert werden
  - Phase Noise kann effektiv im System als Jitter mitsimuliert werden, da keine zusätzlichen Ereignisse erzeugt werden müssen
- ⇒ **Schließen der Verifikationslücke zwischen System- und Schaltungsebene für Mixed-Signal Systeme**

# Ausblick

---

- **Bessere Modellierung von Rausch / Jittereffekten (Flickernoise, bandbegrenztes Rauschen, Amplitudenrauschen)**
- **Berücksichtigung von diskreten Störern (Spurious Sidebands)**
- **Modellierung von Systemen mit Rückwirkung**
- **Verbessertes Postprocessing, mehr Meßfunktionalität in VHDL (FFT etc.)**