Short Course On
Phase-Locked Loops and Their Applications
Day 3, AM Lecture

Advanced Analog Synthesizer Techniques

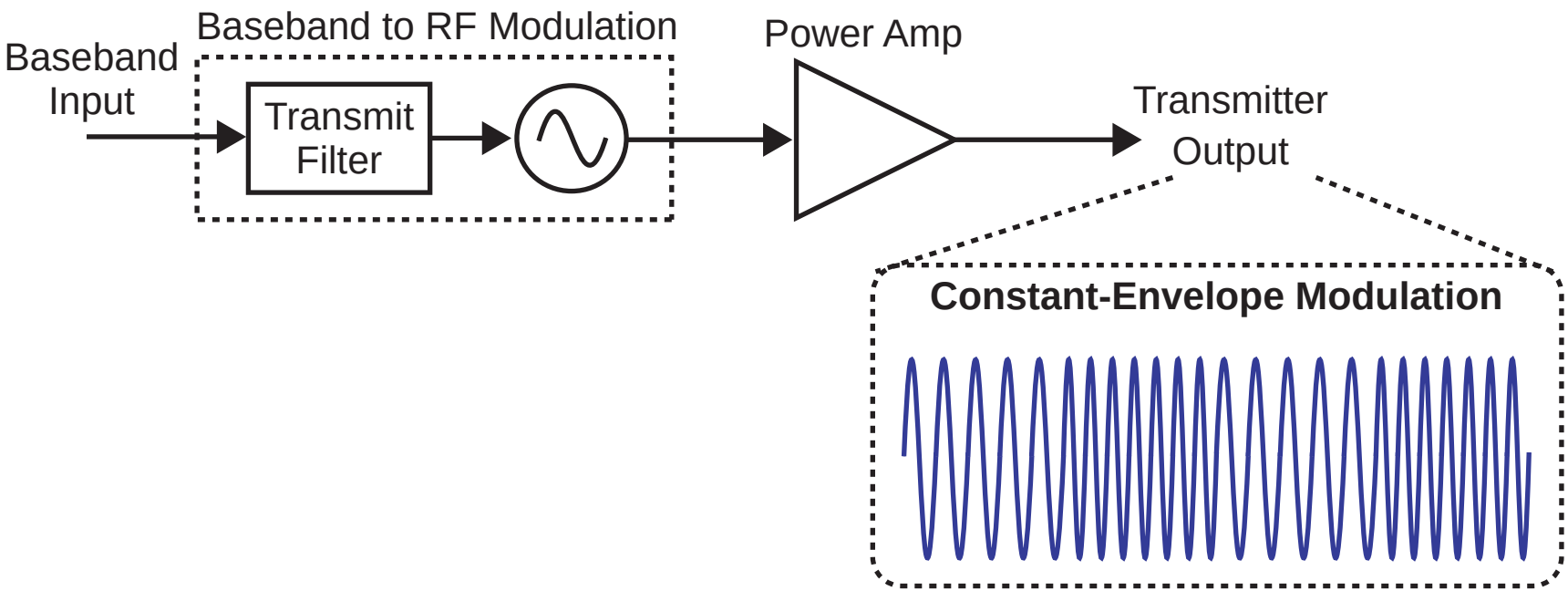
Michael Perrott

August 13, 2008

Outline

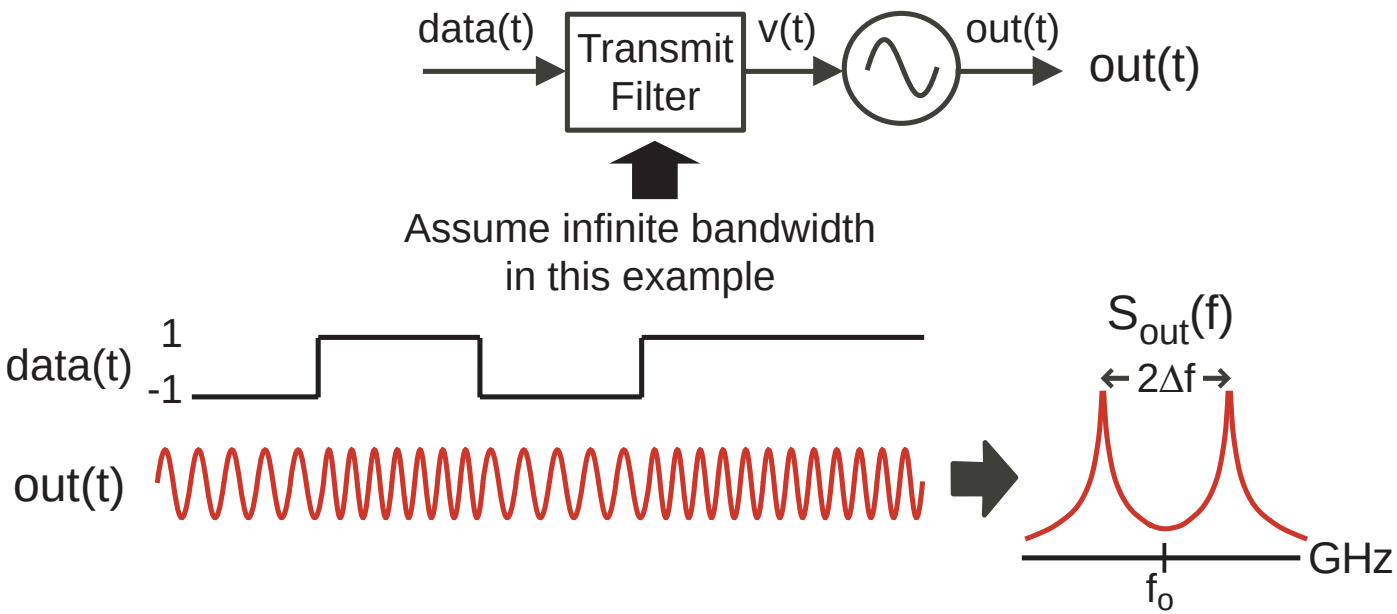
- **Frequency and phase modulation**
  - **Leveraging Fractional-N synthesizers for this task**
- **PLL filter compensation**
- **Sigma-Delta quantization noise cancellation**
- **Fast and accurate behavioral simulation**

Constant Envelope Modulation



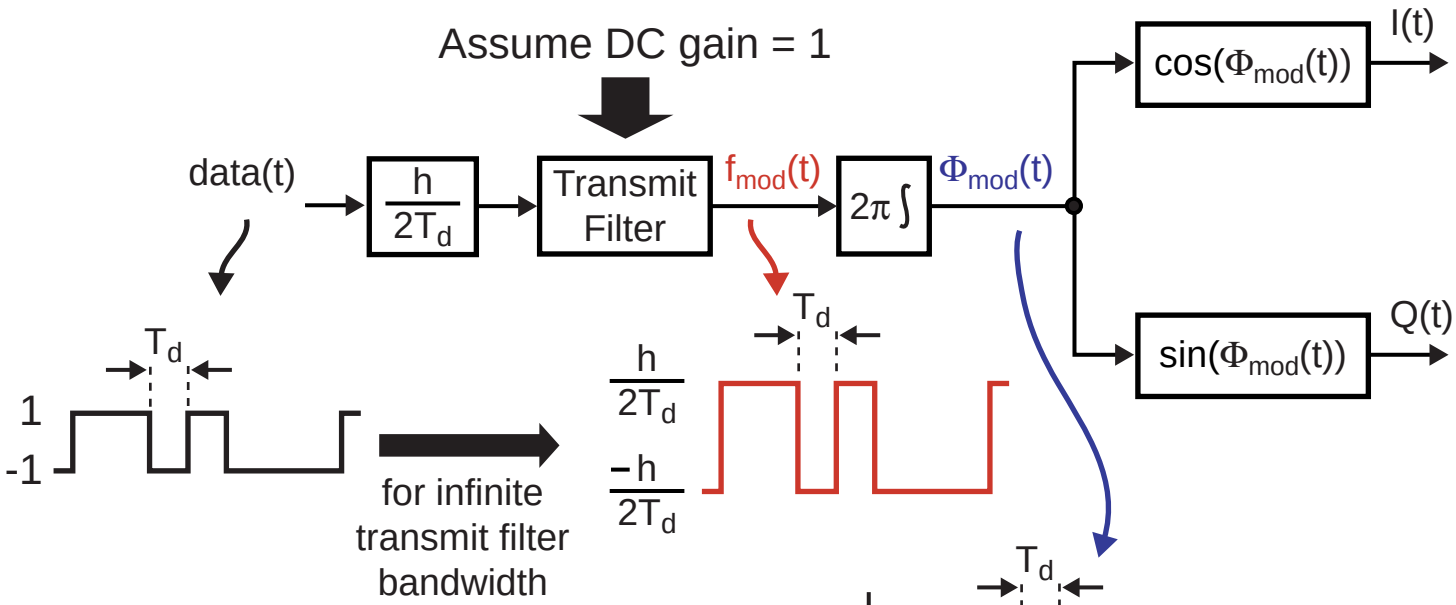
- **Popular for cell phones and cordless phones due to the reduced linearity requirements on the power amp**
  - **Allows a more efficient power amp design**
    - Transmitter power is reduced

Frequency Shift Keying



- **Sends information encoded in instantaneous frequency**
  - **Can build simple transmitters and receivers**
    - Pagers use this modulation method
- **Issue – want to obtain high spectral efficiency**
  - **Need to choose an appropriate transmit filter**
  - **Need to choose an appropriate value of Δf**

A More Detailed Model



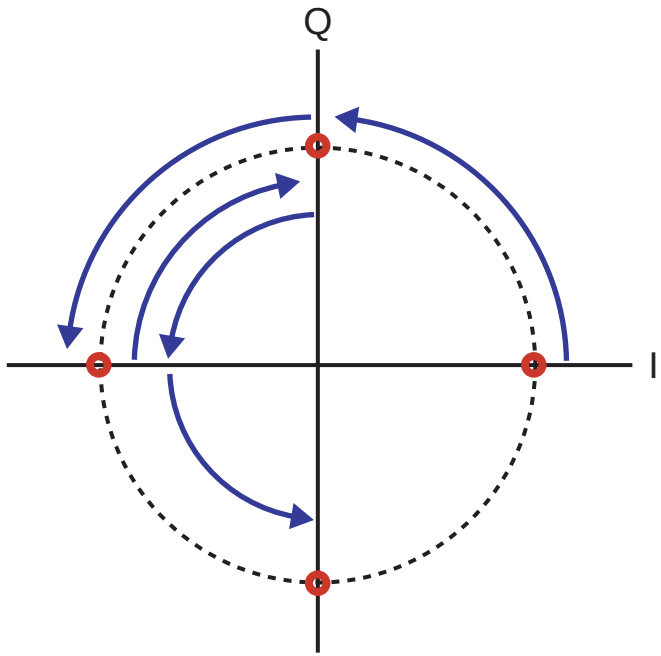
- **By inspection of figure**

$$\Delta f = \frac{h}{2T_d}$$

- **The choice of Δf is now parameterized by h and T_d**
  - **h is called the modulation index, T_d is symbol period**

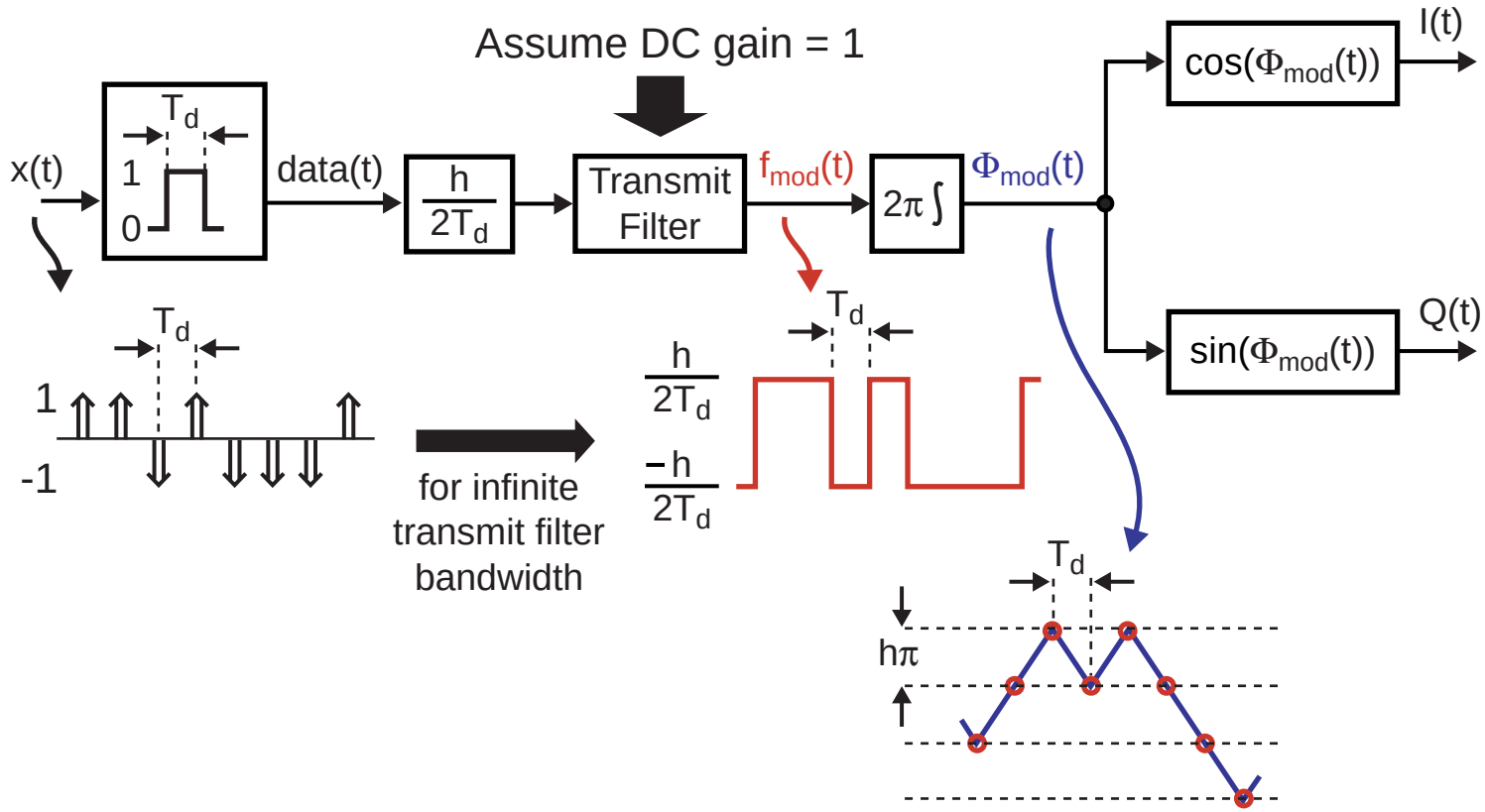
Note: phase modulation has *nonlinear* impact on I and Q!

MSK Modulation



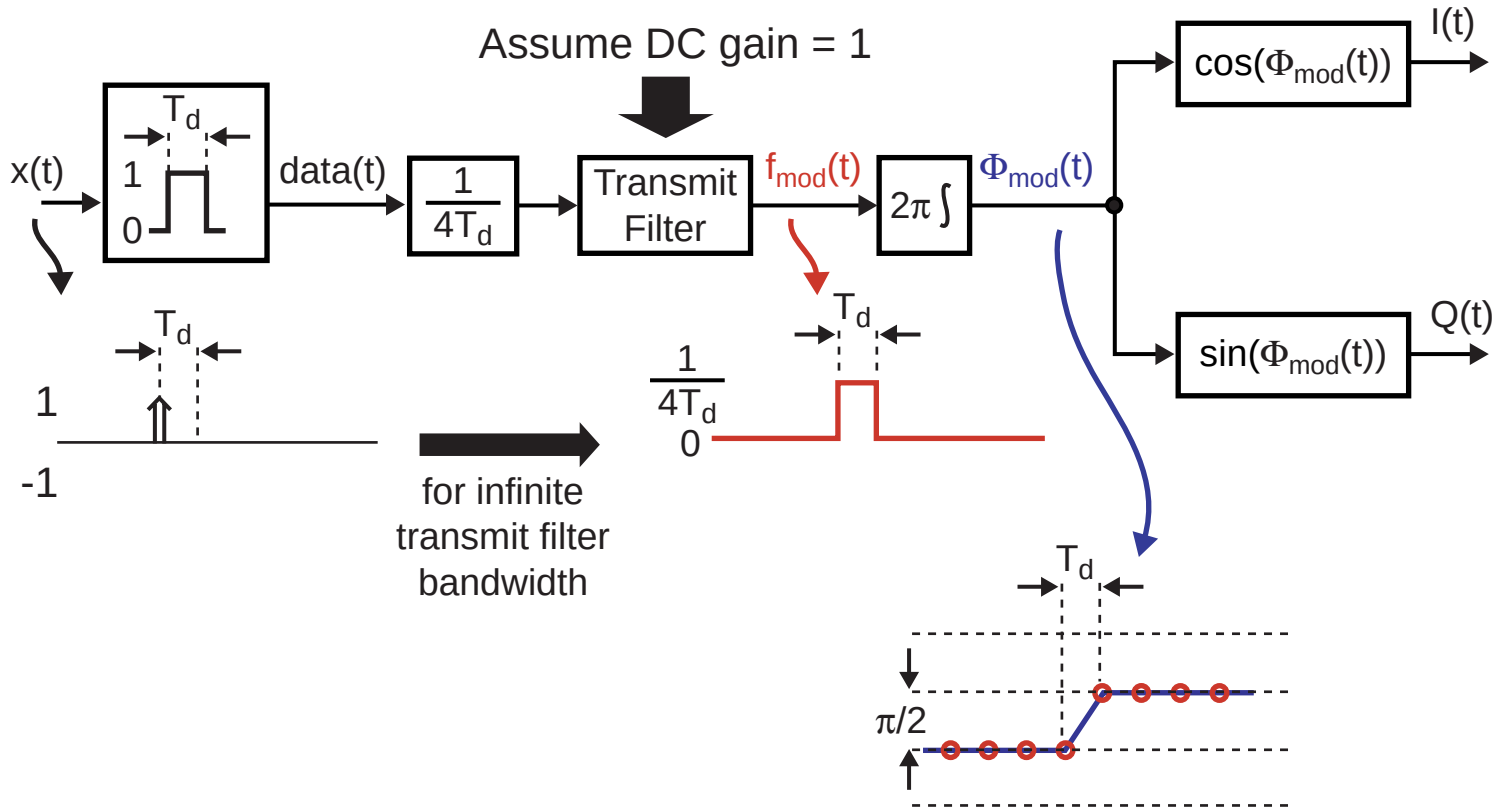
- **Choose h such that the phase rotates § 90º each symbol period**
  - **Based on previous slide, we need h = 1/2**
  - **Note: 1-bit of information per symbol period**
    - Bit rate = symbol rate

A More Convenient Model for Analysis



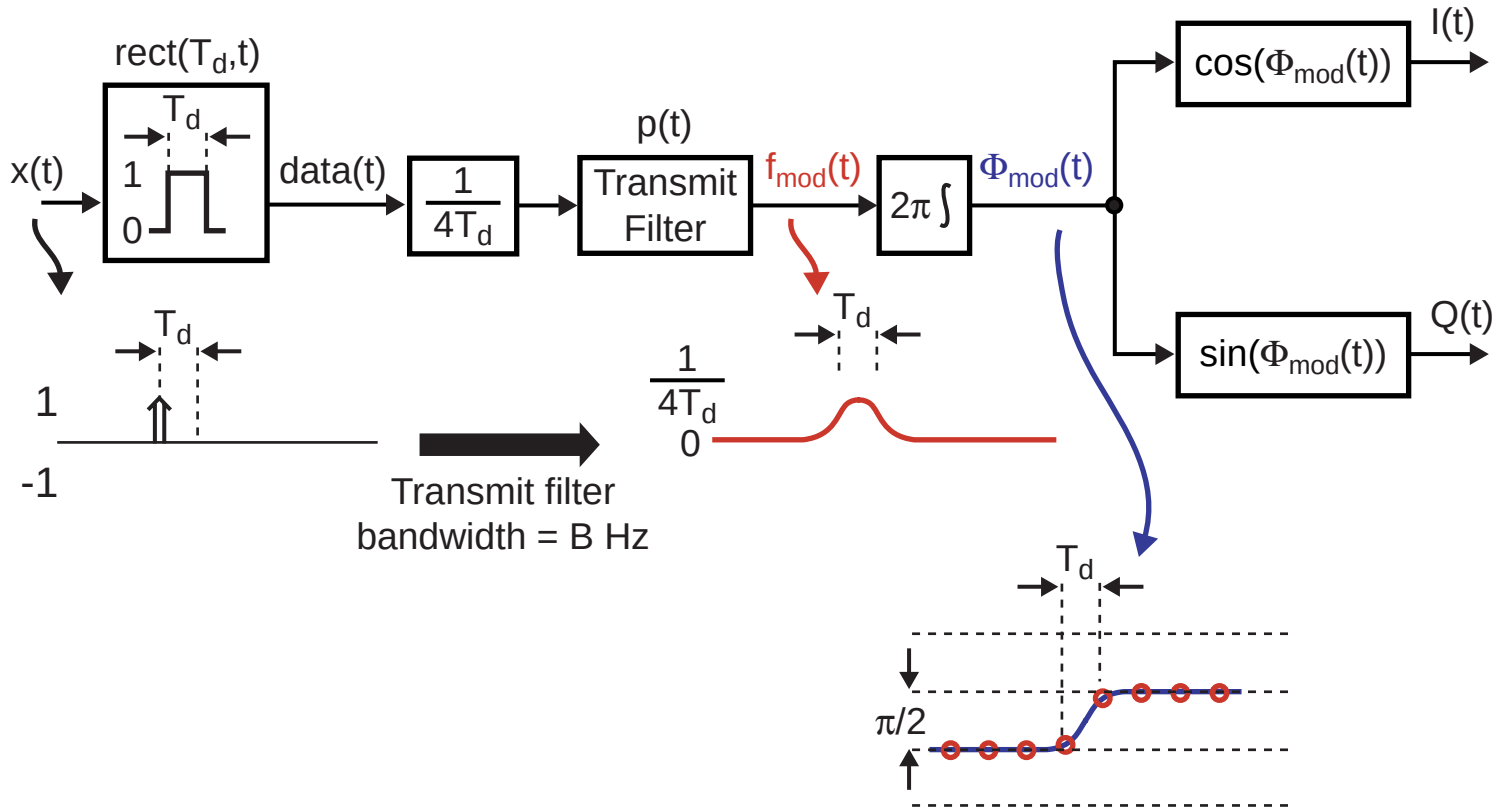
- **Same as previous model, but we represent data as impulses convolved with a rectangular pulse**
  - **Note that h = 1/2 for MSK**

Impact of Sending a Single Data Impulse



- **To achieve MSK modulation, resulting phase shift must be +/- 90º (i.e., $\pi/4$)**

Include Influence of Transmit Filter



- **For MSK modulation**

$$2\pi \int_{-\infty}^{\infty} \text{rect}(T_d, t) * \frac{1}{4T_d} p(t) = \frac{\pi}{2}$$

  - **Where * denotes convolution**

Gaussian Minimum Shift Keying

- **Definition**
  - **Minimum shift keying in which the transmit filter is chosen to have a Gaussian shape (in time and frequency) with bandwidth = B Hz**

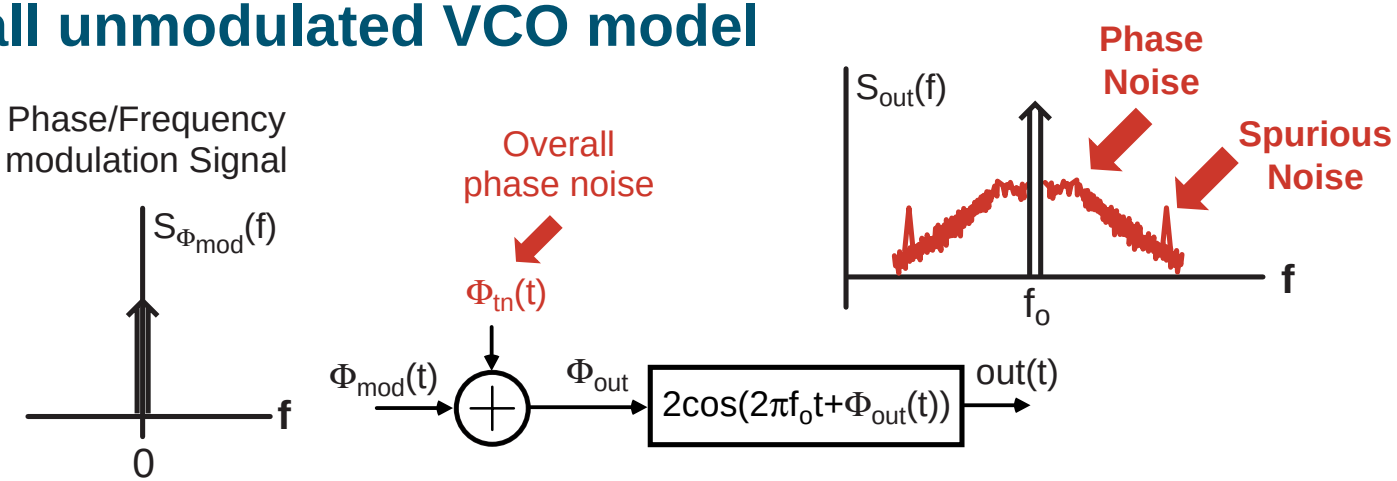
$$p(t) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{1}{2}\left(\frac{t}{\sigma}\right)^2}$$

$$\text{where : } \quad \sigma = \frac{.833 T_d}{(BT_d)2\pi}$$

- **Key parameters**
  - **Modulation index: as previously discussed**
    - h = 1/2
  - **BT_d product: ratio of transmit filter bandwidth to data rate**
    - For GSM phones: $BT_d = 0.3$

Modeling The Impact of VCO Phase Modulation

- **Recall unmodulated VCO model**



- **Relationship between sine wave output and instantaneous phase**

$$out(t) = 2\cos(2\pi f_o t + \Phi_{out}(t))$$

- **Impact of modulation**
  - **Same as examined with VCO/PLL modeling, but now we consider $\Phi_{out}(t)$ as sum of *modulation* and noise components**

$$\Phi_{out}(t) = \Phi_{mod}(t) + \Phi_{tn}(t)$$

Relationship Between Sine Wave Output and its Phase

- **Key relationship**

$$out(t) = \cos(2\pi f_o t + \Phi_{mod}(t) + \Phi_{tn}(t))$$

- **Using a familiar trigonometric identity**

$$out(t) = \cos(2\pi f_o t + \Phi_{mod}(t))\cos(\Phi_{tn}(t)) - \sin(2\pi f_o t + \Phi_{mod}(t))\sin(\Phi_{tn}(t))$$

- **Approximation given $|\Phi_{tn}(t)| << 1$**

$$out(t) \approx \cos(2\pi f_o t + \Phi_{mod}(t)) - \sin(2\pi f_o t + \Phi_{mod}(t))\Phi_{tn}(t)$$

Relationship Between Output and Phase Spectra

- **Approximation from previous slide**

$$out(t) \approx \cos(2\pi f_o t + \Phi_{mod}(t)) - \sin(2\pi f_o t + \Phi_{mod}(t))\Phi_{tn}(t)$$

- **Autocorrelation (assume modulation signal independent of noise)**

$$R\{out(t)\} = R\{\cos(2\pi f_o t + \Phi_{mod}(t))\} + R\{\sin(2\pi f_o t + \Phi_{mod}(t))\}R\{\Phi_{tn}(t)\}$$

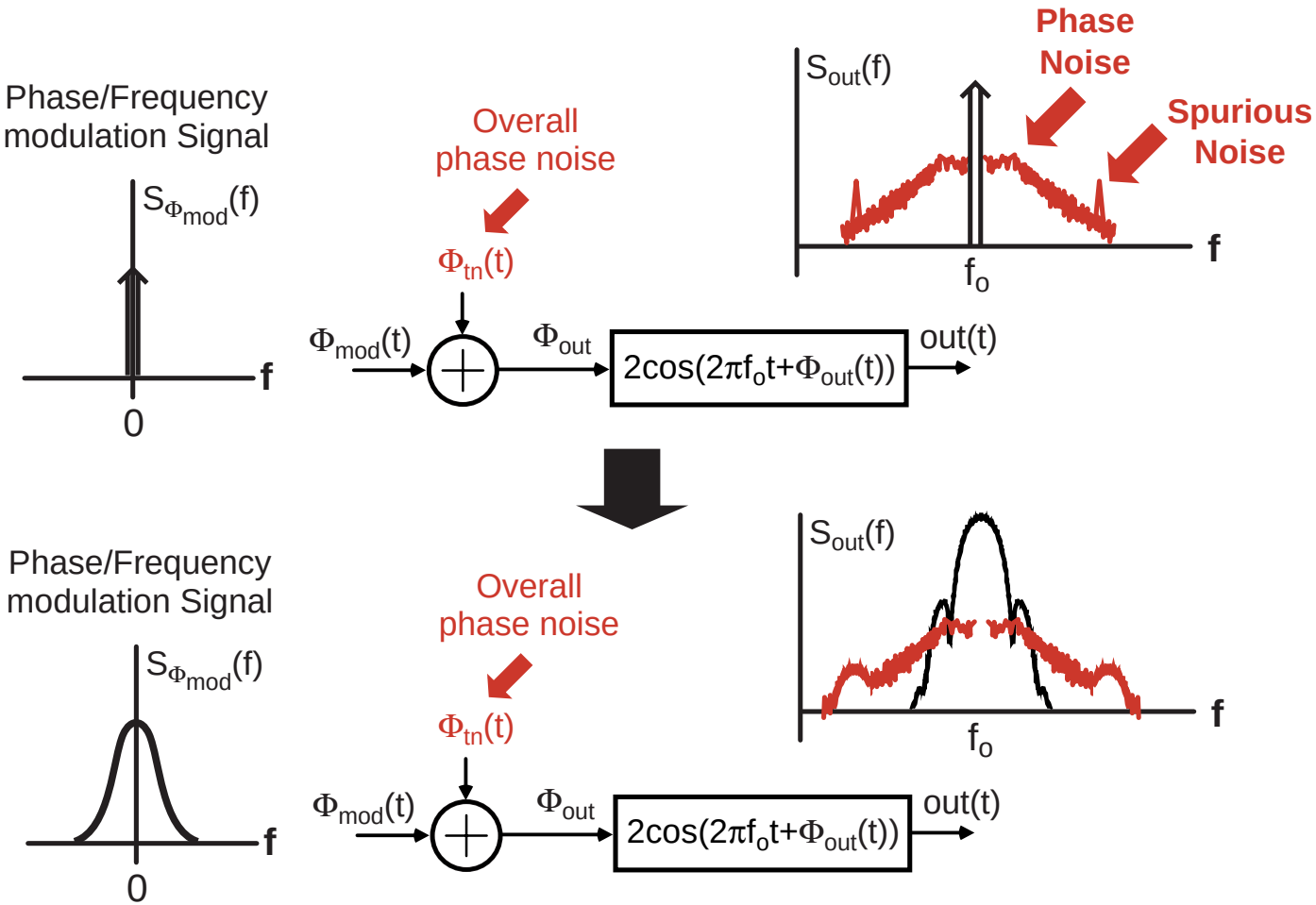
- **Output spectral density (Fourier transform of autocorrelation)**

$$S_{out}(f) = S_{out_m}(f) + S_{out_m}(f) * S_{\Phi_{tn}}(f)$$

  - **Where * represents convolution and**

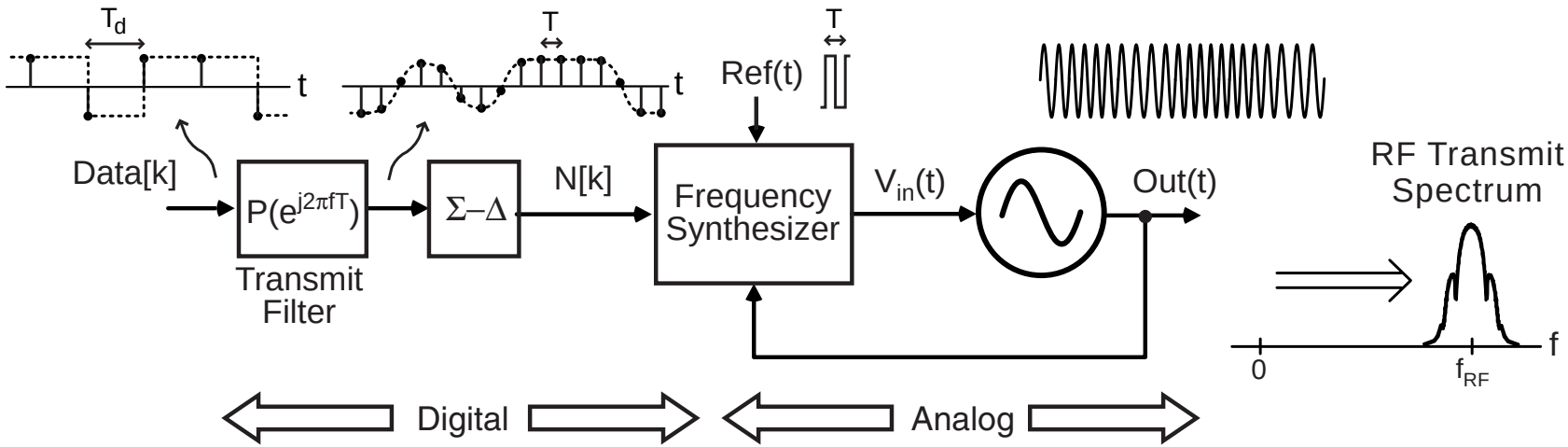
$$S_{out_m}(f) = S\{\cos(2\pi f_o t + \Phi_{mod}(t))\}, \quad S_{\Phi_{tn}}(f) = S\{\Phi_{tn}(t)\}$$

Impact of Phase Modulation on the Output Spectrum



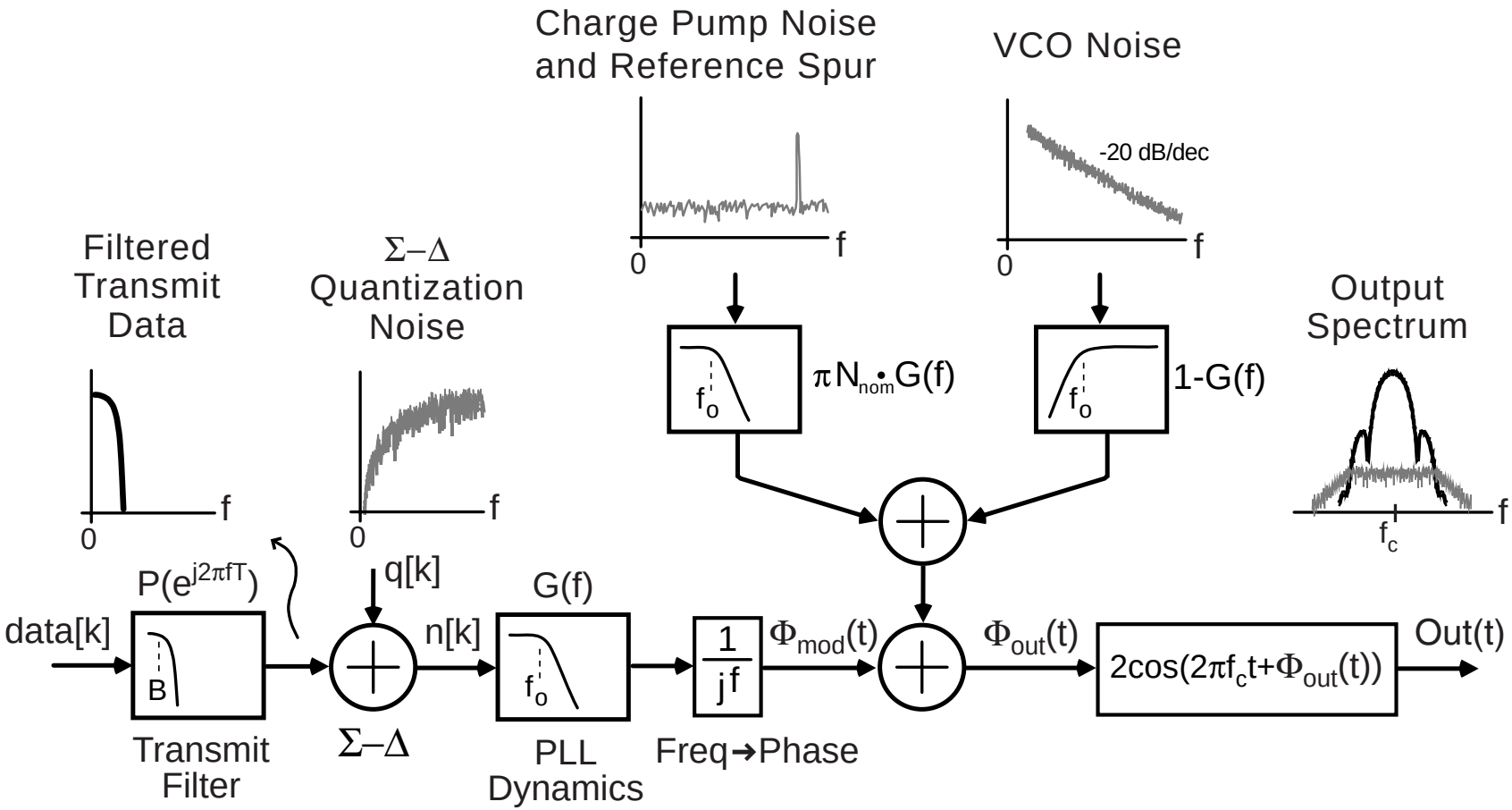
- **Spectrum of output is distorted compared to $S_{\Phi mod}(f)$**
- **Spurs converted to phase noise**

Leveraging a Fractional-N Synth for Phase Modulation



- **Provides a practical means of achieving accurate phase modulation**
- **Primarily digital structure**
  - **Analog components consist of charge pump, loop filter, and VCO**

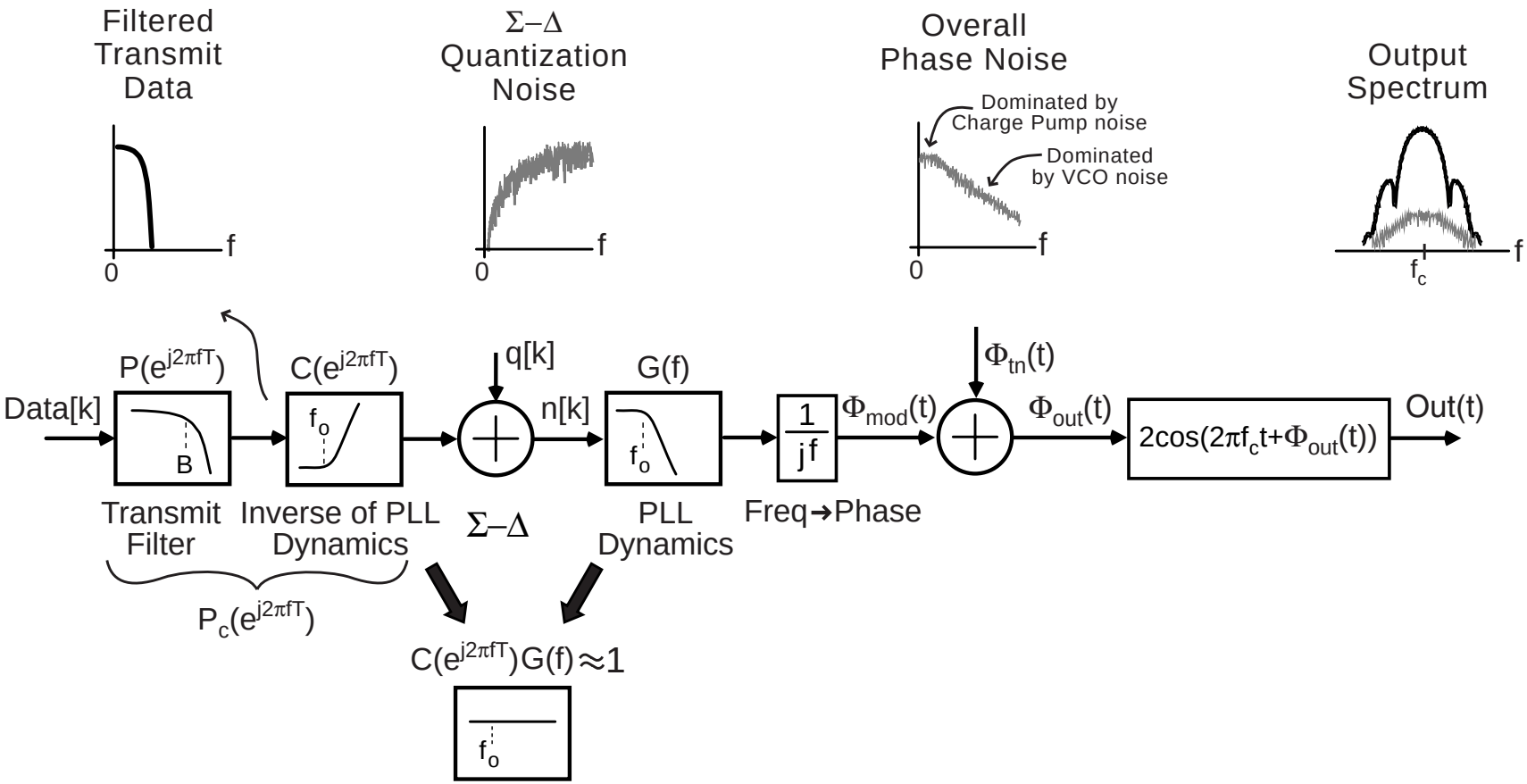
Linearized Model of Fractional-N Modulator



- **Wider modulation allows faster data rate**
  - **Increases impact of Sigma-Delta and Charge Pump noise**

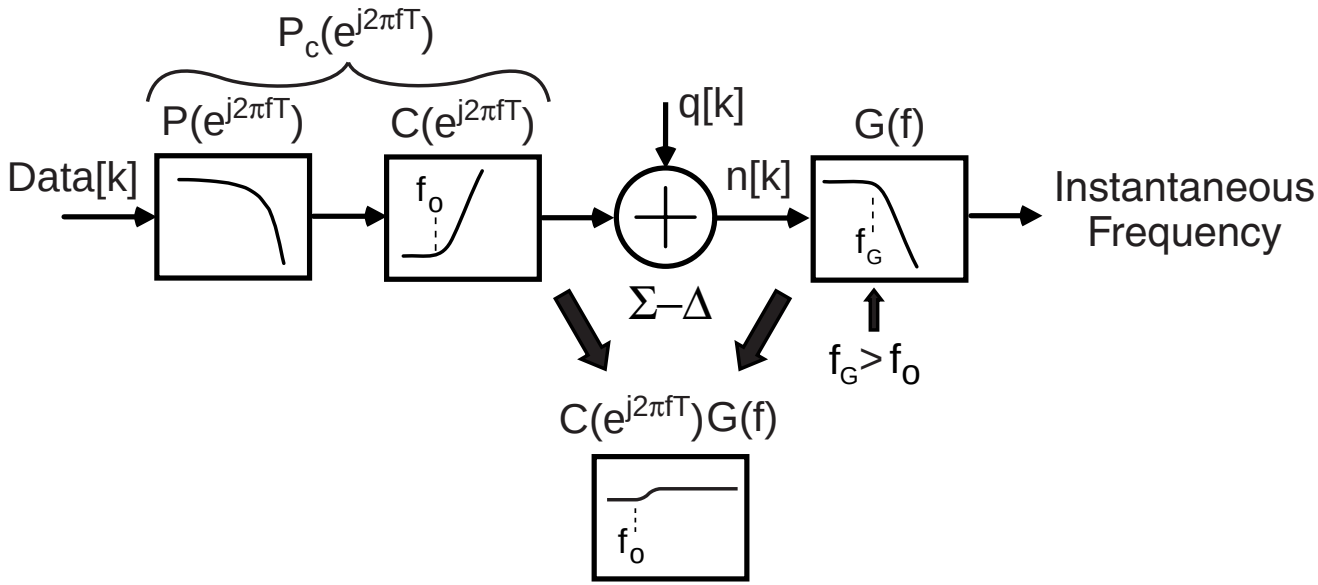
Tradeoff between data rate and noise performance

Improving the Data Rate/Noise Tradeoff



- **Compensation filter allows data rate to *exceed* PLL bandwidth**
  - **Allows higher data rates**
  - **Improves SNR and out-of-band emission performance**

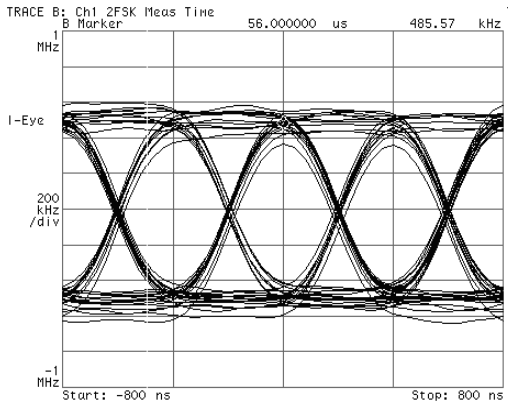
The Issue of Mismatch

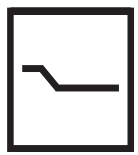


- **Mismatch between compensation filter and PLL forms parasitic pole/zero pair**
  - **Causes intersymbol interference (ISI)**

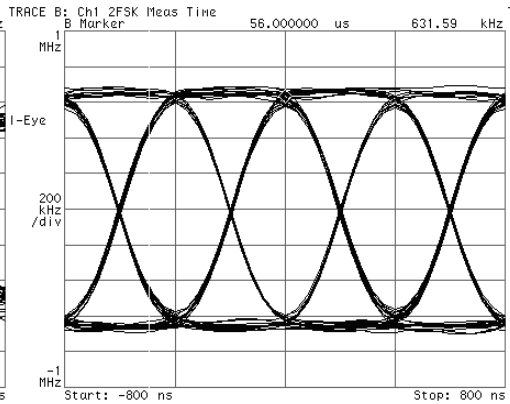
Example of ISI Due to Mismatch

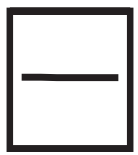
-23% Gain Error



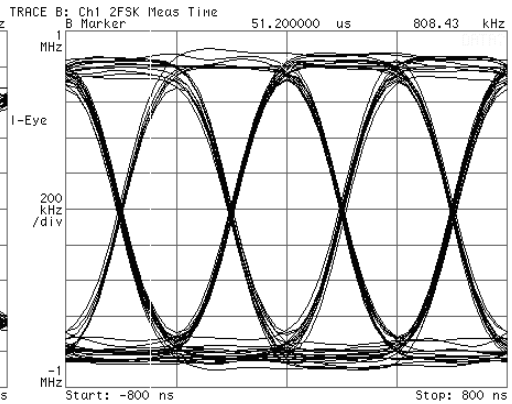


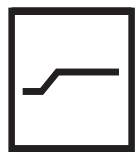
0% Gain Error

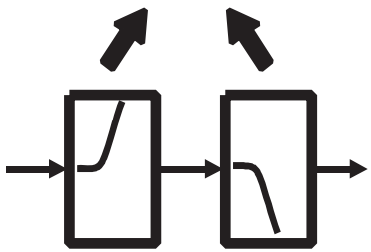




25% Gain Error



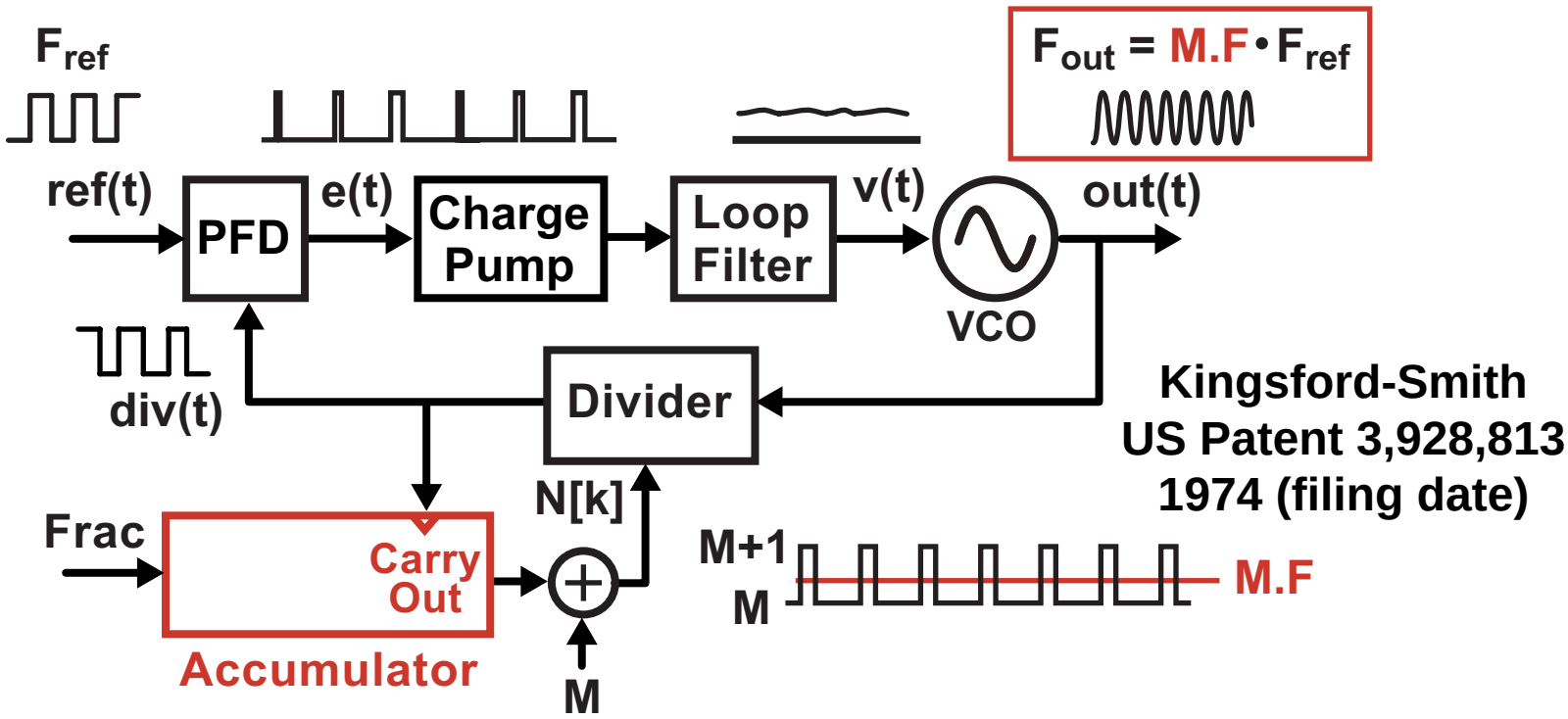




- **Frequency modulation is fairly insensitive to mismatch**
  - **Phase modulation is much more sensitive**

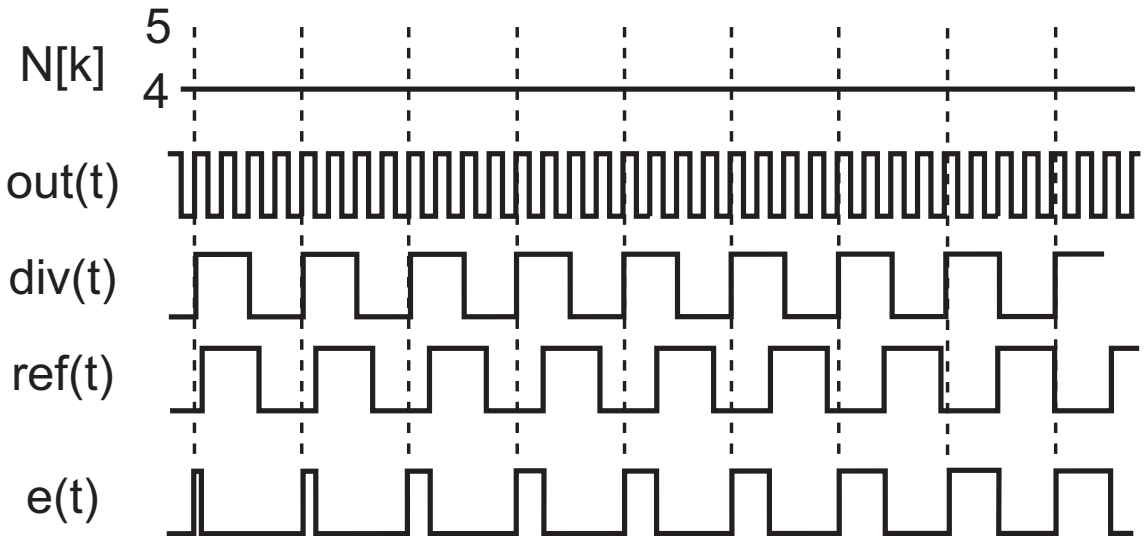
Is There An Alternate Means of Increasing Data Rate?

Classical Fractional-N Synthesizer Architecture



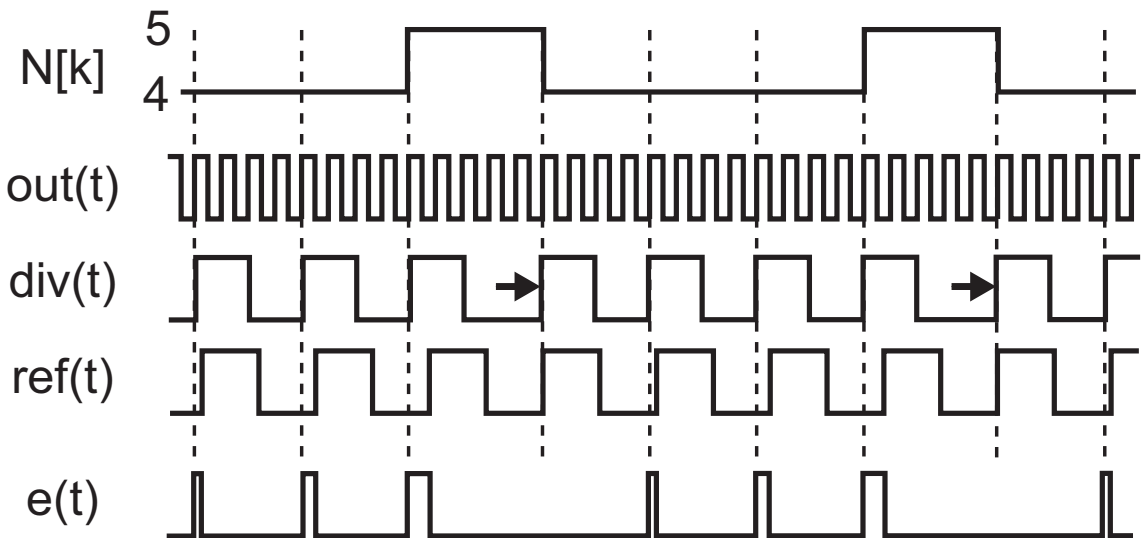
- Use an accumulator to perform dithering operation
  - Fractional input value fed into accumulator
  - Carry out bit of accumulator fed into divider

Integer-N Synthesizer Signals with $F_{out} = 4.25F_{ref}$



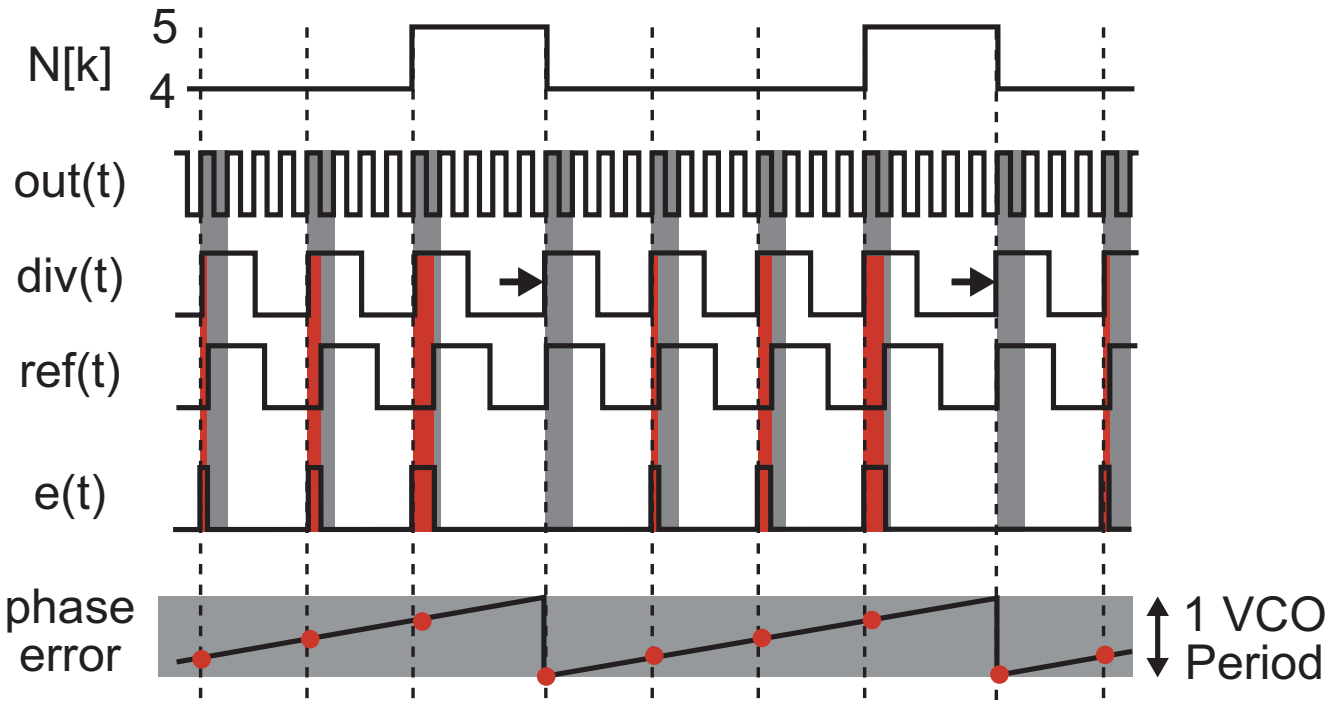
- **Constant divide value of N = 4 leads to frequency error**
  - **Error pulse widths increase as phase error accumulates**

Fractional-N Synthesizer Signals with $F_{out} = 4.25F_{ref}$



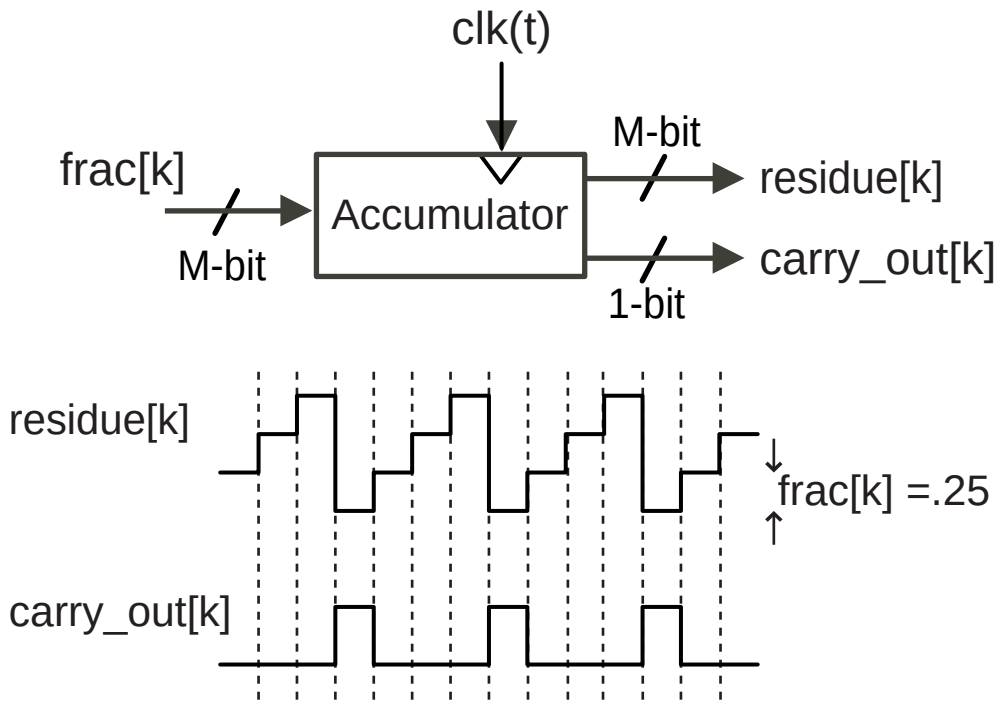
- **Dithering allows average divide value of N = 4.25**
  - **Reset phase error by periodically "swallowing" a VCO cycle**
    - Achieved by dividing by 5 every 4 reference cycles

Key Observations for Classical Fractional-N Dithering



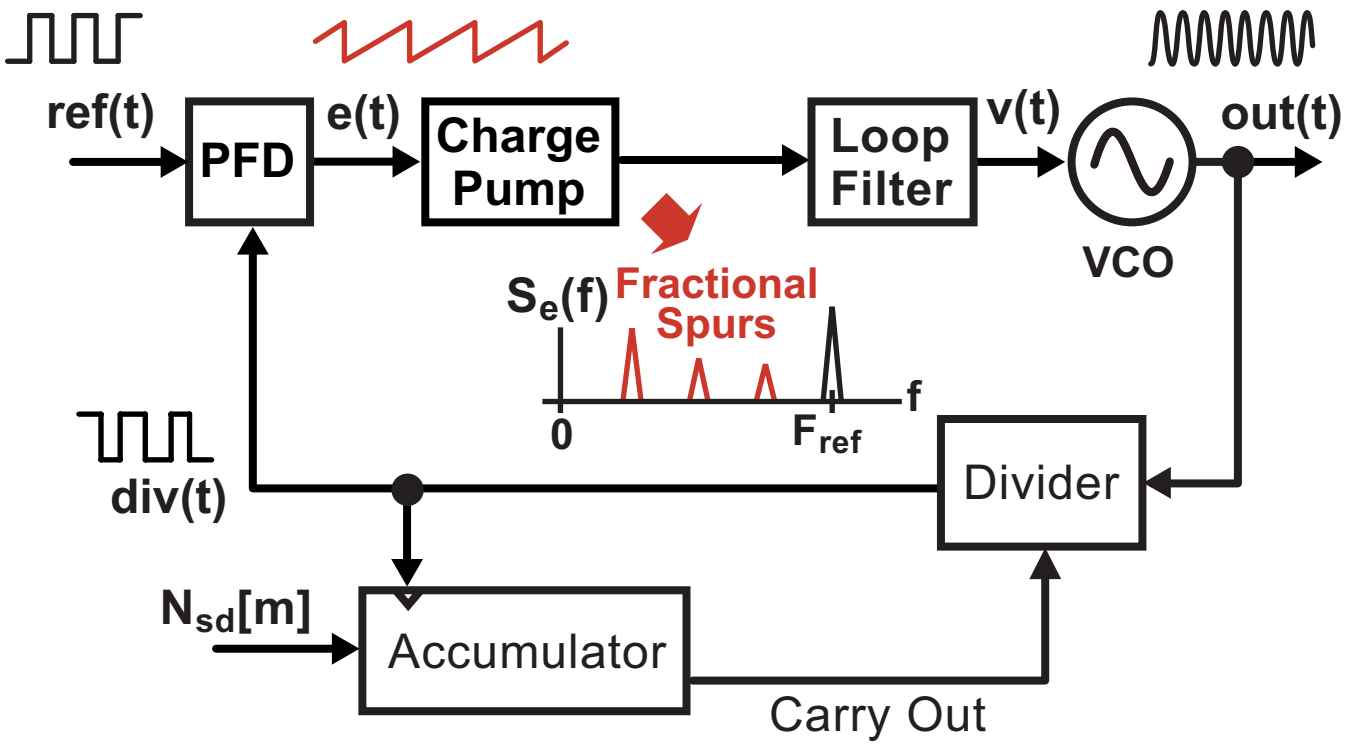
- The instantaneous phase error always remains less than one VCO cycle
- We can directly relate the phase error to the residue of the accumulator that is providing the dithering

Accumulator Operation



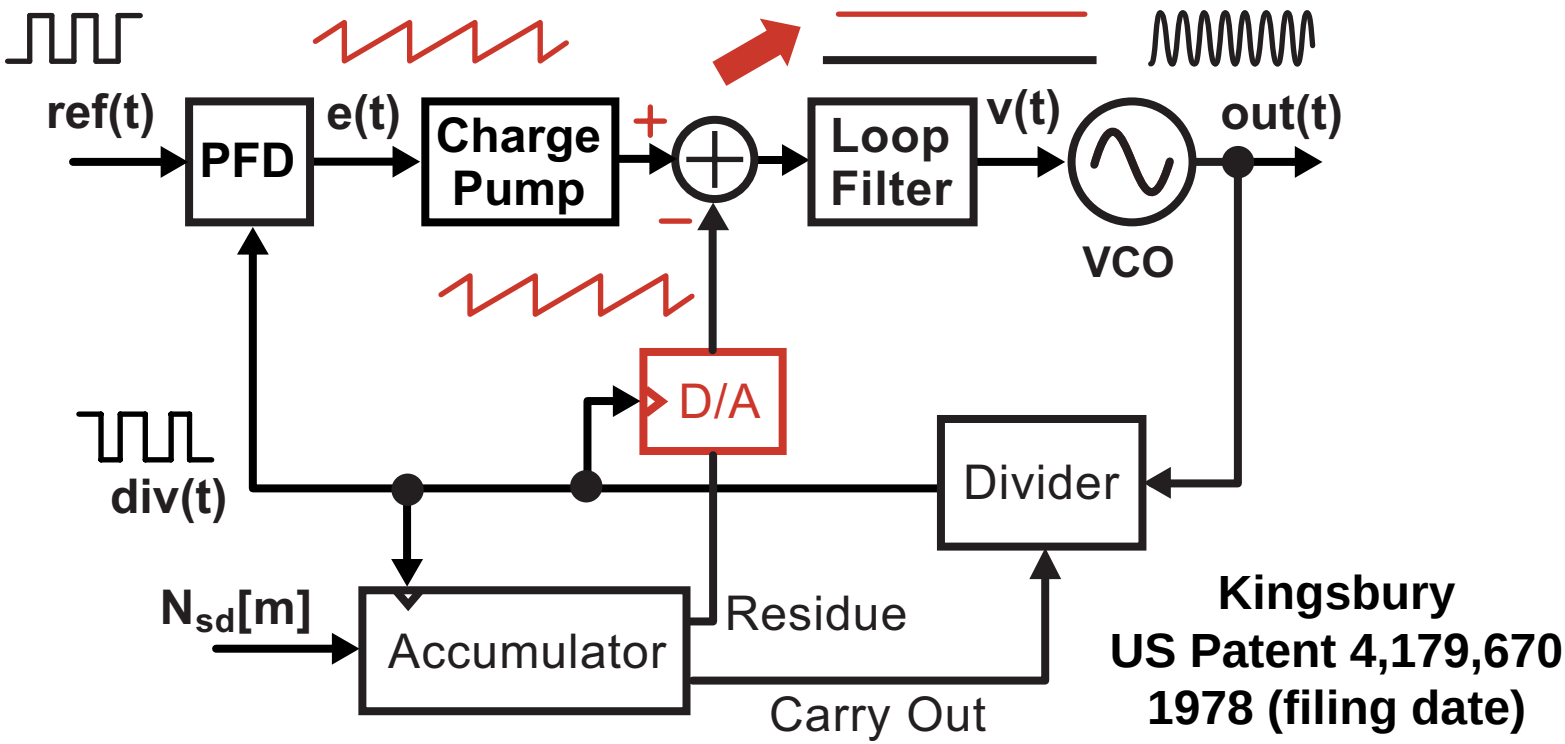
- Carry out bit is asserted when accumulator residue reaches or surpasses its full scale value
- Accumulator residue corresponds to instantaneous phase error
  - Increments by the fractional value input into the accumulator

The Issue of Spurious Tones



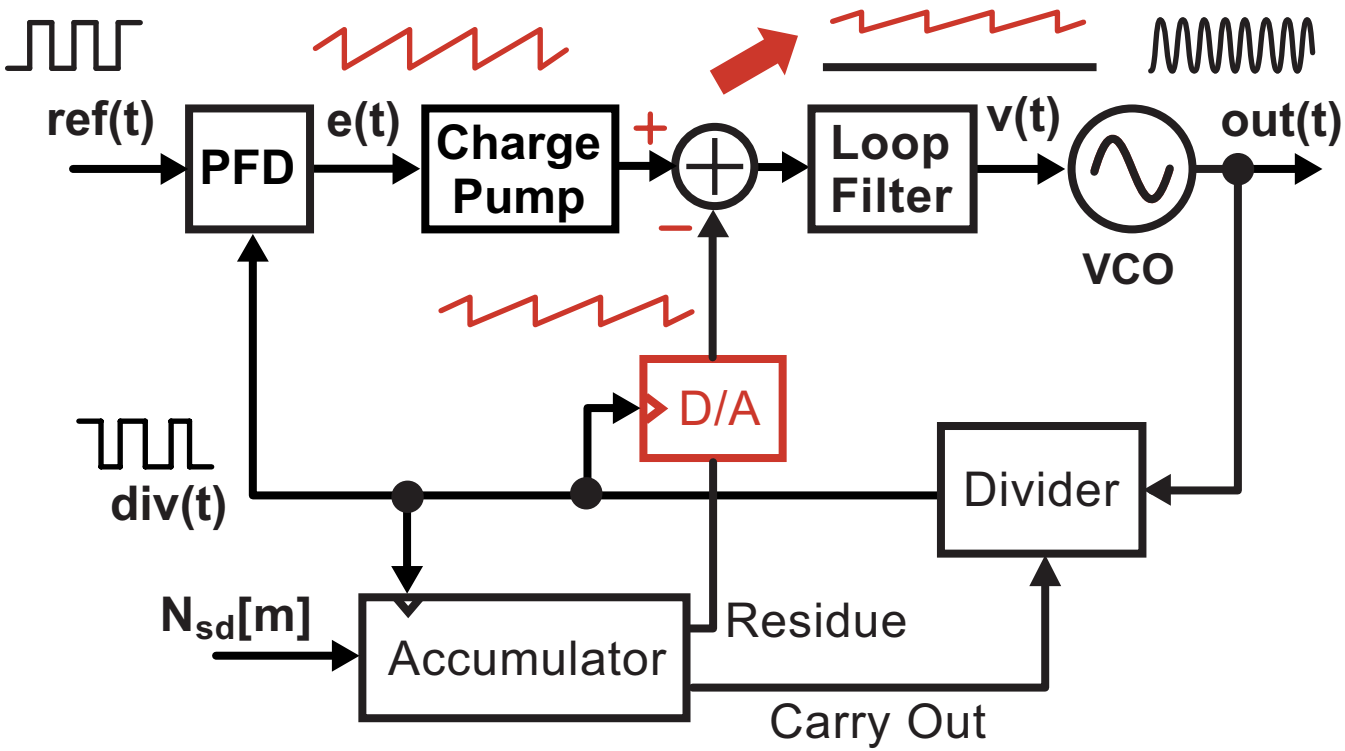
- PFD error waveform is periodic
  - Creates spurious tones in synthesizer output at lower frequencies than the reference
  - Ruins noise performance of the synthesizer

The Phase Interpolation Technique



- Leverage the fact that the phase error due to fractional technique is predicted by the instantaneous residue of the accumulator
  - Cancel out phase error based on accumulator residue

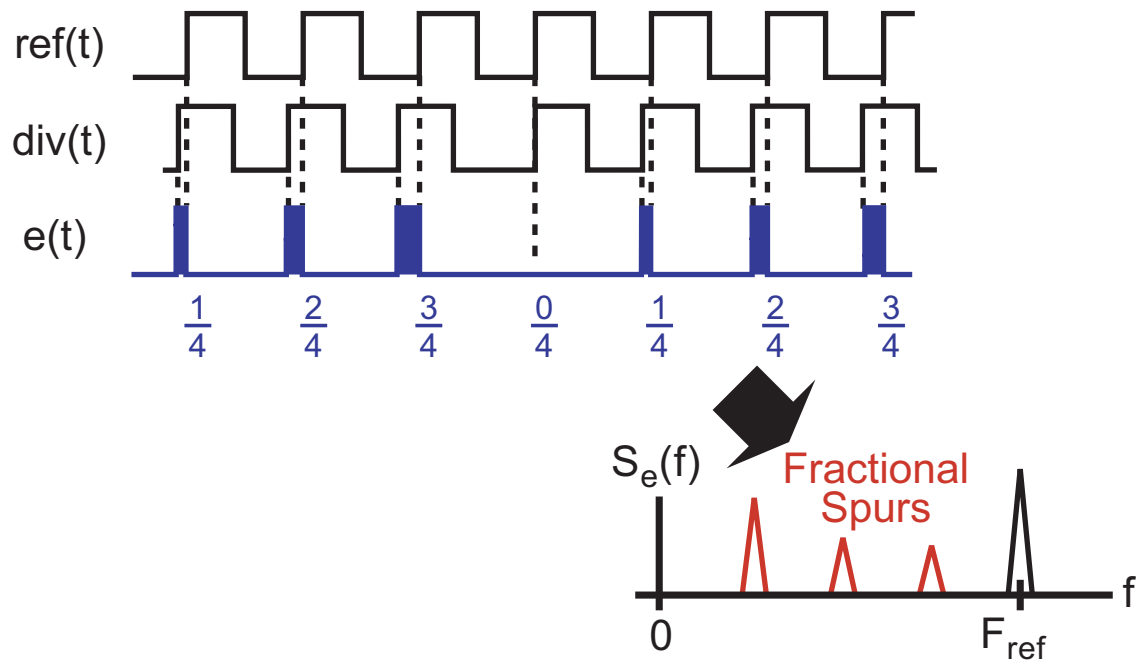
The Problem With Phase Interpolation



- Gain matching between PFD error and scaled D/A output must be extremely precise
  - Any mismatch will lead to spurious tones at PLL output

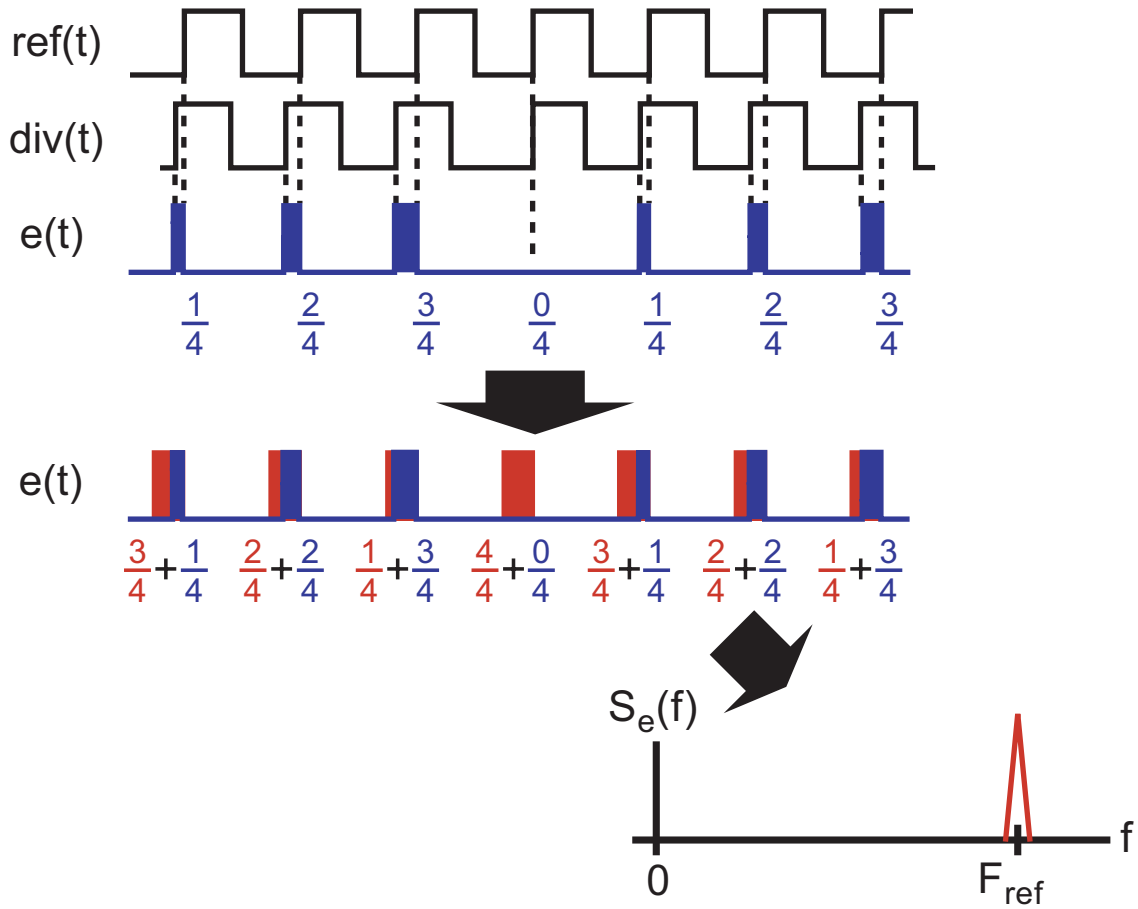
Matching issue prevented this technique from catching on

Examine Classical Fractional-N Signals



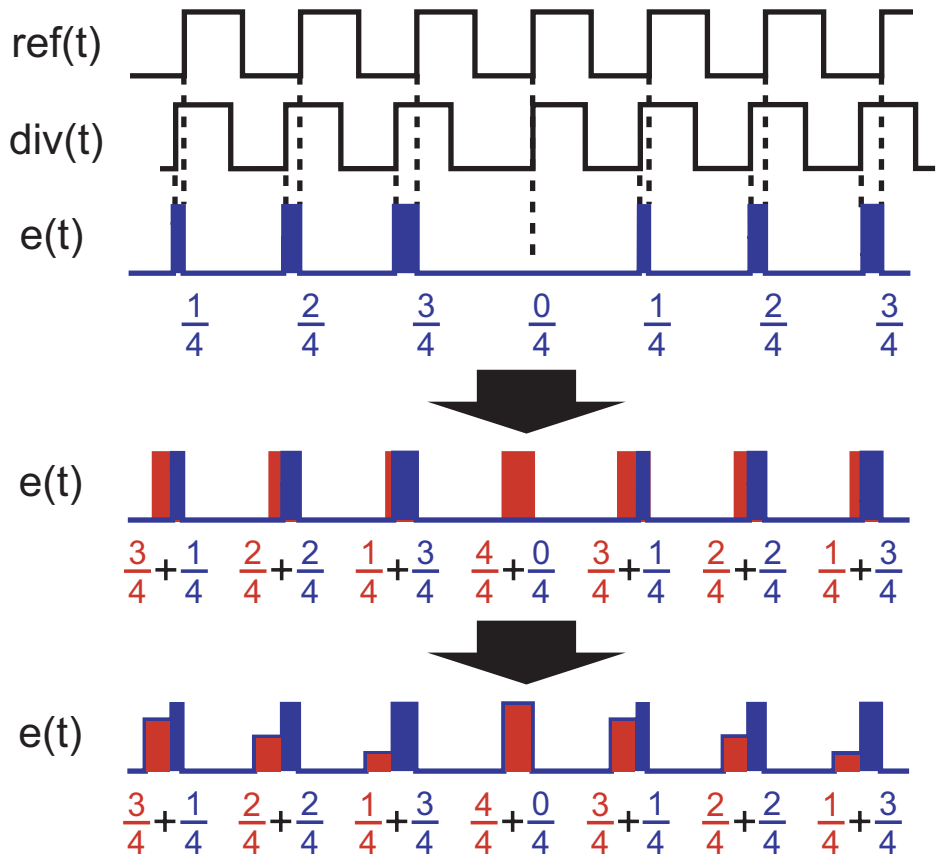
- **Goal: eliminate the fractional spurs**

Method 1: Vertical Compensation

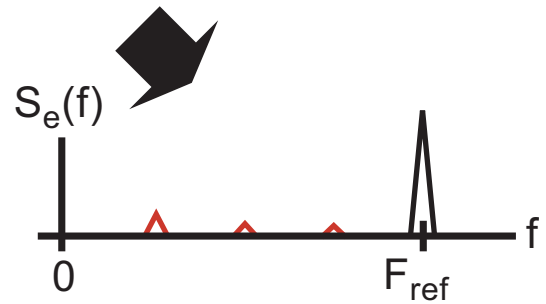


- "Fill in" pulses so that they are constant area
  - Fractional spurs are eliminated!

Method 2: Horizontal Compensation

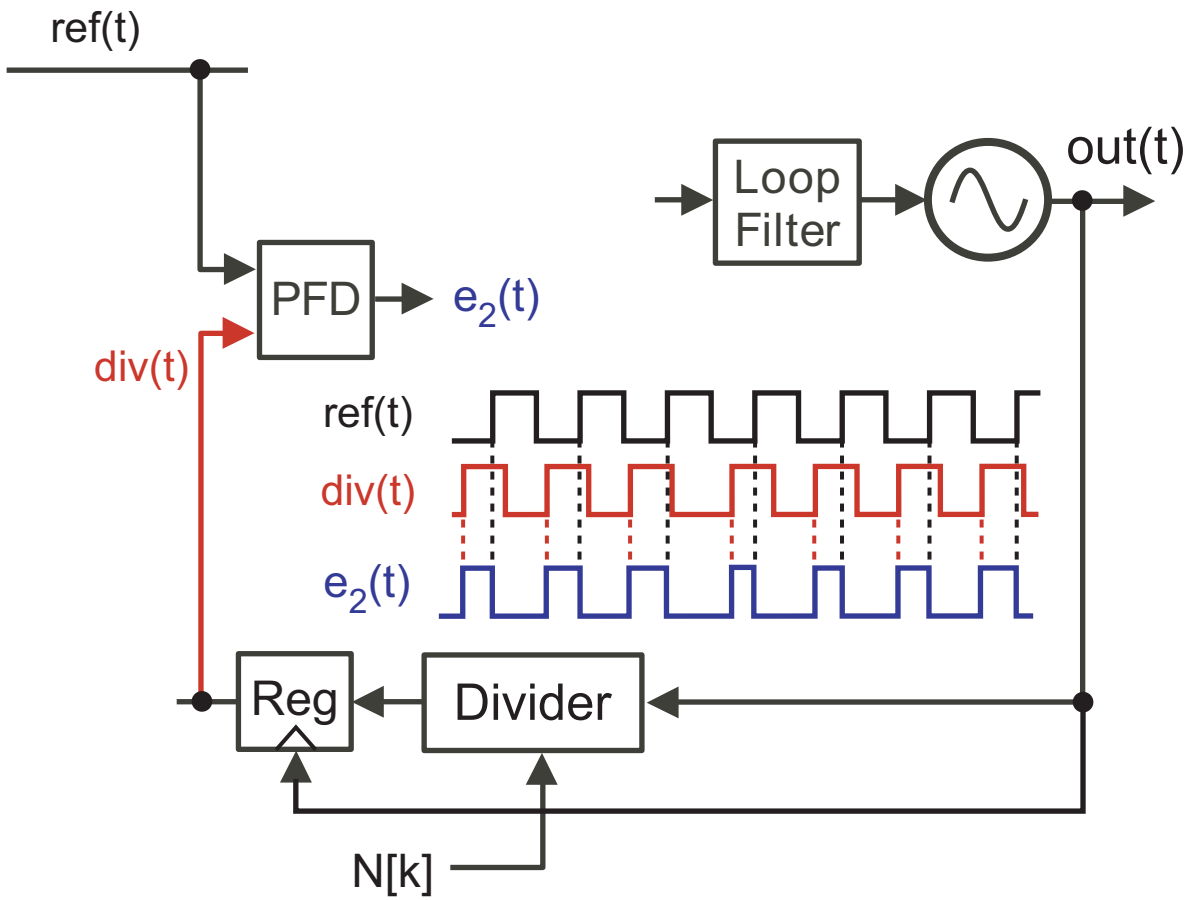


- **Use constant width pulses of varying height to achieve constant area pulses**
  - **Largely eliminates fractional spurs**

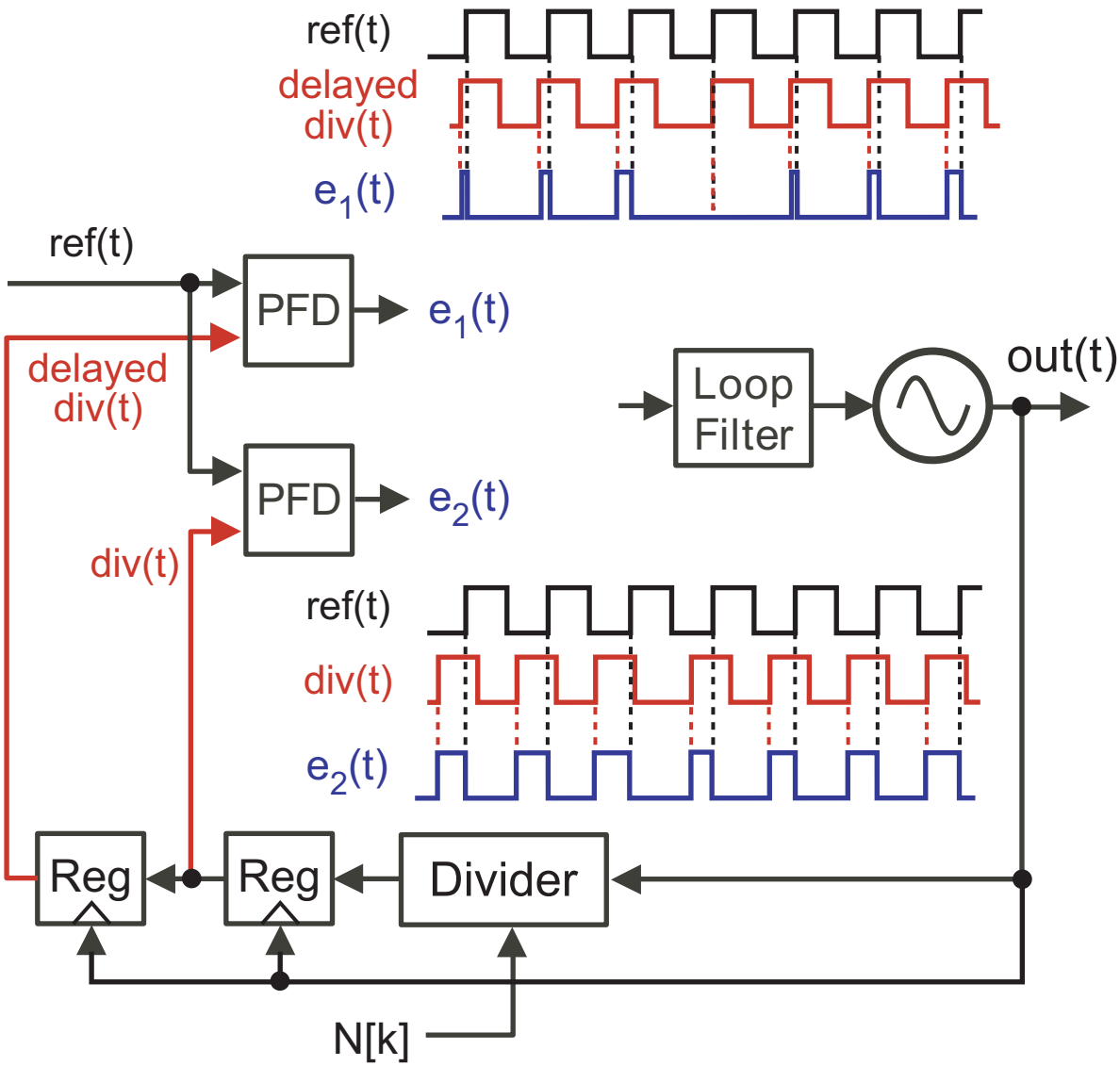


Implementation of Horizontal Cancellation

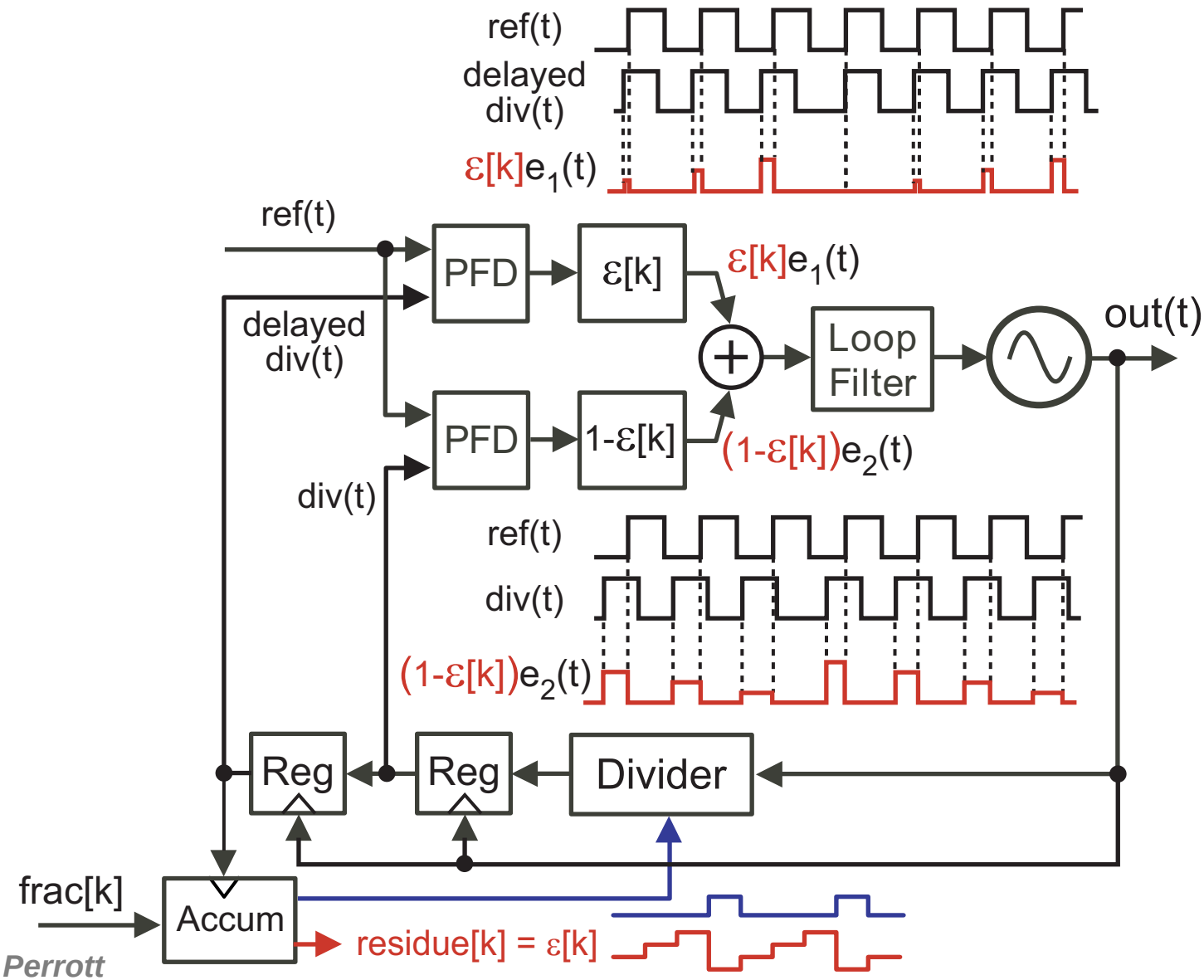
- We begin with the basic fractional-N structure



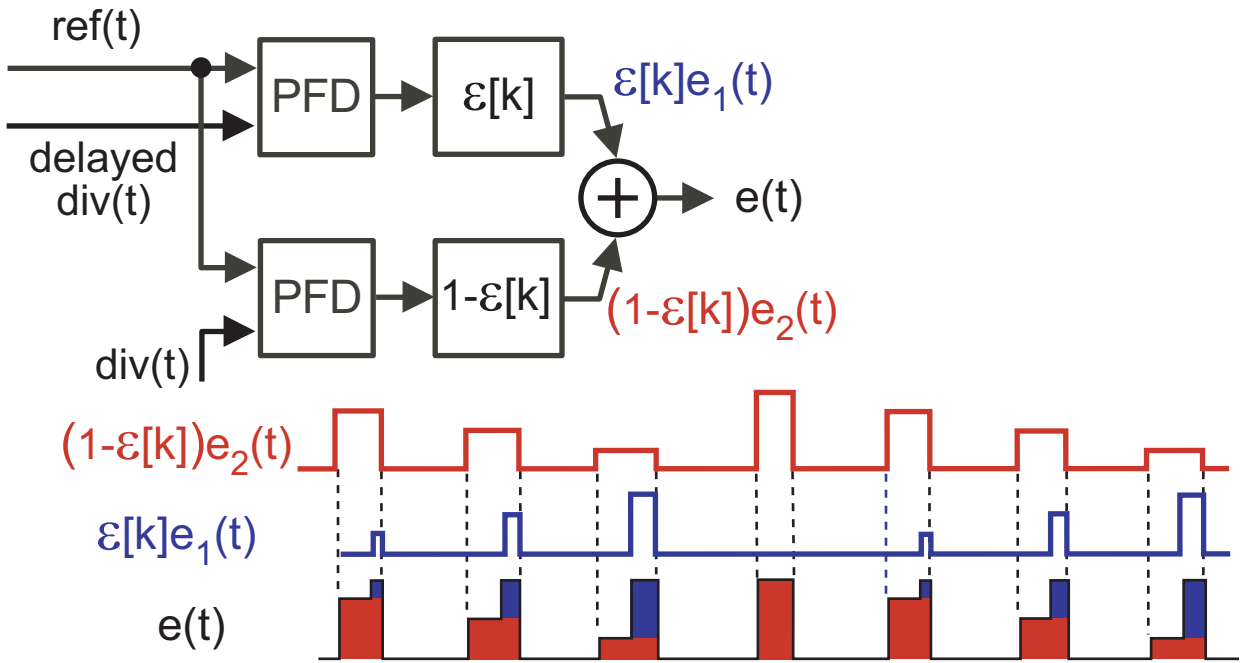
Add a Second PFD with Delayed Divider Signal



Scale Error Pulses According to Accumulator Residue



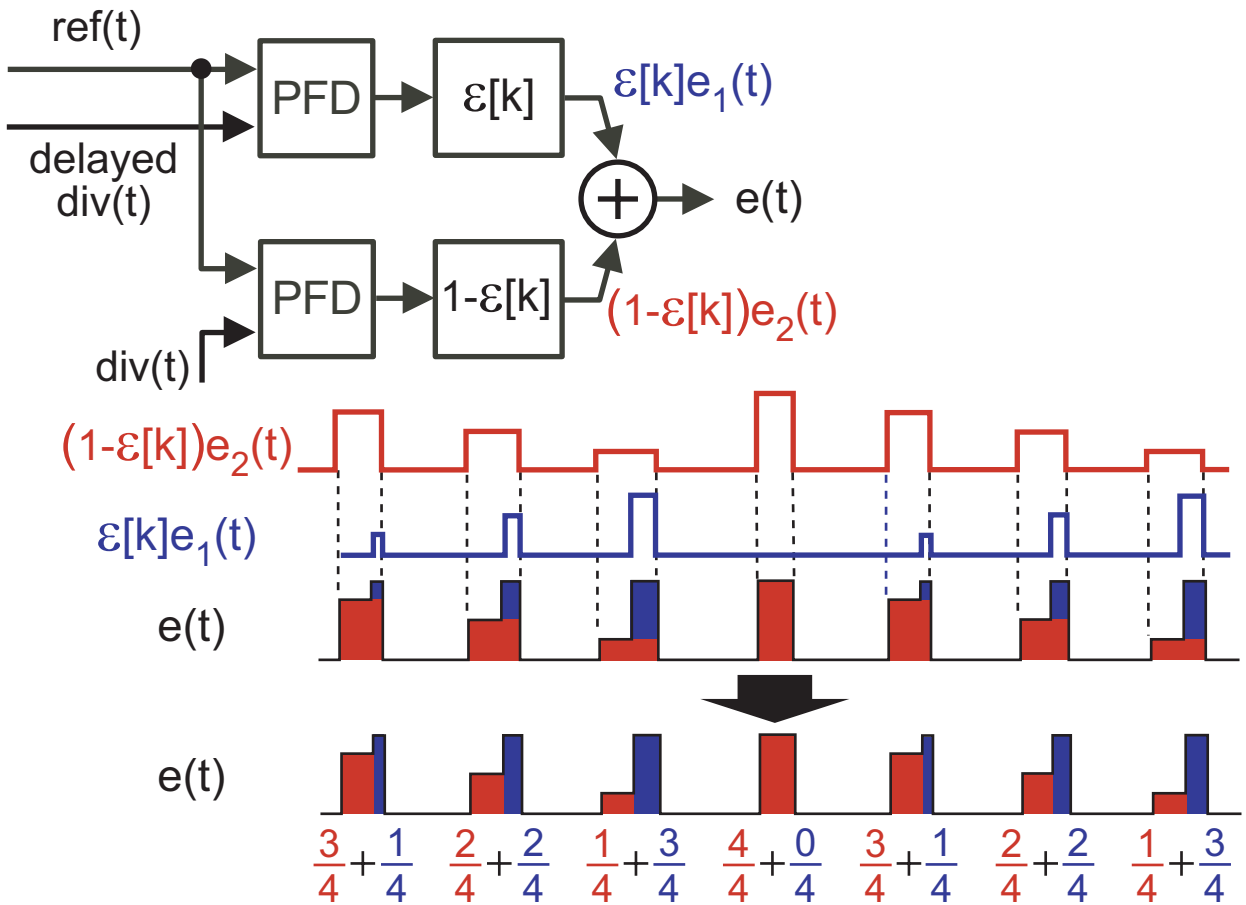
A Closer Look at Adding the Scaled Error Pulses



- Goal – keep area constant for each pulse
  - It's easier to see this from a slightly different viewpoint

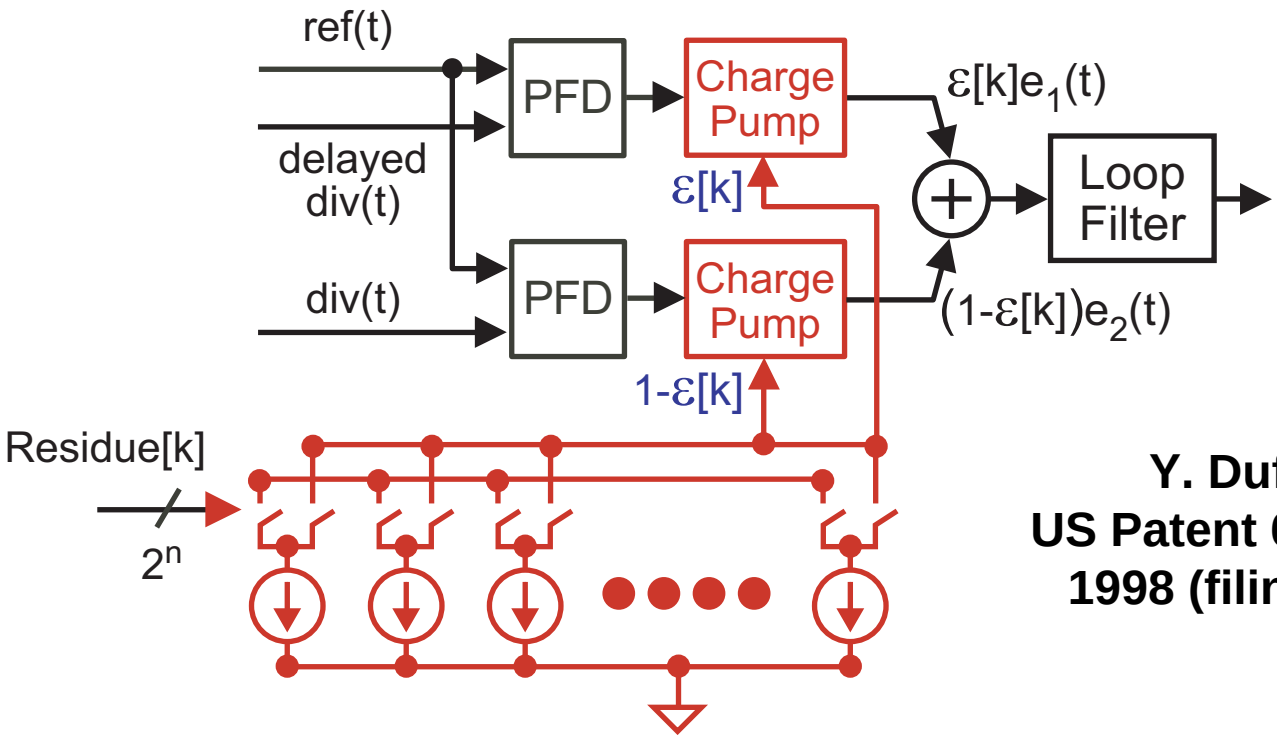
Alternate Viewpoint

- **The sum of scaled pulses can now be viewed as horizontal cancellation**



Implementation of Pulse Scaling Operation

- **Direct output of a differential current DAC into two charge pumps**



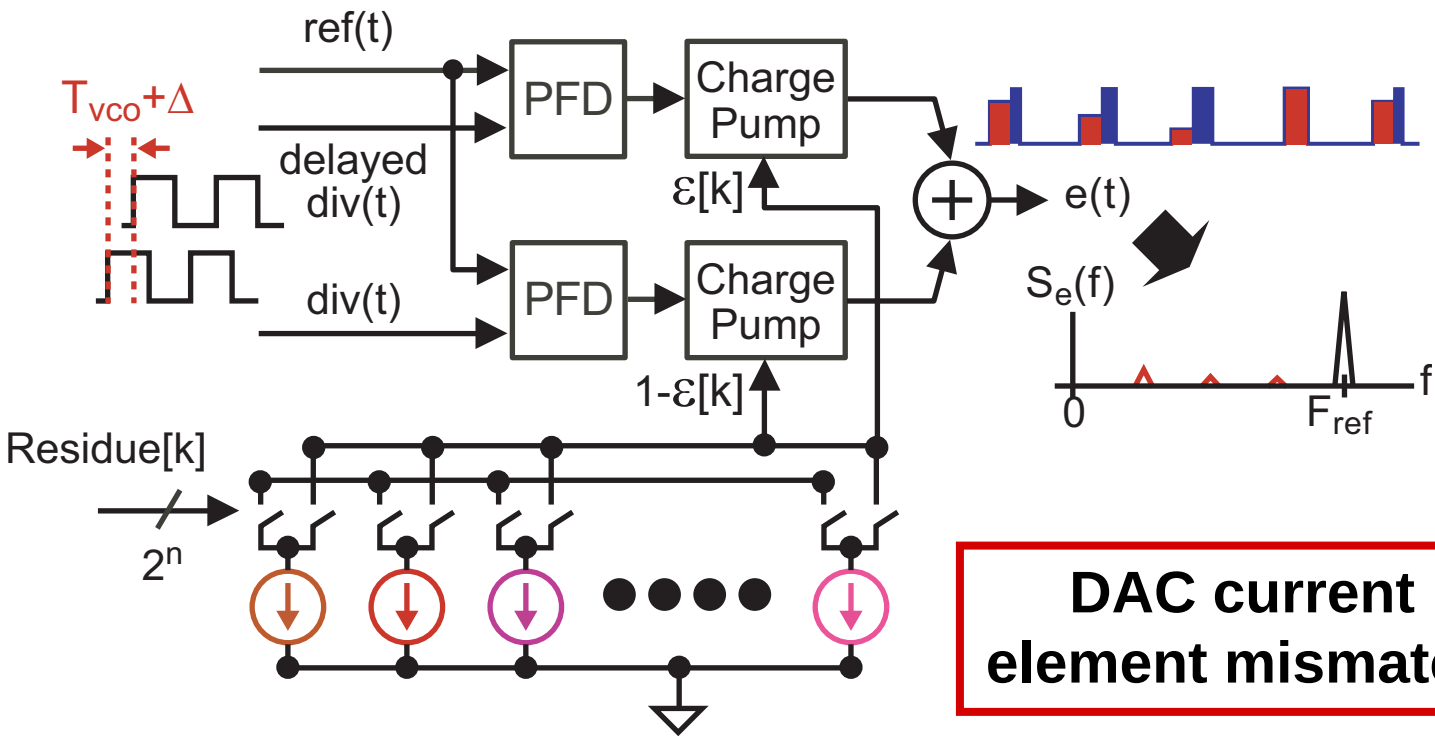
Y. Dufour
US Patent 6,130,561
1998 (filing date)

- **Issue: practical non-idealities kill performance**

Primary Non-idealities of Concern

Delay mismatch

Incomplete Fractional Spur Suppression

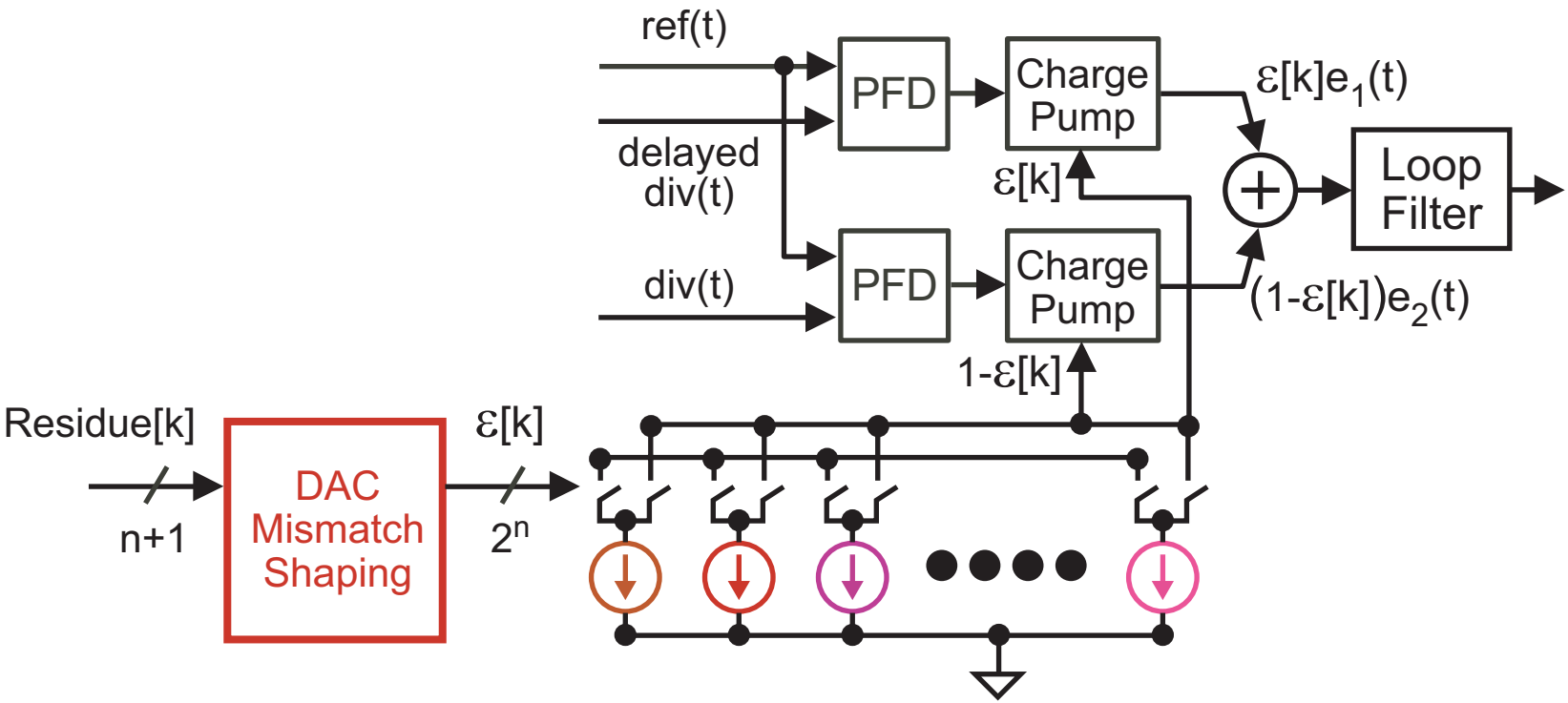


DAC current element mismatch

Proposed approach: dramatically reduce impact of these non-idealities using mixed-signal processing techniques

Eliminate Impact of DAC Current Element Mismatch

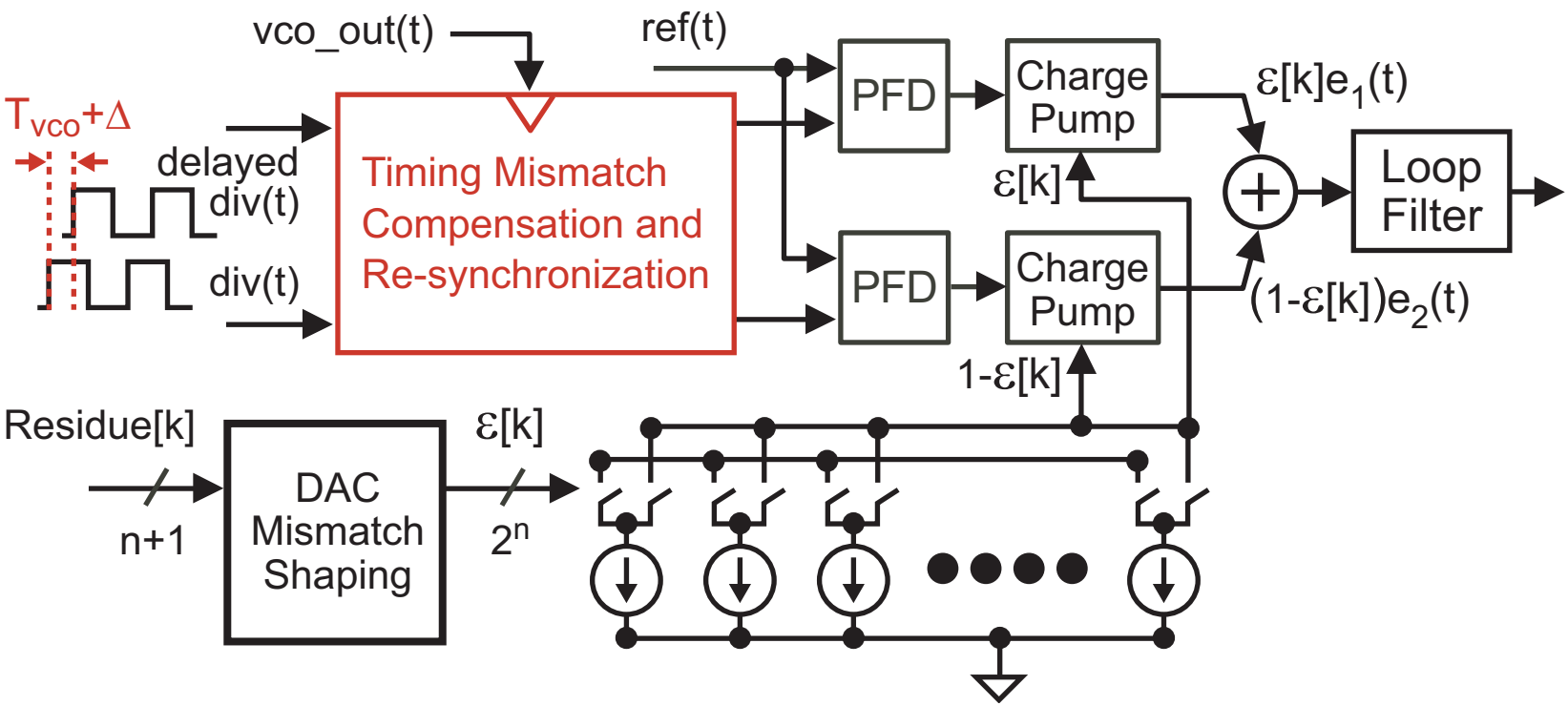
- Apply standard DAC noise shaping techniques to shape mismatch noise to high frequencies
  - See Baird and Fiez, TCAS II, Dec 1995



- Allows up to 5% mismatch between unit elements without degrading our desired performance targets

Eliminate Impact of Timing Mismatch

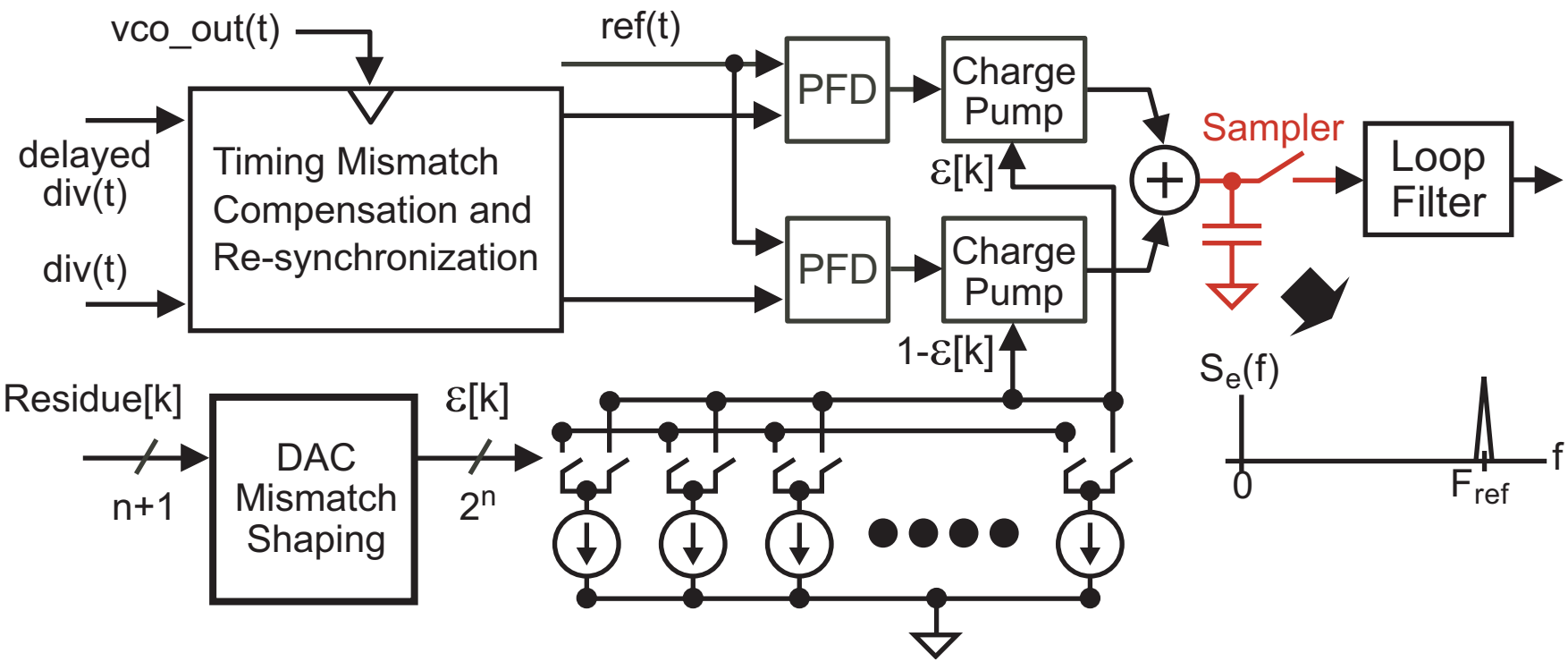
- **Swap paths between divider outputs in a pseudo-random fashion**
  - **Need to also swap ε[k] and 1-ε[k] sequence**



- **Allows up to 5 ps mismatch without degrading our desired performance targets**

Improve Horizontal Cancellation Performance

- **Sampling circuit accumulates error pulses before passing their information to the loop filter**
  - **A common analog trick used for decades**



- **Eliminates issue of having non-square error pulse shapes**

Application:
A 1 MHz Bandwidth Fractional-N Frequency Synthesizer Implementation

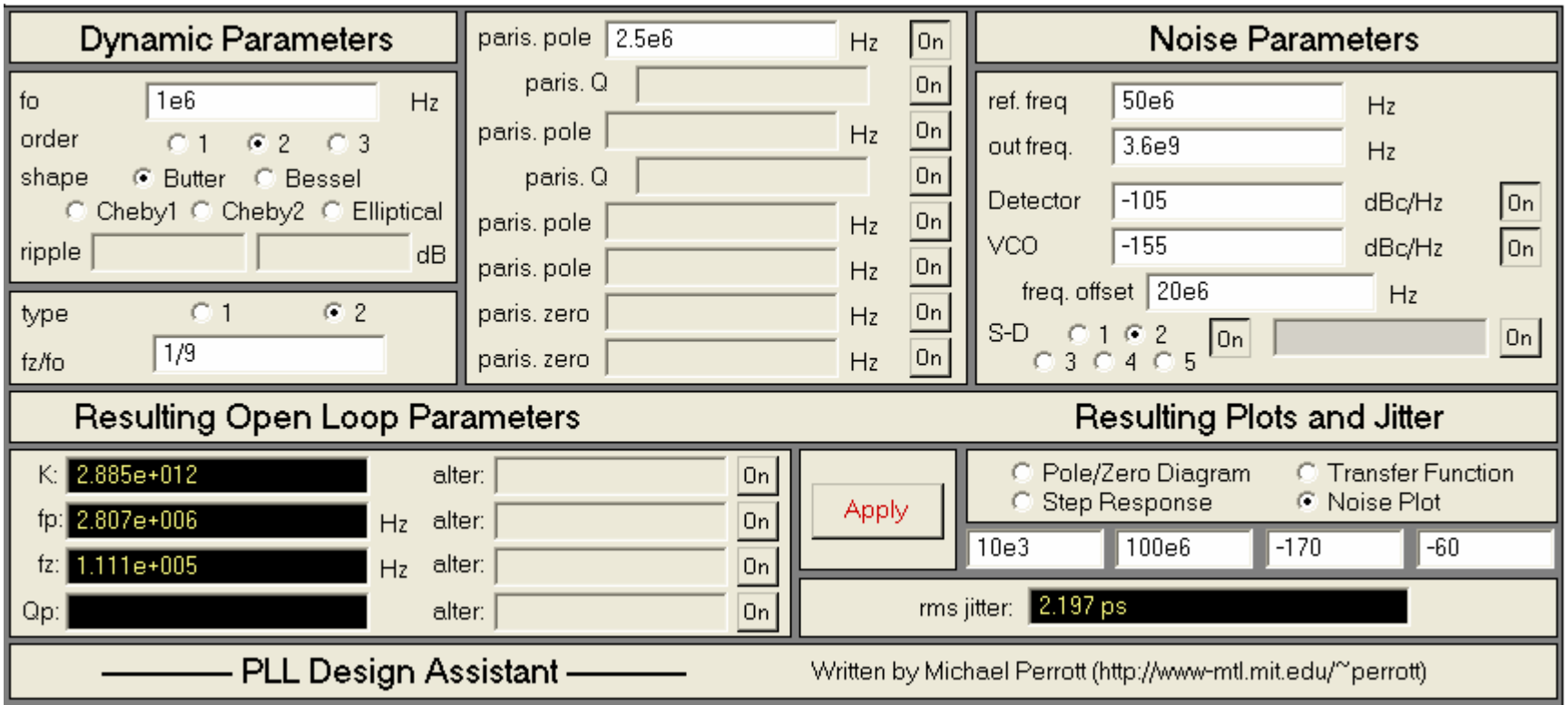
Design Goals

- **Output frequency: 3.6 GHz**
  - **Allows dual-band output (1.8 GHz and 900 MHz)**
- **Reference frequency: 50 MHz**
  - **Allows low cost crystal reference**
- **Bandwidth: 1 MHz**
  - **Allows fast settling time and ~1 Mbit/s modulation rate**
- **Noise: < -150 dBc/Hz at 20 MHz offset (3.6 GHz carrier)**
  - **Phase noise at the 20 MHz frequency offset is very challenging for GSM and DCS transmitters**
    - GSM: -162 dBc/Hz at 20 MHz offset (900 MHz carrier)
    - DCS: -151 dBc/Hz at 20 MHz offset (1.8 GHz carrier)

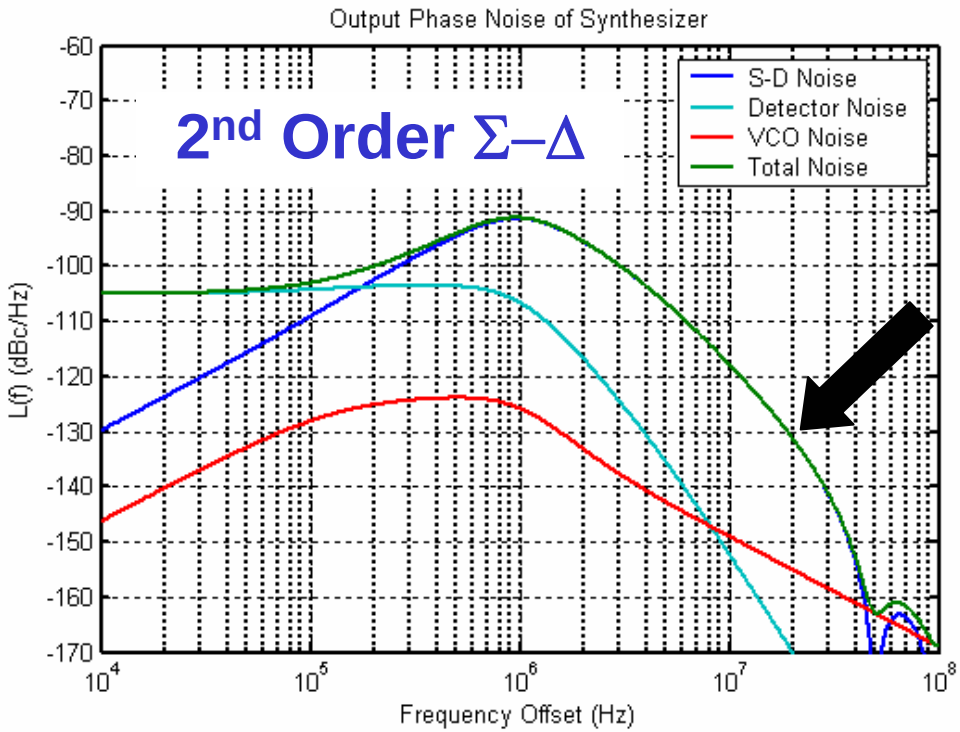
Simultaneous achievement of the above bandwidth and noise targets is very challenging

Evaluate Noise Performance with 1 MHz PLL BW

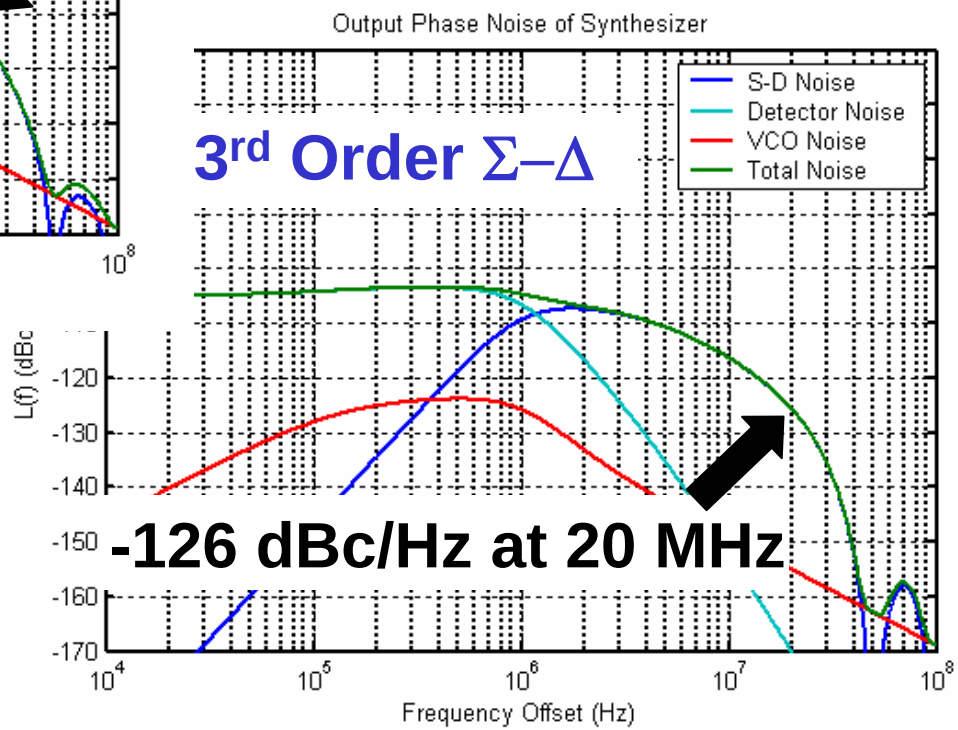
- G(f) parameters
  - 1 MHz BW, Type II, 2nd order rolloff, extra pole at 2.5 MHz
- Required PLL noise parameters (with a few dB of margin)
  - Output-referred charge pump noise: -105 dBc/Hz
  - VCO noise: -155 dBc/Hz at 20 MHz offset (3.6 GHz carrier)



Calculated Phase Noise for Classical Fractional-N



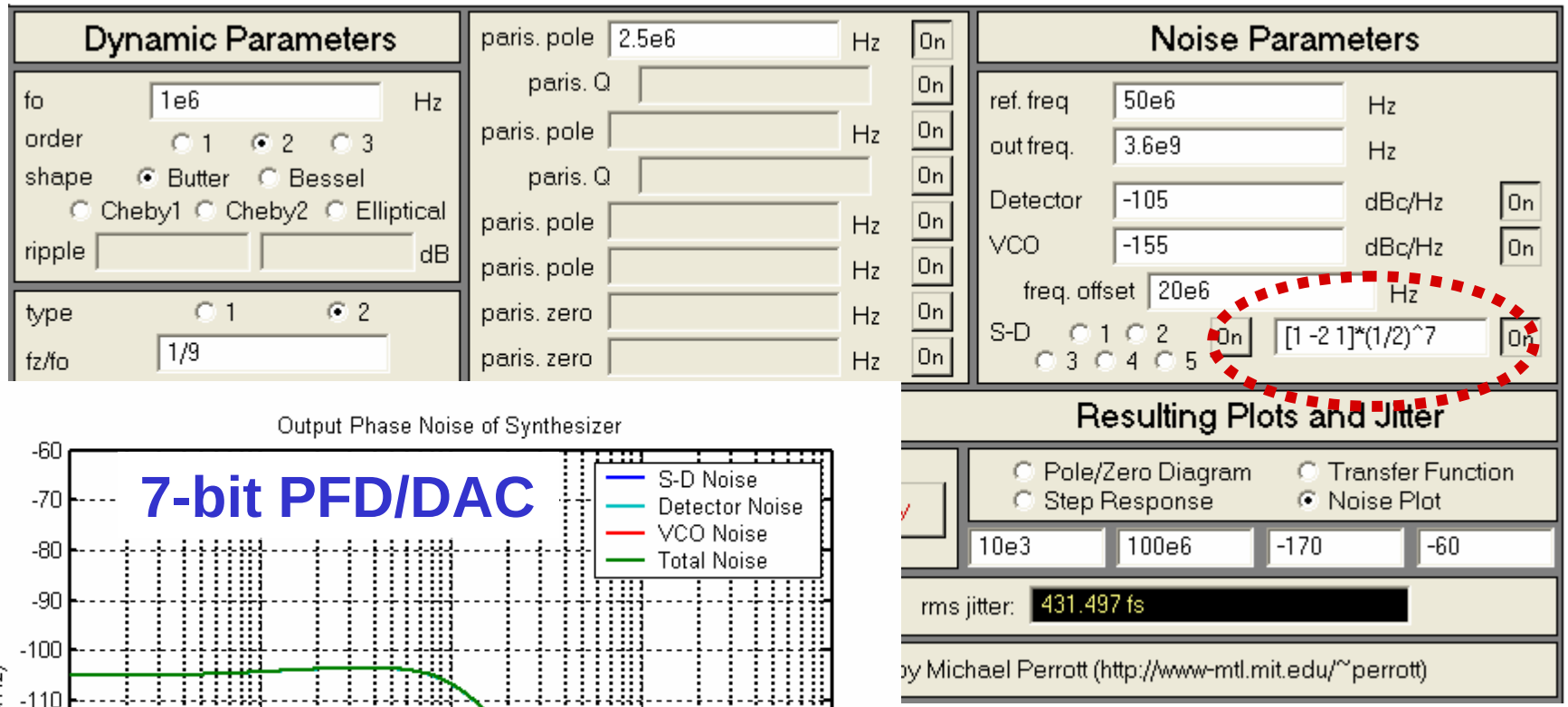
-132 dBc/Hz at 20 MHz

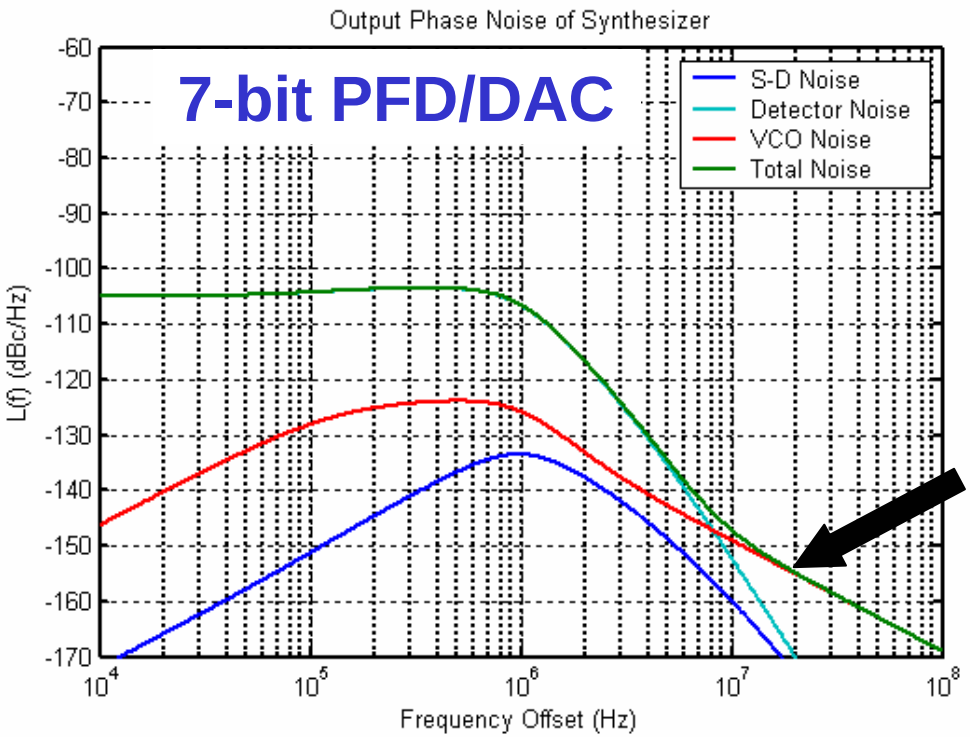


-126 dBc/Hz at 20 MHz

These do NOT meet our target of -150 dBc/Hz at 20 MHz (3.6 GHz carrier freq.)

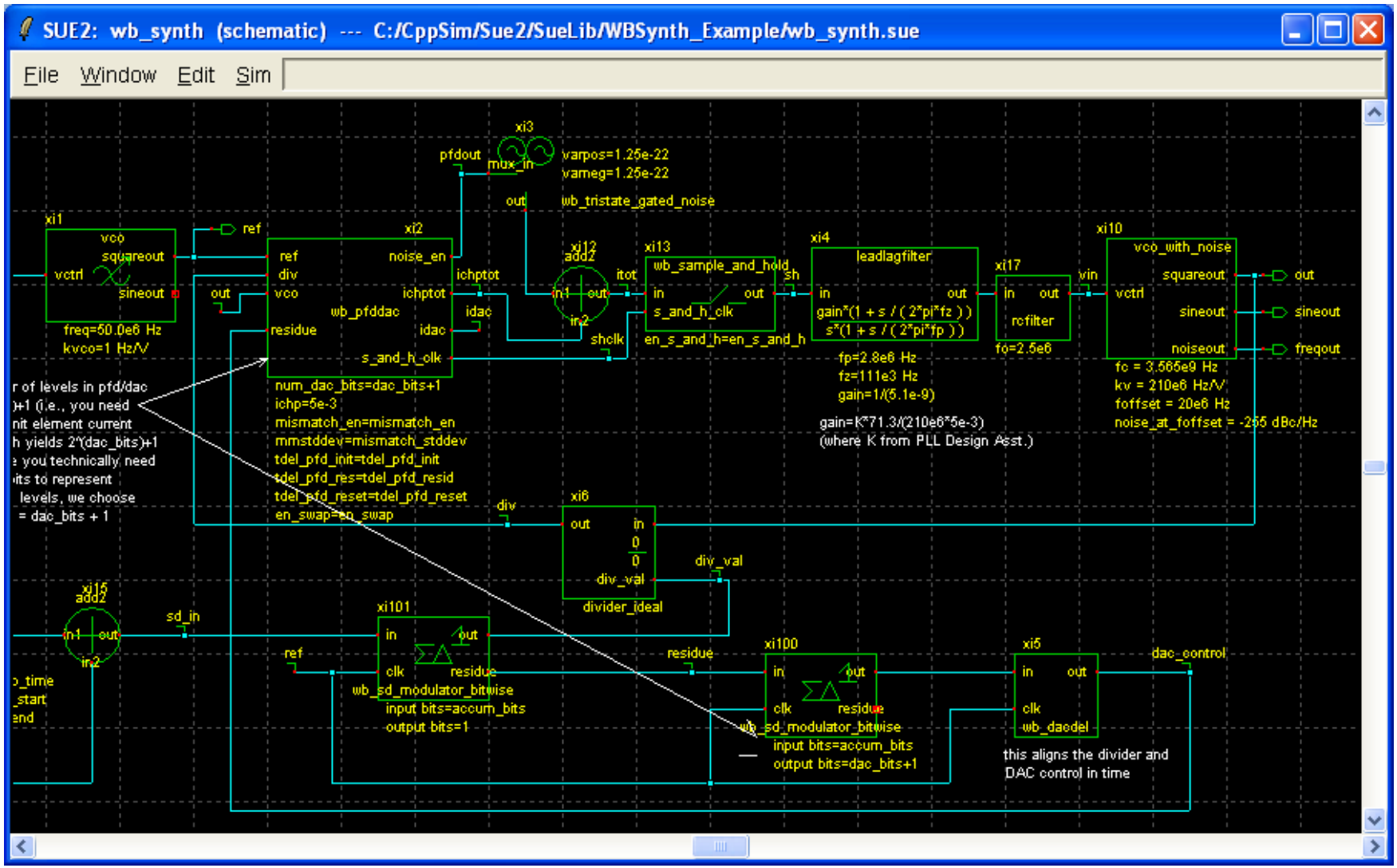
Calculated Phase Noise for 7-bit PFD/DAC Synth





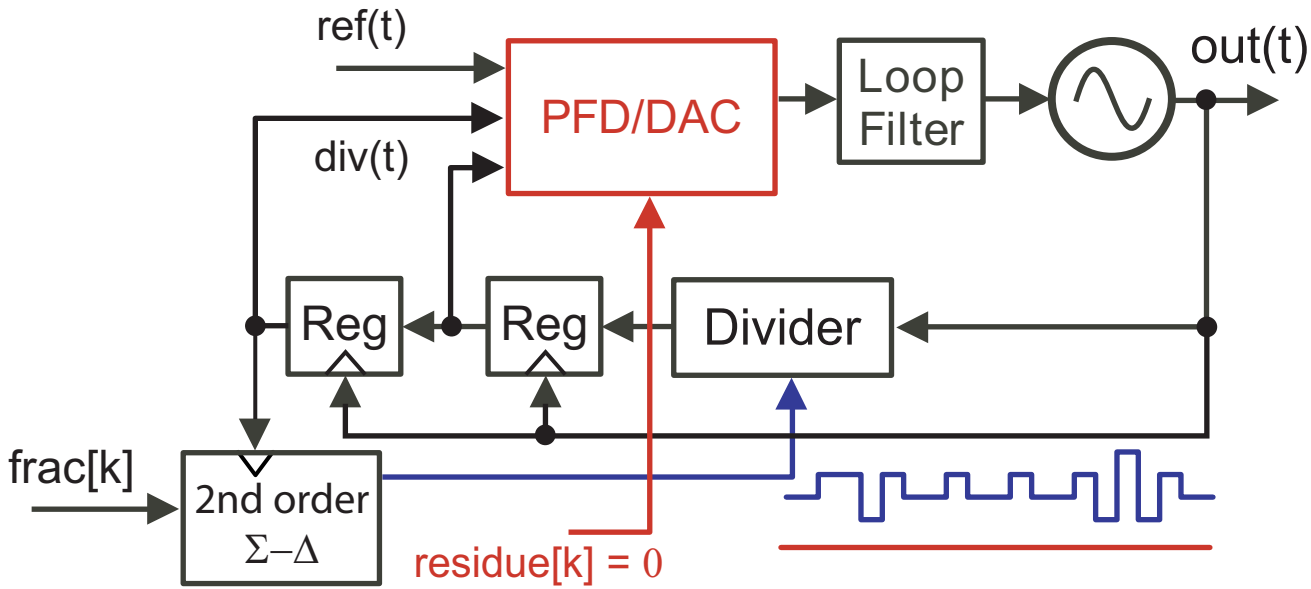
-155 dBc/Hz at 20 MHz !

Simulation of PFD/DAC Synthesizer using CppSim



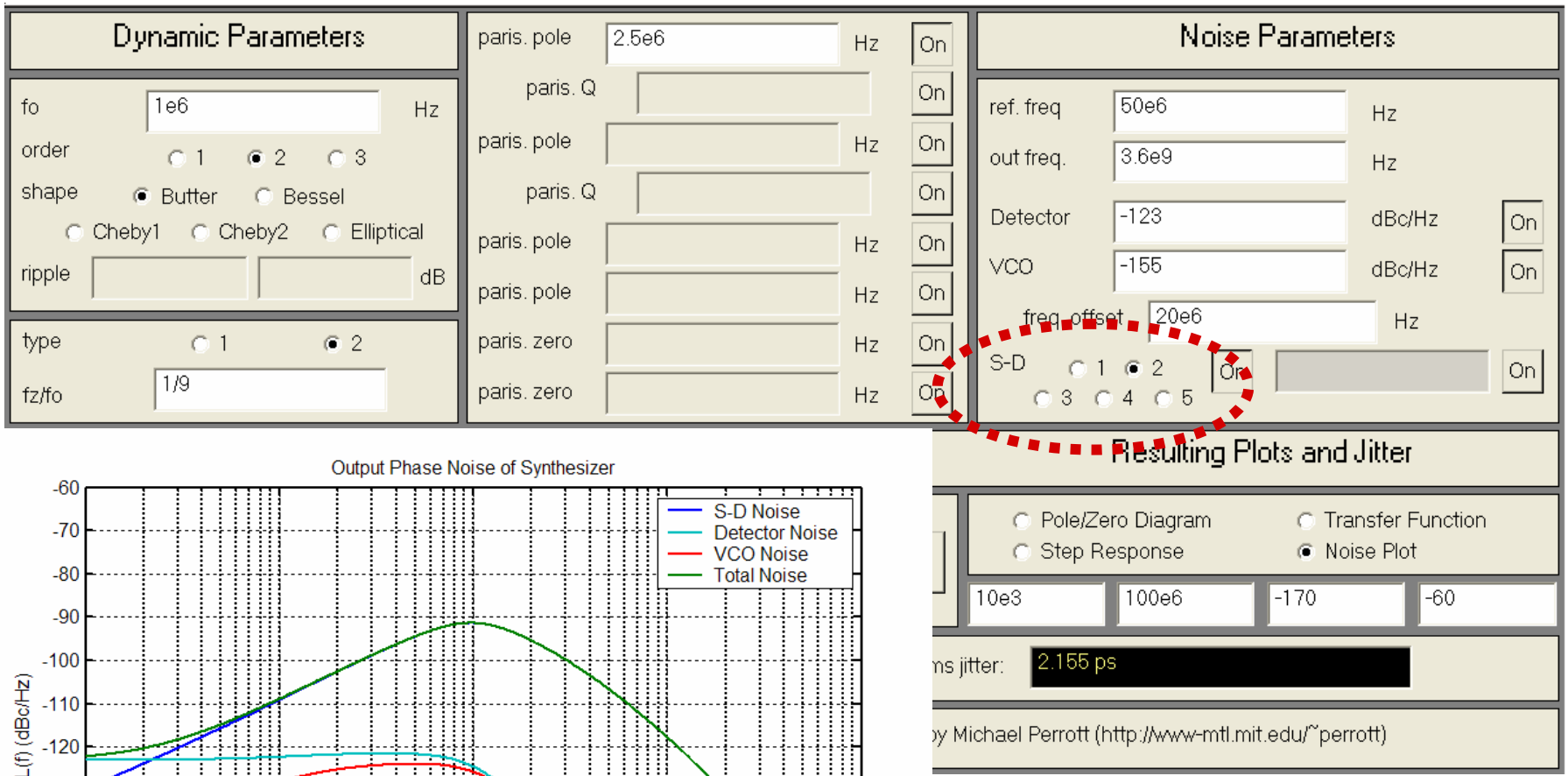
- Phase noise plots to follow: 40e6 time steps in 11 min

2nd Order Σ–Δ Fractional-N Performance



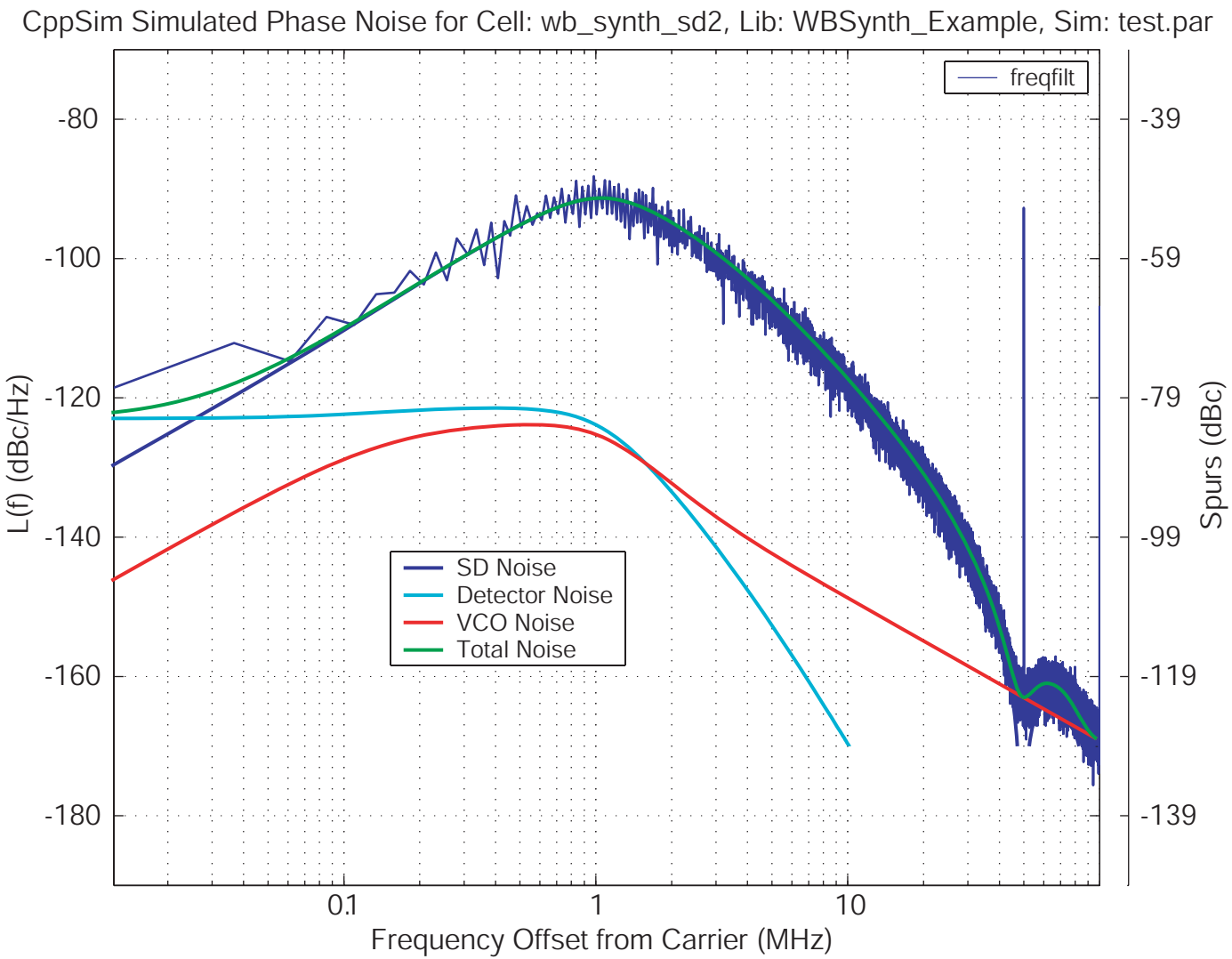
- Replace accumulator with second order Σ–Δ modulator
- Set residue into PFD/DAC equal to zero

Calculate PLL Noise for 2nd Order Σ–Δ Synthesizer



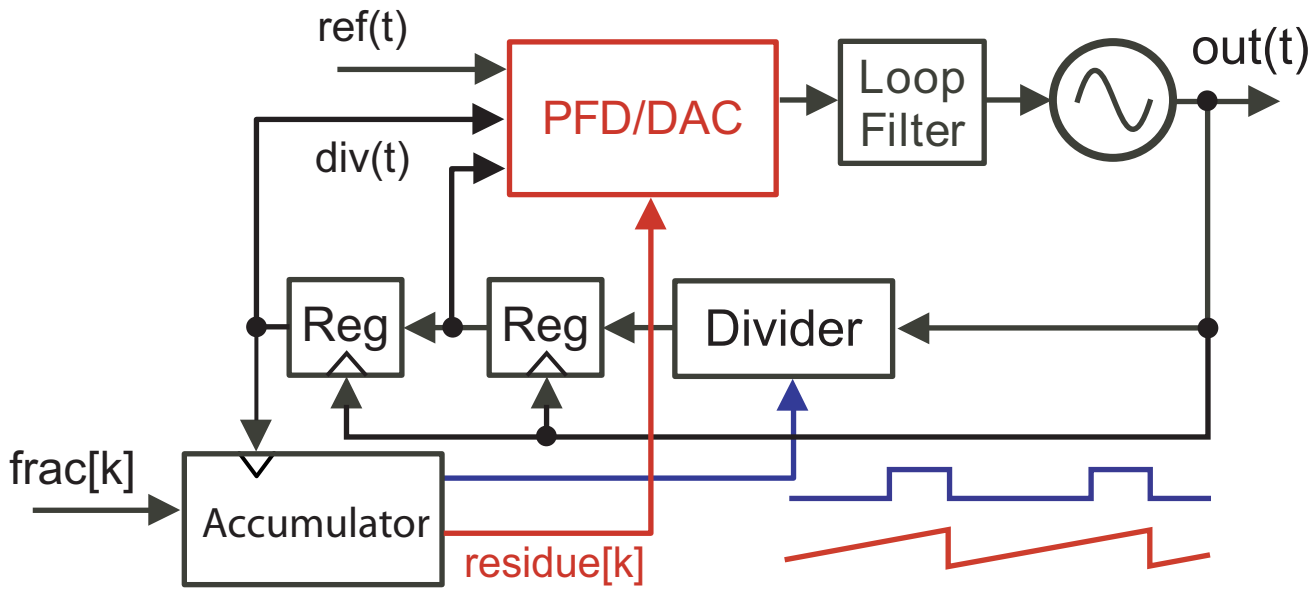
- 2nd order Σ–Δ
  - Click on 2nd order S-D quantization noise in tool

Simulated Phase Noise of 2nd Order Σ–Δ Synthesizer



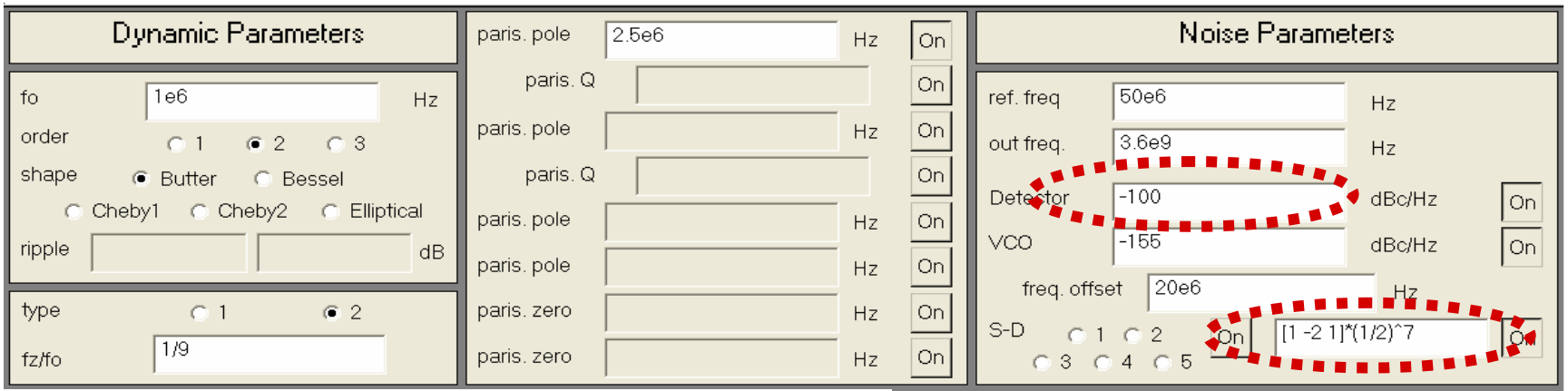
- **PLL Design Assistant accurately models simulated noise!**

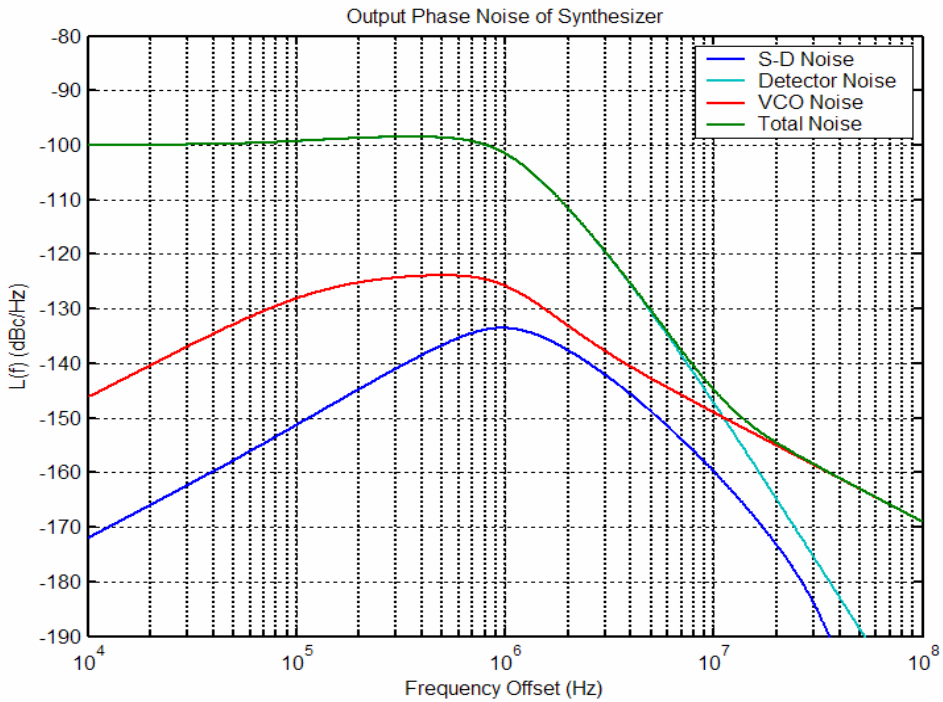
7-bit PFD/DAC Synthesizer Performance



- **Use accumulator for dithering**
- **Enable cancellation by connecting accumulator residue to PFD/DAC**

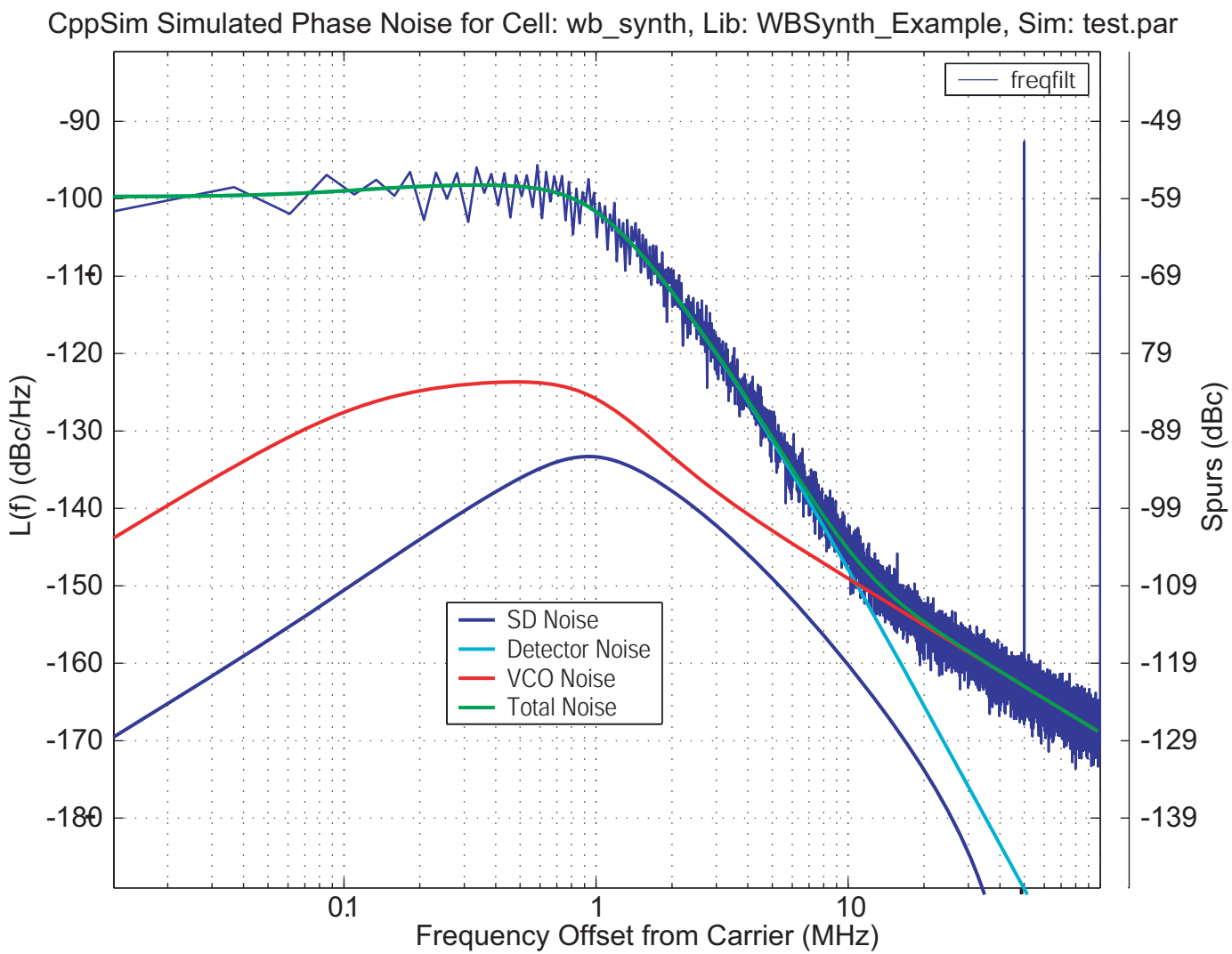
Calculate PLL Noise for 7-bit PFD/DAC Synthesizer





- **PFD/DAC**
  - **Adjust S-D Quant. Noise**
- **Delay mismatch (11 ps)**
  - **Adjust Detector noise**

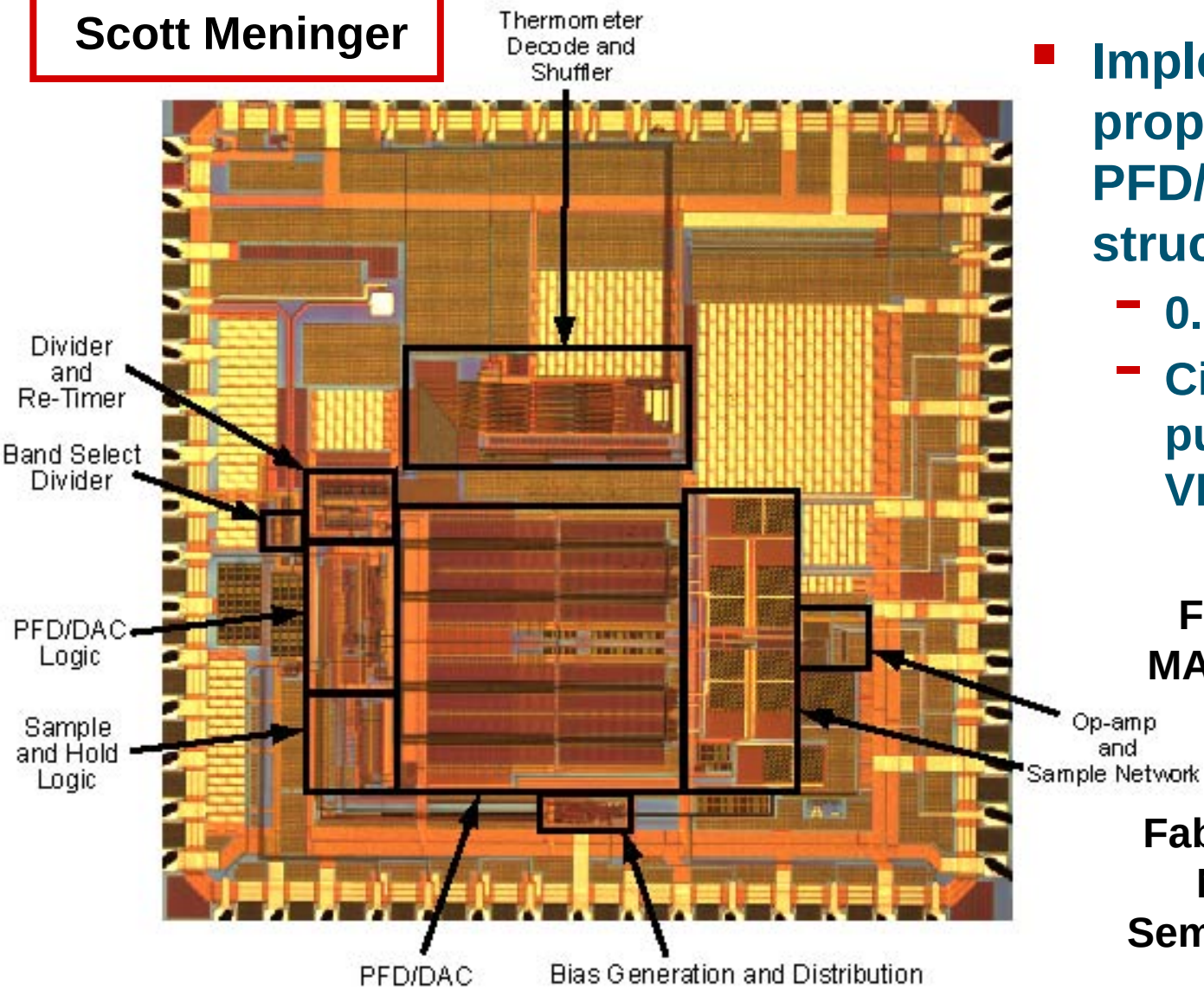
Simulated PLL Phase Noise of 7-bit PFD/DAC



- **PLL Design Assistant accurately models simulated noise!**

A 1 MHz BW Fractional-N Frequency Synthesizer IC

Scott Meninger

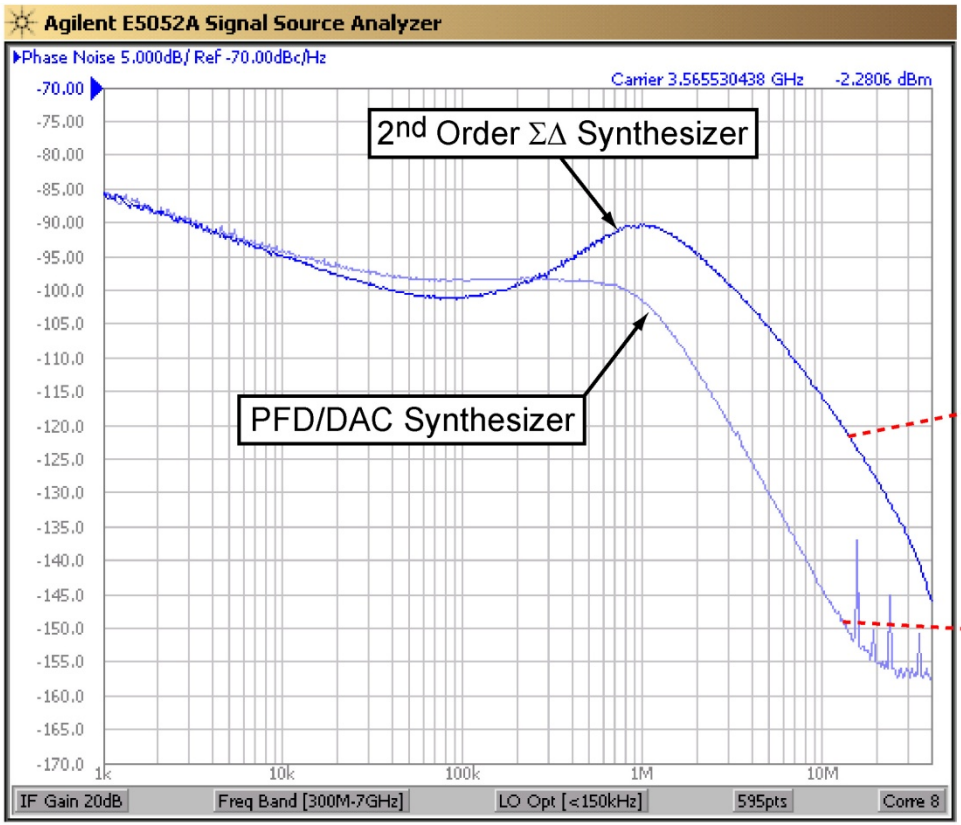


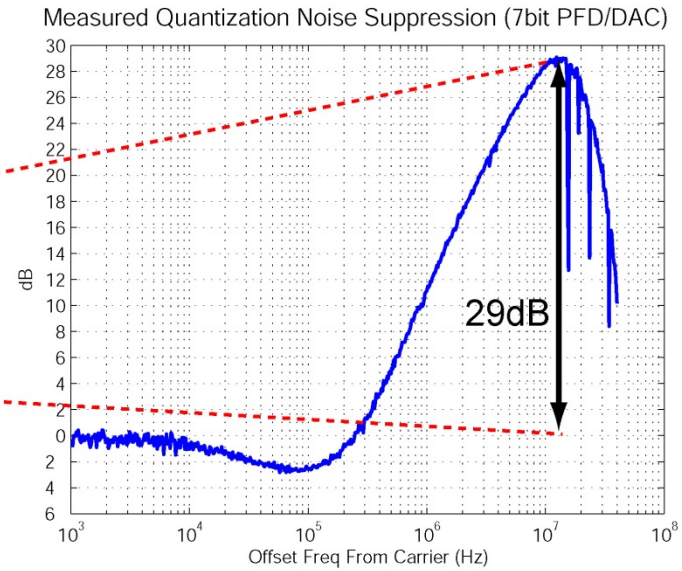
- **Implements proposed 7-bit PFD/DAC structure**
  - **0.18u CMOS**
  - **Circuit details published in VLSI 2005**

Funded by MARCO C2S2

Fabricated by National Semiconductor

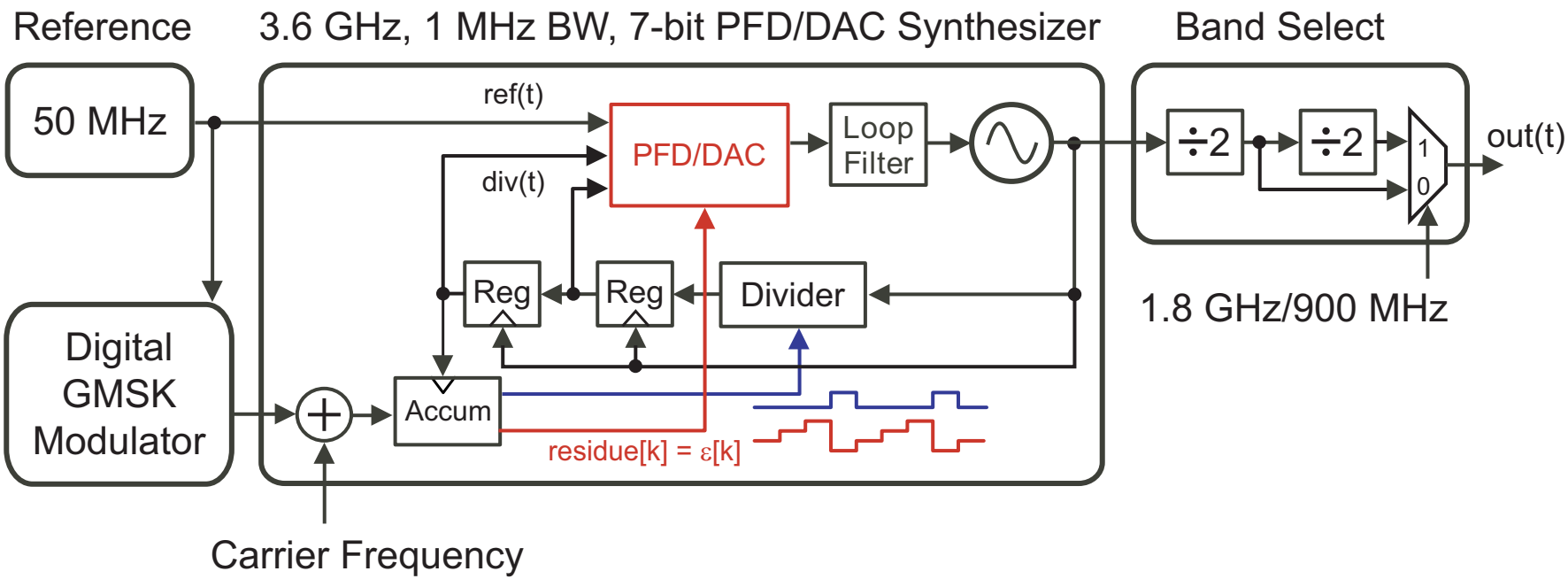
Measured Noise Suppression





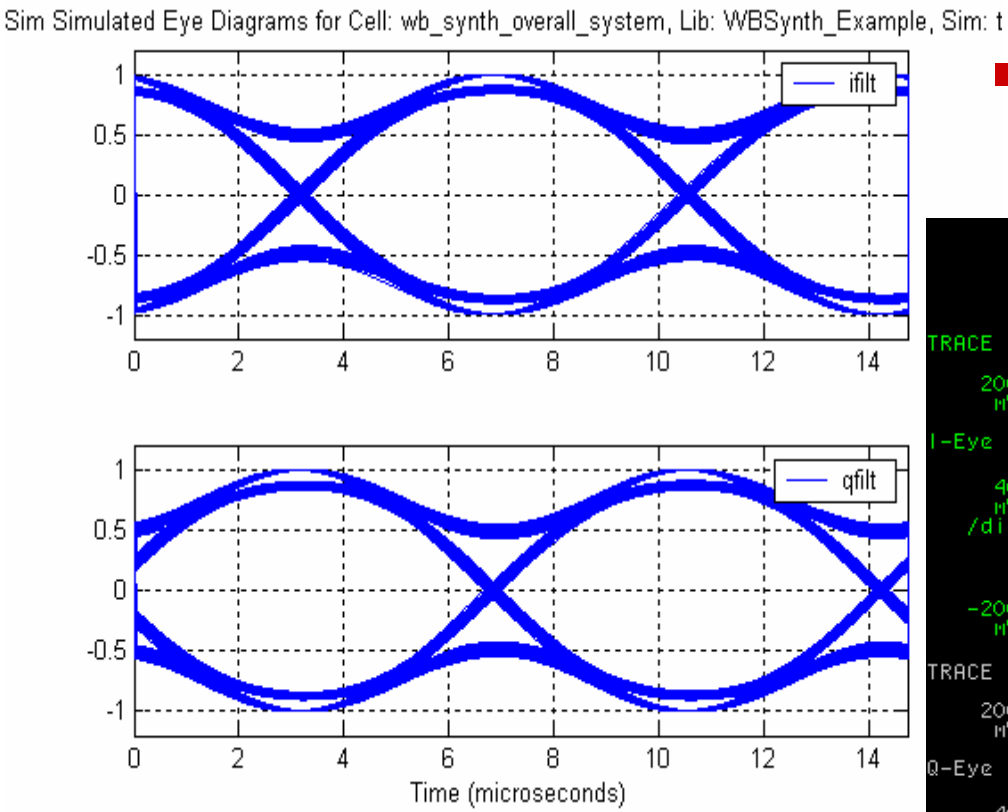
- Comparison of 7-bit PFD/DAC synthesizer with 2nd order ΣΔ Synthesizer
- Low freq noise ~2dB worse because of phase swapping
- 29dB quantization noise suppression measured at 10MHz !

A Highly Digital Implementation of a GMSK Transmitter

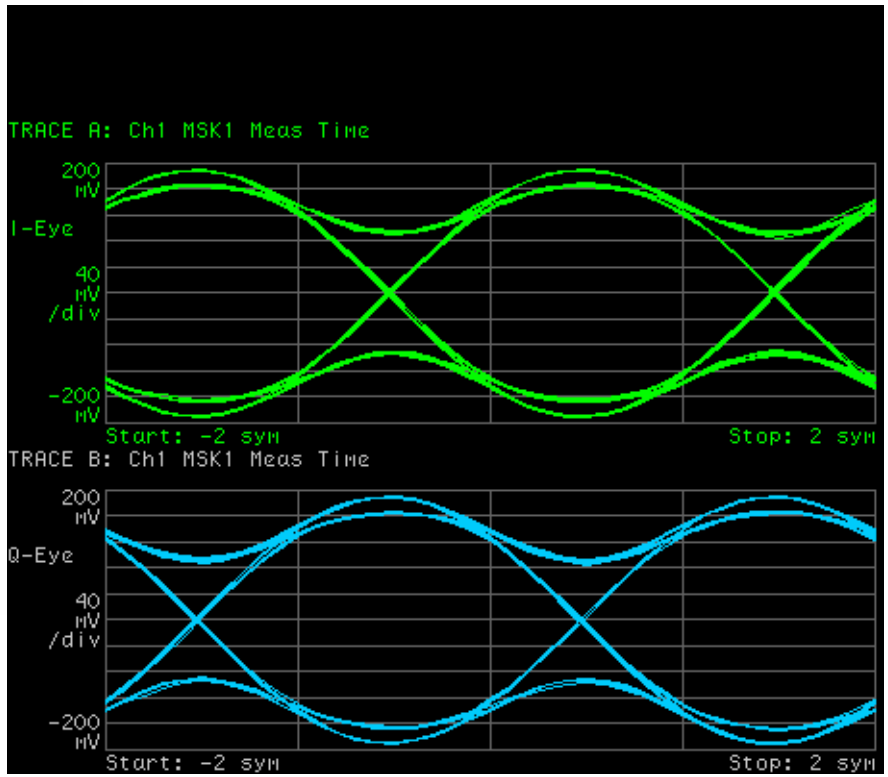


- **A fractional-N frequency synthesizer provides highly accurate phase/frequency modulation capability**
  - **Multiple carrier frequencies easily achieved with digital frequency division**
  - **N-bit PFD/DAC extends achievable data rate for a given noise performance (at higher frequency offsets)**

GMSK Eye Diagrams at 271 kbit/s (~900 MHz Carrier)



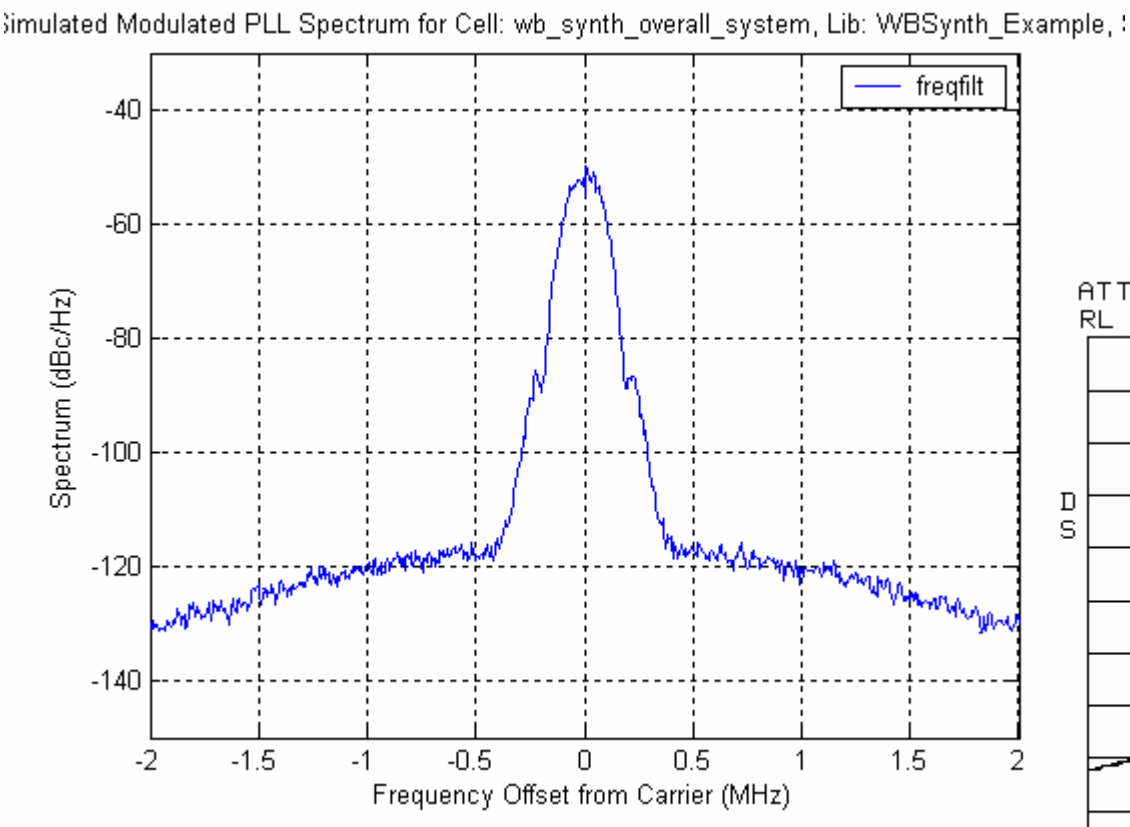
- **Measured**
  - **HP 89441 Vec. Analyzer**



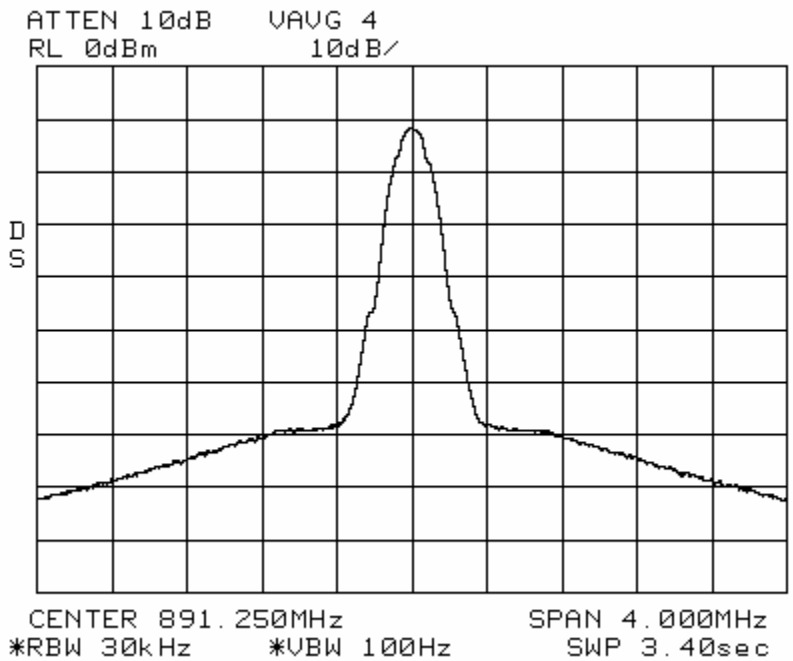
- **CppSim simulation**
  - **100e6 points: < 44 min**

Close agreement between simulated and measured results!

GMSK Spectra Plots at 271 kbit/s (~900 MHz Carrier)

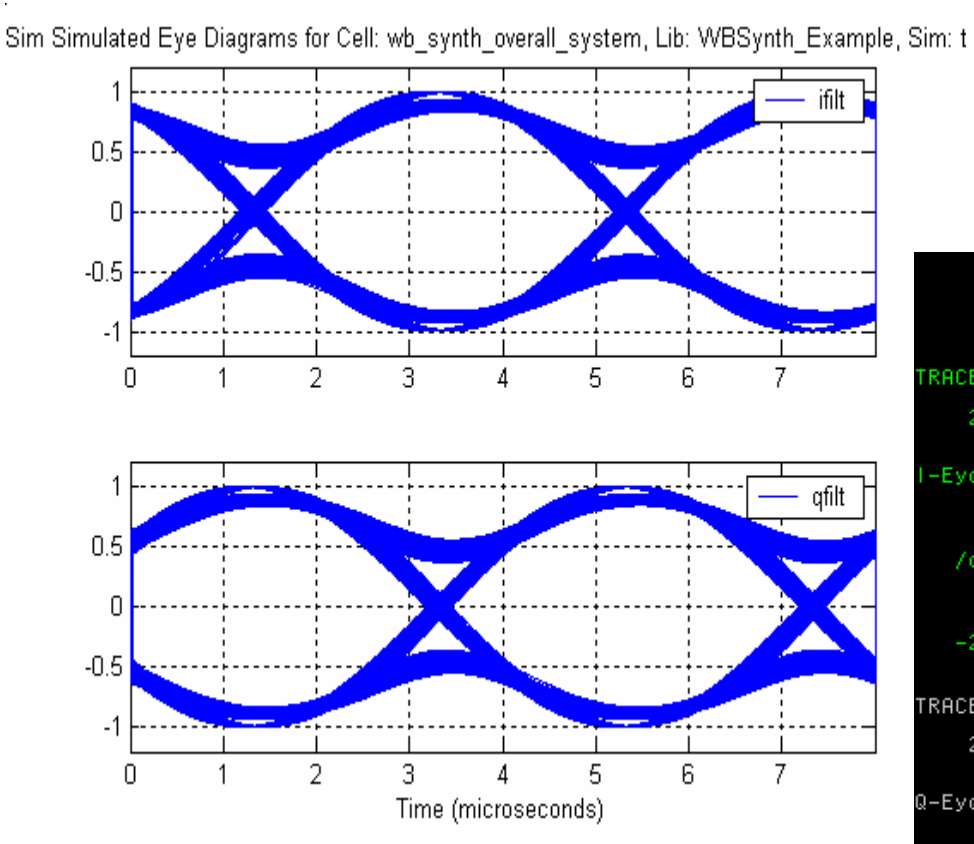


- **Measured**
  - **HP 8563E Spectrum Analyzer**

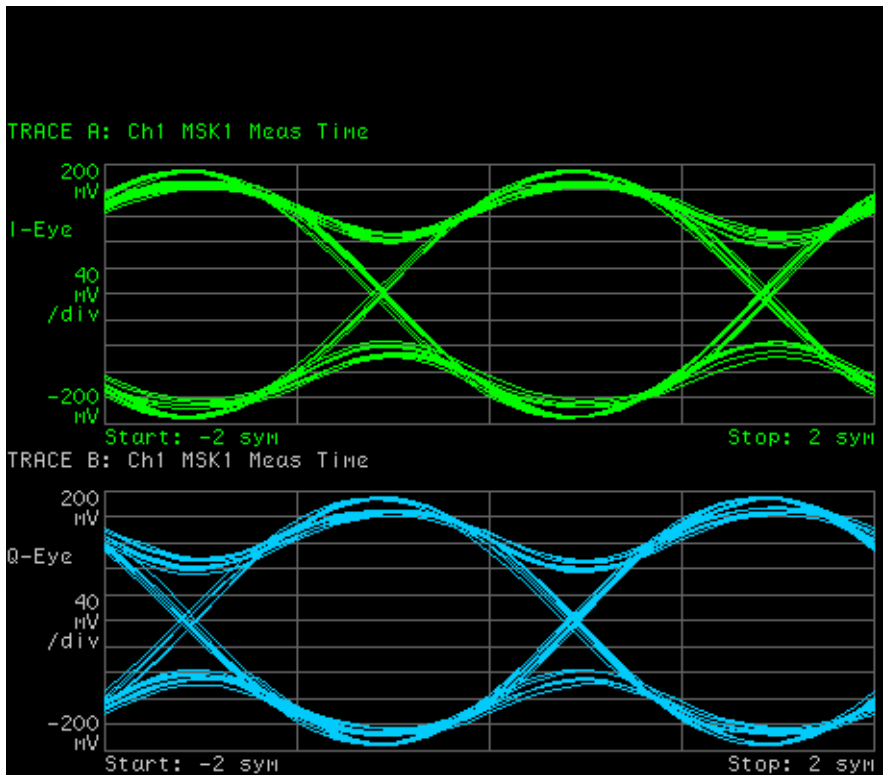


- **CppSim simulation**
  - **100e6 points: < 44 min**

GMSK Eye Diagrams at 500 kbit/s (~900 MHz Carrier)

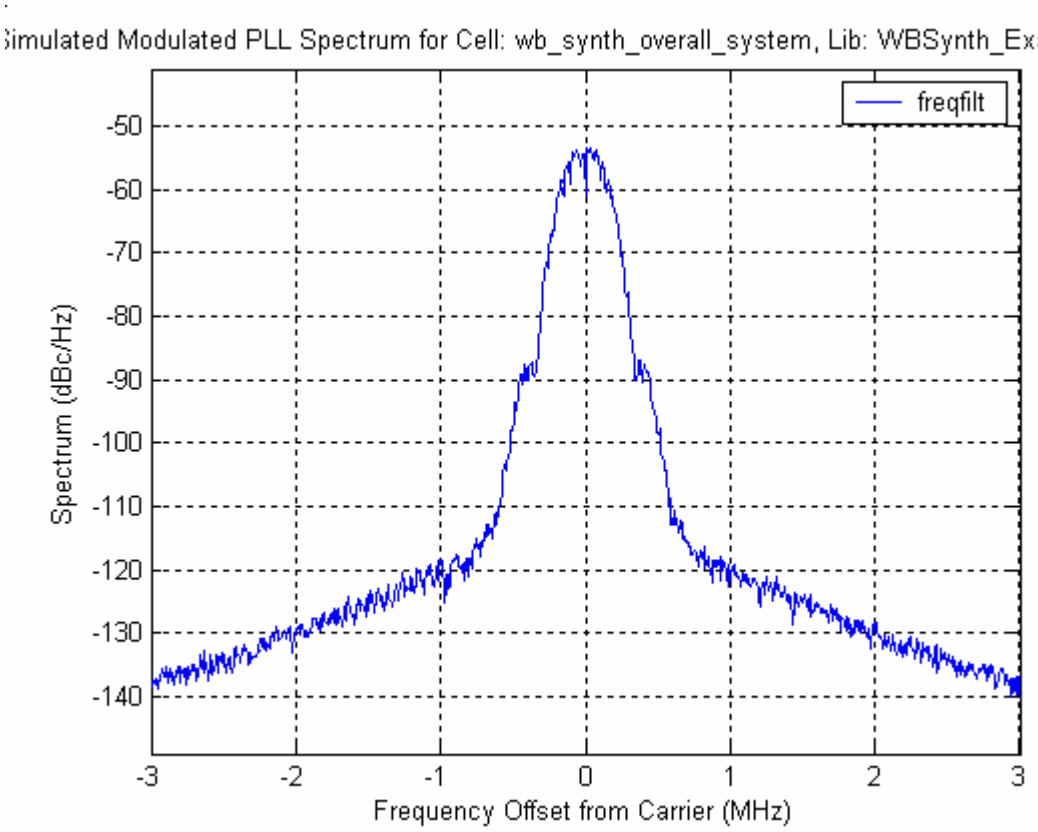


- **Measured**
  - **HP 89441 Vec. Analyzer**

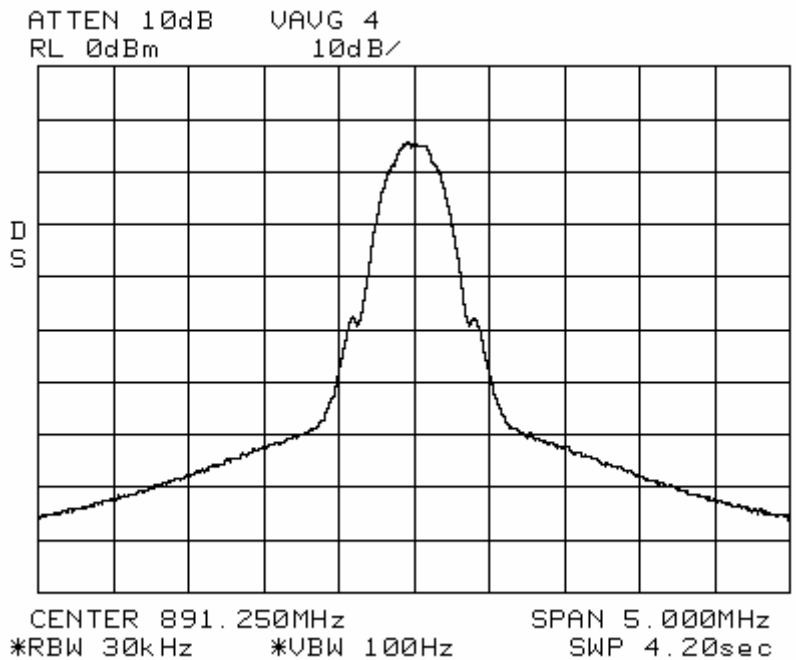


- **CppSim simulation**
  - **100e6 points: < 44 min**

GMSK Spectra Plots at 500 kbit/s (~900 MHz Carrier)

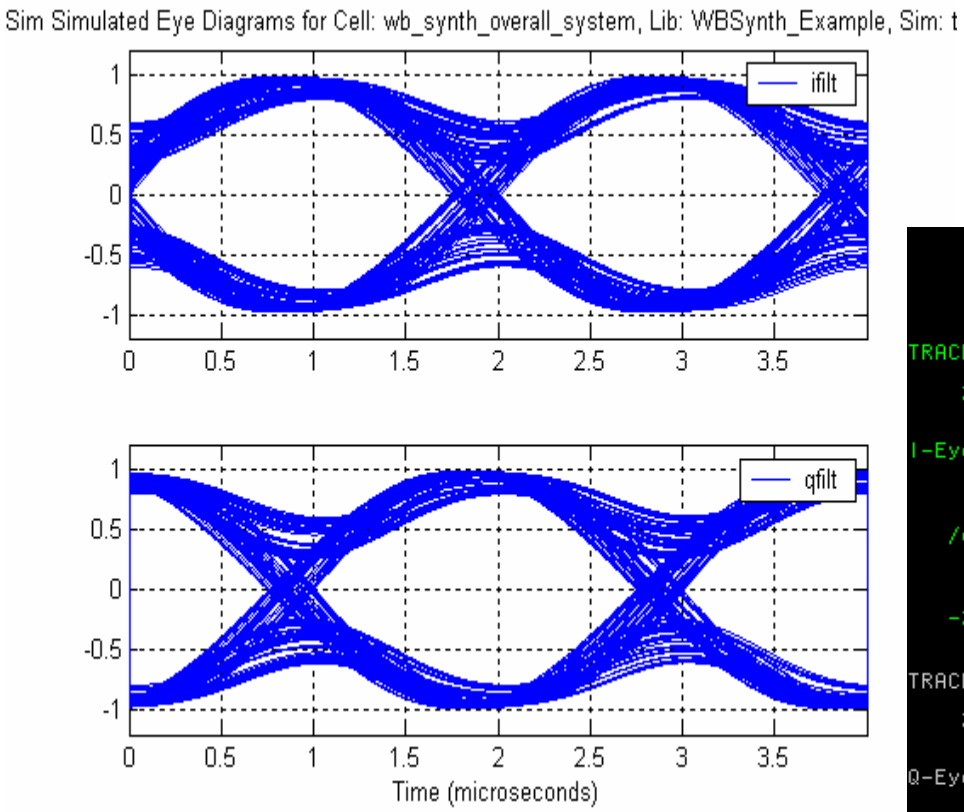


- **Measured**
  - **HP 8563E Spectrum Analyzer**

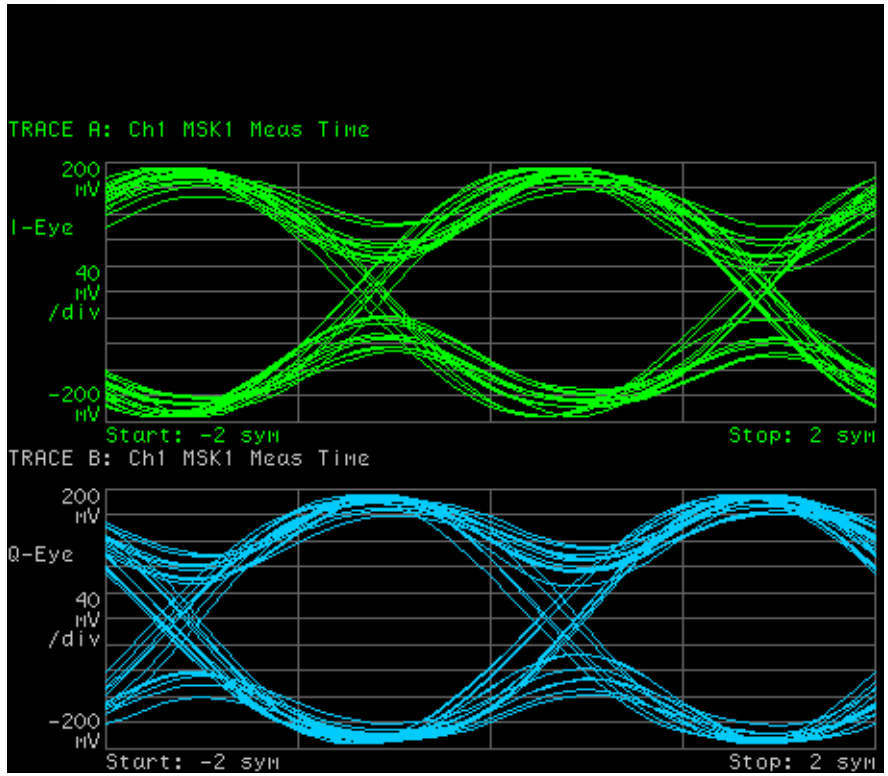


- **CppSim simulation**
  - **100e6 points: < 44 min**

GMSK Eye Diagrams at 1 Mbit/s (~900 MHz Carrier)

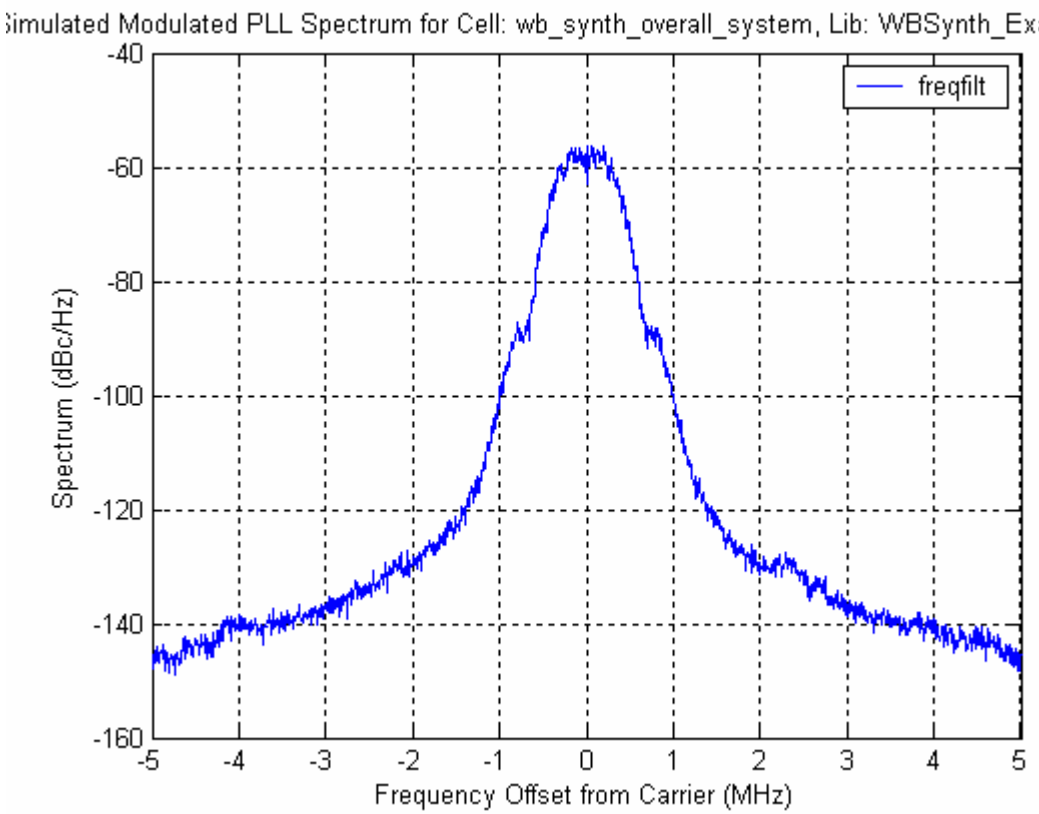


- **Measured**
  - **HP 89441 Vec. Analyzer**

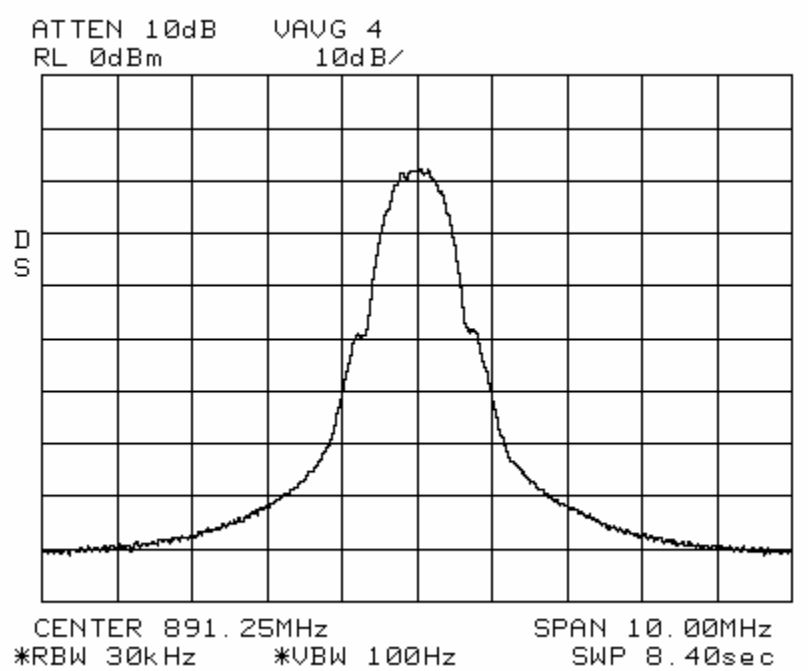


- **CppSim simulation**
  - **100e6 points: < 44 min**

GMSK Spectra Plots at 1 Mbit/s (~900 MHz Carrier)



- **Measured**
  - **HP 8563E Spectrum Analyzer**



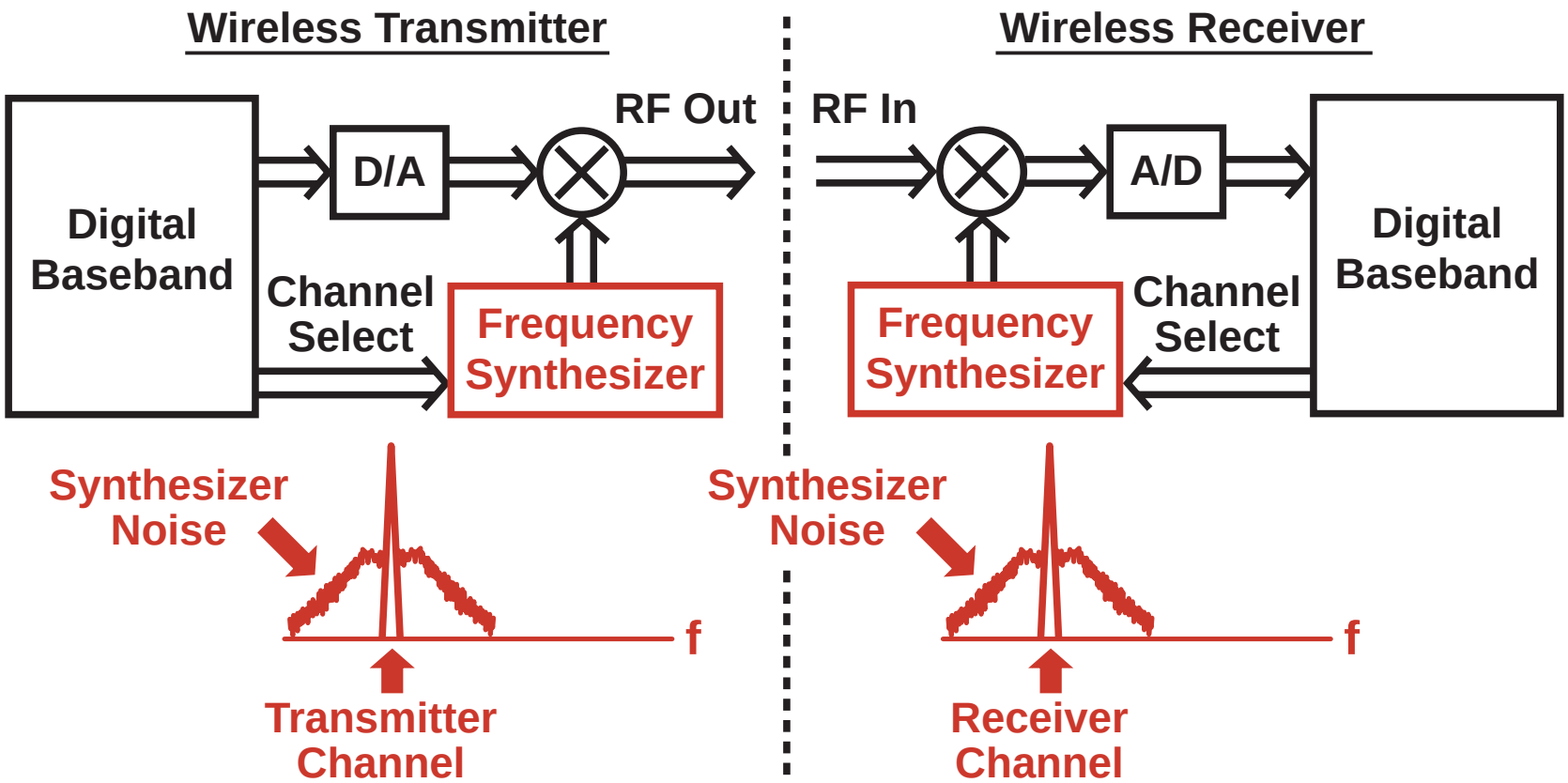
- **CppSim simulation**
  - **100e6 points: < 44 min**

Conclusions

- **Fractional-N frequency synthesizers can achieve dramatic improvement in achieving high PLL bandwidth with excellent noise performance**
  - **The PFD/DAC approach presented here is only one of many possibilities to achieve this goal**
- **Design and simulation methodologies are invaluable for achieving better performance**
  - **Analytical modeling of noise can be quite accurate**
    - The PLL Design Assistant can be useful in this area
  - **Behavioral simulation can be used to verify analytical models**
    - CppSim offers a convenient and fast framework for this

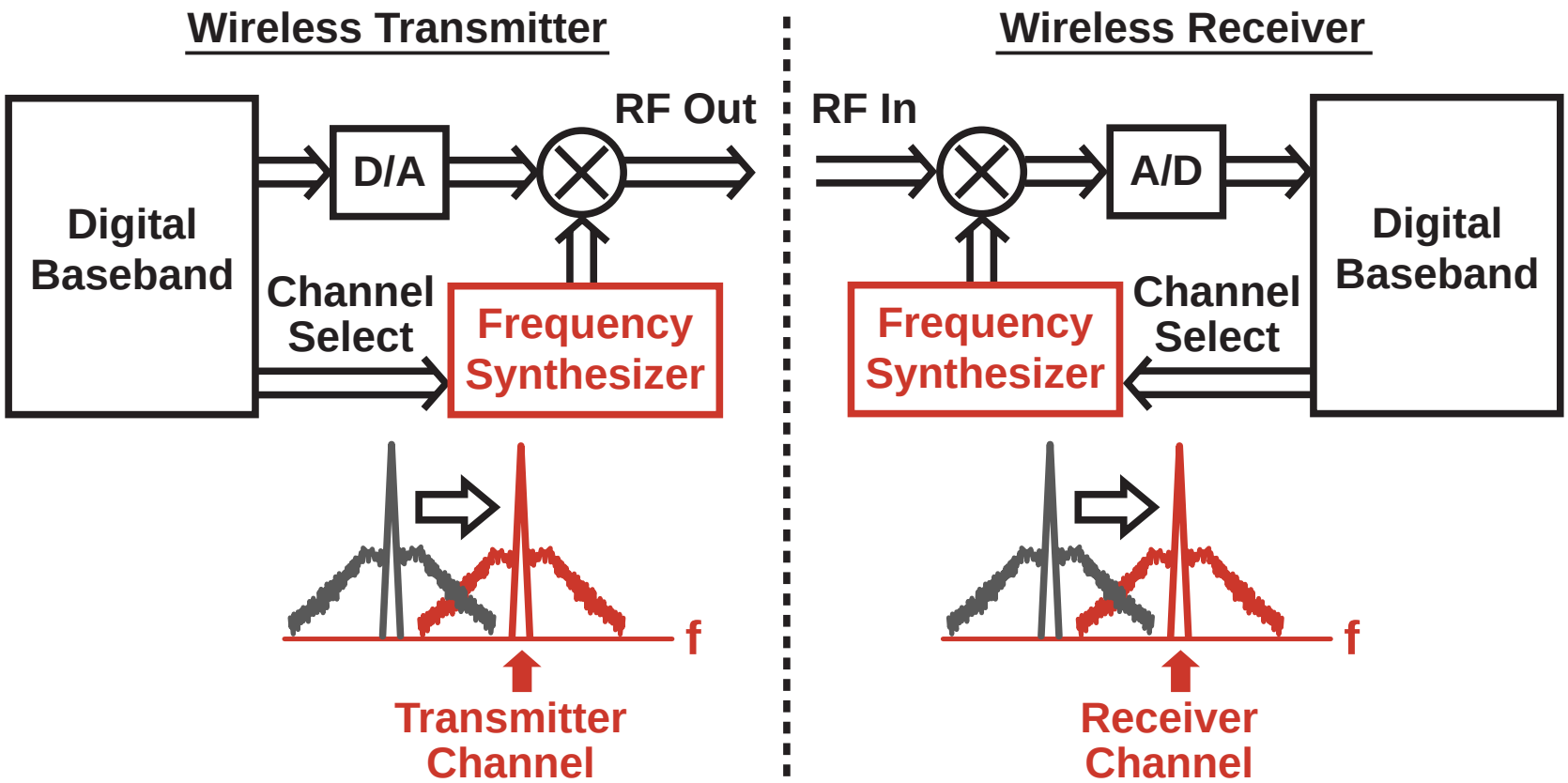
Simulation of Frequency Synthesizers

Impact of Synthesizer Noise



- Noise must be low to meet transmit mask requirement
- Noise must be low to meet receiver SNR and blocking requirements

Impact of Synthesizer Dynamic Behavior



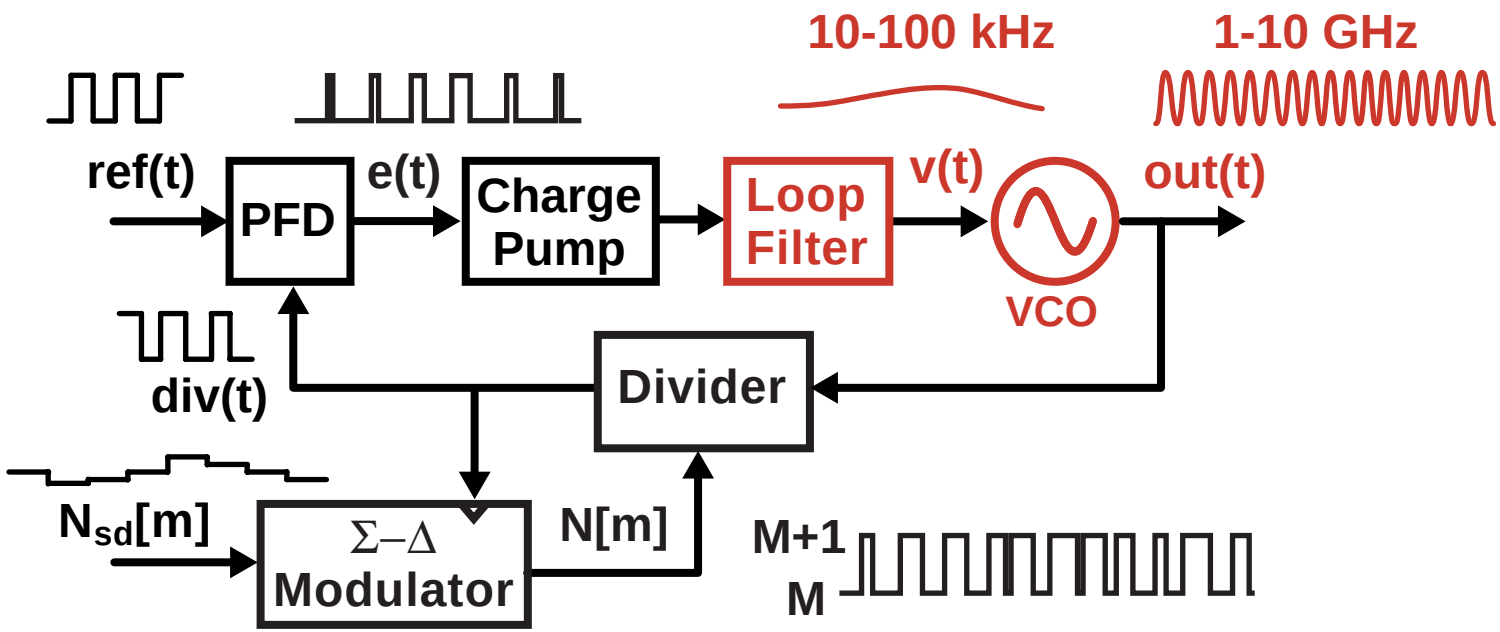
- **Settling time must be fast to support channel hopping requirements**

What Do We Want From a Simulator?

- Accurate estimation of synthesizer performance
  - Noise spectral density
  - Dynamic behavior
- Fast computation to allow use in IC design flow
- Simple to use
  - C++, Verilog, Matlab

Problems with Current Simulators

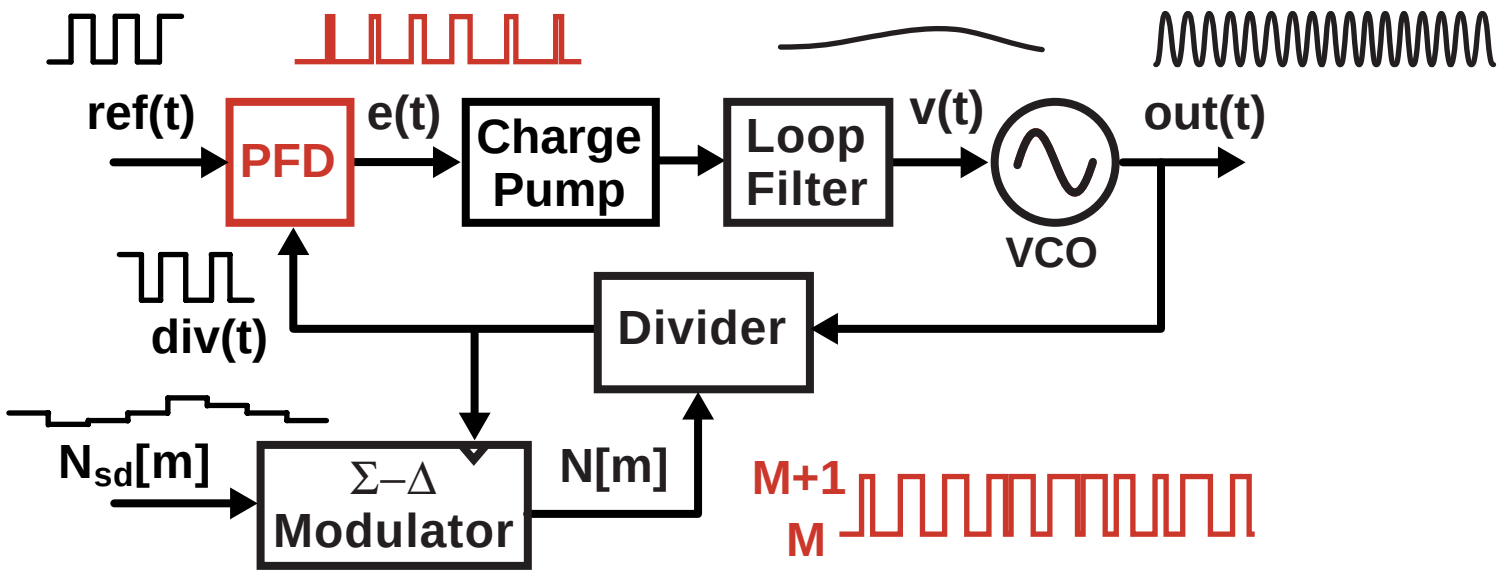
Problem 1: Classical Simulators are Slow



- High output frequency ⇨ High sample rate
- Long time constants ⇨ Long time span for transients

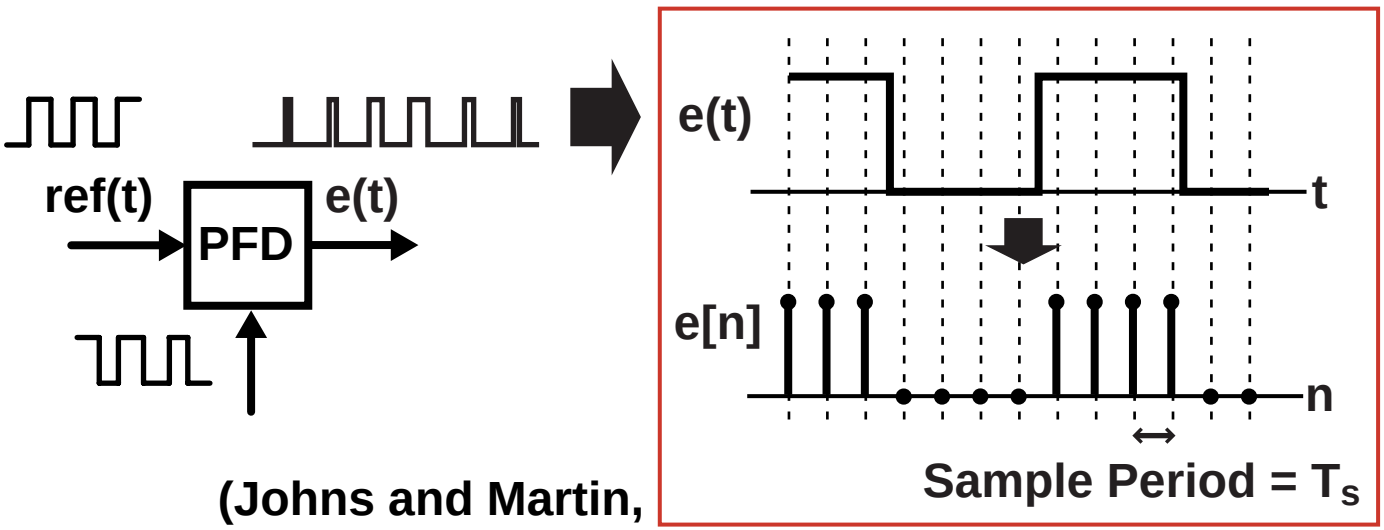
Large number of simulation time steps required

Problem 2: Classical Simulators Are Inaccurate



- PFD output is not bandlimited
  - PFD output must be simulated in discrete-time
- Phase error is inaccurately simulated
- Non-periodic dithering of divider complicates matters

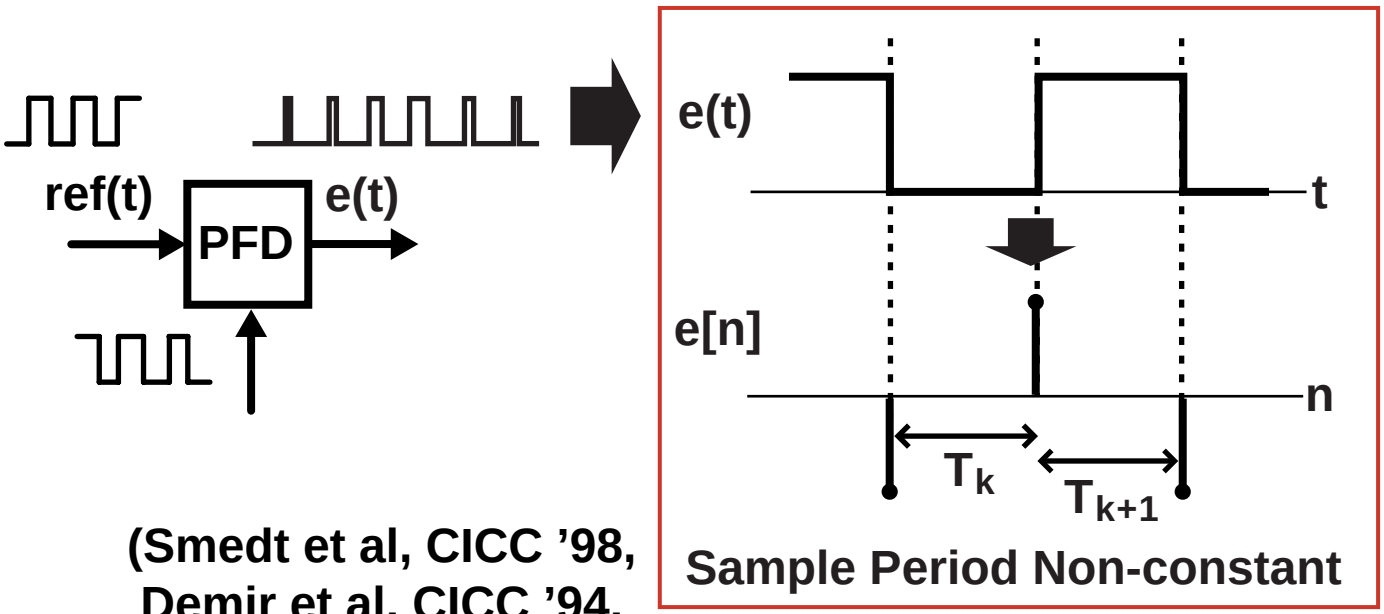
Example: Classical Constant-Time Step Method



(Johns and Martin, Analog Integrated Circuit Design)

- Directly sample the PFD output according to the simulation sample period
  - Simple, fast, readily implemented in Matlab, Verilog, C++
- Issue – quantization noise is introduced
  - This noise overwhelms the PLL noise sources we are trying to simulate

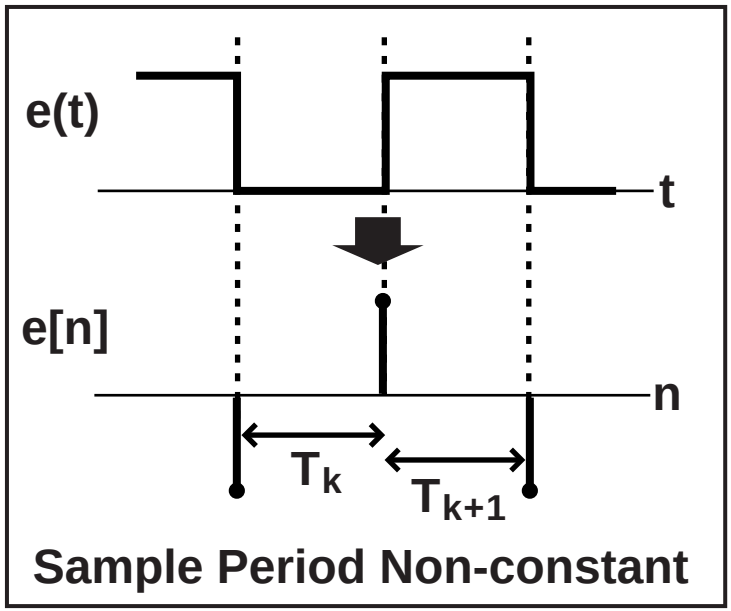
Alternative: Event Driven Simulation

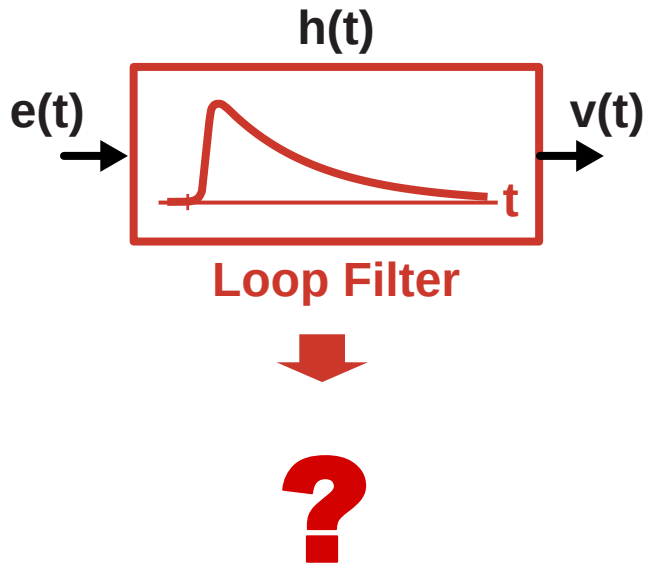


(Smedt et al, CICC '98,
Demir et al, CICC '94,
Hinz et al, Circuits and Systems '00)

- Set simulation time samples at PFD edges
  - Sample rate can be lowered to edge rate!

Issue: Simulation of Filter Blocks is Complicated

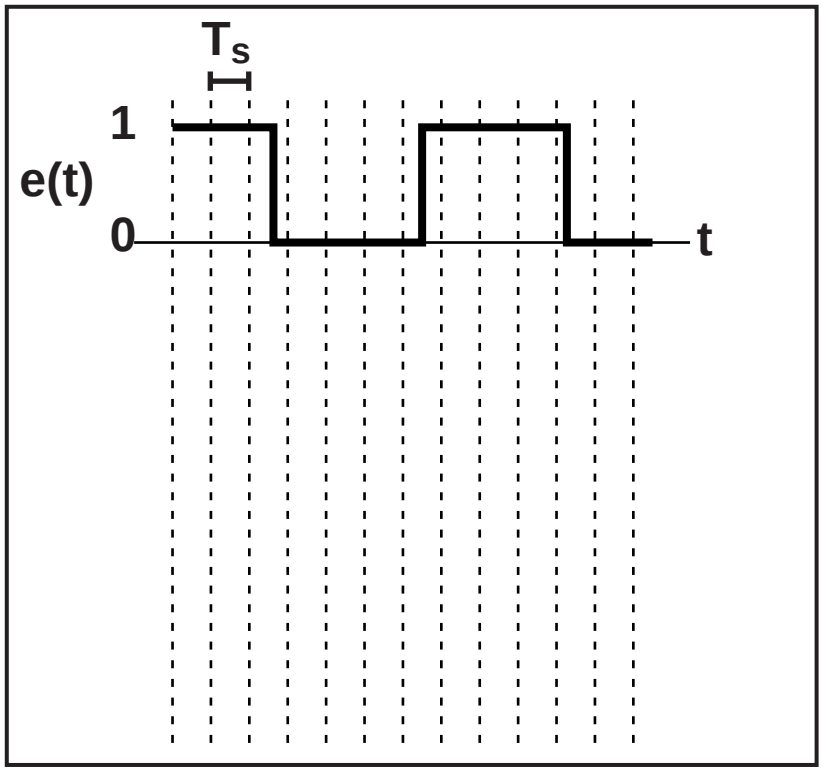


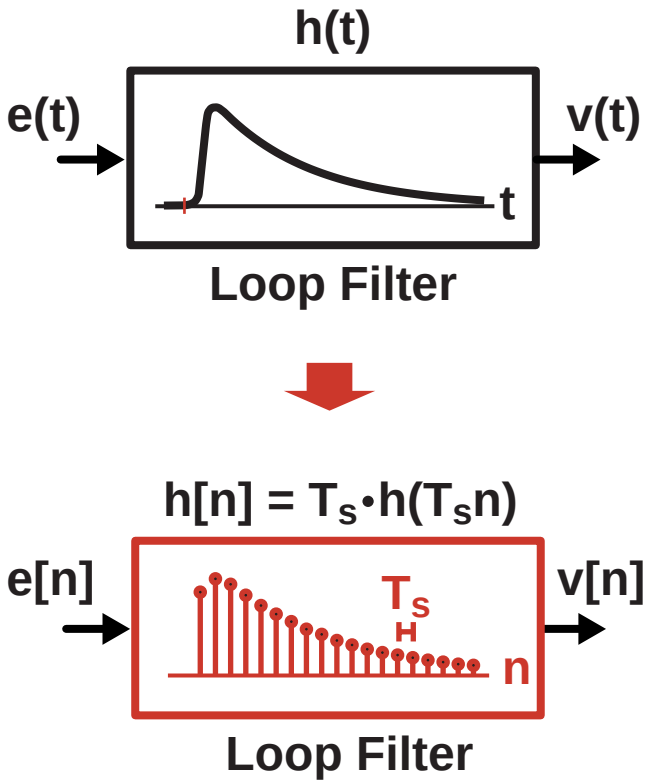


- Filtering computation must deal with non-constant time step
  - Closed-form calculation is tedious
  - Iterative computation is time-consuming
- Complicates Verilog, Matlab, or C++ implementation

Is there a better way?

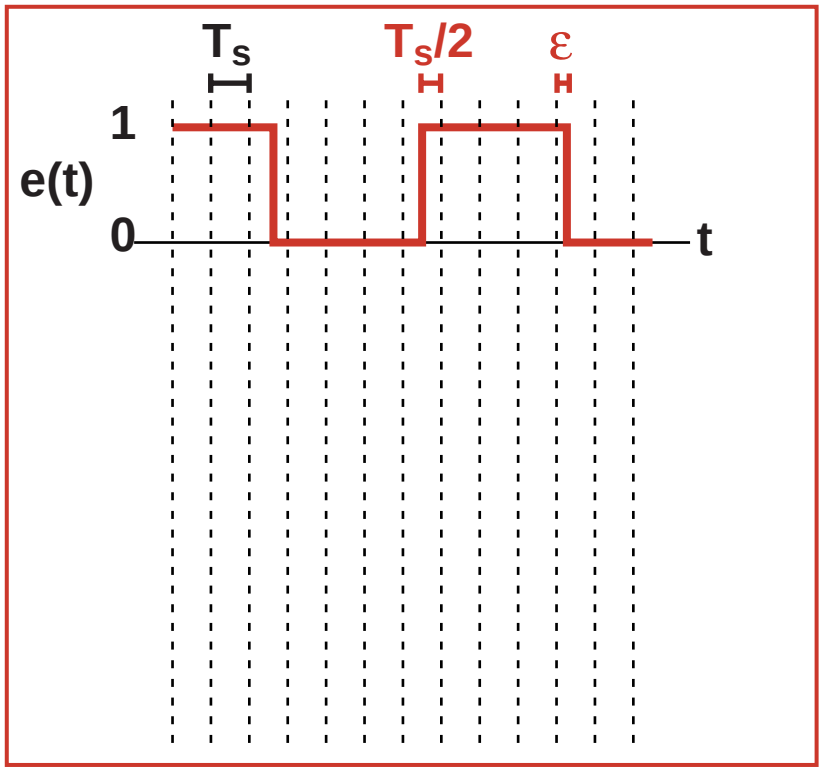
Proposed Approach: Use Constant Time Step

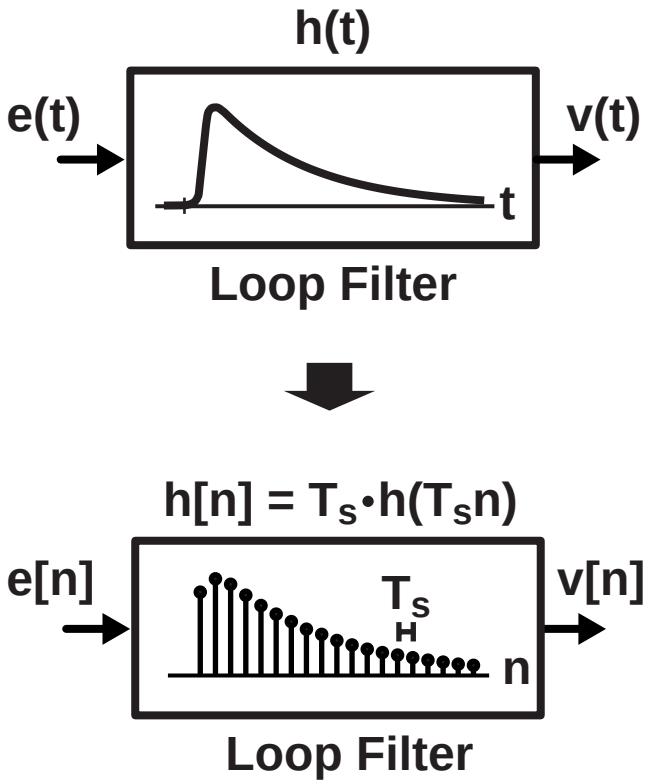




- Straightforward CT to DT transformation of filter blocks
  - Use bilinear transform or impulse invariance methods
- Overall computation framework is fast and simple
  - Simulator can be based on Verilog, Matlab, C++

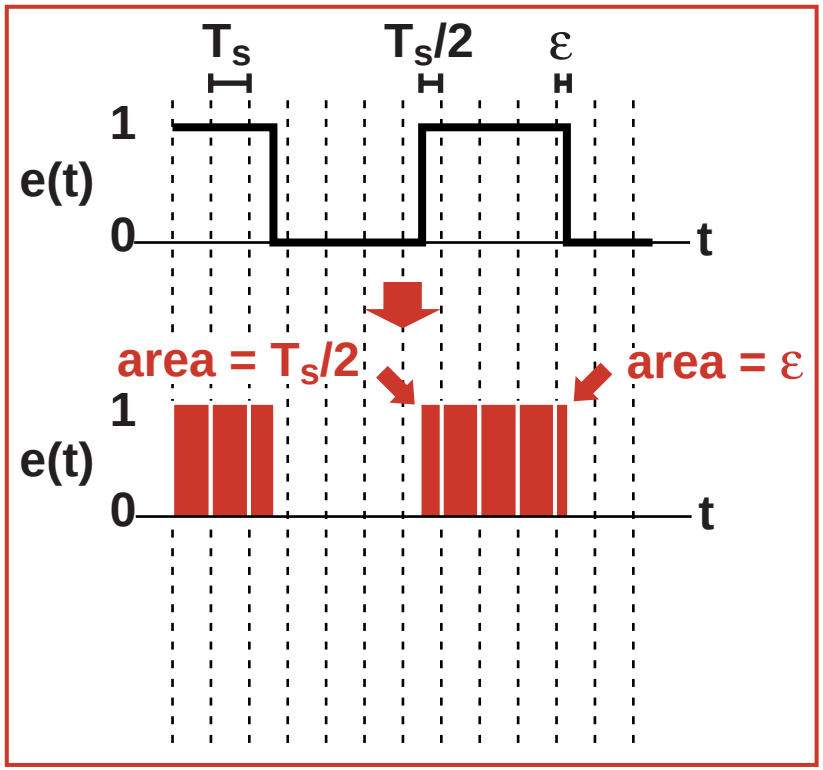
Problem: Quantization Noise at PFD Output

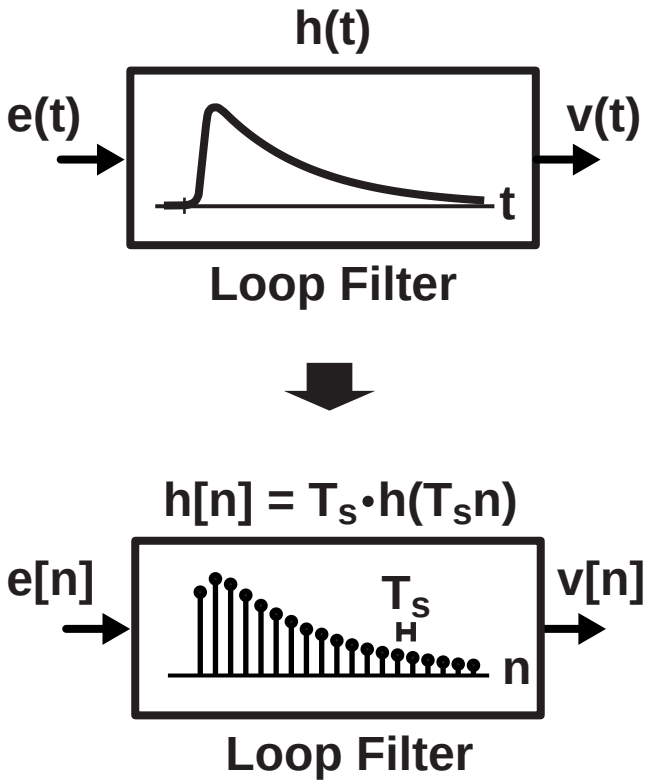




- Edge locations of PFD output are quantized
  - Resolution set by time step: T_s
- Reduction of T_s leads to long simulation times

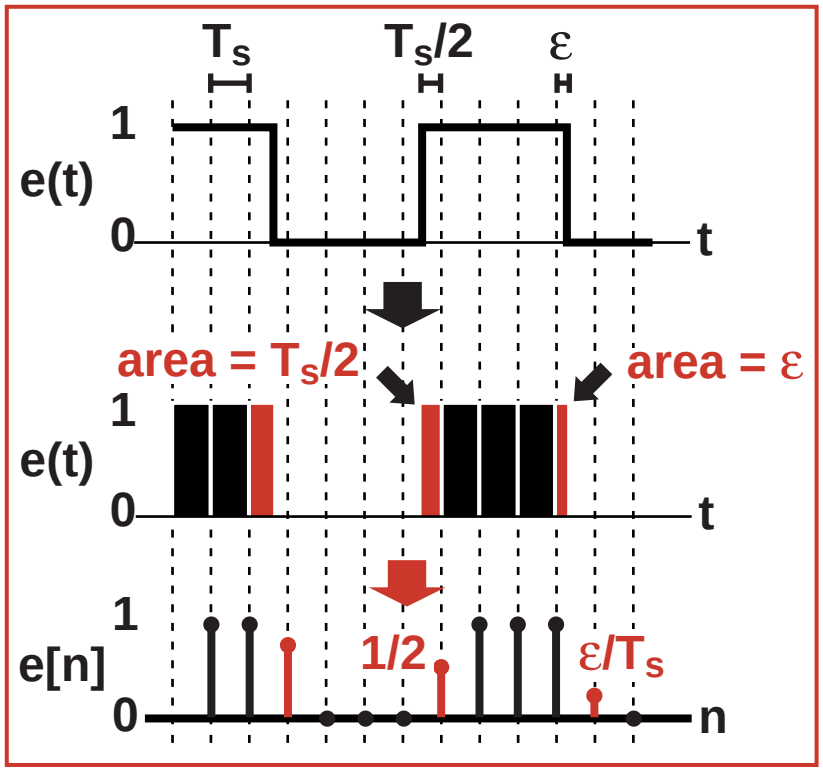
Proposed Approach: View as Series of Pulses

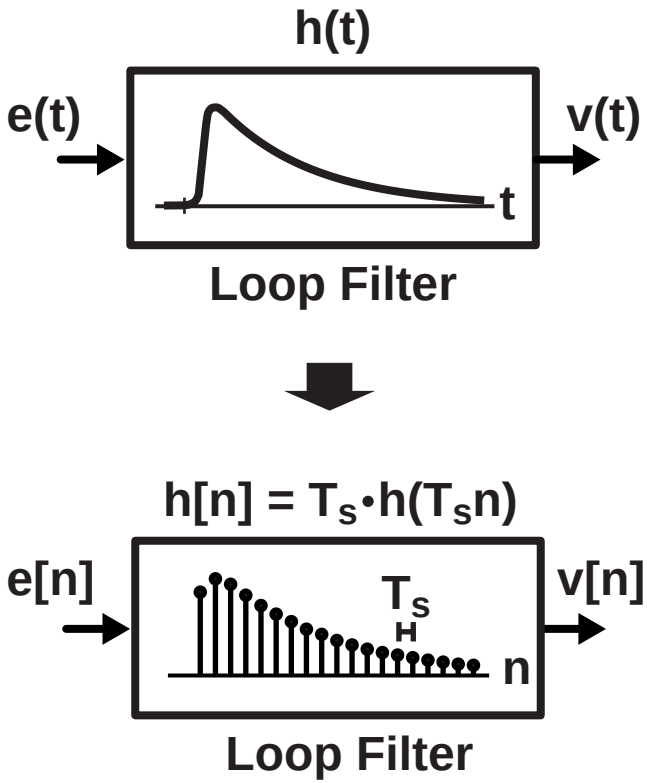




- Area of each pulse set by edge locations
- Key observations:
  - Pulses look like impulses to loop filter
  - Impulses are parameterized by their area and time offset

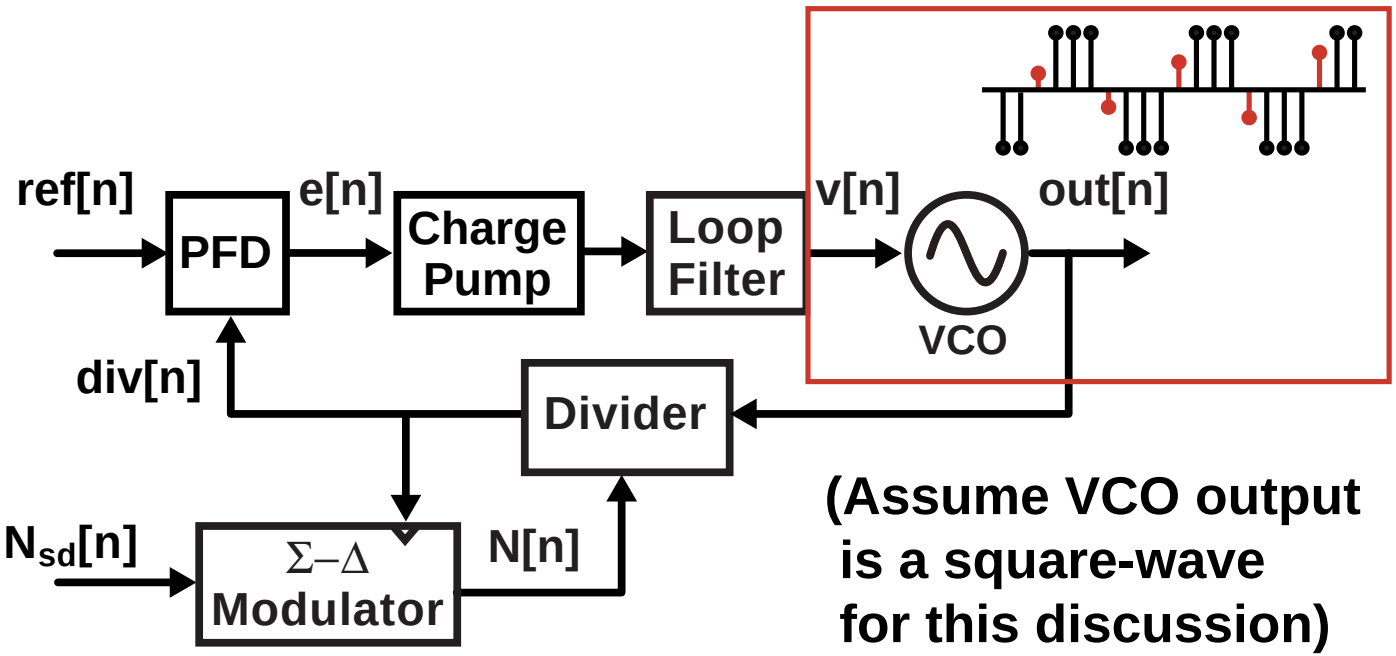
Proposed Method





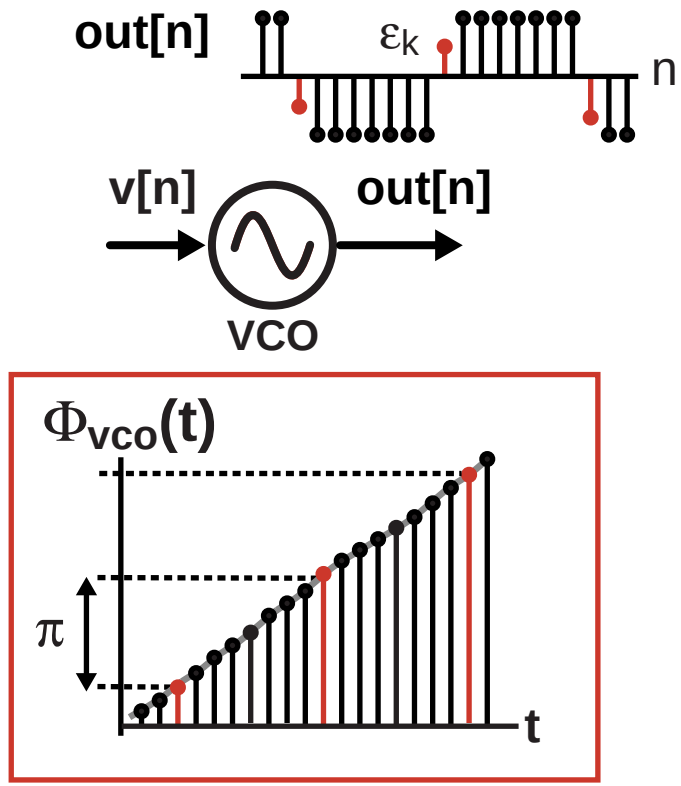
- Set e[n] samples according to pulse areas
  - Leads to very accurate results
    - Mathematical analysis given in paper
  - Fast computation

Implementation Overview



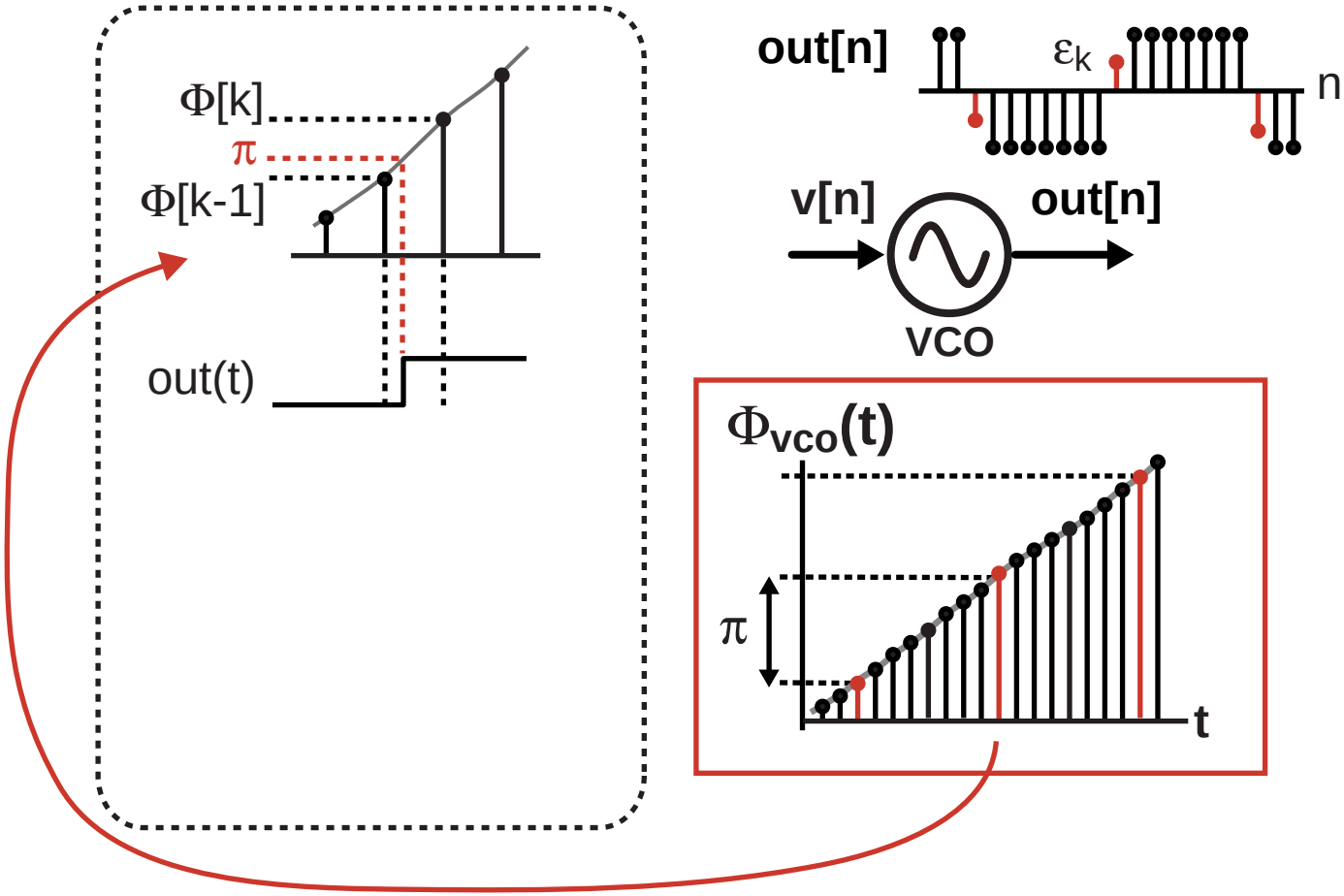
- Compute transition values in VCO block

Calculation of Transition Values



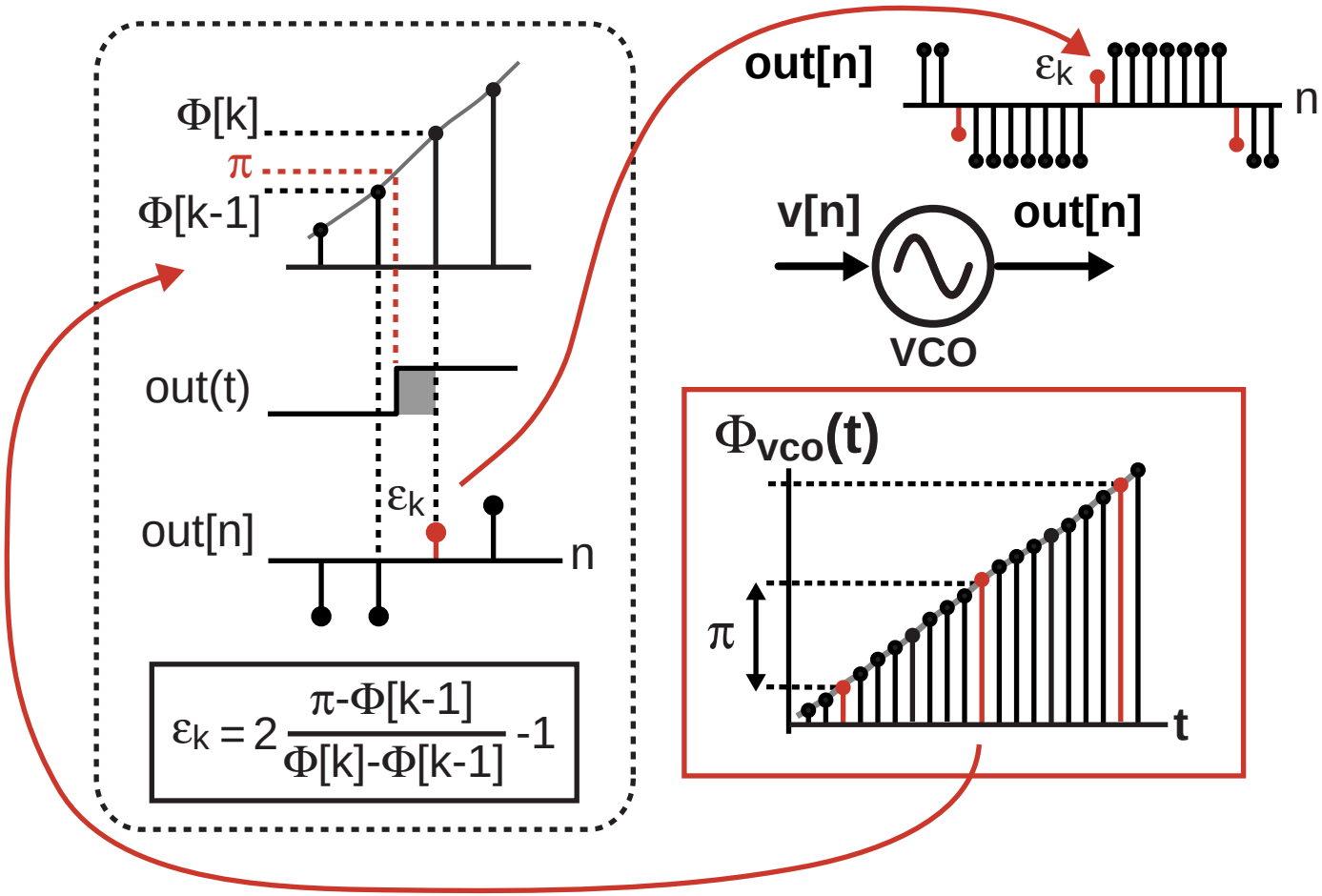
- **Model VCO based on its phase**

Calculation of Transition Values



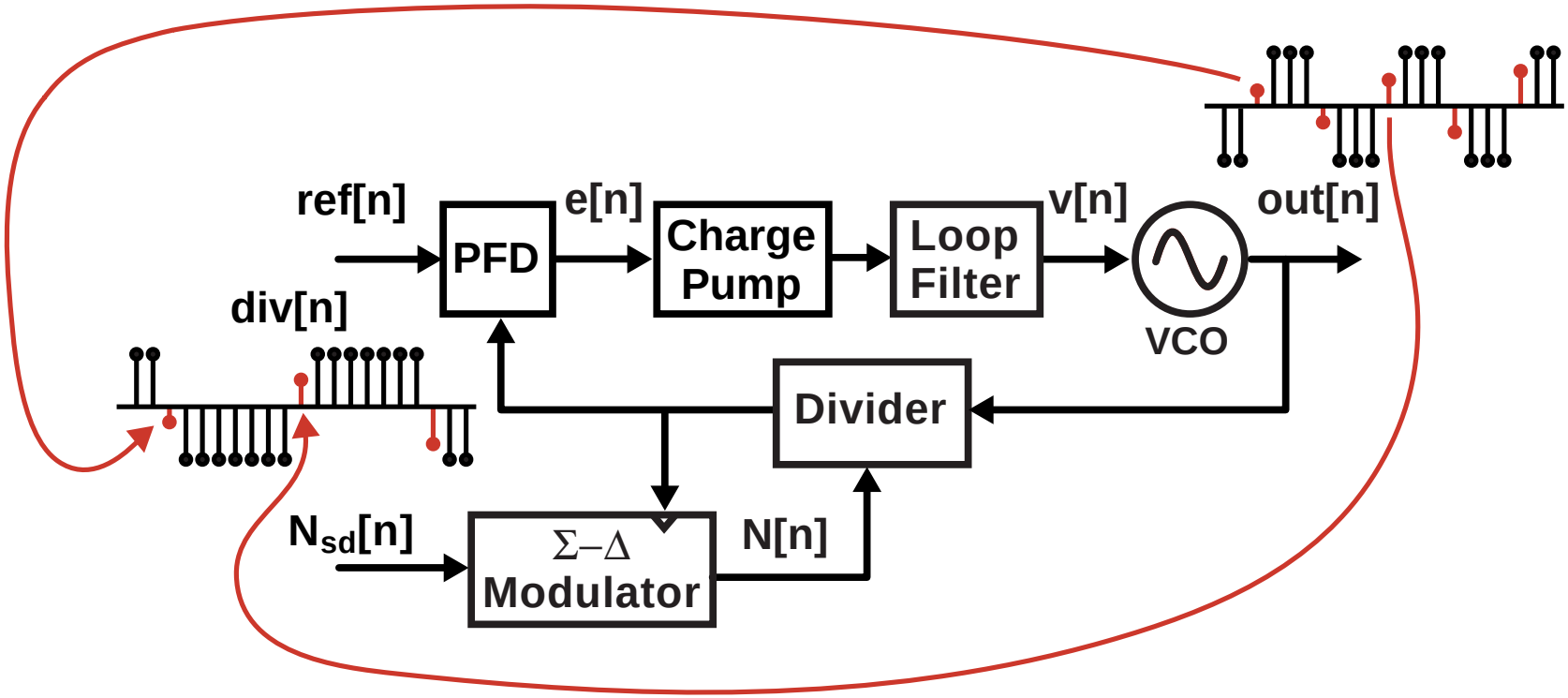
- **Determine output transition time according to phase**

Calculation of Transition Values



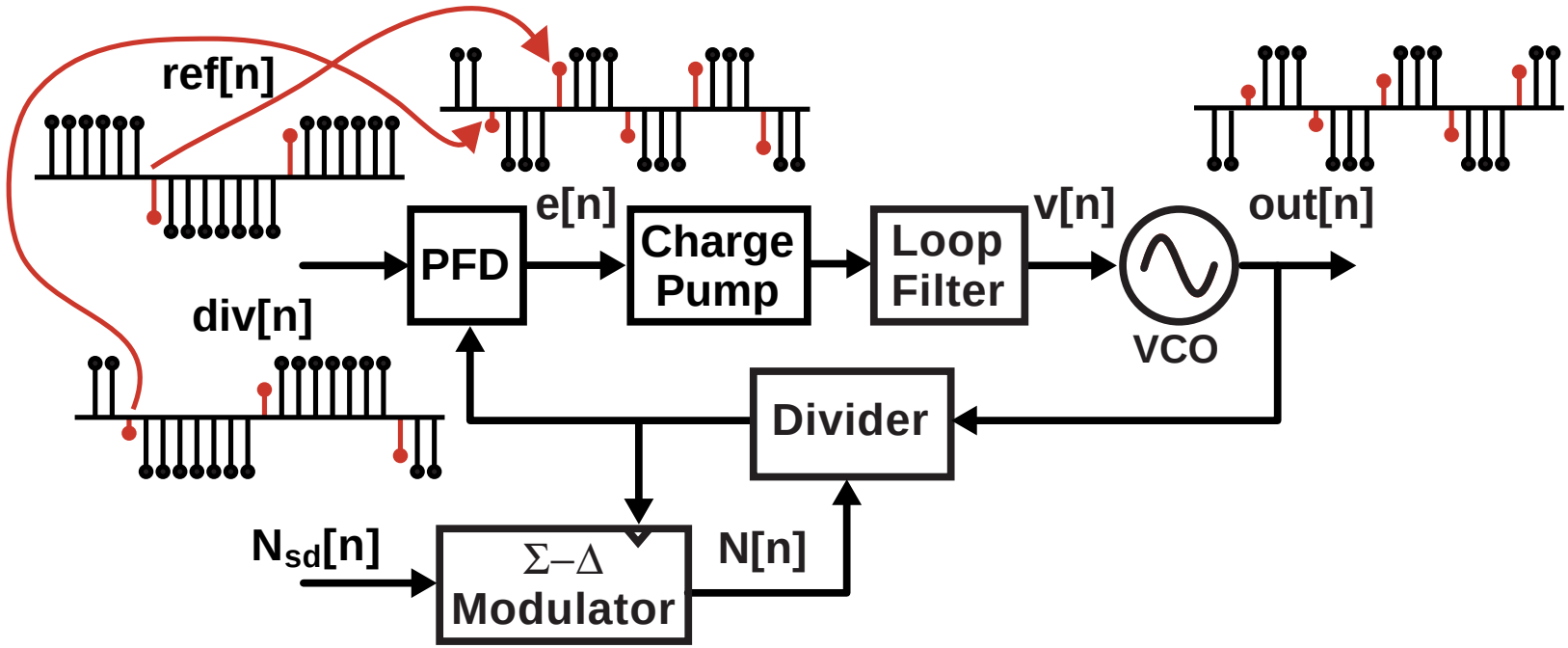
- **Use first order interpolation to determine transition value**

Implementation Overview



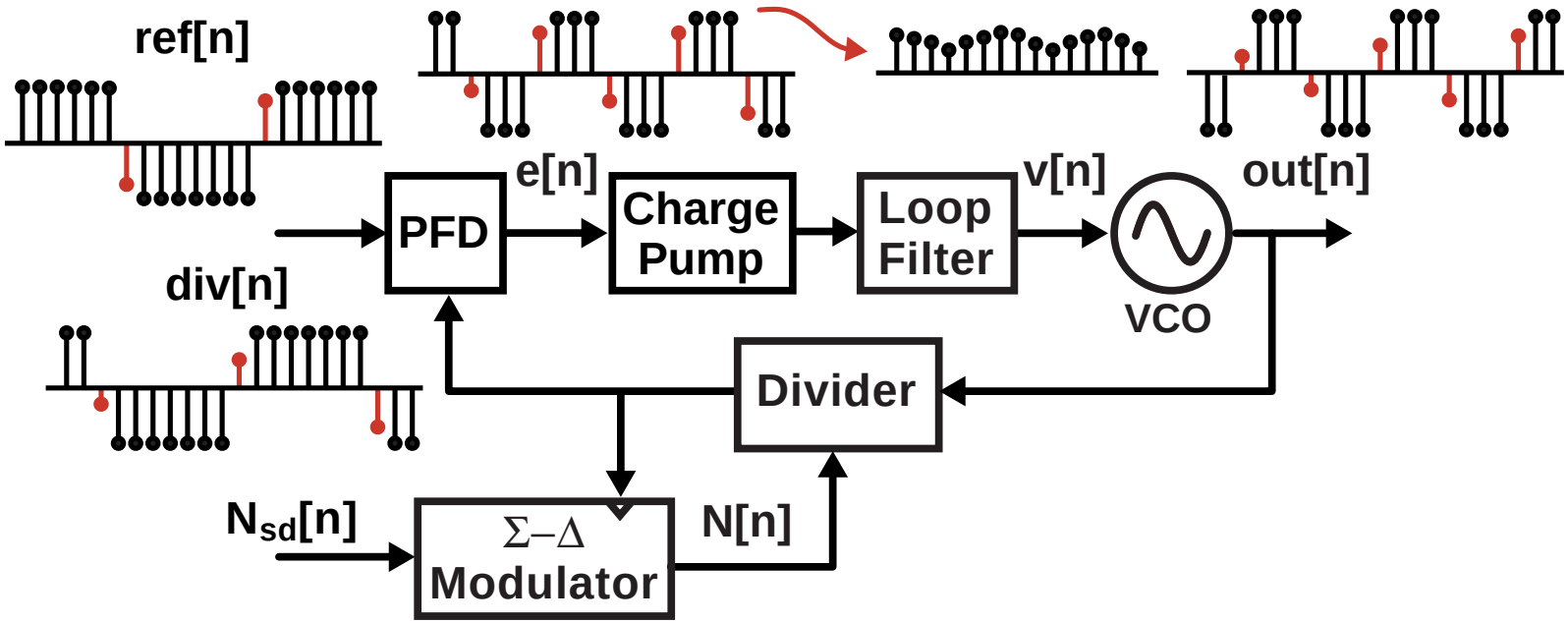
- Compute transition values in VCO block
- Pass transition information in Divider block

Implementation Overview



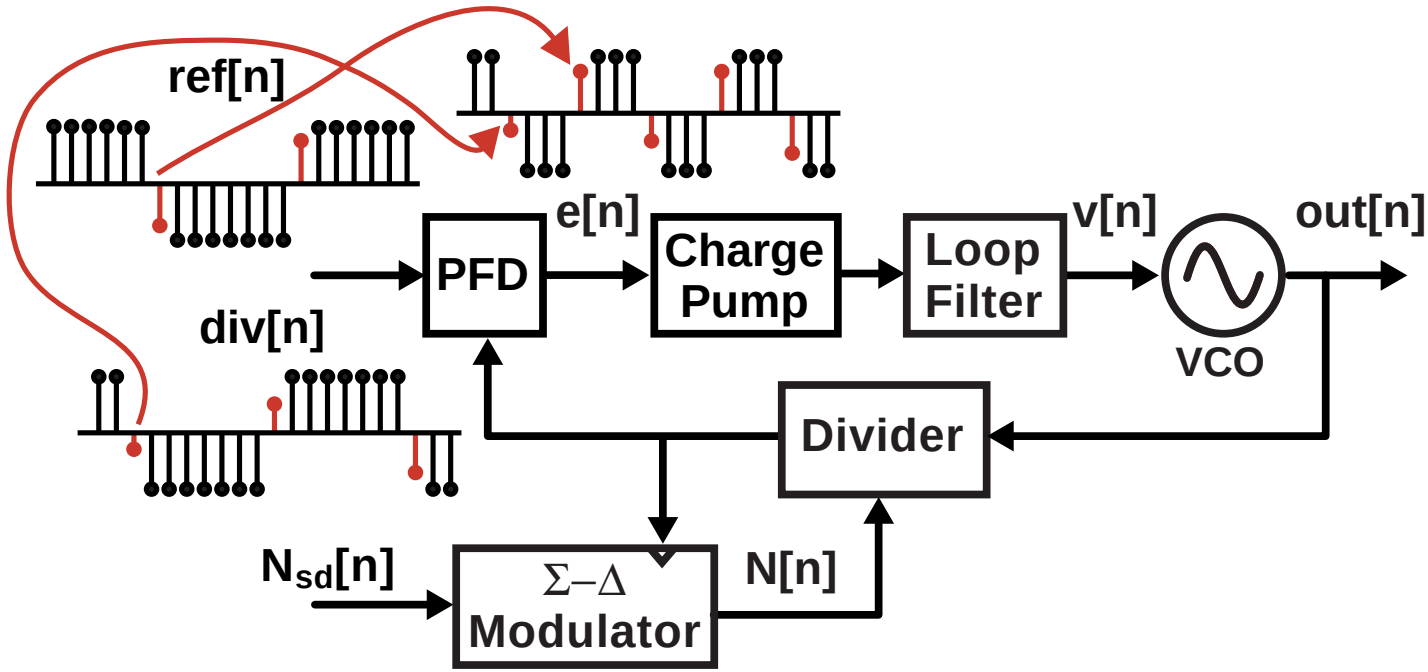
- Compute transition values in VCO block
- Pass transition information in Divider block
- Compute transition values for PFD output

Implementation Overview



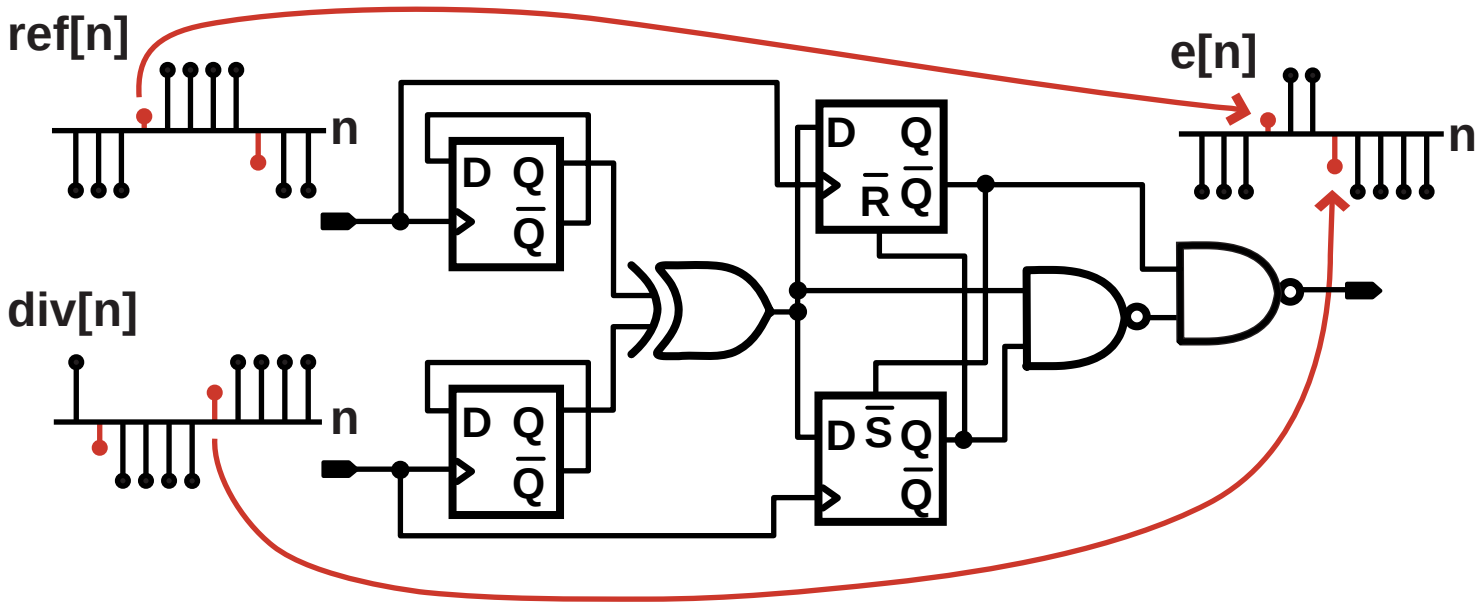
- Compute transition values in VCO block
- Pass transition information in Divider block
- Compute transition values for PFD output
- Compute Filter output

Implementation Overview



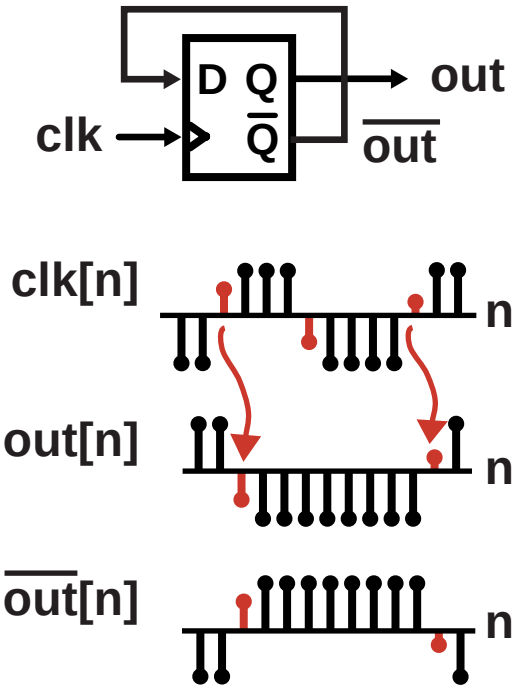
- Compute transition values in VCO block
- Pass transition information in Divider block
- Compute transition values for PFD output
- Compute Filter output

Computation of PFD Output



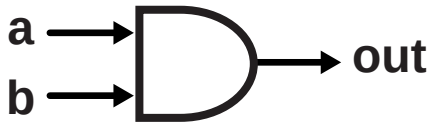
- **Goal: compute transition information in terms of primitive blocks (registers, XOR gates, etc.)**
  - **Allows straightforward implementation in simulator**
  - **Accommodates a rich variety of PFD structures**

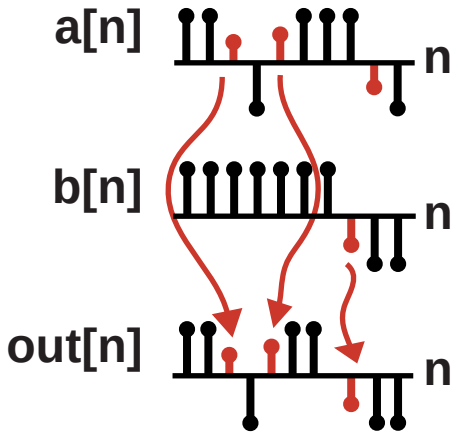
Implementation of Primitives - Registers



- Relevant timing information is contained in the clock signal
  - Transfer transition information from the clock to the register output
  - Complement output using a sign change

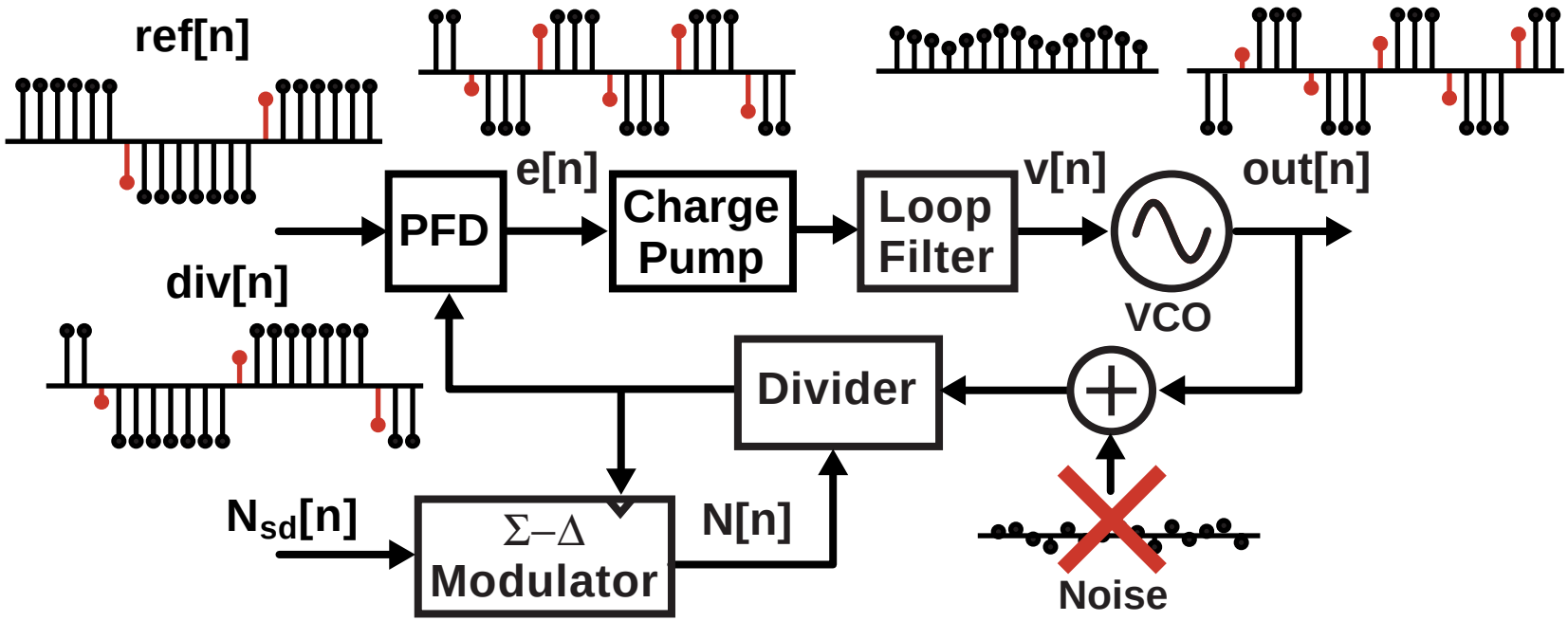
Implementation of Primitives – Logic Gates





- Relevant timing information contained in the input that causes the output to transition
  - Determine which input causes the transition, then pass its transition value to the output

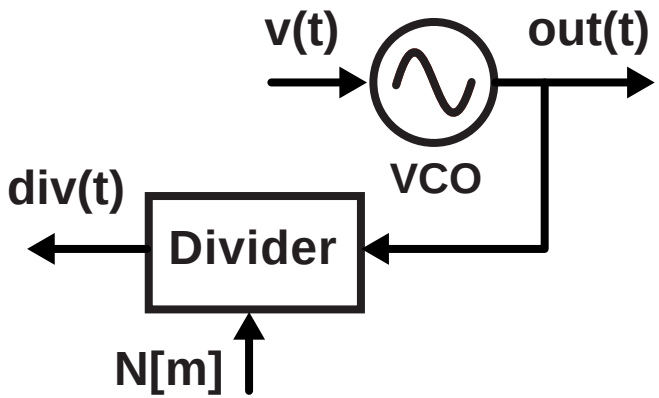
Issue: Must Observe Protocol When Adding Noise



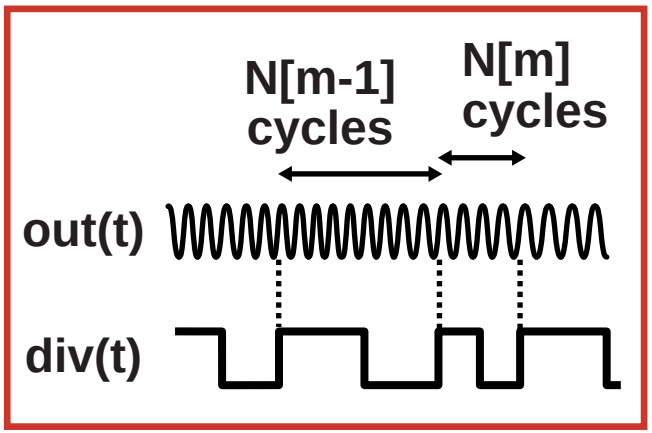
- Divider and PFD blocks operate on a strict protocol for their incoming signals
  - Values other than 1 or -1 are interpreted as edges
  - Example: inputting noise at divider input breaks protocol!
- Add noise only at places where signal is "analog"
  - PFD, charge pump, and loop filter outputs are fine

Can we speed the simulation up further?

Sample Rate Set by Highest Frequency Signal

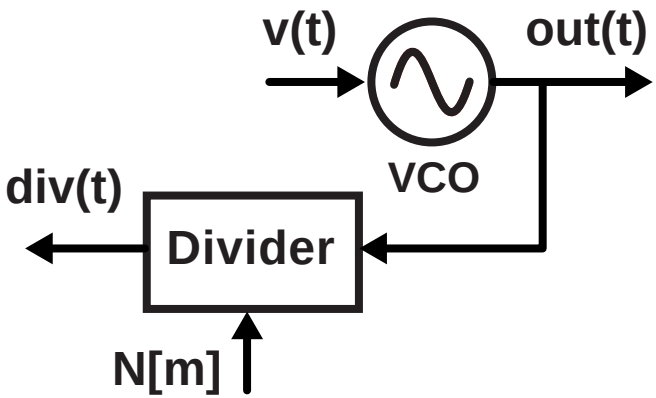


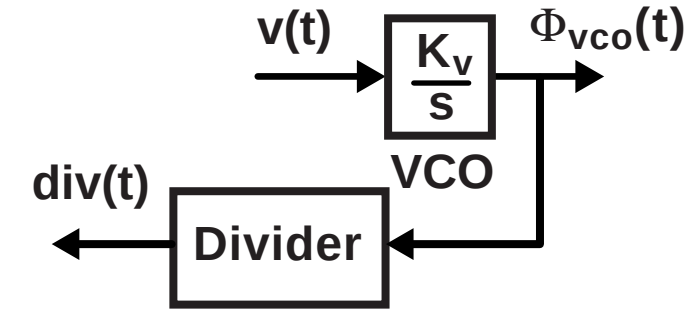
- Time step of simulation typically set by VCO output
- Small time steps means long simulation runs
- Divider output often 100 times lower in frequency

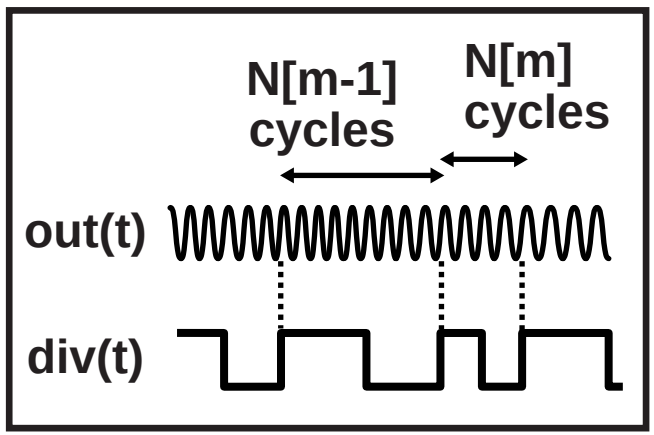


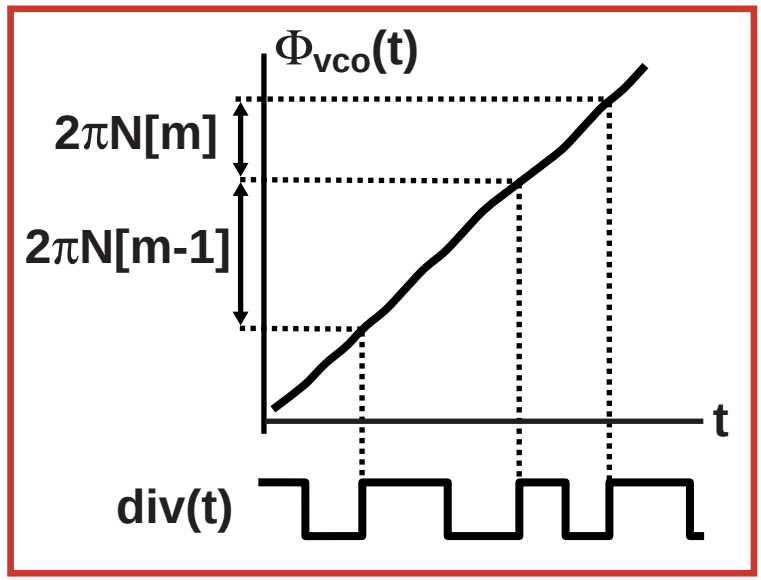
Can we sample according to divider output?

Divider Output Can Be Computed from VCO Phase







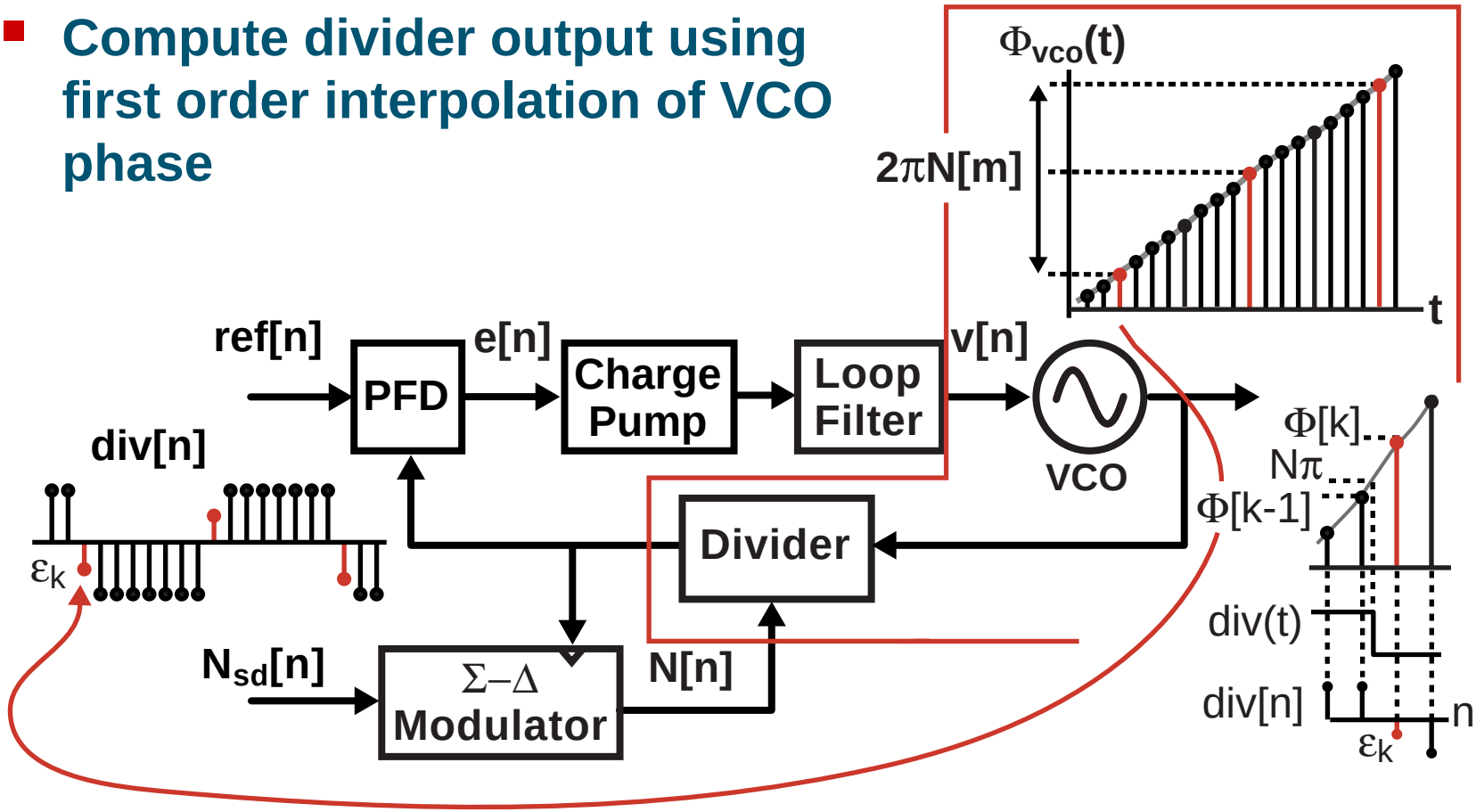


(Van Halen et al, Circuits and Systems '96)

Key Idea: Model VCO and Divider using Phase

Combine VCO and Divider Blocks

- Compute divider output using first order interpolation of VCO phase



Transient simulations run 2 orders of magnitude faster!

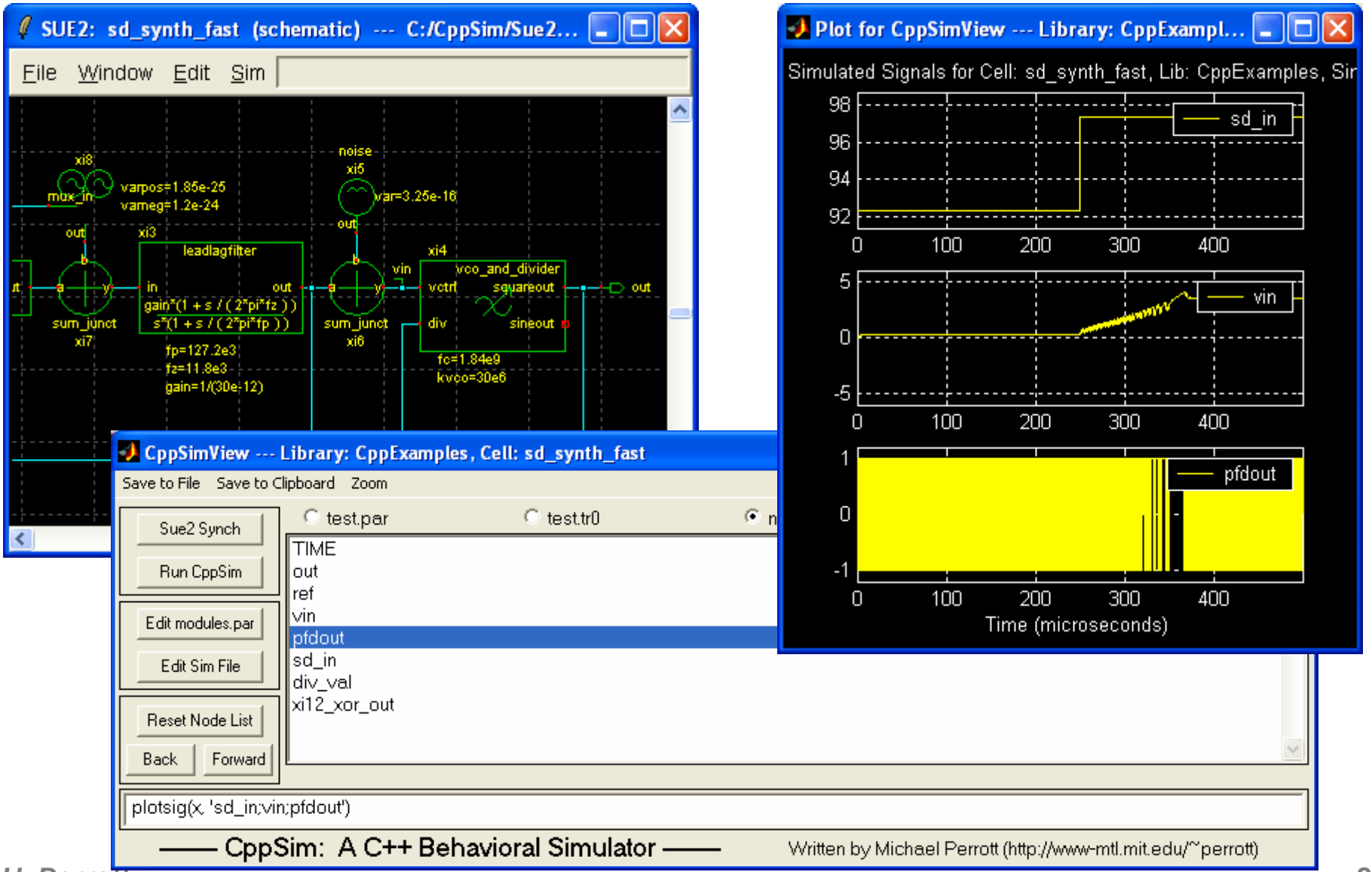
Does it really work?

The CppSim Simulator

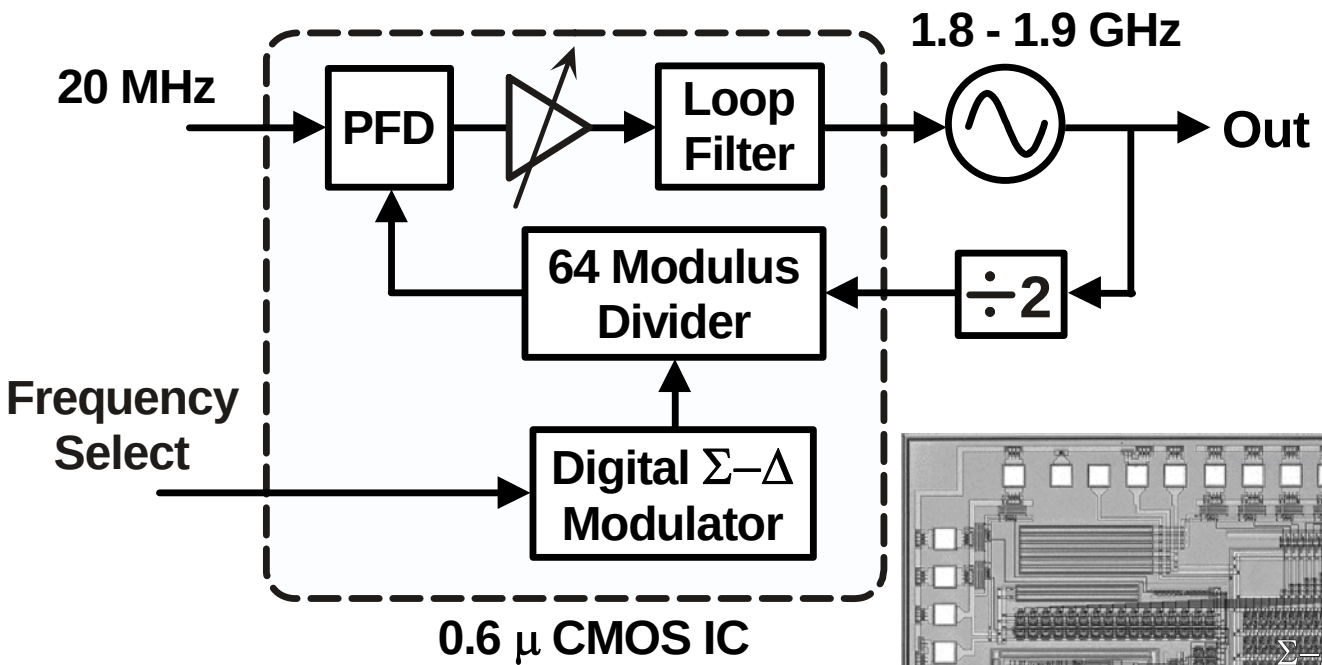
- **Blocks are implemented with C/C++ code**
  - **High computation speed**
  - **Complex block descriptions**
- **Users enter designs in graphical form using Cadence or Sue2 schematic capture**
  - **System analysis and transistor level analysis are possible in the same CAD framework**
- **Resulting signals are viewed in Matlab or CppSimView**
  - **Powerful post-processing and viewing capability**

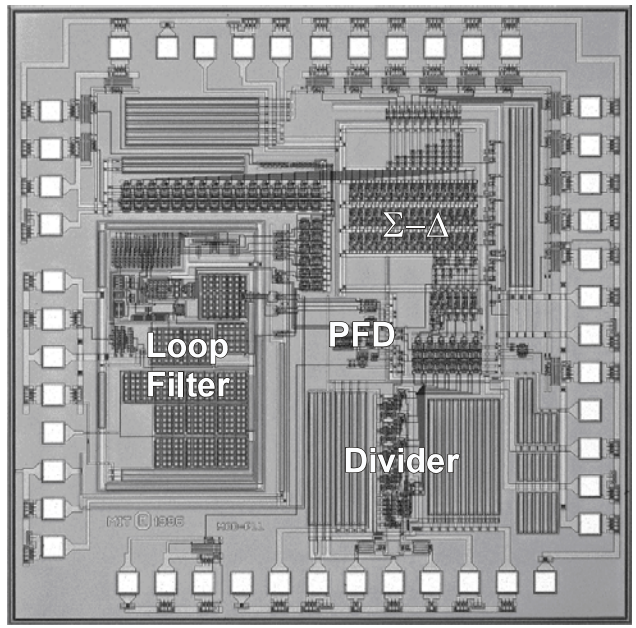
Simulation package freely downloadable at http://www.cppsim.com

The Sue2 and CppSimView Environment



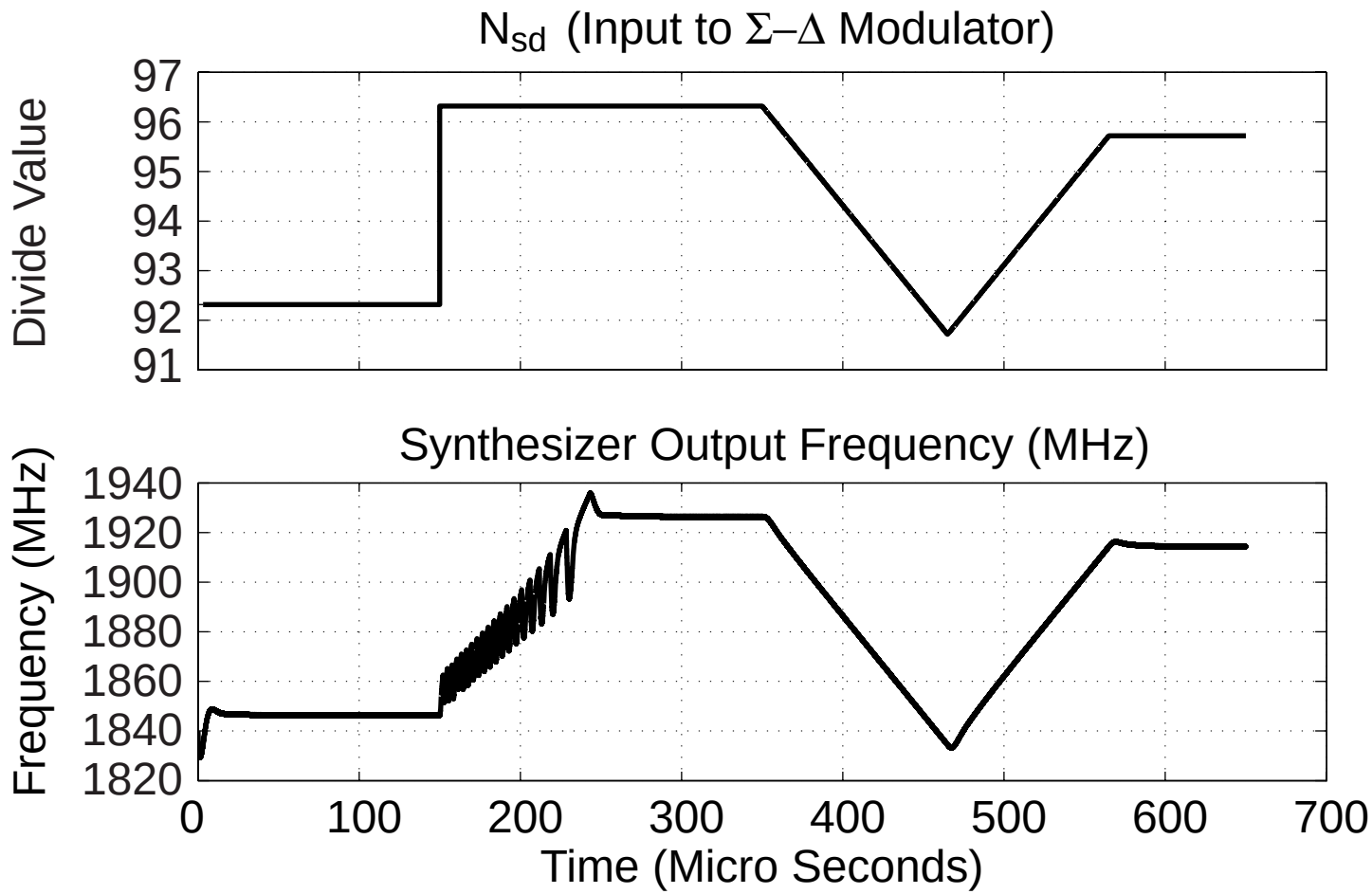
Experimental Prototype to Verify Approach





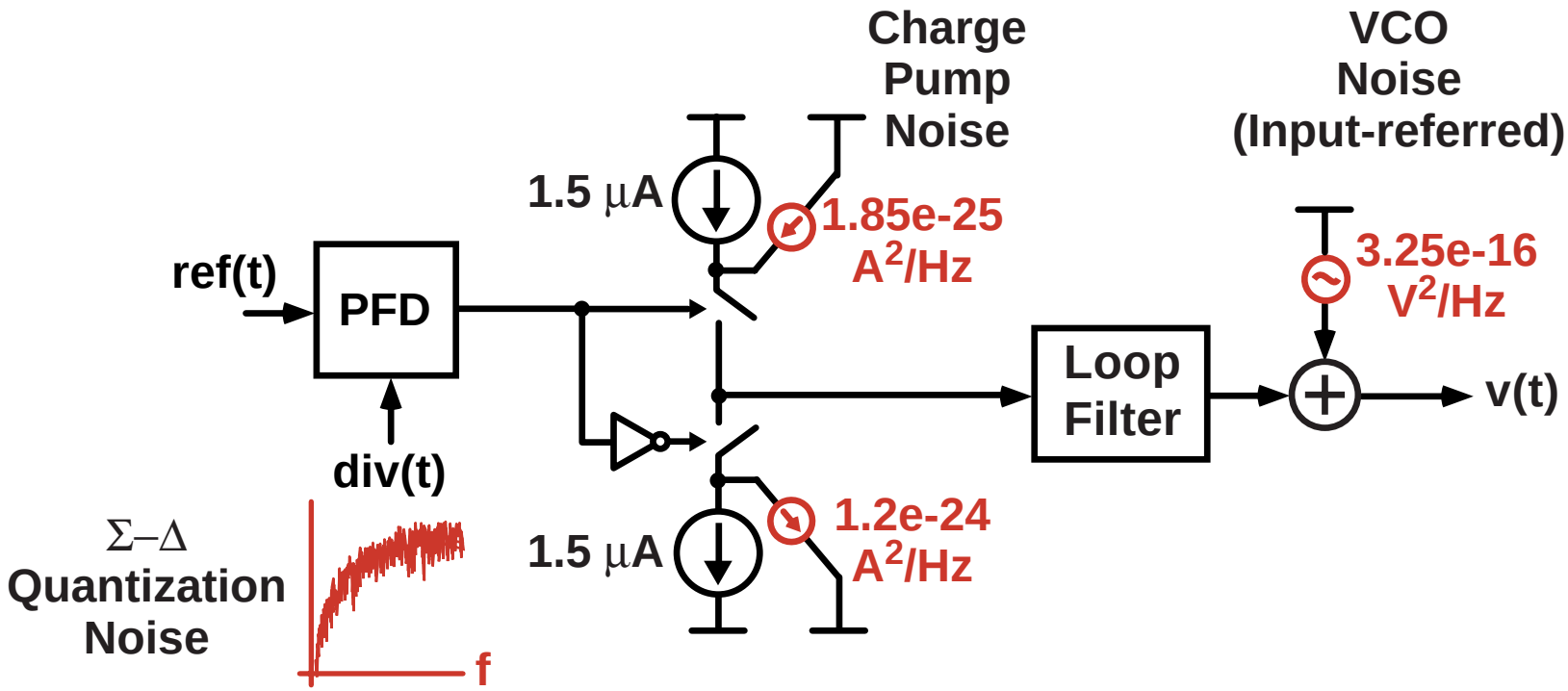
Perrott et al
JSSC, Dec 97

Simulation Results - Dynamic Behavior



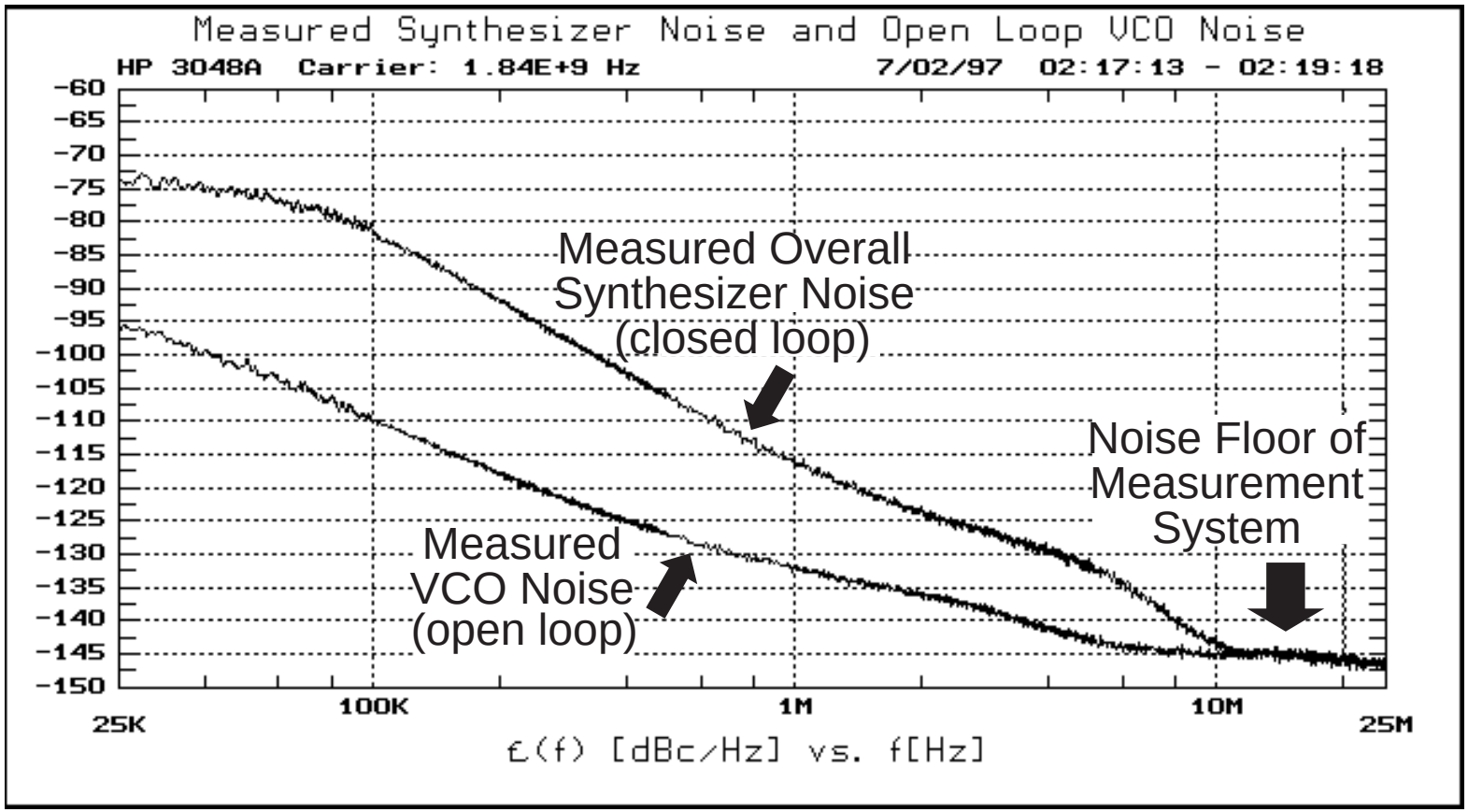
- **Simulation time: 260 thousand time steps in 5 seconds on a 650 MHz Pentium III Laptop (custom C++ simulator)**

Noise Sources Included in Simulation

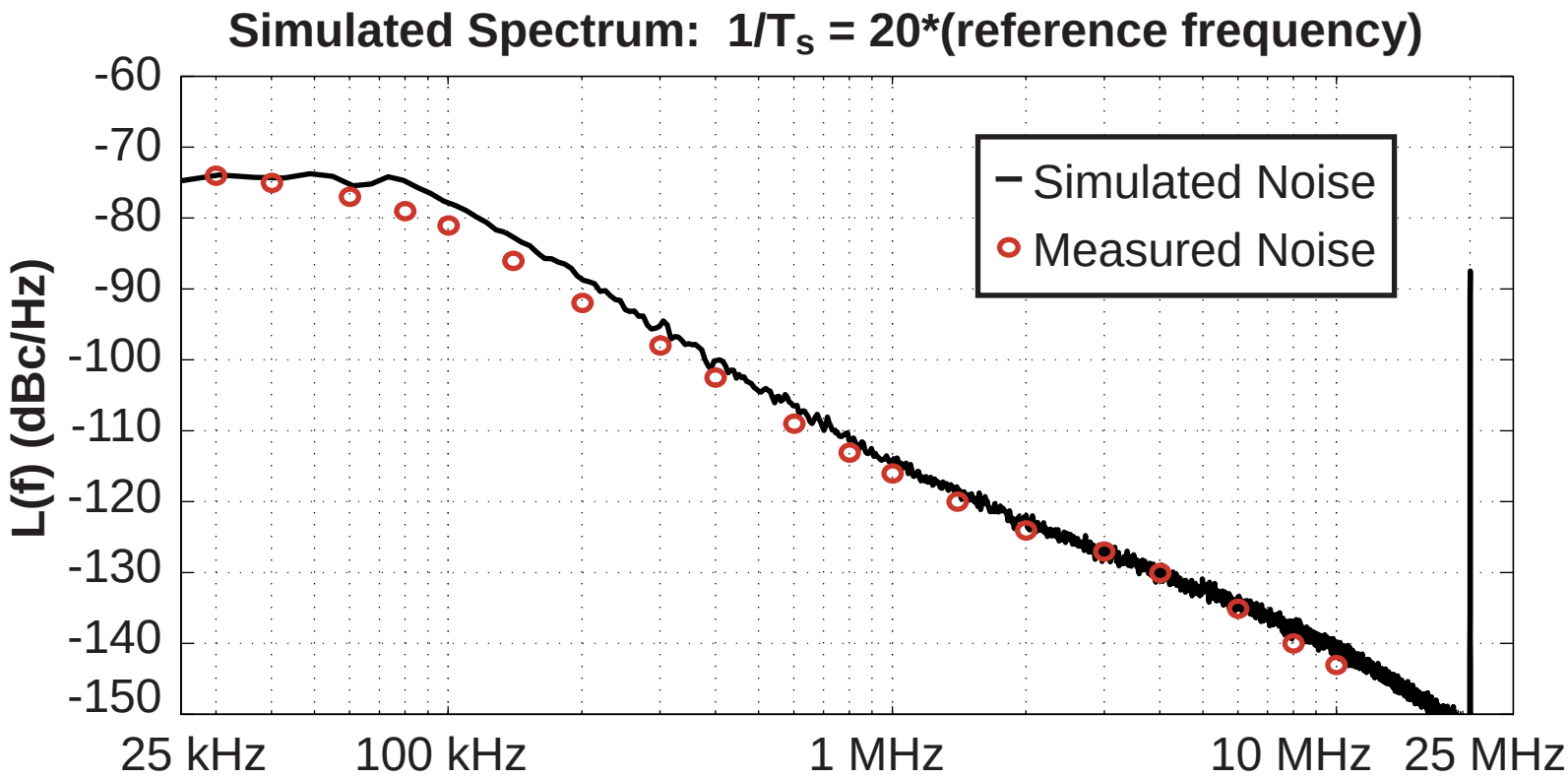


- Dominant noise sources in synthesizer
  - Quantization noise of Σ–Δ (produced by Σ–Δ block)
  - Charge pump noise (calculated from Hspice)
  - VCO noise (input-referred – calculated from measurement)

Measured Synthesizer Noise Performance



Simulated Synthesizer Noise Performance



- **Simulated results compare quite well to measured!**
- **Simulation time: 5 million time steps in 80 seconds**

Conclusion

- **Phase locked loop circuits can be quickly and accurately simulated**
  - **Accuracy achieved with area conservation principle**
  - **Fast computation by combining VCO and Divider blocks**

- **A variety of simulation frameworks can be used**
  - **C++, Matlab, Verilog**
  - **Circuit primitives are supported**

Noise and dynamic performance of fractional-N frequency synthesizers can be investigated at system level