Ultra-Low Power PLL for Wake-up Receiver Applications

Specialization Project Progress - 9th Week

Cole Nielsen
Department of Electronic Systems, NTNU

18 October 2019 (Calendar week 43)

Old Timeline

Week	Dates	Tasks	Outcomes
36	2.9 - 8.9	Review PLL Design	Refreshed Knowledge
37	9.9 - 15.9	Modeling/simulation (set up)	–
38	16.9 - 22.9	Modeling/simulation	TDC/DCO Requirements
39	23.9 - 29.9	Modeling/simulation	Loop Filter/Digital Algorithms
40	30.9 - 6.10	Modeling/simulation	Loop filter, DCO, TDC, calibration
41	7.10 - 13.10	Circuit Research	DCO/Divider topologies, Ideal Virtuoso implementation
42	14.10 - 20.10	Circuit Research	TDC/other topologies
43	21.10 - 27.10	Circuit Implementation	Digital logic (schematic)
44	28.10 - 3.11	Circuit Implementation	DCO (schematic)
45	4.11 - 10.11	Circuit Implementation	Divider/other (schematic)
46	11.11 - 17.11	Circuit Implementation (TDC)	
47	18.11 - 24.11	Circuit Implementation (TDC)	TDC (schematic)
48	25.11 - 1.12	Full Circuit testing	Testbenches, find bugs, design fixes
49	2.12 - 8.12	Full Circuit testing	Design Fixes/iteration
50	9.12 - 15.12	–	–

Legend: **Done** **Current** **Revised**

New Timeline

Week	Dates	Tasks	Outcomes
36	2.9 - 8.9	Review PLL Design	Refreshed Knowledge
37	9.9 - 15.9	Modeling/simulation (set up)	–
38	16.9 - 22.9	Modeling/simulation	TDC/DCO Requirements
39	23.9 - 29.9	Modeling/simulation	Loop Filter/Digital Algorithms
40	30.9 - 6.10	Modeling/simulation	Loop filter, DCO, TDC, calibration
41	7.10 - 13.10	Circuit Research	DCO/Divider topologies
42	14.10 - 20.10	Circuit Research	TDC/other topologies
43	21.10 - 27.10	Spur analysis, filter automation	
44	28.10 - 3.11	Filter automation, variance analysis	DSP round-off optimization
45	4.11 - 10.11	Variation analysis, flicker noise	Histograms/yield estimates
46	11.11 - 17.11	Real DCO sensitivity, TDC/divider jitter	Simlate ring-DCO in Virtuoso
47	18.11 - 24.11	PLL + Radio simulation	BER estimate
48	25.11 - 1.12	Agglomerate into cohesive framework	(I have an Exam on 30.11)
49	2.12 - 8.12	Finish framework, report writing	
50	9.12 - 15.12	Report writing	Complete before 15.12

Legend: **Done** **Current** **Revised**

(New) Timeline Tasks

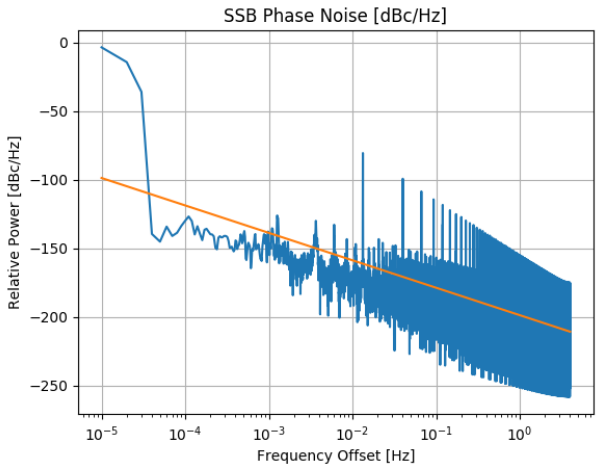
This week

- **Spur analysis:**
  - Implemented phase noise reference spur analysis.
- **Filter automation**:
  - Now that project is a a PLL simulation framework, improve filter design
  - Automatic filter design was lacking
    - Previously required manual selection of gain parameter, so not fully automated
    - Now fully automated.
  - Improve resemblence of generated filter to prototype filter.
- **Improved code-reusability**:
  - Previously used several mostly independent scripts, now all use same core code for simulating PLL.
  - PLL core code configured with dictionary.

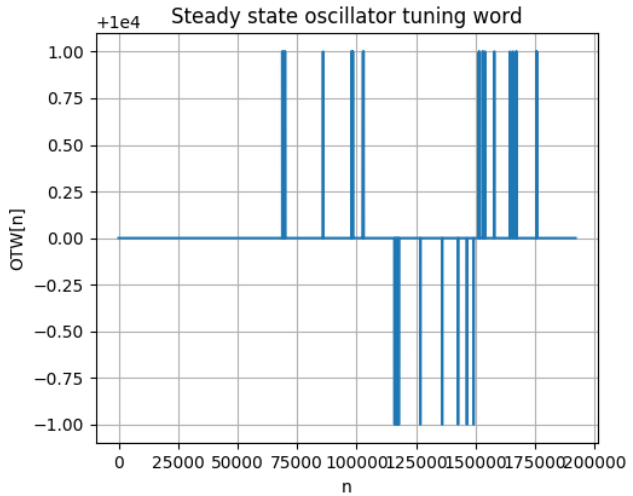
Reference spur analysis

Reference spur simulation

- Due to discrete time/tuning nature of DCO, spurs are produced in the phase noise spectrum at offsets from the carrier that are multiples of the reference frequency.
- In steady state, relatively few changes occur in oscillator tuning word (ca. 1 LSB in 100's or 1000's of reference cycles).
- Time between OTW changes in steady state is relatively stochastic compared to reference cycle frequency, so spurs should be quite small in these conditions.



(a) Severe spurs.



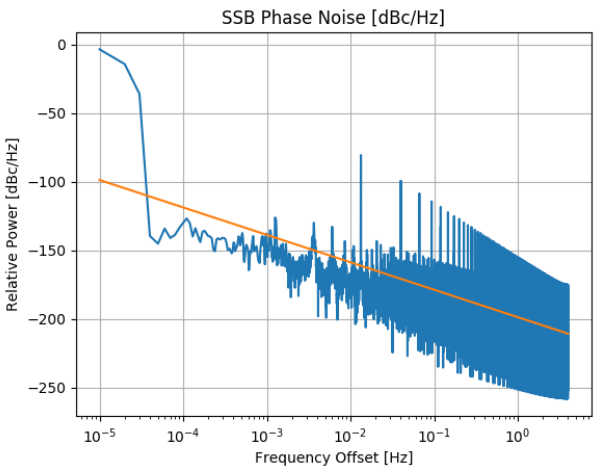
(b) OTW in steady state.

Reference spur analysis

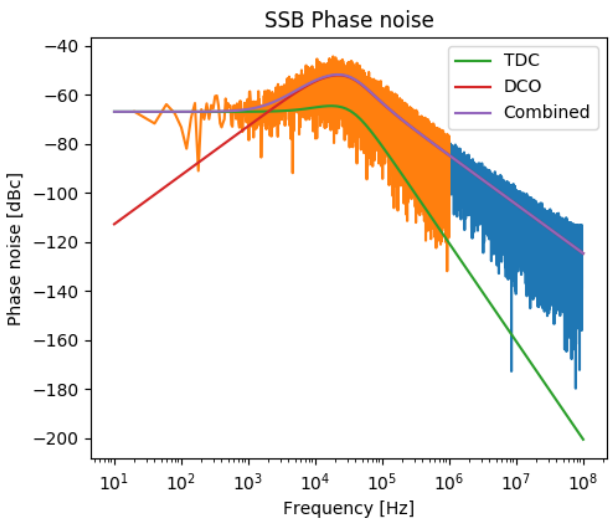
Simulation

— Had to rewrite phase domain simulation to support oversampling. Now simulate twice:

- Once sampled at the reference frequency with long span to capture low frequency phase noise.
- Once oversampled sampled with short span to capure reference spurs.
- Results are combined, with about 10x faster run-time than a single simulation capturing the same frequency range in plot (b) below.



(a) Severe spurs.



(b) Simulated PLL spurs (none).

Reference spur analysis

Simulation

— It does not appear spurs will be an issue in this PLL design. They are not measureable under the current assumed PLL design parameters.

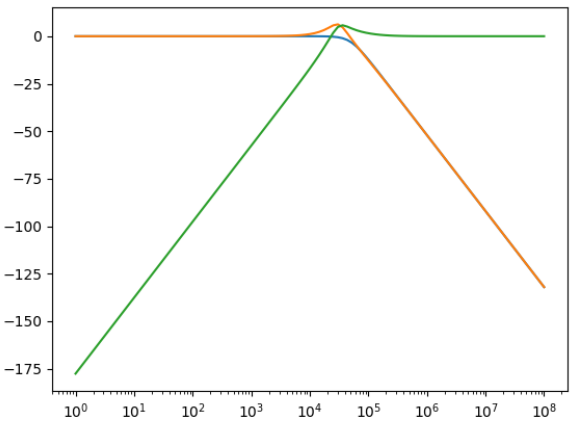
— Generally, the DCO resolution should be high enough and the rate of change of the OTW is small enough where not much power is pushed into the reference spurs for an integer-N PLL design.

Filter automation

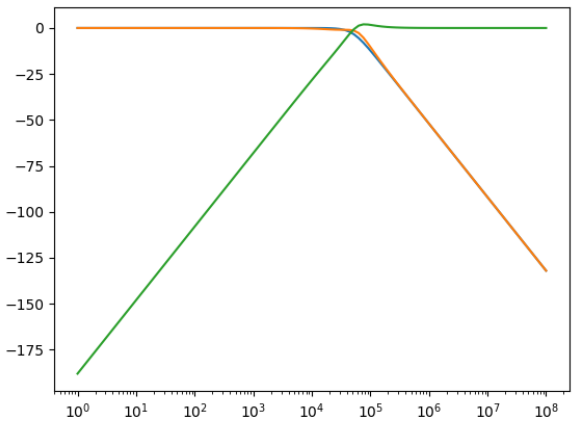
Deficiencies

— Old automated filter design was a "quick and dirty" approach.

- High/low frequency response was not well optimized. At frequencies near the loop bandwidth frequency, peaking was not well controlled.
- Most phase noise power is integrated in frequencies near to the loop bandwidth frequency, so excessive peaking is a bad problem.



(a) Optimized by old method.



(b) Optimized by new method.

Filter automation

New approach

— Old approach optimized the following PLL open loop transfer function for poles/zero and parameter K (put into closed loop configuration):

$$A(s) = \frac{K}{s^2} \cdot \frac{(s/\omega_z + 1)}{(s/\omega_p + 1)} \tag{1}$$

— The parameter K has very a small rate of change compared to the pole/zero frequencies, so in gradient descent little optimization occurs to K.

— If the desired prototype transfer function is:

$$G(s) = \frac{\omega_n^2}{s^2 + 2\zeta\omega_n s + \omega_n^2} \tag{2}$$

It turns out for the tail roll-off to be the same at high frequency:

$$K = \frac{\omega_z \omega_n^2}{\omega_p} \tag{3}$$

Filter automation

New approach

— Now to find the optimized loop filter parameters, a loop is run where:
  - The pole/zero are optimized with gradient descent for a fixed K.
  - The aforementioned equation for K is used to update K based on the optimized pole/zero

— The above is repeated until convergence.

Code-resuability

More generic PLL simulation engine

— Made generic engine to simulate integer-N PLL, configured with a dictionary.

— Less code fragmentation everything now uses same core.

```
################################################################################
# Main phase noise simulation
################################################################################

print("\n*********************************************")
print("* Running main phase noise simulation         *")
print("*********************************************")
SIM_STEPS = int(round(FS/FBIN)) # make simulation long enough to get desired fbin
print("\n* Simulation samples = %E Sa"%SIM_STEPS)
print("\t Fs = %E Hz"%FS)
print("\t Fbin = %E Hz"%FBIN)

KRWRO = ro_rw_model_param(f0=FOSC, power=RO_POWER, temp=TEMP, n=SIM_STEPS, tstep=1.0/FS)

main_sim_params = {
    "fclk"        : FCLK,
    "fs"          : FS,
    "sim_steps"   : SIM_STEPS,
    "div_n"       : DIV_N,
    "tdc_steps"   : TDC_STEPS,
    "kbbpd"       : KBBPD,
    "bbpd_tsu"    : BBPD_TSU,
    "bbpd_th"     : BBPD_TH,
    "kdco"        : KDCO,
    "fl_dco"      : FL_DCO,
    "krwro_dco"   : KRWRO,
    "lf_i_bits"   : INT_BITS,
    "lf_f_bits"   : FRAC_BITS,
    "lf_params"   : lf_params,
    "init_params" : {
        "osc"   : 0,
        "clk"   : 0,
        "lf"    : LF_INIT,
        "div"   : 0,
        "tdc"   : 0,
        "bbpd"  : 1,
        "error" : 0,
        "kbbpd" : 0.25,
    },
}

main_pn_data = pllsim_int_n(**main_sim_params)
```

Specification (unchanged)

System Performance Targets

Parameter	Value	Unit	Notes
Frequency	2.4-2.4835	GHz	2.4G ISM Band
Ref. frequency	16	MHz	Yields 6 channels
Power	$\leq$ 100	μW	
FSK BER	$\leq$ 1e-2		2FSK with f_{dev}=$\pm$250 KHz
Initial Lock Time	$\leq$ 50	μs	Upon cold start
Re-lock Time	$\leq$ 5	μs	Coming out of standby
Bandwidth	50	kHz	(nominally), tunable

Additionally: PLL output should support IQ sampling at LO frequency.

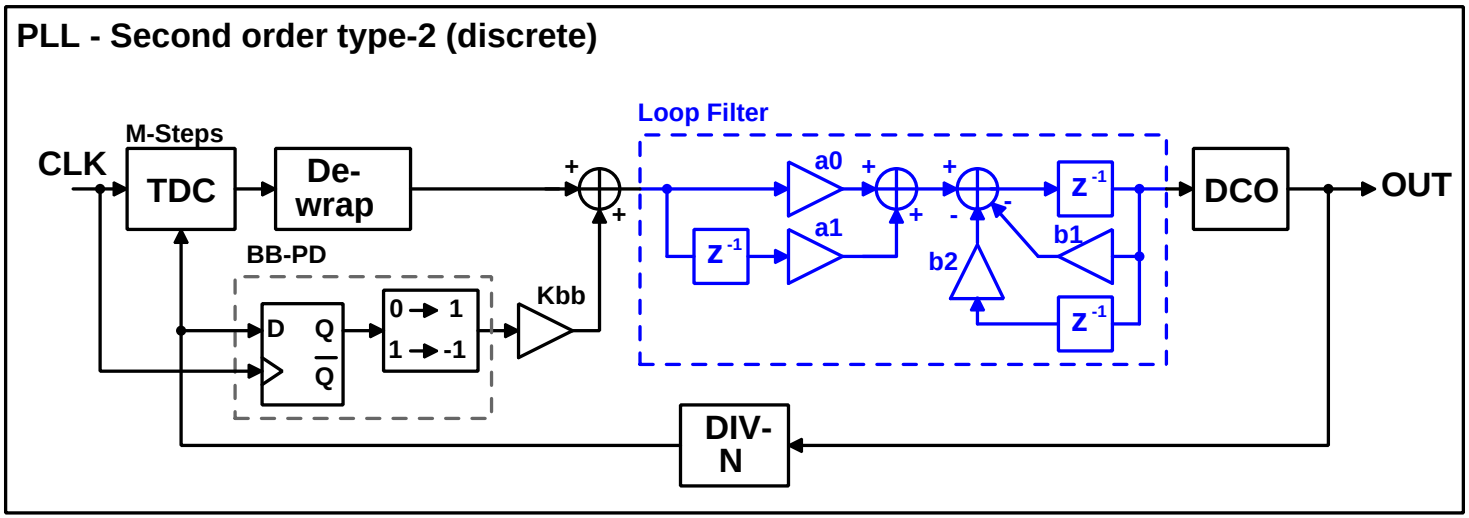
Specification (unchanged)

PLL Component Performance Targets

Parameter	Value	Unit	Notes
DCO LSB Resolution	≤ 50	kHz	Determined from quantization noise.
DCO DNL	< 1	LSB	Ensures monotonicity
TDC Resolution	0.95	ns	
TDC Resolution (bits)	6	bits	

Architecture (updated)

Block Diagram



Power Targets

DCO	TDC	Divider	Other	SUM
70 μW	20 μW	10 μW	$<<$ 1 μW	100 μW

Project Phases

Autumn 2019

— System modeling and simulation.
  - Learn PLL theory in detail
  - Evaluate feasability of PLL architectures (counter, TDC-based)
  - Determine requirements for TDC/DCO/Divider/logic (bits of resolution, accuracy etc) to meet PLL performance specifications.
  - Determine digital logic for loop filter, validate stability and lock time performance.

— Research ultra-low power circuit topologies to implement system components that will meet determined requirements.

— Translate component-level specifications into schematic-level circuit designs.
  - Try, fail, try again until functional at schematic level.
    — I expect the TDC to be difficult.

Project Phases (continued)

Spring 2020

- Finalize schematic-level design.
- Establish thorough tests for PLL performance (automated?) to help in layout.
- Layout of PLL.
  - Design iteration until design specs met.
  - Probably very time consuming.
- Full characterization/validation of design performance.
  - Comprehensive Corners/Monte-Carlo testing (time consuming??)
  - More design iteration if new issues crop up...
- Thesis paper writing.

References

[1] "Ultra-Low Power Wake-Up Receivers for Wireless Sensor Networks", N. Pletcher, J.M Rabaey, 2008.

`http://www.eecs.berkeley.edu/Pubs/TechRpts/2008/EECS-2008-59.html`