An Automated and Process-Portable Generator for Phase-Locked Loop

Wang, Zhongkai; Choi, Minsoo ; Chang, Eric ; Wright, John; Bae, Wooham; Du, Sijun; Liu, Zhaokai ; Narevsky, Nathan; Schmidt, Colin; Biwas, Ayan

DOI
10.1109/DAC18074.2021.9586318

Publication date
2021

Document Version
Final published version

Published in
2021 58th ACM/IEEE Design Automation Conference, DAC 2021

An Automated and Process-Portable Generator for Phase-Locked Loop

Zhongkai Wang[1*], Minsoo Choi[2], Eric Chang[3], John Wright[1], Wooham Bae[4], Sijun Du[5], Zhaokai Liu[1],
Nathan Narevsky[1], Colin Schmidt[1], Ayan Biwas[1], Borivoje Nikolic[1*], Elad Alon[1*]
[1]University of California, Berkeley, CA, [2]Samsung Semiconductor, Inc., San Jose, CA
[3]Blue Cheetah Analog Design, San Francisco, CA, [4]Ayar Labs, Inc., Emeryville, CA
[5]Delft University of Technology, Delft, Netherlands, [*]{zhongkai, bora, elad}@berkeley.edu

2021 58th ACM/IEEE Design Automation Conference (DAC) | 978-1-6654-3274-0/21/$31.00 ©2021 IEEE | DOI: 10.1109/DAC18074.2021.9586318

***Abstract*—We present a bang-bang phase-locked loop (PLL) generator that encapsulates design methodologies for its circuit blocks and the complete PLL system. The generator is fully automated and parameterized, producing the layout and schematic based on process characterization and top-level specifications. Three 14GHz PLLs are instantiated in TSMC 16nm, GF 14nm and Intel 22nm technologies, demonstrating the process portability. The rapid generation time of less than four days enables fast PLL design and technology porting. The PLL design fabricated in TSMC 16nm shows RMS jitter of 565.4fs and power of 6.64mW from a 0.9V supply.**

***Index Terms*—phase-locked loop, Berkeley Analog Generator, phase noise, jitter, voltage-controlled oscillator**

I. Introduction

As new process technologies are developed, analog and mixed-signal (AMS) circuit design becomes increasingly complex and time-consuming. The design rule explosion at advanced technology nodes and increased sensitivity to routing parasitics with smaller feature sizes require repetitive design iterations. Additionally, reliability issues such as electromigration and dynamic voltage drop further lengthen the AMS circuit design cycle.

To address these concerns, the Berkeley Analog Generator (BAG) framework [1] was introduced, to enable fast AMS circuit designs using executable generators. The generators, built within a Python-based framework, can produce DRC- and LVS-clean schematics and layouts alongside verification test benches with parameterized specification files. In addition, the framework allows designers to codify their design procedure into design scripts, which enables schematic and layout generation, extraction, simulation, and resizing procedures into automatic design iteration loops, drastically reducing the design and validation time.

With the BAG framework, [2] and [3] demonstrate circuit design methodologies based on machine learning by converging circuit parameters to particular design specifications in a large design space using hundreds to thousands of simulations. While this procedure works effectively, the long convergence time limits both the size of the circuit and the number of design parameters. For this reason, the state-of-art ML-based design methodologies mainly focus on small-scale circuits, such as an operational amplifier and a simplified wireline receiver frontend. In designing large AMS circuits or systems, a traditional top-down or bottom-up design methodology is still preferred. By leveraging the experience of circuit designers, the number of iterations and the run time of the design script are drastically reduced.

In this paper, we propose a phase-locked loop (PLL) generator developed based on BAG using top-down and bottom-up approaches. The proposed PLL generator automatically produces a top-level schematic and layout of a PLL whose circuit parameters satisfy a target performance specification. The proposed generator is process-portable and proven in three different technology nodes from three different foundries. The total generation time took less than four days, showing that the proposed PLL generator enables fast PLL circuit design and technology portability. The simulated and measured performance of the generated PLL is comparable to one that is manually designed.

The rest of this paper is organized as follows. Section II briefly introduces the BAG framework and employed PLL architecture. Section III presents the schematic and layout generation of the PLL, its top-level analysis, and detailed design methodologies. Section IV summarizes the simulated and measured results. Section V presents the conclusion.

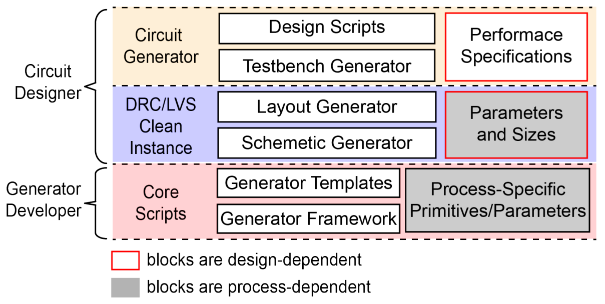


Fig. 1. The hierarchy of Berkeley Analog Generator.

II. Background

A. Berkeley Analog Generator

Fig. 1 shows BAG structure [1] split into three levels. In core scripts, the generator framework provides interfaces between

Python and commercial CAD tools (Virtuoso, Calibre, EMX, etc.). It also provides various functions to draw or modify layouts and schematics and utilizes a routing grid system with a track manager to set wire widths and spacing. With process-specific primitives and technology parameters, various layout templates are designed to generate DRC-clean analog CMOS, digital CMOS, resistor, MOM capacitor and inductor layouts.

By using generator templates and the track system, circuit designers develop process-independent schematic and layout generators and create DRC- and LVS-clean instances with parameters and sizes specific to a particular technology. At the top level, designers codify their design procedure into scripts that can generate testbenches and alter design parameters to meet the target performance specifications. With the framework, designers can easily implement circuits and systems in a different technology merely by updating the process-specific primitives and parameters.

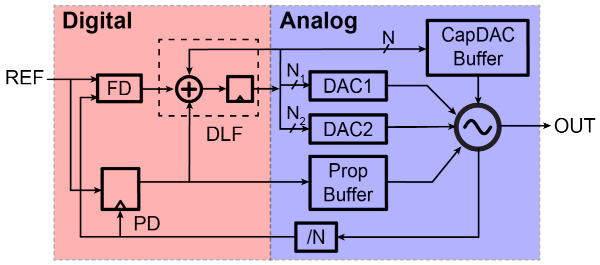


Fig. 2. Bang-bang PLL architecture.

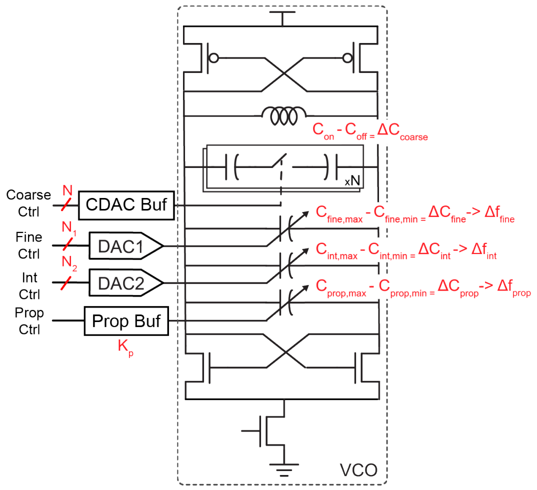

Coarse Ctrl
Fine Ctrl
Int Ctrl
Prop Ctrl
N
N1
N2
CDAC Buf
DAC1
DAC2
Prop Buf
Kp
Con - Coff = ΔCcoarse
xN
Cfine,max - Cfine,min = ΔCfine -> Δffine
Cint,max - Cint,min = ΔCint -> Δfint
Cprop,max - Cprop,min = ΔCprop -> Δfprop
VCO

Fig. 3. Voltage-controlled oscillator (VCO) schematic.

B. Bang-Bang PLL

Bang-bang loops are widely used in wireline communication system components such as clock and data recovery (CDR) circuits [4], [5] and PLLs [6], [7]. The phase detector (PD) translates the input phase difference into a binary output, which simplifies integration with a digital or hybrid loop filter (LF). Compared to an analog filter, the required areas scale

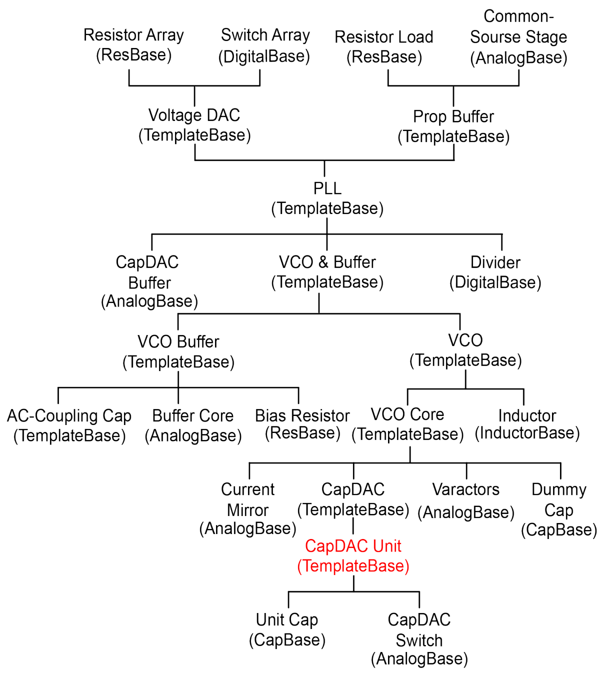
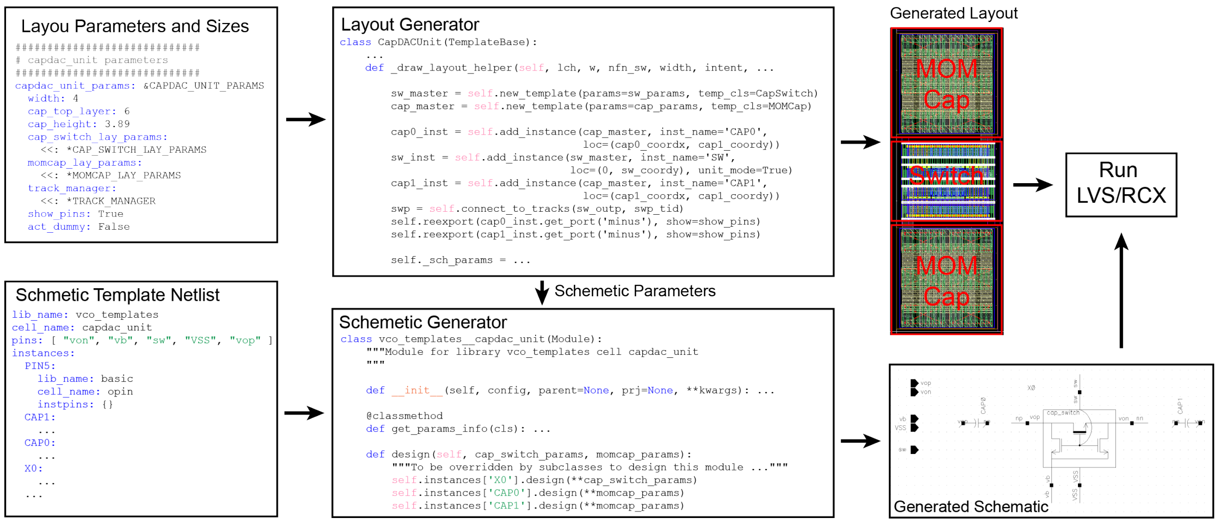

Resistor Array (ResBase)
Switch Array (DigitalBase)
Resistor Load (ResBase)
Common-Sourse Stage (AnalogBase)
Voltage DAC (TemplateBase)
Prop Buffer (TemplateBase)
PLL (TemplateBase)
CapDAC Buffer (AnalogBase)
VCO & Buffer (TemplateBase)
Divider (DigitalBase)
VCO Buffer (TemplateBase)
VCO (TemplateBase)
AC-Coupling Cap (TemplateBase)
Buffer Core (AnalogBase)
Bias Resistor (ResBase)
VCO Core (TemplateBase)
Inductor (InductorBase)
Current Mirror (AnalogBase)
CapDAC (TemplateBase)
Varactors (AnalogBase)
Dummy Cap (CapBase)
CapDAC Unit (TemplateBase)
Unit Cap (CapBase)
CapDAC Switch (AnalogBase)

Fig. 4. The hierarchy of the PLL design.

with technology node without performance degradation. A time-to-digital converter (TDC) can be also used for PDs in the digital domain, increasing the complexity of the circuit implementation. Since it is widely used in low-jitter applications, a bang-bang PLL is selected for this evaluation.

Fig. 2 shows the proposed bang-bang PLL architecture, which is based on a hybrid loop architecture containing a digital frequency loop and an analog phase loop. The digital frequency loop includes a frequency detector (FD), a bang-bang PD and an integrator as digital frequency loop filter. The FD detects the frequency error by comparing the number of counted clock cycles from the reference clock to the divider ratio. The bang-bang PD is a single flip-flop (FF), which generates phase early and late information by sampling the reference clock by the divider output. The FD and PD outputs are combined with programmable gain to drive the digital loop filter to generate control signals for frequency locking. The digital blocks are developed with Chisel ([8]) and elaborated to Verilog for digital synthesis.

The analog block contains a clock divider and an LC voltage-controlled oscillator (VCO), as well as peripherals such as a capacitor digital-to-analog converter (DAC) driver, voltage DACs, and a proportional path buffer driving different frequency knobs of the VCO. Fig. 3 shows a schematic diagram of the VCO, which includes a current mirror, NMOS/PMOS cross-coupled pairs, a capacitor DAC (CapDAC) and three varactors. One of the varactors is driven by a proportional buffer for phase control, with programmable gain K_p to tune the PLL loop bandwidth. The frequency control includes an N-bit CapDAC and two other varactors driven by N_1-bit and N_2-bit DACs. For the CapDAC and



```
##############################
# capdac_unit parameters
##############################
capdac_unit_params: &CAPDAC_UNIT_PARAMS
  width: 4
  cap_top_layer: 6
  cap_height: 3.89
  cap_switch_lay_params:
    <<: *CAP_SWITCH_LAY_PARAMS
  momcap_lay_params:
    <<: *MOMCAP_LAY_PARAMS
  track_manager:
    <<: *TRACK_MANAGER
  show_pins: True
  act_dummy: False
```

```
class CapDACUnit(TemplateBase):
    ...
    def _draw_layout_helper(self, lch, w, nfn_sw, width, intent, ...

        sw_master = self.new_template(params=sw_params, temp_cls=CapSwitch)
        cap_master = self.new_template(params=cap_params, temp_cls=MOMCap)

        cap0_inst = self.add_instance(cap_master, inst_name='CAP0',
                                      loc=(cap0_coordx, cap1_coordy))
        sw_inst = self.add_instance(sw_master, inst_name='SW',
                                    loc=(0, sw_coordy), unit_mode=True)
        cap1_inst = self.add_instance(cap_master, inst_name='CAP1',
                                      loc=(cap1_coordx, cap1_coordy))
        swp = self.connect_to_tracks(sw_outp, swp_tid)
        self.reexport(cap0_inst.get_port('minus'), show=show_pins)
        self.reexport(cap1_inst.get_port('minus'), show=show_pins)

        self._sch_params = ...
```

```
lib_name: vco_templates
cell_name: capdac_unit
pins: [ "von", "vb", "sw", "VSS", "vop" ]
instances:
  PIN5:
    lib_name: basic
    cell_name: opin
    instpins: {}
  CAP1:
    ...
  CAP0:
    ...
  X0:
    ...
  ...
```

```
class vco_templates__capdac_unit(Module):
    """Module for library vco_templates cell capdac_unit
    """

    def __init__(self, config, parent=None, prj=None, **kwargs): ...

    @classmethod
    def get_params_info(cls): ...

    def design(self, cap_switch_params, momcap_params):
        """To be overridden by subclasses to design this module ..."""
        self.instances['X0'].design(**cap_switch_params)
        self.instances['CAP0'].design(**momcap_params)
        self.instances['CAP1'].design(**momcap_params)
```

Fig. 5. Schematic and layout generation flow example with the analog generator.

varactors, the capacitor and corresponding frequency changes are labeled in Fig. 3. The analog blocks are implemented with the BAG, to create process-portable PLL instances.

III. PLL Generator and Design Methodologies

To generate the PLL, the parameterized schematic and layout generators are initially implemented to create subblock instances from bottom to top. Then, the block-level design scripts create block instances to meet block-level specifications from system models. Finally, the generated block instances are combined to produce a top-level instance that satisfies all PLL specifications.

The design methodologies for the PLL generator are classified into three types. The direct-sizing method adjusts the block size simply from fanout analysis or pre-defined interfaces. This method is suitable for top-level and non-critical blocks, including CapDAC buffer, divider, proportional path buffer and voltage DAC of the PLL. The design method based on look-up tables is used for single devices or low-level circuits such as cross-coupled pairs, inductors, MOM capacitors, and CapDAC switches. The iteration loop method is used for critical blocks, which are power-consuming and determine the performance of the whole PLL system.

A. *Schematic and Layout Generation*

The process-portable circuit generator requires that the design instances, produced from the schematic and layout generators, are DRC- and LVS-clean in different technologies. Fig. 4 shows the hierarchy of the PLL design using the XBase layout engine, including *TemplateBase*, *AnalogBase*, *DigitalBase*, *ResBase*, *CapBase* and *InductorBase*. It is notable that all layout generators are inherited from *TemplateBase* and merged within.

As shown in Fig. 5, the layout generator first retrieves the layout parameters, and sizes the design to generate the layout for each block in Fig. 4, where the *new_template()* function

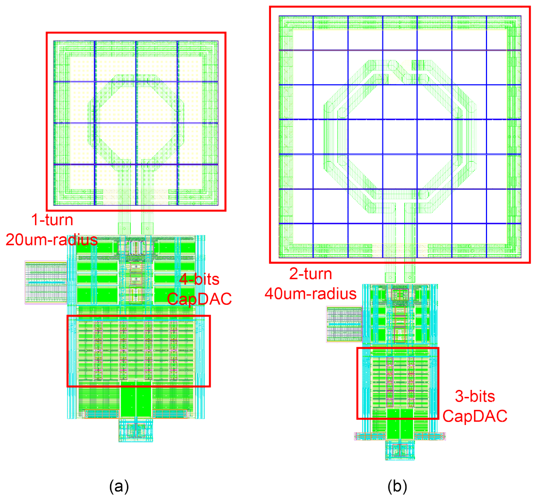


Fig. 6. VCO layout with (a) 4-bit CapDAC, 1-turn inductor with radius of $20\mu m$ and (b) 3-bit CapDAC, 2-turn inductor with radius of $40\mu m$.

creates a lower-level instance in the Cadence library and the *add_instance()* function instantiates the instance in current layout. The schematic generator gets the schematic parameters from the layout generator and sizes the schematic template with *design()* functions to generate the schematic instance. LVS and extraction are run on the views of the instance to generate the extracted netlist for post-layout simulation. Fig. 6 shows two VCO layouts with the 4-bit CapDAC, 1-turn inductor with radius of $20\mu m$ and the 3-bit CapDAC, 2-turn inductor with radius of $40\mu m$, respectively, demonstrating the flexibility and effectiveness of the layout generator.

B. *PLL Top-Level Analysis*

The block generators get block-level specifications from the system-level model. In this design example, the VCO generator within the PLL generator requires the phase noise and frequency tuning range of the CapDAC and the fine, integral and proportional varactors.

VCO phase noise is high-pass filtered by the loop filter and decreases with a higher loop bandwidth (proportional to $K_p\Delta f_{prop}$). On the other hand, the limit cycle jitter from the loop latency and frequency step of the bang-bang PD increases when $K_p\Delta f_{prop}$ rises. Similar to the analysis in [6] and [7], the optimal $K_p\Delta f_{prop}$ can be found, which determines Δf_{prop} if K_p is assumed. The VCO is also required to cover the frequency range under process, voltage and temperature (PVT) variations. The maximum and minimum frequency are restricted by $f_{max} - f_{min} = \alpha f_0$, where α is the frequency range ratio and f_0 is the center frequency. The frequency overlaps between CapDAC and fine varactor settings, as well as fine varactor and integral varactor settings, guarantee continuous frequency tuning. Finally, to avoid jitter contribution from the frequency control loop, the frequency resolution of the integral varactor is required to be much higher than that of the proportional varactor. These conditions are listed as constraints in the optimizations of the VCO design.

C. Inductor Generator

When running block-level generators, device-level generators are required to provide the optimal devices for given specifications. For example, the device-level generators for the MOS transistors, inductors, MOM capacitors, and CapDAC switches in the VCO are based on the look-up table design method. These device generators follow a similar optimization approach except for their input parameters and performance specifications. In the following, the inductor generator is used as an example to show the development of a device-level generator.

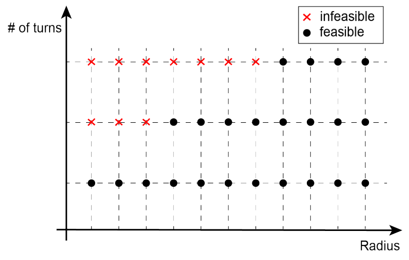


Fig. 7. Inductor feasibility check example (for given width and space).

One particular issue in inductor generation is the potential for DRC errors in inductor layouts with certain parameters. For instance, when the radius of an inductor with multiple turns is too small, the layout is unfeasible because of some of the required paths are too short to meet DRC rules. To address this problem, a feasibility function is defined

$$Feasibility(n,r,w,s) = \begin{cases} 1 & \text{if } DRC\ clean \\ 0 & \text{if } DRC\ unclean \end{cases} \quad (1)$$

where n is number of turns, r is radius, w is metal width and s is metal spacing. This function supports computations such as interpolation of the four variables and optimization for Q or area. When this function value is smaller than 1, the inductor is considered infeasible (Fig. 7).

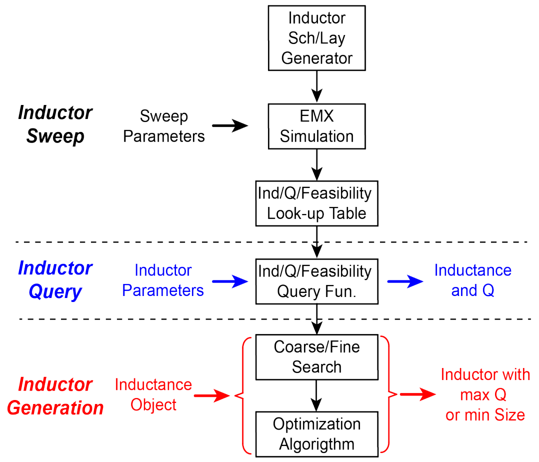


Fig. 8. Inductor generation flow.

As shown in Fig. 8, the inductor generator includes three parts. The first part is sweeping inductors of different sizes. Starting from the inductor schematic and layout generator, four parameters including radius, number of turns, metal width and space are swept. The generated layouts are sent to EMX for electromagnetic (EM) simulation and the inductance and quality factor (Q) are calculated based on the S-parameter results. The feasibility value is also included by checking the inductor layout. In the second step, inductor query, an interpolation method is performed across the whole look-up table, to produce the inductor, Q and feasibility functions. With these functions, the inductance and Q are estimated to within an error of 1%, as shown in Table I. Finally, with the query functions, a search algorithm is implemented based on the look-up table for maximum Q or minimum size. The search result is used as a initial value for the openMDAO [9] optimization algorithm to further optimize the result. After an optimized inductor is generated, the S-parameter and model files are extracted by EM simulation for higher-level circuit simulations.

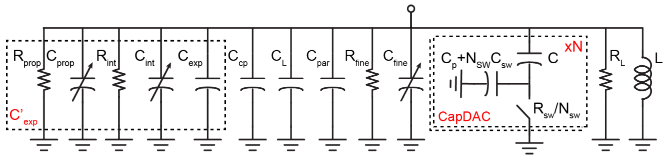


Fig. 9. VCO tank model.

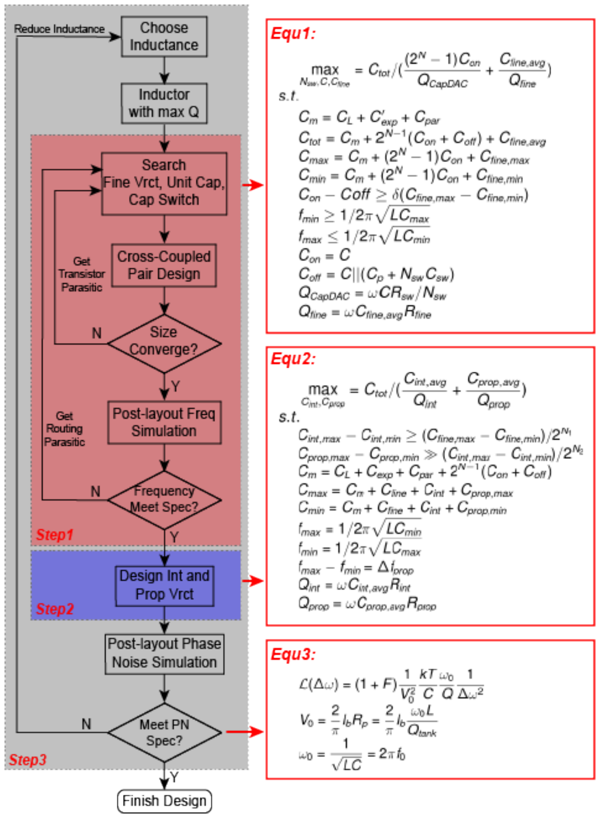
D. VCO Design Procedure

Fig. 9 depicts a single-ended model of the VCO tank. It can be modeled as an inductor L, a capacitor DAC, a fine-tuning varactor C_{fine}, a routing parasitic capacitor C_{par}, a load capacitance C_L, a explicit capacitor C_{exp}, an integral-tuning varactor C_{int} and a proportional path varactor C_{prop}, while R_L, R_{fine}, R_{int} and R_{prop} are the parasitic resistances of

TABLE I
INDUCTANCE AND Q-FACTOR COMPARISON FROM INDUCTOR QUERY FUNCTION AND EM SIMULATION AT 14GHZ

#	turn	radius	width	space	EM Sim		Calculation		Relative Error	
		(μm)	(μm)	(μm)	Inductance (nH)	Q	Inductance (nH)	Q	Inductance	Q
1	2	62	4.5	2.5	0.894	11.56	0.891	11.60	0.34%	0.34%
2	1	62	4.5	2.5	0.324	9.27	0.322	9.28	0.62%	0.11%
3	3	62	4.2	2.5	1.78	10.76	1.77	10.70	0.56%	0.56%
4	2	75	7	3.3	1.02	12.62	1.02	12.64	0%	0.16%

the inductor and the fine, integral, and proportional varactors. The Capacitor DAC is split into N units with a capacitor and switch of N_{sw} fingers. Fig. 10 shows the VCO design flow as three steps. The design flow starts by generating an inductor with the maximum Q factor for a given initial inductance value estimated from an initial phase noise factor.



Reduce Inductance
Choose Inductance
Inductor with max Q
Search Fine Vrct, Unit Cap, Cap Switch
Get Transistor Parasitic
Cross-Coupled Pair Design
N
Size Converge?
Y
Get Routing Parasitic
Post-layout Freq Simulation
N
Frequency Meet Spec?
Y
Step1
Design Int and Prop Vrct
Step2
Post-layout Phase Noise Simulation
N
Meet PN Spec?
Y
Step3
Finish Design
Equ1:
$\max_{N_{sw},C,C_{fine}} = C_{tot}/(\frac{(2^N-1)C_{on}}{Q_{CapDAC}} + \frac{C_{fine,avg}}{Q_{fine}})$
s.t.
$C_m = C_L + C'_{exp} + C_{par}$
$C_{tot} = C_m + 2^{N-1}(C_{on} + C_{off}) + C_{fine,avg}$
$C_{max} = C_m + (2^N - 1)C_{on} + C_{fine,max}$
$C_{min} = C_m + (2^N - 1)C_{on} + C_{fine,min}$
$C_{on} - Coff \geq \delta(C_{fine,max} - C_{fine,min})$
$f_{min} \geq 1/2\pi\sqrt{LC_{max}}$
$f_{max} \leq 1/2\pi\sqrt{LC_{min}}$
$C_{on} = C$
$C_{off} = C||(C_p + N_{sw}C_{sw})$
$Q_{CapDAC} = \omega CR_{sw}/N_{sw}$
$Q_{fine} = \omega C_{fine,avg}R_{fine}$
Equ2:
$\max_{C_{int},C_{prop}} = C_{tot}/(\frac{C_{int,avg}}{Q_{int}} + \frac{C_{prop,avg}}{Q_{prop}})$
s.t.
$C_{int,max} - C_{int,min} \geq (C_{fine,max} - C_{fine,min})/2^{N_1}$
$C_{prop,max} - C_{prop,min} \gg (C_{int,max} - C_{int,min})/2^{N_2}$
$C_m = C_L + C_{exp} + C_{par} + 2^{N-1}(C_{on} + C_{off})$
$C_{max} = C_m + C_{fine} + C_{int} + C_{prop,max}$
$C_{min} = C_m + C_{fine} + C_{int} + C_{prop,min}$
$f_{max} = 1/2\pi\sqrt{LC_{min}}$
$f_{min} = 1/2\pi\sqrt{LC_{max}}$
$f_{max} - f_{min} = \Delta f_{prop}$
$Q_{int} = \omega C_{int,avg}R_{int}$
$Q_{prop} = \omega C_{prop,avg}R_{prop}$
Equ3:
$\mathcal{L}(\Delta\omega) = (1+F)\frac{1}{V_0^2}\frac{kT}{C}\frac{\omega_0}{Q}\frac{1}{\Delta\omega^2}$
$V_0 = \frac{2}{\pi}I_bR_p = \frac{2}{\pi}I_b\frac{\omega_0 L}{Q_{tank}}$
$\omega_0 = \frac{1}{\sqrt{LC}} = 2\pi f_0$

Fig. 10. Automatic VCO design flow.

In step one, C_{prop}, C_{int} and C_{exp} are modeled as a fixed capacitor C'_{exp}. Based on the initial inductance, the VCO generator decides the fine tuning varactor capacitance, the CapDAC unit capacitance C and the capacitor switch N_{sw} (Fig. 9) by maximizing the resonant tank Q factor with frequency range constraints, as shown in Equations 1 of Fig. 10. However, the size of the cross-coupled pair and its parasitic capacitance are still undetermined when running the searching algorithm, and the frequency drops after taking it into consideration. To address this problem, when the cross-coupled pair is sized, its parasitic capacitance is returned to the searching algorithm until the size of the cross-coupled pair converges to a stable value.

Similarly, the routing parasitics were initially neglected, causing a frequency gap between calculation and simulation. This parasitic capacitance is included in C_{par} and updated by equation (2) after each simulation and sent back to the search algorithm until the frequency converges to the design specifications.

$$C_{par,new} = C_{par,old} + \frac{1}{4\pi^2 L}\left(\frac{1}{f_{sim}^2} - \frac{1}{f_{cal}^2}\right) \quad (2)$$

Another search algorithm designs the integral and proportional varactors, by maximizing tank quality factor with frequency overlap and resolution constraints, as shown in Equations 2 of Fig. 10. From Equations 3 of Fig. 10, once output amplitude V_0 and center frequency f_0 are determined, the reduction in inductance L causes a increase of total capacitance C and bias current I_b, as well as an improvement in phase noise. Therefore, after checking the phase noise result from post-layout simulation, the loop iterates with decreasing inductance until the phase noise specification is satisfied.

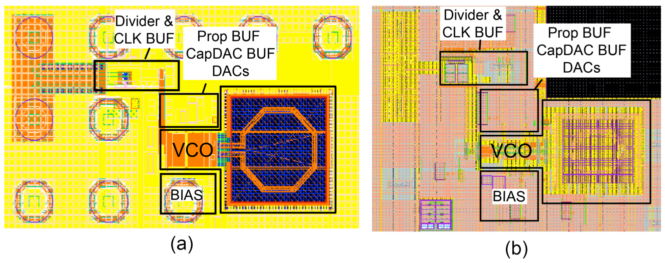


Fig. 11. Generated PLL layouts at (a) GF 14nm and (b) Intel 22nm technologies.

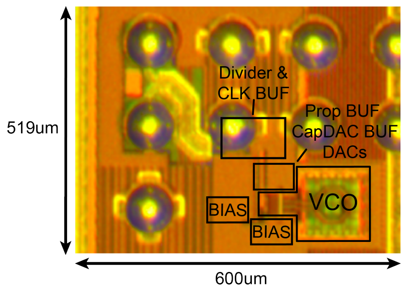
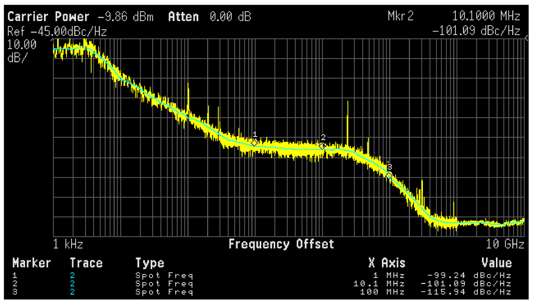


Fig. 12. Chip micrograph of the PLL in TSMC 16nm.

IV. Results

The PLL generator is tested in three CMOS FinFET technologies, TSMC 16nm, GF 14nm and Intel 22nm. The design time to generate a full layout is less than four days, including less than 20 minutes for schematic and layout generation, 12-37 hours for VCO generation and one day for adding dummy fill and I/O pads with electrostatic discharge (ESD) protection. The time differences of VCO generation are mainly caused by the extraction and simulation procedures in different technologies. The specifications and parameters are shown in Table II. Fig. 11 shows the PLL layouts in GF 14nm and Intel 22nm technologies. The core areas of the two layouts are $0.043mm^2$ and $0.048mm^2$. Fig. 12 shows the PLL micrograph of TSMC16nm process, with the core area being $0.042mm^2$. Fig. 13 shows the measured phase noise of PLL; the integrated jitter is 565.4fs in the 1KHz to 100MHz frequency range with a power of 6.64mW from a 0.9V supply.



```
Marker   Trace   Type                X Axis      Value
1        2       Spot Freq            1 MHz      -99.24 dBc/Hz
2        2       Spot Freq         10.1 MHz     -101.09 dBc/Hz
3        2       Spot Freq          100 MHz     -115.94 dBc/Hz
```

Fig. 13. Measured PLL phase noise at 14GHz in TSMC 16nm.

TABLE II
Top-level specifications for PLL generator

Center frequency f_0	14 GHz
Total RMS Jitter J_{tot}	600 fs
Frequency range ratio α	20%
Overlap ratio δ	20%
CapDAC bits	4
Fine DAC bits	8
Integral DAC bits	7
Proportional gain K_p	0.15
Proportional frequency step Δf_{prop}	33 MHz

Figure 14 shows the simulated phase noise and frequencies with different CapDAC settings of the generated VCO in three different processes under specifications of -100dBc/Hz phase noise at 1MHz offset and 20% range of the 14GHz center frequency. From the results, the three designs meet the target design specifications, validating the effectiveness of the VCO design scripts and demonstrating their process portability.

V. Conclusion

A complete process-portable LC PLL generator is developed and with instances generated in multiple technology nodes. The layout and schematics instances are DRC- and LVS-clean. Three different kinds of design methodologies are introduced

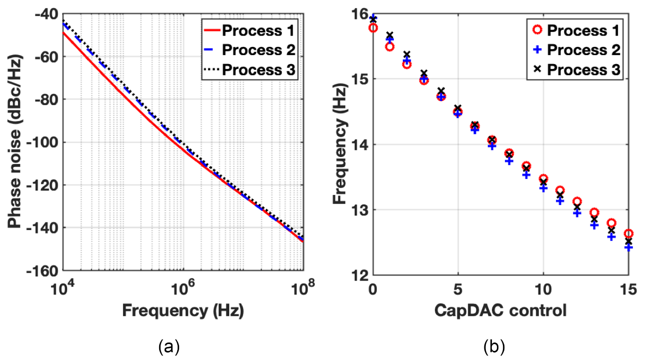


Fig. 14. Simulated VCO performance in three different technologies: (a) Phase noise and (b) frequency range with CapDAC control.

and used to generate the PLL instance meeting design specs automatically. The measurement results of the fabricated 16nm chip demonstrate the performance is comparable to a manually designed circuit, meeting the specifications for high-speed wireline links.

Acknowledgment

The authors would like to thank A. Flynn for designing the test board, TSMC for chip fabrication, and the sponsors, students, and faculty of the Berkeley Wireless Research Center.

References

[1] E. Chang et al., ”BAG2: A process-portable framework for generator-based AMS circuit design,” *2018 IEEE Custom Integrated Circuits Conference (CICC)*, San Diego, CA, USA, 2018, pp. 1-8.

[2] K. Hakhamaneshi, et al., ”BagNet: Berkeley Analog Generator with Layout Optimizer Boosted with Deep Neural Networks,” *2019 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, Westminster, CO, USA, 2019, pp. 1-8.

[3] K. Settaluri, et al., ”AutoCkt: Deep Reinforcement Learning of Analog Circuit Designs,” *2020 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, Grenoble, France, 2020, pp. 490-495.

[4] J. D. H. Alexander, “Clock Recovery from Random Binary Data,”. *Electronics Letters*, vol. 11, pp. 541-542, Oct. 1975.

[5] J. Savoj, et al., ”A 10-Gb/s CMOS clock and data recovery circuit with a half-rate binary phase/frequency detector,” in *IEEE Journal of Solid-State Circuits*, vol. 38, no. 1, pp. 13-21, Jan. 2003.

[6] M. Zanuso, et al., ”Noise Analysis and Minimization in Bang-Bang Digital PLLs,” in *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 56, no. 11, pp. 835-839, Nov. 2009

[7] J. Crossley, et al., ”An energy-efficient ring-oscillator digital PLL,” *IEEE Custom Integrated Circuits Conference 2010*, San Jose, CA, USA, 2010, pp. 1-4.

[8] J. Bachrach et al., ”Chisel: Constructing hardware in a Scala embedded language,” *DAC Design Automation Conference 2012*, San Francisco, CA, USA, 2012, pp. 1212-1221.

[9] J. S. Gray, et al., “OpenMDAO: An Open-Source Framework for Multidisciplinary Design, Analysis, and Optimization,” *Structural and Multidisciplinary Optimization*, 2019.