Analog Layout Generation using Optimized Primitives

Meghna Madhusudan[1], Arvind K. Sharma[1], Yaguang Li[2], Jiang Hu[2], Sachin S. Sapatnekar[1], Ramesh Harjani[1]
[1]University of Minnesota, Minneapolis, MN [2]Texas A&M University, College Station, TX

***Abstract*—Hierarchical analog layout generators proceed from leaf cells ("primitives") to progressively larger blocks that are placed and routed. The quality of primitive cell layout is critical for design performance. This paper proposes a methodology that defines and optimizes the performance metrics of primitives during leaf cell layout. It incorporates layout parasitics and layout-dependent effects, providing a set of optimized layout choices for use by the place-and-route engine, as well as wire sizing guidelines for connections outside the cell. For FinFET-based designs of a high-frequency amplifier, a StrongARM comparator, and a fully differential VCO, our approach outperforms existing methods and is competitive with time-intensive manual layout.**

I. Introduction

Analog designers iterate between schematic and layout design until a set of target specifications are met. Schematic designers must use intuition and experience to guess at parasitics, and layout designers must match those guesses to achieve the performance of the schematic, respecting trade-offs made during schematic design. The slow turnaround (days/weeks) of manual layout hampers the iterative loop by allowing only a small number of iterations. By generating high-quality layouts in minutes, analog layout automation can potentially enable more design iterations to thoroughly explore the design space.

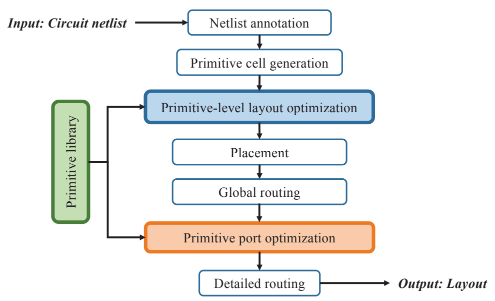
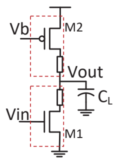


Fig. 1: A hierarchical analog layout synthesis flow.

Analog layout automation tools typically work hierarchically [1]–[3]. One such hierarchical flow, illustrated in Fig. 1, assembles building blocks starting with the lowest level structures, *primitives*. Primitives are groups of devices/passives (e.g., differential pairs and current mirrors) that form the basic building blocks of an analog circuit. In the flow, the circuit netlist is first annotated, either manually or automatically [4]–[6], to describe the circuit hierarchy from primitives to higher-level blocks, the connectivity between building blocks, and layout constraints. Next, based on a parameterizable primitive library, a cell generator builds primitive layouts with the right device sizes, incorporating symmetry, common-centroid, or interdigitation requirements. In FinFET nodes, it is common to use mesh-like routing to reduce resistive parasitics in lower metal layers. Primitives are assembled into larger blocks, which undergo hierarchical placement and global routing [7], incorporating performance and geometric constraints [8], [9]. Finally, detailed routing completes the layout.

This work is supported in part by the DARPA IDEA program, as part of the ALIGN project, under SPAWAR Contract N660011824048. The authors would like to thank M. Hassanpourghadi and Prof. M. Chen at USC for providing the VCO circuit.



	Schematic	Wire width in layout		
		Narrow	Wide	Optimized
Gain (dB)	18.04	17.90	18.03	18.02
UGF (GHz)	6.7	6.6	5.3	6.6
Power (μW)	291	290	290	290

Fig. 2: Parasitic RC trade-off in a common-source amplifier.

At the primitive level, there is substantial scope for optimizing circuit performance to match expectations from the schematic. For the common-source amplifier in Fig. 2, the sources of M1 and M2 are normally optimized to minimize R. We focus on the drain net, Vout, we examine the impact of varying wire width: performance degrades from the schematic for narrow (high R, low C) and wide (high C, low R) routes, but an optimized width approaches schematic performance.

This work considers optimizations at the primitive cell level. By targeting the schematic performance, we maintain designer intent (bias conditions, performance trade-offs). We insert two new optimization steps into the layout generation flow in Fig. 1:

- **Primitive-level layout optimization** selects a set of high-performance primitive layouts for the place/route engine, providing multiple options, with different aspect ratios, for optimizing the overall layout. We account for the performance of the *devices* by including layout dependent effects (LDEs) [10], [11]. and *wires* by optimizing wire widths within the primitive.
- **Primitive port optimization** optimizes a primitive in a global-routed layout and optimizes wire widths for external connections from primitive ports to external blocks. The optimized widths are a requirement for the detailed router.

As far as we know, this precise problem formulation has not been addressed in past work. In related work, [12] derives parasitic R and C limits of a circuit using top-level schematic simulations, but this method is computationally expensive and is not well suited for a hierarchical layout flow. In [8], [9], a machine learning (ML) model is used to distinguish good layout placements from bad based on circuit performance, but RCs on individual routes are not optimized. In [13], an algorithm to reduce the wire RC on each routing net is presented, but RC trade-offs for a net are not captured. In [14], analytical models for the impact of common LDEs on threshold voltage and mobility are used during placement optimization, but these are not as accurate as post-layout simulations considering all LDEs (BSIM models) and optimizing circuit performance metrics. We optimize primitive layouts and routes connected to them based on primitive performance, using post-layout simulations considering LDEs and parasitics simultaneously.

We demonstrate our methodology on a FinFET technology as analog design is rapidly moving to these nodes where layout design is more difficult [15]. Device and routing parasitics worsen and LDEs become more prominent [11], which could lead to large performance shifts from schematic to layout We honor gridded FinFET layout design rules, e.g., we achieve effective increases in wire width using parallel wires/vias.

II. Performance Metrics for Primitives

A. Primitives

Primitives are common building blocks of larger analog designs such as OTAs, comparators, and LNAs. A primitive library [2], [3] contains 20–30 primitive netlists and procedural layout generation code for various parameterizations (e.g., various device sizes, array structures, symmetries). A few classes of primitives, grouped by functionality, are:

- *Differential pairs (DPs)*, including cascoded versions in amplifiers/comparators, and switched variants in data converters.
- *Current mirrors (CMs)*, including active and passive CMs, and cascoded/low-voltage cascode structures.
- *Amplifiers*: Common source, common gate, common drain.
- *Loads*: Current sources, diode-connected structures, cascoded diode connected structures.
- *Digital-like analog structures*: Cross-coupled inverters, cross-coupled pairs, switches, current-starved inverters.
- *Passives*: Resistors, capacitors, inductors.

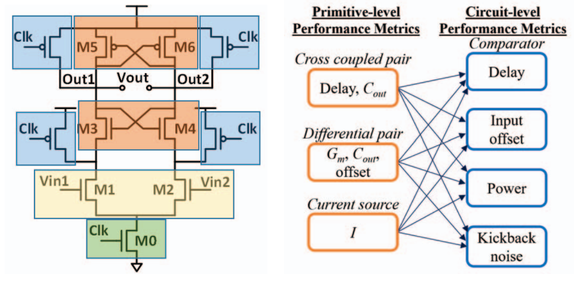
We develop a set of primitive-level performance metrics, such that if we limit the deviation of these metrics from their value in the schematic, the circuit performance remains within specifications. The layout of a primitive impacts the parameters of the elements within it. For example, device parameters, which are affected by LDEs, impact primitive performance metrics such as the effective transconductance, G_m, which depends on the transistor's transconductance, g_m, and the parasitic resistance of wires connected to its source.

The common-source amplifier in Fig. 2 has two primitives, a common-source amplifier stage (M1) and a current source load (M2). The primitive performance metrics of the common-source amplifier stage are the effective transconductance (G_m), output resistance, R_{out}, and the output capacitance, C_{out}, whose contributions from M1 are optimized to be close to the schematic value; other contributors outside the primitive are assumed to be at the schematic value. For the current source load, the performance metric is the current, I_{M2}, resistance, R_{out}, and output capacitance, C_{out}. Table I shows some performance metrics of both primitives. The optimized solution shows the best match between layout and schematic performance.

For each primitive, we survey its circuit-level use cases and identify the primitive-level performance metric that influences

TABLE I: Primitive-level metrics: Common source amplifier.

	Schematic	Narrow wire	Wide wire	Optimized wire
$G_{m,M1}$ (mA/V)	1.96	1.93	1.96	1.95
$R_{out,M1}$ (KΩ)	16.1	15.1	16.1	16.1
C_{total} (fF)	50.40	50.58	54.04	50.66
I_{M2} (μA)	290	290	290	290



(a) Annotated schematic (b) Performance metric relationships

Fig. 3: Relating primitive performance metrics to circuit-level performance metrics for a StrongARM comparator.

circuit performance. We are guided by simplified performance equations, e.g., for the common-source amplifier circuit [16]:

$$\text{Gain} = G_{m,M1}(R_{out,M1}||R_{out,M2}) \quad (1)$$

$$\text{UGF} = G_{m,M1}/(2\pi C_{total}) \quad (2)$$

$$\text{3dB Bandwidth} = 1/(2\pi(R_{out,M1}||R_{out,M2})C_{total}) \quad (3)$$

$$g_{m,M1} = \sqrt{2\mu_n C_{ox}(W_{M1}/L_{M1})I_{M2}} \quad (4)$$

where $R_{out(\mathrm{Mi})}$ is the output resistance of transistor M, and capacitance C_{total} has contributions from C_{out} of each primitive, the capacitance of the connecting wire, and the load.

We emphasize that we use these equations only to identify primitive performance parameters, and *the equations are never directly used in our methodology*: we analyze performance through cheap SPICE simulations on small structures.

Similar relationships can be drawn for other circuits as well. Fig. 3(a) shows a schematic for a StrongARM comparator, with primitives shown in shaded boxes. We draw a correspondence between primitive performance metrics (see [17] for performance equations) and top-level performance metrics (e.g., dynamic offset and delay), which are nonlinear functions of the primitive performance metrics listed in the figure.

The general principle illustrated on the above two circuits can be extended to other primitives by examining their usage in circuits and identifying their performance parameters of interest. This one-time exercise, for 20–30 primitive in a primitive library [2], [3], constitutes a manageable overhead.

B. Primitive performance metric evaluation

Primitive performance is affected by DC bias conditions. We get this information as input from circuit-level schematic simulations. To enable the analysis of primitive performance metrics, we augment the primitive library with the following:

- The important *performance metrics* for each primitive, and a *weight* for each metric to indicate its relative importance.
- *Tuning terminals* for the primitive, listing terminals at which RC parasitics can be traded off to improve performance, and

annotations on which tuning terminals are correlated (i.e., the optimum value of one depends on that of the other).

- A *primitive testbench* for each performance metric, i.e., a SPICE file that contains excitation and measure statements required to compute the metric through circuit simulation.

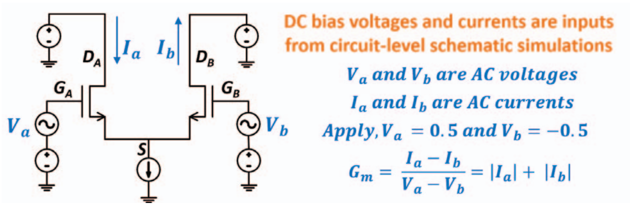
Because primitives are small (e.g., with < 4 transistors), these SPICE simulations are extremely fast, while including effects such as LDEs that are not well captured by analytical models. These augmentations are topology-dependent and technology-independent and can be used across multiple process nodes.

For the i^{th} metric, we assign a weight, high ($\alpha_i = 1$), medium ($\alpha_i = 0.5$), or low ($\alpha_i = 0.1$). A larger number indicates greater relative importance. This metric can be provided by the designer while creating the library entry. In a passive CM, the current ratio weight will be high ($\alpha_i = 1$) and the output capacitance weight will be low ($\alpha_i = 0.1$); in an active CM, the weight on output capacitance is medium ($\alpha_i = 0.5$).

For a set of common primitives, we highlight the performance metrics, weights, and tuning terminals in Table II. This information can be extended to other primitives in the same family. For example, DPs and their cascodes will have similar library entries. DP circuits for low-power amplifiers and comparators will have a high weight on G_m and input offset respectively. We illustrate an example setup for measuring the G_m of the DP in Fig. 4. We apply an AC voltage at the gate and measure the AC drain current.

TABLE II: Primitives metrics, tuning terminals, weights α.

Objectives	Tuning terminals	Objectives	Tuning terminals
DIFFERENTIAL PAIR		*CURRENT-STARVED INVERTER*	
G_m ($\alpha = 0.5$)	Source/drain RC	Delay ($\alpha = 1$)	Source/drain RC
G_m/C_{out} ($\alpha = 0.5$)		Current ($\alpha = 1$)	
Input offset ($\alpha = 1$)		Gain ($\alpha = 0.5$)	
CURRENT MIRROR		*COMMON-SOURCE AMPLIFIER*	
Output current ($\alpha = 1$)	Source/drain RC	G_m ($\alpha = 1$)	Source/drain RC
C_{out} ($\alpha = 0.1$)		r_o ($\alpha = 0.5$)	
CURRENT SORUCE		*CAPACITOR*	
Current ($\alpha = 1$)	Source/drain RC	C ($\alpha = 1$)	RC at terminals
r_o ($\alpha = 0.5$)		Frequency ($\alpha = 0.1$)	



D_A
I_a
I_b
D_B
G_A
G_B
V_a
V_b
S
DC bias voltages and currents are inputs from circuit-level schematic simulations
V_a and V_b are AC voltages
I_a and I_b are AC currents
Apply, $V_a = 0.5$ and $V_b = -0.5$
$G_m = \frac{I_a - I_b}{V_a - V_b} = |I_a| + |I_b|$

Fig. 4: Testbench for the G_m of the DP.

III. Methodology

We describe the two steps in our methodology, shown in Fig. 1: (1) primitive-level layout optimization, which ensures that the best performing primitive layouts are selected, tuned and passed to the placer, and (2) primitive port optimization, which uses the initial global route information to size the wires connected to the ports, providing specifications to the router.

The methodology optimizes primitive layouts so that their performance metrics approach those in the schematic. For example, in Table II, the important performance metrics for a DP depend on G_m, C_{out}, and input offset. With increasing layout parasitics, G_m reduces, and C_{out} and input offset increase, degrading circuit performance. We evaluate the post-layout values of these metrics to build a cost function, weighted by the importance of the metric, as described earlier.

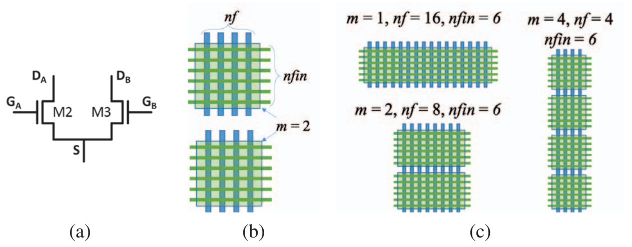


Fig. 5: (a) DP schematic. (b) Illustration of *m*, *nf*, *nfin*. (c) Transistor layout options for a DP with 96 FinFETs.

$$Cost = \sum_{i=1}^{k} \alpha_i \cdot \Delta x_i \tag{5}$$

where α_i is the weight given to each of the k performance metrics, and Δx_i, the deviation of the performance metric, is:

$$\Delta x_i = \begin{cases} \frac{|x_{i,sch} - x_{i,layout}|}{x_{i,sch}} & \text{if } x_{i,sch} \neq 0 \\ \max\left[0, \frac{|x_{i,spec} - x_{i,layout}|}{x_{i,spec}}\right] & \text{if } x_{i,sch} = 0 \end{cases} \tag{6}$$

Here, $x_{i,sch}$, $x_{i,layout}$, and $x_{i,spec}$ are the values of the metric in the schematic, layout, and specification, respectively. The first case covers the deviation from the schematic, while the second applies when $x_{i,sch} = 0$ (e.g., for DP input offset, where we define $x_{i,spec}$ as 10% of the random offset).

A. Primitive layout optimization

In a FinFET technology, for the DP primitive in Fig. 5(a) with specified (W/L), several layouts can be generated by changing layout parameters, as defined in Fig. 5(b): *nfin*, the number of fins per finger; *nf*, the number of fingers, and *m*, the multiplicity, i.e., repeated structures. Fig. 5(c) shows three DP transistor configurations, each with 96 FinFETs.

This step is carried out in two parts: *primitive selection*, which selects the best performing subset of layouts, accounting for layout parasitics and LDEs, and *primitive tuning*, which further optimizes routing parasitics by varying the effective wire widths (i.e., the number of parallel wires).

1) Primitive selection: For any primitive in the library, a parameterized analog primitive cell generator, such as [3], can generate multiple layout configurations in the style of Fig. 5(c) for a specified W/L ratio. Layout parameters such as the *placement pattern* (for example, common-centroid or interdigitated) and *aspect ratio* (by varying *nfin*, *nf* and *m*), lead to several layout options for each primitive, each corresponding to different tradeoff points for aspect ratio and performance shifts induced by wire parasitics and device LDEs.

As an example, diffusion sharing between transistors lowers the output capacitance and also reduces the area and interconnect lengths, which is generally desirable. However, in a 1:8 CM, it may be more important to generate $m = 8$ copies of the reference for good matching without sharing diffusion.

For a primitive performance metric, x_i, we simulate these structures using SPICE and the primitive testbench for metric x_i (Section II-B). This is inexpensive: the 96 FinFETs in Fig. 5(c) are modeled as a single device in SPICE with model parameters for *nfin*, *nf* and *m*. The enables the computation of Δx_i in (6) by capturing the $x_{i,sch}$-to-$x_{i,layout}$ drift due to:

- *Wire parasitics*: For each layout synthesized by the primitive cell generator, we extract all parasitics, incorporating diffusion sharing between transistors, the placement pattern, and within-primitive routing, and external loads from the schematic.
- *LDEs*: LDEs, such as length of diffusion (LOD) mismatch and well proximity effects (WPE), lead to threshold voltage shifts and are prominent in FinFET nodes. LDEs are modeled in layout extraction [11] and their impact on performance can be evaluated using SPICE. LDEs significantly impact the current ratio in current mirrors depending on the aspect ratio as shown in [10]. Other tradeoffs arise from the use of dummies, which reduce LOD effects, but increase area and wire parasitics.
- *Process variations*: Designers consider random variations during circuit sizing. They can also specify different layout patterns to minimize systematic variations.

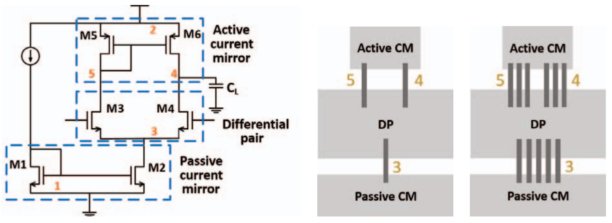
TABLE III: Cost components for various DP layout options.

Primitive	Pattern	ΔG_m	$\Delta \frac{G_m}{C_{total}}$	ΔOffset	Cost
nfin=8; *nf*=20; *m*=6 (**Aspect Ratio Bin 1**)	ABBA	1.4%	6.7%	0%	4.0
	ABAB	0.8%	6.4%	0%	**3.6**
	AABB	3.3%	7.4%	0%	5.3
nfin=16; *nf*=12; *m*=5 (**Aspect Ratio Bin 2**)	ABBA	2.0%	6.7%	0%	4.3
	ABAB	1.5%	6.3%	0%	**3.9**
nfin=24; *nf*=20; *m*=2 (**Aspect Ratio Bin 3**)	ABBA	1.7%	8.4%	0%	5.0
	ABAB	0.9%	7.8%	0%	4.4
	AABB	6.6%	12.1%	92%	101.7
nfin=12; *nf*=20; *m*=4 (**Aspect Ratio Bin 3**)	ABBA	1.4%	5.6%	0%	3.5
	ABAB	0.8%	5.2%	0%	**3.0**
	AABB	4.0%	7.2%	0%	5.6

By evaluating the performance of layout options, we select a small number of layout choices for the placer. To keep the number of options manageable, we bin options of similar layout (bounding box) aspect ratio and provide one option per bin.

For an example DP primitive with $W/L = 46\mu$m/14nm, we evaluate 11 layouts with different *nfin*, *nf*, and *m* and placement patterns such as common centroid (ABBA), interdigitated (ABAB), and non-common centroid (AABB), and divide the results into three bins with different layout aspect ratios. The performance deviation from the schematic and the *Cost* (Eq. (6)) are shown in Table III. Choosing the best (boldfaced) layout in each bin, we provide the placer with a manageable number of options. Bin 3 has 6 layouts and one is chosen.

2) Primitive tuning: After selecting a subset of primitive layouts, we tune them further to improve by replacing single wires by multiple parallel wires at their tuning terminal in Table II. For example, in the DP circuit shown in Fig. 5, reducing the resistance at the source node improves ΔG_m and reduces the cost. We start with adding a single wire, and continue until the the performance is closest to the schematic (minimum cost), or at the point of maximum curvature for a monotonically decreasing cost curve (typically not seen because increasing cell height will cause performance degradation).



(a) (b) (c)

Fig. 6: Primitive port optimization methodology, (a) 5T OTA schematic, (b) Global routes and (c) Routing constraints

3) Overall algorithm: The overall methodology for primitive layout optimization is summarized in Algorithm 1. We first simulate the circuit schematic to get an estimate of the DC bias conditions and the capacitance at different nets (line 3). These are inputs to the testbenches used to measure the primitive performance and layout cost (line 4). To provide options to the placer, we segregate layouts into n different aspect ratio bins (line 6) and select the primitives with the minimum cost in each bin (line 7). For each selected layout, we add parallel wires at the tuning terminals and measure the layout cost (line 11–15). If the tuning terminals are uncorrelated (affect the cost independently), we optimize them separately. If they are correlated we carry out a multi-way optimization where we enumerate combinations of the wire widths. Practically, this is manageable as it is uncommon to have more than two correlated terminals. We retain the n minimum cost layouts (line 16).

Algorithm 1 Primitive layout optimization

```
1: Input: Primitive layouts with varying nfin, nf, m, and placement pattern such that
   (nfin × nf × m) = constant as shown in Fig. 5, primitive library entry
2: Output: Set of $n$ optimized primitive layouts
3: for each input primitive layout do                    // Step 1: Primitive selection
4:     Simulate primitive performance and compute cost
5: end for
6: Split primitives into $n$ different aspect ratio bins
7: Select the primitive with minimum cost in each bin
8: for each primitive layout in best set do              //Step 2: Primitive tuning
9:     if uncorrelated then
10:        Optimize tuning terminals separately
11:    else
12:        Optimize terminals together
13:    end if
14:    Increase # of wires at tuning terminal and measure cost
15: end for
16: Select the primitive configuration with minimum cost
```

B. Primitive port optimization

The optimized primitive cells are sent to a simulated annealing placer, and global router based on [18]. Fig. 6(a) illustrates an OTA circuit whose primitives are global-routed (Fig. 6(b)). The global routes provide information about the wire lengths in each layer and via information. We use this information to optimize the wires connected to the ports of each primitive, selecting the number of parallel routes at the primitive ports (Fig. 6(c)). We proceed in two steps: primitive port constraint generation, which finds the optimal number of parallel wires on each route, and port constraint reconciliation, which reconciles the case where multiple blocks constrain the same route.

1) Primitive port constraint generation: In this step, the primitives generate constraints at their ports independently. For example, in Fig. 6(a), the DP constrains nets 3, 4 and 5, the passive current mirror constrains nets 1 and 3, and the active current mirror constrains nets 2, 4 and 5. We increase the number of parallel routes at the ports of the primitive, using the specific distance, layer and via information provided by the global router, and observe the impact on the primitive cost.

For the DP in Fig. 6(a), the G_m testbench in Fig. 4 is used. We alter the load to include the parasitics of all wires connected to the drain nodes. As we increase the number of parallel routes (decrease R, increase C), the G_m improves but C_{total} degrades. We find the range $[w_{min}, w_{max}]$ of the number of parallel routes for which the cost of the primitive is optimized. Here, w_{min} is the point of maximum curvature and w_{max} is either the point where the cost starts increasing with the wire count, or is unbounded if cost increases are not seen.

For the DP connected to the CM in Fig. 6(a), the global routes are on metal 3 and are 2um long. Table IV shows how each primitive is optimized separately in this step. The test setup measures the change in G_m, ΔG_m and the change in G_m/C_{total}, $\Delta\frac{G_m}{C_{total}}$. The input offset is maintained by the detailed router through a geometric constraint that keeps symmetric routes. As more wires are added at the drain, the DP performance initially improves due to G_m improvement and then degrades due to increased C_{total}. The allowable interval here is $[w_{min} = 3, w_{max} = 5]$, and becomes $[w_{min} = 4, w_{max} = 6]$ if ΔG_m is weighted higher. For the passive CM, we measure the current ratio and output capacitance.

TABLE IV: DP, CM cost during primitive port optimization

	Differential pair			Passive current mirror		
# Wires	ΔG_m	$\Delta\frac{G_m}{C_{total}}$	Cost	ΔCurrent ratio	ΔC_{total}	Cost
1	3.4%	6.9%	5.17	4.2%	3.6%	4.54
2	2.1%	6.7%	4.40	2.5%	8.4%	3.36
3	1.6%	6.8%	**4.23**	10.4%	4.1%	3.00
4	1.4%	7.0%	**4.21**	11.8%	2.9%	**2.85**
5	1.3%	7.2%	**4.25**	12.8%	2.0%	**2.77**
6	1.2%	7.5%	4.33	1.4%	13.6%	**2.74**
7	1.1%	7.7%	4.42	1.3%	14.5%	**2.75**

We handle Steiner nodes in two ways. Practically, most Steiner nodes arise from within-primitive connections that also connect to an external block. These are naturally handled by the methodology: primitive tuning and port optimization address internal and external connections as two-pin elements, respectively. For external global routes with Steiner nodes, the global router determines the Steiner points, and all branches of the Steiner tree use the same number of parallel wires.

2) Port Constraint Reconciliation: After the primitives generate interval constraints on the number of wires for each of their ports, we combine the constraints at nets that are constrained by multiple primitives. For example, in Fig. 6, net 3 is constrained to have $w_{min} = 1$ by the DP, and $w_{min} = 4$ by the passive CM, with no upper bound w_{max} over the range explored. We combine the intervals to choose 4 routes at net 3.

If the constraint intervals are $[w_{min,i}, w_{max,i}]$ for block i at a net, and the intervals overlap, we choose the minimum number of parallel routes in the overlapping interval ($\max(w_{min,i})$) for low routing congestion. If the intervals do not overlap, we minimize the deviation from schematic by performing further simulations in the range $[\min(w_{max,i}), \max(w_{min,i})]$. This interval represents the gap between the most constrained lower and upper bounds. We then determine the number of parallel wires that has the lowest total Cost for all primitives.

3) Overall algorithm: Algorithm 2 describes the primitive port optimization methodology. For each primitive, the effect of single routes on the primitive cost is analyzed (line 5). The primitive is simulated with parallel routes and when the improvement on cost stops we define a constraint range for each net connected to a primitive port (line 6). To combine multiple constraints at a net we select the lowest number of routes that satisfies requirements when constraint ranges overlap (line 11). If constraints do not overlap we choose the route that results in a minimum combined cost for the primitives (line 13-14).

Algorithm 2 Primitive port optimization

```
1: **Input**: Primitive layout; global routes at primitive ports (distance, layer, via usage)
2: **Output**: Optimized # of parallel routes at the ports
3: //Step 1: Primitive port constraints
4: for each primitive do
5:     Add wire RC models (global routes) to primitive layout
6:     Simulate over a range of parallel wires to find $[w_{min,i}, w_{max,i}]$ at a net
7: end for
8: //Step 2: Combining the constraints at a net
9: for each net do
10:     if Constraint intervals overlap then
11:         Choose # of parallel routes = $\max(w_{min,i})$
12:     else
13:         Simulate all primitives in $[\min(w_{max,i}), \max(w_{min,i})]$
14:         Choose # of parallel routes that minimize $\sum$ Cost
15:     end if
16: end for
```

C. Example of computational cost

The primitive-based methodology is accurate and computationally efficient. Each primitive layout simulation takes around 10s, including file handling time. Table V illustrates the different number of simulations required for a set of primitives, and the total time needed after accounting for the simulations that can be done simultaneously or in parallel. For a DP primitive with 3 metrics in the cost and 20 different layout configurations, we perform 60 simulations and choose a set of 3 options for place-and-route. During primitive tuning, we increase the effective wire size of the source net of the 3 selected layouts. For each layout, we find the optimum after 7 G_m testbench simulations. While generating port constraints, 8 simulations each on 2 testbenches are required to find the optimum number of parallel routes at the drain and source terminal. Although this sums up to 113 runs, the simulations at each of the three steps can be performed in parallel, the effective time needed is $3 \times 10 = 30$s. A similar analysis is shown for a current mirror and a current-starved inverter. Note that even in cases where more simulations are required (e.g., in infrequent cases where tuning parameters are correlated, or when intervals do not overlap), all simulations are independent, and the runtime is only limited by the number of parallel simulations. The increase in the number of simulations is not large even in these cases.

TABLE V: Number of simulations for a set of primitives.

	Differential pair	Current mirror	Current starved inverter
1. Primitive selection	20×3	10×2	20×3
2. Primitive tuning	$3 \times 7 \times 1$	$3 \times 5 \times 2$	$3 \times 9 \times 3$
3. Net routing constraints	$2 \times 8 \times 2$	$1 \times 12 \times 2$	$1 \times 8 \times 2$
Total simulations	113	74	157
Total time	$3 \times 10s = 30$s	3×10s = 30s	3×10s = 30s

IV. Results

We apply our methodology on a set of analog circuits covering a range of circuits that require optimization of layout parasitics and LDEs. We demonstrate our approach on a high frequency five transistor OTA, a StrongARM comparator, and a ring-oscillator-based VCO. We apply our methodology using the framework of the open-source ALIGN [3] flow for analog layout. The geometrical constraint of matching two nets discussed in [19] have been implemented manually in all the cases, and power routing is also performed manually. These steps lie outside our methodology, but ensure that the evaluated results represent a full layout of the circuits, and incorporate degradations due to IR drop on supply lines.

We compare the layouts generated using our methodology, against schematic simulation, manual layout, and a conventional automated layout approach. The conventional approach, similar to [19], [20], is a non-hierarchical layout where transistors are laid out to meet geometrical constraints in placement and routing, but performs no optimizations for parasitics. The runtime of the circuits are presented in Table VIII. It includes the time taken for primitive cell generation and layout optimization, placement, global routing and primitive port optimization.

5T OTA for high frequency applications: Table VI shows a comparison of the results of our approach against the schematic, manual layout, and conventional layout with geometric constraints. There is a clear improvement in the performance of layouts generated by our methodology: in this case, the performance is comparable to manual layout.

In the conventional approach, the current decreases due to the resistance at the drain of the passive CM and channel length modulation. Hence, all the other metrics are also degraded. Moreover, the parasitics at the source of the DP net degrade the G_m and impact the gain and the unity gain frequency.

TABLE VI: High-frequency OTA & StrongARM comparator

	Specification	Schematic	Manual	Automatic layout	
				Conventional	This work
High-frequency 5T OTA	Current (μA)	706	706	675	708
	Gain (dB)	22.6	22.4	21.8	22.4
	UGF (GHz)	5.1	4.8	4.2	4.8
	3-dB freq. (MHz)	389	384	362	383
	Phase margin (°)	77.9	78.0	75.5	77.2
StrongARM Comparator	Delay (ps)	19.2	25.4	35.0	31.5
	Power (μW)	145	161	172	168

StrongARM comparator: Results for this circuit are shown in Table VI. This circuit transient simulations to measure the performance at the top-level. We observe that the offset is similar in all cases, because it is a function of matching nets [19]. For our method, the delay, which depends on parasitics, is better than the conventional method.

Eight-stage differential RO-VCO: For a differential eight-stage RO-VCO circuit the primitive (current starved inverter) and its ports are optimized for delay and current. This circuit has an RC trade-off at the output of each current-starved inverter stage. The results of our approach (Table VII) demonstrates its effectiveness over the conventional method.

TABLE VII: Eight-stage differential RO-VCO

	Specification	Schematic	Automatic layout	
			Conventional	This work
Eight-stage RO-VCO	Max. frequency (GHz)	7.5	3.8	5.5
	Min. frequency (GHz)	0.20	0.26	0.25
	Voltage range (V)	0 – 0.5	0.1 – 0.5	0 – 0.5

TABLE VIII: Runtime of our approach for different circuits.

Circuit	High-frequency 5T OTA	StrongARM comparator	RO-VCO
Runtime (s)	80	85	135

V. Conclusion

The design of primitive leaf cells is crucial for analog performance, especially for FinFET nodes. This paper presents a systematic method for ensuring that primitive performance tracks schematic performance. The effectiveness of the method shows significant improvements over prior work. This work can readily be extended to other technologies including bulk nodes.

References

[1] G. G. E. Gielen and R. A. Rutenbar, “Computer-aided design of analog and mixed-signal integrated circuits,” *Proc. of the IEEE*, vol. 88, no. 12, pp. 1825–1854, 2000.

[2] M. Eick, *et al.*, “Comprehensive Generation of Hierarchical Placement Rules for Analog Integrated Circuits,” *IEEE T. Comput. Aid. D.*, vol. 30, no. 2, pp. 180–193, 2011.

[3] K. Kunal, *et al.*, “ALIGN: Open-source analog layout automation from the ground up,” in *Proc. DAC*, pp. 77–80, 2019.

[4] T. Massier, *et al.*, “The Sizing Rules Method for CMOS and Bipolar Analog Integrated Circuit Synthesis,” *IEEE T. Comput. Aid. D.*, vol. 27, no. 12, pp. 2209–2222, 2008.

[5] M. Meissner and L. Hedrich, “FEATS: Framework for Explorative Analog Topology Synthesis,” *IEEE T. Comput. Aid. D.*, vol. 34, no. 2, pp. 213–226, 2015.

[6] K. Kunal, *et al.*, “A general approach for identifying hierarchical symmetry constraints for analog circuit layout,” in *Proc. ICCAD*, 2020.

[7] M. P. Lin, *et al.*, “Recent research development and new challenges in analog layout synthesis,” in *Proc. ASP-DAC*, pp. 617–622, 2016.

[8] Y. Li, *et al.*, “A Customized Graph Neural Network Model for Guiding Analog IC Placement,” in *Proc. ICCAD*, 2020.

[9] M. Liu, *et al.*, “Towards Decrypting the Art of Analog Layout: Placement Quality Prediction via Transfer Learning,” in *Proc. DATE*, pp. 496–501, 2020.

[10] P. G. Drennan, *et al.*, “Implications of Proximity Effects for Analog Design,” in *Proc. CICC*, pp. 169–176, 2006.

[11] A. L. S. Loke, *et al.*, “Analog/Mixed-Signal Design Challenges in 7-nm CMOS and Beyond,” in *Proc. CICC*, pp. 1–8, 2019.

[12] E. Malavasi, *et al.*, “Automation of IC layout with analog constraints,” *IEEE T. Comput. Aid. D.*, vol. 15, no. 8, pp. 923–942, 1996.

[13] H. Chi, *et al.*, “Performance-preserved Analog Routing Methodology via Wire Load Reduction,” in *Proc. ASP-DAC*, pp. 482–487, 2018.

[14] H. Ou, *et al.*, “Layout-dependent effects-aware analytical analog placement,” *IEEE T. Comput. Aid. D.*, vol. 35, no. 8, pp. 1243–1254, 2016.

[15] R. A. Rutenbar, “Analog Circuit and Layout Synthesis Revisited,” in *Proc. ISPD*, p. 83, 2015.

[16] B. Razavi, *Design of Analog CMOS Integrated Circuits*. New York, NY: McGraw-Hill, 2001.

[17] B. Razavi, “The StrongARM Latch [A Circuit for All Seasons],” *IEEE Solid-State Circuits Magazine*, vol. 7, no. 2, pp. 12–17, 2015.

[18] Q. Ma, *et al.*, “Simultaneous Handling of Symmetry, Common Centroid, and General Placement Constraints,” *IEEE T. Comput. Aid D.*, vol. 30, pp. 85–95, Jan. 2011.

[19] H. Ou, *et al.*, “Non-uniform Multilevel Analog Routing with Matching Constraints,” in *Proc. DAC*, pp. 549–554, 2012.

[20] L. Xiao, *et al.*, “Practical Placement and Routing Techniques for Analog Circuit Designs,” in *Proc. ICCAD*, pp. 675–679, 2010.