Generative-Adversarial-Network-Guided Well-Aware Placement for Analog Circuits

Keren Zhu, Hao Chen, Mingjie Liu, Xiyuan Tang, Wei Shi, Nan Sun, and David Z. Pan
ECE Department, The University of Texas at Austin, Austin, TX, USA
{keren.zhu, haoc,jay_liu, xitang, weishi0079}@utexas.edu, nansun@mail.utexas.edu, dpan@ece.utexas.edu

***Abstract*—Generating wells for transistors is an essential challenge in analog circuit layout synthesis. While it is closely related to analog placement, very little research has explicitly considered well generation within the placement process. In this work, we propose a new analytical well-aware analog placer. It uses a generative adversarial network (GAN) for generating wells and guides the placement process. A global placement algorithm spreads the modules given the GAN guidance and optimizes for area and wirelength. Well-aware legalization techniques then legalize the global placement results and produce the final placement solutions. By allowing well sharing between transistors and explicitly considering wells in placement, the proposed framework achieves more than 74% improvement in the area and more than 26% reduction in half-perimeter wirelength over existing placement methodologies in experimental results.**

I. Introduction

Layout synthesis automation for analog integrated circuits (IC) is rapidly evolving in recent years. Automated place and route (PnR), as one of the mainstream analog layout synthesis paradigms, has demonstrated success in generating high-quality layouts [1]. However, as analog layout design methodology is significantly different from its digital counterpart, various challenges specific to analog IC are yet to be solved [2].

Generating well in placement is one of the remaining unsolved problems. The well layer defines the doping area that acts as the bulks of MOSFETs. For example, a typical P-MOS device needs to be built on an N-well layer and needs contacts to supply the bulk voltage (usually VDD). In a typical digital design methodology, wells are pre-designed within standard cells so that well generation is not needed in layout automation flow. However, analog design methodologies often use customized device layouts, and wells are usually distinctly drawn in manual designs. Manual designs often share wells between transistors to reduce spacing and the number of contacts to optimize area and interconnection. Furthermore, well geometries also impact circuit performance through layout-dependent effects, such as the well proximity effect (WPE). Therefore, inserting wells is an additional task to analog layout flow compared to its digital counterpart.

Existing analog placement techniques seldom consider wells, and automatic layout frameworks often use individual wells for each PMOS, e.g., ALIGN [3] and MAGICAL [4], [5]. Ou et al. [6] consider the wells and the assignments of devices to wells as given inputs. Their proposed framework optimizes the placement of the devices given fixed wells. Cohn et al. [7] propose to generate wells after the placement. After the placement, the well for each device is generated, and the intersected wells are merged. These two techniques do not consider the well explicitly in the placement. Nakatake et al. [8] consider the well merging in a randomized simulated annealing-based analog placer with sequence pair formulation. It bounds a sub-sequence in a simple rectangular well if the devices happen to the same well type. Martins et al. [9] randomly select pairs of transistors and add a rectangular well around the transistors if they have the same well type in a simulated annealing-based placement engine. These two methods benefit the area by sharing some wells but limit to simple well shapes and randomized optimization processes. From a high-level overview, the well generation problem decides which devices shall be in the same well and shapes the wells shall have. This problem is, in fact, intuitive to the perspective of humans. Experienced designers often place the layouts in a well-sharing-friendly manner, and the well generation is naturally inferred from the placement. Xu et al. [10] leverage the manual design expertise and propose to use machine learning (ML) for generating the wells. Instead of designing rule-based procedures, they learn the manual well drawing strategies with a generative adversarial network (GAN) and use the trained model to guide the well generation process. The so-called `WellGAN` framework produces more regular well geometries and results in more well sharing compared to rule-based algorithms. However, `WellGAN` generates the wells after the placement which might be sub-optimal. In this work, we propose adapt machine learning-based well generation during the core placement engine to guide the analog placement process.

Furthermore, considering the device to edge spacing is important as transistors are vulnerable to the well proximity effect (WPE). Several techniques have been proposed to consider WPE in the analog placement problem. Ou et al. [6] consider the distances from device to the edges of pre-defined fixed wells and formulate the WPE effects as a cost in a non-linear programming-based analog placer. Dong et al. [11] consider the device to well edge distances in layout mitigation problem. They adjust the distances in an existing layout to reduce WPE. Martins et al. [9] generate rectangular wells on randomly selected pairs of transistors, and the well shape is adjusted to optimize WPE. In this work, we formulate the WPE handling as constraints and solve them in our placement engine.

In this work, we propose a generative-adversarial-network-guided well-aware analog placement. Following the successes of guiding analog layout automation with ML techniques [12], [13], we guide our non-linear programming-based analog placer with the GAN-guided well generation framework. By iterating between the analog placer and the well generation, the proposed framework can effectively share wells while optimizing area and wirelength. Our main contributions are summarized as follows.

1) We propose a new well-aware placement methodology of guiding the analog global placement with the ML-based well generation. A ML-guided analog placement paradigm is presented to seamlessly integrate ML models with optimization-based placer.
2) We propose to consider wells as fence region constraints and develop effective global placement techniques to solve them.
3) We formulate WPE as minimum device-well-edge distance constraints and propose an efficient well legalization algorithm.
4) We develop a well-aware legalization scheme to legalize the global placement.
5) Experimental results show that our method can significantly

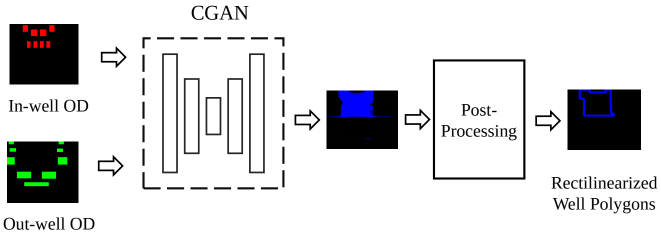


Fig. 1: The `wellgan` inference flow.

reduce area and half-perimeter wirelength by encouraging more well sharing when mitigating the WPE through enforcing the WPE distance constraints.

The rest of the paper is organized as follows. Sec. II introduces the preliminaries and presents the problem formulation. Sec. III describes the details of the proposed framework. Sec. IV shows the experimental results. Sec. V concludes the paper.

II. Preliminaries

In this section, we introduce the preliminaries on the machine learning-guided well generation framework(Sec. II-A) and the well proximity effect (Sec. II-B). Then, the problem formulation is presented in Sec. II-C.

A. ML-guided well generation framework

ML-guided well generation generates wells following the guidance from generative ML models, such adversarial networks (GAN). Figure 1 shows the flow for a ML-guided well generation framework, `WellGAN` [10]. Given a placement, it extracts the oxide diffusion (OD) layer shapes as input features to denote the locations for the transistors. The OD shapes inside the in-well transistors, such as PMOS transistors, are encoded as the red channel of the input image, while OD shapes inside the out-well transistors, such as NMOS transistors, are extracted into the green channel. The input images are then fed to a conditional GAN model, resulting in the well guidance through model inference. The well guidance then is extracted, rectilinearized, and legalized into rectilinear polygons representing the generated wells.

In this work, we propose to leverage the such well generation framework to guide our well-aware placement engine. Instead of generating wells after the placement, the placement engine iteratively interacts with the well generation procedures to update the well guidance.

B. Well Proximity Effect

Well proximity effect (WPE) changes the uniformity of doping concentration and shifts the threshold voltage (V_{th}) of the devices near the well edges. The WPE effect-induced V_{th} shifts for a finger of transistor can be decomposed into two directions: the horizontal and the vertical [6] as shown in Eq. 1.

$$\begin{aligned}\hat{P}_x &= r_1^2\left(\frac{1}{U^L(U^L+L)} + \frac{1}{U^R(U^R+L)}\right),\\ \hat{P}_y &= r_1^2\left(\frac{1}{U^B(U^B+W)} + \frac{1}{U^T(U^T+W)}\right),\end{aligned} \tag{1}$$

where r_1 is a technology-dependent constant, W (L) is the channel width (length), and U^L, U^R, U^B and U^T are the distance from the channel to the well edges as shown in Fig. 2. The V_{th} shift is the sum of $\hat{P}_x$ and $\hat{P}_y$. The V_{th} shift in a multiple-finger transistor is viewed as the average of that in each finger.

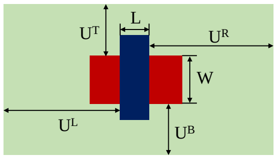


Fig. 2: Parameters for the WPE model.

In this work, we propose to handle WPE by enforcing hard constraints on device-to-well edge distances, instead of formulating as an objective as proposed by [6], [9]. By ensuring the V_{th} is below the tolerance, the circuit performance can be preserved against WPE. We constrain the minimum vertical distances, U^B and U^T, based on the channel width of the finger, and we constrain the minimum horizontal distances, U^R and U^L, based on the channel length.

We obtain the required device to well edge distances through the following experiment. We perform DC simulation on the test circuit (Fig. 3 (a)) with different layout implementations of the transistor. We change the vertical (horizontal) device to well edge distances, keep the horizontal (vertical) distances and length (width) larg, and record the resulting V_{th} deviation from the pre-layout simulation. Figure 3 (b) shows the V_{th} deviation for three transistors with the same channel length but different widths. For the WPE constraints, we select the minimum distances so that the V_{th} shift in this experiment is lower than 1 mV. For the cases that V_{th} deviation cannot reach 1 mV, we select the minimum distance at the point where V_{th} deviation converges. To find where it converges, we set U^T (U^L) to be large and sweep the U^B (U^R). We compare the resulting V_{th} deviation from the previous experiment. We set the required distance where difference of V_{th} shift in two experiments is lower than 0.1 mV. Intuitively, this convergence criterion means the device to well edge distances has little impact on V_{th}.

We obtain the minimum horizontal (vertical) distances over different lengths (widths) for the technology through the experiments above. For each transistor, its minimum U^T and U^B (U^L and U^R) are then set based on its finger channel width (length) in the rest of this paper.

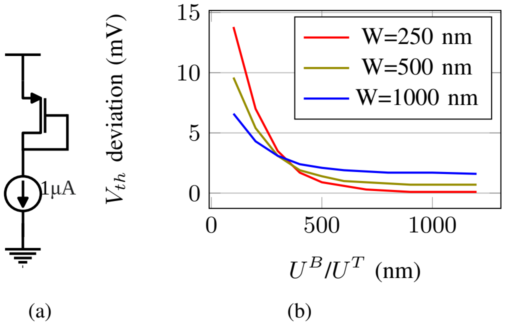


Fig. 3: Simulation on WPE-induced V_{th} shift. (a) The schematic of the experiment test circuit. (b) The resulting V_{th} shifts for different channel widths and device to well edge distances.

C. Problem formulation

The well-aware analog placement problem can be formulated as follows. Given a set of modules $M = \{m_1, \ldots m_{|M|}\}$ with widths,

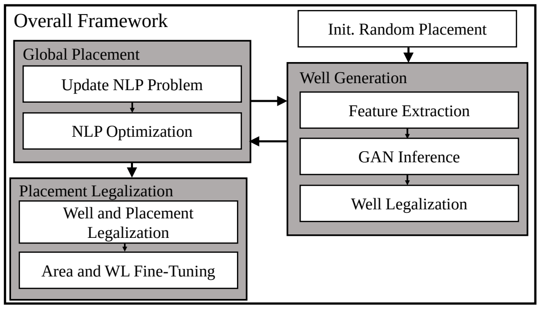
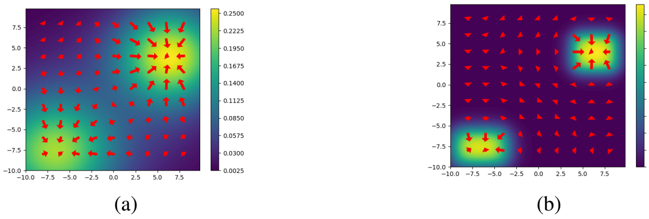


Fig. 4: The overall flow of the framework.



(a)

(b)

Fig. 5: Examples of the fence region cost field. Red arrows denote the gradient. (a) $\gamma = 1.0$. (b) $\gamma = 0.3$.

heightsk and well types, a set of nets $N = \{n_1, \ldots n_{|N|}\}$, a set of self-symmetry constraints $SS = \{ss_1, \ldots ss_{|SS|}\}$, a set of symmetry pair constrains $SP = \{sp_1, \ldots sp_{|SP|}\}$, minimum WPE distances for transistors and design rules, determine the coordinates of all modules and generate the wells such that no modules are overlapped, no wells are overlapped, all modules are correctly placed inside or outside the wells, minimum device to well edge distances are all satisfied, all the symmetry constraints are met and no design rule is violated.

III. Algorithms

In this section, we present the algorithm of the proposed framework. Sec. III-A describes the overall flow of the framework, and the rest of this section explains each component in detail. Sec. III-B presents the algorithm for the non-linear programming-based (NLP) global placement engine. Sec. III-C explains the well generation procedures, including the GAN inference and well legalization. Finally, Sec. III-D gives the legalization algorithms for legalizing the placement in awareness of wells.

A. Overall Flow

As shown in Fig. 4, the proposed framework mainly consists of three components: global placement, well generation, and placement legalization. The three components interact with each other and compose the complete well-aware placement flow.

The framework begins with initializing a random and highly congested placement. Then the GAN-guided well generation engine produces the initial well guides. The generated wells are considered as fence regions in the non-linear programming-based (NLP) global placement engine. The NLP global placement engine optimizes a non-linear function composed with wirelength, area, fence region, overlapping, and asymmetry costs. After the optimization converges or the locations of the modules have significantly changed, the current global placement iteration terminates. Then wells are re-generated, fence regions are updated, and another global iteration begins. This loop continues until the overlapping and asymmetry are both low. After the loop ends, the placement legalization stage then legalizes the placements in honor of design rules, WPE constraints, and symmetry.

Each iteration of well generation and global placement updates the wells, and the wells guide the spreading together with the wirelength and area. As a result, the global placement spreading process can optimize wirelength and area while effectively sharing the wells by following the guidance from the GAN model.

B. Global Placement

The well-aware global placement treats wells as fence regions. Inspired by several existing studies in analog placement [6], [14], [15], we solve the global placement by optimizing a non-linear programming (NLP) objective function. The non-linear objective function includes costs of half-perimeter wirelength (HPWL), area (AREA), violation of fence region (FENCE), overlapping (OVL), and asymmetry (ASYM). We introduce multipliers on the overlapping and asymmetry costs to control the module spreading process. In summary, the NLP global placement can be formulated as a minimization problem, as stated in Eq. 2.

$$\begin{aligned} &\underset{\boldsymbol{x},\boldsymbol{y}}{min} f^{HPWL}(\boldsymbol{x},\boldsymbol{y}) + \alpha \cdot f^{AREA}(\boldsymbol{x},\boldsymbol{y}) + \beta \cdot f^{FENCE}(\boldsymbol{x},\boldsymbol{y}) \\ &\quad + \lambda_{OVL} \cdot \Phi^{OVL}(\boldsymbol{x},\boldsymbol{y}) + \lambda_{ASYM} \cdot \Phi^{ASYM}(\boldsymbol{x},\boldsymbol{y}), \end{aligned} \tag{2}$$

where α and β are weighing constants and λ denotes the multipliers. The multipliers are gradually increased over iterations to emphasize more on solving the hard constraints. In each iteration, we re-generate the well fence regions based on the current module locations. The global placement stage ends when the overlapping and asymmetry are lower than the preset thresholds. The rest of Sec. III-B explains each cost and gives the details of the algorithm.

We use the cost formulations for HPWL, AREA, OVL and ASYM from the work [15]. We propose a new fence region cost to consider wells in global placement.

FENCE: The Fence region cost function attracts the modules into the correct well region and expels the others out. We construct a cost field using `sigmoid` functions for each type of well, and the fence region cost for each cell is inferred from the fields. To be specific, the attractive cost for each well type is defined as shown in Eq. 3.

$$\begin{aligned} f_i^{FENCE} = &\sum_{m_j \in W_i} \sum_{p_k \in W_i} (f_s((x_j - X_k^l) \cdot \gamma) \cdot f_s(-(x_j - X_k^h) \cdot \gamma) \\ &\cdot f_s((y_j - Y_k^l) \cdot \gamma) \cdot f_s(-(y_j - Y_k^h) \cdot \gamma)), \\ f_s(x) = &\frac{1}{1 + e^{(-x)}} \end{aligned} , \tag{3}$$

where x_j and y_j denote the x and y coordinates of module m_j of the well type i, X_k^l, X_k^h, Y_k^l and Y_k^h define the bounding box of a well polygon p_k, and γ is a parameter to control the steepness of the cost. The four `sigmoid` functions constructs a field across the layout to attract the cells into the regions of wells. Fig. 5 shows an example of a cost field with two well polygons. Large γ is mainly used in early iterations to give a more global view (see Fig. 5 (a)). After the placement is less congested in late iterations, we use a smaller γ to represent the wells more accurately (see Fig. 5 (b)). We update γ each global placement iteration to control the steepness of the cost. γ is determined linearly based on the overlapping ratio. The overlapping ratio is the ratio of overlapping module area over the total module area. γ is trimmed with the max and min allowed values, and they are selected to be 1.2 and 0.2 in the experiments.

The attractive cost can also be modified to expel the modules out of the well. We use $1 - f^{FENCE}$ to compose the expelling force for the other modules. However, we only apply the expelling cost in late iterations. Intuitively, this trick is intended to avoid converging too fast before exploring a more extensive search space. To be specific, we do not use expelling cost if the overlapping ratio is above 0.3.

We optimize the overall NLP cost function with the ADAM optimizer [16]. Each optimization iteration can terminate in two ways: the optimization converges, or the placement has significantly changed from the previous iteration. The second stop criterion intends to update the well fence regions to follow the placement process closely. We measure the L1 distance of the module locations between the current and those of the last iteration. The criterion is triggered if the distance is above a preset value. In other words, it is triggered when

$$\|(\boldsymbol{x}, \boldsymbol{y}) - (\boldsymbol{x}^*, \boldsymbol{y}^*)\|_1 > \delta,$$

where $(\boldsymbol{x}^*, \boldsymbol{y}^*)$ denotes the module coordinates from the last iteration and δ is a constant.

The multipliers are updated using the subgradient method [17]. However, if the second criterion terminates the previous iteration, we do not update the multipliers. Intuitively, the non-linear optimization has not converged at the moment, and we therefore postpone the updates of multipliers.

C. Well Generation

The well generation subroutine provides well guidance to the global placement and generates the wells for the final layouts. It generates the wells leveraging the ML-guided well generation and legalizes them for design rules, contacts, and WPE constraints. Algorithm 1 shows the flow of the well generation process. We first extract the OD shapes into images for each well type and generate the rectilinear polygons using the `WellGAN` framework (Line 4-9). Then we assign each module of the well type to its closest well based on the Manhattan distance of center points (Line 10). Each well shape is expanded to handle minimum device to well edge distance constraint for the modules assigned (Line 12). A set of contacts are created to provide bulk voltage for each well (Line 12). Finally, we apply patches to the well polygons for more regular shapes and remove design rule violations (Line 14). The rest of this section describes the details of these procedures.

1) Generate Rectilinear Wells: We generate the rectilinear well polygons using the ML-guided well generation as introduced in Sec. II-A. However, we add special handling on the out-well OD image G. Since the GAN model is trained with legal manual layouts, it is crucial to align the inference inputs with the same scope of its training data. However, highly overlapped in-well and out-well OD shapes empirically are likely to results in unreasonable outputs. Therefore, we keep the out-well OD channel empty if the ratio of the overlapped area over the total module area is above a constant ϵ. The constant ϵ is selected to be 0.3 in the experiments. After the placement has reasonably spread out, out-well shapes are considered to give more accurate well guidance.

2) Legalize minimum WPE distance: We enforce the minimum device to well edges distances by legalizing the polygon geometries. For each module assigned and inside to the polygon, we first obtain the required horizontal and vertical distance based on its finger channel length and finger channel width as described in Sec. II-B. We expand the bounding boxes of the modules by the required distances and merge them with the polygon.

Algorithm 1 *Well Generation and Legalization*

```
Require: A placement $L$ and a set of well types $W$.
Ensure: A list of polygon set $P$ of wells and a set of contacts $C$.
 1: $P \leftarrow \emptyset$; $C \leftarrow \emptyset$;
 2: Overlapping ratio $r \leftarrow ComputeOverlappingRatio(L)$;
 3: for each $w_i \in W$ do
 4:     In-well OD shapes $R \leftarrow ExtractInWellOD(L, w_i)$;
 5:     Out-well OD shapes $G \leftarrow \emptyset$;
 6:     if $r < \epsilon$ then
 7:         $G \leftarrow ExtractOutWellOD(L, w_i)$
 8:     Rectilinear well polygons $P_i \leftarrow WellGAN(R, G)$;
 9:     Assign modules to their cloest well $M \leftarrow FindClosest(P_i)$;
10:     for each $p_{i,j} \in P_i$ do
11:         $p_{i,j} \leftarrow LegalizeWPE(p_{i,j}, M_j)$;
12:         $p_{i,j}, c_{i,j} \leftarrow GenerateContact(p_{i,j})$; $C \leftarrow C \cup c_i, j$;
13:         $p_{i,j} \leftarrow Patch(p_{i,j})$;
14:     $P.append(p_i)$;
15: return $P, C$;
```

3) Contact Generation: Contacts are used to connect the wells to external voltage and provide potential to transistor bulks.

In this work, we follow manual layout design practice to insert as many contacts as possible as long as they do not introduce area overhead. To be specific, we cast the problem as follows. Given a set of contact templates T, where each template has weight, width and height, and the minimum spacing between contact and other layout edges, find a maximum weight set of contacts with the lowest cost available. The cost is defined as the sum of two areas: (1) the area overhead introduced to the well, and (2) the area overlaps of the contact with other modules in the placement.

We first generate contact candidates by sweeping the locations of available templates and prune those overlapping with in-well modules. We calculate the cost for each candidate and their pair-wise exclusive relationship. If there is no or only one zero-cost candidate, we return the one with the lowest cost. Otherwise, we choose the maximum weight independent set of the zero-cost candidates. The candidate selection problem can be formulated as a variant of the maximum weight independent set problem on trapezoid graphs, where the spacing needs also be considered besides the rectangle geometries. The problem can be solved in $O(n^2)$ time complexity using the algorithm from [18]. Empirically, zero-cost candidates exist for most of the time. We merge the contact geometry expanded by the required spacing with the well polygon if needed.

In the end of well generation stage, we resolve the remaining spacing design rule violations by patching the polygon geometries.

D. Placement Legalization

The placement legalization stage legalizes the placement to enforce non-overlapping, spacing, and symmetry constraints to placement. In this work, we extend the legalization algorithm proposed in the work [14] to consider wells. It is mainly composed of two subroutines: the constraint graph construction and the linear-programming-based (LP) legalization. The constraint graph construction captures the positional relationship of the modules and wells. There are two types of constraint graphs in this work: the redundant and the irredundant. On the other hand, the LP legalization takes a constraint graph, treats positional relations as constraints, and compacts the placement. The LP legalization can have an objective of either area and wirelength.

Algorithm 2 *Placement Legalization*

```
Require: A global placement $L$.
Ensure: A legalized placement $L$ with well generated.
 1: Clear current wells $L.well \leftarrow \emptyset$;
 2: A redundant constraint graph $G_0 \leftarrow RCG(L)$;
 3: Extra spacing $S \leftarrow 0$;
 4: $L \leftarrow AreaDrivenCompaction(L, G_0, S)$;
 5: $L.well \leftarrow WellGeneration(L)$;
 6: while True do
 7:     A horizontal irredundant constraint graph $G_1^H \leftarrow ICG(L)$;
 8:     if $AreaDrivenCompaction(L, G_1^H, S)$ is infeasible then
 9:         Continue;
10:     An irredundant vertical constraint graph $G_1^V \leftarrow ICG(L)$;
11:     if $AreaDrivenCompaction(L, G_1^V, S)$ is feasible then
12:         $L \leftarrow AreaDrivenCompaction(L, G_1^V, S)$; break;
13:     $S \leftarrow S + GridStep$;
14: An Irredundant constraint graph $G_1 \leftarrow ICG(L)$;
15: $L \leftarrow WireLengthDrivenCompaction(L, G_1)$;
16: return $L$;
```

Algorithm 2 illustrates the core steps in the placement legalization. Given a global placement result, we first legalize the placement in the absence of the wells (Line 1-4). In this step, we use the irredundant constraint graph to preserve more positional relations from the global placement to avoid significantly deviating from the global placement well guidance. Then we generate the wells and legalize them using the algorithms presented in Sec. III-C (Line 5). After generating legal wells, we perform area-driven LP legalization in awareness of the wells (Line 6-14). As the well shapes might be complicated and contradict the symmetry constraints, we repeat the LP legalization and iteratively add extra spacing between the modules until we find a feasible solution. Empirically, the proposed framework rarely need multiple iterations. But this fixing loop is frequently triggered if the global placement is not well-aware. After obtaining a legal placement, the HPWL-driven compaction is used to fine-tune the placement (Line 17-18). The rest of this section explains the details in the constraint graph construction (Sec. III-D1) and the LP-based compaction (Sec. III-D2).

1) Constraint Graph Construction: Constraint graphs capture the positional relations between modules and wells from the current placement. In this work, constraint graphs are later translated into linear constraints on positional relations of movable objects for the LP-based legalization.

A constraint graph either describes the horizontal relations or the vertical. An edge in horizontal constraint graphs from node n_i and n_j implies that n_i is placed to the left of n_j. Similarly, a edge (n_i, n_j) in a vertical constraint graph indicates that n_i is below n_j. We use two methods to construct constraint graphs depending on the need: the redundant and the redundant.

The redundant constraint graph construction is used right after the global placement. As the global placement result still consists of minor overlapping and asymmetry, we first resolve the two hard constraints without considering the wells. In this step, we keep the redundant constraint edges to preserve the overall positional relationship between modules amid the legalization. Fig. 6 (a) shows an example of redundant constraint graph construction. For each pair of modules n_i and n_j, we add at least one constraint edge to the horizontal and vertical constraint graphs. The edge we add falls into four cases. **Case 1:** If n_i and n_j are overlapped, we add either a horizontal or a vertical edge. A horizontal edge is added when the vertical overlapping distance is shorter and vice versa. The edge (n_1, n_2) in Fig. 6 (a) shows an example. **Case 2:** If n_i and n_j are disjointed and either $x_i^h > x_j^l$ or $x_j^h > x_i^l$, we add a vertical edge. The edge (n_1, n_3) in Fig. 6 (a) shows an example. **Case 3:** If n_i and n_j are disjointed and either $y_i^h > y_j^l$ or $y_j^h > y_i^l$, we add a horizontal edge. The edge (n_3, n_4) in Fig. 6 (a) shows an example. **Case 4:** If n_i and n_j are not belonging to any of the cases above, we add both the horizontal and the vertical edge. The edge (n_1, n_4) in Fig. 6 (a) shows an example.

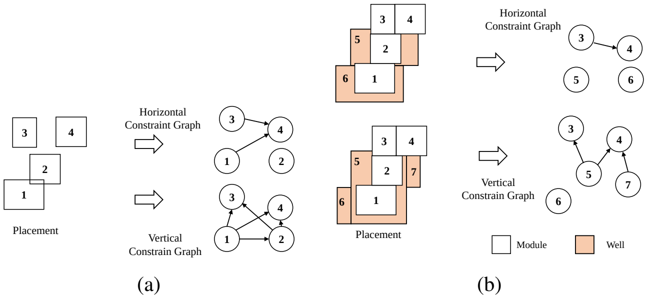


Fig. 6: Examples of constraint graph construction. (a) Redundant constraint graph. (b) Irredundant constraint graph construction for well-aware legalization.

The irredundant constraint graph construction, on the other hand, targets to have the minimal constraint edges presented so that the LP-based compaction can explore larger solution space. The irredundant constraint graph construction also needs to consider the wells. Fig. 6 (b) shows an example of irredundant constraint graph construction. In this example, modules 1 and 2 are assigned to the well, while modules 3 and 4 need to be moved out of the well. We ignore the in-well modules in irredundant constraint graph construction. The in-well modules are moved together with the well in the LP-based compaction. Their positional relationship is indirectly inferred without the need to present an edge in constraint graphs. The procedures for constructing an irredundant constraint graph are presented as follows. We first split the well polygon into rectangles by applying horizontal or vertical slicing. Horizontal slicing is used when generating the horizontal constraint graph and vice versa. We then construct the constraint graph based on the plane sweep line algorithm from [19]. The horizontal edges between overlapped nodes are exempted if their vertical overlapping distance is shorter than the horizontal. The exempted positional relations are to be later handled by the vertical constraint graph. For example, the edges (n_5, n_3) and (n_5, n_4) are not added to the horizontal constraint graph in Fig. 6 (b).

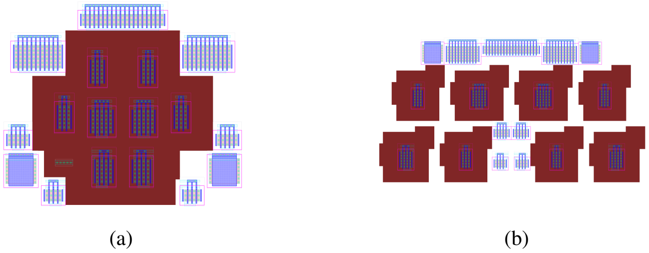
2) LP-based Legalization: After obtaining the constraint graphs, the LP-based legalization then legalizes and compacts the layout. The LP-based legalization works on one dimension each time. At each compaction step, we solve a linear programming problem to compact the layout to honor the constraints while optimizing either area or HPWL. After compacting both the horizontal and vertical direction, a legal placement is found.

In the first pair of LP problems (Line 4 in Algorithm 2), the same LP formulations are used as proposed by [14]. The LP constraint honors the symmetry constraints as well as resolving the overlapping between modules. The objective is set to optimize for either area or HPWL. Due to the page limit, we omit the details.

In the later well-aware legalization (Line 8, 12, 13 and 18 in

TABLE I: Comparison of area(μm^2), half-perimeter wirelength (HPWL(μm)) and runtime (RT(s)).

CKTS	Individual wells			[10]			This work		
	Area	HPWL	RT	Area	HPWL	RT	Area	HPWL	RT
OTA1	360.2	72.3	**1.3**	318.0	68.7	3.2	**290.3**	**60.3**	3.6
OTA2	756.2	234.7	**4.8**	750.7	**203.1**	7.9	**599.0**	205.2	10.6
OTA3	1055.4	586.6	48.9	1325.6	**559.5**	43.2	**965.6**	651.3	**34.1**
OTA4	3255.2	837.1	**39.7**	3313.6	**799.6**	40.1	**3033.7**	866	42.6
COMP1	175.1	78.8	**2.0**	144.4	95.1	6.6	**82.2**	**61.8**	3.5
COMP2	192.2	93.1	**3.0**	194.2	105.0	5.6	**84.7**	**48.1**	3.6
BOOTSTRAP	177.9	64.5	**2.0**	130.8	83.4	5.0	**97.5**	**63.2**	4.8
RDAC	361.5	209.2	**12.4**	370.4	287.0	30.2	**144.3**	**137**	23.7
Norm.	1.82	1.26	**0.64**	1.74	1.46	1.33	**1.00**	**1.00**	1.00



(a)

(b)

Fig. 7: Generated COMP layouts. The wells are in crimson. (a) This work. (b) "WellGAN after placement".

Algorithm 2), we treat each module and rectangle split from the well polygon as a movable object. The positional relations of out-well modules and wells are enforced similar to the previous stage by following the redundant constraint graphs. In addition, we restrict all the in-well modules to move together with the well. As the overlapping and spacing of in-well modules have been resolved in the earlier legalization steps, the legal placement is achieved without altering the relative positions for in-well modules. We add additional well movement constraints to the previous LP formulation. The well movement constraints restrict that the split well rectangles and the in-well modules move in the same distance and direction as follows.

Well Movement constraint: For each well polygon P_i, we add a set of constrains as shown in Eq. 4

$$x_j - x_i^0 = x_j^*, \forall n_j \in P_i, \tag{4}$$

where n_j denotes a module inside P_i or a rectangle split from P_i, x_j denotes the x or y coordinate of n_j, x_i^0 is a variable to represent the x or y movement of the entire well P_i, and x_j^* denotes the x or y coordinate of n_j before solving the LP problem. By integrating this constraint and the variable x_i^0 for each well polygon into the LP formulation from [14], the LP compaction can work together with the constraint graph construction to legalize the final layout.

IV. Experimental Results

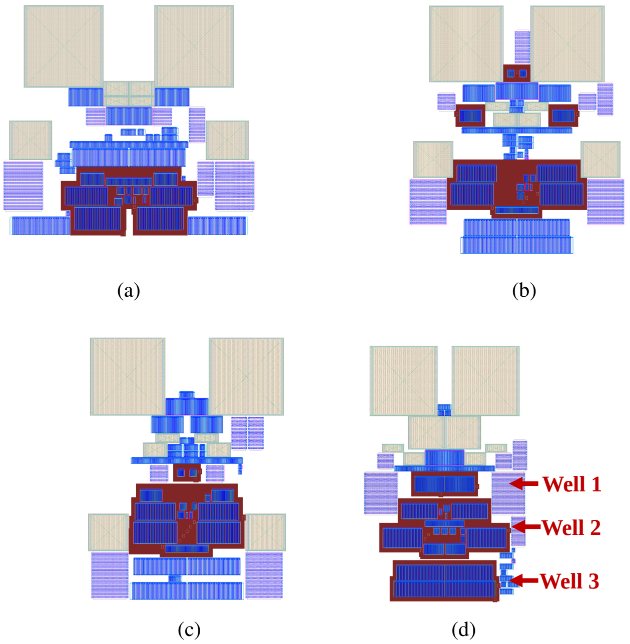
The proposed framework is implemented in C++ and Python. All experiments are conducted on a Linux workstation with an Intel Core i9-7900X 3.0 GHz CPUz and one NVIDIA TITAN Xp GPU. We use ten threads in the CPU for the placement framework and GPU for neural network inference. We obtained the training data and source codes from [10]. We re-trained the model and used the same model in all experiments.

We conduct experiments on four operational transconductance amplifier (OTA) with different architectures (OTA1, OTA2, OTA3, and OTA4), two comparators with the same topology but with different sizing (COMP1 and COMP2), a bootstrap switch (BOOTSTRAP), and a resistor digital-to-analog converter unit (RDAC). All PMOS transistors have VDD bulk voltage within these designs, and all NMOS transistors have VSS voltage. In other words, there is only one well type for the designs, and all PMOS are assigned to this type. We further verify the framework for multiple well types cases in a separate experiment discussed later in this section. All the benchmark circuits are under TSMC 40nm technology. The required WPE distances are obtained as described in Sec. II-B. We verify the placement design rules with Mentor Calibre nmDRC.

To evaluate the effectiveness of our proposed framework, we compare area, wirelength, and runtime with two baselines. The first baseline uses individual wells, denoted as "individual wells" later in this section. In this baseline, we close the well-awareness in the framework. We increase the area of each PMOS transistor before the placement so that its shape is equivalent to having an individual well added. The "individual well" approach is widely adopted in existing analog layout synthesis frameworks, such as MAGICAL [5]. The second baseline is to generate wells after placement using GAN-based well generation, denoted as "WellGAN after placements". To comply with WPE distance and contact insertion requirements, we perform the same legalization methods from this work after the well generation in "WellGAN after placements". It enforces the same WPE distance requirement on this baseline.

We compare this work with two baselines on area, half-perimeter wirelength (HPWL) and runtime. Table I shows the results on eight benchmark circuits. On the average of ratios, this work reduces area (HPWL) by 82% (26%) over "individual wells" and 74% (46%) over "WellGAN after placements". The runtime is comparable. Fig. 7 and Fig. 8 show the generated layouts of COMP2 and OTA4 from this work and "WellGAN after placements". From the results, we observe that the area reduction is more effective for smaller designs. The reason is that larger analog designs often include large transistors, capacitors, and resistors and well areas become less significant in percentage. We also observe that the "WellGAN after placements" baseline results in lower HPWL on OTA2, OTA3, and OTA4 over the proposed framework, despite significant area overhead. We notice that as white space in wells becomes less critical in larger layouts, splitting wells sometimes results in lower HPWL but higher area. Fig. 8 (c) shows an alternative OTA2 placement from the proposed framework to verify our understanding. It is obtained by lower the ratio of area cost over HPWL cost in the non-linear global placement engine, and it has an area of 3031.6 μm^2 and HPWL of 695.8 μm.. We also notice that the area of the baseline "WellGAN after placements" sometimes is even worse than the "Individual wells". The primary reason behind this is that the placement sometimes places PMOS and NMOS transistors interleaving with each other, and hence the legalized results are likely to fail in sharing the wells. Fig. 7 (b) shows an example of this issue. As it is not well-aware, the baseline placement places two NMOS transistors at the center surrounded by PMOS transistors. After legalizing the wells, the wells and transistors overlap, and the legalization subroutine increases the spacing between devices until a legal solution is found. As a result, the layout in Fig. 7 (b) assigns the individual well to each PMOS transistor and creates area overhead. It also increases the runtime due to iteratively legalization efforts.

We then verify the function of considering multiple-well types. Although all benchmark circuits only have one well type in this section, giving different bulk voltage and using particular transistor types (such as deep N-well devices) requires separating the wells into several groups. We manually add two additional well types to OTA4 to verify our proposed framework. Fig. 8 (d) shows the


(a)

(b)

(c)

(d)

Fig. 8: Generated OTA4 layouts. The wells are in crimson. (a) This work. (b) "`WellGAN` after placement". (c) This work with lower area-to-HPWL cost ratio. (d) This work with two additional well types.

TABLE II: Comparsion of threshold voltage (mV) on COMP.

	M1	M2	M3	M4	M5	M6	M7	M8
Schematic	-435.0	-435.0	-566.5	-566.5	-558.6	-558.6	-563.5	-563.5
[10]	-450.2	-451.1	-573.3	-573.9	-561.9	-562.4	-580.1	-570.0
This Work	-441.8	-442.1	-562.9	-563.0	-561.8	-561.8	-559.4	-559.5

resulting placement. In addition to the original PMOS well ("Well 2" in the figure), we add two NMOS transistors and four other NMOS transistors to two new well types, respectively. The resulting layout clearly shows two separate wells for the two types, as "Well 1" and "Well 3" in the figure. The area, HPWL, and runtime for this experiment are 3023.3 μm^2, 1070.5 μm, and 42.2 seconds, respectively.

We then verify the effectiveness of our framework on mitigating the WPE effect. We compare two different implementations of wells on the same placement of COMP. The first is to generate the wells directly from `WellGAN`. The second is to use our framework, where the WPE distance constraints are satisfied. We perform a post-layout simulation to obtain the threshold voltages for each PMOS device. Table II shows the results. COMP2 have eight PMOS transistors and they compose four symmetry pairs, i.e., $(M1, M2)$, $(M3, M4)$,$(M5, M6)$ and $(M7, M8)$. From the table, we can observe that the V_{th} deviations from the schematic reduce with our proposed method. Furthermore, our proposed framework effectively keeps V_{th} for the two transistors in a symmetry pair close. On the other hand, there are significantly V_{th} difference in $(M7, M8)$ for the baseline. Therefore, the proposed methodology of constraining minimum WPE distance benefits the robustness of circuit performance.

V. Conclusion

This work proposes a new generative adversarial network-guided well-aware analog placement framework. Our methods leverage the state-of-the-art machine learning techniques in the well generation to consider the wells in placement. Experimental results demonstrate the proposed framework provides notable benefits in area and wirelength by encouraging sharing wells in analog circuits.

Acknowledgement

This work is supported in part by the NSF under Grant No. 1704758, and the DARPA IDEA program.

References

[1] H. Chen, M. Liu, X. Tang, K. Zhu, N. Sun, and D. Z. Pan, "Challenges and opportunities toward fully automated analog layout design," *Journal of Semiconductors*, vol. 41, no. 20070021, p. 111407, 2020.

[2] M. P.-H. Lin, Y.-W. Chang, and C.-M. Hung, "Recent research development and new challenges in analog layout synthesis," in *Proc. ASPDAC*, 2016.

[3] K. Kunal, M. Madhusudan, A. K. Sharma, W. Xu, S. M. Burns, R. Harjani, J. Hu, D. A. Kirkpatrick, and S. S. Sapatnekar, "ALIGN: Open-source analog layout automation from the ground up," in *Proc. DAC*, 2019.

[4] B. Xu, K. Zhu, M. Liu, Y. Lin, S. Li, X. Tang, N. Sun, and D. Z. Pan, "MAGICAL: Toward fully automated analog IC layout leveraging human and machine intelligence," in *Proc. ICCAD*, 2019.

[5] H. Chen, M. Liu, X. Tang, K. Zhu, A. Mukherjee, N. Sun, and D. Z. Pan, "MAGICAL 1.0: An open-source fully-automated ams layout synthesis framework verified with a 40-nm 1 GS/s $\Delta\Sigma$ ADC," in *Proc. CICC*, 2021.

[6] H.-C. Ou, K.-H. Tseng, J.-Y. Liu, I.-P. Wu, and Y.-W. Chang, "Layout-dependent effects-aware analytical analog placement," *IEEE TCAD*, vol. 35, no. 8, pp. 1243–1254, 2016.

[7] J. M. Cohn, D. J. Garrod, R. A. Rutenbar, and L. R. Carley, "KOAN/ANAGRAM II: New tools for device-level analog placement and routing," *IEEE Journal Solid-State Circuits*, vol. 26, no. 3, pp. 330–342, 1991.

[8] S. Nakatake, M. Kawakita, T. Ito, M. Kojima, M. Kojima, K. Izumi, and T. Habasaki, "Regularity-oriented analog placement with diffusion sharing and well island generation," in *Proc. ASPDAC*, 2010.

[9] R. Martins, N. Lourenço, R. Póvoa, and N. Horta, "On the exploration of design tradeoffs in analog IC placement with layout-dependent effects," in *International Conference on Synthesis, Modeling, Analysis and Simulation Methods and Applications to Circuit Design (SMACD)*, 2019.

[10] B. Xu, Y. Lin, X. Tang, S. Li, L. Shen, N. Sun, and D. Z. Pan, "WellGAN: Generative-adversarial-network-guided well generation for analog/mixed-signal circuit layout," in *Proc. DAC*, 2019.

[11] X. Dong and L. Zhang, "EA-based LDE-aware fast analog layout retargeting with device abstraction," *IEEE TVLSI*, vol. 27, no. 4, pp. 854–863, 2019.

[12] K. Zhu, M. Liu, Y. Lin, B. Xu, S. Li, X. Tang, N. Sun, and D. Z. Pan, "GeniusRoute: A new analog routing paradigm using generative neural network guidance," in *Proc. ICCAD*, 2019.

[13] Y. Li, Y. Lin, M. Madhusudan, A. Sharma, W. Xu, S. Sapatnekar, R. Harjani, and J. Hu, "A customized graph neural network model for guiding analog ic placement," in *Proc. ICCAD*, 2020.

[14] B. Xu, S. Li, C.-W. Pui, D. Liu, L. Shen, Y. Lin, N. Sun, and D. Z. Pan, "Device layer-aware analytical placement for analog circuits," in *Proc. ISPD*, 2019.

[15] K. Zhu, H. Chen, M. Liu, X. Tang, N. Sun, and D. Z. Pan, "Effective analog/mixed-signal circuit placement considering system signal flow," in *Proc. ICCAD*, 2020.

[16] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," in *Proc. ICLR*, 2015.

[17] C. Lemaréchal, "Lagrangian relaxation," in *Computational Combinatorial Optimization*, M. Jünger and D. Naddef, Eds. Springer, 2001, pp. 112–156.

[18] M. Hota, M. Pal, and T. K. Pal, "An efficient algorithm for finding a maximum weight k-independent set on trapezoid graphs," *Computational Optimization and Applications*, vol. 18, no. 1, p. 49–62, Jan. 2001.

[19] J. Doenhardt and T. Lengauer, "Algorithmic aspects of one-dimensional layout compaction," *IEEE TCAD*, vol. 6, no. 5, pp. 863–878, 1987.