WellGAN: Generative-Adversarial-Network-Guided Well Generation for Analog/Mixed-Signal Circuit Layout

Biying Xu, Yibo Lin, Xiyuan Tang, Shaolan Li, Linxiao Shen, Nan Sun, and David Z. Pan
ECE Department, University of Texas at Austin
{biying, yibolin, xitang, slliandy, lynn.shenlx}@utexas.edu, nansun@mail.utexas.edu, dpan@ece.utexas.edu

ABSTRACT

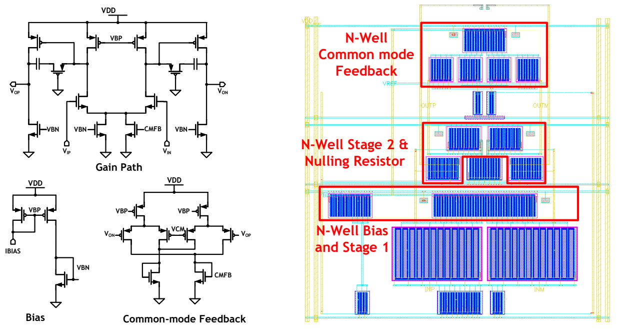
In back-end analog/mixed-signal (AMS) design flow, well generation persists as a fundamental challenge for layout compactness, routing complexity, circuit performance and robustness. The immaturity of AMS layout automation tools comes to a large extent from the difficulty in comprehending and incorporating designer expertise. To mimic the behavior of experienced designers in well generation, we propose a generative adversarial network (GAN) guided well generation framework with a post-refinement stage leveraging the previous high-quality manually-crafted layouts. Guiding regions for wells are first created by a trained GAN model, after which the well generation results are legalized through post-refinement to satisfy design rules. Experimental results show that the proposed technique is able to generate wells close to manual designs with comparable post-layout circuit performance.

1 INTRODUCTION

Analog/mixed-signal (AMS) integrated circuit (IC) layout design relies heavily on human experience due to the lack of effective optimization approaches. Current AMS layout automation tools would generate solutions inconsistent with designer behavior. Practically, with such solutions, it would require significant overhead to unravel and debug the problematic chips if any failure happens. As a consequence, automatic AMS layout tools have not been widely adopted. Therefore, it is highly desirable to develop practical AMS layout tools following designer experience and esthetics to ensure chip functionality and performance after tape-out.

Well layers are crucial mask layers that define the doping area serving as the bulk of MOSFETs. Thus, well generation is essential in establishing the bulk regions which will affect the subsequent optimization stages. Different from a digital circuit layout where the well regions are pre-designed in the standard cell layouts and automatically connected through cell abutting, well regions for analog layouts usually need to be distinctively drawn. Although some process design kits (PDKs) provide parametric cells with pre-drawn well regions, these preliminary shapes often need to be properly connected by extending the well regions or inserting well contacts and routing to pass layout versus schematic (LVS) and design rule check (DRC). Well generation can influence the layout compactness and the routing complexity due to the well spacing and the interconnection of well contacts. Moreover, the geometry of wells can influence the well proximity effect (WPE) [1], the substrate noise coupling [2], etc., potentially degrading the performance and robustness of AMS circuits. Given the above constraints, manual AMS layout well generation practice generally requires careful design with designer experience and insights to ensure the circuit functionality and robustness. The manual layout of a two-stage miller-compensated operational amplifier (op-amp) shown in Fig. 1 provides an example of such practice, where the capacitors are omitted to conserve space.

DAC '19, June 2–6, 2019, Las Vegas, NV, USA
© 2019 Association for Computing Machinery.
ACM ISBN 978-1-4503-6725-7/19/06…$15.00
https://doi.org/10.1145/3316781.3317930



(a) circuit (b) layout
Figure 1: Op-amp circuit and layout example.

While the placement and routing problems for AMS circuits have been actively explored recently [1, 3–5], well generation remains an unresolved challenge. Prior work [6] generated wells with a series of simple computational geometry operations, including geometric expansion and union. More recent works [1, 7] imposed rectangular-shaped constraints during well generation. Nevertheless, the previous approaches didn't consider the designer expertise to guide the well region optimization. Recent advances in deep learning have achieved great success in learning domain-specific knowledge from existing images and generating new ones mimicking the learned styles [8, 9]. The techniques have been widely applied to image generation, image-to-image translation, etc. Thus, it would be highly beneficial if the layout automation tools can learn the designer experience and expertise brought by the existing high-quality manual designs leveraging deep learning techniques.

In this paper, we propose *WellGAN*, a generative adversarial network (GAN) guided well generation framework. Given the previous high-quality manual layouts by experienced designers, WellGAN attempts to incorporate the designer expertise by mimicking their layout behavior via a conditional-GAN model. With a lightweight

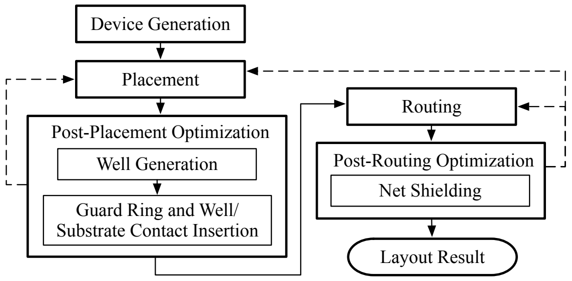

Device Generation
Placement
Post-Placement Optimization
Well Generation
Guard Ring and Well/ Substrate Contact Insertion
Routing
Post-Routing Optimization
Net Shielding
Layout Result

Figure 2: A typical back-end design flow for AMS circuits.

post-refinement, we are able to generate wells close to manual designs. Our main contributions are summarized as follows:

- A GAN-guided well generation framework which incorporates designer expertise by leveraging previous quality-proven manually-crafted layouts is presented.
- To the best of our knowledge, this is the first work to apply deep learning techniques to guide the AMS layout well generation process.
- An effective post-refinement algorithm is also developed to satisfy design rules following the guidance of solutions generated by GAN.
- Experimental results show that the proposed technique is able to generate wells close to manual designs.

The rest of this paper is organized as follows: Section 2 introduces the preliminaries and the well generation problem. Section 3 explains the details of WellGAN. Section 4 shows the experimental results. And finally, Section 5 concludes the paper.

2 PRELIMINARIES

Fig. 2 shows a typical back-end design flow for AMS circuits. As an essential step in post-placement optimization, well generation takes the placement result as input to establish the bulk regions. In this work, by assuming manually-crafted layouts from experienced designers have guaranteed superior performance and robustness, our goal is to leverage a GAN model to generate wells as close to the manual designs as possible. The formal definition of the GAN-guided well generation problem is shown in Problem 1.

Problem 1 (GAN-Guided Well Generation). Given the device placement result, the objective of the GAN-guided well generation problem is to generate the well regions following the guidance of GAN to minimize the differences between the generated well regions and manual designs, with the constraints of correct electrical connections and clean design rules.

The well generation results shall satisfy two constraints, i.e., correct LVS for electrical connections and clean design rules. LVS correctness must ensure all the devices that should be covered by the wells are enclosed, and that the well regions should not overlap the devices which are not supposed to be covered. There are various design rules in well generation including the minimum spacing, enclosure, width, and area rules. The minimum spacing rules specify the minimum distance between the well and the objects outside of it. The minimum enclosure rules specify that the well edges should keep certain distance from the objects enclosed by the well. The minimum width rules require that the length of any edge and the distance between any two edges of a well should be no less than a certain value. Finally, the minimum area rules suggest the enclosing area of the well should be no less than a certain value.

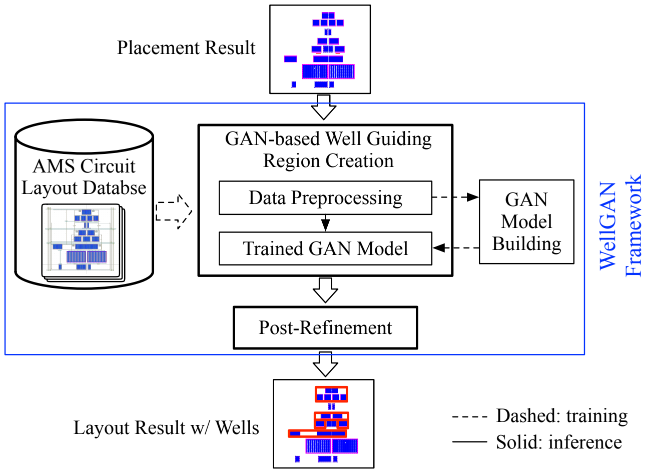


Figure 3: Proposed well generation framework (WellGAN).

Without loss of generality, the scope of this paper is the case where a single type of well is used for layout, e.g., N-well, and the wells share the same potential so that they can be merged without violating the electrical connections. Note that, our techniques are general and can be extended to handle double/triple well processes and different well potentials with minor modifications, which will be introduced in Section 3.2.

3 THE WELLGAN ALGORITHM

This section will introduce the overall flow, data representation and preprocessing, together with the detailed algorithms in WellGAN.

3.1 Overall Flow

As illustrated in Fig. 3, the WellGAN flow takes the placement result as input and produces the layout result with generated wells as output. The flow consists of two phases: training and inference. The training phase is shown with dashed arrows, where an AMS circuit layout database is utilized to build a conditional-GAN model for the inference. The inference phase is indicated with solid arrows, where the GAN model predicts the well region guidance for the input, and the post-refinement stage generates legal well regions. The entire flow requires the implementations of the following major tasks: (1) Preprocessing and extracting layout data for learning. (2) Designing and training a GAN model leveraging the layout database, which will guide the well generation at inference time to produce close-to-manual well generation results. (3) Performing a post-refinement step to legalize the resulting layout.

We have built a database of AMS circuit layouts for our GAN-guided well generation task, which will be described in Section 4 in detail. All of the layouts in our database satisfy the assumption of this work: (1) a single type of wells (N-well) is used, and (2) the wells share the same electric potential.

3.2 Data Representation and Preprocessing

Raw layouts in the database require special processing before they can be used as data samples in our learning model. The key techniques include customized data representation and preprocessing to abstract the data.

Data Representation. The data representation scheme has a tremendous impact on the GAN behavior. The key components that may impact our well generation are the locations of the devices both with N-wells (e.g., PMOS and certain types of capacitors, etc.) and without N-wells. The device placement result is composed of various layers, including polysilicon, diffusion, etc. We choose the oxide diffusion (OD) layers as the input patterns, as they are common among most devices in our layout database. Other irrelevant layers are omitted to reduce the effect of noise during training. To preserve the geometric and spatial relationship, we encode the layout patterns into red-green-blue (RGB) channels of images, which contain adequate information and are friendly for visualization. Note that as an extension to our framework, more channels can be added to handle more well types and different well potentials. The red channel covers all the OD patterns within the wells; the green channel covers all the OD patterns outside of the wells; and the blue channel encodes the wells. Thus, the inputs are images with patterns only in R and G channels, and the output images contain patterns in all RGB channels.

Data Preprocessing. To prepare the data samples, it is necessary to extract the OD layer patterns and differentiate the ones which are covered by wells and the others which are not. We perform geometric intersection and subtraction operations on the well layer and the OD layer geometries to obtain the two types of patterns, respectively. Note that we consider both the analog and customized digital parts in the AMS circuits, as we observe the GAN model generates high quality guidance for both. Furthermore, since the layout dimensions vary from design to design, we apply clipping, zero-padding, and scaling to transform the layouts into equally-sized image clips to facilitate the modeling. The clip size and scaling factor are determined in a way that there are sufficient layout patterns inside each clip and that the resolution (precision) loss caused by scaling is acceptable. Detailed settings will be explained in the experimental results section.

3.3 GAN-based Well Guidance Creation

Recently, GANs [10] have shown promising potential in many applications, including image processing, computer vision, and design for manufacturability, etc. [11, 12]. A conventional GAN architecture is diagrammed in Fig. 4. It simultaneously trains two models: a generative model G and a discriminative model D. The generator G tries to capture the distribution over the training data y to deceive D, while the discriminator D learns to distinguish between the "real" samples coming from the training data and the synthetic samples from G. In a conventional GAN, G is trained to map a random noise vector $z \sim p_z$ (which is the distribution of the random noise) to the data space, with the objective to maximize the probability that D classifies the samples from G as "real" data:

$$\mathbb{E}_{z\sim p_z}[\log D(G(z))]$$

In contrast, D is trained to maximize the probability of assigning correct labels to the samples:

$$\mathbb{E}_{y\sim p_d}[\log D(y)] + \mathbb{E}_{z\sim p_z}[\log(1 - D(G(z)))]$$

where p_d is the distribution of the dataset. However, for our well generation problem, we are given the placement results containing the layout patterns. Therefore, instead of mapping from the random

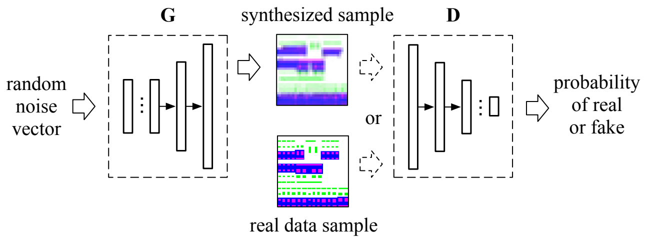


Figure 4: Architecture of a conventional GAN.

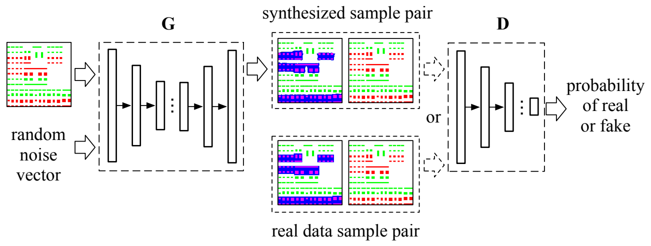


Figure 5: Architecture of a CGAN.

noise to the output, we are looking for a mapping from the input layout patterns to the layout with wells. The conventional GAN architecture is thus not directly applicable to our well generation task.

Conditional GAN (CGAN) [9, 11] is a type of GAN architectures that learns a conditional generative model as depicted in Fig. 5. Different from the conventional architecture, both the generator G and discriminator D see the additional input x. Apart from the random noise z, G also observes x, and it learns a mapping from both z and x to the output y. The loss function of the CGAN is in Equation 1.

$$\begin{aligned}\mathcal{L}_{CGAN}(G, D) = \ & \mathbb{E}_{x,y\sim p_d(x,y)}[\log D(x, y)] \\ & + \mathbb{E}_{x\sim p_d(x), z\sim p_z}[\log(1 - D(x, G(x, z)))] \\ & + \lambda_{L_1}\mathbb{E}_{x,y\sim p_d(x,y), z\sim p_z}[||y - G(x, z)||_1].\end{aligned} \tag{1}$$

And the objective of CGAN is as follows:

$$\min_G \max_D \mathcal{L}_{CGAN}(G, D)$$

The first two terms of $\mathcal{L}_{CGAN}$ are similar to the conventional GAN loss function where G tries to minimize the objective against D that tries to maximize it. The third term is the L_1−norm, which is used to encourage less blurring. Since our well generation problem is conditioned on the input layout patterns, the CGAN architecture is suitable.

Our generator and discriminator architecture designs are adapted from [11] and [13]. One characteristic of our well generation task is that our input and output share common structures which are aligned, e.g., the geometries in the placement result. In view of this, instead of using a conventional encoder-decoder [14] structure for our generator design (Fig. 6(a)), we embrace an encoder-decoder with skip connections that follows the U-Net structure [15] (Fig. 6(b)). This allows the low-level information to bypass the bottleneck layer in the encoder-decoder network and directly shuttle to the layers closer to the output.

The CGAN training process alternates between gradient descent steps on G and D, while during inference, only G is run to generate the output. In our implementation, we employ mini-batch stochastic gradient descent (SGD) and solve it with the Adam method [16].

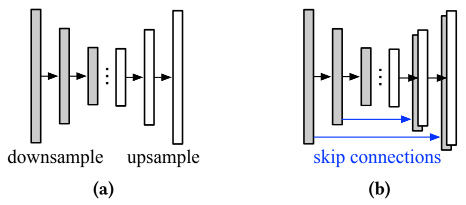


Figure 6: (a) Conventional encoder-decoder architecture, and (b) Encoder-decoder with U-Net-like skip connections.

In practice, previous work [11] has demonstrated that the noise vector is typically ignored by G. Hence, in our experiments, noise is introduced through dropout in the generator instead.

3.4 Post-Refinement

During the usage of our well generation framework, the placement result is preprocessed and input to the trained GAN model to produce the well guiding regions. We then perform post-refinement to legalize the wells based on these guiding regions, such that the results satisfy the design rules applicable to the well layer, including minimum spacing, enclosure, width, and area rules. In our post-refinement, we first merge the GAN-output images of all the clips into a single image. After that, we perform computational geometry operations to refine our results, and convert the obtained polygons back to the final well regions in the layout. The refinement process is discussed in the following.

3.4.1 Rectilinearization. The well guiding regions generated by the trained GAN model are essentially polygons. By extracting the image channel corresponding to the well layer and transforming it to a binary image through thresholding, we can directly apply the classical border following algorithm in image processing [17] to find the polygons (contours) defining the guiding regions. Since it is illegal to have arbitary shapes for wells, we need to transform our results to rectilinear polygons.

Our rectilinearization algorithm is described in Alg. 1. First, we assign either horizontal or vertical directions to the polygon edges based on their slope (lines 14-16). If the absolute value of the slope of an edge is less than 1, it is categorized as closer to the horizontal direction. Otherwise, the edge is assigned to the vertical direction. We then iteratively merge the neighboring edges with the same assigned direction, such that the sequence of edges alternates between horizontal and vertical directions. Subsequently, the merged edges are rectilinearized. The locations of the rectilinearized edges are determined according to the average coordinates of the points on the curve along the specific direction (lines 17-21). Afterwards, the resultant rectilinearized edges are connected to form the rectilinear polygon. As an example, Fig. 7(b) is the rectilinearization result for the well guiding regions shown in Fig. 7(a).

3.4.2 Legalization. Since the rectilinearized polygons defining the well regions may suffer from design rules violations, it is imperative to legalize them. For each rectilinearized well region, the devices outside of it expanded by minimum spacing are first subtracted from it. Then, it is unioned with the devices inside it expanded by minimum enclosure.

To address the minimum width design rule violations, we propose the algorithm presented in Alg. 2. The algorithm works by

Algorithm 1 Rectilinearization.

```
Input:  Polygon $p$
Output: Rectilinearized polygon $p_r$
1:  $E \leftarrow$ vector of edges of $p$, $E_{new} \leftarrow \emptyset$
2:  for all $e \in E$ do
3:      if AssignEdgeDirection($e$) == $E_{new}.last.dir$ then
4:          merge $e$ with $E_{new}.last$
5:      else
6:          $E_{new} \leftarrow E_{new} \cup e$
7:      end if
8:  end for
9:  merge $E_{new}.first$ with $E_{new}.last$ if same direction
10: for all $e \in E_{new}$ do
11:     $e.loc \leftarrow$ getLocation($e$)
12: end for
13: $p_r \leftarrow$ polygon formed by $E_{new}$

14: function AssignEdgeDirection($e$)
15:     return $|\text{slope}(e)| \geq 1$
16: end function

17: function getLocation($e$)
18:     $A \leftarrow$ area under the curve formed by points of $e$
19:     $d \leftarrow$ distance b/w the first and last points of $e$
20:     return $A/d$
21: end function
```

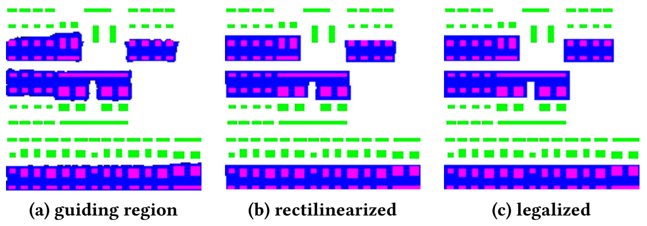
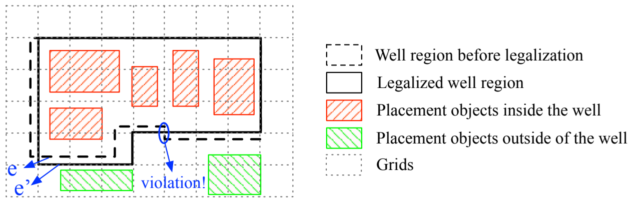


Figure 7: Example post-refinement results after each step.

mapping and aligning the polygon edges to a grid whose grid size equals to the specified minimum width, as illustrated in Fig. 8. Each polygon builds its own grid. The algorithm will make decisions on which grid each edge should be aligned to. It will first try to align the edge to the nearest grid, if the location does not incur the minimum enclosure nor spacing rule violations. Otherwise, it will try to align the edge to the other side which would cause no violation. If that is again infeasible, the algorithm will choose to align to the grid which satisfies the minimum enclosure rules, and leave the minimum spacing violations to be fixed at a later stage. For example, edge e in Fig. 8 is aligned to location e' to satisfy the enclosure requirement. Empirically, since the minimum width is small compared to the size of the wells and the devices, we are able to align to the grids without violating the spacing and enclosure rules for all the layouts in our test set.

Alg. 2 can guarantee to resolve all the minimum width violations after aligning to grids since no edge will have length less than the grid size which is the specified minimum width, neither will the edge distances. It will not cause minimum enclosure rule violations,



e
e'
violation!
Well region before legalization
Legalized well region
Placement objects inside the well
Placement objects outside of the well
Grids

Figure 8: Grid-based method for the minimum width rule.

Algorithm 2 Legalization.

```
Input: Polygon $p$, minimum width $W_{min}$
Output: Legalized polygon $p_l$
1: build grids with size $W_{min}$
2: for all edge $e \in p$ do
3:     if nearest grid is feasible then
4:         align $e$ to the nearest grid
5:     else if second nearest grid is feasible then
6:         align $e$ to the second nearest grid
7:     else
8:         align $e$ to the grid satisfying min. enclosure rules
9:     end if
10: end for
```

either. Besides, the minimum area rule can be easily checked by calculating the area of each well region. Since the minimum enclosure rule ensures that the wells containing any device will be larger than the required minimum area, the wells violating the minimum area rule shall not contain any device, thereby they can be simply removed to fix the violations.

As discussed previously, although we completely eliminate the minimum width, enclosure, and area rule violations, there might still exist minimum spacing rule violations. To settle the spacing violations, a linear programming formulation can be employed based on the constraint graphs to spread the devices and wells, with the objective of displacement minimization, which is widely adopted for placement legalization [18]. Details are omitted here to conserve space. The example of a legalization result is shown in Fig. 7(c).

4 EXPERIMENTAL RESULTS

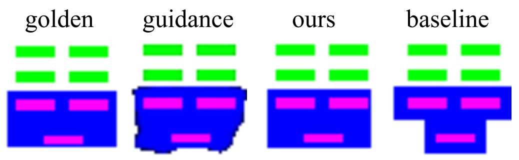
All of our algorithms are implemented in Python and C/C++, and the experiments are performed on a Linux server with 3.3GHz Intel i9 CPU, 128GB memory, and one Nvidia Titan Xp GPU. The generative adversarial net is developed based on `TensorFlow` [19] library. To build our database for well generation, we gather 131 distinguished silicon-measurement-proven AMS circuit layouts (in GDSII format) from the taped-out chips in TSMC 40nm process technology. The layouts are split into a total of 881 clips, with each layout clip transformed into a 256-by-256 RGB image (as in Section 3.2). Random selection of 80% clips are used for training and the remaining 20% form the test set. Note that all the clips belonging to the same circuit shall be allocated to the same set, so that during inference we can recover the entire layout for the test circuit solely from the test set. We implement the algorithm in [6] as the baseline for comparison. It performs union on the devices inside the well expanded by certain distance. In our experiments, it is set to the minimum enclosure distance.

The training is run for 100 epochs with a mini-batch size of 1. After that, we run the inference flow on the test dataset with the trained model. We compare WellGAN with the baseline algorithm on the same dataset. Both algorithms are very efficient, i.e., within a few seconds for a typical layout size. The visualization of the well generation results of several test circuit examples are shown in Fig. 9, where "golden" means the manual layouts by experienced designers, "guidance" refers to the well guiding regions output by our GAN model, "ours" means the final well generation results from WellGAN after post-refinement, and "baseline" refers to the results of the baseline algorithm. Fig. 9(a) shows the results for circuit A, and Fig. 9(b) are for circuit B, etc. Recall that blue channel encodes the wells and the other channels are the input patterns. From the test results, we can see that WellGAN is able to successfully mimic the designers' behavior from the manual layout data.

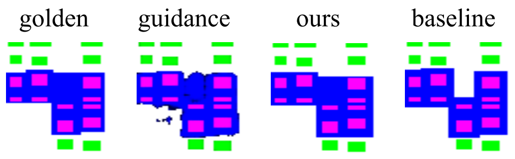
To quantify the similarity between the results, the metric we use is the Manhattan norm of the element-wise (pixel-by-pixel) difference between two images on the blue channel which represents the well regions. If the two images have different sizes due to spacing rule settling, the smaller image is center-aligned with the larger one and is zero-padded for comparison. The norm is calculated as the summation of the absolute values of the difference for all pixels divided by the image size. The distribution of the Manhattan norm of the element-wise difference over a total of 43 test circuits is plotted in Fig. 10. The x-axis is the range of the Manhattan norm of the element-wise difference, and the y-axis represents the frequency that the norm of difference occurs in the test results. For example, 19% of the test results for WellGAN are within a difference of 0 to 2% from the manual layouts, whereas only 5% of the baseline results fall in this range. From the figure, we can see that the difference of the majority of our results are within a small scale, while the baseline algorithm may result in larger differences (up to 44% as shown in the histogram). Our element-wise difference distribution demonstrates a smaller mean and standard deviation than the baseline, as shown in Table 1. Moreover, our results are free from minimum spacing, enclosure, width, and area design rule violations.

To evaluate the routing complexity, we count the number of wells generated by both WellGAN and the baseline for all test circuits. A larger number of wells means more well contacts to be connected by routing, thereby potentially increasing the routing complexity. Our results show that on average, the number of wells generated by the baseline is 75% more than WellGAN. This demonstrates that our well generation results are expected to have better routability.

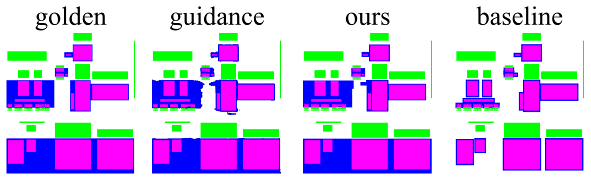
Furthermore, we compare the post-layout circuit performance simulation results between the layout with our generated wells and the manual one for the op-amp circuit (see Fig. 1), as shown in Table 2. The placement is the same for both layouts, while routing is slightly different due to different well contact locations. The WellGAN-generated layout circuit performance is comparable to the manual design. Nonetheless, the well generation result from the baseline requires dramatic change to both placement and routing to accommodate the increased number of well contacts, thus the comparison is less meaningful.



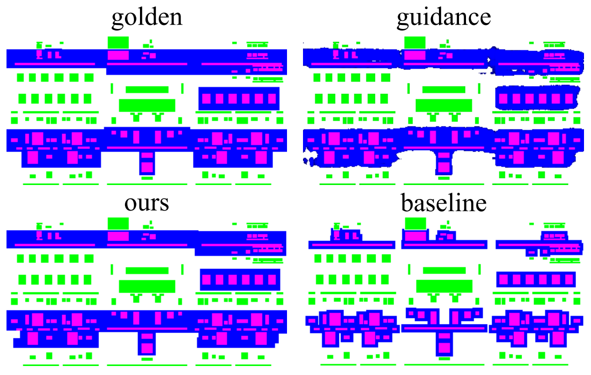
(a) Circuit A



(b) Circuit B



(c) Circuit C



(d) Circuit D

Figure 9: Test circuit examples A, B, C, and D.

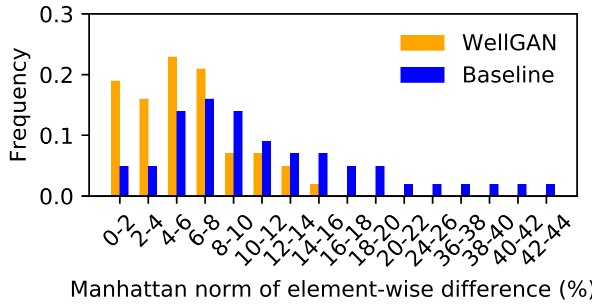


Figure 10: Distribution of the Manhattan norm of the element-wise difference.

5 CONCLUSION

Well generation is a fundamental challenge in back-end AMS design flow. This work attempts to generate wells by mimicking experienced designers' behavior from high-quality manually-crafted layouts. A holistic well generation framework, *WellGAN*, is proposed, with GAN-guided well geometry creation and post-refinement. Experimental results demonstrate the proposed framework is able to generate wells close to manual designs with comparable circuit performance.

Table 1: Statistics of the Manhattan norm of element-wise difference for the test results.

Metric	WellGAN	Baseline
Mean	5.64%	12.65%
Standard Deviation	3.58	10.25

Table 2: Post-layout simulation results of the op-amp layout with wells generated by WellGAN.

Design	Unity Gain Bandwidth (MHz)	Phase Margin (deg.)	Loop Gain (dB)
Manual	103.70	71.89	36.33
Ours	104.64	65.38	37.61

ACKNOWLEDGMENT

This work is supported in part by the NSF under Grant No. 1527320, No. 1704758, and the DARPA ERI IDEA program.

REFERENCES

[1] H.-C. Ou, K.-H. Tseng, J.-Y. Liu, I. Wu, Y.-W. Chang *et al.*, "Layout-dependent-effects-aware analytical analog placement," in *Proc. DAC*, 2015, p. 189.
[2] R. A. Hastings and R. A. Hastings, *The art of analog layout.* Pearson Prentice Hall New Jersey, 2006, vol. 2.
[3] M. M. Ozdal and R. F. Hentschke, "Algorithms for maze routing with exact matching constraints," *IEEE TCAD*, vol. 33, no. 1, pp. 101–112, 2014.
[4] M. Lin, P.-H. Chang, S.-Y. Lee, and H. Graeb, "Demixgen: Deterministic mixed-signal layout generation with separated analog and digital signal paths," *IEEE TCAD*, vol. 35, no. 8, 2016.
[5] B. Xu, S. Li, X. Xu, N. Sun, and D. Z. Pan, "Hierarchical and analytical placement techniques for high-performance analog circuits," in *Proc. ISPD*, 2017, pp. 55–62.
[6] 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.
[7] 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, pp. 305–311.
[8] I. Goodfellow, Y. Bengio, A. Courville, and Y. Bengio, *Deep learning.* MIT press Cambridge, 2016, vol. 1.
[9] M. Mirza and S. Osindero, "Conditional generative adversarial nets," *arXiv preprint arXiv:1411.1784*, 2014.
[10] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative adversarial nets," in *Advances in neural information processing systems (NIPS)*, 2014, pp. 2672–2680.
[11] P. Isola, J.-Y. Zhu, T. Zhou, and A. A. Efros, "Image-to-image translation with conditional adversarial networks," in *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR).* IEEE, 2017, pp. 5967–5976.
[12] H. Yang, S. Li, Y. Ma, B. Yu, and E. F. Young, "GAN-OPC: Mask optimization with lithography-guided generative adversarial nets," in *Proc. DAC*, 2018, pp. 131:1–131:6.
[13] A. Radford, L. Metz, and S. Chintala, "Unsupervised representation learning with deep convolutional generative adversarial networks," *arXiv preprint arXiv:1511.06434*, 2015.
[14] G. E. Hinton and R. R. Salakhutdinov, "Reducing the dimensionality of data with neural networks," *science*, vol. 313, no. 5786, pp. 504–507, 2006.
[15] O. Ronneberger, P. Fischer, and T. Brox, "U-net: Convolutional networks for biomedical image segmentation," in *International Conference on Medical image computing and computer-assisted intervention.* Springer, 2015, pp. 234–241.
[16] D. P. Kingma and J. L. Ba, "Adam: A method for stochastic optimization," in *Proc. 3rd Int. Conf. Learn. Representations (ICLR)*, 2014.
[17] S. Suzuki *et al.*, "Topological structural analysis of digitized binary images by border following," *Computer vision, graphics, and image processing*, vol. 30, no. 1, pp. 32–46, 1985.
[18] J. Cong and M. Xie, "A robust mixed-size legalization and detailed placement algorithm," *IEEE TCAD*, vol. 27, no. 8, pp. 1349–1362, 2008.
[19] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean *et al.*, "Tensorflow: a system for large-scale machine learning." in *OSDI*, vol. 16, 2016, pp. 265–283.