Spring 2022

Custom Cell Placement Automation for Asynchronous VLSI

Yihang Yang
Yale University Graduate School of Arts and Sciences, yihangyang1993@gmail.com

Abstract

Custom Cell Placement Automation for Asynchronous VLSI

Yihang Yang

2022

Asynchronous Very-Large-Scale-Integration (VLSI) integrated circuits have demonstrated many advantages over their synchronous counterparts, including low power consumption, elastic pipelining, robustness against manufacturing and temperature variations, etc. However, the lack of dedicated electronic design automation (EDA) tools, especially physical layout automation tools, largely limits the adoption of asynchronous circuits.

Existing commercial placement tools are optimized for synchronous circuits, and require a standard cell library provided by semiconductor foundries to complete the physical design. The physical layouts of cells in this library have the same height to simplify the placement problem and the power distribution network. Although the standard cell methodology also works for asynchronous designs, the performance is inferior compared with counterparts designed using the full-custom design methodology.

To tackle this challenge, we propose a *gridded cell* layout methodology for asynchronous circuits, in which the cell height and cell width can be any integer multiple of two grid values. The gridded cell approach combines the shape regularity of standard cells with the size flexibility of full-custom layouts. Therefore, this approach can achieve a better space utilization ratio and lower wire length for asynchronous designs. Experiments have shown that the gridded cell placement approach reduces area without impacting the routability. We have also used this placer to tape out a chip in a 65nm process technology, demonstrating that our placer generates design-rule clean results.

Custom Cell Placement Automation for Asynchronous VLSI

A Dissertation
Presented to the Faculty of the Graduate School
of
Yale University
in Candidacy for the Degree of
Doctor of Philosophy

by
Yihang Yang

Dissertation Director: Rajit Manohar

May, 2022

Contents

List of Figures

List of Tables

Acknowledgments

I would like to express my deepest appreciation to my supervisor, Prof. Rajit Manohar, for accepting me into his research group and providing unwavering guidance and support during my Ph.D. studies.

I would also like to extend my deepest gratitude to Prof. Leandros Tassiulas and Prof. Jakub Szefer as members of my Ph.D. committee for insightful feedback on this work.

I am also grateful to my teammates Wenmian Hua, Rui Li, Ruslan Dashkin, Prafull Purohit, and Xiayuan Wen for their tremendous help and contribution. I very much appreciate Samira Ataei, Edward Bingham, Ioannis Karageorgos, Congyang Li, Zhan Liu, Tayyar Rzayev, and Xiang Wu for helpful discussions and valuable advice.

I also had the great pleasure of working with Jiayuan He, Yi-Shan Lu, and Sepideh Maleki on many challenging yet rewarding projects.

Thanks to my friends at Yale for their encouragement throughout my Ph.D. studies.

I'm extremely grateful to my family members and my love for their unconditional love and support.

Glossary

ASIC	: application-specific integrated circuit
CHP	: communicating hardware process
DI	: delay-insensitive
EDA	: electronic design automation
FPGA	: field-programmable gate array
GND	: ground (low voltage in circuits)
HPWL	: half-perimeter wire length
I/O	: input/output
QDI	: quasi-delay-insensitive
RTL	: register-transfer level
Vdd	: voltage drain drain (high voltage in circuits)
VLSI	: very-large-scale-integration

Chapter 1

Introduction

Application-specific integrated circuits (ASICs) are of great significance to modern society [1]. As a consequence of the steady miniaturization of silicon transistors from 20 micrometers in the 1970s to a few nanometers in the 2020s, Very-Large-Scale-Integration (VLSI) makes it possible to create an ASIC containing millions to billions of transistors on a single silicon chip [2–4].

Due to the relative simplicity of design, optimization, and verification, nearly all integrated circuits are synchronous circuits with a global clock signal controlling the data orchestration [4, 5]. As the scale of integrated circuits increases, it is impractical for chip designers to manage the growing complexity without automation tools [6–8], which stimulates the research on automation algorithms for synchronous circuits. With the steady development and improvement for several decades [9, 10], commercial electronic design automation (EDA) tools are very mature to facilitate the circuit design process [11].

Despite the astonishing achievement of synchronous circuits, the global clock signal itself has become a limiting factor since the beginning of the 2000s for two major reasons. First, the clock distribution network can consume more than 40% of the overall power [12–15]. Second, it is challenging to distribute a clean clock signal across the whole chip region for modern designs [16–18]. Clock gating technique [12, 19–21] and multiple clock domain technique [22, 23] are proposed to mitigate these two problems, respectively.

Asynchronous circuits, also known as clockless or self-timed circuits, use the handshaking protocol among circuit components to replace the global clock signal, which by

nature are clock-gated and have many local clock domains [24–27]. Despite the appearance of asynchronous designs in the 1950s, the high design complexity and area overhead hamper the wide adoption of asynchronous circuits in the industry, which, in turn, leads to the lack of dedicated automation tools from EDA companies and layout support from semiconductor foundries. Researchers have to manually design the physical layout for high-performance asynchronous designs over the past decades [28–30]. Globally-asynchronous locally-synchronous circuits circumvent this problem by exploiting existing EDA tools to create synchronous modules and using asynchronous channels for inter-module communications [31, 32].

This work aims to identify the placement problem for asynchronous circuits and develop novel and efficient algorithms to handle large-scale designs. Since there are no existing placement tools specialized for asynchronous circuits, this chapter will introduce the standard cell placement flow developed for synchronous circuits.

1.1 Background

1.1.1 Circuit Design Flow

Commonly adopted design methods for ASICs include the *full-custom* design methodology and the *standard cell* design methodology [33]. These two approaches balance the trade-off between circuit performance and design time differently [34]. The full-custom design methodology requires designers to carefully handcraft the circuit, which gives designers the freedom to optimize each circuit component for superior power, performance, and area. However, because this approach demands a huge amount of engineer hours for large designs, it is only used for high-volume and high-performance circuits to amortize the design cost [35].

The standard cell design methodology requires a standard cell library to construct the circuit. This library consists of pre-designed and pre-characterized basic logic gates and storage elements like AND-gate, OR-gate, NOT-gate, latches, and flip-flops, but applies restrictions on their physical layouts [36]. This library, together with its layout restrictions, largely simplifies the design problem and thus enables automation tools to facilitate the design process [37].

Modern ASIC design is a complicated task. To better handle the design complexity, the design process is divided into many steps at different levels of abstraction [38]. Take the standard cell methodology as an example, these steps are listed below.

- *System specification*: this step determines the functionality, performance, power, chip size, technology node, and other high-level requirements.
- *Architectural design*: a basic architecture is specified to meet those system-level requirements. This architecture defines the interface between the environment and the system.
- *Functional and logic design*: function modules and their interconnections are implemented for this architecture. These modules are usually described at the register-transfer level (RTL) using hardware description languages.
- *Circuit design*: logic synthesis tools translate an RTL description to a circuit diagram with a given standard cell library. This circuit diagram is also called a netlist, consisting of cells, macros, and their interconnections. Macros are pre-designed and pre-characterized function modules, which are optimized for power, performance, and area. Typical macros include memory banks, digital signal processing units, and other function modules.
- *Physical design*: this step takes the physical geometry of cells and macros into consideration. During this process, placement tools map cells and macros onto a two-dimensional grid, and then routing tools map their interconnections onto a three-dimension grid. A verification step is needed to ensure the final layout meets all high-level requirements.
- *Fabrication*: the physical layout is sent to semiconductor foundries, and then converted into a series of photolithography masks. These masks are used to create patterns forming transistors and metal wires on silicon wafers.
- Packaging and testing: silicon wafers are cut into individual chips, which are then packed and tested to ensure they meet all requirements.

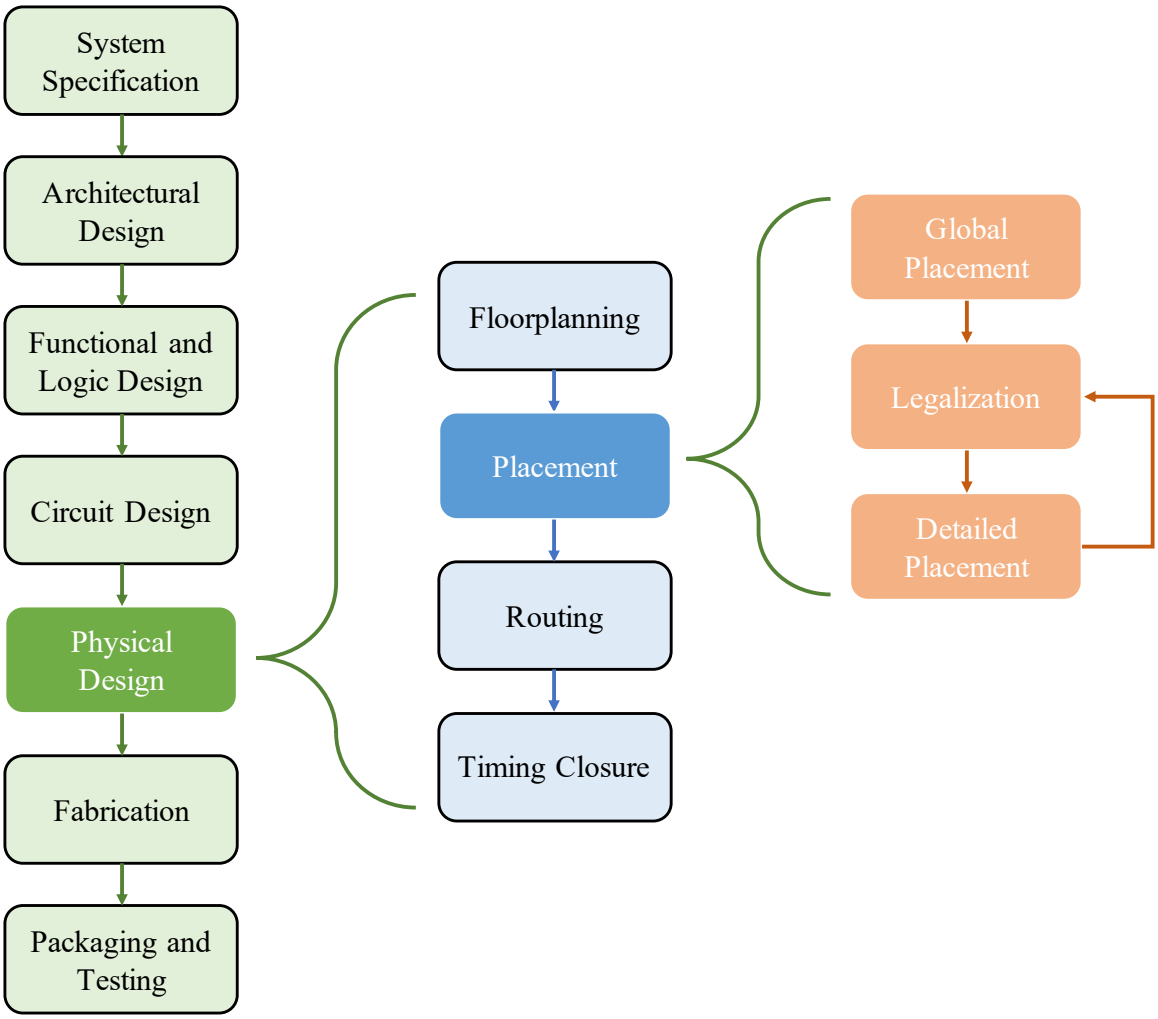


Figure 1.1: A typical digital circuit design flow. This work focuses on the placement problem in the physical design stage.

1.1.2 Physical Design Flow

As a step in the circuit design flow, the physical design process consists of four major components: floorplanning, placement, routing, and timing closure [39].

- *Floorplanning*: this step seeks to determine the location of circuit input/output (I/O) ports and macros. For standard cell designs, this step also specifies the power distribution network, the location of rows, and the location of auxiliary cells, like well tap cells and end cap cells. The power distribution network and rows put a requirement on the location of cells.

- *Placement*: the task of this step is to assign an exact location for every cell. These

locations must respect physical design rules and other layout constraints. A typical example is the overlap constraint: no cells can overlap with other cells. Besides, the placer also needs to improve circuit performance via optimizing the critical path delay, total wirelength, routability, die area, power consumption, heat dissipation, and other crucial metrics.

- *Routing*: this step aims to create metal wires for cell interconnections using limited routing resources. These metal segments must also respect physical design rules. For example, metal wire spacing must be no less than the minimum spacing. A routing tool also needs to optimize critical path delay and other metrics to ensure correctness and improve performance.

- *Timing closure*: designers specify two kinds of timing constraints for digital circuits. The first type is the correctness constraint, and violations of these constraints can make a circuit malfunction. Another type is the performance constraint, which aims to ensure circuit performance. Timing analysis tools can identify violations of timing constraints after the extraction of resistance and capacitance from a physical layout. This step is usually performed iteratively during placement and routing such that the final result can meet all timing constraints. For synchronous circuits, timing violations can also be mitigated and fixed via a dedicated clock signal distribution and placement refinement step.

1.2 Placement Problem Formulation

In general, the placement problem can be formulated as follows:

- **Input**: netlist, placement region, I/O pins;
- **Output**: cell locations;
- **Constraint**: cell overlap, design rule, timing constraint, cell density;
- **Objective**: wirelength, critical path delay, power, routability, etc.

It has been shown that for non-trivial objective functions, finding the optimal placement is known to be an NP-hard problem [40].

The input netlist consists of cells, macros, and nets: a cell or a macro usually has a rectangular shape, the location of which can be either pre-specified or unknown; a net specifies the interconnection among cell pins, macro pins, and I/O pins. The placement region defines the range of cell locations, and I/O pins need to be placed on the periphery of the placement region.

The output of the placement problem is the location of cells and macros. The solution space is discrete due to the manufacturing process and pre-placed rows. Moreover, the placement result must respect all physical design rules and satisfy all timing constraints; otherwise, the circuit may not work as expected.

The placement result also needs to satisfy various constraints. The cell overlap constraint requires that cells cannot overlap with each other. This constraint (or even the existence of cell boundary) is essentially a heuristic to ensure no design rule violations when cells are placed close to each other. Although cell overlap may not necessarily lead to design rule violations, the placer has to make the worst-case assumption if the detailed cell layout is abstracted away during placement. Besides, the placement result is required to respect other design rules, like N/P-well design rules. The cell locations also need to satisfy timing constraints to ensure the design can work as expected. The cell density constraint requires that heat generated by cells can dissipate easily to keep the silicon chip working under a safe temperature.

Crucial circuit metrics need to be optimized during placement. For example, the critical path delay determines the circuit performance; total wirelength influences power and performance. However, it is often timing-consuming to evaluate the exact metrics during placement, and thus placement tools choose to calculate and optimize surrogate functions instead of metrics themselves. Take the wirelength as an example, there are many wirelength estimation models, and half-perimeter wirelength (HPWL) is the most popular method due to its low computational cost [38].

Wirelength-driven placement have been extensive studied in the past decades. The

general form of the objective function is the following:

$$obj(X, Y) = W(X, Y) + D(X, Y), \tag{1.1}$$

where $X = \{x_1, x_2, ..., x_n\}$ and $Y = \{y_1, y_2, ..., y_n\}$ are the x and y coordinates of cells, respectively, W is the wirelength cost, and D is the overlap or density penalty. In the next, we will briefly introduce the standard cell methodology and its placement flow for optimizing this objective function.

1.3 Standard Cell Methodology

The standard cell methodology requires a standard cell library provided by semiconductor foundries to construct circuits. This library contains many simple logic functions and storage elements, such as inverters, electrical buffers, NAND/AND gates, NOR/OR gates, half/full adders, latches, and flip-flops. An important feature of the physical layout of standard cells is that they all have a Vdd rail at the top boundary and a GND rail at the bottom boundary, and they share the same height to simplify the physical design problem [4].

1.3.1 Standard Cell Layout Style

Wirelength-driven placement tools usually abstract away the functionality and detailed layout of cells and macros to simplify the placement problem. Fig. 1.2 (a) shows the physical layout of the inverter implemented in the standard cell methodology. When the input signal *in* is high, the pull-down network connects the output signal *out* to the low voltage GND. When the input signal *in* is low, the pull-up network connects the output signal *out* to the high voltage Vdd. The pull-down network sits in a P-well, while the pull-up network sits in an N-well. As can be easily seen, this physical layout implements the Boolean function $out = \neg in$ using the following production rules:

$$\begin{aligned} in &\rightarrow out\downarrow \\ \neg in &\rightarrow out\uparrow . \end{aligned} \tag{1.2}$$

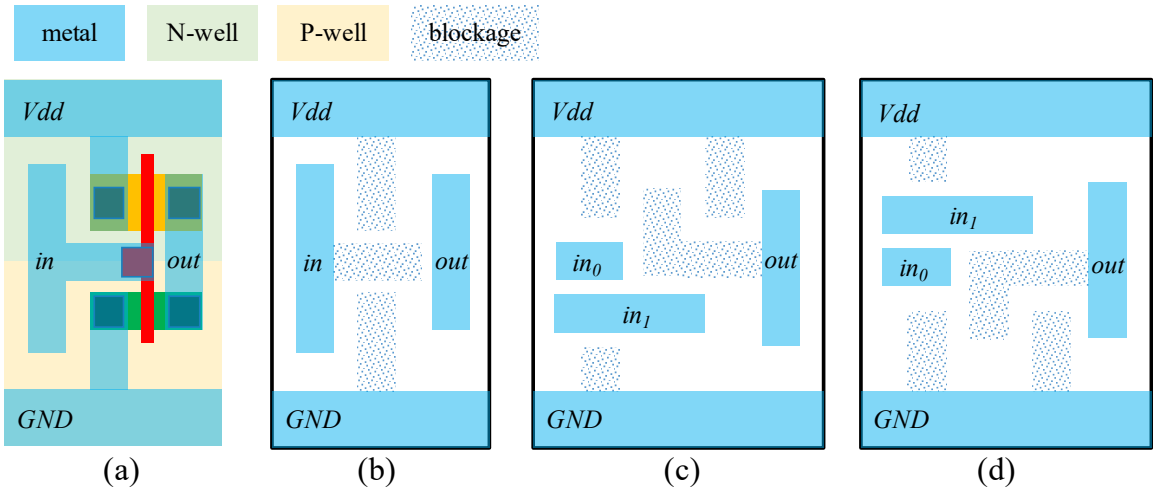


Figure 1.2: Standard cell layout style. (a) INV gate layout, (b) INV gate during placement, (c) NAND gate during placement, (d), NOR-gate during placement.

Fig. 1.2 (b) shows the abstraction of the INV gate during placement. For the ease of signal routing, some parts of input/output/power metal segments are marked as routing blockages. Similarly, Fig. 1.2 (c) and (d) display the abstraction of the NAND gate and NOR gate, respectively. As can be seen, these cell layouts share the same height and have Vdd rail at the top and GND rail at the bottom. Something not shown in Fig. 1.2 is that N/P-wells in these cells also have the same height. Since N-wells need to accommodate all pull-up networks, the largest pull-up network in the cell library determines the height of N-wells in all cells. A similar relationship holds for the height of P-wells.

This style of cell layout together with the row-based placement technique can simplify the power distribution network, and fully abstract N/P-well design rules away from the placement problem.

1.3.2 Row-based Placement

Fig. 1.3 shows an example of the row-based placement for a standard cell design. In this example, cells are placed into rows: cells in the same row share the same orientation and vertical location; cells in an adjacent row are mirrored vertically to avoid a short circuit and share a common power rail. The advantages of row-based placement are listed below.

- *Power distribution.* Power rails of cells in the same row are perfectly aligned, and

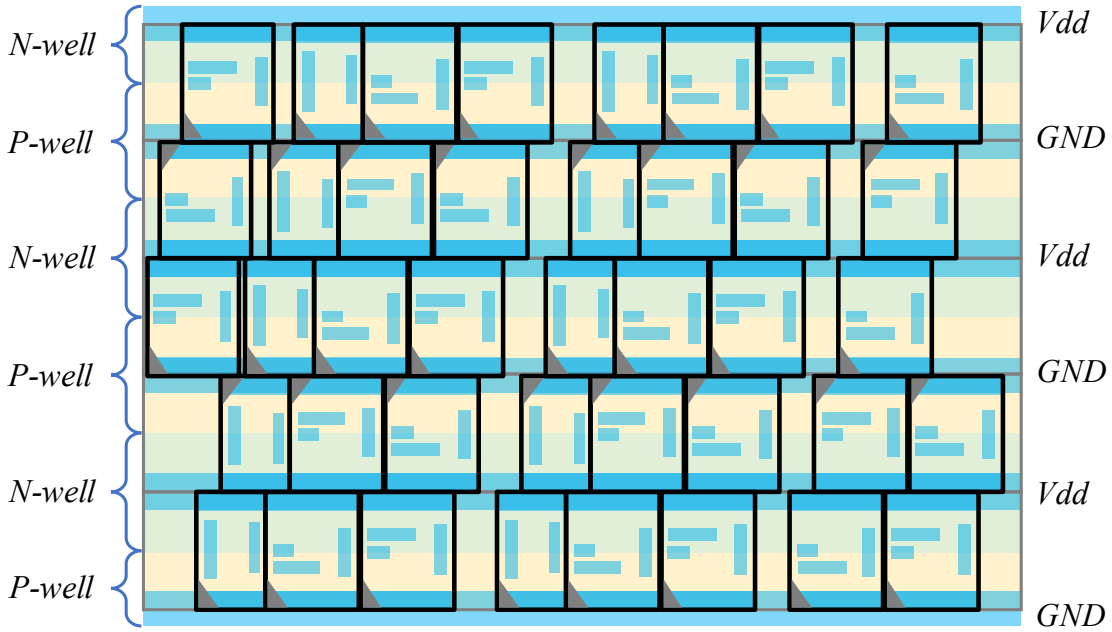


Figure 1.3: Row-based placement for standard cells. Gray triangles indicate the orientation of standard cells.

adjacent rows with opposite orientations allow them to share a common power rail. These rows simplify the power distribution network to be regular horizontal metal wires strapped together with vertical metal wires.

- *N/P-well design rules.* Design rules related to N/P-wells need to be satisfied during placement. For example, the width and height of a well must be no less than a minimum length, and the area of a well must be no less than a minimum area. There are more delicate design rules, especially in modern technology nodes. N/P-wells of cells in the same row are automatically abutted and united into a large well, which can satisfy almost all N/P-well design rules, even delicate ones. Therefore, row-based placement can abstract complicated N/P-well design rules from the standard cell placement problem.

- *Cell location correlation.* Rows divide the whole placement region into many sub-regions. Cells in the same row are forbidden to overlap with each other, and thus their horizontal locations are strongly correlated. Cells in different rows can have the same horizontal location because they are guaranteed to be overlap-free. The presence

of rows splits the global overlap constraint into local overlap constraints.

1.4 Standard Cell Placement Flow

A placement automation tool, also known as a placer, determines the exact physical location of all cells in the placement region [41]. For modern technology processes, this location assignment largely influences power and performance. However, an exact evaluation of the circuit performance is a computationally expensive task. To mitigate this problem, chip performance is estimated using simple metrics, which also act as objective functions during placement. These metrics include HPWL, routability, critical path delay, power, area, and other metrics [7]. Placers also need to consider layout constraints and timing constraints. These constraints are modeled as smooth functions that take large or infinite values if unsatisfied, which turns the placement problem into an optimization problem.

Modern circuits can have millions of logic gates, which puts a strong requirement on the time complexity of placement algorithms. A common way to manage the complexity is to divide the placement process into several stages: global placement, legalization, and detailed placement [7, 8].

- *Global placement* relaxes the cell overlap constraint and other constraints for generating a coarse-grained placement solution. The goal of this step is to quickly find a high-quality yet illegal solution. Since the overlap constraint is relaxed, this solution usually contains many cell overlaps.

- *Legalization* cleans up design rule violations in the coarse placement and generates a legal placement by moving cells locally. Because the global placement result is expected to be close to a high-quality solution, a legalizer should perturb this result as little as possible.

- *Detailed placement* aims to explore the solution space around the legalized placement result. A detailed placer takes the legal placement as its input and invokes local refinement algorithms to improve this result iteratively.

1.4.1 Global Placement

Problem Formulation

During this process, HPWL is usually used to estimate the computationally expensive wirelength, and thus the objective function in Eqn. 1.1 becomes

$$obj(X,Y) = HPWL(X,Y) + D(X,Y). \tag{1.3}$$

Since the netlist consists of cells and interconnections, we can denote it as (C, E), where C is the set of cells, and E is the set of nets. Moreover, each net e consists of a collection of cell pins: if the cell i has a pin in net e, we denote it as $i \in e$. With this, we can write HPWL as

$$\begin{aligned} HPWL(X,Y) &= \sum_{e\in E}[(\max_{i\in e} x_i - \min_{i\in e} x_i) + (\max_{i\in e} y_i - \min_{i\in e} y_i)] \\ &= \sum_{e\in E}(\max_{i\in e} x_i - \min_{i\in e} x_i) + \sum_{e\in E}(\max_{i\in e} y_i - \min_{i\in e} y_i)] \\ &= HPWL(X) + HPWL(Y), \end{aligned} \tag{1.4}$$

which is the total half perimeter of the minimum bounding box of each net. Note that in the above formula, the shape and location of cell pins are ignored.

Eqn. 1.4 seems to imply that the x component and the y component of HPWL are independent. Reality is not what it seems: due to the existence of the placement region and placement blockages, the solution space is usually irregular, and thus a simple relationship, like $X \in \mathbb{R}^{dim(X)}$ or $X \in [low, high]^{dim(X)}$, is inaccurate because the assignment of the variable X determines the solution space of the variable Y, and vice versa. Therefore, HPWL(X) and HPWL(Y) are entangled tightly and implicitly. They can be treated as independent functions only when the placement region is a rectangle and placement blockages are absent or ignored.

Common strategies for global placement include the partitioning approach, simulated annealing approach, and analytical approach.

Partitioning Approach

This approach determines cell locations in a top-down fashion by recursively cutting the netlist and placement region into sub-netlists and placement bins. The intuition behind this approach is that the minimum cut should be able to reduce the number of nets connecting sub-netlists. Since these nets usually span placement bins, they are more likely to have large bounding boxes. Reducing these nets should be able to make more nets physically local and thus improve wirelength.

Commonly adopted heuristics for netlist partitioning are the Kernighan–Lin (KL) algorithm [42] and the Fiduccia-Mattheyses (FM) algorithm [43]. Although the time complexity of the FM algorithm is linear to the number of pins, the quality of partitioning degrades with the increase of the netlist size. A multilevel paradigm is proposed to mitigate this problem [44, 45]. This paradigm starts with coarsening the original hypergraph by clustering strongly connected vertices. This process can repeat several times until the hypergraph is small enough. Then the partitioning heuristic is applied to the coarsened hypergraph level by level until the original hypergraph is restored. Typical partitioning-based placers include PROUND [46], GORDIAN [47], FengShui [48], and Capo [49].

Simulated Annealing Approach

For optimizing a discrete objective function, simulated annealing is capable of escaping from local minima and thus can lead to a placement result with superior quality. The TimberWolf placement and routing package has demonstrated the effectiveness of this technique [50–52]. First, the placer starts with a random placement as the initial state. Then, a new state is generated by exchanging two units in a range limiter, moving a cell to another location, or changing the orientation of cells, pads, and macros. Next, the placer evaluates the cost function of the new state, which consists of wirelength, cell overlap penalty, and row overcrowding penalty. Last, the new state is accepted at random with probability of $\min(1, \exp(-\Delta C/T))$, where ΔC is the cost variation, and T is the decreasing annealing temperature. These steps are in a loop with a stopping criterion that there is no further improvement on the cost function.

TimberWolf can generate superior placement results for circuits with thousands of cells. Since the state space increases quickly with the circuit size, a clustering technique is introduced to improve the time complexity for large designs [53, 54].

Analytical Approach

Analytical placement techniques have been extensively studied over the past decades due to their excellent scalability [55–57]. The core idea is to use a smooth approximation of HPWL as the surrogate function during placement.

Quadratic wirelength. Quadratic placers use the quadratic wirelength (QW) to approximate HPWL. For a net e, the x component of its HPWL is

$$\begin{aligned} HPWL &= \max_{i\in e} x_i - \min_{i\in e} x_i \\ &= \max_{i,j\in e} |x_i - x_j| \\ &= \sum_{i,j\in e} w_{ij}|x_i - x_j| \\ &= \sum_{i,j\in e} \frac{w_{ij}}{|x_i - x_j|}(x_i - x_j)^2 \\ &= \sum_{i,j\in e} \frac{w_{ij}}{\Delta x_{ij}}(x_i - x_j)^2 \\ &\approx \sum_{i,j\in e} c_{ij}(x_i - x_j)^2 = QW, \end{aligned} \tag{1.5}$$

where w_{ij} is 1 only when the i-th cell is the maximum pin and the j-th cell is the minimum pin; otherwise, w_{ij} is 0. w_{ij} in this example is chosen in this way to illustrate the idea of the quadratic approximation.

There are many other ways to make the above equation hold. A popular method is the Bound2Bound net model [58]: if we denote the number of cell pins in net e as $|e|$, when one of i and j is the maximum/minimum pin, we have

$$c_{ij} = \frac{1}{(|e|-1)}\frac{1}{|x_i - x_j|}, \tag{1.6}$$

otherwise, c_{ij} is 0.

The global minimum of the quadratic wirelength is reached when its first derivative is zero, which can be obtained by solving the following linear system

$$\sum_{e \in E} \sum_{i,j \in e} c_{ij}(x_i - x_j) = 0. \tag{1.7}$$

Note that c_{ij} is a function of x_i and x_j, and it needs to be updated when the new set of locations is obtained. This also means that the above linear system should be solved iteratively until all cell locations converge. Typical quadratic placers include FastPlace [59], Kraftwerk [58], SimPL [60], MAPLE [61], and POLAR [62].

Nonlinear wirelength. Log-sum-exponential wirelength (LSEW) is another way to approximate HPWL:

$$\begin{aligned} HPWL &= \max_{i \in e} x_i - \min_{i \in e} x_i \\ &= \lim_{\gamma \to 0} \left[\ln \sum_{i \in e} e^{x_i/\gamma} + \ln \sum_{i \in e} e^{-x_i/\gamma} \right] \\ &\approx \ln \sum_{i \in e} e^{x_i/\gamma} + \ln \sum_{i \in e} e^{-x_i/\gamma} = LSEW, \end{aligned} \tag{1.8}$$

where γ is a parameter controlling the accuracy and smoothness of the approximation. APlace [63] and NTUPlace [64] adopt LSEW for global placement.

Weighted-average wirelength (WA) is a popular approximation in recent years:

$$\begin{aligned} HPWL &= \max_{i \in e} x_i - \min_{i \in e} x_i \\ &= \lim_{\gamma \to 0} \left[\frac{\sum_{i \in e} x_i e^{x_i/\gamma}}{\sum_{i \in e} e^{x_i/\gamma}} + \frac{\sum_{i \in e} -x_i e^{-x_i/\gamma}}{\sum_{i \in e} e^{-x_i/\gamma}} \right] \\ &\approx \frac{\sum_{i \in e} x_i e^{x_i/\gamma}}{\sum_{i \in e} e^{x_i/\gamma}} + \frac{\sum_{i \in e} -x_i e^{-x_i/\gamma}}{\sum_{i \in e} e^{-x_i/\gamma}} = WA. \end{aligned} \tag{1.9}$$

ePlace [65], RePlAce [66], and DREAMPlace [67] use Nesterov's method to optimize this nonlinear cost function.

Density constraint. Because optimizing HPWL usually leads to a lot of cell overlaps, a smoothed wirelength function itself is inadequate for generating a high-quality global placement. Therefore, analytical placers need various heuristics to remove density hotspots.

For example, quadratic placers often use anchors and look-ahead legalization to keep cells away from dense regions [60, 62]. Some placers choose a smooth overlap function to spread out cells [64]. Other placers model cells as an electrical system and use electrostatics to distribute cells in the placement region [67, 68].

1.4.2 Legalization

The primary goal of the legalization step is to eliminate cell overlaps and ensure each cell has a legal location inside the placement region. Because the global placement result is expected to be of high quality, it is undesirable to perturb cell locations during this step, and thus a typical cost function during this process is the following:

$$obj(X, Y) = (\|X - X^0\|^p + \|Y - Y^0\|^p) + D(X, Y), \tag{1.10}$$

where p is 1 or 2, (X^0, Y^0) is the global placement result, and (X, Y) is the placement after legalization. The first term in the above objective measures the location variation after legalization, and the second term is the penalty function for cell overlaps.

Tetris is a simple but effective legalization algorithm for standard cells [69]. This heuristic implicitly optimizes cell displacement by partially preserving the cell order during legalization. The simplest version of this legalization method is to snap cells to rows and compactly pack cells toward the left boundary. First, cells are sorted according to the x-coordinate of the lower-left corner in the ascending order. Then, for each cell, the legalizer finds the row whose leftmost free location is closest to the current cell location. Next, the legalizer moves this cell to the greedy optimal position and updates the free space of the corresponding row. These steps repeat until every cell has a legal location.

Fig. 1.4 (a) shows a global placement result: some cells overlap with other cells, and some cells are out of the placement boundary. Fig. 1.4 (b) shows the legalized placement result using the above Tetris legalization algorithm: cells are closely packed toward the left placement boundary; the order of cells is preserved in every single row. This technique can be easily extended to support cells with various heights.

Abacus is the state-of-the-art standard cell legalizer, which can simultaneously opti-

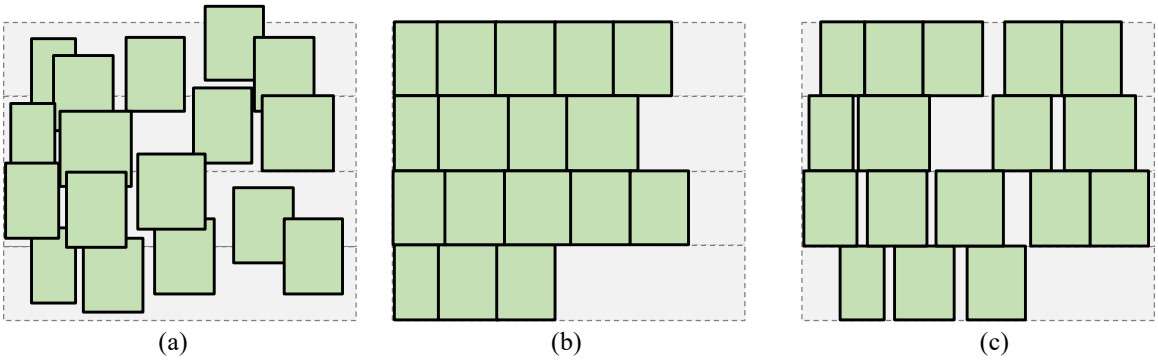


Figure 1.4: Example of different legalization techniques: (a) global placement result, (b) legalized result using the simplest Tetris legalization algorithm, (c) legalized result using Abacus legalization algorithm.

mize cell displacement and eliminate cell overlaps by treating the legalization problem as a quadratic programming problem. The cost function is the quadratic displacement, and the constraints are the overlap constraint and the boundary constraint. Abacus can find the optimal legal placement solution for a fixed cell order.

For a given list of ordered cells $C = \{c_1, ..., c_n\}$ that need to be placed into a specific row, the cost function is

$$cost(X) = \sum_{i \in C} (x_i - x_i^0)^2, \tag{1.11}$$

where x_i^0 is the initial location of the i-th cell. Because Y and Y^0 are fixed in this case, they are excluded from the above cost function. The solution needs to satisfy the following constraints:

$$\begin{aligned} lower\_bound &\leq x_1, \\ x_1 + w_1 &\leq x_2, \\ &... \\ x_{n-1} + w_{n-1} &\leq x_n, \\ x_n + w_n &\leq upper\_bound, \end{aligned} \tag{1.12}$$

where w_i is the width of cell c_i, *lower_bound* and *upper_bound* specify the free space in this row. As can be seen, this is a typical quadratic programming problem. Instead of calling a quadratic programming solver to find the solution, Abacus uses a more efficient way to solve this problem based on the observation that cells are usually closely packed into segments

in the legalized placement. This procedure is an important subroutine in Abacus called *PlaceRow*.

Similar to the Tetris legalization algorithm, the Abacus legalization algorithm starts with sorting cells according to the x-location. Then, for each cell, the legalizer uses the *PlaceRow* subroutine to find the row with the optimal cost, and places this cell into that row. These steps repeat until all cells are processed. Fig. 1.4 (c) shows the legalized placement result using the Abacus legalization algorithm: cells are closer to their initial location compared with the Tetris legalization result. As can be seen, Abacus is also a greedy algorithm and explicitly relies on the existence of rows. However, it is difficult to extend this algorithm to support cells with various heights.

1.4.3 Detailed Placement

The detailed placement algorithms are usually more computationally expensive than global placement algorithms. Since the global placer, together with the legalizer, gives a reasonably good placement result, the task of the detailed placer is to refine this result and keep the density/overlap penalty from increasing.

FastPlace-DP is a greedy detailed placement algorithm which can generate a high-quality solution quickly [70]. This detailed placement improves wirelength using the following four techniques:

- *global swap*: for a given cell, it is easy to find the optimal location to minimize HPWL if all other cells remain unmoved. In reality, this optimal location turns out to be a rectangular region. If this optimal region can accommodate this cell without influencing other cells, we can improve HPWL by simply moving this cell into its optimal region. However, this operation usually leads to cell overlap or cell density overflow. A safer way is to swap this cell with a candidate cell in the optimal region, which is also why this step is called global swap.

- *vertical swap*: under certain circumstances, it is impossible to successfully perform a global swap operation for a given cell. Notice that moving a cell closer to its optimal region can also improve HPWL. The vertical swap technique tries to swap a cell with

another candidate cell in adjacent rows but closer to its optimal region.

- *local reordering*: the vertical swap step adjusts the cell location vertically to reduce HPWL, while the local reordering changes the cell location/order horizontally to improve wirelength. For every row, we can use a sliding window containing 3 or 4 cells to traverse cells in this row. It is fast to enumerate all possible ordering of cells in this window and choose the optimal order to improve HPWL.

- *single-segment clustering*: this step aims to find the optimal placement for cells in a row without changing the cell order. Assuming cells in other rows are fixed, cells in this row can slide along the row to improve wirelength. Although an exact algorithm exists for finding the optimal solution [71], FastPlace-DP uses a fast heuristic to improve HPWL.

These four techniques are applied iteratively until HPWL converges.

There are many other detailed placement techniques: NRG uses simulated annealing to improve HPWL [72]; Domino decomposes the detailed placement problem to be a network flow problem [73]; some placers uses mixed integer programming for detailed placement [74].

1.5 Contribution

The contributions made by this dissertation are listed below.

- Due to the adoption of custom logic gates in asynchronous circuits, standard cell methodology usually leads to physical layouts with larger areas, more power consumption, and additional timing constraints. To mitigate these problems, a gridded cell methodology is proposed for asynchronous designs in Chapter 2. This new methodology combines the shape regularity of standard cells with the size flexibility of full-custom layouts. Therefore, this approach is capable of generating a more compact layout with lower power consumption and better performance.

- The gridded cell methodology brings new challenges to the placement process: N/P-well design rules and power distribution network need to be explicitly handled during

placement. It is shown in Section 2.4.4 that a cluster-based placement technique can solve these new problems.

- A gridded cell placement flow, Dali, is implemented to automate the cluster-based placement process. Experiment results in Chapter 3 show that the gridded cell placement approach can reduce the area by 15% and improve the wirelength.

- Complex custom logic gates are often used in asynchronous designs. Chapter 4 introduces efficient and effective algorithms to handle these complex gates during legalization.

- The timing-driven placement flow for asynchronous circuits is introduced in Chapter 5 as future works.

Chapter 2

Physical Design of Asynchronous Circuit

This chapter will start with an introduction to asynchronous circuits and commonly adopted physical design methodologies. Then, we will identify the placement dilemma for asynchronous designs and propose the gridded cell layout style to solve this problem.

2.1 Asynchronous Circuit

Asynchronous circuits have demonstrated many advantages over their synchronous counterparts [25, 26, 75–77]. These advantages are listed below.

- *Low power consumption*: asynchronous circuits use distributed handshake protocol instead of a global clock signal to control data processing, and thus they are active on-demand, which makes them more energy-efficient.
- *No clock distribution*: it is challenging to distribute a clean clock signal across the whole die area for synchronous designs, but there is no global clock signal in asynchronous designs.
- *Average-case performance*: the delay of asynchronous circuits can be data-dependent, while that of synchronous circuits is always the worst-case delay.

- *Robustness*: since signal delays are influenced by many external factors, such as voltage or temperature fluctuation and manufacturing variation, asynchronous circuits can be insensitive to signal delays, which makes them robust against these noises.

- *Design modularity*: the clock frequency of a synchronous pipelined design may need to be reduced when new stages are added, while asynchronous circuits treat stages as plug-ins because of the handshake protocol and local timing constraints.

2.2 Circuit Family

There are many asynchronous circuit families, each of which has a different set of delay assumptions [77]. Among these circuit families, quasi-delay-insensitive (QDI) [78] circuits and micropipelines [79, 80] are very popular due to their high operation speed and relative simplicity of physical implementation.

2.2.1 Quasi-Delay-Insensitive Circuit

Delay-insensitive (DI) circuit makes no assumption on the delay of wires and gates, making it a very robust circuit family [81]. However, the functionality of this circuit family is limited when all logic gates have a single output [78]. It has been shown that two-output gates are sufficient to remove this limitation, where timing constraints are hidden behind these gates [82].

A quasi-delay-insensitive circuit is a DI circuit with the isochronic fork assumption [78], which is the minimal set of timing assumptions that makes this circuit family Turing-complete [83]. This isochronic fork timing constraint can be easily satisfied during the physical design stage [28, 30].

QDI circuits use the dual-rail encoding for datapath, where two wires represent one bit of data [84, 85], as shown in Fig. 2.1 (a). Take the d_0 bit as an example, this bit has a true rail $d_0.t$ and a false rail $d_0.f$. When both rails are 0, there is no data present in d_0. The logic value "1" is encoded as $d_0.t = 1$ and $d_0.f = 0$, while the logic value "0" is encoded as $d_0.t = 0$ and $d_0.f = 1$. It is forbidden that both rails are high at the same time.

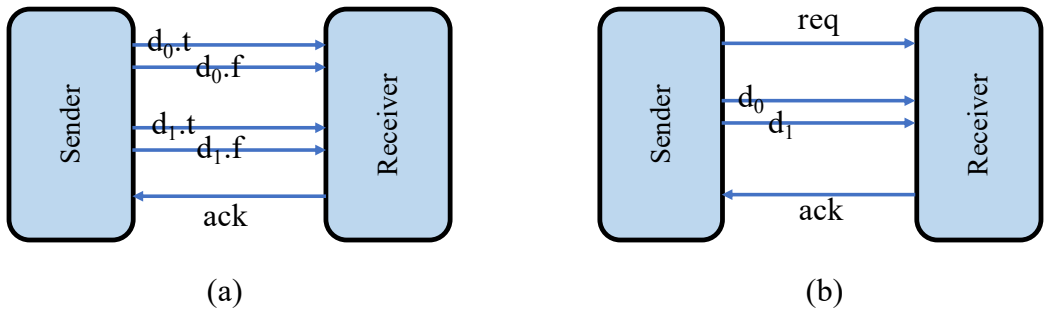


Figure 2.1: Asynchronous data encoding: (a) dual-rail encoding, (b) bundled data encoding.

The four-phase handshake protocol for the dual-rail encoding works in this way: initially, all signals are low; when the sender initiates the communication, it sets the true or false rail to high for both data bits; when the receiver senses the presence of data, it will set the *ack* (acknowledge) signal to high; then the sender reset all bit rails to low; finally, the receiver reset the *ack* signal to low, and the system restores its initial state and ready for the next round of communication. Although the dual-rail encoding makes the area of most logic gates twice as big as their single-rail counterparts [86,87], it makes cell communication very robust to cell placement and signal routing. The correctness of a QDI circuit layout is largely guaranteed by its delay-insensitivity; timing-driven placement and routing are only for performance improvement.

There are many techniques for synthesizing a QDI circuit. A classic method is communicating hardware process (CHP) decomposition [88], which has been used for building a famous asynchronous MIPS microprocessor [28]. The syntax-directed translation approach is another popular method: Tangram [89] developed by Philips has been used for designing many asynchronous circuits [90]. Other methods include data-driven decomposition [91], concurrent dataflow decomposition [92], and high-level synthesis [93].

2.2.2 Micropipeline

Pipelined asynchronous designs have many advantages over their synchronous counterparts, including design modularity, pipeline elasticity, intrinsic data flow control, and on-demand power consumption [80, 94]. However, asynchronous logic functions tend to have a larger area because of the dual-rail encoding, and suffer from the lack of dedicated support from

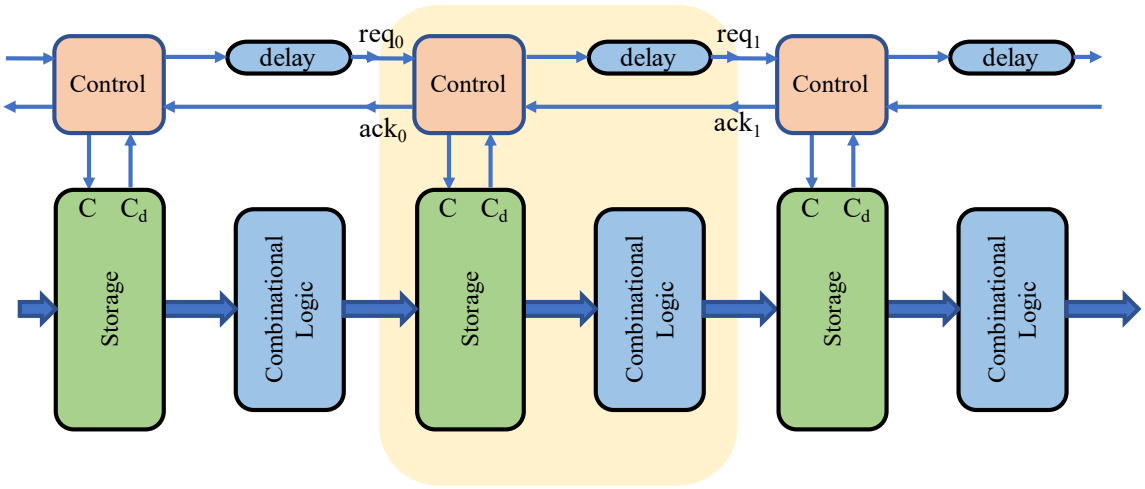


Figure 2.2: A linear micropipeline structure.

EDA companies and semiconductor foundries.

Micropipeline, introduced by Ivan Sutherland during his Turing award lecture [79], combines the distributed control from asynchronous designs and the simplicity of synchronous circuits [95]. Since then, this design style has been extensively studied and adopted [80].

Fig. 2.2 shows a linear micropipeline structure with three stages. Each stage consists of three components: control circuitry, storage unit, and logic function.

- *Control*: the control circuitry follows the handshake protocol and generates a local clock signal. When the input data to this stage is ready, the request signal req_0 from the previous stage becomes high. And if this stage is ready to accept new data, the control part will generate a local clock signal, instructing the storage unit to capture the input data. Once the input data is cached, the acknowledge signal ack_0 becomes high to notify its previous stage to take new data and its logic function to process the cached data. When the output data is ready, the request signal req_1 becomes high, and once the next stage completes data capturing, the acknowledge signal ack_1 will become high. This control circuitry essentially acts as a function wrapper and provides a communication interface.

- *Storage*: the storage unit caches the data from the previous stage so that the previous stage can reset itself and then start working on new data. There are many ways to implement this storage unit as long as the implementation is compatible with the

control circuit. For example, the latch is a popular design choice due to its area and power efficiency. When the control signal C is high, the latch becomes transparent to pass the input data. When the data capturing process is complete, the signal C_d becomes high, and the latch becomes opaque to protect the data from being overwritten by accident.

- *Logic function*: the bundled data encoding and combinational logic are selected to implement the logic function and simplify the datapath. Fig. 2.1 (b) shows the bundled data encoding, where each data wire represents one logic bit. The request signal *req* indicates the validity of all data bits, and this is also the bundled data timing constraint, which needs to be satisfied during physical implementation. Since it takes time for the combinational logic to process the input data, a delay line is needed to show the completion of the computation. Principally speaking, the function can be implemented using an analog circuit as long as the timing constraint is satisfied.

There are many different kinds of micropipelines [80]. Mousetrap pipeline uses standard cells to construct its control part and the storage part, which makes it possible to exploit mature EDA tools to complete the physical implementation [96]. GasP pipeline uses a single-track wire channel, dynamic logic, and custom gates to build the control circuit, which leads to very low latency and high operation speed but requires manual layout [95]. Precharge half-buffer pipeline and weak-condition half-buffer pipeline use QDI control circuits, which provides timing robustness [97].

2.3 Physical Design

Because asynchronous circuits contain many complex custom logic gates, mature EDA tools have difficulties in handling them: standard cell libraries provided by semiconductor foundries do not have these custom cells [28, 89, 95, 97]; static timing analysis does not automatically support asynchronous designs [98]; the placement tool requires that custom cells must be designed in the standard cell fashion [99]. The routing process is the only step that supports custom cells but with a preference over cell pin shapes and locations [100].

A	B	C
0	0	0
0	1	C_{prev}
1	0	C_{prev}
1	1	1

$$A\&B \rightarrow C\uparrow$$
$$\neg A\&\neg B \rightarrow C\downarrow$$

Figure 2.3: C-element truth table and production rules.

In the following, we use the Muller C-element to illustrate the difference among different physical implementation strategies. C-element is a widely used state-holding gate in many asynchronous circuit families [24]: the output becomes high when all inputs are high; the output becomes low when all inputs are low; otherwise, the output stays unchanged. The next state logic function of a two-input Muller C-element is

$$C' = AB + AC + BC. \tag{2.1}$$

The truth table and production rules of C-element is shown in Fig. 2.3, and Fig. 2.4 (a) shows the schematic of its semi-static implementation.

2.3.1 Full-Custom Design

Fig. 2.4 (b) shows the full-custom layout of the two-input C-element: when input A and B are different, the output C remains unchanged due to the weak feedback inverter; when input A and B are the same, the pull-up/pull-down network has a stronger drive strength than the weak inverter to switch the value of C. The sizing of transistors is crucial to ensure the correctness of this implementation, which also means that timing constraints are inside this gate. And once the functionality and correctness are verified, we can safely use this implementation in the physical layout.

The arrangement of these full-custom cells needs to satisfy N/P-well design rules and other design constraints. After the placement of these cells, designers need to manually create a power delivery network and connect power pins in each cell to the corresponding power supplies. The amount of effort to design such a network depends on the regularity of cell placement. Although the full-custom design methodology can lead to superior per-

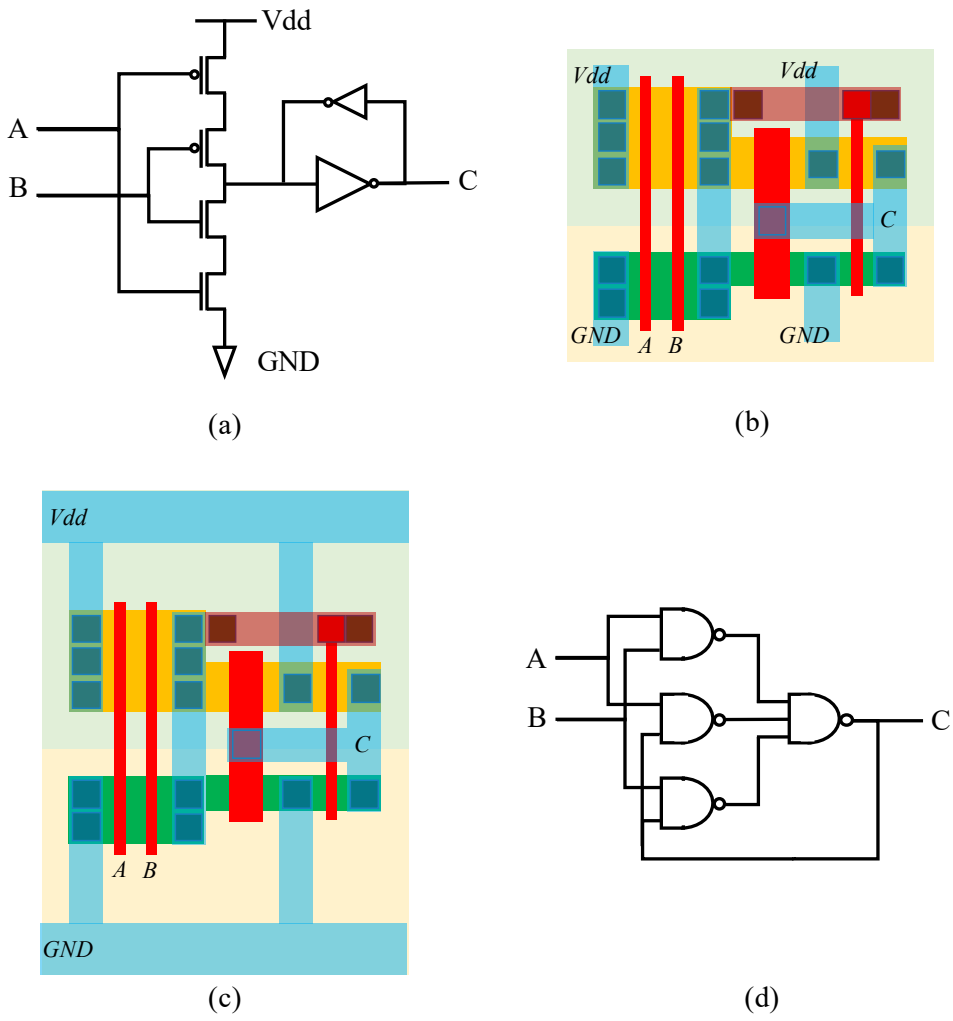


Figure 2.4: Muller C-element and its implementations. (a) semi-static implementation, (b) full-custom layout, (c) custom standard cell layout, (d) static standard-cell implementation.

formance, power, and area, this approach is very labor-intensive, and thus only adopted for high-volume or state-of-the-art asynchronous circuits [28–30, 35].

2.3.2 Custom Standard Cell Design

This approach aims to leverage existing standard cell placement tools by designing the physical layout of custom logic gates in the standard cell fashion. Fig. 2.4 (c) shows the custom standard cell layout of the two-input C-element. Compared with the full-custom layout, this physical implementation has local power rails acting as a cell wrapper to simplify the construction of the power distribution network. After the placement of cells, this local

network can automatically form a global power distribution network, eliminating the need for explicit power network construction.

This approach requires that every cell must have the same height, whose minimum value is the sum of the maximum N-well height and the maximum P-well height among all cells. Since asynchronous circuits contain custom logic gates with very different pull-up and pull-down networks, this approach often leads to the waste of space in simple gates. This is also why the custom standard cell layout in Fig. 2.4 (c) is taller than its full-custom layout in Fig. 2.4 (b). Existing works have confirmed that this approach has a large area overhead due to the high variance of custom cell heights [99].

2.3.3 Standard Cell Design

This approach seeks to design an asynchronous circuit using standard cells provided by semiconductor foundries. Since nearly all asynchronous designs contain custom logic gates, these logic gates need to be reimplemented using standard cells. However, this technology mapping process usually leads to a larger area, more power consumption, and even extra timing constraints.

Fig. 2.4 (d) shows the static standard cell implementation of the two-input C-element. This implementation uses three 2-input NAND gates and one 3-input NAND gate, which takes more space than the full-custom implementation. The extra timing constraint is that the feedback loop must stabilize before the arrival of the next input, which depends on the external environment. This also means the placement and routing tools need to handle this timing constraint.

2.4 Placement Problem for Asynchronous Circuits

Although pre-designed power rails simplify the power distribution network and abstract away N/P-well design rules, it limits the flexibility of the physical layout for an asynchronous circuit. To obtain a high-quality physical implementation, we have to discard the standard cell methodology. This choice impels us to use the full-custom cell layout during the placement and routing, and brings two new problems: N/P-well design rules and power

distribution network.

2.4.1 Gridded Cell Layout Style

A full-custom layout can be very flexible, for example, the shape of a cell is unnecessarily a rectangle, and the placement boundary of a custom cell can be ambiguous. These high degrees of freedom make the placement problem intractable, and we need some constraints on the cell layout to make this problem manageable. Therefore, we propose a new cell layout style, called **gridded cell layout**, and we term custom logic cells implemented in this style as gridded cells. This new layout style is the full-custom layout with constraints listed below.

- *Placement boundary*: the shape of a cell is a well-defined rectangle, and the width and height of this rectangle are an integer multiple of a predefined grid value g_x and g_y, respectively.
- *Well boundary*: the N/P-well height is an integer multiple of the vertical grid value plus a constant, i.e., $ng_y + c$, where n is a non-negative integer, and c is a constant for all cells.
- *Placement grid*: the lower-left corner of a cell must be placed on a placement grid, whose horizontal/vertical grid width is also g_x/g_y.
- *Cell abutment*: when two cells are abutted horizontally with the same orientation, there is no design rule violations from this abutment if their N/P-wells are aligned; when two cells are abutted vertically with opposite orientations, there is no design rule violations from this abutment.

Fig. 2.5 (a) shows three cells designed in the gridded cell layout style. As can be seen, the above constraints make every corner of a cell on the placement grid. *cell*1 and *cell*2 are abutted horizontally with no design rule violations; *cell*1 and *cell*3 are abutted vertically with no design rule violations.

Modern VLSI routing tools usually adopt grid-based algorithms, which prefer cell pins on a three-dimensional routing grid to simplify the routing problem [101]. To comply with

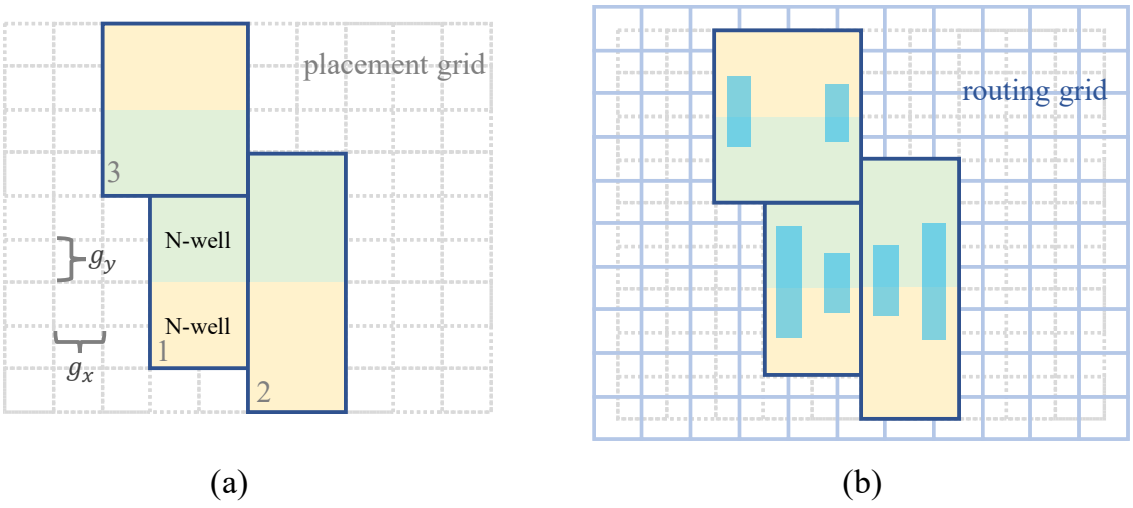


Figure 2.5: Gridded cell layout style. (a) example gridded cells and the placement grid, (b) example cell pins and the routing grid.

this preference, some optional constraints are listed below.

- *Routing grid*: the routing grid shares the same grid values with the placement grid.
- *Cell pin*: cell pins are on the routing grid when cells are on the placement grid, which means the shape and location of cell pins need to compensate the offset between the placement grid and the routing grid.

Fig. 2.5 (b) shows an example of cell pins satisfying these optional constraints.

2.4.2 Placement Problem Formulation

The gridded cell placement problem is formulated as the following:

- **Input**: netlist, placement region, I/O pins
- **Output**: cell locations
- **Constraint**: cell overlap, timing constraints, cell density, N/P-well design rules
- **Objective**: wirelength, critical path delay, power, routability, etc.

This placement problem can also be modeled as an optimization problem, the objective function is the following:

$$obj(X,Y) = W(X,Y) + D(X,Y) + Well(X,Y), \tag{2.2}$$

where $W(X,Y)$ is the wirelength cost, $D(X,Y)$ is the density and overlap penalty, and $Well(X,Y)$ is the penalty for N/P-well design rule violations. The previous chapter gives the formulas of the wirelength cost and the density penalty function. Next, we will analyze N/P-well design rules to figure out its mathematical expression.

2.4.3 N/P-well Design Rules

For gridded cells, we need to handle the following N/P-well design rules during placement:

- *min-width rule*: the distance of two non-intersecting edges in an N/P-well polygon must be no less than a pre-defined value;
- *min-space rule*: the distance between an edge of an N/P-well polygon and an edge of another N/P-well polygon must be no less than a pre-defined value;
- *min-area rule*: the area of an N/P-well polygon must be no less than a pre-defined value;
- *latch-up prevention rule*: the distance between a point in an N/P-well to the nearest well tap cell in the same well must be no more than a pre-defined value.

These rules are complicated to satisfy at the first glance, especially when there are a large number of cells.

We can start with only two gridded cells to learn the consequence of these design rules. The width of the first cell is w_1, and its N/P-well height is n_1/p_1; similarly for the second cell. For the sake of simplicity, we assume the minimum width of N-well and P-well are the same, and we denote it as m_w; similarly for the minimum space m_s. Since the relative arrangement of cells determines the N/P-well penalty, we can safely assume the first cell is fixed with the center of its N/P-well borderline being the origin, and the second cell can freely move with the center of its N/P-well borderline being (x, y).

There are many ways to define penalty functions for N/P-well design rule violations and cell overlaps. For illustrative purposes, we will choose penalty functions in the following way: the penalty function takes the value 0 if there are no design rule violations or cell overlaps and 1 otherwise.

Two Cells with the Same Orientation

Fig. 2.6 (a) shows two gridded cells with the same orientation. For these two cells, the overlap cost function can be written as

$$overlap(x, y) = b(x, -w, w)b(y, -h_l, h_u), \tag{2.3}$$

where

$$\begin{aligned} w &= (w_1 + w_2)/2, \\ h_l &= (p_1 + n_2), \\ h_u &= (n_1 + p_2), \end{aligned} \tag{2.4}$$

and $b(x, a, b)$ is the boxcar function with value 1 in range $[a, b]$ and 0 otherwise.

If we only consider the min-width and the min-space rules, the N/P-well design rule penalty function can be written as

$$well(x, y) = b(x, x_l, x_u)\left[b(y, y_{l1}, y_{l2}) + b(y, y_{u1}, y_{u2})\right], \tag{2.5}$$

where

$$\begin{aligned} x_l &= -(w + m_s), \\ x_u &= w + m_s, \\ y_{l1} &= \min(-(p_1 + m_s), -(n_2 + m_s)), \\ y_{l2} &= \max(-(p_1 - m_w), -(n_2 - m_w)), \\ y_{u1} &= \min(n_1 - m_w, p_2 - m_w), \\ y_{u2} &= \max(n_1 + m_s, p_2 + m_s). \end{aligned} \tag{2.6}$$

Figure (b) shows the total penalty function after smoothing: the placement is legal if the second cell is in the blue region; the second cell violates the N/P-well design rules or the

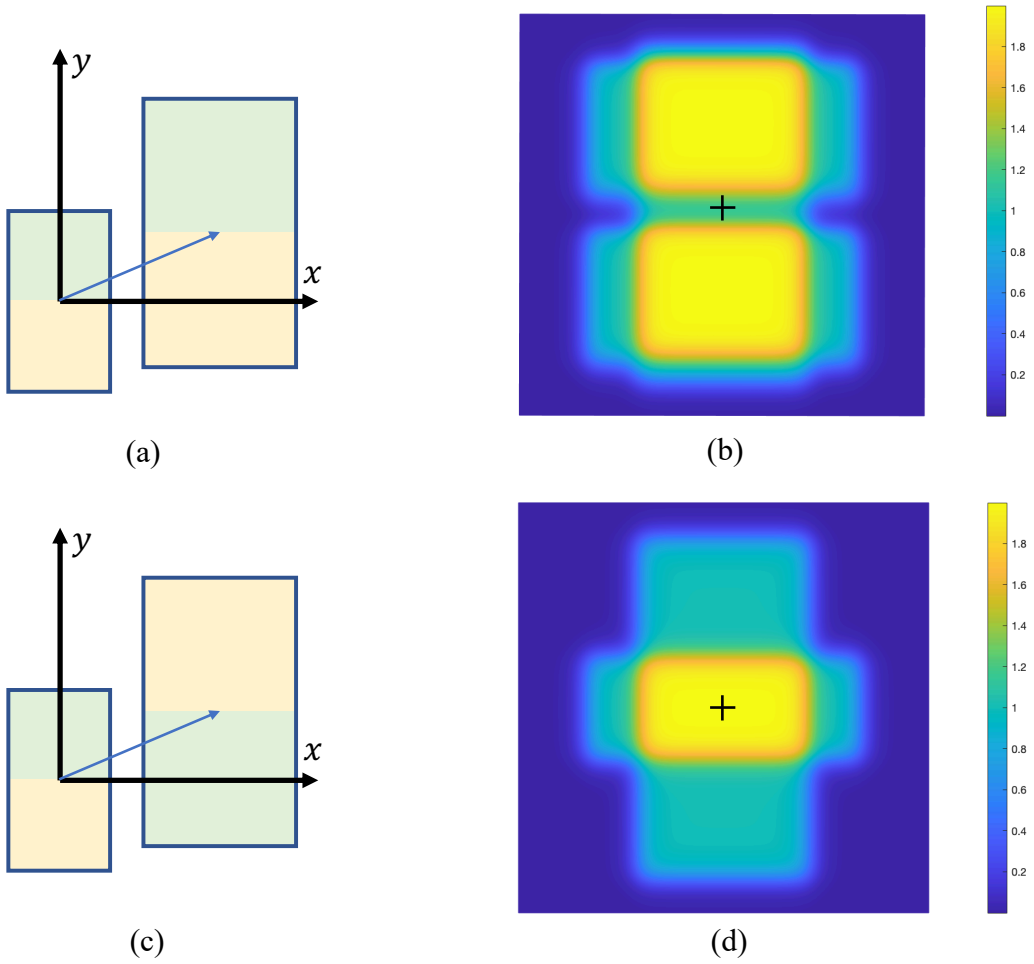


Figure 2.6: Interaction between two gridded cells. (a) two gridded cells with the same orientation, (b) penalty function v.s. relative location for the same orientation, (c) two gridded with opposite orientations, (d) penalty function v.s. relative location for different orientations. The "+" mark indicates the origin point.

overlap constraint in the green region; the cell arrangement violates both constraints in the yellow region. As can be seen from this figure, the penalty function has two notches on both sides of the first cell around its N/P-well borderline. The width of these notches is m_s, and the height is determined by N/P-well heights in both cells. Strictly speaking, we need to extend N/P-wells in one cell towards another to see these notches; otherwise, these two notches will degenerate into two line segments. If the second cell is not in any notch, this cell needs to be at least m_s away from the first cell. In other words, to legally abut two cells, their N/P-well borderlines must be close enough.

Two Cells with Opposite Orientations

Fig. 2.6 (c) shows two cells with opposite orientations. The overlap cost function is similar:

$$overlap(x, y) = b(x, -w, w)b(y, -h_l, h_u), \tag{2.7}$$

where

$$\begin{aligned} w &= (w_1 + w_2)/2, \\ h_l &= (p_1 + p_2), \\ h_u &= (n_1 + n_2). \end{aligned} \tag{2.8}$$

The N/P-well design rule penalty function becomes

$$well(x, y) = b(x, x_l, x_u)b(y, y_l, y_u), \tag{2.9}$$

where

$$\begin{aligned} x_l &= -(w + m_s), \\ x_u &= w + m_s, \\ y_l &= \min(-m_w, -m_s), \\ y_u &= \max(m_w, m_s). \end{aligned} \tag{2.10}$$

Fig. 2.6 (d) shows the total penalty after smoothing. Unlike the previous case, there are two bumps on both sides of the first cell, which means the second cell needs to be at least m_s away from the first cell when their N/P-well borderlines have similar vertical locations.

Cell Cluster

From Fig. 2.6 (b) and (d), we can find that when two gridded cells are close to each other horizontally: if they have the same orientation, an attractive force will try to align their N/P-well borderlines; if they have opposite orientations, a repulsive force pushes them away from each other. And if the objective function contains the placement area, two adjacent cells should have the same orientation and abut each other to reduce the total area.

The underlying assumption of the above analysis is that a cell is fully covered by its

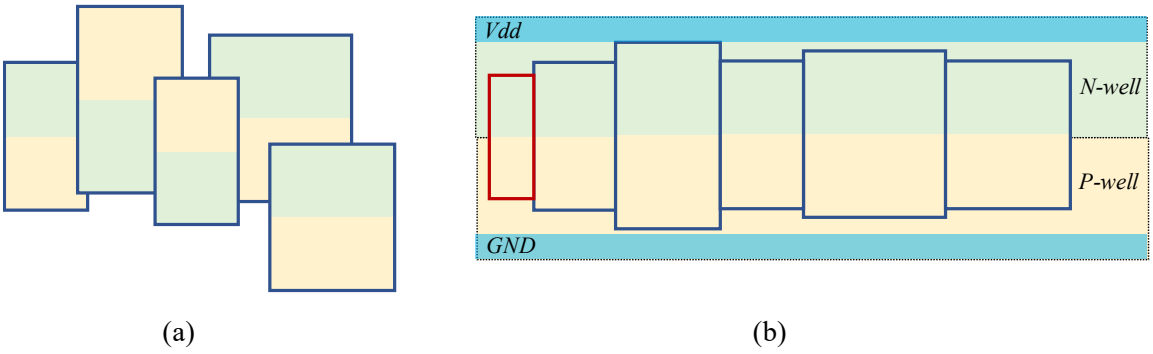


Figure 2.7: Individual gridded cells form clusters. (a) individual gridded cells with different orientations, (b) gridded cells form a cluster to share a large N/P-well and a well tap cell (the cell with red boundary).

N-well and P-well. This is valid for modern technology nodes due to the min-area rule: the minimum N/P-well area is big enough to cover several cells, and a simple cell cannot satisfy this rule. There are two ways to make the N/P-well in a cell satisfy this rule: making the well in a cell larger or sharing a big N/P-well among cells. The latter solution is consistent with horizontal and vertical cell abutment.

Fig. 2.7 shows the process of several individual gridded cells forming a cluster to satisfy N/P-well design rules: cells in this cluster have the same orientation and share a common N/P-well, and the area of the shared N/P-well is larger than the minimum area; a well tap cell is created in this cluster to prevent the latch-up effect; the alignment of N/P-well boundaries together with the shared N/P-wells satisfies the min-space and min-width rule.

There is some placement space wasted after the formation of clusters because the worst-case N/P-well height determines the cluster height. But we must also notice that the formation of clusters allows the horizontal abutment of cells, which saves the placement area and satisfies design rules.

The formation of clusters can also simplify the power distribution network. As shown in Fig. 2.7 (b), all cells have the same orientation, which means all pull-up networks are on the top half of the cluster. This allows us to create a horizontal power rail along the top boundary and connect all Vdd cell pins to this power rail. We can do the same thing for GND cell pins, and this leads to a cluster surrounded by power rails.

2.4.4 Cluster-based Placement

The emergence of the cluster adds a new level to the placement hierarchy. Now the gridded cell placement flow can be divided into the following three steps:

- *cell clustering*: for every cell, the gridded cell placer creates a new cluster or finds an existing cluster to accommodate this cell;
- *cluster legalization*: design rule violations among clusters are eliminated by adjusting the location and orientation of clusters;
- *power distribution*: the power rails in clusters are connected to the corresponding power supplies.

The objective function is still circuit performance, power, and area.

We have tried various methods for cell clustering and cluster legalization. One way to cluster cells is the following: for each cell, if there are any clusters around it, this cell will be grouped into the nearest cluster; otherwise, a new cluster will be created around this cell. This clustering method will lead to many overlaps among clusters because clusters remain unchanged once created. Therefore, we need to figure out a way to legalize cluster locations, but there are two problems during this process. The first one is the unconsolidated layout: it is challenging to place clusters compactly because they have various widths. The second problem is large cell displacements: a cluster can carry cells moving a long distance to find a legal position. A declustering and reclustering technique can mitigate these problems, but the result is still barely satisfactory.

It is observed that when clusters have similar widths, the final layout can be very compact because these clusters will form cluster stacks after legalization. This equal-width heuristic solves the first problem, but the cluster displacement problem still exists. It is desirable to construct clusters without overlap in the first place. The solution for this problem is the second heuristic: since equal-width clusters will eventually form stacks after legalization, we can create a floorplan for these cluster stacks in advance, and clusters can be developed in each sub-region without overlaps among them. This heuristic can avoid cluster displacement and thus large cell displacements.

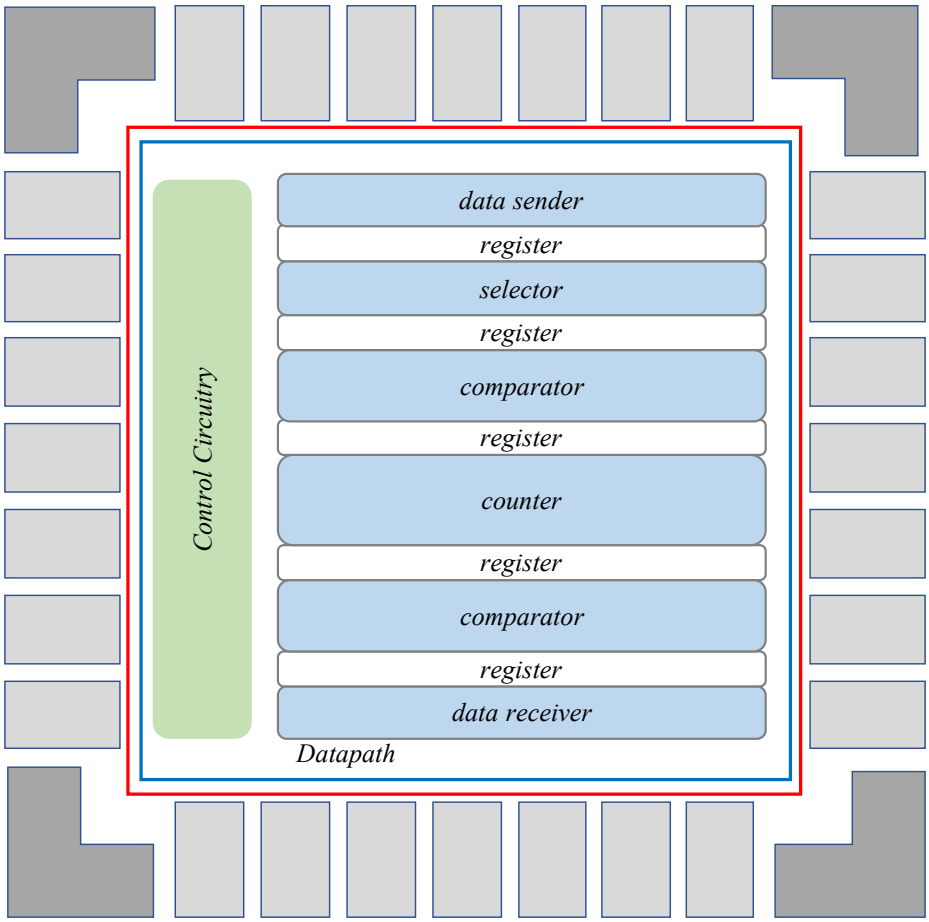


Figure 2.8: The floorplan for an asynchronous data decompressor.

These two heuristics coincide with the practices adopted by the full-custom methodology: the former corresponds to the design of wordslices or bitslices [4], and the latter corresponds to the floorplanning step. Fig. 2.8 shows the floorplan for an asynchronous design using the full-custom methodology. The datapath has many stages, each of which consists of one or more cell clusters with a similar width. Although the control circuitry is not as organized as the datapath, control cells also form local clusters.

The lesson from these trials is that it is unfitting to take the formation of clusters as a goal. Instead, we should treat the cluster formation process as a way to organize gridded cells. The following chapter will introduce our algorithms and implementations in detail.

Chapter 3

Dali: A Gridded Cell Placement Flow

Asynchronous Very-Large-Scale-Integration (VLSI) has several potential benefits over its synchronous counterparts, such as reduced power consumption, elastic pipelining, and robustness to variations. However, the lack of electronic design automation (EDA) support for asynchronous circuits, especially physical layout automation tools, largely limits their adoption. To tackle this challenge, we propose a *gridded cell* layout methodology for asynchronous circuits, in which the cell height and cell width can be any integer multiple of two grid values. The gridded cell approach combines the shape regularity of standard cells with the size flexibility of custom design, and thus achieves a better space utilization ratio and lower wire-length for asynchronous designs. We present the algorithms and our implementation of Dali, a gridded cell placer, that consists of an analytical global placer, a forward-backward legalizer, an N/P-well legalizer, and a power grid router. We show that the gridded cell placement approach reduces area by **15**% without impacting the routability of the design. We have also used Dali to tape out a chip in a 65nm process technology, demonstrating that our placer generates design-rule clean placement.

3.1 Introduction

Asynchronous circuits, also known as self-timed circuits, use local control signals instead of a global clock signal for synchronization [75]. The absence of the clock signal eliminates clock tree synthesis and distribution, alleviates global timing issues, and simplifies timing closure. The use of local handshake control signal allows asynchronous designs to achieve average-case instead of worst-case performance, and also makes circuits robust against environmental noises and fabrication variations [94]. However, despite these advantages, the lack of dedicated EDA support and the absence of commercial dedicated standard cell libraries hamper the acceptance of asynchronous circuits [102, 103].

Many asynchronous circuit families use nonstandard and customized logic gates to achieve small area and high performance [77, 97]. Although these gates can be converted to commercial standard cells via technology mapping, this often leads to a substantially larger area and power, and sometimes additional timing constraints, leading to an inferior final circuit [77, 99]. Therefore, the full-custom design methodology is typically used for implementing asynchronous circuits that have state-of-the-art power/performance [28, 35]. However, this approach is infeasible for most ASIC design projects due to the long design time and high design cost [104].

Another option is a hybrid approach, which manually creates a custom standard cell library for asynchronous design, and uses commercial EDA tools for layout automation [99, 105–107]. However, previous studies have shown that the space utilization ratio of customized standard cells can be very inefficient [99], because the standard cell height is determined by the worst-case cell height across all the customized cells. Smaller cells have to be "bloated" to the maximum height across the entire library [108, 109]. This leads to a large area gap between the customized standard cell approach and the full-custom approach for the same asynchronous circuit netlist. This phenomenon is more prominent when a circuit consists of many cells with small transistors but few cells with large ones.

To address this challenge, this paper introduces the *gridded cell* approach for asynchronous circuit layout automation. This approach combines the shape regularity of standard cells and size flexibility of custom cells. Similar to a standard cell, a gridded cell has

a rectangular shape, but its height can be any integer multiple of a given grid value, which we take to be the metal track pitch. This approach opens new degrees of freedom, and thus can potentially lead to a better space utilization ratio, especially in a design that combines different circuit styles.

Gridded cells introduce two additional problems for placement: (i) power delivery, since the standard metal straps at the top and bottom of standard cell libraries cannot be used as the means for connecting power and ground; and (ii) well legalization, since simply abutting cells with different heights can cause design rule errors. We present solutions to both of these problems as well.

We implement this placement approach in *Dali*, an open-source placer for asynchronous circuits. We demonstrate that Dali reduces the die area by **15**% without impacting the routability of the placed design or the runtime of the place and route flow compared to the standard cell approach for the same design netlist.

The key contributions of this work are listed as follows:

- We propose and implement the gridded cell layout methodology, specially tailored for asynchronous circuits;
- We present an efficient legalization algorithm for gridded cells, loosely based on Tetris-style legalization in standard cells;
- We present a new N/P-well legalization approach and algorithm that is capable of legalizing well geometries in the presence of varying cell heights;
- We present a power grid routing strategy, and implement it as part of the placement flow;
- To demonstrate the effectiveness of our approach, we have submitted a chip for fabrication in a 65nm process that was implemented with this methodology.

The rest of this paper is organized as follows: Section 3.2 presents a brief overview of standard cell-based placement, and Section 3.3 describes differences with the gridded cell approach. Section 3.4 explains the core placement techniques for gridded cell designs.

Section 3.5 reports the experimental results. Finally, the summary and future work are given in Section 3.6.

3.2 Background

3.2.1 Standard Cell Layout

Fig. 3.1 (a) and (b) shows the CMOS layout of an INV gate and a NAND gate designed in the standard cell fashion. In this approach, the worst-case N/P-well height determines the height of standard cells, and thus the height of the INV gate is the same as the NAND gate, although the INV gate has a smaller pull-down transistor as it has fewer gates in series. For modern CMOS processes, the width and height of a standard cell are integer multiple of the low-layer metal pitch, and the standard cell height is typically between seven and twelve metal pitches.

Another feature of standard cell layout is power rails at the bottom and the top of the cell. Standard cells with possible fillers can be abutted horizontally along a row, and their power rails and N/P-wells are connected automatically, as shown in Fig. 3.1 (a) and (b). To abut two cells vertically, one of the cells is mirrored along the horizontal direction to avoid the direct contact of VDD and GND and for sharing of the power/ground rails. Additional cells are needed to avoid design rule violations, including well tap cells that are often pre-placed in a uniform pattern across the placement region to prevent latch-up, and filler cells that the placer inserts into the gaps between standard cells to provide power rail and well continuity along each row. Finally, in more modern technologies, end cap cells are also necessary to handle additional design rule restrictions.

3.2.2 Row-based Placement

Standard cells are usually provided by semiconductor foundries, and the physical layout of nearly all simple cells have the same height, although a small percentage of cells may have multi-heights [110]. A commonly adopted practice for standard cell design is row-based placement, also known as standard cell placement: many rows of empty spaces are created inside the die area during floor-planning for accommodating logic cells of the same

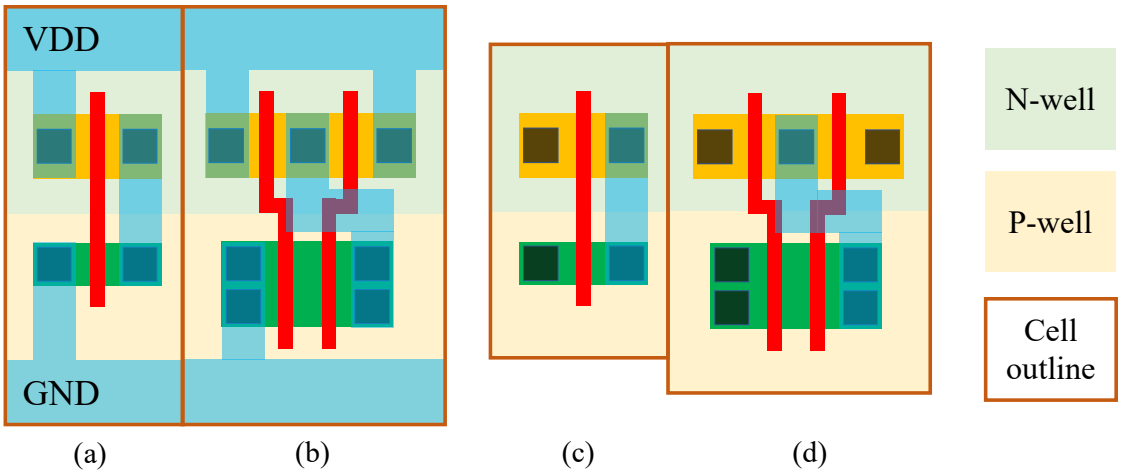


Figure 3.1: CMOS layout of the INV gate and NAND gate. Standard cell implementation: (a) & (b) standard cell height and power rails. Gridded cell implementation: (c) & (d) flexible cell height and no power rails.

height [38]. These rows come with N/P-wells of regular shapes and pre-placed well tap cells. This kind of N/P-well arrangement relieves the standard cell placement from N/P-well design rules, and thus simplifies the automation of the placement process [111]. Row-based placement can lead to a compact physical layout and high space utilization ratio if every standard cell conforms to these properties [99].

Row-based placement automation has been extensively studied in the past several decades. There are many excellent placement techniques for standard cell designs. Academic standard cell placers include: TimberWolf [50] uses simulated annealing to optimize wire-length cost; FengShui5 [48] and Capo [49] adopt a recursive bisection approach; NTUPlace3 [64] models wire-length cost and cell overlap as non-linear functions; FastPlace [59], SimPL [60], and POLAR [62] approximate wire-length cost as a quadratic function and use white-space allocation heuristics to roughly remove cell overlaps; ePlace [68] and RePlAce [66] view cells as uniformly charged plates to balance density and wire-length. A more detailed review of placement techniques can be found in [57]. All of these techniques treat the placement problem as a constraint optimization problem. Together with detailed placement algorithms [70] and legalization algorithms [69,112], these techniques can tackle the standard cell placement problem effectively and efficiently.

3.3 Gridded cell

3.3.1 Gridded cell layout

As shown in Fig. 3.1 (c) and (d), a gridded cell layout is similar to its standard cell counterpart, except that the former has (i) more flexible height and (ii) no power rails. These two differences bring new problems to gridded cell placement: N/P-well design rule satisfaction and new power grid design.

The height of a gridded cell is integer multiples of the low-layer metal pitch, while the standard cells have a fixed height. Therefore, in the placement region, the y-coordinate of standard cells is discretized by the standard cell height, while that of gridded cells is discretized by the metal pitch. The higher resolution along the y-direction enables more flexible cell placement but results in additional problems to satisfy N/P-well design rules. These problems are discussed in more detail in Section 3.4.5.

Gridded cell abutment is similar to standard cell abutment. For horizontal abutment, one needs to align the N/P-well boundaries of two gridded cells to share their wells, as shown in Fig. 3.1 (c) & (d). The vertical abutment also requires one of the gridded cells to be mirrored along the horizontal direction. The N/P-well width and height of gridded cells are selected to be wide enough to avoid design rule violations during abutment.

The removal of power rails in gridded cells is to respect metal layer design rules. Imagine the horizontal abutment of two gridded cells with power rails: due to different cell heights, power rails in these two cells may disconnect with each other and potentially violate metal layer design rules. The absence of power rails requires a new power grid design that must be constructed after placement.

3.3.2 Cluster-based Placement

We have explored a large number of placement and legalization options for gridded cell designs, including adapting existing legalization approaches [50, 64, 69, 113]. However, permitting fully flexible placement resulted in significant increases in wire-length (up to 50%) during the N/P-well legalization phase. To avoid this problem, we developed a *cluster-based* placement strategy for gridded cell layout that is detailed in Section 3.4.

Cluster-based placement is based on our empirical observation that groups of neighboring cells looked like "mini-rows" after all cell overlaps were removed. In our approach, cells are placed into clusters created dynamically during the N/P-well legalization phase. These clusters can be viewed as local rows, whose heights and orientations are determined at runtime. Within an individual cluster, the shapes of N/P-well regions can be easily determined, and well tap cells can be inserted for each cluster. To the best of our knowledge, this is the first time that N/P-well design rules are considered during placement for a large-scale digital design.

3.4 Gridded Cell Placement

3.4.1 Placement Problem Formulation

A circuit can be represented as a hyper-graph $G = (C, E)$, where C is the set of gridded cells, and E is the set of nets. Given such an input circuit, the output of the placement flow should be a legal cell arrangement, which satisfies the following constraints:

1. there is no overlapping among cells;
2. every cell resides in appropriate N/P-wells;
3. VDD/GND pins and N/P-wells are connected to the correct power rails;

Various objective functions need to be optimized during placement, including cell density, wire-length, critical path delay, area, power, etc. Among these objective functions, cell density and wire-length are the most common performance metrics. Cell density can be easily evaluated using a grid mesh defined across the placement region. Nevertheless, the exact wire-length is computationally expensive to obtain, and thus efficient wire-length estimations are adopted in practice. The half-perimeter wire-length (HPWL) is the most commonly used metric for wire length estimation, and is half the perimeter of the bounding box of the endpoints of the net:

$$HPWL(\boldsymbol{x}, \boldsymbol{y}) = \sum_{e \in E} \left[(\max_{i \in e} x_i - \min_{i \in e} x_i) + (\max_{i \in e} y_i - \min_{i \in e} y_i) \right], \tag{3.1}$$

where vector $\boldsymbol{x} = \{x_1, x_2, ..., x_n\}$ and vector $\boldsymbol{y} = \{y_1, y_2, ..., y_n\}$ are the x and y-coordinates of cells, respectively, and $i \in e$ denotes the relationship that net e contains a pin in cell c_i. Note that vector $\boldsymbol{x}$ and $\boldsymbol{y}$ and independent in Eqn. 3.1, and thus HPWL can be decomposed into two components containing either $\boldsymbol{x}$ or $\boldsymbol{y}$.

3.4.2 Placement Flow

As shown in Fig. 3.2, our gridded cell placement flow consists of four stages: global placement, forward-backward legalization, N/P-well legalization, and power grid design. These four stages are designed to optimize HPWL and satisfy different constraints. The global placement step uses existing approaches to optimize wire-length cost and meanwhile keep the cell density less than a given upper limit [60, 62]. The forward-backward legalization step removes overlapping among gridded cells, and only makes local cell displacement using a forward-backward legalization technique. Based on this overlap-free result, cells are assigned to placement sub-regions, and in each sub-region, cell clustering is performed for N/P-well generation and well tap cell creation. At last, based on the structure of cell clusters, a power mesh is generated, and a detailed power routing process connects VDD and GND pins in each cell to their closest power-lines.

3.4.3 Global Placement

We implement an analytical global placer for gridded cells using techniques similar to SimPL [60] and POLAR [62]. The goal of this step is to minimize HPWL and eliminate cell density overflow. As shown in the first panel of Fig. 3.2, our global placement implementation consists of two major parts: wire-length optimization and cell density control. However, these two steps compete with each other: the former step optimizes wire-length but ignores cell density, while the latter step eliminates density overflow but largely neglects wire-length. The global placer finds a balance point for the trade-off between wire-length and cell density using anchors.

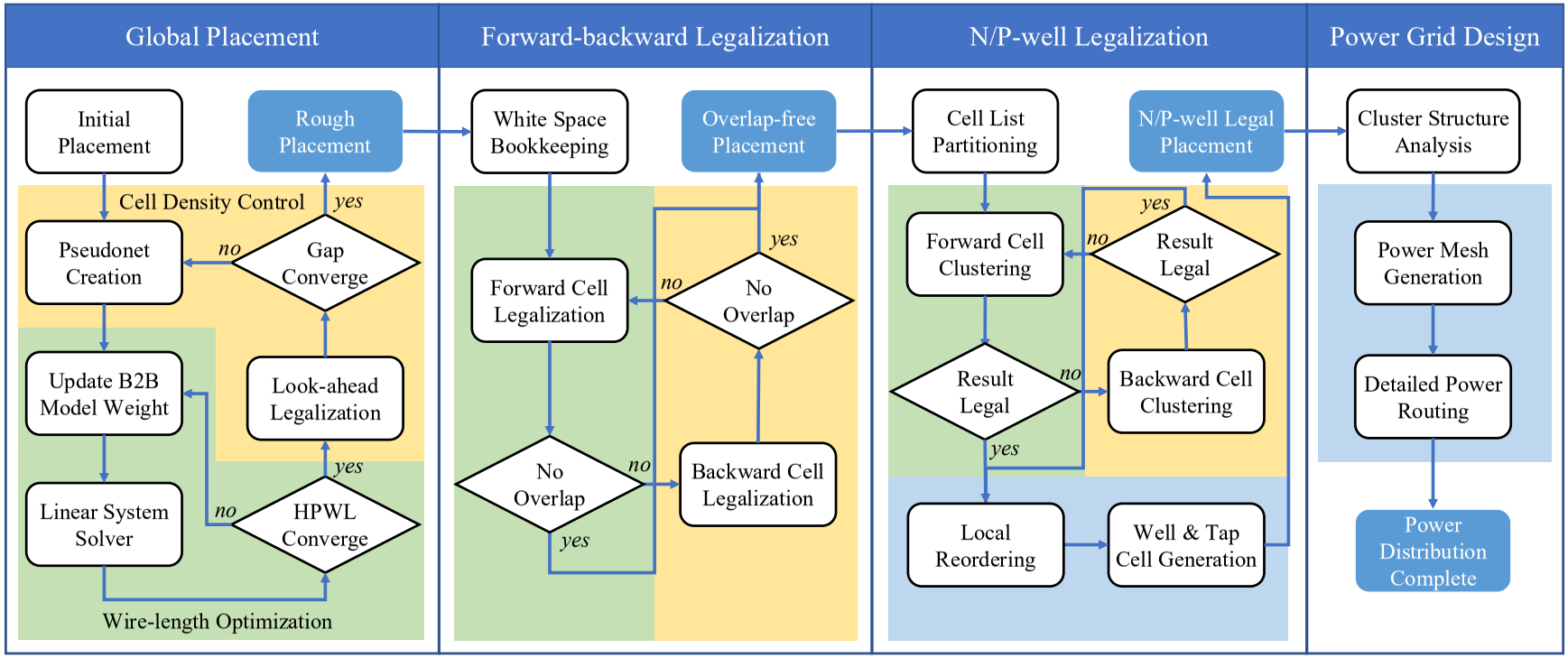


Figure 3.2: Schematic diagram of the whole gridded cell placement flow: global placement, forward-backward Legalization, N/P-well legalization, and power grid design. Each section in the diagram contains a blue box, showing its output.

Wire-length optimization

Although the direct optimization of HPWL in the form of Eqn. 3.1 is possible by using steepest descent, a faster approach is conjugate gradient descent by converting HPWL to a quadratic form [59]. The x-component (similar for y-component) of HPWL can be reformed as

$$HPWL(\boldsymbol{x}) = \sum_{e \in E} \sum_{i,j \in e} w_{ij,e}(x_i - x_j)^2, \tag{3.2}$$

and coefficient $w_{ij,e}$ is determined by the Bound2Bound (B2B) net model [58]: if cell c_i or cell c_j contains an extreme pin (max or min) in net e, the corresponding net weight is set to

$$w_{ij,e} = \frac{1}{(p_e - 1)|x_i - x_j|}, \tag{3.3}$$

where p_e is the number of pins connected by net e; otherwise, coefficient $w_{ij,e}$ is zero.

As shown in the global placement flow in Fig. 3.2, the initial value of $w_{ij,e}$ is generated by uniformly distributing cells on the placement region. With these initial net weights, the global minimum of the quadratic HPWL can be obtained by setting its first derivative to zero as the following:

$$\sum_{e \in E} \sum_{i,j \in e} w_{ij,e}(x_i - x_j) = 0. \tag{3.4}$$

This set of linear equations can be solved efficiently using an iterative linear solver with a Jacobi preconditioner [114]. Note that the solution of x-coordinates can be different from their initial values, and net weights $w_{ij,e}$ should change accordingly. Therefore, with these new x-coordinates, the B2B net model is updated, and a new linear system is constructed and solved. This process is repeated until the solution converges.

Look-ahead legalization

A placement from solving Eqn. 3.4 contains considerable cell density overflow and overlapping. Look-ahead legalization eliminates density overflow using a recursive top-down geometric partitioning technique.

Our implementation of look-ahead legalization is similar to POLAR [62]. First, we build a uniform bin grid across the placement region, and each bin can accommodate around 15×

average movable cell area without exceeding the target density. Second, density overflow bins are identified and grouped into bin clusters. Third, the legalizer finds a big enough expansion region for the largest bin cluster. After that, a top-down geometric partitioning step distributes movable cells to available space in the expansion region. These steps are repeated until no overflow bin exists.

Anchor

A placement solution obtained from wire-length optimization has a small HPWL but a significant amount of cell overlap. On the contrary, a solution from look-ahead legalization has a large HPWL but no density overflow. The introduction of anchor makes it possible to obtain a placement with reasonably small HPWL and no density overflow by including a penalty term in the wire-length cost:

$$WL(\boldsymbol{x}, \boldsymbol{a}) = \sum_{e \in E} \sum_{i,j \in e} w_{ij,e}(x_i - x_j)^2 + \alpha \sum_{i \in V} w_i (x_i - a_i)^2, \tag{3.5}$$

where $\boldsymbol{a}$ is a known anchor solution, $\alpha \geq 0$ is a hyper-parameter, and w_i is the B2B net weight of a pseudo-net connecting each cell c_i to its own anchor with x-coordinate a_i. In Eqn. 3.5, the first term is the HPWL of all nets, and the second term is the weighted HPWL of pseudo-nets. As can be seen, when α is zero, this new wire-length cost becomes HPWL; when α is large, the final value of $\boldsymbol{x}$ is close to $\boldsymbol{a}$ in order to minimize total pseudo-net wire-length.

In practice, look-ahead legalization results act as anchor solutions, and wire-length cost in Eqn. 3.5 is the objective function during wire-length optimization. Wire-length optimization followed by look-ahead legalization is repeated, and meanwhile the hyper-parameter α increases in proportion to the iteration number with initial value zero. With anchors acting as an effective way to integrate wire-length optimization and look-ahead legalization, the repetition of these two steps gradually improves the quality of placement and makes the HPWL gap converge.

3.4.4 Forward-backward Legalization

Among existing legalization techniques, Tetris legalization can be easily adapted to handle gridded cell legalization [69], unlike legalization techniques like Abacus [112] that depend on pre-defined rows to reduce cell movement during legalization. However, Tetris legalization introduces a bias toward the left boundary of the placement region and sometimes leads to large cell displacements. To avoid these problems, we extend the Tetris algorithm and propose an iterative forward-backward legalization algorithm for removing gridded cell overlaps. As shown in Fig. 3.2, our implementation consists of a forward legalization phase and a backward legalization phase. This algorithm aims to remove cell overlaps, and meanwhile seeks to maintain the order of cells. To preserve the quality of global placement, only cells with illegal locations are moved, and large displacements are discouraged by the use of transient locations.

Forward legalization

Algorithm 1 shows the outline of the forward legalization algorithm. First, the list of cells is sorted by their x-coordinates in ascending order. Then for each cell, the forward legalizer checks its overlap with previously placed cells and fixed blockages: if there is no overlap, this cell will remain unmoved; otherwise, the legalizer will try to find a legal location for this cell by a local search around its current location. Note that there may be no legal location in a small search range. In this case, the conventional Tetris legalization enlarges the search region until a legal location is found, while the forward legalization places this cell on a transient yet illegal location temporarily to avoid large displacement. A legal location for this cell will be found in subsequent legalization iterations.

Transient locations

Fig. 3.3 shows an example to illustrate the forward-backward legalization using transient locations. A global placement is given in Fig. 3.3 (a), where blue boxes are movable cells and the green box is a fixed macro. As shown in Fig. 3.3 (b), during the first iteration of the forward legalization, the preceding three cells have no overlap with previously placed

Algorithm 1: Forward cell legalization

Input: cell list C, global iteration number k
Output: *status* of this iteration
Result: placement with less cell overlaps

```
Sort cell list C by x-coordinate in ascending order;
Initialize the cell front contour to the left boundary;
k := k + 1; status := True;
foreach cell c ∈ C do
    if c overlaps with blockages or placed cells then
        S_t := ∅, S_l := ∅;
        h := k * c.height;
        lx := c.x − k * c.width;
        foreach y ∈ range(c.y − h, c.y + h) do
            if a transient location x > lx found then
                Append location (x, y) to S_t;
            if a legal location x > lx found then
                Append location (x, y) to S_l;
        if S_l is an empty set then
            status := False;
            Place c to the closest transient location in S_t;
        else
            Place c to the closest legal location in S_l;
    Update cell front contour;
```

cells and the fixed macro, and thus their locations remain unchanged. The fourth cell, highlighted in dark blue, has an illegal location because it overlaps with two preceding cells and the blockage. To find a new location for this cell, a local search range (dashed blue box) is constructed in the following way: the center of the search range sits on the center of this cell, and the width and height are proportional to the current iteration number. As can be seen, candidate possible legal locations (dashed black boxes) are out of this local search range, and thus there are no legal locations found during the first iteration.

As shown in Fig. 3.3 (c), since no vacant site is found for the dark blue cell, this cell is temporarily moved to a transient location, which is the closest grid location to its initial position but on the right-hand side of all previously placed cells. A cell front contour (green dashed line) is constructed to track the right edges of placed cells. In this way, the order of cells is largely maintained, and overlapping among movable cells is reduced by moving cells away from a dense region. Because the dark blue cell in the transient location still overlaps with the blockage, this iteration of the forward legalization fails, and then a backward

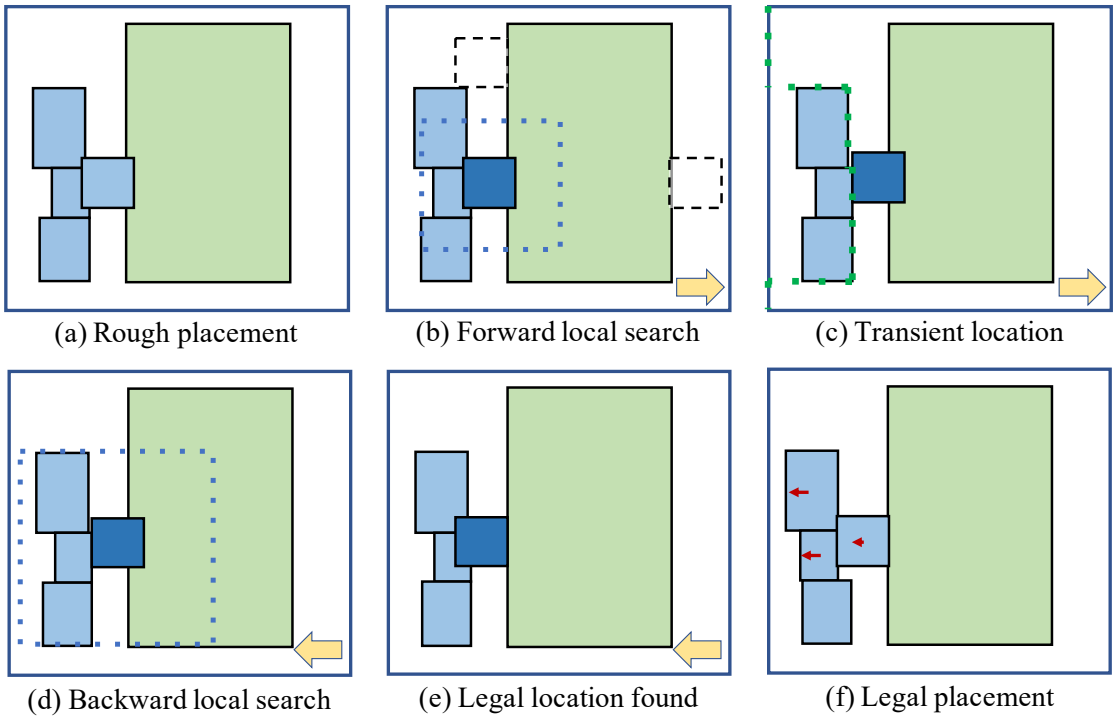


Figure 3.3: Illustration of the use of transient locations during forward-backward legalization. Yellow arrows indicate the direction of legalization.

legalization phase is invoked with the global iteration number increased by one.

Backward legalization

During the second iteration of backward legalization, cells are sorted by their x-coordinates in descending order, and thus the dark blue cell is the first cell checked by the legalizer. Since this cell overlaps with the blockage, a local search range is constructed, as shown in Fig. 3.3 (d). Because the size of a local search range is proportional to the iteration number, this search range is larger than the search window during the first iteration. A local legal location can be easily found in this rang, as shown in Fig. 3.3 (e). The legal locations of subsequent cells can be found in the same way. As can be seen from cell displacements (red arrows) shown in Fig. 3.3 (f), all cells are moved locally in the final overlap-free placement.

This forward-backward legalization algorithm is efficient and easy to implement. Experimental results show that it only takes one to two iterations to legalize a gridded cell design. Besides, if the first several iterations fail to legalize the placement, the forward-backward

legalization will gradually degrade to the conventional Tetris legalization during the local search.

3.4.5 N/P-well Legalization

Due to the flexibility of the gridded cell height, a gridded cell placement is unlikely to be design rule checking (DRC) clean even if it is overlap-free. Without an organized placement of gridded cells, N/P-well design rules can be violated, including minimum width rule, minimum spacing rule, minimum area rule, short vertex rules, and latch-up prevention rule. We propose an N/P-well legalization algorithm to eliminate these violations by partitioning the placement region and clustering gridded cells.

Our N/P-well legalization algorithm consists of four major steps, each of which aims to remove different kinds of design rule violations. First, the placement region is partitioned into many sub-regions, and each cell is assigned to its closest sub-region. Second, in each sub-region, a forward-backward cell clustering is performed vertically to construct clusters. Then, these clusters are arranged in a back-to-back manner for power delivery and area. After that, a local reordering step optimizes wire-length cost using an existing technique [70]. Finally, based on the orientation of clusters, well tap cells are placed in each cluster. The design is then fully legalized by constructing a rectangular region for the N/P wells for the cluster, thereby adding additional geometry to the layout during placement.

N/P-well design rule violations

An overlap-free placement is shown in Fig. 3.4 (a). As can be seen, although there is no gridded cell overlap, the disorganized arrangement of N/P-wells leads to many design rule violations. For example, small gridded cells have N/P-wells with an area less than the minimum requirement, and thus when these cells abut no other cells, minimum area violations are triggered. For some other cells, although they satisfy minimum area rule by N/P-well sharing, their common well boundaries are too short to satisfy minimum width rule. Besides, no wells in gridded cells are connected to power supplies, leading to latch-up prevention violations.

Algorithm 2: Forward cell clustering

Input: placement region, sub-region set S
Output: *status* of this iteration
Result: N/P-well design-rule clean placement

```
1  status := True;
2  foreach sub-region s ∈ S do
3      Initialize cluster queue in s: Q := ∅;
4      Sort cells in sub-region s by y in ascending order;
5      foreach cell c belongs to s do
6          if cluster Q.back() cannot accommodate c then
7              Append an empty cluster to Q;
8          Insert cell c to Q.back();
9          Update the shape and location of cluster Q.back();
10         Align N/P-well boundaries of cells in Q.back();
11     if all clusters inside the region of s then
12         Set cell orientation in cluster Q.front() to be N;
13         foreach cluster ∈ s do
14             Legalize cells in cluster;
15             Set orientation opposite to its previous cluster;
16     else
17         status := False;
```

Placement region partitioning

We borrow some ideas from the macro design methodology to develop our N/P-well legalization algorithm [104]. In the example shown in Fig. 3.4, the whole placement region is divided into three sub-regions, each of which can be viewed as a macro. Small gaps are reserved among sub-regions for both power delivery and minimum spacing rule. The entire cell list is partitioned into three sub-lists based on cell distance to sub-regions, and each sub-list belongs to its corresponding sub-region.

Cell clustering

The forward cell clustering algorithm is outlined in Algorithm 2. In each sub-region, cells are sorted by their y-coordinate. For the first cell in a sub-region, the legalizer constructs an empty cluster and places this cell into it. The width and lower left x-coordinate of this cluster are the same as those of the sub-region it belongs to, and the height and lower left y-coordinate are the same as those of the first cell. For the subsequent cell, if the last cluster has enough white space and overlaps with this cell, the legalizer will place this cell

into the last cluster; otherwise, a new cluster for this cell will be constructed similarly. When a cell is placed into a cluster, its N/P-well boundary is aligned with other cells in this cluster, and the cluster height is adjusted accordingly. This process from bottom to top along each sub-region repeats in this way until all cells are placed into clusters, as shown in Fig. 3.4 (b). The backward clustering places cells in the reverse order when cells spill out of a sub-region. Clusters together with cells are further arranged in back-to-back fashion for compact layout and power delivery, as shown in Fig. 3.4 (c).

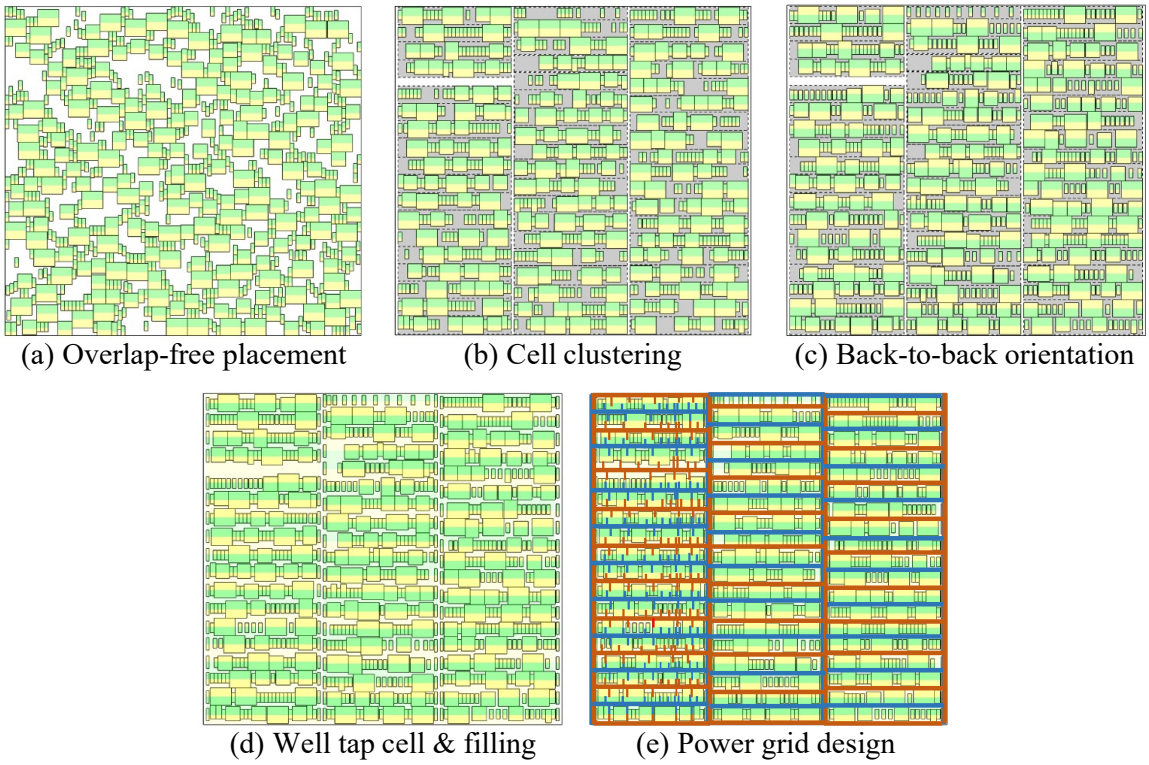


Figure 3.4: A simple example to illustrate N/P-well legalization and power grid design. (a) N-wells in gridded cells are shown in light green, and P-wells in light yellow. (b) & (c) Grey boxes with dashed black edges are clusters. (d) N/P-wells are filled between clusters. (e) Orange and blue lines denote VDD and GND wires.

Local reordering

With clusters constructed, a local cell reordering technique optimizes wire-length cost without creating any design rule violations [70]. For gridded cells in each cluster, we try all possible local reordering using a sliding window with capacity 3 to obtain the best cell or-

der. Finally, the legalizer creates well tap cells for each cluster and draws N/P-well fillings to prevent latch-up. In this example, well tap cells are placed at both ends of each cluster, as shown in Fig. 3.4 (d).

3.4.6 Power Grid Design

Since standard cell design widely adopts row-based placement and the VDD/GND pins are at the top and bottom edge of the cell, power grid design for standard cells builds alternate VDD/GND stripes between rows to make pins automatically connect. Thus power grid design only needs the row information and is completed before placement. In this work, the power grid design must be adaptive to the cluster structure and is constructed after placement. There are two major differences from standard cell power design: (1) Rows must be adaptive to the cluster structure. (2) A special detailed router is needed to connect VDD/GND pins to the mesh since they may not be at the top or bottom edge of the cell.

As shown in the design flow Fig. 3.2, the power grid design consists of two steps: power mesh generation and detailed power routing. First, power mesh generation builds chip-height vertical mesh and column-width horizontal mesh according to the cluster structure. The chip-height vertical mesh is placed between columns in a pair of VDD/GND wires. The column-width horizontal mesh is placed between rows inside a column and alternates between VDD/GND. As shown in Fig. 3.4 (e), these meshes partition the chip area into many rows and provide power delivery to each row. Similar to power design for standard cells, wide high layer stripes are generated as well to reduce IR drop. The wide high layer stripes are not shown in the figure. The second step detailed power routing connects VDD/GND pins of each cell to the horizontal mesh. This step generates standard-width wires on track inside a cell respecting all design rules. It hooks up the closest points of VDD/GND pins to the corresponding mesh on each track, and tries to use the lowest metal layer first to avoid consuming the space for signal routing. If such a solution is not found, then higher layers are used. Note that it only routes inside a cell: a solution using the free space of other cells is not allowed. In Fig. 3.4 (e), the vertical short wires in the first column represent wires generated by this step. More details can be found in [115].

3.5 Experimental Results

We implement Dali in C++11, and compile it using GCC 9.3.0 on a Linux machine with 32 GB memory and Intel Core i7-8750H CPU running at 3.95 GHz. Dali uses only one CPU core to perform experiments and benchmark runs. A commercial tool completes the routing of normal signals and evaluates the HPWL of placement solutions as well as post-routing wire-length.

To validate the effectiveness of our algorithms and implementation, we used Dali as the placer for an asynchronous microprocessor (a stack machine) implemented with the gridded cell approach. We used commercial DRC signoff tools to verify that the placement result was DRC clean, and have taped out this chip in a 65nm CMOS process with routing completed using a commercial detailed router. We observed that the tool run-time for detailed routing was similar for both standard cell and gridded cell placement. We remark that because we use the lower-level metal track pitches as the placement grid, pins on the routing grid in a cell remain on the routing grid for any legal gridded placement.

Benchmarks are challenging to obtain, and asynchronous circuit benchmarks are even more rare. We report results from two sets of benchmarks: (i) a collection of asynchronous stack-based microprocessors generated using different parameterizations (des1, des3, des4), and a RISC-V microprocessor (des2); and (ii) a collection of synthetic benchmarks of asynchronous pipelines, developed to test tool/algorithm scalability.

3.5.1 Comparison for Asynchronous Circuits

A commercial placer acts as our comparison point. Since existing placers for asynchronous circuits are modified versions of standard synchronous placers [116–118], a commercial placer is superior when the goal is smaller wire-length and DRC-clean layout. To evaluate the placement quality, we built a standard cell library for each design that matches its gridded cell library using the approach from [99]. The commercial tool performs the placement and routing for these standard cell counterparts.

Table 3.1 summarizes our placement results for real asynchronous designs, and shows the comparison between gridded cell designs and their standard cell counterparts. “des1”

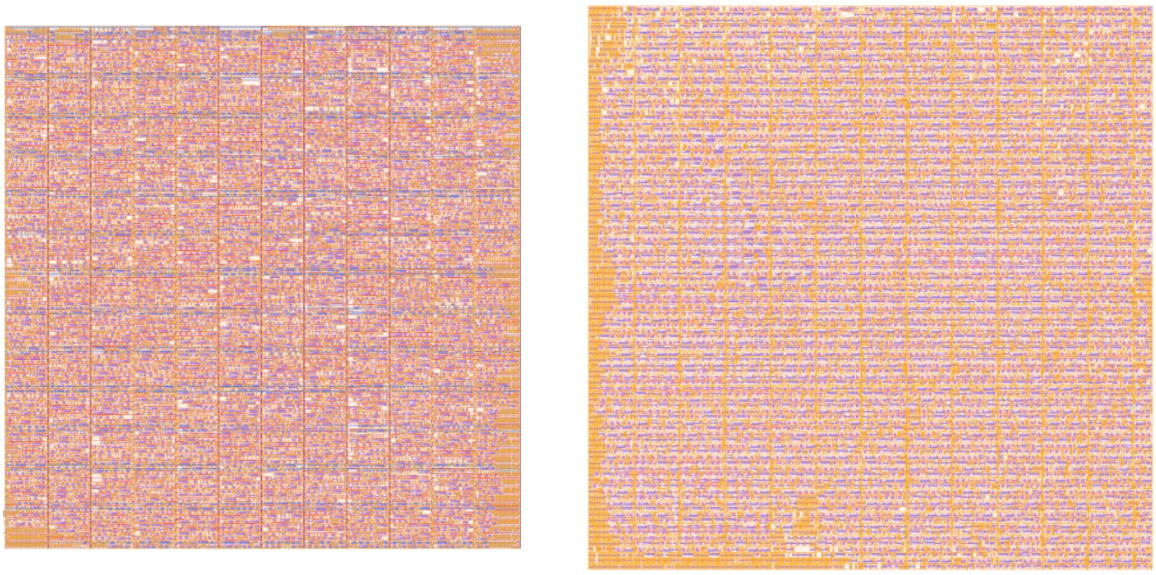
Ckt	#cells	#nets	Standard cell approach				
			cell area	die area	density	HPWL	wirelength
des1	16k	14k	134689	138756	0.97	216782	267017
des2	19k	20k	84314	86724	0.97	**228879**	**314284**
des3	81k	71k	754135	776949	0.97	1044571	1287450
des4	718k	627k	6721014	6927911	0.97	**8599981**	**10805787**
ratio	-	-	1.00	1.00	-	1.00	1.00
Ckt	#cells	#nets	Gridded cell approach				
			cell area	die area	density	HPWL	wirelength
des1	16k	14k	80451	115801	0.69	**195721**	**254938**
des2	19k	20k	51863	78792	0.66	229656	330857
des3	81k	71k	447645	643685	0.70	**980099**	**1242999**
des4	718k	627k	3990622	5744170	0.69	8832368	11169479
ratio	-	-	0.60	**0.85**	-	**0.97**	1.00

Table 3.1: Comparison between gridded cell designs and their standard cell counterparts.

corresponds to the 65nm test chip, and the core placement region of the design is shown in Fig. 3.5. Both kinds of designs use the smallest possible die area and have well tap cells in their layout. The target density of gridded cell designs is around 70%, while that of standard cell designs is 97%. While normally a lower cell density is used for standard cell designs, we used the highest feasible cell density for both approaches for comparison purposes.

As expected, since white space counts towards the cell area, the total standard cell area is much larger than the total gridded cell area. However, when we group gridded cells into clusters, the effective cell area increases when a short cell is grouped with a tall cell. The net result is that the gridded cell approach uses 15% less die area due to the new power grid design and the smaller cell size. To study the impact of the modified power distribution approach, we: (i) removed power stripes from each standard cell, and made these cells smaller accordingly; and (ii) re-ran the commercial placement and routing flow. The area gap drops from 15% to 10%, which indicates that an area reduction of 5% can be attributed to the new power grid design, and the remaining 10% comes from the more compact layout of the individual cells.

For the largest benchmark (des4), the commercial tool takes 102min to finish the routing for the standard cell placement, and 124min for the gridded cell placement, running on a single core. The average runtime of routing across the real designs is around 15% higher



(a) Gridded cell design

(b) Standard cell design

Figure 3.5: Layout of the core chip area using (a), gridded cell design methodology, (b), standard cell design methodology with standard cell height of 19 tracks. Both images use the same scale. Signal routes and I/O pins are not shown.

for gridded cells compared to standard cells, because the same collection of nets must now fit into a more compact area, and part of high metal tracks are taken by power routing. In spite of this, we did not see any pin access issues because pins that were on routing tracks in the standard cell design remain on track in the gridded cell design. The final wire-length show that the gridded cell approach requires less chip area than the standard cell approach for asynchronous designs with negligible routing quality degradation.

3.5.2 Scalability Study

To test the scalability and efficiency of our algorithms and implementation, we construct a set of synthetic benchmarks, with the number of cells ranging from 1k to 750k and a similar amount of two-pin nets. These benchmarks have similar connectivity structures and contain only local nets. Therefore, principally speaking, the optimal wire-length is proportional to the number of cells. The target density of this benchmark suite is set to 65%, and Dali runs on a single core and completes the largest benchmark in 8 min using 0.96 GB of memory.

Table 3.2 lists the detailed runtime breakdown of Dali on this benchmark suite: global

#cells	HPWL	GP(s)	LG(s)	WLG(s)	PWR(s)	Sum(s)
1k	0.5	0.21	0.01	0.01	0.04	0.27
8k	5.9	1.90	0.03	0.11	0.08	2.12
76k	86.7	22.87	0.32	1.17	1.67	26.03
151k	217.1	58.94	0.64	2.39	3.39	65.36
222k	529.0	106.57	0.97	3.68	4.78	116.00
299k	643.2	111.51	1.25	5.04	6.73	124.53
372k	958.5	220.67	1.62	6.10	8.41	236.80
450k	1112.4	200.85	1.83	7.33	9.96	219.97
525k	1362.6	255.54	2.12	8.52	11.69	277.87
598k	1758.5	467.23	2.60	9.96	13.28	493.07
677k	1962.3	599.40	2.90	11.36	14.71	628.37
742k	2787.5	409.80	3.06	12.33	16.68	441.87
Geomean ratio	0.91	0.01	0.03	0.05	1.00	

Table 3.2: Experimental results on synthetic benchmarks. The unit of HPWL is 10,000 base unit.

placement (GP), legalization (LG), well legalization (WLG), and power routing (PWR). The last row shows the geometric mean of the runtime percentage for each step. On average, global placement iterations take 91% of the runtime, and the remaining steps take 9%. Fig. 3.6 plots the runtime and HPWL against the number of cells, showing that Dali has linear scalability and consistency in the placement quality with the increase of the circuit size.

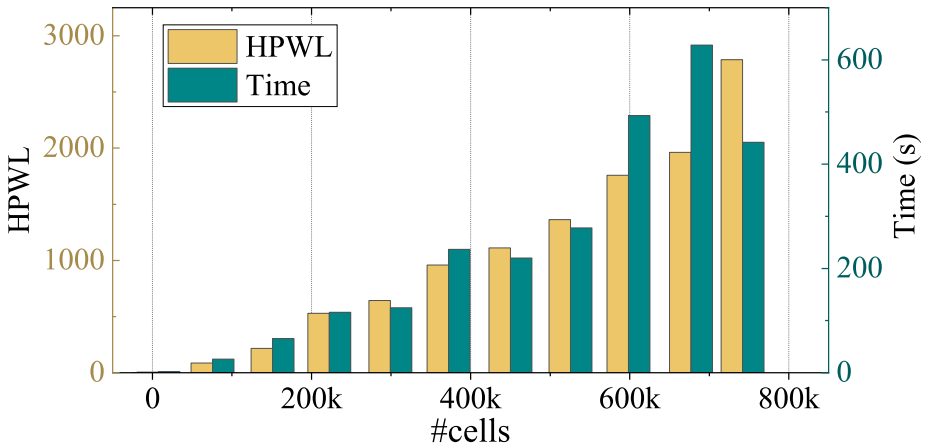


Figure 3.6: Runtime and HPWL scaling for Dali on the synthetic benchmark suite.

3.6 Summary and Future Work

This paper presents the gridded cell design methodology together with a placement flow, Dali, for asynchronous circuits. A chip tape-out verifies the correctness and effectiveness of this methodology. The placement and routing quality of a gridded cell design are comparable to its standard cell counterpart produced by a commercial tool with considerably less area. The experimental results demonstrate the scalability of our implementation.

Future works are listed as follows: first, we plan to explore additional detailed placement technique for wire-length cost improvement; second, integration with a timer for timing-driven placement; third, structure-aware placement for sub-region creation and compact datapath layout; fourth, incremental placement for buffer insertion and pipeline balancing; and finally, hierarchical placement to handle increasing design complexity.

Chapter 4

Legalization Algorithm for Multideck Gridded Cells

The previous chapter introduces the placement flow for gridded cells, and this chapter will focus on the legalization problem. We will formulate this problem as a cell displacement optimization problem. To quickly solve this problem, gridded cells are first grouped into local clusters, and then cell displacement is optimized in each cluster. This idea is further extended to support gridded cells with complex structures.

Since an asynchronous design usually contains many complex and multi-output logic gates, the physical layout of these gates may consist of several transistor stacks forming a multideck structure. These multideck cells make the legalization problem more challenging by introducing correlations between cells in adjacent clusters. In this chapter, we formulate the multideck gridded cell legalization problem as a mixed-integer quadratic programming problem and present a greedy algorithm followed by a distributed algorithm to optimize cell displacement. Experiment results demonstrate that this legalization flow is both effective and efficient.

4.1 Gridded Cell Legalization Problem Formulation

For a given list of gridded cells C and a rough placement (X^0, Y^0), the output of the legalization step is a legal placement (X, Y), which satisfies the following constraints:

- all cells are inside the placement region;
- the lower left coordinate of each cell is on the placement grid;
- all cells have no overlap with each other and placement blockages;
- N/P-well design rules are satisfied, including minimum width rule, minimum space rule, minimum area rule, and latch-up prevention rule.

The initial placement is expected to be of high-quality. To ensure the placement quality, the distance between the initial placement and the final placement acts as the objective function and needs to be optimized during this process:

$$obj(X, Y) = \sum_{c_i \in C} e_i(\|x_i - x_i^0\|^p + \|y_i - y_i^0\|^p), \tag{4.1}$$

where p is usually 1 or 2 and $e_i \geq 0$ is a scale factor for cell c_i, which is usually 1.

When $p = 1$, the objective function is the linear displacement, which is also a commonly adopted metric for measuring the quality of new placement besides wirelength metrics. When $p = 2$, the objective function is the quadratic displacement, which is a commonly adopted objective function due to its smoothness.

4.2 Single-Deck Gridded Cell

The previous chapter introduces an algorithm to legalize gridded cells, which is illustrated in Fig. 4.1 (a). This legalization algorithm starts with an illegal placement. The legalizer sorts cells according to their y-locations. Then, for each cell, if there is an existing cluster that can accommodate this cell, this cell will be grouped into this cluster; otherwise, a new cluster will be created based on the location of this cell. Since clusters are created based on the need of cells, this approach is called cell-centric legalization. This legalization algorithm can also handle standard cells by setting the standard cell height as the grid value along the y-direction and adjusting cluster orientations when necessary.

Another variant of the gridded cell legalization algorithm is the cluster-centric legalization algorithm, as illustrated in Fig. 4.1 (b). Initially, the legalizer fills the placement

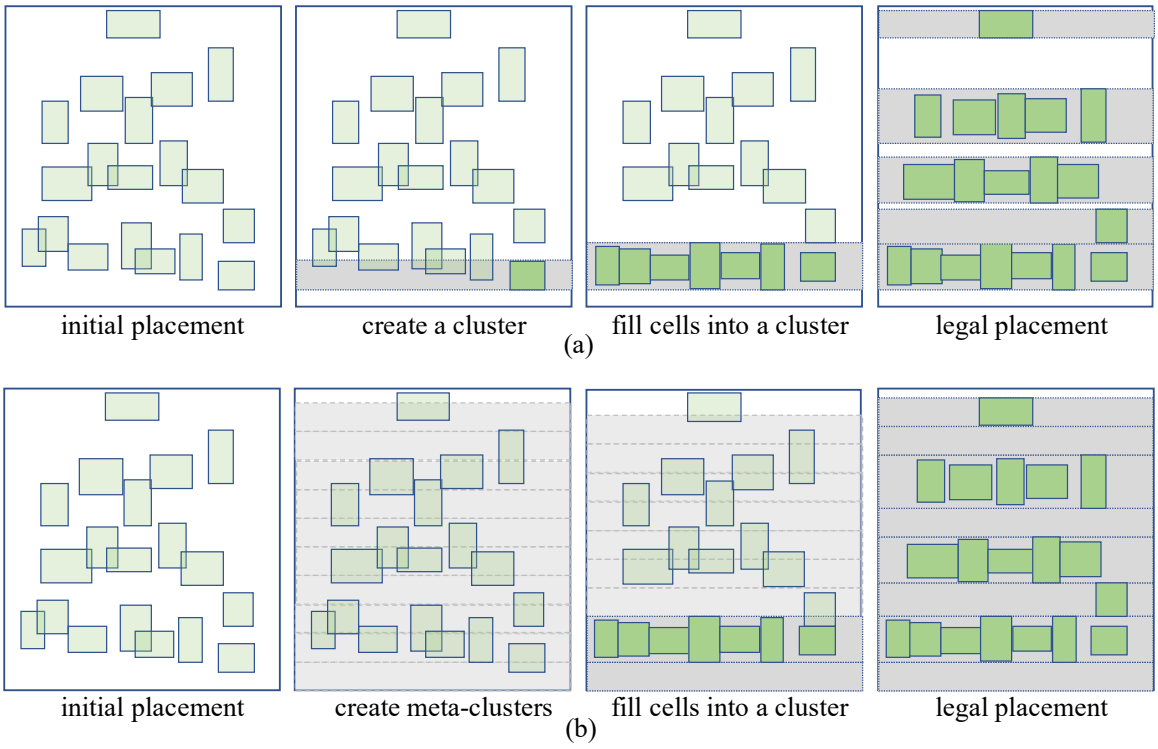


Figure 4.1: Cell-centric legalization vs. cluster-centric legalization. (a) cell-centric legalization, (b) cluster-centric legalization. Gray boxes are clusters. Legal cells are in green, and cells with illegal locations are in light green.

region with close-packed clusters. These clusters have the same height, which is the height of a well tap cell. Then, cells are sorted according to the y-coordinate. Next, for each cell, the legalizer puts this cell into the nearest cluster and updates the height of this cluster. Finally, cell overlaps are removed in each cluster. This approach ensures the abutment of clusters and can automatically handle the standard cell legalization problem. Since cells are always placed into a nearby cluster, this approach is called cluster-centric legalization. The cell-centric legalization and the cluster-centric legalization lead to similar placement results.

To obtain a high-quality placement result, we need to optimize the cell displacement during the cell legalization process. Swapping cells among clusters can potentially reduce cell displacement. However, this swapping operation can potentially change the cluster height and thus make clusters out of the placement region, especially for a high placement density. Therefore, it is better to keep cells in their original clusters and optimize cell

displacement by moving cells inside each cluster and adjusting the location of clusters if applicable.

4.2.1 Intra-Cluster Optimization

For each cluster, the cell displacement optimization problem is the following: given an ordered cell list $C = \{c_1, c_2, ..., c_n\}$ with cell widths $W = \{w_1, w_2, ..., w_n\}$ and initial locations $X^0 = \{x_1^0, x_2^0, ..., x_n^0\}$, the goal is to find a new location $X = \{x_1, x_2, ..., x_n\}$ to optimize the following displacement:

$$obj(X) = \sum_i e_i \|x_i - x_i^0\|^p, \tag{4.2}$$

where e_i is the weight of the initial location of the i-th cells. The solution also needs to satisfy the overlap constraint and placement region constraint:

$$\begin{aligned} lo &\leq x_1, \\ x_i + w_i &\leq x_{i+1}, \quad i = 1, ..., n-1, \\ x_n + w_n &\leq hi. \end{aligned} \tag{4.3}$$

In the following, we will introduce the solution for this problem when $p = 2$ and $p = 1$.

Quadratic Displacement

When $p = 2$, the problem becomes a mixed-integer quadratic programming problem. In general, it is time-consuming to solve such a quadratic program. Abacus provides a fast way to find the optimal solution: the abutment of cells changes those inequality relationships into equality relationships, turning the quadratic program into a series of linear equations [112].

For example, with equality constraints, constraints in Eqn. 4.3 become

$$\begin{aligned} lo &\leq x_1, \\ x_i + w_i &= x_{i+1}, \quad i = 1, ..., n-1, \\ x_n + w_n &\leq hi. \end{aligned} \tag{4.4}$$

These equality constraints further change the objective function from a multivariable into

a single variable function as the following:

$$\begin{aligned} obj(X) &= \sum_i e_i(x_i - x_i^0)^2 \\ &= \sum_i e_i(x_1 + w_1 + ... + w_{i-1} - x_i^0)^2 \\ &= \sum_i e_i(x_1 - s_i)^2, \end{aligned} \tag{4.5}$$

where

$$s_i = x_i^0 - \sum_{j<i} w_j. \tag{4.6}$$

To obtain the optimal solution, we only need to solve the following linear equation

$$\frac{\mathrm{d}obj(X)}{\mathrm{d}x_1} = \sum_i 2e_i(x_1 - s_i) = 0, \tag{4.7}$$

and the solution is

$$x_1 = \frac{\sum_i e_i s_i}{\sum_i e_i}. \tag{4.8}$$

With the location of the first cell and the equality constraints, we can easily compute locations for subsequent cells. The final solution needs to be rounded to the nearest placement grid and shifted into the legal range.

Linear Displacement

When $p = 1$, this problem can be formulated as a linear programming problem by introducing auxiliary variables and additional constraints. The optimization problem is

$$\min \sum_i e_i t_i, \tag{4.9}$$

with the following constraints

$$\begin{aligned} (x_i - x_i^0) &\le t_i, \quad i = 1, ..., n, \\ -(x_i - x_i^0) &\le t_i, \quad i = 1, ..., n, \\ lo &\le x_1, \end{aligned} \tag{4.10}$$

$$x_i + w_i = x_{i+1}, \quad i = 1, ..., n-1,$$
$$x_n + w_n \leq hi.$$

In general, it is time-consuming to solve this problem using a linear programming solver. There is a faster way to find the optimal solution similar to the method for quadratic displacement. When these inequality constraints become equality constraints, all cells are abutted and form a long segment, which changes the objective function to

$$\begin{aligned} obj(X) &= \sum_i e_i |x_i - x_i^0| \\ &= \sum_i e_i |x_1 + w_1 + ... + w_{i-1} - x_i^0| \\ &= \sum_i e_i |x_1 - s_i|, \end{aligned} \tag{4.11}$$

where

$$s_i = x_i^0 - \sum_{j<i} w_j. \tag{4.12}$$

It is straightforward to prove that the above objective function is convex when e_i is non-negative, and its first derivative is

$$\begin{aligned} \frac{\mathrm{d}obj(X)}{\mathrm{d}x_1} &= \sum_i e_i \mathrm{sgn}(x_1 - s_i) \\ &= \sum_{s_i \leq x_1} e_i - \sum_{s_i > x_1} e_i \\ &= 2 \sum_{s_i \leq x_1} e_i - \sum_i e_i. \end{aligned} \tag{4.13}$$

From the last formula, it is easy to see that the optimal value of the objective function is reached when x_1 takes the value in $\{s_i\}$ where the first derivative changes from negative to non-negative.

Algorithm 3 shows the implementation of the linear displacement optimization procedure. The workhorse of this procedure is the *Merge* subroutine. Given two cell segments, this subroutine merges them into one segment and computes the optimal location using the conclusion introduced above. Because each segment has a sorted (s, e) list, the time

Algorithm 3: Single cluster linear displacement optimization

Input: ordered cell list C and weight list E
Output: the location of each cell
Result: minimum total cell displacement

```
1  Create a cell segment list G;
2  foreach cell (c, e) ∈ (C, E) do
3      Create a segment g;
4      g.push(c.x, e);
5      g.width := c.width;
6      g.x := c.x;
7      Append g to G;
8      Collapse(G);
9  i := 1;
10 foreach segment g ∈ G do
11     x := g.x;
12     ns := g.size();
13     foreach j=1,...,ns do
14         C[i].x := x;
15         x := x + C[i].width;
16         i := i + 1;
17 Function Collapse(G):
18     g_{-1} := last segment in G;
19     g_{-2} := second to last segment in G;
20     if g_{-2}.x + g_{-2}.width > g_{-1}.x then
21         Merge(g_{-2}, g_{-1});
22         Pop the last segment from G;
23         Collapse(G);
24 Function Merge(g, g'):
25     foreach (s, e) ∈ g' do
26         s := s − g.width;
27     Merge (s, e) list in g' to g;
28     Traverse (s, e) list in g to get the optimal g.x;
```

complexity of merging two sorted lists is linear. Moreover, finding the optimal location requires traversal of the merge list, the time complexity of which is also linear. Therefore, the time complexity of the $Merge$ subroutine is linear. However, the cluster has N cells, and this $Merge$ subroutine can be executed up to $N-1$ times, making the worst-case time complexity of the whole procedure $O(N^2)$. The time complexity of the corresponding procedure for quadratic displacement is only $O(N)$ [112]. It is preferable to use quadratic displacement as the objective function when a cluster contains a large number of cells.

4.2.2 Inter-Cluster Optimization

Because the placement region is partitioned into several sub-regions, the location of clusters can be adjusted vertically to optimize the cell displacement along the y-direction. For clusters in each sub-region, the optimization problem can be formulated as the following: given an ordered cluster list $L = \{l_1, l_2, ..., l_n\}$ with P-well heights $P = \{p_1, p_2, ..., p_n\}$ and N-well heights $N = \{n_1, n_2, ..., n_n\}$, the goal is to find the N/P-well borderline for each cluster $Y = \{y_1, y_2, ..., y_n\}$ to optimize the following objective function:

$$\begin{aligned} obj(Y) &= \sum_{c \in C} e_c \|y_c - y_c^0\|^p \\ &= \sum_{l_i \in L} \sum_{c \in l_i} e_c \|y_c - y_c^0\|^p \\ &= \sum_{l_i \in L} \sum_{c \in l_i} e_c \|y_i - y_c^0\|^p, \end{aligned} \tag{4.14}$$

where y_c^0 is the initial N/P-well boundary of cell c. The above equations are valid because every cell is in and only in a single cluster, and cells in the same cluster share a common N/P-well boundary. For illustrative purposes, we assume the orientation of all clusters is unflipped, and thus the overlap constraint and the placement region constraint are:

$$\begin{aligned} lo &\leq y_1 - p_1, \\ y_i + n_i &\leq y_{i+1} - p_{i+1}, \quad i = 1, ..., n-1, \\ y_n + n_n &\leq hi. \end{aligned} \tag{4.15}$$

When $p = 2$, it is easy to show that

$$\sum_{c \in l_i} e_c (y_i - y_c^0)^2 = e_i (y_i - \bar{y_i}) + const, \tag{4.16}$$

where

$$e_i = \sum_{c_j \in l_i} e_{c_j}, \tag{4.17}$$

and

$$\bar{y_i} = \frac{\sum_{c_j \in l_i} e_{c_j} y^0_{c_j}}{\sum_{c_j \in l_i} e_{c_j}}. \tag{4.18}$$

These relationships change the objective function to

$$obj(Y) = \sum_i e_i (y_i - \bar{y_i})^2, \tag{4.19}$$

where constant terms are ignored because they do not affect the global optimality. This objective function can be optimized using the same method for minimizing the quadratic displacement of cells in a cluster. In addition, it is straightforward to modify the constraints to support clusters with alternating or even arbitrary orientations.

When $p = 1$, we can also modify the method for linear displacement to solve this problem. However, the time complexity becomes worse after this modification. For example, to merge two clusters into one cluster segment, we need to traverse through the sorted list of initial cell locations in these two clusters, which is more computationally expensive than merging two cells into a cell segment.

4.3 Multideck Gridded Cell

Asynchronous designs often contain complex logic gates with their physical layouts consisting of a few simple gridded cells stacking on top of one another [97]. These multideck gridded cells make the locations of cells in adjacent clusters entangled and thus largely increase the complexity of the legalization problem. Although it is possible to replace multideck cells with multiple single-deck cells and apply existing techniques to solve the placement problem, this decomposition process may lead to a layout with a large area, degraded performance, and timing issues.

Multideck Standard Cell

Fig. 4.2 (a) shows an example of the power rail alignment constraints for standard cells. For an odd-deck standard cell, its power pins can match power rails in any row via vertical cell flipping; for an even-deck standard cell, its power pins only match power rails in every

other row.

There are many legalization algorithms for designs with multideck standard cells. Wu and Chu proposed to pair single-deck cells into double-deck cells and apply conventional detailed placement techniques to solve the legalization problem [119]. However, this algorithm cannot handle multideck cells in general. Chow et al. considered the power rail alignment constraint during legalization and proposed a local adjustment algorithm [120]. MrDP uses a dynamic programming-based technique to solve double-row placement and a network flow-based technique to solve multi-row placement [121]. Wang et al. analyzed the insufficiencies of algorithms in Abacus and extended the advantages of those algorithms to handle multideck cells [122]. Hung et al. proposed a parallel algorithm using integer linear programming to optimize cell displacement, which leads to high-quality legalization results but suffers from slow execution time [110]. Some works also consider constraints in addition to power rail alignment, such as minimum implant area [123], drain-to-drain abutment [124], pin-accessibility [125], fence region [126], and technology [127]. An analytical legalizer treats the legalization problem as a linear complementarity problem, and uses a modulus-based matrix splitting iteration method to solve this problem efficiently [128].

Multideck Gridded Cell

In general, compared with single-deck gridded cells, multideck gridded cells bring the following new problems to the legalization process:

- even-deck gridded cells cannot be placed in an arbitrary cluster;
- multideck cells need to be stretched to fit into clusters;
- multideck gridded cells make adjacent clusters entangled.

Fig. 4.2 (b) shows an example of the N/P-well alignment constraints for gridded cells. Similar to multideck standard cells, for an odd-deck gridded cell, its N/P-well can match the orientation of any cluster via vertical cell flipping; for an even-deck gridded cell, its N/P-well only match the orientation of every other cluster.

Since a multideck gridded cell sits in multiple cell clusters, and different clusters may have different N/P-well heights, the legalizer needs to stretch multideck gridded cells to

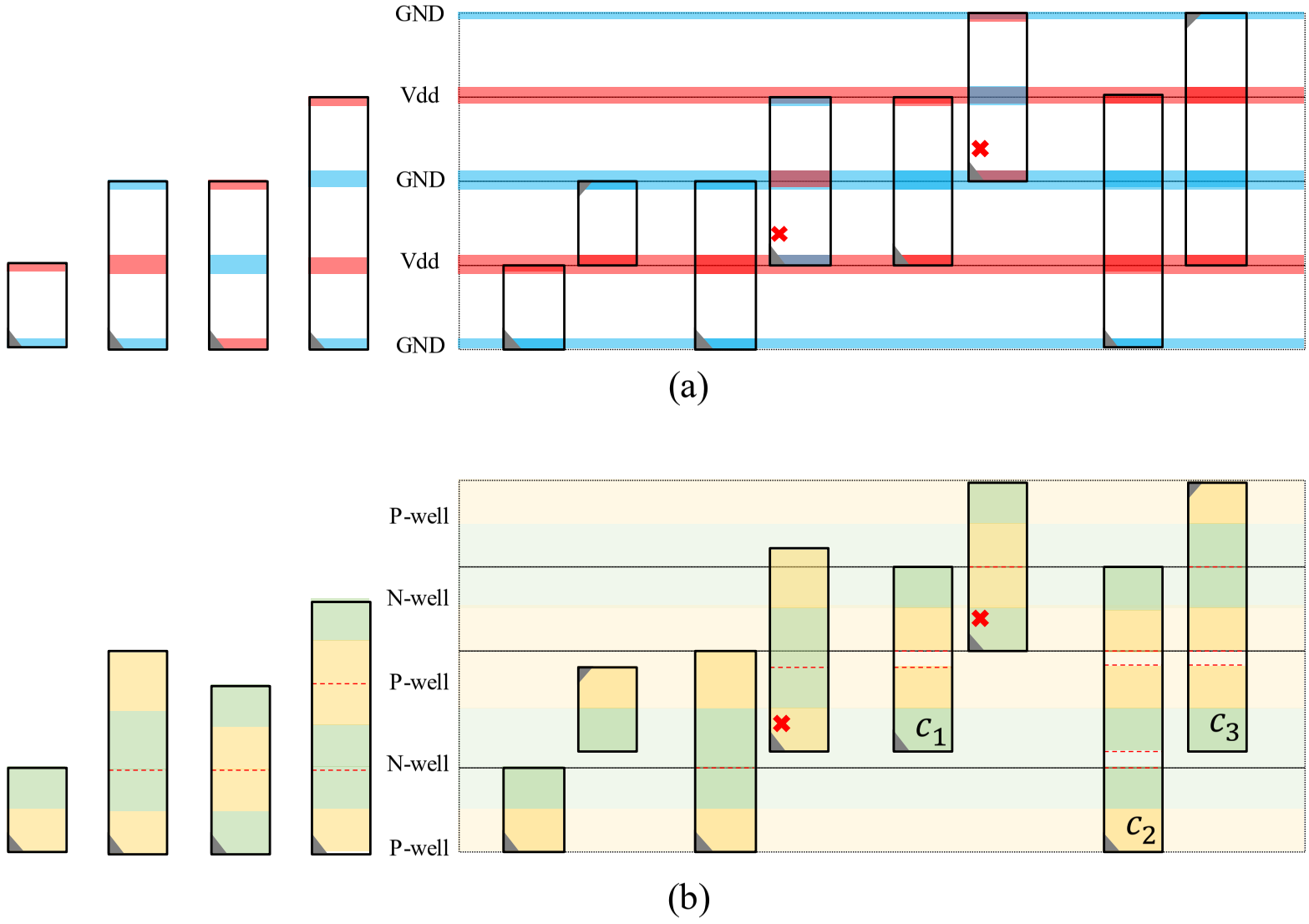


Figure 4.2: Cell alignment constraints. (a) Power rail alignment constraints for standard cells, (b) N/P-well alignment constraints for gridded cells. Cells with illegal locations are marked with a red cross symbol.

satisfy the alignment constraints when necessary. For example, initially, two sub-cells in cell c_1 are abutted vertically, the boundary of which is marked with a red dashed line. During the legalization process, the alignment constraint requires a non-zero distance between these two sub-cells, and thus cell c_1 needs to be stretched. Similarly, the legalizer also needs to stretch cell c_2 and cell c_3: although these two cells are of the same type before legalization, they are stretched differently due to their local environment. The stretching of cells also has an impact on the delay of cells.

Multideck gridded cells introduce more correlations among cells in adjacent clusters. This is because the locations of cells are coupled in the presence of multideck gridded cells. For example, when the location of cell c_2 needs a local adjustment for optimizing some metrics, the locations of other cells in the first three clusters need to be considered to avoid cell overlaps.

4.4 Multideck Gridded Cell Legalization

4.4.1 Legalization Problem Formulation

The legalization problem formulation given in Section 4.1 is a general formulation for both single-deck and multideck gridded cells. We choose the weighted quadratic displacement as the objective due to its smoothness and other properties:

$$obj(X, Y) = \sum_{c_i \in C} e_i((x_i - x_i^0)^2 + (y_i - y_i^0)^2), \tag{4.20}$$

where $e_i \geq 0$ is the weighting parameter for cell c_i.

The main difference between the standard cell legalization problem and the gridded cell legalization problem is the following:

- rows for standard cells specify the legal y-locations for cells before legalization;
- clusters for gridded cells specify the legal y-location for cells but need to be constructed during legalization.

When legal y-locations are known before legalization, cells can be legalized one by one based

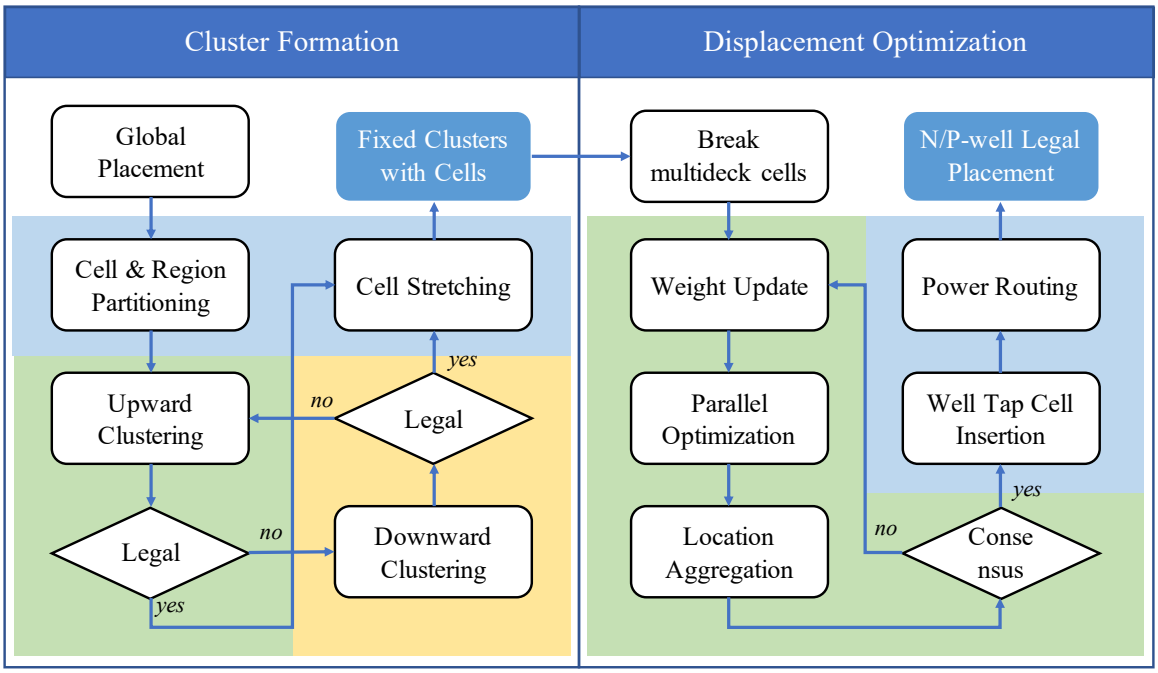


Figure 4.3: Multideck gridded cell legalization flow: cluster formation and displacement optimization.

on the order of their x-coordinates using a Tetris-like legalization algorithm; cells can also be legalized row by row based on the order of their y-coordinates using an Abacus-like legalization algorithm. For gridded cells, we have to adopt the latter approach. In this case, the gridded cell legalization problem can be viewed as a combinatorial optimization problem: the task is to divide cells into many subsets, each of which forms a cluster; the goal is to find the optimal partitioning and cell location to minimize the total displacement.

4.4.2 Legalization Flow

The multideck gridded cell legalization flow consists of two stages: cluster formation and displacement optimization, as shown in Fig. 4.3. These two stages are designed to satisfy various N/P-well design rules and optimize the quadratic cell displacement. The cluster formation step aims to maintain the relative cell order using a greedy algorithm to group cells into clusters. During this step, gridded cells are stretched to satisfy N/P-well design rules. After cell clustering, the displacement optimization step refines the quadratic displacement using a distributed algorithm.

4.4.3 Cluster Formation

This step takes the global placement result as the input and eliminates cell overlaps and N/P-well design rule violations. First, the cell list and the placement region are partitioned into sub-lists and corresponding sub-regions. Then, in each sub-region, cells are grouped into clustered using the upward-downward legalization algorithm. Finally, multideck gridded cells are stretched to ensure their N/P-well boundaries align with those boundaries in clusters.

Cell & Region Partitioning

In the presence of placement blockages, the placement region may have an irregular shape, and a large cell displacement can occur when the legalizer moves a cell across a placement blockage. To avoid large cell displacements, the placement region is divided into simple sub-regions, and cells are assigned to their nearest sub-regions. After this partitioning step, the legalizer eliminates cell overlaps and N/P-well design rule violations in each sub-region independently.

Greedy Cell Clustering

Local cell clustering is an effective way to satisfy N/P-well design rules [129]. For modern technology nodes, the area of an N/P-well in a single gridded cell is smaller than the minimum requirement. A cluster provides a large N/P-well shared among cells to satisfy the minimum area rule. This shared N/P-well also has a large width and height, avoiding the violation of the minimum width rule. Moreover, well tap cells in a cluster connect N/P-wells to the corresponding power rail, eliminating the latch-up phenomenon.

Algorithm 4 shows the implementation of the cell clustering algorithm. This algorithm combines the single-deck cell clustering technique presented in Dali [129] and the cell displacement optimization technique used in Abacus [112]. This algorithm starts with sorting cells according to their y-coordinates. Then, the legalizer creates a list of meta-clusters to fill the sub-region. The height of these clusters is initially the same as that of a well tap cell and can change during the legalization process. Next, for each cluster, the legalizer fills

Algorithm 4: Upward/downward cell clustering

Input: cell list C
Output: *status* of this iteration
Result: placement with less cell overlaps

```
1  Sort cell list C by y-coordinate in ascending order;
2  status := True;
3  N_c := C.size();
4  Initialize the number of processed cells N_p := 0;
5  repeat
6      Initialize a front cluster f;
7      Initialize a legalized list G;
8      Initialize a skipped list K;
9      foreach i = N_p, ..., N_c − 1 do
10         if !f.hasOverlap(C[i]) then
11             break;
12         if f.hasMatchOrient(C[i]) and f.hasSpace(C[i]) then
13             f.place(C[i]);
14             G.append(C[i]);
15         else
16             K.append(C[i]);
17     foreach i = 0, ..., G.size() − 1 do
18         C[i + N_p] := G[i];
19     N_p := N_p + G.size();
20     foreach i = 0, ..., K.size() − 1 do
21         C[i + N_p] := K[i];
22     Update the N/P-well height of the front cluster;
23     Stretch cells in the front cluster;
24     Minimize the x quadratic displacement of the front cluster;
25 until N_p = N_c;
```

nearby gridded cells into this cluster. These gridded cells need to satisfy the cell-cluster overlap criterion and match the orientation of this cluster. During this filling process, the height of this cluster is updated to fit all gridded cells in it. Finally, the quadratic displacement of cells in this cluster is optimized, and multideck gridded cells will be treated as fixed cells for subsequent clusters. These steps are repeated until all cells are processed. An extension of this greedy legalization is that the legalizer only fills a cell into a cluster when this operation does not lead to a large cell displacement.

Fig. 4.4 shows an example of the cluster formation process for a design containing multideck gridded cells. As shown in Fig. 4.4 (a), cells in the global placement overlap with each other. Fig. 4.4 (b) shows the legal placement result using the upward-downward

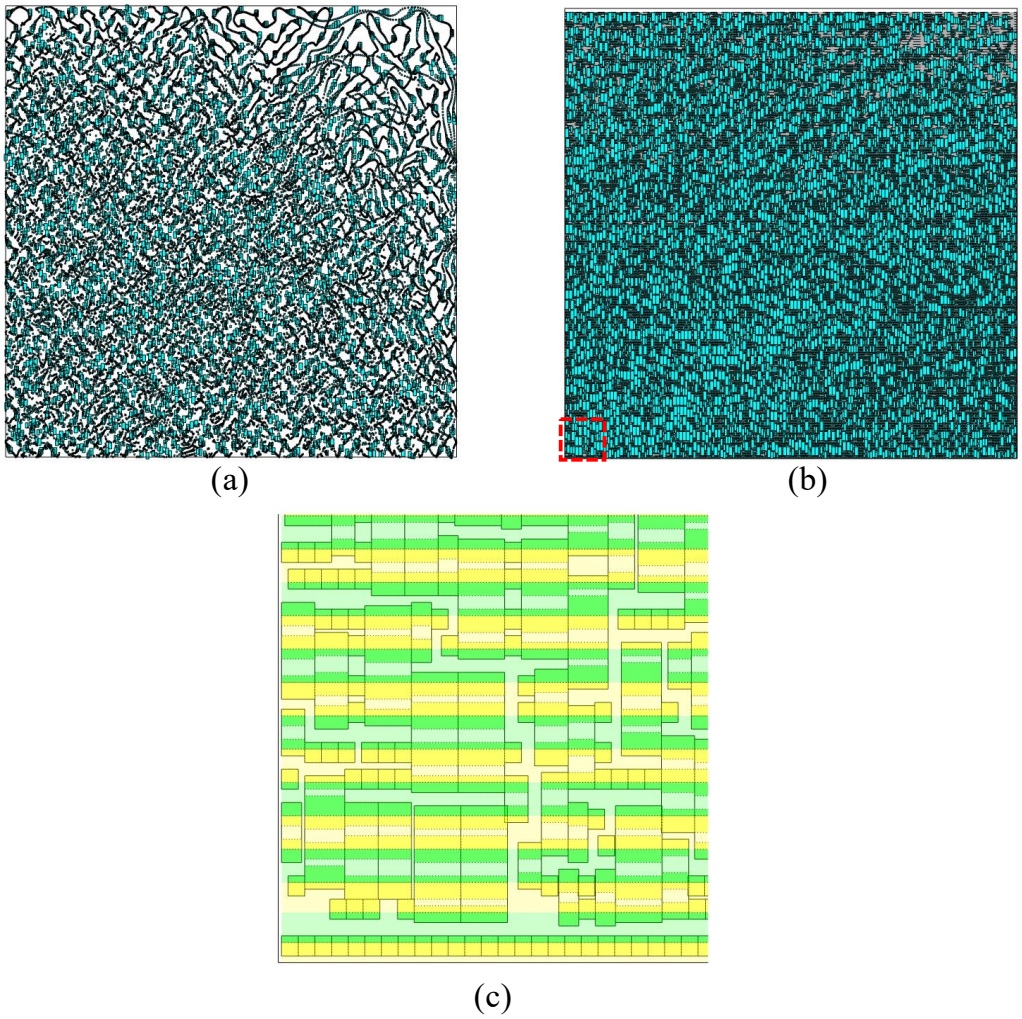


Figure 4.4: An example of cluster formation for multideck gridded cells. (a) global placement, (b) legal placement via cluster formation, (c) N/P-well alignment and cell stretching in a local region.

legalization, which eliminates cell overlaps and N/P-well design rule violations via cluster formation. As can be seen from Fig. 4.4 (c), gridded cells are stretched so that they can share the same N/P-well in clusters.

4.4.4 Displacement Optimization

This cluster formation step aims to optimize the cell displacement by preserving the cell order, which is also sufficient to generate a legal placement result. However, this greedy algorithm may not lead to the globally optimal solution, especially when the placement

density is high. Although it is possible to improve cell displacement by swapping cells among clusters, this can potentially change cluster heights, making clusters out of the sub-region. Note it is always safe to improve the cell displacement along the x-direction, and the objective function is the following:

$$\begin{aligned} obj(X) &= \sum_{c_i \in C} e_i (x_i - x_i^0)^2 \\ &= \sum_{l \in L} \sum_{c_i \in l} \frac{e_i}{h_i} (x_i - x_i^0)^2, \end{aligned} \tag{4.21}$$

where C is the list of cells, L is the list of clusters, and h_i is the number of clusters that contain a part of cell c_i. The solution must satisfy the constraint that cells have no overlap with each other and are located in a cluster. For example, for a given cluster l and the list of sorted cells $C_l = \{c_1, c_2, ..., c_n\}$ with cell widths $W_l = \{w_1, w_2, ..., w_n\}$, the constraints are the following:

$$\begin{aligned} lo &\leq x_1 \\ x_i + w_i &\leq x_{i+1}, \quad i = 1, ..., n-1 \\ x_n + w_n &\leq hi, \end{aligned} \tag{4.22}$$

where $[lo, hi]$ is the available space in the cluster l. When all cells are single-deck cells, each cluster has an independent set of constraints. However, the presence of multideck gridded cells makes constraints for different clusters entangled.

As can be seen, this is a typical quadratic programming problem, and thus we can use a quadratic programming solver to find the solution. However, this process is time-consuming in general. In this work, we propose a distributed algorithm to quickly optimize the cell displacement. This algorithm decouples clusters by introducing auxiliary variables, making it possible to optimize the displacement of cells in each cluster independently.

4.4.5 Problem Reformulation

The core idea of the distributed algorithm is to make every cluster an open system and allow them to exchange information about shared multideck gridded cells. This algorithm makes use of the theorem described below.

Optimal Anchor

Theorem 1 (Optimal anchor theorem)**.** *Let X^{opt} be the global minimum point of the function*

$$f(X) = \sum_{i=1}^{n} e_i(x_i - x_i^0)^2$$

subject to a set of constraints S, where $X = \{x_1, x_2, ..., x_n\}$ and $\forall i$, $e_i \geq 0$, $x_i^0 \in \mathbb{R}$, then X^{opt} is also the global minimum point of the following function subject to the same set of constraints:

$$g(X) = \sum_{i=1}^{n} [e_i(x_i - x_i^0)^2 + a_i(x_i - x_i^{opt})^2],$$

where $X^{opt} = \{x_1^{opt}, x_2^{opt}, ..., x_n^{opt}\}$ and $\forall i$, $a_i \geq 0$.

Proof. If the global minimum point X^{opt} exists, then X^{opt} satisfies all constraints in S. Moreover, because a_i is non-negative, $a_i(x_i - x_i^{opt})^2$ reaches its global minimum when $x_i = x_i^{opt}$ for all i. This means X^{opt} is the global minimum point of the function

$$h(X) = \sum_{i=1}^{n} a_i(x_i - x_i^{opt})^2$$

subject to constraints S. Since X^{opt} is the global minimum point of function f, it is obvious that X^{opt} is also the global minimum point of any function in the set $F = \{q(X) | q(X) = af(X) + bh(X), \forall a, b \geq 0\}$ subject to constraints S. Because $g \in F$, X^{opt} is the global minimum point of $g(X)$. □

This theorem can be generalized to cover the linear displacement function and other function families.

Multideck Cell Decomposition

Multideck gridded cells can be decomposed into simple cells. Since a multideck gridded cell is shared among multiple clusters, it can be viewed as multiple single-deck gridded cells with additional location constraints: sub-cells belonging to the same multideck cell must share the same x-coordinate. If we denote $x_{i,l}$ as the location of the sub-cell of c_i in cluster

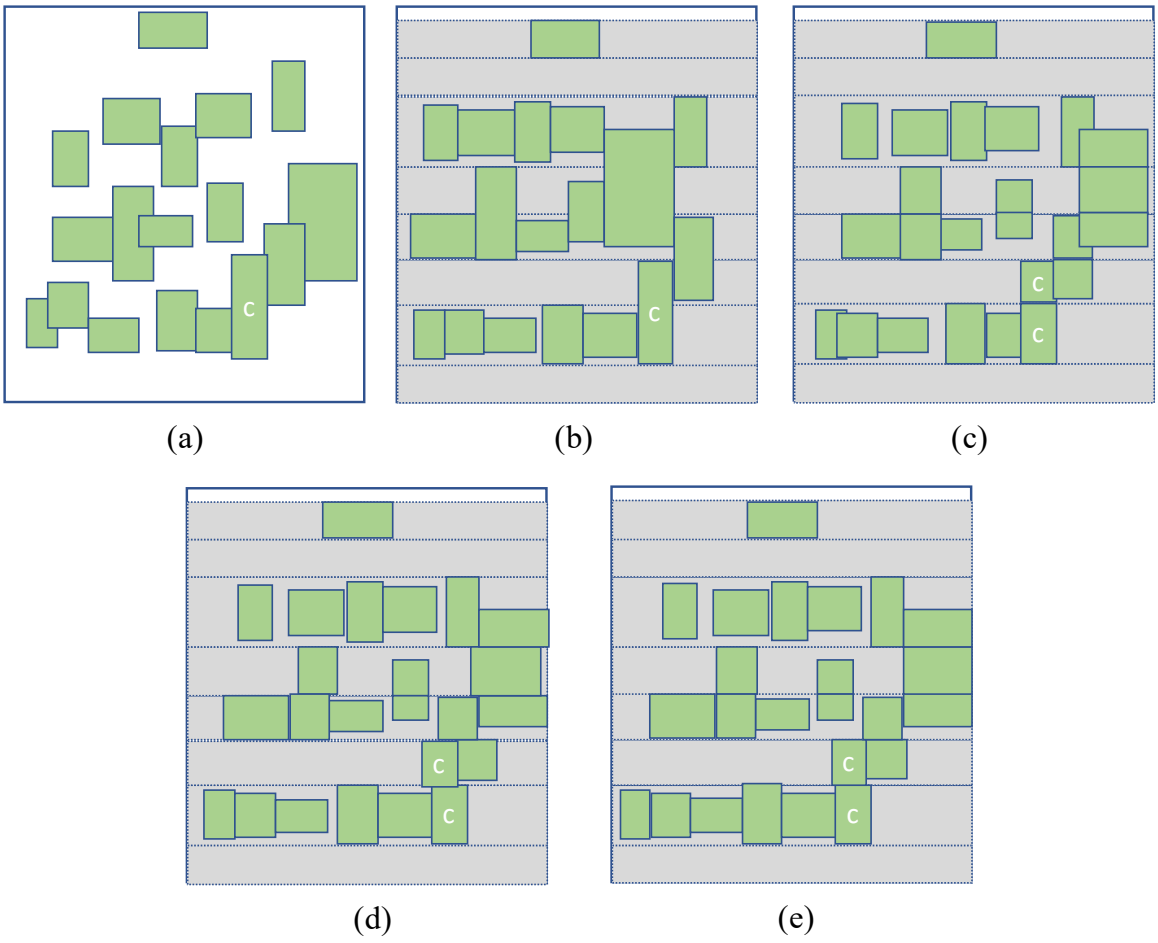


Figure 4.5: Breaking multideck gridded cell into sub-cells. (a) global placement, (b) cluster formation, (c) cell breaking, (d) sub-cell displacement optimization with no discrepancy penalty, (e) sub-cell displacement optimization with discrepancy penalty. N/P-wells are not shown, and multideck gridded cells are not stretched.

l, we can rewrite the objective function in Eqn. 4.21 as

$$obj(X) = \sum_{l \in L} \sum_{c_i \in l} \frac{e_i}{h_i} (x_{i,l} - x_i^0)^2, \tag{4.23}$$

where X represents the location of all sub-cells, and additional constraints on sub-cell locations are:

$$\forall c_i \in C, \forall l \ni c_i, x_{i,l} = x_i. \tag{4.24}$$

Fig.4.5 shows the process of cell decomposition. The cluster formation step groups cells into clusters to eliminate cell overlaps and N/P-well design rule violations, as shown

in Fig.4.5 (a) and (b). Because it is preferred to maintain the initial cell order, the x coordinate cells are restored after cluster formation, and then multideck gridded cells are split into sub-cells. As can be seen from Fig.4.5 (c), sub-cells in each cluster honor the cell order in the global placement result.

Distributed optimization

If we apply theorem 1 to the objective function for sub-cells in Eqn. 4.23, we can get a new objective function:

$$obj(X) = \sum_{l \in L} \sum_{c_i \in l} \left[w_i (x_{i,l} - x_i^0)^2 + a_{i,l} (x_{i,l} - x_i^{opt})^2 \right], \tag{4.25}$$

where $w_i = e_i / h_i$ and $a_{i,l} \geq 0$. The goal is to minimize the above objective function to get the optimal location X^{opt}. Since X^{opt} is also a part of the objective function, we can use an iterative scheme to find the solution.

For a given trial solution $X^a = \{x_1^a, x_1^a, ..., x_{n-1}^a\}$, the objective function for a cluster l becomes

$$\begin{aligned} obj_l(X) &= \sum_{c_i \in l} \left[w_i (x_{i,l} - x_i^0)^2 + a_{i,l} (x_{i,l} - x_i^a)^2 \right] \\ &= \sum_{c_i \in l} (w_i + a_i)(x_{i,l} - \frac{w_i x_i^0 + a_{i,l} x_i^a}{w_i + a_{i,l}})^2 + const, \end{aligned} \tag{4.26}$$

where the constant term can be ignored because it does not change the optimal solution, and the solution needs to respect the overlap constraint in each cluster.

As can be seen, this problem is essentially the same as the problem described in Eqn. 4.2, and thus we can use the same approach to optimize this objective function. Fig.4.5 (d) shows an example of sub-cell locations with the rough placement result as the trial solution and all $a_{i,l}$ being 0 for simplicity.

With these sub-cell locations, a new trial solution can be constructed from the aggregation sub-cell locations. For example, the new location for cell c in Fig.4.5 (d) can be the weighted average of sub-cell locations:

$$x_c = \frac{\sum_l \bar{w}_{c,l} x_{c,l}}{\bar{w}_{c,l}}, \tag{4.27}$$

where $\bar{w}_{c,l}$ is the weight of the cell segment that contains this sub-cell. The intuition behind this choice is that cells in a segment move together: the cost of moving a cell in this segment is effectively the same as that of moving this whole segment when the moving distance is small. Fig. 4.6 shows an example to illustrate this aggregation process.

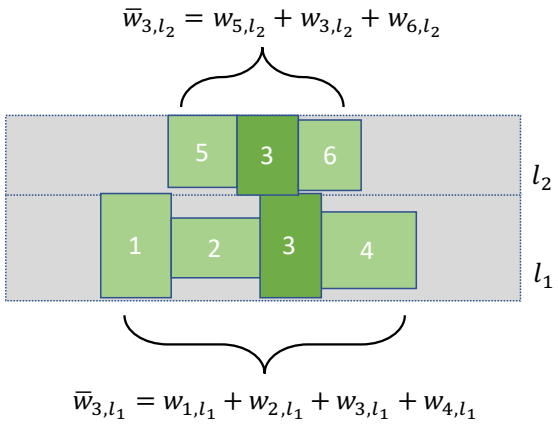


Figure 4.6: Sub-cell location aggregation. The weight of a sub-cell during aggregation is the total weight of the segment containing this sub-cell.

It is easy to show that when $a_{i,l}$ in the objective function approaches infinity, the final solution can satisfy the additional location constraints in Eqn. 4.24. Because the iterative scheme is used to find the optimal solution, we can increase $a_{i,l}$ with the number of iterations. Moreover, notice that the cell displacement in the objective function is the only term that can violate additional location constraints. Therefore, instead of making $a_{i,l}$ very big in late iterations, we can gradually diminish the cell displacement term to get the same result by adding a decay factor like the following:

$$X^{t+1} = \text{argmin}_X \sum_{c_i \in l} \left[e^{-t/\tau} w_i (x_{i,l} - x_i^0)^2 + (1 - e^{-t/\tau})(x_{i,l} - x_i^t)^2 \right], \tag{4.28}$$

where t is the current iteration number and τ is a decay factor. The second term in the above objective function measures the distance between the sub-cell locations and average locations. We call this term the cell discrepancy. Fig.4.5 (e) shows an example of sub-cell locations after a few iterations. The stopping criterion of the iterative scheme is that the objective function is close to zero, indicating clusters reach a consensus on sub-cell locations.

4.4.6 Adaptive Weight

The weight decay itself cannot guarantee the system to reach a consensus state. Fig. 4.7 (a)-(c) show such an example. It is expected that the total cell displacement keeps increasing, while the total sub-cell location discrepancy keeps decreasing. However, the sub-cell discrepancy cannot converge to zero after many iterations, as can be seen from Fig. 4.7 (a). Fig. 4.7 (b) shows the result of this non-zero discrepancy: multideck gridded cells may overlap with each other. The reason for the non-zero discrepancy is that sub-cells in a row can prevent each other from getting into their corresponding target locations, as shown in Fig. 4.7 (c). A red arrow indicates the difference between the current location of a sub-cell and its target location.

Among many sub-cells in Fig. 4.7 (c), those two sub-cells connected by a black double-headed arrow have an equal but opposite discrepancy, preventing the whole system from satisfying the additional location constraints. A possible solution is to use a legalizer to remove overlaps among multideck gridded cells after each iteration. However, this kind of procedure needs a centralized control system to resolve conflicts, which has to be a sequential program.

If we carefully examine those two sub-cells, we can find that it is impossible to move them closer to their target locations without violating the overlap constraint. If these sub-cells are forbidden to move toward their target locations, moving target locations toward these sub-cells is the only choice. Because the target location is the weighted average of all sub-cells, increasing the weight of a sub-cell with a large discrepancy can make the average location closer to this sub-cell. Therefore, it is necessary to add an adaptive weight to the objective function like the following:

$$X^{t+1} = \text{argmin}_X \sum_{c_i \in l} \left[e^{-t/\tau} w_i (x_{i,l} - x_i^0)^2 + (1 - e^{-t/\tau}) a_{i,l}^t (x_{i,l} - x_i^t)^2 \right], \tag{4.29}$$

where $a_{i,l}^t$ is 1 when the sub-cell is at its target location, and greater than 1 when away from its target location. Experimental results show that the following two forms of adaptive

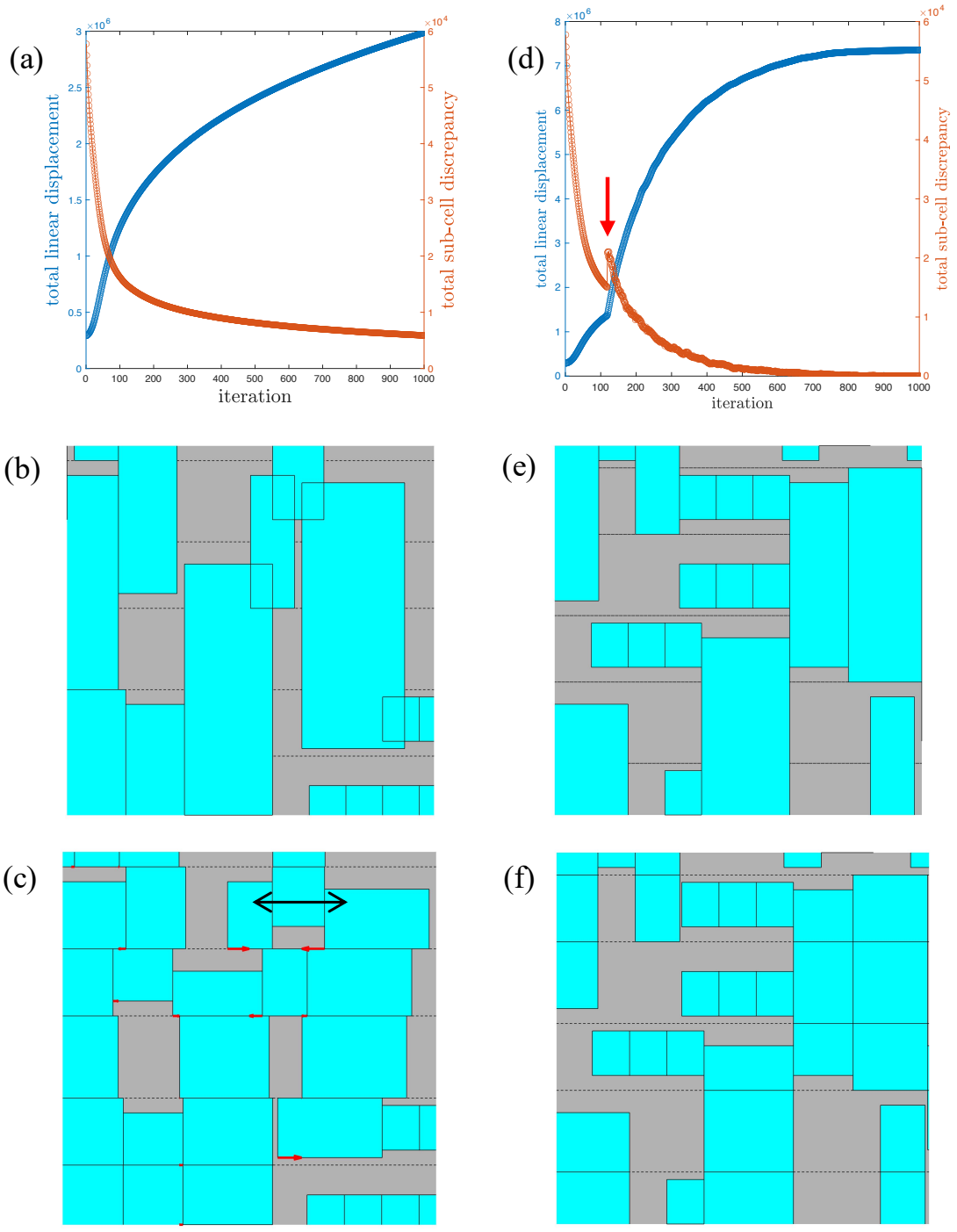


Figure 4.7: An example of placement with and without adaptive weights. (a) cell displacement and discrepancy before adding adaptive weights, (b) cells still overlap with each other after the last iteration, (c) sub-cells prevent each other from getting to their target locations, (d) cell displacement and discrepancy after adding adaptive weights, (e) cells have no overlap with each other after the discrepancy converges to zero, (f) sub-cells are at their target locations after the convergence.

weight can effectively speed up the convergence of the sub-cell discrepancy:

$$a_{i,l}^t = \left(1 + \frac{|x_{i,l}^t - x_i^t|}{\text{ave}_{j \in l}(|x_{j,l}^t - x_j^t|) + \epsilon}\right)^{\alpha}, \tag{4.30}$$

where α is a positive number no less than 2, or

$$a_{i,l}^t = \exp\left(\frac{|x_{i,l}^t - x_i^t|}{\text{ave}_{j \in l}(|x_{j,l}^t - x_j^t|) + \epsilon}\right), \tag{4.31}$$

where $x_{i,l}^t$ is the initial location of sub-cell $c_{i,l}$ at the beginning of the t-th iteration, and ϵ is a small positive number for numerical stability.

The red arrow in Fig. 4.7 (d) indicates the introduction of adaptive weights to the system. With adaptive weights, the sub-cell discrepancy converges to zero quickly. Fig. 4.7 (e) and (f) show the placement of cells and sub-cells in a local placement region after the convergence.

4.4.7 Cell Reordering

The above techniques optimize the cell displacement without changing the order of sub-cells in each cluster. The cell order remains the same as that in the global placement result. It is preferred to keep the cell order unchanged for two reasons: first, this can narrow down the search space and thus speed up the legalization process; second, maintaining the relative cell order is important for keeping wirelength from increasing. However, multideck cells can influence the location of cells in other clusters, and thus this heuristic can lead to large cell displacements, especially in dense placement regions.

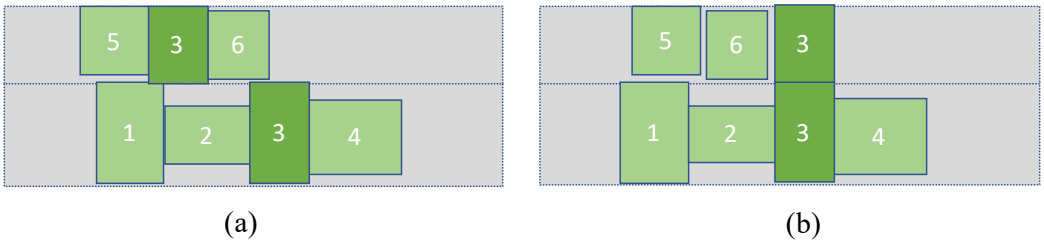


Figure 4.8: An example of cell reordering. (a) before cell reordering, (b) after cell reordering.

Fig. 4.8 shows an example of cell reordering. Before reordering, cells $\{3, 5, 6\}$ form a cell

segment. It is easy to see that cell 5 and cell 6 tend to move toward the space occupied by cell 3. In this case, local cell reordering may improve the cell displacement and sub-cell discrepancy. However, the number of permutations of all sub-cells in a cluster can be huge. To tackle this problem, we can use a sliding window to change the order of a few cells at a time. Another method is to use the cell locations after each iteration to update the cell order. With these techniques, the displacement optimization flow is the following:

- compute the scale factor for the cell displacement term and the sub-cell discrepancy term in the objective function;
- for each cluster, sort sub-cells based on their target locations, optimize the objective function, and store sub-cell locations to the corresponding multideck cell;
- for each cell, compute the weighted average location, and set it as the new cell location;
- repeat the above steps until the sub-cell discrepancy converges to zero.

Because there is no dependency among clusters, the objective function can be optimized in parallel. Cell aggregation can also be done in parallel for a similar reason.

4.5 Experimental Results

We implement the multideck gridded cell legalization algorithm in C++17, and compile it using GCC 9.3.0 on a Linux machine with 32 GB memory and Intel Core i7-8750H CPU running at 3.95 GHz. This legalizer uses one CPU core to perform experiments and benchmark runs. A commercial quadratic programming solver, CPLEX, acts as the baseline. Since there are no public asynchronous circuit benchmarks, we report results from a collection of synthetic benchmarks developed to test the tool/algorithm scalability. These benchmarks consist of 1X, 2X, 3X, and 4X multideck gridded cells. The percentage of single-deck gridded cells is around 89%, and that of other heights is around 3.6%. The number of cells in these benchmarks ranges from ten thousand to one million. Moreover, each design has three different die areas to test the legalizer under different cell densities. The global placement results are generated using the gridded cell placer, Dali [129].

ckt	#cells	density/%	greedy				CPLEX			iterative		
			t/s	ave disp x	ave disp y	HPWL	t/s	ave disp x	HPWL	t/s	ave disp x	HPWL
des1	12k	83.15	0.02	33.59	11.85	6.96e5	0.85	20.78	4.21e5	0.07	15.27	4.20e5
		65.55	0.02	18.98	9.86	4.96e5	0.76	13.54	3.78e5	0.06	10.83	3.80e5
		37.84	0.01	6.28	5.44	3.96e5	0.58	5.06	3.77e5	0.04	4.71	3.76e5
des2	28k	83.39	0.06	37.29	14.71	1.76e6	2.82	19.90	1.01e6	0.17	15.07	1.01e6
		65.30	0.05	19.82	8.70	1.23e6	2.47	13.42	9.25e5	0.14	10.49	9.16e5
		38.75	0.04	7.84	6.64	9.39e5	2.15	6.07	8.68e5	0.11	5.59	8.66e5
des3	74k	83.45	0.12	30.05	12.93	4.55e6	11.85	17.84	3.09e6	0.50	13.41	3.01e6
		65.76	0.14	21.99	10.91	3.69e6	11.44	13.43	2.67e6	0.47	10.66	2.63e6
		38.84	0.13	7.10	6.32	2.76e6	9.28	5.49	2.59e6	0.31	5.08	2.58e6
des4	112k	78.00	0.21	31.14	12.44	6.79e6	22.63	17.25	4.38e6	0.81	13.21	4.29e6
		65.97	0.18	19.98	8.97	5.49e6	20.81	12.32	4.20e6	0.75	9.94	4.15e6
		38.87	0.18	6.75	6.14	4.30e6	17.80	5.15	4.04e6	0.48	4.78	4.03e6
des5	261k	77.93	0.53	29.83	12.65	1.65e7	80.66	16.33	1.10e7	2.03	12.56	1.08e7
		66.50	0.47	20.34	9.88	1.42e7	74.73	12.38	1.09e7	2.11	9.96	1.07e7
		38.98	0.43	6.15	6.20	1.30e7	65.77	4.41	1.24e7	1.51	4.30	1.24e7
des6	507k	75.91	1.00	25.07	11.23	3.22e7	216.53	14.21	2.37e7	4.49	11.24	2.32e7
		66.47	0.97	20.80	10.26	3.03e7	217.27	12.25	2.26e7	4.58	9.93	2.32e7
		39.03	0.74	6.05	6.08	2.63e7	180.27	4.23	2.53e7	2.93	4.15	2.52e7
des7	1147k	78.38	2.49	30.28	13.12	9.27e7	780.84	15.70	6.92e7	14.33	12.27	6.73e7
		66.41	2.23	19.52	10.01	7.52e7	712.84	11.31	6.18e7	12.39	9.27	6.09e7
		38.95	1.97	5.36	6.15	6.73e7	582.09	3.90	6.54e7	7.12	3.88	6.53e7
geo-mean		-	0.21	15.90	-	6.36e6	23.54	10.20	4.98e6	0.77	8.52	4.94e6
ratio		-	**0.27**	1.87	-	1.29	30.42	1.20	1.01	1.00	**1.00**	**1.00**

Table 4.1: Results in synthetic benchmarks. "ave disp x/y" is the average linear displacement along the x/y-direction. Cell displacement and HPWL have the same base unit. The grid value is 2.4 base units. Average cell width is 13 base unit.

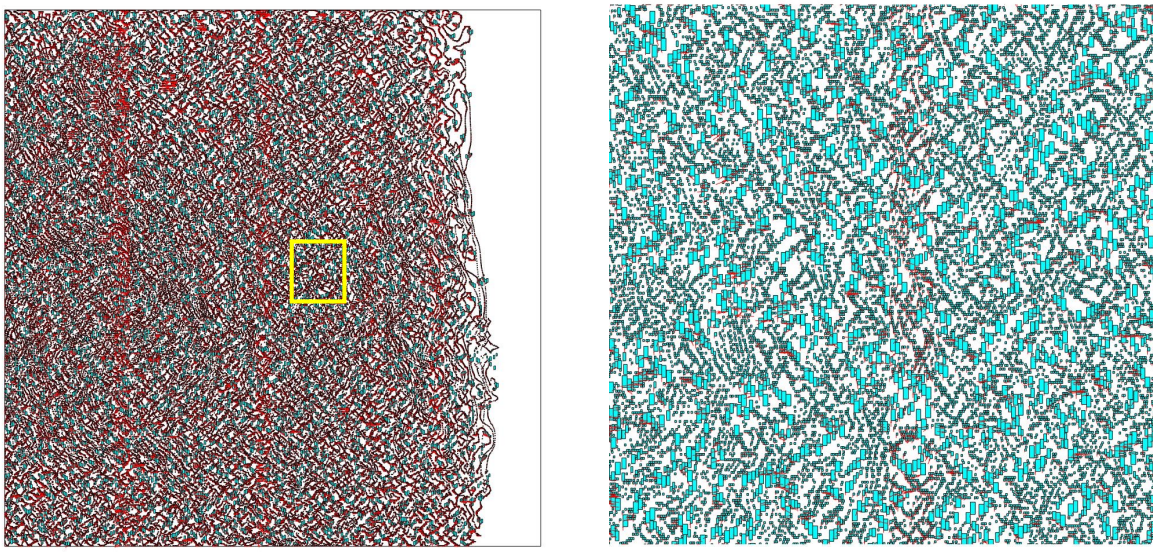


Figure 4.9: (a) legalization result of benchmark des4 with low placement density, (b) local region marked in (a). Cells are in cyan, and displacement is in red.

Although this work focuses on optimizing the quadratic cell displacement, this section uses the linear cell displacement as a metric. Table 4.1 shows the performance of the greedy cell clustering algorithm, CPLEX, and the iterative algorithm. There is no detailed placement step to adjust the location and orientation of cells after the legalization step.

As can be seen from the experimental results, although the iterative algorithm is 3.7X slower than the greedy legalization algorithm, it leads to 87% less cell displacement and 29% less HPWL. Moreover, this algorithm only takes around ten seconds to legalize a design with more than one million cells, while the time for global placement is around ten minutes. The iterative algorithm is 30X faster than the CPLEX approach, and meanwhile, it gives 20% less cell displacement and slightly better HPWL. Fig. 4.9 (a) and (b) show the legalized result of the benchmark des4 using the iterative algorithm.

Fig. 4.10 displays the CPU time v.s. the circuit size. The empirical time complexity for the iterative algorithm is $O(N^{1.17})$ from the curve fitting. As a comparison, the empirical time complexity is $O(N^{1.06})$ for the greedy algorithm, and $O(N^{1.50})$ for CPLEX.

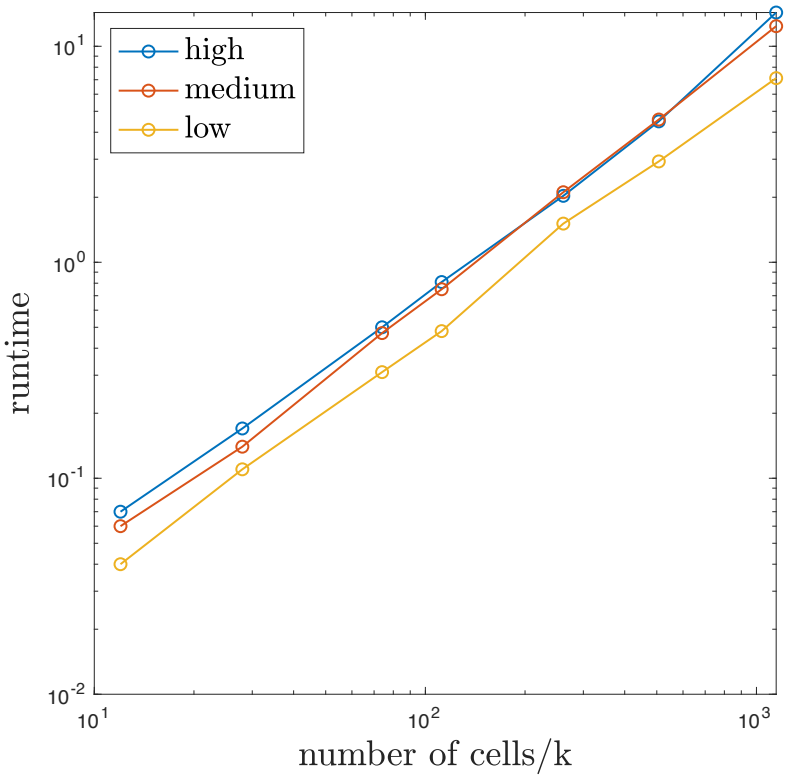


Figure 4.10: Average-case time complexity of the iterative algorithm.

4.6 Summary and Future Work

This chapter introduces a legalization algorithm for asynchronous designs containing multi-deck gridded cells. The legalizer first organizes gridded cells into clusters and then optimizes cell displacement using a fast iterative algorithm. The experimental results show the effectiveness of this algorithm and the efficiency of our implementation.

Future works are the following: first, we want to refine and extend this algorithm to support standard cell designs; second, since this algorithm is very efficient, it is promising to adopt this framework for wire length optimization.

Chapter 5

Future Work

5.1 Timing-Driven Placement for Asynchronous Circuits

The gridded cell methodology and its automation flow are capable of generating a compact physical layout for asynchronous circuits. The timing-driven placement flow is crucial for the correctness of the physical design, especially for circuit families with non-trivial timing constraints.

5.1.1 Timing Constraints

In general, timing constraints are needed to simplify the circuit design and ensure circuit performance. Since a circuit consists of components and communication channels, its functionality is realized via data exchange among components.

As introduced in Chapter 2, dual-rail channels use completion detection to ensure correct data capturing, while bundled data channels use timing constraints to get rid of complete detection circuitries. Fig. 5.1 shows such a bundled data design. Take the stage in the middle as an example. The signal req_1 indicates the validity of the output data, and when this signal changes from low to high, the data must have been ready. To be more precise, when the request signal req_0 changes from low to high, we know the output data will eventually be stable before the request signal req_1 changes from low to high.

Fig. 5.2 (a) shows a more concrete example. The *start* signal corresponds to the req_0 signal, and the *validity* signal corresponds to the req_1 signal. The correctness constraint

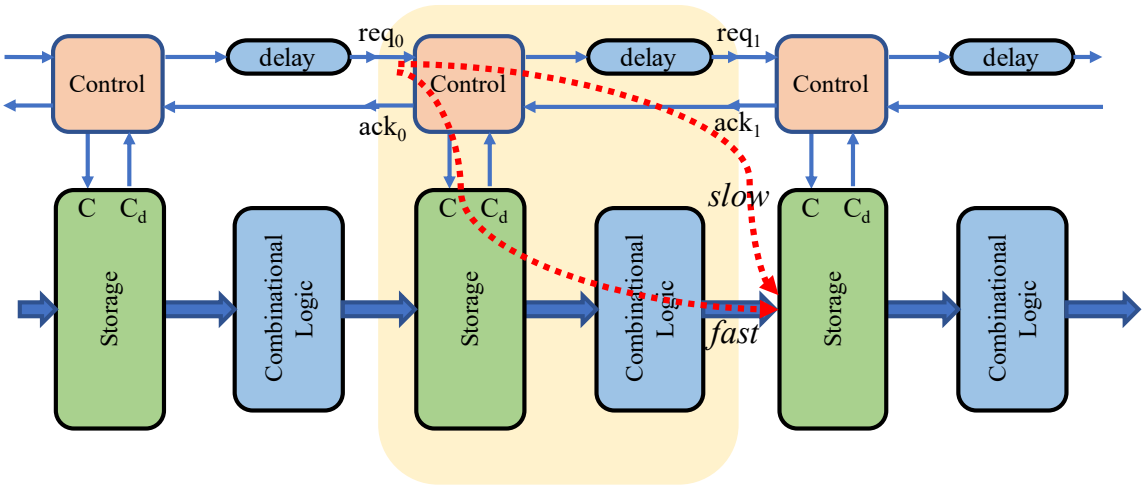


Figure 5.1: Timing constraints in a bundled data design.

can be written as [82, 130]

$$start+: \quad data+ \prec validity+, \tag{5.1}$$

which means when the *start* signal changes from low to high, the transition of the *data* signal from low to high must happen before the transition of the *validity* signal from low to high. Similarly, we can write the correctness constraint for *data* signal transition from high to low:

$$start+: \quad data- \prec validity+. \tag{5.2}$$

The above two constraints must be satisfied to ensure the correctness of the physical layout. Fig. 5.2 (b) shows all possible paths of this stage: the red path and the cyan path must be faster than the black path to satisfy the correctness constraints.

The delay of this stage is determined by the black path, and thus the performance constraint can be written as

$$delay(start+, validity+) \leq target, \tag{5.3}$$

which means the black path must be faster than this *target* delay to achieve the expected performance.

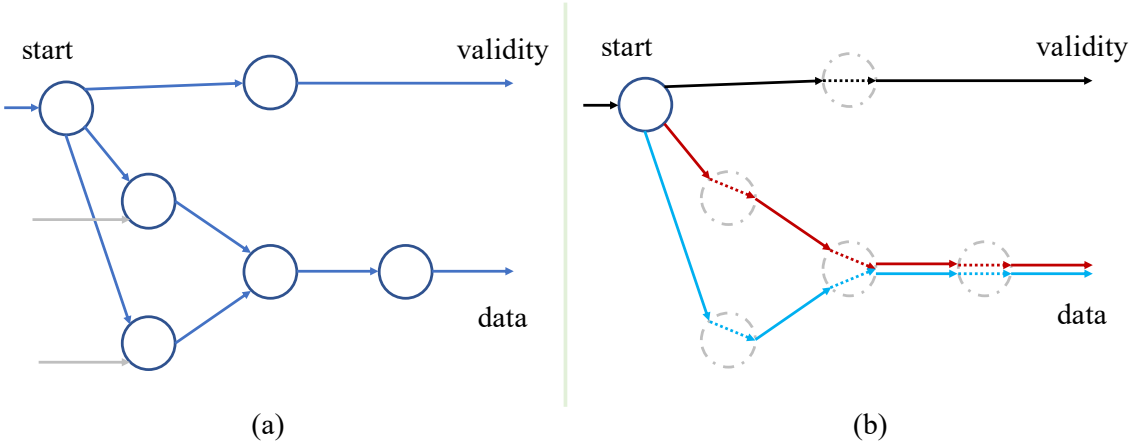


Figure 5.2: An example of correctness constraint and performance constraint. (a) the topology of a pipeline stage, (b) paths in this stage.

5.1.2 Strategy

There are two options to satisfy the correctness constraints. The first option uses an ordinary placement tool to place all cells, after which a timing analysis tool is used to find the worst delay in the datapath. With this delay information, one can synthesize a delay line and perform an incremental update to place this delay line. However, this approach ignores the performance constraints and thus may lead to poor performance.

The second approach explicitly optimizes the datapath delay at runtime. Assuming the target delay of a stage is known, we can use the timing analysis tool to find the top-k paths that violate the performance constraint, and then increase the weight of nets containing these paths accordingly. A weighted wirelength-driven placement is performed to optimize the weighted wirelength and thus improve the delay of these paths. These steps are repeated until delays cannot be further improved. Finally, a delay line is synthesized and carefully placed to satisfy the correctness constraints.

We have done many preparatory works for integrating the placement flow with a timing analysis tool for asynchronous designs. Once the integration between the timer and the placer is complete, the placer can fetch timing information, making it possible to synthesize delay lines during placement.

5.2 Detailed Placement for Gridded Cells

The gridded cell placement flow does not have a dedicated detailed placement step. The main reason is that swapping cells among clusters can change the height of clusters, and eventually make cells out of the placement boundary. Statistically speaking, the influence may be tiny when swapping a large number of cells. Therefore, during detailed placement, we can pretend all gridded cells have the same height, and thus all the swapping operations are legal and safe. After cell swapping, we can recompute the cluster locations to finalize the physical layout.

Bibliography

[1] M. J. S. Smith. *Application-specific integrated circuits*, volume 7. Addison-Wesley Reading, MA, 1997.

[2] C. Mead. Introduction to vlsi systems. *IEE Proceedings I-Solid-State and Electron Devices*, 128(1):18, 1980.

[3] B. Lojek. *History of semiconductor engineering*. Springer, 2007.

[4] N. H. Weste and D. Harris. *CMOS VLSI design: a circuits and systems perspective*. Pearson Education India, 2015.

[5] T. Xanthopoulos. *Clocking in modern VLSI systems*. Springer Science & Business Media, 2009.

[6] D. MacMillen, R. Camposano, D. Hill, and T. W. Williams. An industrial view of electronic design automation. *IEEE transactions on computer-aided design of integrated circuits and systems*, 19(12):1428–1448, 2000.

[7] L.-T. Wang, Y.-W. Chang, and K.-T. T. Cheng. *Electronic design automation: synthesis, verification, and test*. Morgan Kaufmann, 2009.

[8] L. Lavagno, I. L. Markov, G. Martin, and L. K. Scheffer. *Electronic design automation for IC implementation, circuit design, and process technology: circuit design, and process technology*. CRC Press, 2016.

[9] D. Jansen et al. *The electronic design automation handbook*. Springer, 2003.

[10] L. Lavagno, L. Scheffer, and G. Martin. *EDA for IC implementation, circuit design, and process technology*. CRC press, 2018.

[11] E. Brunvand. *Digital VLSI chip design with Cadence and Synopsys CAD tools*. Addison-Wesley New York, 2010.

[12] M. Donno, A. Ivaldi, L. Benini, and E. Macii. Clock-tree power optimization based on rtl clock-gating. In *Proceedings of the 40th annual Design Automation Conference*, pages 622–627, 2003.

[13] M. Donno, E. Macii, and L. Mazzoni. Power-aware clock tree planning. In *Proceedings of the 2004 international symposium on Physical design*, pages 138–147, 2004.

[14] S. A. Butt, S. Schmermbeck, J. Rosenthal, A. Pratsch, and E. Schmidt. System level clock tree synthesis for power optimization. In *2007 Design, Automation & Test in Europe Conference & Exhibition*, pages 1–6. IEEE, 2007.

[15] H. Homayoun, S. Golshan, E. Bozorgzadeh, A. Veidenbaum, and F. J. Kurdahi. On leakage power optimization in clock tree networks for asics and general-purpose processors. *Sustainable Computing: Informatics and Systems*, 1(1):75–87, 2011.

[16] Y. Chen and D. Wong. An algorithm for zero-skew clock tree routing with buffer insertion. In *Proceedings ED&TC European Design and Test Conference*, pages 230–236. IEEE, 1996.

[17] C.-W. A. Tsao and C.-K. Koh. Ust/dme: a clock tree router for general skew constraints. *ACM Transactions on Design Automation of Electronic Systems (TODAES)*, 7(3):359–379, 2002.

[18] C.-M. Chang, S.-H. Huang, Y.-K. Ho, J.-Z. Lin, H.-P. Wang, and Y.-S. Lu. Type-matching clock tree for zero skew clock gating. In *Proceedings of the 45th annual Design Automation Conference*, pages 714–719, 2008.

[19] Q. Wu, M. Pedram, and X. Wu. Clock-gating and its application to low power design of sequential circuits. *IEEE Transactions on Circuits and Systems I: Fundamental Theory and Applications*, 47(3):415–420, 2000.

[20] F. Emnett and M. Biegel. Power reduction through rtl clock gating. *SNUG, San Jose*, pages 1–11, 2000.

[21] J. Shinde and S. Salankar. Clock gating—a power optimizing technique for vlsi circuits. In *2011 annual IEEE India conference*, pages 1–4. IEEE, 2011.

[22] G. Semeraro, G. Magklis, R. Balasubramonian, D. H. Albonesi, S. Dwarkadas, and M. L. Scott. Energy-efficient processor design using multiple clock domains with dynamic voltage and frequency scaling. In *Proceedings Eighth International Symposium on High Performance Computer Architecture*, pages 29–40. IEEE, 2002.

[23] G. Magklis, M. L. Scott, G. Semeraro, D. H. Albonesi, and S. Dropsho. Profile-based dynamic voltage and frequency scaling for a multiple clock domain microprocessor. In *Proceedings of the 30th annual international symposium on Computer architecture*, pages 14–27, 2003.

[24] D. E. Muller. A theory of asynchronous circuits. *Report 75, University of Illinois*, 1956.

[25] C. J. Myers. *Asynchronous circuit design.* John Wiley & Sons, 2001.

[26] J. Spars and S. Furber. *Principles asynchronous circuit design.* Springer, 2002.

[27] J. Sparsø. *Introduction to Asynchronous Circuit Design.* DTU Compute, Technical University of Denmark, 2020.

[28] A. J. Martin, A. Lines, R. Manohar, M. Nystrom, P. Penzes, R. Southworth, U. Cummings, and T. K. Lee. The design of an asynchronous mips r3000 microprocessor. In *Proceedings Seventeenth Conference on Advanced Research in VLSI*, pages 164–181. IEEE, 1997.

[29] J. Teifel and R. Manohar. Highly pipelined asynchronous fpgas. In *proceedings of the 2004 ACM/SIGDA 12th International symposium on field programmable gate arrays*, pages 133–142, 2004.

[30] F. Akopyan, J. Sawada, A. Cassidy, R. Alvarez-Icaza, J. Arthur, P. Merolla, N. Imam, Y. Nakamura, P. Datta, G.-J. Nam, et al. Truenorth: Design and tool flow of a 65 mw 1 million neuron programmable neurosynaptic chip. *IEEE transactions on computer-aided design of integrated circuits and systems*, 34(10):1537–1557, 2015.

[31] D. M. Chapiro. Globally-asynchronous locally-synchronous systems. Technical report, Stanford Univ CA Dept of Computer Science, 1984.

[32] M. Krstic, E. Grass, F. K. Gürkaynak, and P. Vivet. Globally asynchronous, locally synchronous circuits: Overview and outlook. *IEEE Design & Test of computers*, 24(5):430–441, 2007.

[33] L. Xiu. *VLSI circuit design methodology demystified: a conceptual taxonomy*. John Wiley & Sons, 2007.

[34] H. Eriksson, P. Larsson-Edefors, T. Henriksson, and C. Svensson. Full-custom vs. standard-cell design flow: an adder case study. In *Proceedings of the 2003 Asia and South Pacific Design Automation Conference*, pages 507–510, 2003.

[35] P. A. Merolla, J. V. Arthur, R. Alvarez-Icaza, A. S. Cassidy, J. Sawada, F. Akopyan, B. L. Jackson, N. Imam, C. Guo, Y. Nakamura, et al. A million spiking-neuron integrated circuit with a scalable communication network and interface. *Science*, 345(6197):668–673, 2014.

[36] J. Grad and J. E. Stine. A standard cell library for student projects. In *Proceedings 2003 IEEE International Conference on Microelectronic Systems Education. MSE'03*, pages 98–99. IEEE, 2003.

[37] W.-W. Hu, J.-Y. Zhao, S.-Q. Zhong, X. Yang, E. Guidetti, and C. Wu. Implementing a 1ghz four-issue out-of-order execution microprocessor in a standard cell asic methodology. *Journal of Computer Science and Technology*, 22(1):1–14, 2007.

[38] A. B. Kahng, J. Lienig, I. L. Markov, and J. Hu. *VLSI physical design: from graph partitioning to timing closure*. Springer Science & Business Media, 2011.

[39] C. J. Alpert, D. P. Mehta, and S. S. Sapatnekar. *Handbook of algorithms for physical design automation*. CRC press, 2008.

[40] H. Murata, K. Fujiyoshi, S. Nakatake, and Y. Kajitani. Vlsi module placement based on rectangle-packing by the sequence-pair. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 15(12):1518–1524, 1996.

[41] M. Sarrafzadeh, M. Wang, and X. Yang. *Modern placement techniques*. Springer Science & Business Media, 2003.

[42] B. W. Kernighan and S. Lin. An efficient heuristic procedure for partitioning graphs. *Bell system technical journal*, 49(2):291–307, 1970.

[43] C. M. Fiduccia and R. M. Mattheyses. A linear-time heuristic for improving network partitions. In *19th design automation conference*, pages 175–181. IEEE, 1982.

[44] C. J. Alpert, J.-H. Huang, and A. B. Kahng. Multilevel circuit partitioning. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 17(8):655–667, 1998.

[45] G. Karypis, R. Aggarwal, V. Kumar, and S. Shekhar. Multilevel hypergraph partitioning: Applications in vlsi domain. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 7(1):69–79, 1999.

[46] R.-S. Tsay, E. S. Kuh, and C.-P. Hsu. Proud: A sea-of-gates placement algorithm. *IEEE Design & Test of Computers*, 5(6):44–56, 1988.

[47] J. M. Kleinhans, G. Sigl, F. M. Johannes, and K. J. Antreich. Gordian: Vlsi placement by quadratic programming and slicing optimization. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 10(3):356–365, 1991.

[48] A. R. Agnihotri, S. Ono, and P. H. Madden. Recursive bisection placement: Feng shui 5.0 implementation details. In *Proceedings of the 2005 international symposium on Physical design*, pages 230–232, 2005.

[49] J. A. Roy, D. A. Papa, S. N. Adya, H. H. Chan, A. N. Ng, J. F. Lu, and I. L. Markov. Capo: robust and scalable open-source min-cut floorplacer. In *Proceedings of the 2005 international symposium on Physical design*, pages 224–226, 2005.

[50] C. Sechen and A. Sangiovanni-Vincentelli. The timberwolf placement and routing package. *IEEE Journal of Solid-State Circuits*, 20(2):510–522, 1985.

[51] C. Sechen and A. Sangiovanni-Vincentelli. Timberwolf3. 2: A new standard cell placement and global routing package. In *23rd ACM/IEEE Design Automation Conference*, pages 432–439. IEEE, 1986.

[52] C. Sechen. An improved simulated annealing algorithm for row-based placement. In *Proc. of International Conference on Computer-Aided Design, 1987*, 1987.

[53] W.-J. Sun and C. Sechen. Efficient and effective placement for very large circuits. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 14(3):349–359, 1995.

[54] X. Yang, M. Sarrafzadeh, et al. Dragon2000: Standard-cell placement tool for large industry circuits. In *IEEE/ACM International Conference on Computer Aided Design. ICCAD-2000. IEEE/ACM Digest of Technical Papers (Cat. No. 00CH37140)*, pages 260–263. IEEE, 2000.

[55] K. M. Hall. An r-dimensional quadratic placement algorithm. *Management science*, 17(3):219–229, 1970.

[56] Y.-W. Chang, Z.-W. Jiang, and T.-C. Chen. Essential issues in analytical placement algorithms. *IPSJ Transactions on System LSI Design Methodology*, 2:145–166, 2009.

[57] I. L. Markov, J. Hu, and M.-C. Kim. Progress and challenges in vlsi placement research. *Proceedings of the IEEE*, 103(11):1985–2003, 2015.

[58] P. Spindler, U. Schlichtmann, and F. M. Johannes. Kraftwerk2—a fast force-directed quadratic placement approach using an accurate net model. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 27(8):1398–1411, 2008.

[59] N. Viswanathan and C.-N. Chu. Fastplace: efficient analytical placement using cell shifting, iterative local refinement, and a hybrid net model. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 24(5):722–733, 2005.

[60] M.-C. Kim, D.-J. Lee, and I. L. Markov. Simpl: An effective placement algorithm. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 31(1):50–60, 2011.

[61] M.-C. Kim, N. Viswanathan, C. J. Alpert, I. L. Markov, and S. Ramji. Maple: Multilevel adaptive placement for mixed-size designs. In *Proceedings of the 2012 ACM international symposium on International Symposium on Physical Design*, pages 193–200, 2012.

[62] T. Lin, C. Chu, J. R. Shinnerl, I. Bustany, and I. Nedelchev. Polar: Placement based on novel rough legalization and refinement. In *2013 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pages 357–362. IEEE, 2013.

[63] A. B. Kahng and Q. Wang. Implementation and extensibility of an analytic placer. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 24(5):734–747, 2005.

[64] T.-C. Chen, Z.-W. Jiang, T.-C. Hsu, H.-C. Chen, and Y.-W. Chang. Ntuplace3: An analytical placer for large-scale mixed-size designs with preplaced blocks and density constraints. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 27(7):1228–1240, 2008.

[65] J. Lu, P. Chen, C.-C. Chang, L. Sha, D. J.-H. Huang, C.-C. Teng, and C.-K. Cheng. eplace: Electrostatics-based placement using fast fourier transform and nesterov's method. *ACM Transactions on Design Automation of Electronic Systems (TODAES)*, 20(2):1–34, 2015.

[66] C.-K. Cheng, A. B. Kahng, I. Kang, and L. Wang. Replace: Advancing solution quality and routability validation in global placement. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 38(9):1717–1730, 2018.

[67] Y. Lin, Z. Jiang, J. Gu, W. Li, S. Dhar, H. Ren, B. Khailany, and D. Z. Pan. Dreamplace: Deep learning toolkit-enabled gpu acceleration for modern vlsi placement. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 40(4):748–761, 2020.

[68] J. Lu, P. Chen, C.-C. Chang, L. Sha, J. Dennis, H. Huang, C.-C. Teng, and C.-K. Cheng. eplace: Electrostatics based placement using nesterov's method. In *2014 51st ACM/EDAC/IEEE Design Automation Conference (DAC)*, pages 1–6. IEEE, 2014.

[69] D. Hill. Method and system for high speed detailed placement of cells within an integrated circuit design, April 9 2002. US Patent 6,370,673.

[70] M. Pan, N. Viswanathan, and C. Chu. An efficient and effective detailed placement algorithm. In *ICCAD-2005. IEEE/ACM International Conference on Computer-Aided Design, 2005.*, pages 48–55. IEEE, 2005.

[71] A. B. Kahng, P. Tucker, and A. Zelikovsky. Optimization of linear placements for wirelength minimization with free sites. In *Proceedings of the ASP-DAC'99 Asia and South Pacific Design Automation Conference 1999 (Cat. No. 99EX198)*, pages 241–244. IEEE, 1999.

[72] M. Wang et al. Nrg: Global and detailed placement. In *1997 Proceedings of IEEE International Conference on Computer Aided Design (ICCAD)*, pages 532–537. IEEE, 1997.

[73] K. Doll, F. M. Johannes, and K. J. Antreich. Iterative placement improvement by network flow methods. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 13(10):1189–1200, 1994.

[74] S. Cauley, V. Balakrishnan, Y. C. Hu, and C.-K. Koh. A parallel branch-and-cut approach for detailed placement. *ACM Transactions on Design Automation of Electronic Systems (TODAES)*, 16(2):1–19, 2011.

[75] A. Davis and S. M. Nowick. An introduction to asynchronous circuit design. *The Encyclopedia of Computer Science and Technology*, 38:1–58, 1997.

[76] J. Sparsø. Asynchronous circuit design–a tutorial. 2006.

[77] P. A. Beerel, R. O. Ozdag, and M. Ferretti. *A designer's guide to asynchronous VLSI*. Cambridge University Press, 2010.

[78] A. J. Martin. The limitations to delay-insensitivity in asynchronous circuits. In *Beauty is our business*, pages 302–311. Springer, 1990.

[79] I. E. Sutherland. Micropipelines. *Communications of the ACM*, 32(6):720–738, 1989.

[80] S. M. Nowick and M. Singh. High-performance asynchronous pipelines: An overview. *Ieee design & test of computers*, 28(5):8–22, 2011.

[81] J. T. Udding. A formal model for defining and classifying delay-insensitive circuits and systems. *Distributed Computing*, 1(4):197–204, 1986.

[82] R. Manohar and Y. Moses. Asynchronous signalling processes. In *2019 25th IEEE International Symposium on Asynchronous Circuits and Systems (ASYNC)*, pages 68–75. IEEE, 2019.

[83] R. Manohar and A. J. Martin. Quasi-delay-insensitive circuits are turing-complete. Technical report, CALIFORNIA INST OF TECH PASADENA DEPT OF COMPUTER SCIENCE, 1995.

[84] D. E. Muller. Asynchronous logics and application to information processing. *Switching Theory in Space Technology*, 4, 1963.

[85] C. L. Seitz and C. Mead. System timing. *Introduction to VLSI systems*, pages 218–262, 1980.

[86] P. Maurine, J.-B. Rigaud, F. Bouesse, G. Sicard, and M. Renaudin. Static implementation of qdi asynchronous primitives. In *International Workshop on Power and Timing Modeling, Optimization and Simulation*, pages 181–191. Springer, 2003.

[87] D. Sokolov, J. Murphy, A. Bystrov, and A. Yakovlev. Design and analysis of dual-rail circuits for security applications. *IEEE Transactions on Computers*, 54(4):449–460, 2005.

[88] A. J. Martin. Compiling communicating processes into delay-insensitive vlsi circuits. *Distributed computing*, 1(4):226–234, 1986.

[89] K. van Berkel, J. Kessels, M. Roncken, R. Saeijs, and F. Schalij. The vlsi-programming language tangram and its translation into handshake circuits. In *Proceedings of the European Conference on Design Automation.*, pages 384–389. IEEE, 1991.

[90] H. Van Gageldonk, K. Van Berkel, A. Peeters, D. Baumann, D. Gloor, and G. Stegmann. An asynchronous low-power 80c51 microcontroller. In *Proceedings Fourth International Symposium on Advanced Research in Asynchronous Circuits and Systems*, pages 96–107. IEEE, 1998.

[91] C. G. Wong and A. J. Martin. High-level synthesis of asynchronous systems by data-driven decomposition. In *Proceedings of the 40th annual Design Automation Conference*, pages 508–513, 2003.

[92] J. Teifel and R. Manohar. Static tokens: Using dataflow to automate concurrent pipeline synthesis. In *10th International Symposium on Asynchronous Circuits and Systems, 2004. Proceedings.*, pages 17–27. IEEE, 2004.

[93] R. Li, L. Berkley, Y. Yang, and R. Manohar. Fluid: An asynchronous high-level synthesis tool for complex program structures. In *2021 27th IEEE International Symposium on Asynchronous Circuits and Systems (ASYNC)*, pages 1–8. IEEE, 2021.

[94] S. Hauck. Asynchronous design methodologies: An overview. *Proceedings of the IEEE*, 83(1):69–93, 1995.

[95] I. Sutherland and S. Fairbanks. Gasp: A minimal fifo control. In *Proceedings Seventh International Symposium on Asynchronous Circuits and Systems. ASYNC 2001*, pages 46–53. IEEE, 2001.

[96] M. Singh and S. M. Nowick. Mousetrap: High-speed transition-signaling asynchronous pipelines. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 15(6):684–698, 2007.

[97] A. M. Lines. Pipelined asynchronous circuits. 1998.

[98] A. Devgan and C. Kashyap. Block-based static timing analysis with uncertainty. In *ICCAD-2003. International Conference on Computer Aided Design (IEEE Cat. No. 03CH37486)*, pages 607–614. IEEE, 2003.

[99] R. Karmazin, C. T. O. Otero, and R. Manohar. celltk: Automated layout for asynchronous circuits with nonstandard cells. In *2013 IEEE 19th International Symposium on Asynchronous Circuits and Systems*, pages 58–66. IEEE, 2013.

[100] A. B. Kahng and G. Robins. *On optimal interconnections for VLSI*, volume 301. Springer Science & Business Media, 1994.

[101] S. Peyer, D. Rautenbach, and J. Vygen. A generalization of dijkstra's shortest path algorithm with applications to vlsi routing. *Journal of Discrete Algorithms*, 7(4):377–390, 2009.

[102] M. Ligthart, K. Fant, R. Smith, A. Taubin, and A. Kondratyev. Asynchronous design using commercial hdl synthesis tools. In *Proceedings Sixth International Symposium on Advanced Research in Asynchronous Circuits and Systems (ASYNC 2000)(Cat. No. PR00586)*, pages 114–125. IEEE, 2000.

[103] A. Kondratyev and K. Lwin. Design of asynchronous circuits using synchronous cad tools. *IEEE Design & Test of Computers*, 19(4):107–117, 2002.

[104] S. M. Sait and H. Youssef. *VLSI physical design automation: theory and practice*, volume 6. World Scientific Publishing Company, 1999.

[105] P. A. Beerel, G. D. Dimou, and A. M. Lines. Proteus: An ASIC flow for GHz asynchronous designs. *IEEE Design and Test of Computers*, 28(5):36–51, 2011.

[106] M. Moreira, B. Oliveira, J. Pontes, and N. Calazans. A 65nm standard cell set and flow dedicated to automated asynchronous circuits design. In *2011 IEEE International SOC Conference*, pages 99–104. IEEE, 2011.

[107] M. Trevisan, M. Arendt, A. Ziesemer, R. Reis, and N. L. V. Calazans. Automated synthesis of cell libraries for asynchronous circuits. In *Proceedings of the 27th Symposium on Integrated Circuits and Systems Design*, pages 1–7, 2014.

[108] C. J. Poirier. Excellerator: custom cmos leaf cell layout generator. *IEEE transactions on computer-aided design of integrated circuits and systems*, 8(7):744–755, 1989.

[109] G. A. Northrop and P.-F. Lu. A semi-custom design flow in high-performance microprocessor design. In *Proceedings of the 38th annual Design Automation Conference*, pages 426–431, 2001.

[110] C.-Y. Hung, P.-Y. Chou, and W.-K. Mak. Mixed-cell-height standard cell placement legalization. In *Proceedings of the on Great Lakes Symposium on VLSI 2017*, pages 149–154, 2017.

[111] P. H. Madden. Reporting of standard cell placement results. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 21(2):240–247, 2002.

[112] P. Spindler, U. Schlichtmann, and F. M. Johannes. Abacus: fast legalization of standard cell circuits with minimal movement. In *Proceedings of the 2008 international symposium on Physical design*, pages 47–53, 2008.

[113] H. Ren, D. Z. Pan, C. J. Alpert, P. G. Villarrubia, and G.-J. Nam. Diffusion-based placement migration with application on legalization. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 26(12):2158–2172, 2007.

[114] Y. Saad. *Iterative methods for sparse linear systems*, volume 82. siam, 2003.

[115] J. He, Y. Yang, and R. Manohar. A power router for gridded cell placement. In *Workshop on Open-Source EDA Technology, International Conference on Computer-Aided Design (ICCAD)*, 2020.

[116] G. Wu, T. Lin, H.-H. Huang, C. Chu, and P. A. Beerel. Asynchronous circuit placement by lagrangian relaxation. In *2014 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pages 641–646. IEEE, 2014.

[117] R. Karmazin, S. Longfield, C. T. O. Otero, and R. Manohar. Timing driven placement for quasi delay-insensitive circuits. In *2015 21st IEEE International Symposium on Asynchronous Circuits and Systems*, pages 45–52. IEEE, 2015.

[118] C. P. Sotiriou. Implementing asynchronous circuits using a conventional eda tool-flow. In *Proceedings of the 39th annual Design Automation Conference*, pages 415–418, 2002.

[119] G. Wu and C. Chu. Detailed placement algorithm for vlsi design with double-row height standard cells. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 35(9):1569–1573, 2015.

[120] W.-K. Chow, C.-W. Pui, and E. F. Young. Legalization algorithm for multiple-row height standard cell design. In *2016 53nd ACM/EDAC/IEEE Design Automation Conference (DAC)*, pages 1–6. IEEE, 2016.

[121] Y. Lin, B. Yu, X. Xu, J.-R. Gao, N. Viswanathan, W.-H. Liu, Z. Li, C. J. Alpert, and D. Z. Pan. Mrdp: Multiple-row detailed placement of heterogeneous-sized cells for advanced nodes. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 37(6):1237–1250, 2017.

[122] C.-H. Wang, Y.-Y. Wu, J. Chen, Y.-W. Chang, S.-Y. Kuo, W. Zhu, and G. Fan. An effective legalization algorithm for mixed-cell-height standard cells. In *2017 22nd Asia and South Pacific Design Automation Conference (ASP-DAC)*, pages 450–455. IEEE, 2017.

[123] J. Chen, Y.-W. Chang, and Y.-Y. Wu. Mixed-cell-height detailed placement considering complex minimum-implant-area constraints. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 40(10):2128–2141, 2020.

[124] Y.-W. Tseng and Y.-W. Chang. Mixed-cell-height placement considering drain-to-drain abutment. In *2018 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pages 1–6. IEEE, 2018.

[125] H. Li, W.-K. Chow, G. Chen, B. Yu, and E. F. Young. Pin-accessible legalization for mixed-cell-height circuits. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 41(1):143–154, 2021.

[126] H. Li, W.-K. Chow, G. Chen, E. F. Young, and B. Yu. Routability-driven and fence-aware legalization for mixed-cell-height circuits. In *Proceedings of the 55th Annual Design Automation Conference*, pages 1–6, 2018.

[127] Z. Zhu, J. Chen, W. Zhu, and Y.-W. Chang. Mixed-cell-height legalization considering technology and region constraints. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 39(12):5128–5141, 2020.

[128] X. Li, J. Chen, W. Zhu, and Y.-W. Chang. Analytical mixed-cell-height legalization considering average and maximum movement minimization. In *Proceedings of the 2019 International Symposium on Physical Design*, pages 27–34, 2019.

[129] Y. Yang, J. He, and R. Manohar. Dali: A gridded cell placement flow. In *2020 IEEE/ACM International Conference On Computer Aided Design (ICCAD)*, pages 1–9. IEEE, 2020.

[130] W. Hua, Y.-S. Lu, K. Pingali, and R. Manohar. Cyclone: A static timing and power engine for asynchronous circuits. In *2020 26th IEEE International Symposium on Asynchronous Circuits and Systems (ASYNC)*, pages 11–19. IEEE, 2020.