OpenLANE: The Open-Source Digital ASIC Implementation Flow

Ahmed Alaa Ghazy[1] and Mohamed Shalan[2]
[1]Efabless Corporation, San Jose, USA
[2]The American University in Cairo, New Cairo, EGYPT

***Abstract*—OpenLANE is a tape-out-hardened flow that addresses two main use cases: hardening a macro and integrating a System-on-a-Chip (SoC). It was used successfully to tape out a family of RISC-V based SoCs known as "striVe". This paper reviews the various components of the flow with a particular focus on the challenges that faced SoC integration while working on the first of the striVe chips and the main ideas used to overcome them, achieving full automation.**

***Index Terms*—ASIC, EDA, OpenLANE, OpenROAD, SoC, Flow, Macro, Floor-planning, RISC-V, VLSI, PDK, Open-Source**

I. Introduction

As a manufacturing-ready open process development kit (PDK) is emerging [1], and as open-source EDA tools are reaching an unprecedented level of maturity from logic synthesis to placement and routing to physical verification, OpenLANE [2] was at the intersection of all of those outstanding efforts, allowing a new methodology to be born, creating a fully open-source RTL-to-GDSII flow. The methodology was developed with the open PDK (SKY130) in mind, but, at the same time, it remained generic to be relevant and configurable for other technologies.

The OpenLANE flow utilizes tools mainly from the OpenROAD [3], YosysHQ [4], and Open Circuit Design [5] projects. The way those tools are used, augmented by a number of other custom tools and scripts, defines the methodology of the flow. When the project started in October 2019, the OpenROAD tools were all standalone, almost each of them having its own infrastructure, front-ends, and back-ends, which were harder to work with than now. This might seem unintuitive because the basic goal of such architecture is that each tool does one thing and does it well. However, as more thoroughly discussed in [6], this 1980s style EDA tool flow does not work well as re-iterating through some of the flow steps makes the cost of file I/O operations begin to dominate the cost of the actual processing needed. This, as well as other reasons, was the main motivation for the shift to an EDA physical data model shared among the various tools, OpenDB [7]. This provided an infrastructure for the creation of custom tools, which is what was utilized in OpenLANE to supplement the flow tools with other custom tools and utilities needed to fill the compatibility and methodology gaps to achieve full automation.

OpenLANE supports two main use cases. First, It can be used to harden designs from their RTL HDL models obtaining what we will refer to as soft macros. The second use case is integrating macros into a complete chip. To demonstrate its capabilities, OpenLANE has been used to successfully tape out a family of RISC-V based SoCs called striVe.

The following sections will overview the OpenLANE flow, the tools used or created, and both supported use cases with a focus on the SoC integration flow used to fully automate the process from an RTL description to a manufacturable LVS- and DRC-clean *chip* layout.

II. OpenLANE Macro Hardening Flow

Hardening a design is the process of taking it from Hardware Description Language (HDL) model to the various views of the manufacturable mask layouts.

A hardened design (a soft macro) is usually then instantiated within another encompassing design. There are usually two purposes behind hardening a macro before using it in the context of a bigger design.

- First, it may not be possible at all to flatten the macro within the bigger design for several reasons; for instance, the macro and the design may be using vastly different sets of IPs or standard cell libraries that are incompatible either geometrically (using different site dimensions) or electrically (powered using different voltage domains).
- The other reason is simply reusability. Hardening a macro can be thought of as a well-invested one-time effort. Once physically verified, that is, once LVS- and DRC-clean, the same verified macro can be used across different designs. Also, if one is strict with the specification of the macro interface, which means pre-defining its exact dimensions and pin locations, then in cases where the functional definition of the macro itself is subject to change or if issues with the macro were found later in the process, then the consecutive steps in the process do not have to be repeated after fixing the issue, and it is a matter of simply re-hardening the macro, according to the specification, and "plugging" it in instead of its outdated version. This makes it possible to work top-down instead of bottom-up once a preliminary version of the macro is available. This use case is supported through a custom I/O placer available through the flow.

Fig. 1 illustrates the basic default flow; this is what runs in the batch (non-interactive) mode. Most of the steps are configurable and custom flows can be created by the use of interactive scripts. The flow expects the design source HDL

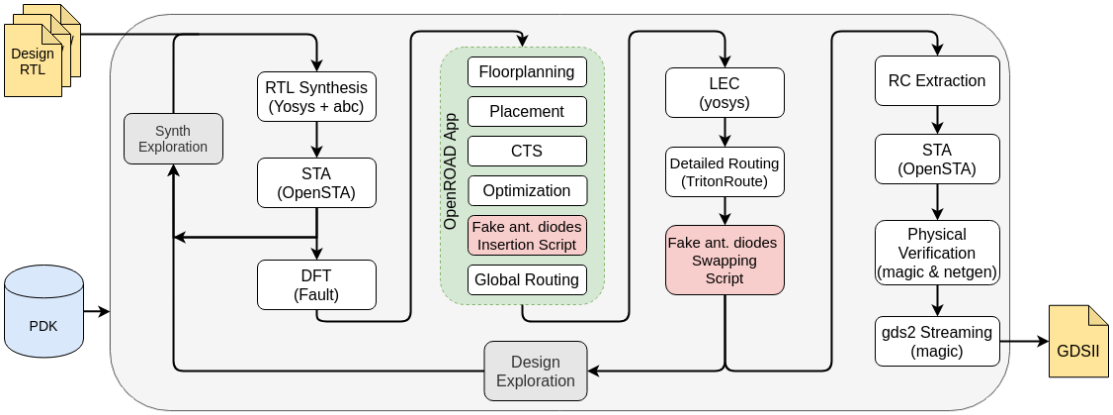

Fig. 1. Macro Hardening Flow

files as an input as well as the desired PDK source files. Below is a summarized breakdown of thestages seen in the figure.

A. RTL Synthesis and STA

The design is synthesized into a gate-level netlist using yosys and static timing analysis is performed on the resulting netlist using OpenSTA [8]. An optional so-called Synthesis Exploration can be performed; this is where the space of gate-level netlists equivalent to the input design is explored. Currently there are four default synthesis strategies generating four different points in the design space; those represent different degrees of area-delay trade-offs as well. A user can also add their own custom synthesis script/strategy if desired. STA is performed on each point in the design space and the result would be represented graphically on an HTML dashboard.

B. Insertion of DFT structures

An open-source Design For Testability (DFT) toolchain, Fault [9], can optionally be used to modify the netlist, inserting scan chains and the necessary IO ports to scan and test the design after fabrication. More on that in [10]. A variant of the striVe chips is being developed with DFT structures included in preparation for a full integration with OpenLANE.

C. Physical Implementation

Advancing with the physical implementation, we note that most of the tools in this stage are used from within the OpenROAD application [11] in combination with other tools, some of them are custom and based on the OpenDB infrastructure, while others are indpendent. For example, every time a tool modifies the netlist, which happens during Clock Tree Synthesis (CTS), post-placement optimizations (using OpenPhySyn [12]), or diode insertion, an optional Logic Equivalence Check (LEC) using yosys is performed to ensure that the new netlist is indeed functionally equivalent to the previous one tracing all the way to the original synthesized netlist output by yosys. For I/O pin placement, as mentioned earlier, OpenLANE supports two more use cases besides the default one in the OpenROAD application; one of them is fully custom I/O pin placement for cases where a user would prefer to have strict control over pin locations. The other custom mode, which is particularly useful during SoC integration to achieve clean routing on the top-level is the so-called contextualized I/O placement; this mode automatically places the I/O pins optimally according to the *context* of their instantiation at a higher level of hierarchy. The full details of this mode will be explained later on. The output of this stage is a routed DEF, ready to then be evaluated.

D. Post-routing Evaluation of Results

DRC and LVS are then performed using magic [13] and netgen [14]. Antenna checking is performed by either OpenROAD's ARC (Antenna Rule Checker) or using magic. Extraction of parasitics from the routed layout is then done using SPEF_EXTRACTOR [15], followed by another round of static timing analysis to have more accurate timing reports that correspond to the actual physical layout.

The final outputs of the flow, among various physical views and reports, are mainly GDSII and LEF views, which can be used in bigger designs.

III. OpenLANE SoC Integration Flow

We begin now looking at the flow that was developed in response to the challenges associated with the fully automatic completion of a manufacturable chip. An outline of the steps can be seen in Fig. 2.

The first striVe chip had many issues that required manual intervention. For example, the chip had several (rectangular) macros scattered within the pad frame, each having pins assigned randomly to its four sides. No kind of optimization was applied to increase the routability of those macros on the top-level; this was again due to the difficulty of controlling the pin placement, which was done before the chip was even floorplanned in accordance with the previous section.

Another issue was the lack of a tool that could perform the power-routing on the top-level; those power routes need to be kept short, wide, and use the highest metal layers available to reduce the IR drop.

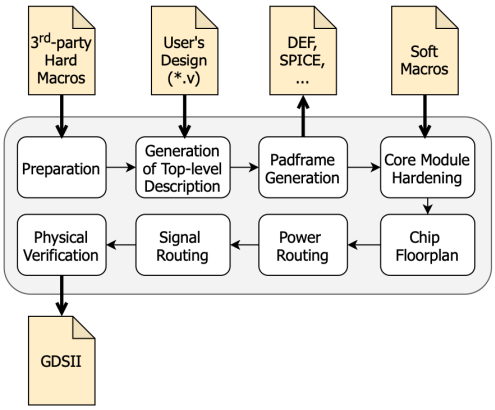

Fig. 2. SoC Integration Flow

The result of all of these difficulties was a sub-optimal chip floorplan with very complex and long routes on the top-level, which, after the signal routing stage, had several DRC violations as well as antenna violations due to the very long wire segments on relatively low metal layers.

Those issues were overcome back then either by manual intervention or by making trade-offs for automation. An example of the former is fixing the resulting DRC violations due to the complex top-level routes; the same goes for the power connections. An example of the latter, however, was an antenna avoidance methodology that trades space for less antenna violations; the idea was to reserve the space needed to protect *all* sinks in the design by inserting a custom fill cell that matches the footprint of an actual diode cell next to every sink. A report of the antenna rule violations is generated after routing, based on which only the smallest number of "fake diodes" are turned into "real diodes" to protect the vulnerable sinks. Since both cells have identical footprints on the metal layers used for routing, we know that this cell substitution step cannot cause more DRC violations. This strategy was already an improvement on its brute-force predecessor that inserts antenna diodes on all sinks by default. The improvement significantly reduces the capacitive loads on the affected nets and power consumption, but it still posed a clear limitation on the maximum achievable core utilization. Nevertheless, this worked well for the tape-outs since a very high core utilization was not a pressing factor. The next natural improvement on this strategy would be inserting diodes only as necessary during *global routing* assuming the detailed router highly honors the routing guides, which is the premise of recent efforts by OpenROAD since the introduction of the new OpenDB-based ARC.

Currently those issues have found solutions that allow full automation, which was made possible by enforcing certain guidelines and supporting the flow with a set of custom methodology utilities and tools that are used.

In Fig. 2, we see that a user usually provides a set of macros that are to be used in the design as is and are not to be flattened. Those macros lie in two categories; first, there are soft macros that were hardened using the flow described in the previous section. Since we had full control over the generation of those macros and their I/O pin pitches, sizes, and layers, we can guarantee that they are fully *routable*, and, thus, are ready to be directly used. The second type includes hard macros, usually designed independently by a third-party; examples of this include analog components, SRAM blocks, and the I/O pads themselves. Some of those macros may be already usable as they are, but oftentimes, as those macros were not designed with any of the tools used in OpenLANE in consideration, they suffer compatibility issues that prevent them from being optimally used in the flow. For example, their routability might not be optimal if the pins do not lie on the routing grid, are too small, or are simply hard to access through barricades that surround them. Moreover, it may be desirable and a good practice to fully separate the problem of DRC and LVS checking of those macros from DRC and LVS checks of the design they are used in. For such purposes, a set of scripts for wrapping and/or abstracting those hard macros are provided; they support a variety of options and modes, like extending pins to macro boundaries and black-boxing (obstructing) everything inside. This mode relies on magic-based scripts and can work directly with a GDSII view of the macro; it does not modify that view directly but instead creates an overlay on top of it, which is important to keep the integrity of the GDSII data obtained from the source. This step (Preparation in the figure) is a one-time effort, and the obtained routable macros, are now useful for any number of designs.

Users also provide the design source files (currently in Verilog) that describe the module they would like to implement as well as a *high-level* description of the I/O pads they want to use to interface with their module. A utility in OpenLANE takes that description and automatically generates a top-level description of the whole chip, with the right pads instantiated and configured to the desired mode. This step is intended for novice users, who would now be able to integrate an SoC without solid knowledge of the full details of I/O pads and how to configure them properly. If needed, experienced users can still simply skip this step and exercise more fine-grained control over their I/O pads by providing their own top-level description of the design.

a) Recommended Hierarchy: It is recommended that the top-level description of the design follows certain guidelines that result in cleaner layouts as well as other side benefits. The main recommendation is to partition the design into two modules that interact together: a pad frame module and a core module; this is the *logical* view of the chip. *Physically*, this corresponds to a pad frame enclosing only a single block.

The core module encapsulates the main user design that contains *all* other macros whereas the pad frame module contains the I/O, power, and corner pads needed in the design. The core module would be hardened by the regular flow just as described in the previous section while the pad frame is generated independently using padring [16] and is then processed by scripts based on either magic or OpenDB (both

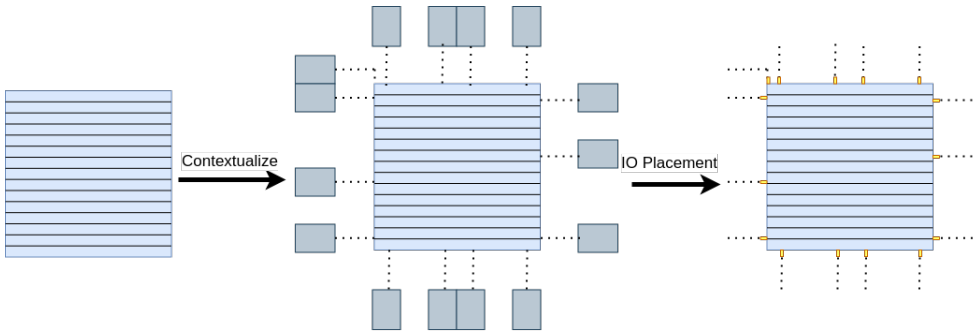

Fig. 3. Contextualized I/O Pin Placement

are available options) to correctly create port labels, which is necessary for top-level LVS and SPICE simulation of the pad frame.

b) Optimizations: When contrasted with the hierarchy of the first of the striVe chips, in which several macros were scrambled across the top-level, the elegance of the suggested hierarchy can be seen clearly. This hierarchy makes optimization on the top level a much easier problem. For example, instead of having to optimize the relative locations of several macros on the top-level as well as their pin positions, the problem is now split into two simpler ones. First, macro and standard cell placement only occurs inside the core module, which the regular macro hardening flow excels at. Having macros (or, even worse, standard cells) scattered at the top-level adds an unnecessary level of complexity associated with powering them directly from the power pads and/or tapping them. Second, assigning the I/O pins to their optimal positions is now only a problem with one module: the core module. However, the subtlety here is that the I/O pins are assigned while hardening the core module, which is before it gets placed within the pad frame. A utility in OpenLANE can be used to create a "virtual" floorplan that shows the *context* in which the design being hardened would be instantiated, that is, it "contextualizes" it, which is bringing in accurate connectivity information on how this macro would later interact with other external macros. This information is then used to place the I/O pins optimally, for example, with respect to the half-perimeter wire length (HPWL) between the nodes, which can be seen in Fig. 3.

c) Power-Routing: In order to automate top-level power routing, the core module must be hardened with two concentric core rings around its perimeter. Besides the electrical benefits of a core ring, it is much easier to automate the power routes from the supplies onto the core ring. The power router, another custom tool, attempts to use the highest metal layer possible to connect the power supplies to the core ring. An obvious reason is reducing the IR drop across the wires since the highest metal layers are usually the thickest; for that purpose, as well, the power router will also maximize the number of vias generated at intersections of connected metal layers. An added benefit of using the highest metal layers is to minimize the antenna effects on some of the gates *inside* the I/O pads which are required to be driven using those same voltage levels used for power. Fig. 4 shows an example of the resulting power-routed chip floorplan. See Appendix B for real layout of striVe2a in comparison with the layouts of Appendix A which did not follow the recommended hierarchy back then.

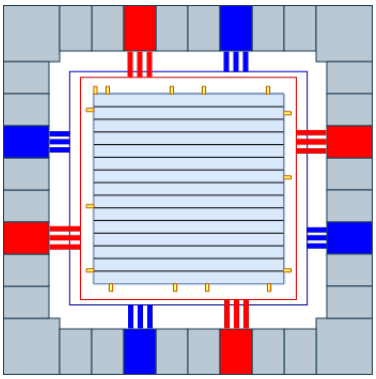

Fig. 4. Power-routed Chip Floorplan

There are other desirable benefits of the recommended hierarchy; for example, as per Fig. 2, since the pad frame is a logically separate module, a SPICE netlist can be extracted from the layout using magic, which can be used to functionally and electrically simulate the pad frame by itself. The latter is useful to test the resilience of the pad frame against electrostatic discharge (ESD) effects.

Conclusion

In this paper, we overviewed the key components of OpenLANE and the challenges that motivated the current methodology to develop in the direction it took.

Both use cases of the flow allow the automatic completion of manufacturable chips within a couple of hours if the guidelines outlined in the previous section are followed. Of course, the recommended hierarchy poses some challenges to some more complex use cases that involve components powered using several different power domains, but for most other purposes, the current methodology is sufficient to meet the needs of the upcoming scheduled public shuttle in November [17].

Most of the remaining work currently is aimed at making the SoC integration flow as easy as possible for users with varying