Integrating Asynchronous Circuits into the Caravel Testing Harness

Thomas Jagielski, Xiayuan Wen, Matthew Dobre, and Rajit Manohar
Computer Systems Lab, Yale University, New Haven, CT 06520
{thomas.jagielski, xiayuan.wen, matthew.dobre, rajit.manohar}@yale.edu

***Abstract*—The Caravel harness has been widely used to tape-out synchronous designs in the open-source SKY130 PDK. To use this flow for asynchronous circuit design using the open-source ACT framework, we can treat ACT-generated layout as an asynchronous macro. We develop an open-source toolflow that supports this approach. This paper presents the challenges encountered during the integration of an asynchronous macro in the Caravel harness and the methods employed to address these issues, including power ring generation and connection, pin extension, and custom fill insertion. In addition, we discuss the process of transferring data to the asynchronous circuit using Caravel's utilities.**

I. Introduction

Asynchronous circuits, also known as clockless or self-timed circuits, use local controllers instead of a global clock signal for synchronization. This significant difference from synchronous circuits introduces unique and more complex problems in circuit design, which necessitate the development of specialized automation tools. The ACT framework [1] is an open-source toolchain for asynchronous logic, which translates design described in `.act` files, a hierarchical design language including communication channels, and can generate layout files in the GDSII format.

To facilitate quick tape-out and verification of the asynchronous circuits with the Skywater 130nm Open PDK, we selected the Efabless Caravel testing harness [2]. The Efabless Caravel chip is divided into two areas, management area and user project area. The management area includes a RISC-V based SoC for configuring hardware resources, managing power supply, and monitoring and controlling modules within the user project area where a user's design is instantiated.

The Caravel chip is developed using `OpenLane` [3], an open-source automated RTL to GDSII flow which includes several tools such as `OpenROAD` [4], `Yosys` [5], `Magic` [6], `Netgen` [7], `SPEF-Extractor` [8], `KLayout` [9] and various custom scripts for exploring and optimizing the design. Integrating an asynchronous macro into the Caravel chip presents various challenges due to numerous difference between conventional synchronous toolflows (e.g. `OpenLane`) and asynchronous toolflow (ACT), as well as the specific organization of Caravel harness frame.

To address these issues, we develop a series of custom scripts to post-process the output files generated by the ACT framework, and make modifications to the configuration settings of `OpenLane`. In this paper, we also discuss the process of transferring data to the asynchronous circuit using Caravel's utilities, and highlight the considerations that users should be aware of. The post-processing steps presented have been used to tapeout an asynchronous MD5 hashing accelerator integrated in the Caravel harness.

II. Challenges

To integrate a design into the Caravel harness, users have two options: either instantiate RTL design within the user_project_wrapper and flatten them together or first harden the design as a macro and then incorporate this macro into the user_project_wrapper.

The asynchronous circuit cannot be appropriately described using RTL and synthesized by `OpenLane`; therefore, the second option is our only viable choice. However, direct integration of an asynchronous macro generated by the ACT framework into the Caravel chip using `OpenLane` introduces various challenges.

a) Fill insertion: Apart from standard cells, ACT supports custom cells due to the differing requirement for synthesizing different asynchronous circuit families. To improve area usage, ACT uses gridded cell placement supported by the `Dali` placer [10], which allows the cell height and cell width to be any integer multiple of the routing grid value. The freedom of cell heights can potentially lead to a more compact design, smaller wire length, and thus better delay and energy. When using this approach for SKY130, we discovered that the provided fill scripts could not meet the density requirements even after several iterations of fill insertion using the commercial tool, `Calibre`. We note that this issue did not arise in other process technologies that have been used previously with the ACT framework, including more advanced process nodes [11].

b) Power distribution: The top-level power distribution network of Caravel chip has two layers of straps, one is met5 which runs horizontally and the other is met4 which runs vertically. The ACT-generated macro also has a power mesh running horizontally and vertically. When the macro doesn't have any signals or powers routed on met5, the top-level power strap can be directly connected to the macro's internal power mesh using vias. However, for more complex designs with reasonable density, routing with met5 is necessary. In such cases, the macro needs a power ring, which is not provided by the current ACT framework for technologies with fewer number of metal layers like SKY130.

Once the power ring is generated, the strategy for distributing power from the ring to the macro's internal power mesh should account for the varying spacings between power lines within the macro, which result from the different height of cells. ACT includes a power detailed router for generating these varying metal stripes, and they must be tied to the power ring.

c) Macro integration: For macro integration, users provide a LEF file that contains port information to the `OpenLane` flow so the router can identify the connection points. If the user exports the LEF file from `Magic` directly, the port information includes all the electrically connected points within the macro, leading to a high routing congestion error during integration. Alternatively, the LEF file can be exported using the `-hide` option which conceals most circuit details and retains only the port locations where the labels are created. However, omitting information about layers near the ports can result in DRC violations or shorts.

d) Wells: In the ACT flow, individual cells do not include wells in the layout; instead, `Dali` generates a well layer for all the cells in the design at the top level. However, when `Magic` opens the design, along with cells and the well layer, it interprets them at different levels of hierarchy. At the individual cell level `Magic` interprets there is a pwell area painted where the nwell is at the top level, resulting numerous DRC violations. While flattening the design can resolve this issue, it requires significant computer resources and takes a very long time for complex designs, making it nearly impractical for large tape-outs.

In what follows, we describe how we addressed these challenges using a combination of existing tools, newly developed scripts, and additional functionality that was added to the ACT framework. The open-source scripts we developed can be found on GitHub [12].

III. Design Flow

A. HDL to LEF/DEF

Asynchronous circuits can be developed using the ACT framework, as illustrated in the blue background of Fig. 1. More specifically, they can be described at a high level using CHP (communicating hardware processes), which is a sub-language of ACT. Several tools are available to synthesize a design written in CHP to a gate-level netlist, including `chp2prs`, and `Maelstrom` [13]. In this tape-out, we select `Maelstrom` to obtain better-performance circuits with more efficient control logic.

Given the gate-level netlist, the design can be placed using `Dali`. `PWRoute` [14] is used to generate power mesh for the placement and connect cells' power pins to the nearest straps by running detailed power routing. Once the design is placed and power routed, it can be global routed with `SPRoute` [15], followed by detailed routing with either `TritonRoute` [16] or `Innovus`. The routed design is described using LEF and DEF files.

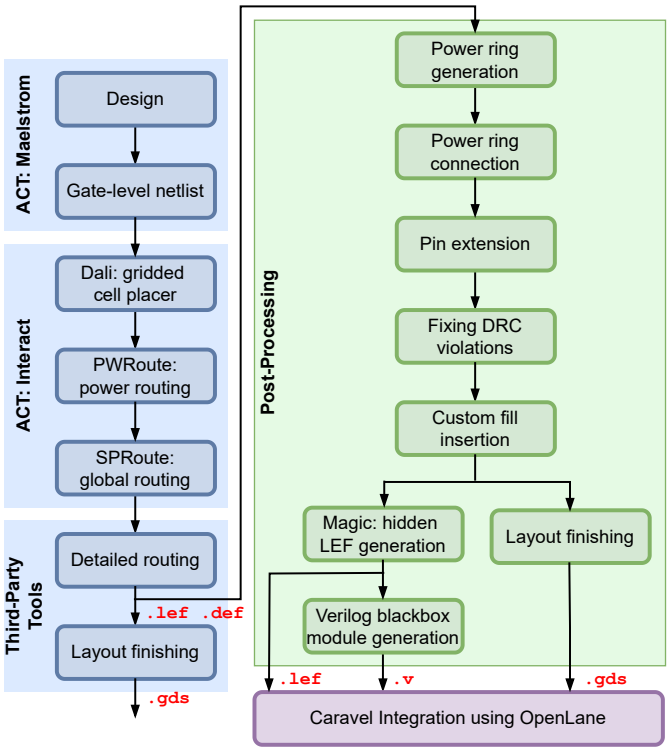

Fig. 1. Overview of the ACT framework and the post-processing flow.

B. Post-processing Routed Design

Prior to integrating the routed design into the Caravel harness using `OpenLane`, you must

- Add a power ring around the design
- Connect the internal power mesh to the power ring
- Extend the I/O pins
- Fix DRC violations added during routing
- Add custom fill to meet minimum density requirements
- Generate a Verilog blackbox module for the asynchronous macro

A summary of these steps is shown highlighted in green in Fig. 1.

1) Power Ring Generation: For some designs where all metal layers are required for routing, the top metal layers become obstructions while power routing when integrating with the `OpenLane` flow. Thus, a power ring is added around each asynchronous design in order to ensure connectivity between the chip's power grid and the local circuit's power connections. To generate a power grid, our flow will find the bounding box coordinates of the circuit in `Magic` and generate a TCL script to draw the power ring based on an input width of the ring, spacing between rings for each power supply, and an offset between the macro and the first power ring loop.

2) Power Ring Connections to Asynchronous Macro: Furthermore, the macro's power grid must be connected to the generated power ring. To do this, we parse the DEF file to find the locations of all power stripes. Separated into vertical and horizontal, they are sorted by their coordinates. For each

vertical stripe, the metal can be extended to connect to the power ring. Similarly, the horizontal power stripes on the left and right edges are also extended.

3) Pin Extension: To include a macro in the OpenLane flow, LEF and GDS for the design must be provided. One of the main uses of the LEF file is to provide information about port locations. There are various options when exporting LEF from `Magic`, but we have found the "hidden" option works best for later integrating into the caravel harness. When using the hidden LEF file, we noticed that the connection to the pins may yield shorts or DRC violations when merging the macro GDS. We combated this issue by extending all of the ports of our macro. Within the generated TCL script, this is done automatically by parsing the LEF file for the port names and locations (to identify which direction to extend the pin).

4) Fixing DRC Violations: For some circuits, a small number of DRC violations arise during routing. We have observed that there are only a few types of DRC violations that appear. It is advisable to fix the few violations after the design is routed. Since many of the violations are of the same type, a short TCL script can be written for each type of error to fix the DRC. Each short script can be integrated into a single script where each DRC violation can be found and the user can select which correction script to run. In the case where a new type of violation is found, the script can be easily adapted to include a custom correction script.

5) Custom Fill: During the final steps of integrating our design in the caravel harness, minimum density DRC violations arose. Since we had access to `Calibre` and the commercial Skywater 130 PDK, we first attempted to run the commercial fill on our design. This did not increase the density for field oxide sufficiently. Thus, we developed custom fill scripts. As `Dali` knows the locations for all of the cells, it also can identify gaps between cells. Respecting the spacing design rules, we can draw a maximally sized fill cell that includes all of the necessary layers to ensure there are not low density DRC violations in the final checks. We modified `Dali` and our `interact` script to export this information so that we could automate fill insertion. In our experience, the required fill cell included diffusion and poly. An image of the generated fill is shown in Fig. 2.

6) Verilog Blackbox Module Generation: To instantiate the asynchronous macro in the user_project_wrapper, the user needs to provide a verilog blackbox module. This is automatically generated by parsing the LEF file to extract the design's name and port information.

7) Magic DRC Check: The default `OpenLane` flow for integrating a macro into the Caravel harness requires that the design be DRC clean. By default, `Magic` is used to check for DRC violations. Due to reading well information from multiple levels of hierarchy, `Magic` identifies numerous DRC errors. Using `Klayout` and `Calibre`, the same GDS is DRC clean. Therefore, in the `OpenLane` flow, the `Magic` DRC checking step should be disabled.

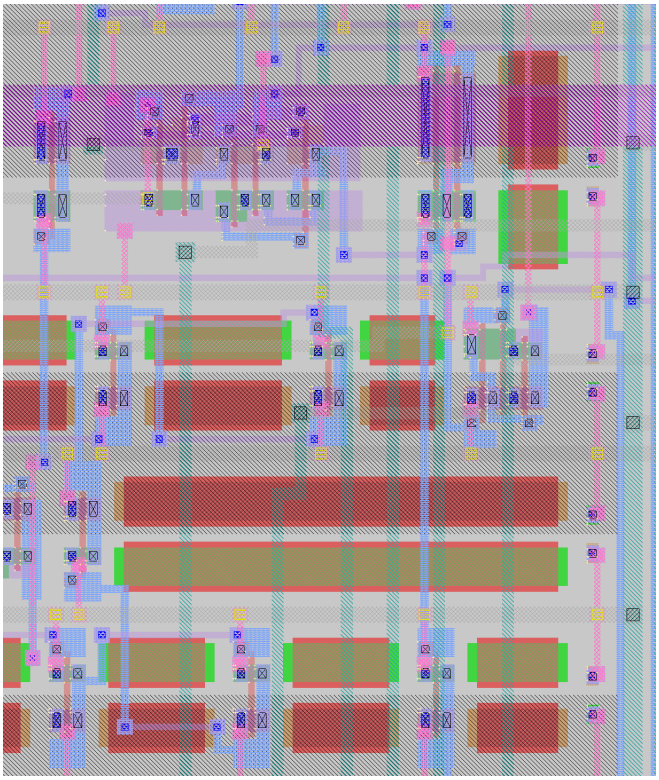

Fig. 2. Fill cells generated by our tool to increase the density of field oxide are shown as the transistor-like disconnected devices between digital cells.

IV. Communication with the Design

Users should determine how to communicate with the design using Caravel harness's utilities. The first method is sending and/or receiving signals through I/O pins. Since our design is asynchronous and capable of handling asynchronous input, no additional logic is needed. However, to mitigate potential antenna violations, users should assign the Caravel ports to the macro's ports with care to minimize wire lengths. Additionally, `Dali` can position the macro's ports according to users' requirements, further reducing the wire length.

The other method is to use the management core in the Caravel chip for data transfers. This method involves clock domain crossing, therefore, we adopt standard two flip-flop synchronizers to avoid metastability issues.

V. Asynchronous MD5 Hasher

Using the process presented above, we have taped-out an asynchronous MD5 hashing accelerator in the SKY130 process and integrated it into the Caravel harness.

VI. Future Developments

Some of our post-processing steps can be integrated with the ACT tools. Specifically, power ring generation and connections can be integrated with `Dali`. Similarly, fill insertion can also be integrated with Dali. The main benefit for doing this is location information is already stored in existing data structures; thus, we will not need to parse LEF/DEF files