A Digital Flow for Asynchronous VLSI Systems: Status Update

Udit Agarwal*, Samira Ataei†, Jiayuan He*, Wenmian Hua†, Yi-Shan Lu*, Sepideh Maleki*, Yihang Yang†, Keshav Pingali*, and Rajit Manohar†

*University of Texas at Austin
{udit,hejy,yishanlu,smaleki,pingali}@cs.utexas.edu

†Yale University
{samira.ataei,wenmian.hua,yihang.yang,rajit.manohar}@yale.edu

***Abstract*—We are developing an open-source EDA flow for asynchronous logic. Key parts of the flow are implemented using the Galois system for parallelization to reduce run-time requirements. We report on the current state of this flow, and some of the issues that we are exploring in order to improve the overall quality of results and expand the class of circuits that can be implemented using the flow.**

I. Introduction

Asynchronous digital circuits provide a method for implementing digital computation without the need for a global clock signal to synchronize all operations. The absence of a global clock is a significant enough change that conventional synchronous design automation tools are insufficient to design correct asynchronous circuits without major manual effort.

To address this challenge, we have been building an open-source design automation flow for asynchronous circuits. The flow includes an open-source design hardware description language dubbed "ACT", which permits the specification of asynchronous circuits at multiple levels of abstraction spanning behavioral descriptions and transistor-level descriptions. The ACT language supports different asynchronous circuit families and timing constraints in a single framework.

We describe the progress we have made over the past year in developing design automation support for asynchronous circuits. We focus on the "gates to GDS2" part of the flow, where a gate-level netlist has to be translated into physical geometry prior to tape-out. The modular nature of the asynchronous design flow makes it amenable to parallelization. Module-level parallelism is easy to achieve, and the higher complexity of some of the design automation steps (relative to synchronous design) requires parallelism to speed up the design process. We are leveraging the Galois framework as a way to simplify the development of highly parallel EDA tools.

II. Current design flow for asynchronous circuits

The flow we are creating includes a design language called ACT (for asynchronous circuit toolkit). This is a hierarchical design language that includes communication channels as first-class objects. By using a single language for multiple levels of abstraction in design, we preserve the relationships between different design phases throughout the flow. A summary of the flow we are developing is shown in Figure 1. We now highlight some of the non-standard aspects of our flow.

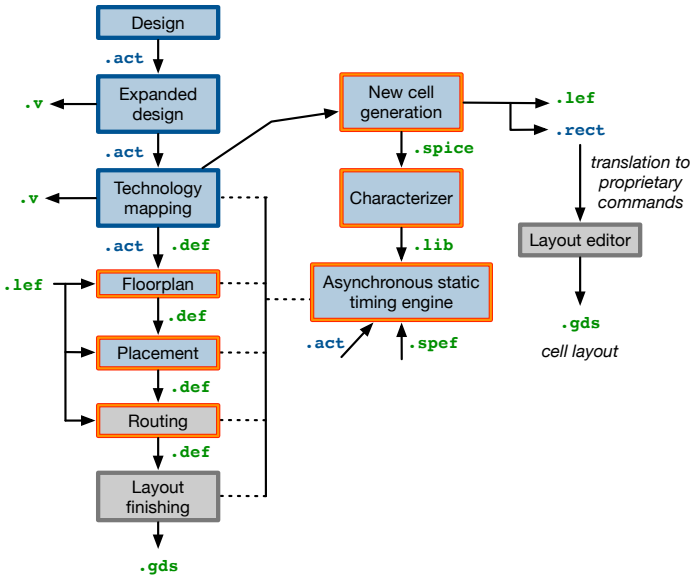

Fig. 1: Overview of the digital design flow for asynchronous circuits that is under development. Gray boxes denote existing tools that we leverage. Orange boxes are the ones where we implement parallel algorithms to improve run-time.

A. Technology mapping and gates

After elaboration, the design consists of a hierarchical description of the asynchronous circuit with the leaves of the description being user-specified gates implemented with pull-up and pull-down networks. Since different asynchronous circuit families require different standard gates, our flow is agnostic to the underlying circuit family used to implement the circuit. Instead, we permit arbitrary circuits to be specified in the ACT language, with the user providing a cell library for implementation.

Given the cell library, we have developed tools to map the design to the specified cell library. When the mapping

fails, a new cell is automatically created. We have also developed preliminary tools for cell creation—including layout of the cell, and characterization for power and performance to generate an updated `.lib` file. Some manual effort is currently needed to ensure that the generated cell layout is DRC/LVS clean, and reducing this effort is ongoing work.

B. Asynchronous static timing analysis

Given a design mapped to a collection of pre-characterized gates, we have developed an asynchronous static timing analysis called Cyclone [5], the first comprehensive timing and power analysis engine capable of handling complete asynchronous circuits. Cyclone, adopting the approach in [6], calculates exact results for asynchronous circuits with max-causality gates, and conservative approximations in all other cases. Cyclone is parallelized using the Galois framework to shorten its turn-around time. Our preliminary experimental results show a speedup in the range of 3× to 19× for large designs by using the Galois framework. We are currently extending the types of asynchronous circuits supported by Cyclone, and further improving the speedup and accuracy of asynchronous timing analysis.

C. Layout generation

We have created tools to convert the ACT design into the industry-standard `.lef`/`.def` formats in preparation for layout generation.

1) Floorplanning: We automate floorplanning for larger designs with a hierarchical min-cut approach. Both layout space and the design are simultaneously partitioned while keeping the ratio of cell area to available space constant. BiPart, a deterministic multi-level min-cut hypergraph partitioner parallelized using the Galois framework, is used for partitioning designs. Internally, BiPart uses compressed sparse row format to represent levels of hypergraphs for better locality; this gives 2× speedup compared to using naive graph representations.

BiPart also lowers the cut size by 15% by introducing more scheduling policies for partitioning a hypergraph. Scheduling policies affect what nodes are merged during the coarsening process in multi-level min-cut partitioning.

2) Placement and routing: For asynchronous circuit layout automation, we propose a gridded cell placement flow, Dali, which allows the cell height and cell width to be any integer multiple of two grid values [8]. The freedom of cell heights can potentially lead to a more compact layout, smaller wire-length, and thus better performance. Dali generates design-rule clean placement. Our experimental results also show that the gridded cell placement approach reduces area by 15% without impacting the routability of the design.

The gridded cell placement flow consists of four stages, which are designed to optimize wire-length cost and satisfy various physical constraints. (a) Global placement optimizes wire-length cost while keeping the cell density less than a given upper limit. (b) Forward-backward legalization removes overlapping among gridded cells, and only makes local cell displacement to avoid large perturbation. (c) N/P-well legalization assigns cells to placement sub-regions, and performs cell clustering for N/P-well generation and well tap cell creation in each sub-region. (d) Power grid design leverages the organized structure of cell clusters to generate a power mesh and connects VDD and GND pins in each cell to their closest power-lines. We are working on detailed placement techniques for gridded cells to further improve the wire-length cost, and integration with Cyclone to make the placement flow timing-driven.

We use commercial routers to route the placed design. However, our approach is agnostic to the choice of detailed router. We plan to integrate SPRoute [4], a global router based on FastRoute [7] and parallelized using the Galois framework, into the layout automation tool chain. We also plan to make routing tools timing-driven by utilizing the feedback from Cyclone.

D. AMC: Asynchronous memory compiler

Almost every digital ASIC and Soc design requires embedded memory, and asynchronous designs are no different. Commercial memory compilers provide only synchronous memories and to address the memory need of asynchronous designs we have built AMC, an asynchronous memory compiler [1, 3]. AMC generates asynchronous pipelined multi-bank SRAM modules with quasi delay-insensitive control and bundled-data datapath. The description of SRAM architecture generated by AMC and the compiler usage can be find in [1, 2]. Recently we have added two new features to AMC: (i) power-gating and (ii) write-masking. In addition, AMC has been successfully ported to a 12nm fabricable FinFET technology using the foundry-provided 6T SRAM cell.

The first new feature is power-gating option which allows to turn off the memory when it is not being used. A CPU cache can be dormant for a long time and power-gating helps to minimize the leakage current of memory by temporarily switching the cache off when it is not required. In addition, leakage power dissipation grows in scaled technology nodes and to address the leakage power of memory, it is highly desirable to provide the power-gating option for memory. Power-gating provides two operation modes for generated asynchronous SRAMs: SLEEP or low power mode and WAKE-UP or active mode. The power-gating option of AMC allows to switch between these two modes to maximize the power savings with minimum impact on performance. Shutting down the SRAM power is scheduled through "SLEEP" control signal. During SLEEP mode, it is necessary to isolate the outputs of memory in order to avoid floating outputs driving inputs of active blocks. Once all output acknowledge signals are correctly isolated, memory goes to a low power mode. Lowering the SLEEP signal, wakes the memory up and returns it to active mode.

We used a coarse grain power gating approach with low area penalty and switch the memory macro power by a collection PMOS switches that gate the VDD supply. Header PMOS switches are a more appropriate choice for memory

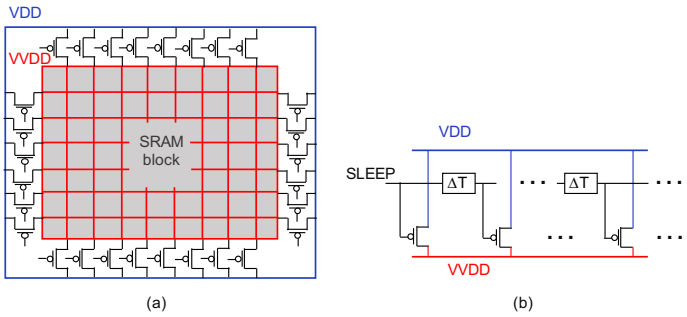

Fig. 2: (a) ring-style power-gating and (b) daisy-chain SLEEP distribution.

block power-gating compared to footer NMOS switches that gates GND. Many SoC and ASIC designs use multiple power supplies with level-shifters and since level shifters are typically designed with a common ground, switching the ground can be a problem in multi-voltage chips. The AMC power-gating is implemented in a ring style network. With a ring style network, memory macro is encapsulated with a ring of PMOS switches that connect VDD to a virtual VDD (VVDD) as shown in figure 2(a). All the switches are added outside the memory macro and memory's VDD power is connected to VVDD.

When memory is reconnected to power supply there is an in-rush current that must be carefully controlled in order to avoid excessive IR drop, spikes on the supply voltage and possible content corruption of other blocks in the chip. To control this in-rush current the control signal (SLEEP) to the PMOS switches is daisy-chained as shown in figure 2(b). The SLEEP signal is connected to a set of switches and then buffered with always-on buffers and send to next set of switches. Daisy chaining the SLEEP signal turns on the sleep transistors gradually and the current increases slowly with the number of turned on transistors to control in-rush current. However, it takes some time from the assertion of activate signal to power up the memory macro. For this reason, an acknowledge signal indicating the memory is powered-up is also provided. This acknowledge signal is the inverted SLEEP signal from the final buffer stage and indicates that memory is completely powered up and memory accesses can be resumed.

The second new feature is write-masking. Write-masking determines the data bits to write during the memory write mode. When write-masking is enabled, data on the input data bus (DIN[n:0]) are written to the memory, as specified by the write mask bus (WM[n:0]). Enable pin of all write drivers are gated with WM pin, making each bit individually selectable. When the write mask pin k is high (WM[k] =1), the corresponding data bit (DIN[k]) is selected, and its data is written to the memory location specified with decoded address bits. If the write mask pin is low, no data is written for that bit and memory cell retains its previous value. To reduce the area overhead when write-masking is enabled, write mask bus is routed over input data bus on a higher metal layer.

E. Overall status

We have used the current version of the flow to successfully tape-out a 65nm asynchronous microprocessor. The tools are available at `http://github.com/asyncvlsi/`. The commercial tools used for the design were: (a) a detailed router; (b) DRC/LVS/extraction for sign-off; (c) and commercial tools for pad frame assembly and routing. We are developing tools to replace (c) with open-source tools, and our plan is to use existing open-source detailed router projects to replace the commercial detailed router.

III. Conclusion

We have presented the current state of a design automation flow for asynchronous circuits. The flow supports modular design, and many of our core tools are written using the Galois framework to support parallel execution on multi-core systems. We believe that this tool chain will promote chip design with clean abstractions and fast turn-around times.

Our two major efforts for the upcoming year are: (a) to combine our efforts on the asynchronous memory compiler with the OpenRAM project, to create a single unified memory compiler that supports both synchronous and asynchronous memories; (b) to generalize our static timing analysis engine to more classes of asynchronous circuits; and (c) to make all the key steps of our flow truly timing driven.

References

[1] S. Ataei and R. Manohar. Amc: An asynchronous memory compiler. *IEEE International Symposium on Asynchronous Circuits and Systems (ASYNC)*, pages 1–8, 2019.

[2] S. Ataei and R. Manohar. AMC: Asynchronous memory compiler. http://avlsi.csl.yale.edu/act/doku.php?id=amc:start, 2019.

[3] S. Ataei and R. Manohar. A unified memory compiler for synchronous and asynchronous circuits. *Workshop on Open-Source EDA Technology (WOSET)*, pages 1–4, 2019.

[4] J. He, M. Burtscher, R. Manohar, and K. Pingali. Sproute: A scalable parallel negotiation-based global router. In *International Conference on Computer-Aided Design*, Nov. 2019.

[5] W. Hua, Y.-S. Lu, K. Pingali, and R. Manohar. Cyclone: A static timing and power engine for asynchronous circuits. In *International Symposium on Asynchronous Circuits and Systems (ASYNC)*, May 2020.

[6] W. Hua and R. Manohar. Exact timing analysis for asynchronous systems. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 37(1):203–216, 2018.

[7] Y. Xu, Y. Zhang, and C. C. N. Chu. Fastroute 4.0: Global router with efficient via minimization. *2009 Asia and South Pacific Design Automation Conference*, pages 576–581, 2009.

[8] Y. Yang, J. He, and R. Manohar. Dali: A gridded cell placement flow. In *International Conference on Computer-Aided Design (ICCAD)*, Nov 2020.