An Open-Source Design Flow for Asynchronous Circuits

Rajit Manohar
Computer Systems Lab
Yale University
New Haven, CT 06520
Email: rajit.manohar@yale.edu

Abstract—**There have been a number of small-scale and large-scale technology demonstrations of asynchronous circuits, showing that they have benefits in performance and power-efficiency in a variety of application domains. Most recently, asynchronous circuits were used in the TrueNorth neuromorphic chip to achieve unprecedented energy-efficiency for neuromorphic systems. However, these circuits cannot be easily adopted, because commercially available design tools do not support asynchronous logic. As part of the DARPA ERI effort, we are addressing this challenge by developing a set of open-source design tools for asynchronous circuits.**

Keywords—**asynchronous circuits; open-source EDA tools**

I. Introduction

Scalable computer systems are designed as a collection of modular components that communicate through well-defined interfaces. The interfaces must be robust to delays and uncertainty in the physical implementation of communication. This view applies to computer systems at many levels of abstraction. The Internet is a collection of communicating computers with message-passing through the Internet protocol. A modern datacenter is a collection of servers that communicate via message-passing over commodity network hardware. Even large software systems consist of a collection of modules that use well-defined application programming interfaces (APIs) to communicate. Almost all computer systems disciplines have made the wise choice to partition their problem into components that communicate via protocols that *are independent of their physical realization—such as timing, energy, or size.*

However, in chip designs today, this modular approach is abandoned in favor of global synchrony. A global synchronization signal (the "clock") dictates the time budget for every step of the computation—regardless of *what* is being computed.

Although this clocked design paradigm dominates the design of computers today, engineers are struggling to preserve the fiction of simultaneity required by the clock, even within an individual chip. This struggle is an inevitable result of advancing technology. As transistors get smaller and faster the delay of communication over wires dominates the cost of local computation with transistors. Such progress renders the clocked paradigm a poorer and poorer abstraction for chip design. Modern application-specific integrated chips (ASICs) are designed as a collection of small clocked "islands" that communicate via interfaces that break the clocking abstraction.

We are developing a collection of electronic design automation (EDA) tools that isolate the designer from the details of the physical implementation technology, especially when it comes to delays and timing uncertainty.[1] The approach is based on an *asynchronous*, *modular* and *hierarchical* design methodology for complex chips,and it permits component re-use from one technology to another with little or no modification. While individual (small) modules of the chip could be clocked, the overall system uses an asynchronous integration approach to achieve modular composition.

II. Motivation

The phrase "asynchronous digital circuits" refers to a large family of circuits that do not use a global clock signal to sequence steps of the computation. There are a wide range of asynchronous logic families, ranging from circuits that are highly robust to timing uncertainty to those that rely on strict timing constraints for correct operation. The absence of a global clock signal has major implications for system design. Since each step of the computation isn't synchronized to a periodic timing signal, a designer can explore a significantly richer design space.

A simple example that illustrates this difference is the design of a decoder in a microprocessor. A decoder can be viewed as a selection tree, where the leaves of the tree are the different instruction classes. We can view the instruction classes as the symbols of an alphabet Σ with some probability distribution over the alphabet determined by instruction frequency. The best clocked implementation of the selection tree is a balanced one—one that minimizes the maximum depth of the tree. This tree has depth $\mathcal{O}(\log|\Sigma|)$, which determines the delay and energy required. However, an asynchronous implementation would use a Huffman tree, with average energy and delay given by $H(\Sigma) \leq \log|\Sigma|$ where $H(\cdot)$ is the information-theoretic entropy of the input distribution. This observation

DISTRIBUTION STATEMENT A. Approved for public release: distribution is unlimited. This research is supported in part by DARPA IDEA grant FA8650-18-2-7850, and in part by DARPA POSH grant HR001117S0054-FP-042.

[1]It is not possible to entirely decouple the logical correctness of a design from timing to create completely delay-insensitive circuits [19], [22]. However, it is possible to make a very mild and local timing assumption that is easy to satisfy in practice [18].

was used in the design of a hierarchical bus in an asynchronous microprocessor to reduce its energy consumption and improve performance of the main execution busses [1].

There are many examples of custom chips designed using asynchronous circuits that demonstrate state-of-the-art performance and/or power consumption. These fabricated chips include a high-performance microprocessor [2], low-power embedded processors [1], [3], high-throughput FPGAs [4], SEU-immune circuits [5], [6], and (most recently) large-scale neuromorphic systems.

Historically neuromorphic chips have used asynchronous digital logic in conjunction with continuous-time analog circuits to emulate neurons, synapses, and axons at vaying levels of detail. We mention three major recent projects in this space that all use asynchronous digital logic for implementing spike-based communication. The 5.4B transistor TrueNorth chip [7] uses asynchronous digital logic to implement its on-chip and chip-to-chip network in a 28nm process. It preserves a deterministic execution model using a hardware spike scheduler in spite of its massively parallel execution architecture [8]. Intel's recently announced Loihi chip also uses asynchronous digital logic in a 14nm process [9]. Loihi includes a flexible routing architecture, and has support for on-chip learning. Most recently, the Braindrop chip uses a mixed-signal approach with analog computation and asynchronous digital communication implemented in a 28nm FDSOI process. Braindrop attains significantly higher energy-efficiency compared to TrueNorth and Loihi after the architectural differences between the various chips are taken into account [10].

Researchers aiming to emulate some of these energy-efficiency results are hard pressed to do so, simply because there are no readily-available design automation tools for asynchronous circuits. As part of the DARPA ERI initiative, we are developing an open-source design flow for asynchronous digital circuits. Our goal is to make asynchronous design accessible to a broad community.

III. EDA for Asynchronous VLSI Systems

While the range of possible asynchronous circuit families are large, they all share a few common features. First, the set of *gates*, or building-blocks, are more general than standard combinational logic and flip-flops. For instance, one of the most commonly used circuit elements is the C-element, which is a state-holding gate whose output changes only when the two inputs are equal—a component not found in a synchronous standard cell library. Second, the performance of these circuits is governed by the delays of *cycles* of gates. A special case where this is clear is one where the circuit contains an odd number of inverters to create an oscillator (the clock), and the oscillator is used to control flip-flops. This is a traditional synchronous circuit, and the performance is governed by the delay of the cycle of inverters. In general, there are many interconnected cycles in an asynchronous circuit, and so the performance analysis problem is very different from the clocked domain. Third, correct operation of an asynchronous circuit requires relative path delay timing constraints on some sets of paths.

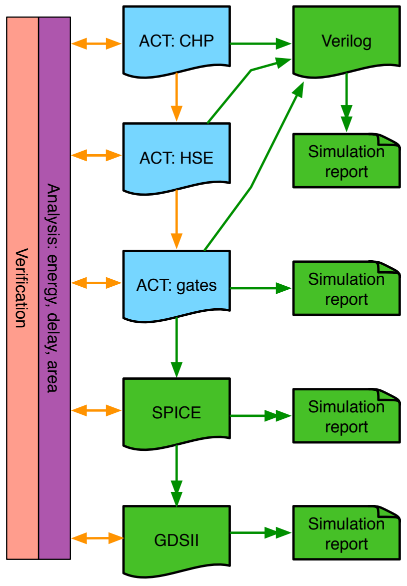

Fig. 1. An overview of the design flow for asynchronous circuits that is under development.

The open-source design flow we plan to create starts with the circuit specified using the Communicating Hardware Processes (CHP) programming language [21]. The language is a simple sequential programming language augmented with communication channels that support send, receive, and probe operations. The next step in the design flow is to decompose the CHP into a parallel collection of processes that cooperate to perform the computation specified by the original program in a hierarchical manner as a sequence of small changes, where each change can be easily verified [23], [20]. After this step, data encoding and protocol choices are made so that every statement in each process can be represented as a collection of operations on Boolean-valued variables; after this step, the program is in handshaking expansion (or HSE) form. Finally, each individual component is converted into a circuit and the concurrent composition of all the circuits implement the original specification [21].

The flow we plan to develop includes a design language called ACT (for Asynchronous circuit toolkit). This is a hierarchical design language that includes communication channels as first-class objects. The language supports representing circuits at multiple levels of abstraction, including CHP, HSE, gate-level, and transistor-level descriptions. By using an integrated language, we preserve the relationships between different levels of abstraction in the design throughout the design flow. Design tools can be viewed as transformations in the ACT framework. For example, logic synthesis elaborates a CHP-level description of a module into a gate-level description of the same module without changing its interface. A summary of the design flow and linkages to standard, commercially used file formats is shown in Figure 1.

This language is the result of an evolution of over almost three decades of research in asynchronous design grounded in the implementation of over a dozen asynchronous VLSI chips ranging in complexity from 0.5M transistors to 5.4B transistors, and in technologies ranging from 0.6μm CMOS to 28nm CMOS. We also plan to develop linkages to and from the ACT language to standard commercially-used languages and formats such as Verilog and SPICE. This will allow interoperability between our flow and commercial flows.

A. *Custom Design*

We have already completed the development of the key components necessary for a custom design flow for asynchronous circuits. This flow supports designing asynchronous digital logic in a manner analogous to the classic "Mead and Conway" digital VLSI class—the first VLSI class offered in the world. While this may seem primitive from the standpoint of a commercial tool flow for synchronous logic, no such open flow was previously available to a designer interested in using asynchronous circuits.

The design flow consists of the following steps and tools. While commercial tools can be used for some of the components, the flow we are building uses only open-source tools.

1) **Logic design** by gate-level design entry. A designer creates custom digital logic gates to implement their desired functionality. This step requires manual entry, which is what makes this a custom flow. A designer specifies pull-up and pull-down networks as Boolean expressions (a notation called *production rules*), to describe arbitrary logic gates. The tool required for this step is a text editor, and the specification of a design language for asynchronous circuits—namely, the ACT language that we have created. ACT provides support for hierarchy, and includes a module system.
2) **Logic simulation.** The gate-level design can be simulated with a logic simulator that we have developed tailored to the needs of asynchronous circuits. Our logic simulator has support for asynchronous-specific requirements, including identifying switching hazards and implementing non-deterministic choice using arbiters/mutex elements. This step is used for simulation-based functional validation of the design. There are two tools required at this step: a converter from ACT into an input format for a simulator, and a digital simulator for asynchronous circuits. We have implemented both these tools.
3) **Circuit optimizations.** The result of circuit optimization by gate sizing can be specified in the ACT language. These circuits can be translated to a standard `.sim/.al` file format. These files can be read by open-source simulators such as `irsim` that include first-order RC delay models. `irsim` is an open-source simulator that has been available for decades, and we have implemented a converter from the ACT language to the `.sim/.al` file format used by `irsim`.
4) **Analog simulation.** More accurate simulations can be performed using an open-source circuit simulator such as `Xyce`, which has been developed by Sandia National Labs [25]. `Xyce` takes a SPICE file as input, and so we created automated translators from ACT to `.spice`. The SPICE format generated by our translator is generic, and has been extensively tested with a number of circuit simulation tools including HSPICE, HSIM, spectre, ngspice, and Xyce.
5) **Layout.** Custom layout is completed using the `magic` VLSI layout editor. We have used this layout editor to tape-out working silicon at process nodes from 0.6μm down to 90nm, even though `magic` does not support modern design rule checking and extraction.[2]
6) **Post-layout simulation.** Extracted layout can be converted into a SPICE format using the built-in `ext2spice` functionality in `magic`. We have developed our own converter that generates Xyce-format files from `magic`'s extract file format.
7) **Layout versus schematic.** Our "schematic" is specified using the ACT production rule language. We provide a tool called `lvp` (layout v/s production rules) that can be used to compare the extracted layout with the original design file. This is used for verifying that the layout matches the digital description of the circuit. This tool, combined with a design rule checker, is used for final sign-off.

There are also some minor tools to simplify tasks such as well plug checking, but these are no different from those necessary for synchronous logic. We remark that the missing piece for a flow of this nature to be completely open-source is an open-source design rule checker that supports modern design rules.

A class where undergraduates and first-year graduate students learned how to design custom asynchronous digital circuits and taped out multi-project wafer chips using the MOSIS VLSI service was taught at Yale in Fall 2018. Students starting the class had only taken an introductory logic design course that covers topics such as basic gates, Karnaugh maps, etc. Students used the tools and flow described above to complete a number of laboratory assignments on asynchronous design. Seven MOSIS Tinychips were taped out at the end of the semester using the MOSIS educational program.

B. *True Digital Design*

We are developing the key components of an open-source EDA flow for asynchronous circuits that implement some of the unique aspects of the design flow. Some of the major components include:

- A cell generator leveraging our previous work in supporting non-standard logic gates [14];
- A tool that can import synchronous logic into an asynchronous framework so as to support synchronous module

[2]We rely on the robustness of asynchronous circuits to reduce the burden on accurate extraction of the layout.

integration based on a previously developed methology [11];
- Timing analysis for asynchronous circuits, based on recently developed theory [13];
- An asynchronous memory compiler based on the openRAM framework [12];
- A timing-aware placement and routing flow that respects the timing constraints required for the correct operation of asynchronous logic extending previous work that only supported a small class of timing requirements [15].

In support of the flow, we also plan to develop formal equivalence checking tools across different levels of abstraction (e.g. CHP versus HSE, HSE versus PRS, etc.) using a combination of traditional model checking and *inverse synthesis*, which attempts to "undo" steps in the design flow [16], [17].

What follows is a notional flow that we envision once we have implemnented the key design tools necessary to support asynchronous design.

1) **Design entry.** A high-level (RTL-level) language is used to specify the digital chip. For users unfamiliar with asynchronous design, a synchronous design can be implemented using Verilog. For users familiar with asynchronous design, they would write their behavioral description in ACT directly, giving them access to a much richer design space than accessible via synchronous design.
2) **Logic synthesis.** For Verilog inputs, synchronous logic synthesis tools would be used to create a Verilog netlist. This netlist would then be translated into a gate-level ACT implementation using our automated conversion tool. For ACT inputs, we would implement a logic synthesis tool to generate a gate-level ACT description.
3) **Floorplanning.** This uses the same approach as in the synchronous logic domain.
4) **Placement and routing.** We would develop custom placement and routing tools that are aware of the timing constraints required for correct operation in the asynchronous circuit domain.
5) **Layout finishing.** This would use the same approach as in the synchronous domain (e.g. fill, IO pads, etc.)

The timing analysis engine we are developing will drive the flow. The memory compiler being created would be invoked as necessary, rather than the standard procedure of requesting memory macros from the foundry.

This flow leverages what it can from standard synchronous flows, while introducing new components for asynchronous circuit design where necessary. Our goal is to have a state-of-the-art flow available in a modern process by the end of the DARPA IDEA program.

IV. A Verified Standard library

Asynchronous digital logic is usually viewed with skepticism by mainstream industry and academia. Hence, almost all the investment in design and development in the (closed) commercial realm has been in synchronous designs.

This disadvantage (the lack of readily available asynchronous components) could be turned into a strength—by taking an open-source approach to design development from the very start. We are embarking on the creation of the "standard ACT library", analogous to the standard template library for the C++ programming language. Our group has a wealth of existing designs to draw from: two microprocessors, multiple neuromorphic chips, digital signal processing hardware, and FPGAs. This means we can start by populating a rich library that can be leveraged by the entire community. Our goal is to have a large open-source collection of hardware that makes it easy for designers to quickly create a customized chip.

As part of the DARPA POSH program, we are developing an FPGA emulation platform for prototyping asynchronous logic on commercial FPGAs, as well as a hardware verification framework for asynchronous design. Asynchronous logic is highly modular, and hence can be verified using some extremely powerful techniques that we have developed [16], [17]. By integrating the verification effort with the standard component library, we believe we can have a design methodology and flow that has verification integrated into it rather than bolted on as an afterthought. This idea is inspired by work on verified software at Digital Equipement Corporation's Systems Research Center in the mid-1990s, where they developed a formally verified systems programming library in Modula-3 using extended static checking [24].

To borrow a phrase from Thomas A. Edison, our goal is to create a design environment and component library where asynchronous chips can be created so easily that only wealthy organizations can afford the large design and verification teams necessary for state-of-the-art synchronous chips.

V. Summary

We are embarking on a journey to create the first open-source design flow for digital asynchronous logic. We have already created tools that can be used for custom digital design, and that have been successfully used by students in a semester-long asynchronous digital VLSI class to tape out chips. We are working toward a true digital flow for asynchronous logic. In addition, we plan to release a verified standard component library for asynchronous design inspired by previous work at the intersection of software engineering and formal methods.

In the spirit of open-source, we welcome participation from the community—either as users of what we develop, or as contributors, or as both! We thank DARPA for supporting our efforts, and for championing open-source hardware as part of its ERI initiative.

References

[1] C. T. O. Otero, J. Tse, R. Karmazin, B. Hill, and R. Manohar, "ULSNAP: An ultra-low power event-driven microcontroller for sensor network nodes," in *15th International Symposium on Quality Electronic Design*. IEEE, 2014, pp. 667–674.

[2] A. 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 of the 17th Conference on Advanced Research in VLSI*, Sep 1997, pp. 164–181.

[3] C. Kelly IV, V. Ekanayake, and R. Manohar, “SNAP: A sensor-network asynchronous processor,” in *Proceedings of the 9th International Symposium on Asynchronous Circuits and Systems*. IEEE, 2003, pp. 24–33.

[4] D. Fang, J. Teifel, and R. Manohar, “A high-performance asynchronous fpga: Test results,” in *Field-Programmable Custom Computing Machines, Annual IEEE Symposium on*. IEEE Computer Society, 2005, pp. 271–272.

[5] S. Ramaswamy, L. Rockett, D. Patel, S. Danziger, R. Manohar, C. W. Kelly, J. Holt, V. Ekanayake, and D. Elftmann, “A radiation hardened reconfigurable fpga,” in *Aerospace conference, 2009 IEEE*. IEEE, 2009, pp. 1–10.

[6] S. Keller, A. J. Martin, and C. Moore, “Dd1: A qdi, radiation-hard-by-design, near-threshold 18uw/mips microcontroller in 40nm bulk cmos,” in *2015 21st IEEE International Symposium on Asynchronous Circuits and Systems (ASYNC)*. IEEE, 2015, pp. 37–44.

[7] 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, B. Brezzo, I. Vo, S. K. Esser, R. Appuswamy, B. Taba, A. Amir, M. Flickner, W. P. Risk, R. Manohar, and D. S. Modha, “A million spiking-neuron integrated circuit with a scalable communication network and interface,” *Science*, vol. 345, no. 6197, pp. 668–673, 2014.

[8] N. Imam, F. Akopyan, J. Arthur, P. Merolla, R. Manohar, and D. S. Modha, “A digital neurosynaptic core using event-driven qdi circuits,” in *Asynchronous Circuits and Systems (ASYNC), 2012 18th IEEE International Symposium on*. IEEE, 2012, pp. 25–32.

[9] M. Davies, N. Srinivasa, T.-H. Lin, G. Chinya, Y. Cao, S. H. Choday, G. Dimou, P. Joshi, N. Imam, S. Jain *et al.*, “Loihi: A neuromorphic manycore processor with on-chip learning,” *IEEE Micro*, vol. 38, no. 1, pp. 82–99, 2018.

[10] A. Neckar, S. Fok, B. V. Benjamin, T. C. Stewart, N. Oza, A. R. Voelker, C. Eliasmith, R. Manohar, and K. Boahen, “Braindrop: A mixed-signal neuromorphic architecture with a dynamical systems-based programming model,” *Proceedings of the IEEE*, vol. 107, no. 1, pp. 144–164, Jan 2019.

[11] F. Akopyan, C. Tadeo, O. Otero, and R. Manohar, “Hybrid synchronous-asynchronous tool flow for emerging vlsi design,” in *IEEE International Workshop on Logic Synthesis*, 2016.

[12] Matthew R. Guthaus, James E. Stine, Samira Ataei, Brian Chen, Bin Wu, and Mehedi Sarwar. Openram: An open-source memory compiler. In *Proc. 35th ICCAD, pages 93:1–93:6, 2016.*

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

[14] Robert Karmazin, Carlos Tadeo Ortega Otero, and Rajit Manohar. celltk: Automated layout for asynchronous circuits with nonstandard cells. In *Asynchronous Circuits and Systems (ASYNC), 2013 IEEE 19th International Symposium on*, pages 58–66. IEEE, 2013.

[15] Robert Karmazin, Stephen Longfield, Carlos Tadeo Ortega Otero, and Rajit Manohar. Timing driven placement for quasi delay-insensitive circuits. In *Asynchronous Circuits and Systems (ASYNC), 2015 21st IEEE International Symposium on*, pages 45–52. IEEE, 2015.

[16] Stephen James Longfield and Rajit Manohar. Inverting martin synthesis for verification. In *Asynchronous Circuits and Systems (ASYNC), 2013 IEEE 19th International Symposium on*, pages 150–157. IEEE, 2013.

[17] Stephen Longfield Jr and Rajit Manohar. Removing concurrency for rapid functional verification. In *Proc. 2014 IEEE/ACM International Conference on Computer-Aided Design*, pages 332–339. IEEE Press, 2014.

[18] Rajit Manohar and Yoram Moses. Analyzing isochronic forks with potential causality. In *Asynchronous Circuits and Systems (ASYNC), 2015 21st IEEE International Symposium on*, pages 69–76. IEEE, 2015.

[19] Rajit Manohar and Yoram Moses. The eventual c-element theorem for delay-insensitive asynchronous circuits. In *Asynchronous Circuits and Systems (ASYNC), 2017 23rd IEEE International Symposium on*, pages 102–109. IEEE, 2017.

[20] Rajit Manohar, Tak-Kwan Lee, and Alain J Martin. Projection: A synthesis technique for concurrent systems. In *IEEE International Symposium on Asynchronous Circuits and Systems*, page 125. IEEE, 1999.

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

[22] Alain J. Martin. The limitations to delay-insensitivity in asynchronous circuits. In William J. Dally, editor, *Sixth MIT Conference on Advanced Research in VLSI*, pages 263–278, 1990.

[23] Alain J Martin. Synthesis of asynchronous vlsi circuits. Technical Report CS-TR-93-28, California Institute of Technology, 1993.

[24] David L. Detlefs, K. Rustan M. Leino, Greg Nelson, and James B. Saxe. Extended Static Checking. SRC Research Report 159, Digital's Systems Research Center, December 1998.

[25] Eric R. Keiter, Heidi K. Thornquist, Robert J. Hoekstra, Thomas V. Russo, Richard L. Shiek, and Eric L. Rankin. Parallel transistor-level circuit simulation. In *Advanced Simulation and Verification of Electronic and Biological Systems*, chapter 1, pp. 1–21, Springer 2011.