From nets to circuits and from circuits to nets

Jordi Cortadella

Department of Computer Science
Universitat Politècnica de Catalunya, Barcelona, Spain

It is a mere coincidence that Petri nets came on stage the same year the author of this paper was born [15]. But it not a coincidence that they had a strong relationship since the early nineties. Petri nets have had a wide-spread use in multiple areas where a computational model capable of expressing concurrency, causality and choice is required. One of these areas is Electronic Design Automation [1], since *hardware is inherently concurrent* [10]. Dataflow systems [11], communication protocols [8], hardware/software co-design [12], etc., are examples of domains where Petri nets have been used for specification, synthesis and verification of electronic systems.

This paper reviews the impact Petri nets in one of the domains in which they have played a predominant role: asynchronous circuits. The paper also discusses challenges and topics of interest for the future.

Minimalist Petri nets

When associating certain semantics to events, Petri nets inherit an interpretation that represents the functioning of a particular system. An event can symbolise the initiation of a task, the arrival of a message, the utilization of a resource, etc. Here is an interesting question:

What is the simplest interpretation one could conceive for a Petri net?

If one bit is the minimum unit of information, changing the state of one bit could be considered the most elementary event. If each bit is implemented by an electronic signal, we end up by having a circuit in which each signal may change its state asynchronously according to a certain behaviour. This is precisely what an *asynchronous circuit* is (see Fig. 1): a set of input signals $(x_1, \ldots, x_n)$, a set of output signals $(y_1, \ldots, y_m)$, a set of components (logic gates) and a protocol that specifies the interaction between the circuit and its environment.

Signal Transition Graphs (STG) [17] are the *minimalist* version of Petri nets that can specify the behaviour of asynchronous circuits. Figure 2 presents an STG specifying a circuit with two input signals (a and b) and one output signal (c). The figure also depicts a possible circuit implementation using a Muller C-element [13]. More precisely, the circuit waits for signals a and b to go high (in any order). After that, the circuit generates a rising transition of c. Next, the circuit waits for a and b to go low and generates a falling transition of c. This is iteratively repeated forever.

Fortunately, the author of this paper had the pleasure to briefly talk about the minimalist nets with Prof. Carl Adam Petri in 2003 (Eindhoven) and the

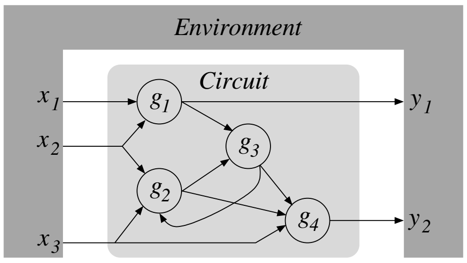

Fig. 1. Asynchronous circuit.

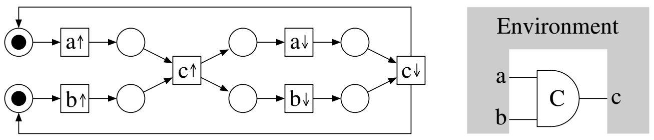

Fig. 2. Signal Transition Graph and implementation.

impact of his contributions in the small community of asynchronous design: the term *Petri net* appears 44 times in the Asynchronous Bibliography[1] [14].

Synthesis and verification

The marriage between Petri nets and asynchronous circuits proposes interesting design automation problems, most of them related to logic synthesis and formal verification. State encoding and logic decomposition have been the most challenging knots in synthesis [5].

Formal verification is also an intricate problem [16] which is computationally expensive when dealing with timed circuits [3]. In this area, Petri net unfoldings have also played a fundamental role when dealing with timed circuits [18].

For most of the synthesis and verification problems, the reachability graph of the system needs to be generated and transformed. But a fundamental problem arises: the system is specified as a Petri net, but the transformations at the level of reachability graph (e.g., insertion of new events) cannot be observed as a Petri net,... unless some method exists to retrieve a Petri net from a reachability graph.

Here is where the theory of regions [9], by Ehrenfeucht and Rozenberg, plays a crucial role in this area. Visualising the transformations of a reachability graph with the same formal model as it was generated is an extremely useful engine to gain intuition about the algorithms used for synthesis. In fact, the theory of regions was the main motivation for creating one of the state-of-the-art tools for synthesis of asynchronous circuits and Petri nets: *petrify* [4]. The tool is not only synthesizing circuits, but also Petri nets. For the authors of the tool, the term

[1] Let us bear in mind that the Asynchronous Bibliography has not been updated since 2004.

petrifying means *returning to the Petri net world* from the reachability graph. The tool is still being used actively in the community of asynchronous circuit designers.

Mining: from circuits to nets

Figure 3 depicts the main problems that can be envisioned when relating specifications and implementations. In the previous section we discussed about synthesis and verification. Process mining [19], also known as process discovery or specification mining, is becoming an area of growing interest in different domains. How can process mining help in asynchronous design?

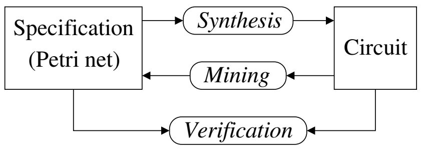

Fig. 3. Interesting problems in asynchronous design.

An interesting problem to solve is as follows: given a circuit, would it be possible to discover environments that can safely interact with it? Moreover, could we describe these environments using Petri nets? This is a kind of reverse engineering problem still in its pre-history, although some initial effort has been recently undertaken [7].

Solving this problem could contribute to improve some of the tasks in circuit design and verification. For example, the discovered specifications could be used to re-synthesise and improve the quality of the circuits. Compositional verification could also benefit by substituting fragments of circuits by their mined specifications (potentially more abstract and simpler than the implementations).

In the area of mining, the recent work by Best and Devillers [2] can be extremely useful since it may help to characterise those environments that preserve certain properties (e.g., persistence) and can still be represented as a Petri net (e.g., a choice-free net). We envision more progress in this direction in the next few years.

The challenge

In the long journey to introduce asynchronous design in industry, we have realized that most designers have an Electrical Engineering background, whereas Petri nets have been mainly used by the academic community with Computer Science background. This creates a cultural gap that makes the adoption of this technology difficult, if not impossible in some cases.

The potential users of this technology are mostly familiar with finite-state machines, waveforms, schematics, and HDLs such as Verilog or VHDL. One of the challenges for the future is to approach designers with a formalism that can have the expressive power of Petri nets (or some subclass of them) while exposing a friendly specification language similar to the formalisms typically used by electrical engineers.

Along this direction, *Waveform Transition Graphs* (WTG) [6] have been recently proposed as an alternative to STGs. In this formalism, the choice-free fragments of the behaviour are represented as waveforms, which is an object very well-known by circuit designers. Additionally, concurrency and choice are mutually exclusive in such a way that choices can only be taken when the system has sequential behaviour. Thus, the choice/join places mimic states of the system that glue the waveforms.

An example is shown in Fig. 4 where the left part represents an STG and the right part represents a WTG. The dotted arc in the STG is required to sacrifice some performance and prevent concurrency during the choice represented by $s0$ in the WTG.

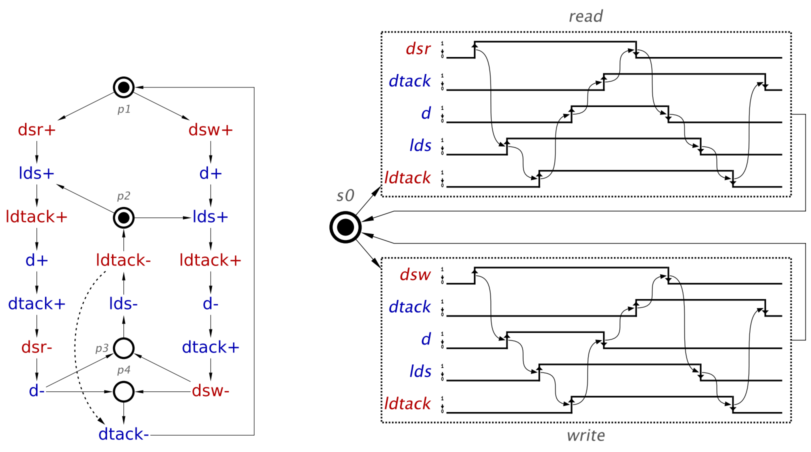

Fig. 4. STG (left) and WTG (right).

Petri nets have played a remarkable role in concurrent hardware and their computational model will prevail in the future. It is now time for academia to create bridges for engineers that enable a broader adoption of Petri nets so that the footprint of Carl Adam Petri's legacy becomes deeper in this area.

Long live Petri nets!

References

1. Design automation of electronic systems: Past accomplishments and challenges ahead. *Proceedings of the IEEE*, 103(11):1941–1943, Nov 2015.
2. Eike Best and Raymond Devillers. Synthesis and reengineering of persistent systems. *Acta Informatica*, 52(1):35–60, February 2015.

3. Robert Clarisó and Jordi Cortadella. Verification of timed circuits with symbolic delays. In *Proc. of Asia and South Pacific Design Automation Conference*, pages 628–633, January 2004.
4. J. Cortadella, M. Kishinevsky, A. Kondratyev, L. Lavagno, and A. Yakovlev. Petrify: a tool for manipulating concurrent specifications and synthesis of asynchronous controllers. *IEICE Transactions on Information and Systems*, E80-D(3):315–325, March 1997.
5. J. Cortadella, M. Kishinevsky, A. Kondratyev, L. Lavagno, and A. Yakovlev. *Logic Synthesis of Asynchronous Controllers and Interfaces.* Springer, 2002.
6. Jordi Cortadella, Alberto Moreno, Danil Sokolov, Alex Yakovlev, and David Lloyd. Waveform Transition Graphs: A designer-friendly formalism for asynchronous behaviours. In *Proc. International Symposium on Advanced Research in Asynchronous Circuits and Systems*, May 2017.
7. Javier de San Pedro, Thomas Bourgeat, and Jordi Cortadella. Specification mining of asynchronous controllers. In *Proc. International Symposium on Advanced Research in Asynchronous Circuits and Systems*, pages 107–114, May 2016.
8. Michel Diaz. Modeling and analysis of communication and cooperation protocols using Petri net based models. *Computer Networks*, 6(6):419 – 441, December 1982.
9. A. Ehrenfeucht and G. Rozenberg. Partial 2-structures; part I: Basic notions and the representation problem, and part II: State spaces of concurrent systems. *Acta Informatica*, 27:315–368, 1990.
10. Michael Kishinevsky, Alex Kondratyev, Alexander Taubin, and Victor Varshavsky. *Concurrent Hardware: The Theory and Practice of Self-Timed Design.* Series in Parallel Computing. John Wiley & Sons, 1994.
11. Edward A. Lee and David G. Messerschmitt. Synchronous data flow. *Proceedings of the IEEE*, 75(9):1235–1245, September 1987.
12. Paulo Maciel, Edna Barros, and Wolfgang Rosenstiel. A Petri Net Model for Hardware/Software Codesign. *Design Automation for Embedded Systems*, 4(4):243–310, 1999.
13. David E. Muller and W. S. Bartky. A theory of asynchronous circuits. In *Proceedings of an International Symposium on the Theory of Switching*, pages 204–243. Harvard University Press, April 1959.
14. Ad Peeters. The 'Asynchronous' Bibliography (BIBTEX) database file `async.bib`. http://www.win.tue.nl/async-bib/doc/async.bib.
15. Carl Adam Petri. Kommunikation mit Automaten. Dissertation, Schriften des IIM 2, Rheinisch-Westfälisches Institut für Instrumentelle Mathematik an der Universität Bonn, Bonn, 1962.
16. I. Poliakov, A. Mokhov, A. Rafiev, D. Sokolov, and A. Yakovlev. Automated Verification of Asynchronous Circuits Using Circuit Petri Nets. In *2008 14th IEEE International Symposium on Asynchronous Circuits and Systems*, pages 161–170, April 2008.
17. L. Y. Rosenblum and A. V. Yakovlev. Signal graphs: from self-timed to timed ones. In *Proceedings of International Workshop on Timed Petri Nets*, pages 199–207, Torino, Italy, July 1985. IEEE Computer Society Press.
18. Alexei Semenov and Alex Yakovlev. Verification of asynchronous circuits using time Petri-net unfolding. In *Proc. ACM/IEEE Design Automation Conference*, pages 59–63, 1996.
19. Wil M. P. van der Aalst. *Process Mining: Discovery, Conformance and Enhancement of Business Processes.* Springer Publishing Company, Incorporated, 1st edition, 2011.