Asynchronous Design—Part 1: Overview and Recent Advances

Steven M. Nowick
Columbia University

Montek Singh
University of North Carolina at Chapel Hill

Editor's notes:
An asynchronous design paradigm is capable of addressing the impact of increased process variability, power and thermal bottlenecks, high fault rates, aging, and scalability issues prevalent in emerging densely packed integrated circuits. The first part of the two-part article on asynchronous design presents a chronicle of past and recent commercial advances, as well as technical foundations, and highlights the enabling role of asynchronous design in two application areas: GALS systems and networks-on-chip.
—*Partha Pratim Pande, Washington State University*

■ **There has been a** continuous growth of interest in asynchronous design over the last two decades, as engineers grapple with a host of challenging trends in the current late-Moore era. As highlighted in the International Technology Roadmap for Semiconductors (ITRS), these include dealing with the impact of increased variability, power and thermal bottlenecks, high fault rates (including due to soft errors), aging, and scalability issues, as individual chips head to the multibillion-transistor range and many-core architectures are targeted.

While the synchronous, i.e., centralized clock, paradigm has prevailed in industry for several decades, asynchronous design—or the use of a hybrid mix of asynchronous and synchronous components—provides the potential for "object-oriented" distributed hardware systems, which naturally support modular and extensible composition, on-demand operation without extensive instrumented power management, and variability-tolerant design. As highlighted by the ITRS report, it is therefore increasingly viewed as a critical component for addressing the above challenges.

This article aims to provide both a short historical and technical overview of asynchronous design, and also a snapshot of the state of the art, with highlights of some recent exciting technical advances and commercial inroads. It also covers some of the remaining challenges, as well as opportunities, of the field.

Asynchronous design is not new: some of the earliest processors used clockless techniques. Overall, its history can be divided into roughly four eras. The early years, from the 1950s to the early 1970s, included the development of classical theory (Huffman [1], Unger [1], McCluskey, Muller [2]), as well as use of asynchronous design in a number of leading commercial processors (Iliac, Iliac II, Atlas, MU-5) and graphics systems (LDS-1). The middle years, from the mid 1970s to early 1980s, were largely an era of retrenchment, with reduced activity, corresponding to the advent of the synchronous VLSI era. The mid 1980's to late 1990's represented a revival or "coming-of-age" era, with the beginning of modern methodologies for asynchronous controller and pipeline design, initial computer-aided design (CAD) tools and optimization techniques, the first academic microprocessors

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.
Digital Object Identifier 10.1109/MDAT.2015.2413759
Date of publication: 16 March 2015; date of current version: 04 June 2015.

(Caltech, University of Manchester, Tokyo Institute of Technology), and initial commercial uptake for use in low-power consumer products (Philips Semiconductors) and high-performance interconnection networks (Myricom).

The modern era, from the early 2000s to present, includes a surge of activity, with modernization of design approaches, CAD tool development and systematic optimization techniques, migration into on-chip interconnection networks, several large-scale demonstrations of cost benefits, industrial uptake at leading companies (IBM, Intel) as well as startups, and application to emerging technologies (sub-/near-threshold circuits, sensor networks, energy harvesting, cellular automata). The approaches in the modern era bear little resemblance to some of the simple asynchronous examples found in older textbooks.

This article is divided into two parts. Part 1 begins with a chronicle of past and recent commercial advances, and highlights the enabling role of asynchronous design in several emerging application areas. Two promising application domains are covered in more detail—GALS systems and networks-on-chip—given their importance in facilitating the integration of large-scale heterogeneous systems-on-chip. Finally, several foundational techniques are introduced: handshaking protocols and data encoding, pipelining, and synchronization and arbitration. Part 2 focuses on methodologies for the design of asynchronous systems, including logic- and high-level synthesis; tool flows for design, analysis, verification and test; as well as examples of asynchronous processors and architectures.

Applications

Asynchronous design has successfully migrated into commercial products from leading companies in recent years. In addition, there have been a number of industry experiments using asynchronous design that were quite successful, even though products did not appear on the market. There are also several exciting emerging application areas where asynchronous design is expected to play a key enabling role.

Commercialization

There have been several promising examples of commercialization of asynchronous designs over the last decade or two, with significant cost benefits demonstrated.

Philips Semiconductors: low-power embedded controllers. In the late 1990s through early 2000s, Philips Semiconductors (now NXP) achieved much commercial success with its asynchronous 80C51 microcontroller [3]. The chip was initially aimed for use in pager chipsets, and the motivation was to lower electromagnetic interference (EMI) noise emissions so that the microcontroller could operate harmoniously with the radio-frequency (RF) data link, without the use of shielding. As a result, encoding and decoding of RF data could now be performed in software instead of requiring a custom fixed-function circuit, allowing easier upgrades to functionality, which was not possible with their synchronous version. It also demonstrated a 4× power reduction over a comparable synchronous design in the same technology. The microcontroller was later used in smart cards for public transport, where the wide range of operating voltages allowed the cards to be battery-free and contactless, powered merely by the brief burst of charge induced when the user's hand waves it through the magnetic field of the card reader. An enhanced version of the asynchronous microcontroller (SmartMX) is now used in more than 75 countries, including the European Union and more recently the United States, for biometric passports and IDs. At last count, the number of copies sold has exceeded 700 million.

Intel/Fulcrum Microsystems: Ethernet switch chips. In 2011, Intel acquired Fulcrum Microsystems, an asynchronous startup producing high-speed networking chips, in a move industry analysts regarded as a bid to compete with Cisco Systems. Intel's current FM5000/FM6000 family of switch chips, which supports industry-leading 40 gigabit Ethernet, includes a fully-asynchronous high-speed crossbar switch that provides high bandwidth, low latency, support for flexible link topologies, and high energy efficiency. The crossbar bandwidth of over 1 terabit per second (in a 130 nm process) is achieved through fine-grain asynchronous pipelining, at the granularity of individual gates, unencumbered by a rigid clock period [4]. When operated at below peak throughput, these chips are highly energy-efficient, since the asynchronous logic provides the benefit of automatic power-down of inactive circuitry without additional instrumentation. This combination of features is considered unique in the marketplace.

Achronix: high-performance FPGA's. Another example of asynchronous commercialization is the Speedster 22i family of FPGAs by Achronix Semiconductor [5]. Manufactured in 22 nm, these chips can operate at 1.5 GHz, and are currently claimed as the world's fastest FPGA's, yet they incur a fraction of the design cost and operating energy cost of leading synchronous designs. Their key to achieving fast operation is the use of asynchronous fine-grain bit-level pipelines, thereby relaxing the constraints of global synchronization. This asynchronous internal implementation is transparent to the end user: the applications mapped can be synchronous, just as on any typical FPGA.

IBM: TrueNorth neuromorphic computer. There has been much excitement recently about neuromorphic computing, which seeks to mimic the functioning of the human brain by using massively-parallel computer systems. In a departure from the traditional von Neumann architecture, these systems employ a highly-distributed memory that is tightly integrated with a large number of parallel computational elements that model neurons. Due to the spatially-distributed nature of computation and communication, along with wide timing unpredictability of data events, neuromorphic computing fits well with the asynchronous paradigm. One interesting example is TrueNorth [6], released in August 2014, which is the largest chip ever developed at IBM, with 5.4 billion transistors. It integrates 4096 neurosynaptic cores on a single chip, modelling 1 million neurons and 256 million synapses. This scale of integration poses a formidable physical design challenge, which was successfully met using fully-asynchronous processing elements and interconnection network. The event-driven asynchronous operation facilitates extremely low power—70 mW for real-time operation over the entire chip—which would be extremely difficult to obtain with current synchronous techniques. Other neuromorphic computers—University of Manchester's SpiNNaker machine (Furber group) and Stanford University's Neurogrid (Boahen group)—also use fully-asynchronous communication networks.

Other commercial designs. Several other industrial applications have explored asynchronous benefits. In the early 1990s, two commercial processors from HaL Computer Systems used a self-timed floating-point divider [7], which was reported as 2–3.5 times faster than the leading commercial synchronous dividers. At Sun Microsystems, now Oracle, high-speed asynchronous pipelines were used commercially in UltraSPARC IIIi computers for smoothing out timing discrepancies in the interface to ultra-fast memories. Theseus Logic has used a NULL convention logic (NCL) methodology to develop chips that are robust to extreme variations in manufacturing and operating conditions [8]. Octasic Inc. has recently developed a clockless DSP technology that takes advantage of the highly-variable data-dependent execution times of different arithmetic operations to achieve a 3× throughput increase over comparable synchronous implementations. Finally, Tiempo IC has developed low-power robust microprocessors, and also has exploited another intrinsic property of asynchronous circuits—the lack of a coherent power or electromagnetic emission signature—to develop chips with secure cryptographic functionality that have increased resilience to side-channel attacks.

Industry experiments

There have also been several industry experiments with asynchronous design that, though quite successful, did not appear in commercial products.

Intel RAPPID. In this experimental project at Intel in the mid-1990's, an asynchronous implementation of the IA32 instruction-length decoder was undertaken because of severe performance bottlenecks that could not be overcome in their commercial version using synchronous techniques [9]. The project focused on making the decoding of the length of the most common CISC instructions fast by exploiting concurrency at sub-clock-period granularity, thereby significantly outperforming the existing synchronous Intel implementation: three times higher throughput, comparable area, half the latency, and half the power.

IBM FIR filter. At IBM Research, a project was undertaken jointly with Columbia University to develop a mixed synchronous-asynchronous implementation of a finite impulse response (FIR) filter for use in the read channels of modern disk drives [10]. The goal was to reduce the filter's latency over its wide range of operating frequencies. In a synchronous implementation that is deeply pipelined for speed, the latency becomes poorer when the data rate, and hence the clock recovered from it, slows down. The hybrid synchronous/asynchronous implementation replaced the core of the filter with an asynchronous pipelined unit featuring a fixed latency, while the remaining circuitry was kept synchronous. The resulting chip exhibited a 50% reduction in worst-case latency, along with a 15% throughput improvement, over IBM's leading commercial clocked implementation in the same technology.

Emerging application areas

Beyond more classical design targets, a number of novel application areas have recently emerged where asynchronous design is poised to make an impact.

Large-scale heterogenous system integration. In multi- and many-core processors and systems-on-chip (SoC's), some level of asynchrony is inevitable in the integration of heterogeneous components. Typically, there are several distinct timing domains, which are glued together using an asynchronous communication fabric. There has been much recent work on asynchronous and mixed synchronous-asynchronous systems (see "GALS Systems" and "Networks-on-Chip" sidebars).

Ultra-low-energy systems and energy harvesting. Asynchronous design is also playing a crucial role in the design of systems that operate in regimes where energy availability is extremely limited. In one application, Liu and Rabaey demonstrated the benefits of asynchrony for sub-threshold operation, with circuits consuming 32% lower energy than a sub-threshold synchronous counterpart [11]. In another application, a collaboration with Oticon Inc., a leading hearing-aid manufacturer, Nielsen and Sparsø proposed an IFIR filter that achieves a 5× power savings over a commercial synchronous counterpart by dynamically adapting the numerical range of the arithmetic circuitry to each individual sample, since most audio samples are numerically small [12]. Such fine-grain adaptation, in which the datapath latency can vary subtly for each input sample, is not possible in a fixed-rate synchronous design. In a recent in-depth case study by Chang et al., focusing on ultra-low-energy 8051 microcontroller cores with voltage scaling, it was shown that under extreme process, voltage, and temperature (PVT) variations, a synchronous core requires its delay margins to be increased by a factor of 12×, while a comparable asynchronous core can operate at actual speed [13]. Finally, Christmann et al. have designed an energy harvesting approach to implement an autonomous sensing application, with a reported 40% power-efficiency gain over synchronous approaches [14]. The use of asynchronous logic provided greater energy efficiency not only due to its event-driven nature, but also by allowing graceful adaptivity to the highly-variable power availability.

Continuous-time digital signal processors (CT-DSP's). Another intriguing direction is the development of continuous-time digital signal processors, where input samples are generated at irregular rates by a level-crossing analog-to-digital converter, depending on the actual rate of change of the input's waveform. An early specialized approach, using finely discretized sampling, demonstrated a 10× power reduction in a speech processing application [15]. The first general-purpose continuous-time DSP was recently proposed by Vezyrtzis et al. [16]; unlike synchronous DSPs, it maintains its frequency response intact over varying sample rates and can support multiple input formats without any internal change, eliminates all aliasing, and demonstrates a signal-to-error ratio for certain inputs which exceeds that of clocked systems. The fine-grain asynchronous processing of irregular sampling is fundamental to the operation of these systems, and would not be possible to support by conventional synchronous techniques.

Handling extreme environments. Asynchronous approaches have also been explored to handle extreme environments, such as for space missions, where temperatures can vary widely. For example, an asynchronous 8-bit data transfer system has been designed that is fully operational over a 400 °C temperature range, from −175 °C to +225 °C, using high-temperature SOI technology [17]. The implementation also shows good resilience to single-event transients (SET's).

Alternative computing paradigms. Finally, there is increasing interest in asynchronous circuits as the organizing backbone of systems based on emerging computing technologies, such as cellular nano-arrays [18] and nanomagnetics [19], where highly-robust asynchronous approaches are crucial to mitigating timing irregularities, both layout-induced and those resulting from the vagaries of quantum behavior. Asynchronous approaches were also shown to be a good match for flexible electronics, in the Seiko/Epson ACT11 microprocessor [20], where the physical bending and manipulation of the material can introduce unpredictable and large delay variations.

Summary

While the above applications are in a wide variety of areas, they exhibit a few commonly-recurring themes, representing beneficial opportunities for asynchrony:

1) *Extreme fine-grain pipelining*: the ability to implement and exploit extremely fine-grain—even gate- and bit-level—pipelines, unconstrained by the need to distribute a high-speed fixed-rate clock [Intel/Fulcrum, Achronix, HaL, Sun Microsystems];
2) *Data-dependent completion times*: to support and micro-architect systems which can exploit subtle and fine-grain differences in data-dependent completion time, i.e., at a subcycle granularity [Intel RAPPID, Oticon IFIR, Ocstasic];
3) *Avoiding challenges due to the rigidity of global timing*: supporting new computation paradigms [CT-DSPs, cellular nano-arrays, nanomagnetics, flexible electronics], extreme micro-parallelism [Intel RAPPID], dynamic computational adaptivity [IBM FIR], and ease of large-scale system integration [GALS, NoCs, neuromorphic computing];
4) *Robustness to voltage, temperature and process variation:* allowing flexible accommodation of dynamic timing variations [energy harvesting, subthreshold computing, space applications]; and
5) *On-demand, i.e. event-driven, operation*: highly energy-proportional computing, without the need for extensive instrumentation of clock-gating at multiple design levels [nearly all applications].

While the above themes capture promising opportunities, and the current industrial uptake indicates increasing commercial viability and interest, there remain open issues and challenges that asynchronous design must overcome to gain wider industry adoption. A brief discussion appears in the conclusion to Part 2.

Foundations

Asynchronous systems are typically organized as a set of components which communicate and synchronize using handshaking channels. These channels are defined by two key parameters: *communication protocol* and *data encoding*. Building on these fundamentals, complex asynchronous systems can be modularly constructed. In particular, we review how these techniques can be used to construct asynchronous pipelines, which are building blocks for many high-performance systems. We then address two basic issues in assembling asynchronous components in larger systems: *synchronization*, which is required when interfacing asynchronous and synchronous domains; and *arbitration*, which is needed to allow the safe competition of multiple asynchronous components for a shared resource.

Classes of asynchronous circuits

Asynchronous circuits fall into several distinct classes, depending on the degree of timing robustness assumed in their operation. This spectrum typically defines a robustness-performance space, where the more robust circuits tend to have lower performance (but not always!). *Delay-insensitive circuits* operate correctly regardless of gate and wire delays. *Quasi-delay insensitive (QDI) circuits* [4], [5], [21] operate correctly assuming arbitrary gate delays, but all wires at each fanout point must have roughly equal delays, i.e., an *isochronic fork* assumption. *Speed-independent (SI) circuits* [2] (an earlier term) assume arbitrary gate delays, but all wire delays are assumed to be zero. Other asynchronous circuits require additional timing constraints, including hold time constraints (i.e., "fundamental mode" [1]) on controllers, one-sided "bundled data" [3], [9], [22]–[24] timing constraints on datapaths (see below), as well two-sided constraints (i.e. both short- and long-path) [25].

Protocols and data encoding

The basic structure of an asynchronous communication channel, between a sender and receiver, is shown in Figure 1. Ignoring data transmission for now, the channel is typically implemented by two wires: *req* and *ack*.

Two common handshaking protocols are used to define a single communication transaction, as illustrated in Figure 2: 1) a *four-phase protocol* (*return-to-zero* [*RZ*]), and 2) a *two-phase protocol* (*non-return-to-zero* [*NRZ*], also known as *transition-signalling*). In a four-phase protocol, *req* and *ack* signals are initially at zero. The sender initiates a transaction by asserting *req*, and the receiver

GALS Systems

An alternative to constructing fully-asynchronous systems is a hybrid approach, integrating synchronous components (i.e. cores, memories, accelerators, I/O units, etc.) using an asynchronous communication network, which together form a *globally-asynchronous locally-synchronous (GALS) system*.[1-3] For some applications, this approach provides the best of both worlds: allowing the design reuse of synchronous intellectual property (IP) blocks, while combining them with flexible asynchronous interconnect as a global integrative medium. The elimination of fixed-rate global clocking on the communication network can provide a highly-scalable, low-power and robust mechanism for assembling complex systems.

A GALS approach was first published in 1980 by Chuck Seitz,[4] in an influential overview of asynchronous design; the approach was used even earlier by Evans & Sutherland Computer Corp. in its first commercial graphics system, LDS-1 [1969]. The term was later introduced and formalized by Chapiro.[1] Synchronous components interact with the asynchronous network using either asynchronous/synchronous interface circuits or pausible clocks.[2,3] Fig. A shows a simplified GALS system with four synchronous cores. Each core can operate as a separate voltage-frequency island (VFI), which is connected to a switch (SW) of an asynchronous communication network through an associated network interface (NI). A number of successful GALS multicore architectures have been developed,[2-3,5-7] including those supporting fine-grain message passing[8] and latency-insensitive communication.[9]

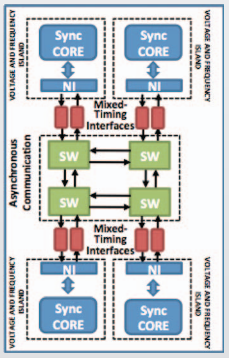

Fig A. A multicore GALS system

As a recent example, STMicroelectronics' Platform 2012 (P2012)[10] includes a fully-asynchronous network-on-chip,[11] supporting a highly-reconfigurable accelerator-based many-core GALS architecture which facilitates fine-grain power, reliability and variability management. The first prototype delivered 80 GOPS performance with only 2W power consumption. It has evolved recently into the company's STHORM Platform.

Interesting specialized applications which benefit from a GALS architecture include large-scale neuromorphic systems (see "Applications" section), as well as approaches to enhance resilience to side-channel attacks.[12] Another interesting approach with a GALS-like flavor is "proximity communication," which aims to overcome the latency bottleneck of inter-chip communication by exploiting capacitive coupling at their interfaces, instead of using traditional wired interconnect.[13]

Overall, there is a surge of interest and activity in GALS design, in both industry and academia, as systems become larger-scale and more heterogeneous, and variability and timing unpredictability become critical factors.

As an addendum, it is worth noting that the term "GALS" has at times been stretched to describe non-GALS systems. In its original and widely-used sense,[1,3,4] a GALS system includes a *fully-asynchronous* interconnection network, including handshaking channels, to integrate synchronous components. Systems containing multiple synchronous cores, operating at different clock rates, which are directly connected using synchronizers, e.g. bi-synchronous FIFO's, are more properly referred to as *multi-synchronous* systems.[7]

A useful general classification scheme for mixed-timing systems was proposed by Messerschmitt,[3,14] based on the relationship between different clock domains. In a *mesochronous* system, synchronous components operate at exactly the same frequency, but with unknown yet stable phase difference.[15] In a *plesiochronous* system, synchronous components operate at the same nominal frequency, but may have a slight frequency mismatch, e.g. a few parts per million. Finally, in a *heterochronous* system, synchronous components can operate at arbitrary unrelated frequencies.

References

1. D. M. Chapiro, "Globally-asynchronous locally-synchronous systems," Ph.D. dissertation, Dept. Comput. Sci., Stanford Univ., Stanford, CA, USA, 1984.
2. M. Krstic et al., "Globally asynchronous, locally synchronous circuits: Overview and outlook," *IEEE Des. Test*, vol. 24, no. 5, pp. 430–441, 2007.
3. P. Teehan, M. Greenstreet, and G. Lemieux, "A survey and taxonomy of GALS design styles," *IEEE Des. Test*, vol. 24, no. 5, pp. 418–428, 2007.
4. C. L. Seitz, ''System Timing,'' in *Introduction to VLSI Systems*, C. A. Mead and L. A. Conway, Eds. Addison-Wesley, 1980, pp. 218–262.
5. J. Muttersbach, T. Villiger and W. Fichtner, "Practical design of globally-synchronous locally-asynchronous systems," in *Proc. 6th IEEE Int. Symp. Adv. Res. ASYNC*, 2000, pp. 52–59.
6. S. Moore et al., "Point to point GALS interconnect," in *Proc. 8th IEEE Int. Symp. ASYNC*, 2002, pp. 69–75.
7. A. Sheibanyrad, A. Greiner, and I. Miro-Panades, ''Multi-synchronous and fully asynchronous NoCs for GALS architectures,'' *IEEE Des. Test*, vol. 25, no. 6, pp. 572–580, 2008.
8. N. J. Boden et al., ''Myrinet: A gigabit-per-second local area network,'' *IEEE Micro*, vol. 15, no. 1, pp. 29–36, Jan./Feb. 1995.
9. M. Singh and M. Theobald, "Generalized latency-insensitive systems for single-clock and multi-clock architectures," in *Proc. ACM/IEEE DATE*, 2004, pp. 1008–1013.
10. L. Benini et al., "P2012: building an ecosystem for a scalable, modular and high-efficiency embedded computing accelerator," in *Proc. ACM/IEEE DATE*, 2012, pp. 983–987.
11. Y. Thonnart, P. Vivet, and F. Clermidy, "A fully-asynchronous low-power framework for GALS NoC integration," in *Proc. ACM/IEEE DATE*, 2010, pp. 33–38.
12. R. Soares et al., "A robust architectural approach for cryptographic algorithms using GALS pipelines," *IEEE Des. Test*, vol. 28, no. 5, pp. 62–71, 2011.
13. D. Hopkins et al., "Circuit techniques to enable 430 Gb/s /mm2 proximity communication," in *Proc. IEEE ISSCC*, 2007, pp. 368–369.
14. D. G. Messerschmitt, ''Synchronization in digital system design,'' *IEEE J. Sel. Areas Commun.*, vol. 8, no. 8, pp. 1404–1419, Oct.1990.
15. M. R. Greenstreet, "Implementing a STARI Chip," in *Proc. ICCD*, 1995, pp. 38–43.

Networks-on-Chip

Over the last decade, networks-on-chip (NoCs) have become the de facto standard approach for structured on-chip communication, both for lowpower embedded systems and high-performance chip multi-processors.[1] These on-chip networks typically replace traditional ad hoc bus-based communication with packet switching, and can be targeted to a variety of cost functions (faulttolerance, power, latency, saturation throughput, quality-of-service [QoS]) and parameters (networktopology, channel width, routing strategies).

Since the NoC approach separates the communication infrastructure, and its timing, from processing elements, it is a natural match for an asynchronous paradigm. Asynchronous interconnect eliminates the need for global clock management across a large network, thereby providing better support for scalability, timing robustness and low power, and avoids the challenge of instrumenting complex clock-gating in a highly distributed communication structure.

A number of asynchronous and GALS NoCs have been proposed in the last decade or so. An early approach, Chain,[2] used delay-insensitive codes on channels for crosstalk mitigation and ease of physical design, and was effectively applied to an ARM-based smart-card chip. Several approaches to support QoS have been proposed, including combining guaranteed service (GS) and best effort (BE) traffic,[3] as well as multiple service levels.[4] A comprehensive asynchronous NoC framework has been developed to provide dynamic voltage and frequency scaling (DVFS) and fine-grain power management,[5] while other approaches have targeted fault tolerance[6] and arbitration for high-radix switches.[7] Automated design flows are also being developed,[8,9] leveraging commercial synchronous CAD tools, which use directives to meet asynchronous path timing and latch constraints, as well as judicious control of optimization modes to avoid the introduction of hazards. A recent asynchronous time-division-multiplexed (TDM) NoC demonstrates correct operation without any global synchronization, while tolerating significant skews on different network interfaces.[10]

Power and performance benefits of asynchronous NoCs have been demonstrated for high-performance shared-memory chip multi-processors[11] and Ethernet switch chips,[12] as well as their facilitation of extreme fine-grain power management and flexible integration of many-core GALS architectures (see STHORM processor discussion in "GALS" sidebar). The end-to-end latency benefits of asynchronous NoCs over synchronous NoCs have also been demonstrated,[8-9,11-12] due to the low forward latency of individual asynchronous router nodes, and the ability of packets to advance without continual alignment to a global clock.

As a recent example, an asynchronous NoC switch architecture,[9] using single-rail bundled data and two-phase communication, obtained a 45% reduction in average energy-per-packet and 71% area reduction compared to a highly-optimized synchronous single-cycle NoC switch, xpipes Lite, in the same 40nm technology. Additional latency benefits have been obtained using low-overhead early arbitration techniques.[13]

One interesting emerging domain where asynchronous and GALS NoCs have played a key role, is in the development of neuromorphic chips (see "Applications" section). These rely on the scalability and ease-of-integration of asynchronous interconnect, and the inherent event-driven operation for low power. One recent example, IBM's TrueNorth, integrates 4096 neurosynaptic cores on a single chip, which models 1 million neurons and 256 million synapses, in the largest chip developed to date by IBM (5.4 billion transistors), where the large-scale-integration is facilitated by using a fully-asynchronous NoC.

Overall, the NoC area is a promising arena where the integrative benefits of asynchronous design are making important inroads.

References

1. W. J. Dally and B. Towles, "Route packets, not wires: On-chip interconnection networks," in *Proc. ACM/IEEE DAC*, 2001, pp. 684–689.
2. J. Bainbridge and S. Furber, ''Chain: A delay-insensitive chip area interconnect,'' *IEEE Micro*, vol. 22, no. 5, pp. 16–23, Sep./Oct. 2002.
3. T. Bjerregaard and J. Sparsø, "A router architecture for connection-oriented service guarantees in the MANGO clockless network-on-chip," in *Proc. ACM/IEEE DATE*, 2005, pp. 1226–1231.
4. R. Dobkin et al., "An asynchronous router for multiple service levels networks on chip", in *Proc. 11th IEEE Int. Symp. ASYNC*, 2005, pp. 44–53.
5. E. Beigne et al., "Dynamic voltage and frequency scaling architecture for units integration within a GALS NoC," in *Proc. ACM NOCS*, 2008, pp. 129–138.
6. M. Imai and T. Yoneda, "Improving dependability and performance of fully asynchronous on-chip networks," in *Proc. 17th IEEE Int. Symp. ASYNC*, 2011, pp. 65–76.
7. S. R. Naqvi and A. Steininger, "A tree arbiter cell for high speed resource sharing in asynchronous environments," in *Proc. ACM/IEEE DATE*, 2014.
8. Y. Thonnart, E. Beigne, and P. Vivet, "A Pseudo-synchronous implementation flow for WCHB QDI asynchronous circuits," in *Proc. 18th IEEE Int. Symp. ASYNC*, 2012, pp. 73–80.
9. A. Ghiribaldi, D. Bertozzi, and S. M. Nowick, "A transition-signaling bundled data NoC switch architecture for cost-effective GALS multicore systems," in *Proc. ACM/IEEE DATE*, 2013, pp. 332–337.
10. E. Kasapaki and J. Sparsø, "Argo: A time-elastic time-division-multiplexed noC using asynchronous routers," in *Proc. 20th IEEE Int. Symp. ASYNC*, 2014, pp. 45–52.
11. M. N. Horak et al., "A low-overhead asynchronous interconnection network for GALS chip multiprocessors," *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.*, vol. 30, no. 4, pp. 494–507, 2011.
12. A. Lines, "Asynchronous interconnect for synchronous SoC design," *IEEE Micro*, vol. 24, no. 1, pp. 32–41, 2004.
13. W. Jiang et al., "A lightweight early arbitration method for low-latency asynchronous 2D-mesh NoC's," in *Proc. ACM/IEEE DAC*, 2015.

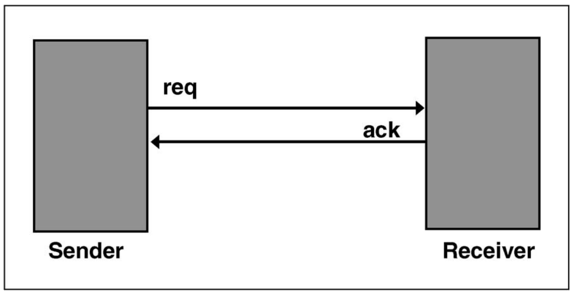

Figure 1. An asynchronous channel.

responds by asserting *ack*, in the active or evaluate phase. The two signals are then deasserted, in turn, in the return-to-zero or reset phase. In contrast, in a two-phase protocol, there is no return-to-zero phase: a single toggle on *req* indicates a request, followed by a toggle on *ack* to indicate an acknowledge.

Both two-phase and four-phase protocols are widely used, with interesting tradeoffs between them. A four-phase protocol has the benefit of returning interfaces to a unique state, i.e., all-zero, which typically simplifies hardware design. It is also a good match for dynamic logic, where the RZ phase directly corresponds to the precharge phase [4], [5], [7], [10]. However, the protocol requires two complete round-trip channel communications per transaction, which can result in lower throughput. A two-phase protocol may involve more complex hardware design, but only requires one round-trip communication per transaction, which can provide higher throughput — and sometimes still has quite low complexity [22], [23], [24]. Alternative protocols using pulse-mode or single-track handshaking have also been proposed.

Once a communication protocol for a channel has been defined, data communication is typically needed. The data itself typically replaces the single *req* wire in the above example. There are two common data encoding schemes: 1) *delay-insensitive (DI) codes*, and 2) *single-rail bundled data*.

Figure 3a illustrates delay-insensitive encoding on a simple two-bit example. A common approach, *dual-rail encoding*, is used for each bit, X and Y, which are each encoded using two rails or wires (X1/X0 for X, Y1/Y0 for Y). Assuming a four-phase protocol, all wires are initially zero, and each bit is encoded as 00, representing a NULL or spacer token (i.e., no valid data). A one-hot encoding scheme is used: the transmission of a 1 (0) value on X involves asserting wire X1 (X0) high, and similarly for the transmission on bit Y. Once the receiver has obtained a complete valid codeword, it asserts *ack*. The reset phase then occurs, where data and *ack* are deasserted in turn. A *completion detector (CD)* is used by the receiver to identify when a valid codeword has been received.

Dual-rail encoding is widely used [5], [7], [17], and is one simple instance of a delay-insensitive code. In particular, note that, regardless of the transmission time and relative skew of the distinct bits, the receiver can unambiguously identify when every bit is valid, by checking for the arrival of a legal codeword (01 or 10) on each bit. As a result, this approach provides great resilience to physical and operating variability. Alternative DI codes have also been widely explored, providing cost tradeoffs in coding efficiency, dynamic power, and hardware overhead, including 1-of-4, m-of-n [24], systematic, level-encoded dual-rail (LEDR) and level-encoded transition-signalling (LETS) [24] codes.

Figure 3b shows an alternative encoding approach, single-rail bundled data. A standard synchronous-style data channel is used, i.e. with binary encoding. One extra *req* wire is added, serving as a “bundling signal” or local strobe, which must arrive at the receiver after all data bits are stable and valid. Both four-phase [3] and two-phase [22], [23] bundled protocols are widely used.

Interestingly, the bundled data scheme allows arbitrary glitches on the data channel, as long as data becomes stable and valid before the *req* signal is transmitted. Typically, data must remain

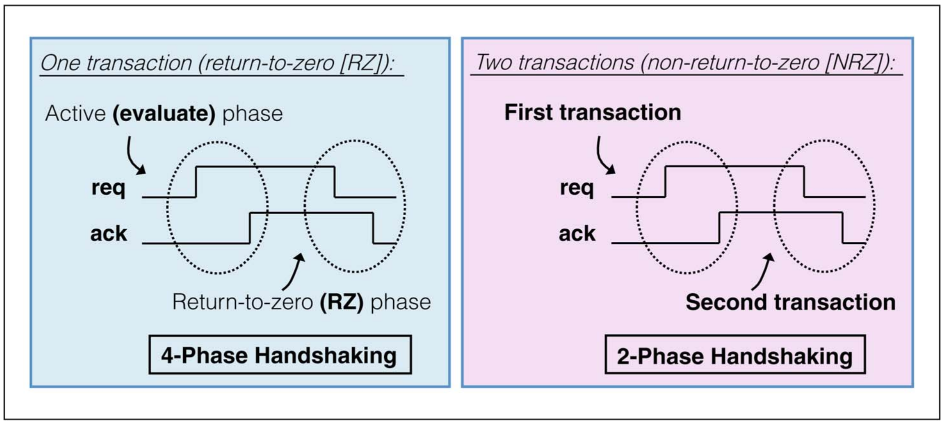

Figure 2. Asynchronous handshake protocols.

valid from before the *req* is transmitted to after an *ack* is received. Therefore, the scheme facilitates the use of synchronous-style computation blocks. It also provides good coding efficiency, with only one extra *req* wire added to the datapath. However, unlike DI codes, a one-sided timing constraint must be enforced: the *req* delay must always be longer than worst-case data transmission. To support this constraint, a small matched delay is added, either an inverter chain or carefully replicated portion of the critical path. Unlike in a clocked system, though, this is a localized constraint: stages can be highly unbalanced, each with its own distinct matched delay. Moreover, the timing margins can be made fairly tight because some parameters (e.g., process, voltage, temperature) tend to be locally more uniform.

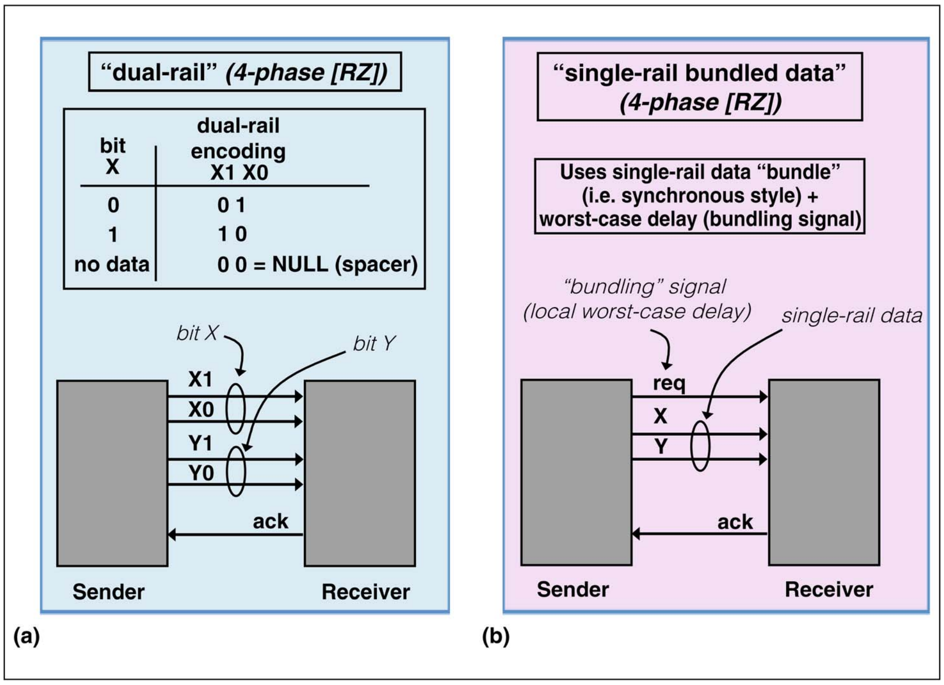
"dual-rail" (4-phase [RZ])
bit X | dual-rail encoding X1 X0
0 | 0 1
1 | 1 0
no data | 0 0 = NULL (spacer)
bit X
bit Y
X1
X0
Y1
Y0
ack
Sender
Receiver
(a)
"single-rail bundled data" (4-phase [RZ])
Uses single-rail data "bundle" (i.e. synchronous style) + worst-case delay (bundling signal)
"bundling" signal (local worst-case delay)
single-rail data
req
X
Y
ack
Sender
Receiver
(b)

Figure 3. Asynchronous data encoding schemes. (a) Dual-rail encoding; (b) single-rail bundled data.

Finally, a hybrid scheme, called *speculative completion* [26], uses bundled data, but also allows variable-latency completion, including better than worst-case, based on the actual data inputs. High-performance parallel prefix adders (Brent-Kung, Kogge-Stone) have been demonstrated, operating at faster rates than synchronous designs.

Pipelining

Pipelining is a fundamental technique to increase concurrency and boost throughput in high-performance digital systems. All modern high-speed processors, multimedia and graphics units, and signal processors are pipelined. In a typical pipelined implementation, complex function blocks are subdivided into smaller blocks, registers are inserted to separate them, and a clock is applied to all registers. In an asynchronous system, no global clock is used and, instead, the interaction of neighboring stages is coordinated by a handshaking protocol. Developing better pipeline protocols and their efficient circuit-level implementation has been the focus of many researchers over the past two to three decades. We review three leading representative styles, starting with the seminal work of Sutherland. More details can be found in a recent survey [24].

Sutherland's micropipeline. Figure 4 shows a basic micropipeline [22], which uses a two-phase handshaking protocol and single-rail bundled data. Each interface between adjacent stages has single-rail data and a bundling signal (req_i) going forward, and an acknowledgment (ack_i) going backward. A delay element is added to match the worst-case delay of the corresponding logic block.

The pipeline operates according to a so-called *capture-pass* protocol. The protocol is implemented using a simple control chain of Muller C-elements[1] [22], [24] (with inversions on the right inputs), operating on a set of specialized capture-pass datapath latches. The latches are initially all normally transparent, unlike synchronous pipelines, so the entire pipeline forms a flow-through combinational path. Locally, only *after* data advances through an individual stage's latches, the corresponding request req_{i-1} causes a transition on the *C* (i.e., capture) control input, which makes those latches opaque, thereby storing and protecting the data. Once data

[1]A C-element is a basic asynchronous storage element; assuming inputs A and B, the output is 1 (0) if both inputs are 1 (0), otherwise it holds its prior value.

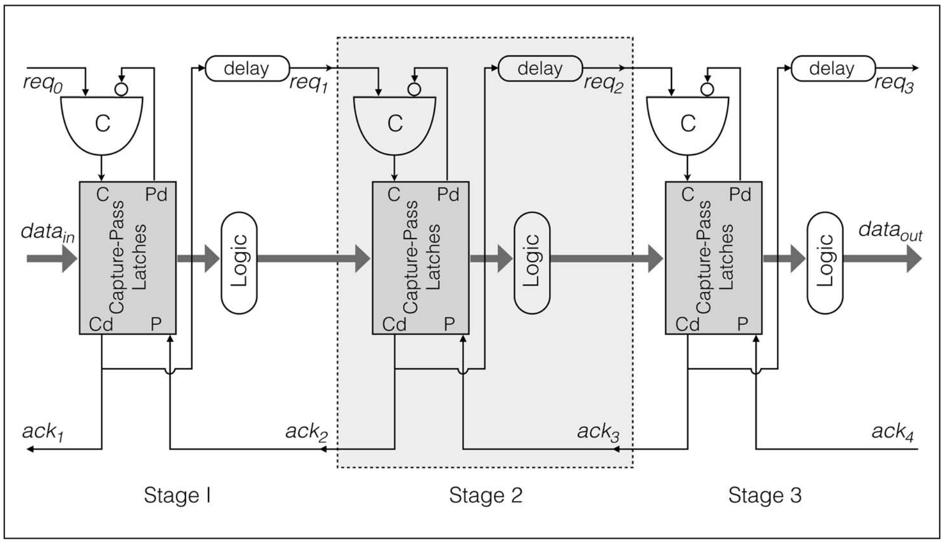

Figure 4. Sutherland's micropipeline.

advances through the next stage's latches, where the data is safely stored, a transition on the *P* (i.e., pass) control input via ack_{i+1}, makes the current stage's latches transparent again, completing an entire cycle. The latches indicate the completion of capture and pass operations via C_d (capture done) and P_d (pass done) outputs, respectively. Effectively, each data item initiates a "wavefront," which advances through the pipeline and is protected by a series of latch-capture operations. Predecessor stages, behind the wavefront, are subsequently freed up through a series of pass operations, once data has been safely copied to the next stage. The old data can then be overwritten by the next wave front.

Although micropipelines require specialized components for implementation, they are remarkable in the simplicity and elegance of their structure and operation, and have inspired several more advanced approaches. Their introduction by Sutherland also provided deeper insights into the nature of asynchronous systems and triggered a resurgence of research activity in asynchronous design.

Mousetrap pipeline. We developed Mousetrap at Columbia University to be a high-performance pipeline that supports the use of an entirely standard cell methodology [23], [24]. Although its two-phase capture-pass protocol is based on that of micropipelines, it has simpler control circuits and data latches, with much lower area and delay overheads. Figure 5 shows a basic Mousetrap pipeline. The local control for each stage is only a single combinational exclusive-NOR (XNOR) gate, and the storage for each stage is a single bank of level-sensitive D-latches, both of which are available in standard cell libraries.

Although the implementation is quite different, the overall operation is similar to that of micropipelines. Initially assume that all req_i and ack_i signals are initially at 0, and all the data latches are therefore transparent. As new data arrives into stage *i* from the left, and passes through the latch, the corresponding req_i bundling signal toggles. As a result, the stage's XNOR toggles from 1 to 0, thereby capturing data in the latch. It also requests the next data item from its left neighbor by toggling ack_i. Subsequently, when stage *i*

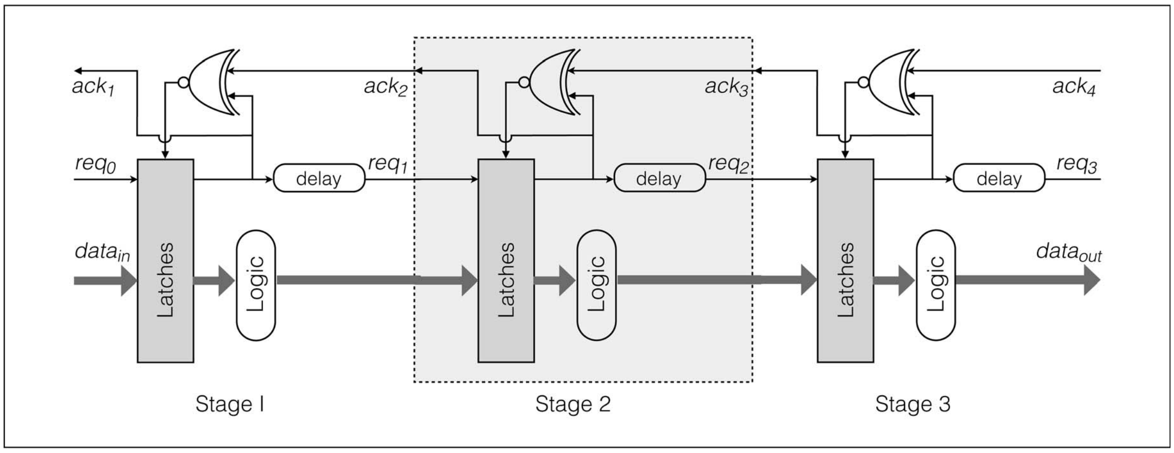

Figure 5. Mousetrap pipeline.

receives an acknowledgment ack_{i+1} from its right neighbor, stage i's XNOR toggles back to 1, making stage i's latch transparent, and completing the cycle. The relatively lightweight control and storage structures allow the pipeline to achieve high throughput: 2.4 giga items/s FIFOs (in 180 nm), and a greatest common divisor (GCD) test chip at 2.1 GHz (in 130 nm technology). Mousetrap circuits have been used in several recently-proposed asynchronous NoC designs (for example, see Horak et al. and Kasapaki/Sparsø in the "Networks-on-Chip" sidebar).

GasP pipeline. GasP was developed at Sun Microsystems Laboratories to push the limit of achievable performance by using an aggressive custom circuit style for specialized applications [25], A distinctive feature is that, instead of the usual pair of request and acknowledge wires, each control channel between adjacent stages consists of a single wire, i.e., "single-track channel," allowing bi-directional communication. Handshaking is performed via carefully generated pulses: a forward request transition sets the state of the control channel, and a subsequent reverse acknowledgment transition resets the channel state. Hence, GasP effectively combines the benefits of both two-phase and four-phase protocols on a single wire. Circuit designs are highly optimized for delay, but the pulse-based protocol imposes two-sided timing constraints (i.e., short and long path requirements), requiring careful balancing of path delays to ensure correct operation [24].

Dynamic logic pipelines. Dynamic logic datapaths are common in high-performance systems—especially in the core of ALUs in high-speed microprocessors and ASICs—despite the greater design and validation effort required. Interestingly, dynamic logic is an especially good match for asynchronous pipelines. In particular, local handshaking obviates the need for the complex and carefully controlled multiphase clocking that is typical of synchronous dynamic circuits, and noise-induced delay variations can be robustly handled through the use of DI encoding [24]. Furthermore, a unique feature of many asynchronous dynamic pipelines is that they are entirely *latchless*, storing data directly on logic block outputs with keepers. As a result, dynamic logic pipelines have been used in several recent high-performance asynchronous commercial products [4], [5], [24].

We review the PS0 pipeline style by Williams and Horowitz [7], [24], which was used in the design of high-speed floating-point dividers at HaL Computers in the 1990s, and was influential on much subsequent research.

Figure 6 shows the basic structure of a PS0 pipeline; each stage consists of a function block composed of dynamic logic, and a completion detector (CD). The datapath uses DI coding (in particular, dual-rail), and there are no explicit registers between adjacent stages. Each function block alternates between an evaluate phase and a precharge phase. Initially, the function block outputs are reset to 0, and in the evaluate phase, awaiting data inputs. In the evaluate phase, each block computes after its data inputs arrive. In the precharge phase, the function block is reset, with all its outputs returning to 0. The CD identifies when the stage's computation is complete, or when its outputs have been reset to 0. The single input control for each stage, Prech/~Eval, is connected from the output of the next stage's CD. The interaction between stages follows a simple protocol: *a stage is precharged whenever the next stage finishes evaluation, and a stage is enabled to evaluate whenever the next stage finishes its precharge*. This protocol ensures that two successive wavefronts of data are always separated by a reset spacer. The use of fast dynamic logic without latches yields purely combinational execution times, even for iterative computations that are implemented using self-timed rings.

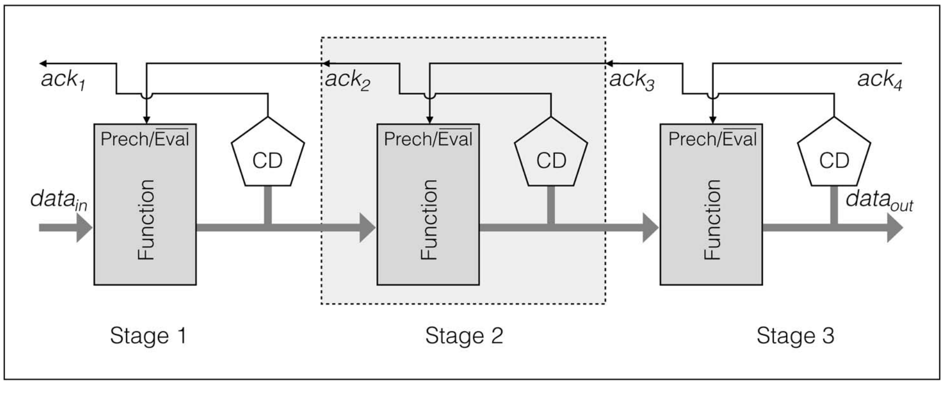

Figure 6. PS0 pipeline.

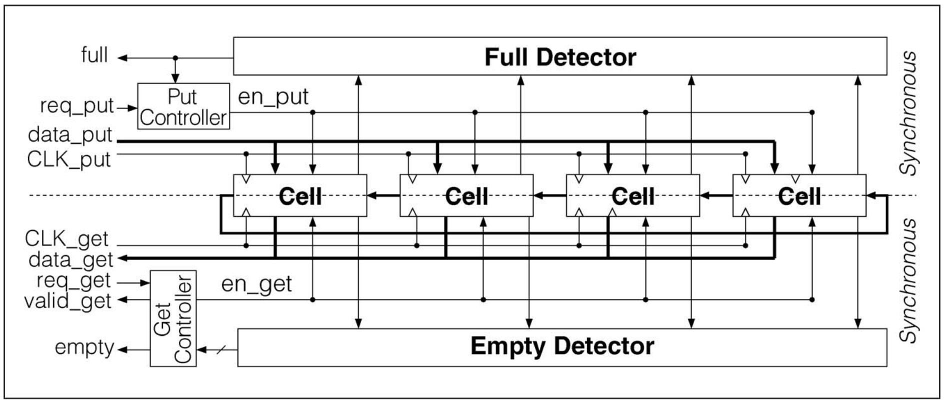

Figure 7. Mixed-clock FIFO.

A number of other dynamic pipeline styles have been proposed [24], with a range of tradeoffs in performance, robustness and other cost metrics. These include dynamic GasP by Ebergen et al. (Sun Microsystems) [25]; PCHB/PCFB by Lines; high-capacity (HC) and lookahead pipelines (LP) by Singh and Nowick; IPCMOS by Schuster et al. (IBM Research); and single-track styles by Beerel et al. Asynchronous pipelines have been used commercially in Sun's UltraSPARC IIIi computers for fast memory access; in Achronix's Speedster 22i FPGA's [5]; in the Ethernet switch chips of Intel/Fulcrum Microsystems [4]; and experimentally at IBM Research for a low-latency finite-impulse response (FIR) filter chip [10].

Synchronization and arbitration

Two related capabilities are needed when handling the continuous-time operation of an asynchronous system: synchronization and arbitration.

Synchronization involves the interfacing of asynchronous and synchronous systems, or two unrelated synchronous systems, where, at the boundary crossing, an asynchronous signal must be safely realigned to a clock domain. A good overview of the topic has been presented by Ginosar [27]. Any direct connection of asynchronous inputs to synchronous registers can cause setup time violations, resulting in metastable operation and possible failure, such as storing of intermediate voltage values or even oscillatory behavior. The first detailed published results identifying and evaluating metastability were presented in 1973 by Chaney and Molnar (see [27]).

The classic solution for a single bit is to provide a basic synchronizer: double or triple flip-flops in series, to ensure sufficient stabilization time to produce a clean synchronous output, with extremely high mean-time-between-failure (MTBF). Detailed synchronizer performance analysis has been proposed, which considers the impact of noise and thermal effects, along with directions to improve circuit design [28]. More general solutions have been proposed for synchronization blocks which support buffering and flow control [29], [30]. The approach by Chakraborty and Greenstreet provides an integrated study of synchronizing two clock domains, ranging from mesochronous to heterochronous communication [29].

Figure 7 illustrates an example of a mixed-clock FIFO by Chelcea and Nowick [30], which can interface two arbitrary clock domains, a sender (put interface) and a receiver (get interface). The design is one of a complete family of modular mixed-timing interfaces, including other variants to support mixed

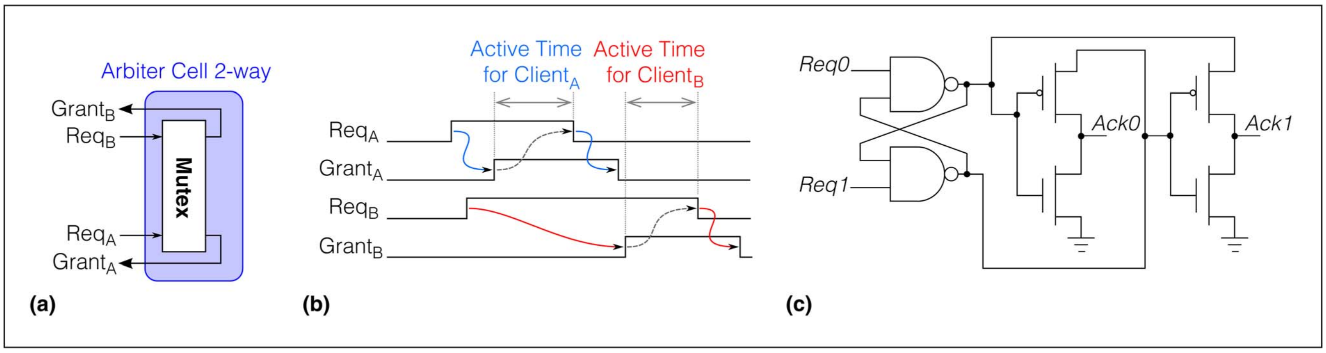

Figure 8. A two-way arbiter. (a) Block diagram, (b) timing, and (c) implementation.

asynchronous/synchronous communication which are needed in GALS systems. The FIFO is constructed as a simple token ring, with pointers to head and tail locations. Each interface operates independently at its own clock rate, and data items do not move once deposited. Full and empty detectors are used to avoid overflow and underflow, respectively. A novel feature is that only three synchronizers are required, regardless of the number of cells in the ring, hence it is highly scalable. It also avoids synchronization performance penalties in steady-state communication scenarios.

Arbitration involves the resolution of two or more competing signals requesting a shared resource. In synchronous design, it is a simple operation: at the start of each clock cycle, existing requests are examined and one is selected as a winner. In asynchronous design, however, inputs arrive in continuous time, and resolution must be guaranteed to be clean and safe, regardless of the signal arrival times.

The basic component to resolve a two-way asynchronous arbitration is a *mutual-exclusion element (mutex, or ME)*, shown in Figure 8, due to Seitz. This analog component guarantees a hazard-free output. In principle, as two competing inputs arrive closer together, its resolution time, i.e., latency, can become arbitrarily long. In practice, though, only extremely close spacing of samples (e.g., < 1 ps) will result in a relatively long delay. More complex asynchronous arbiters typically use mutexes as building blocks. Two- and four-way arbiters are fundamental components in router nodes in asynchronous NoC's. N-way asynchronous arbiters have also been proposed, as well as priority arbiters.

In this part, we have presented an overview of some key advances of asynchronous design, and discussed emerging application areas where asynchrony is poised to play a critical role. We have also reviewed technical foundations, as well as highlighted recent developments in GALS and NoC design. Part 2 of this article focuses on design methodologies and systems, including logic and high-level synthesis, computer-aided design (CAD) tool flows for design and test, and processors and architectures. ■

Acknowledgment

The authors appreciate the funding support of the National Science Foundation under Grants CCF-1219013, CCF-0964606 and OCI-1127361.

References

[1] S. H. Unger, *Asynchronous Sequential Switching Circuits*. New York, NY, USA: Wiley, 1969.

[2] D. E Muller and W. C. Bartky, *A Theory of Asynchronous Circuits*. Cambridge, MA, USA: Annals of Computing Laboratory of Harvard University, 1959, pp. 204–243.

[3] H. van Gageldonk et al., "An asynchronous low-power 80C51 microcontroller," in *Proc. Int. Symp. Adv. Res. Asynch. Circuits Syst. (ASYNC 98)*, 1998, pp. 96–107.

[4] M. Davies et al., "A 72-Port 10G Ethernet switch/router using quasi-delay-insensitive asynchronous design," in *Proc. Int. Symp. Asynch. Circuits Syst. (ASYNC 14)*, 2014, pp. 103–104.

[5] J. Teifel and R. Manohar, "Highly pipelined asynchronous FPGAs," in *Proc. ACM/SIGDA Int. Symp. Field Programmable Gate Arrays (FPGA 04)*, 2004, pp. 133–142.

[6] P. Merolla et al., "A million spiking-neuron integrated circuit with a scalable communication network and interface," *Science*, vol. 345, no. 6197, pp. 668–673, 2014.

[7] T. E. Williams and M. A. Horowitz, "A zero-overhead self-timed 160 ns 54 b CMOS divider," *IEEE J. Solid-State Circuits*, vol. 26, no. 11, pp. 1651–1661, 1991.

[8] K. M. Fant, *Logically Determined Design*. New York, NY, USA: Wiley, 2005.

[9] K. S. Stevens et al., "An asynchronous instruction length decoder," *IEEE J. Solid-State Circuits*, vol. 36, no. 2, pp. 217–228, 2001.

[10] M. Singh et al., "An adaptively pipelined mixed synchronous-asynchronous digital FIR filter chip operating at 1.3 gigahertz," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 18, no. 7, pp. 1043–1056, 2010.

[11] T. Liu et al., "Asynchronous computing in sense amplifier-based pass transistor logic," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 17, no. 7, pp. 883–892, 2009.

[12] L. S. Nielsen and J. Sparsø, "Designing asynchronous circuits for low power: An IFIR filter bank for a digital hearing aid," *Proc. IEEE*, vol. 87, no. 2, pp. 268–281, 1999.

[13] K.-L. Chang et al., "Synchronous-logic and asynchronous-logic 8051 microcontroller cores for realizing the internet of things: A comparative study on dynamic voltage scaling and variation effects," *IEEE J. Emerg. Sel. Topics Circuits Syst.*, vol. 3, no. 1, pp. 23–34, 2013.

[14] J. F. Christmann et al., ‘‘Bringing robustness and power efficiency to autonomous energy harvesting systems,’’ *IEEE Design Test Comput.*, vol. 28, no. 5, pp. 84–94, 2011.

[15] F. Aeschlimann et al., ‘‘Asynchronous FIR filters: Towards a new digital processing chain,’’ in *Proc. Int. Symp. Asynch. Circuits Syst. (ASYNC-04)*, 2004, pp. 198–206.

[16] C. Vezyrtzis, S. M. Nowick, and Y. Tsividis, ‘‘A flexible, event-driven digital filter with frequency response independent of input sample rate,’’ *IEEE J. Solid-State Circuits (JSSC)*, vol. 49, no. 10, pp. 2292–2304, 2014.

[17] P. Shepherd et al., ‘‘A robust, wide-temperature data transmission system for space environments,’’ in *Proc. IEEE Aerospace Conf. (AERO 2013)*, 2013, pp. 1812–1819.

[18] F. Peper et al., ‘‘Laying out circuits on asynchronous cellular arrays: A step towards feasible nanocomputers?’’ *Nanotechnology*, vol. 14, no. 4, pp. 1651–1661, 2003.

[19] M. Vacca, M. Graziano, and M. Zamboni, ‘‘Asynchronous solutions for nanomagnetic logic circuits,’’ *ACM J. Emerg. Technol. Comput. Syst. (JETC)*, vol. 7, no. 4, pp. 15:1–15:18, 2011.

[20] N. Karaki et al., ‘‘A flexible 8b asynchronous microprocessor based on low-temperature poly-silicon TFT technology,’’ in *Proc. IEEE Int. Solid-State Circuits Conf. (ISSCC-05)*, 2005, pp. 272–273, pg. 598.

[21] M. Kishinevsky et al., *Concurrent Hardware: The Theory and Practice of Self-Timed Design*. New York, NY, USA: Wiley, 1994.

[22] I. E. Sutherland, ‘‘Micropipelines,’’ *Commun. ACM,* vol. 32, no. 6, pp. 720–738, 1989.

[23] M. Singh and S. M. Nowick, ‘‘MOUSETRAP: High-speed transition-signaling asynchronous pipelines,’’ *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 15, no. 6, pp. 684–698, 2007.

[24] S. M. Nowick and M. Singh, ‘‘High-performance asynchronous pipelines: An overview,’’ *IEEE Design & Test*, vol. 28, no. 5, pp. 8–22, 2011.

[25] I. Sutherland and S. Fairbanks, ‘‘GasP: A minimal FIFO control,’’ in *Proc. In. Symp. Asynch. Circuits Syst. (ASYNC 01)*, 2001, pp. 46–53.

[26] S. M. Nowick et al., ‘‘Speculative completion for the design of high-performance asynchronous dynamic adders,’’ in *Proc. 3rd Int. Symp. Adv. Res. Asynch. Circuits Syst. (ASYNC 97)*, 1997, pp. 210–223.

[27] R. Ginosar, ‘‘Metastability and synchronization: A tutorial,’’ *IEEE Design & Test*, vol. 28, no. 5, pp. 23–35, 2011.

[28] D. J. Kinniment, A. Bystrov, and A. V. Yakovlev, ‘‘Synchronization circuit performance,’’ *IEEE J. Solid-State Circuits*, vol. 37, no. 2, pp. 202–209, 2002.

[29] A. Chakraborty and M. R. Greenstreet, ‘‘Efficient self-timed interfaces for crossing clock domains,’’ in *Proc. 9th IEEE Int. Symp. Asynch. Circuits Syst. (ASYNC 03)*, 2003, pp. 78–88.

[30] T. Chelcea and S. M. Nowick, ‘‘Robust interfaces for mixed-timing systems,’’ *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 12, no. 8, pp. 857–873, 2004.

Steven M. Nowick is a professor of computer science at Columbia University, New York, NY, USA. His research interests include the design and optimization of asynchronous and mixed-timing (i.e., GALS) digital systems, scalable and low-latency on-chip interconnection networks for shared-memory parallel processors and embedded systems, extreme low-energy digital systems, neuromorphic computing, and variation-tolerant global communication. He has a PhD degree in computer science from Stanford University. He is a Fellow of the IEEE.

Montek Singh is an associate professor of computer science at the University of North Carolina at Chapel Hill, NC, USA. His research interests include asynchronous and mixed-timing circuits and systems; CAD tools for design, analysis, and optimization; high-speed and low-power VLSI design; and applications to emerging computing technologies, energy-efficient graphics, and image sensing hardware. He has a PhD degree in computer science from Columbia University, New York, NY, USA.

■ Direct questions and comments about this article to Steven M. Nowick, Department of Computer Science, Columbia University, New York, NY 10027 USA; nowick@cs.columbia.edu; or to Montek Singh, Department of Computer Science, University of North Carolina, Chapel Hill, NC 27599 USA; montek@cs.unc.edu.