Journal of
Low Power Electronics
Vol. 3, 234–253, 2007

Low Power and Energy Efficient Asynchronous Design

Peter A. Beerel[1,*] and Marly E. Roncken[2,*]

[1] *Ming Hsieh Department of Electrical Engineering, University of Southern California, Los Angeles, CA 90089, USA*
[2] *Strategic CAD Labs, Intel Corporation, Hillsboro, OR, USA*

(Received/Accepted: 15 October 2007)

This paper surveys the most promising low-power and energy-efficient asynchronous design techniques that can lead to substantial advantages over synchronous counterparts. Our discussions cover macro-architectural, micro-architectural, and circuit-level differences between asynchronous and synchronous implementations in a wide range of designs, applications, and domains including microprocessors, application specific designs, and networks on chip.

Keywords: Asynchronous Design, Low-Power, Energy-Efficiency, Latch-Based Design, Dynamic Voltage Scaling, Subthreshold Design, GALS, Asynchronous NoC, Perfect Clock Gating.

1. INTRODUCTION

As integrated circuit (IC) manufacturing processes have shrunk and chip design complexities have grown, the requirements of long battery life, reliable chip operation, reasonable packaging costs, and manageable air-conditioning and cooling have become increasingly difficult to achieve, making low-power and energy-efficiency dominant IC design constraints. Both dynamic power, dictated by switched capacitance, and static power, dictated by gate and channel leakage currents, have become increasingly important.[35]

These issues can be addressed at the system, macro-architectural, micro-architectural, and circuit level—with the most advantages occurring at the highest levels of abstraction. In synchronous design, once the original register transfer level (RTL) specification is defined, the techniques to manage dynamic power include clock gating, multiple voltage supplies, dynamic voltage scaling, and low-swing signaling on long wires. To manage static power, multi- and dynamic-threshold CMOS via power gating and back-body biasing are both used. In addition, power-aware synthesis and place-and-route algorithms addressing both static and dynamic power have been integrated into commercial CAD tool suites.[70]

To appreciate the potential advantages of asynchronous design, it is useful to describe some of the differences and similarities between synchronous and asynchronous designs. At the system and macro-architectural level, asynchronous designs are generally partitioned into channel-connected loosely-synchronized blocks that communicate explicitly via handshaking, when and where needed.[1] Synchronous blocks, on the other hand, are often partitioned functionally or via clock domains; communication between blocks is often implicit in the clocking strategy. Synchronous and asynchronous blocks interact in globally-asynchronous locally-synchronous designs (GALS), a strategy that is growing in interest as routing a single global clock throughout the system is becoming impractical. In fact, communication between blocks in a system on chip (SoC) now often demands networks on chip (NoCs). Asynchronous implementations of these networks promise several advantages, including low power.[94]

At the micro-architectural level, synchronous design is typically flip-flop-based, although latch-based design is beginning to gain interest, particularly because of its low-power benefits.[31, 32] Asynchronous design styles geared towards high performance tend to use *data-driven* channels with latches that are integrated with the logic[16] whereas design styles geared towards low power tend to have more *control-oriented* architectures with explicit latches.[5]

At the circuit level, most synchronous designs are generated with a standard-cell design flow using static CMOS gates. Full-custom designs can use more advanced logic families at the cost of increased design times. Some asynchronous styles have adopted the same standard-cell libraries (e.g., Ref. [2]) whereas others rely on custom libraries and cells, including domino logic (e.g., Refs. [17, 23]).

Some low-power techniques apply equally well to both types of design, including the use of latches instead of flip-flops, low-power logic synthesis and place-and-route,

*Authors to whom correspondence should be addressed.
Email: pabeerel@usc.edu; marly.e.roncken@intel.com

 1546-1998/2007/3/234/020 doi:10.1166/jolpe.2007.138

and techniques to manage leakage currents. However, other techniques are better suited to the more flexible asynchronous macro-architectural, micro-architectural, and circuit structures. In this paper, we survey a variety of different asynchronous designs that cover a range of computation and communication domains from microprocessors to application specific networks on chip. We cover IC designs with high-performance targets for which speed and energy consumption are both critical, as well as designs with relatively low performance targets for which low energy consumption is the primary goal.

The remainder of this paper is organized according to the asynchronous techniques employed and their advantages. Sections 2 and 3 focus on the ability of asynchronous macro- and micro-architectures to reduce switching activity beyond that of their synchronous counterparts, by consuming energy only when and where needed. Section 2 starts by reviewing computational structures and Section 3 continues the analysis with buffering and communication structures. Section 4 then discusses how some asynchronous designs can reduce capacitive loading by using latches instead of flip-flops and managing glitch propagation. Section 5 describes how asynchronous designs can make efficient use of reduced supply voltages, covering both low-swing signaling techniques and the application of multi- and dynamic-voltage supplies. Section 6 discusses general techniques to address static power, and their applicability to asynchronous designs. Finally, Section 7 provides a summary and gives some conclusions.

2. CONSUME ENERGY WHEN AND WHERE NEEDED—COMPUTATION

This section focuses on how asynchronous computational blocks can be designed to consume energy only when and where needed. We focus on the inherent advantages over synchronous clock gating, the ease of implementing asynchronous bit-partitioned architectures, and the energy-efficiency benefits of high-performance dual-rail or 1-of-N-based pipelined implementations. In each case we first explain the advantages qualitatively and then review quantitative examples from the literature.

2.1. Low-Power Asynchronous Systems

Asynchronous designs are characterized by their local data- or control-driven flow of operations, which differs from the more global clock-driven flow of synchronous designs. This enables the different portions of an asynchronous design to operate at their individual ideal "frequencies"—or rather to operate and idle as needed, consuming energy only when and where needed. Clock gating has a similar goal—enabling registers only when needed—but does not address the power drawn by the centralized control and clock-tree buffers. To appreciate this

```
while ( <condition> ){
    x = y ⊗1 z;
    z = y ⊗2 x;
}
```

Fig. 1. Snippet of code used to illustrate differences between asynchronous and synchronous architectural implementations.

difference, we explore the hardware implementation of a trivial code fragment shown in Figure 1: a loop with two subsequent assignments, first to register x then to register z.

A characteristically synchronous implementation is illustrated in Figure 2. Registers for x and z are updated once in each 2-step loop iteration, but the clock toggles every single step. Consequently, for this simple example, the clock must run at twice the frequency that x or z are updated. Clock gating cells (labeled CG) down-sample the clock to the appropriate frequency based on enable signals driven by a centralized control (CTRL). The energy consumed by gated-clock cycles is essentially wasted. In particular, the clock gating cells (CG), portions of the main clock tree (Clk), and the clock buffer cells (labeled CB), as well parts of the centralized control block (CTRL) waste energy—as indicated by the grey shadings in Figure 2.

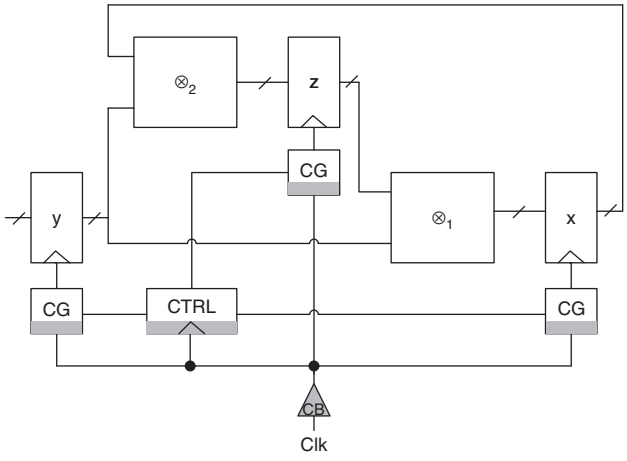

Fig. 2. Synchronous clock-gated architecture for code fragment in Figure 1.

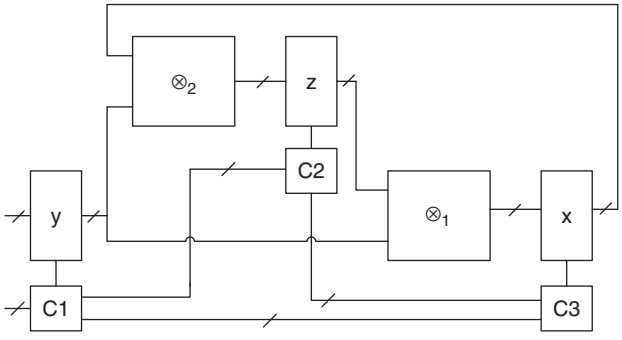

Fig. 3. Asynchronous distributed control architecture for code fragment in Figure 1.

Energy and Power

Power is the rate of energy consumption. The common unit of power, the Watt, represents the consumption of one Joule of energy per second. In the strictest sense, it is no more possible to save power than it is possible to save the speed of an automobile. Rather, it is possible to reduce power just as it is possible to reduce speed. One can reduce power either by using less energy per operation or by reducing the number of operations done per second. In this article we attempt to be consistent with this view by using the word "energy" when considering individual operations and reserving the word "power" for cases where both energy per operation and number of operations per second matter.

This waste of energy becomes more significant as the number of steps in the loop increases, because the ratio of useful-to-gated clock cycles reduces. Register sharing can improve this ratio but at the cost of additional power for multiplexers and higher switching activity in the datapaths. Moreover, register sharing fails to address the wasted energy associated with clock cycles in which the loop is not used, such as in peripherals that are addressed only periodically.[89]

Most asynchronous architectures avoid this energy waste by using distributed control that operates at the ideal rate dictated by the data dependencies in the code. As example, we take an asynchronous implementation for Figure 1 that uses single-rail datapaths—similar to those used in synchronous design.[1, 25] The asynchronous architecture is outlined in Figure 3. Instead of a centralized control, it uses distributed controllers (C1, C2, C3) that communicate via *asynchronous channels* to trigger and update the registers (x, y, z) when necessary. Asynchronous channels typically implement some type of request-acknowledge handshake protocol, and enable the controllers to operate in an event-based manner, at the data update rate. The absence of global clock buffers and global control logic saves energy because the entire block can rest idle when no processing is required.

The cost of these advantages is a more complex control structure, with embedded asynchronous memory elements and matched delay lines per datapath. The area overhead of these controllers can be negligible, but the non-standard circuit structures make supporting these design styles within predominantly synchronous design flows a challenge. This challenge is tackled differently for different asynchronous design styles and target applications, but is usually solved with a separate asynchronous synthesis sub-flow that feeds into an otherwise synchronous flow for static timing analysis, test generation, and place-and-route (e.g., Refs. [4, 89]).

The remainder of this section discusses several low-power asynchronous designs that take advantage of this basic—distributed versus global—architectural difference to achieve significant energy savings over their synchronous counterparts.

2.1.1. DCC Error Correction

The Philips designs for an error correction unit for a digital compact cassette (DCC) player are classical examples of asynchronous implementations that consume energy only when and where needed.[3, 4] In the DCC player, parity symbols are recorded on tape to protect the audio information against tape errors. During play mode, a Reed-Solomon error detector accepts code words of 24 or 32 symbols of 8 bits each, including 4 or 6 parity symbols. The application requires the decoder to decode 3000 code words of 24 symbols and 2300 code words of 32 symbols per second. This implies an input rate of 145,600 symbols per second.

Philips developed 6 designs, two synchronous and four asynchronous. The asynchronous versions were generated using dual-rail and single-rail backend implementations for their Tangram syntax-directed flow for compilation and synthesis.[a] Table I shows a comparison of all 6 designs.

The lowest-power synchronous design (sync 2) is a second-generation design where the clock frequency is reduced to 3.1 Mhz. This is over 20 times the input symbol rate of about 150,000 symbols per second. This implies that a large fraction of the datapath registers may be gated over 95% of all cycles—even with perfect clock gating. And, as mentioned above in the discussion of Figure 2, gated cycles waste energy in the shared clock tree and gating logic. This residual energy consumption can be substantial, and is completely avoided in the asynchronous versions with distributed control.

The best asynchronous circuit (single-rail 2) is the second-generation single-rail design which is 20% larger than the best synchronous version but operates at only 15% its power. This dramatic reduction in power can be attributed largely to saving the energy wasted by even a well-designed clock-gated synchronous circuit.

It is also noteworthy that compared to the best dual-rail asynchronous version, the best single-rail version is 33% smaller, 25% faster, and requires 50% less power. The speed improvement is somewhat counter-intuitive as dual-rail designs are often implemented with domino logic that generally yields higher performances. But Tangram does

[a]Tangram is now part of the Handshake Solutions flow, and goes by the name of "Haste" (language) and "TiDE" (flow).[44]

Table I. Comparison of single-rail, dual-rail, and synchronous DCC error correctors.

	Design					
Quantity	Single-rail 1	Single-rail 2	Dual-rail 1	Dual-rail 2	Sync 1	Sync 2
# of transistors (1,000 s)	21.8	20.3	44.0	30.8	N.A.	N.A.
Core area (mm^2)	4.5	3.9	7.0	5.9	3.4	3.3
Time (ms)	50	40	83	55	N.A.	N.A.
Power (mW)	0.5	0.35	2.3	0.8	12	2.7

Note: The results for single-rail 1, dual-rail 1, sync 1, and sync 2 are obtained from measurements of working ICs. The results for single-rail 2 and dual-rail 2 are obtained from calibrated post-place-and-route simulations.
Source: Reprinted with permission from [4], K. van Berkel et al., A single-rail re-implementation of a DCC error detector using a generic standard-cell library. *Proceedings of the Second Working Conference on Asynchronous Design Methodologies*, May **(1995)**, pp. 72–79. © 1995.

not use domino logic. Its control-oriented design flow does not easily hide the completion detection and reset delays associated with dual-rail logic.

2.1.2. 80C51 Microcontrollers

Interestingly, the energy-only-when-and-where-needed, low-power advantages of asynchronous design extend from application-specific designs into micro-controllers, such as the popular 80C51.[13]

The 80C51 has a complex instruction set architecture which leads to synchronous implementations that use a variable number of clock cycles per instruction. The synchronous baseline design divides each instruction into one, two, or four machine cycles and divides each machine cycle into six slots, each taking one clock cycle.[14] Moreover, in each slot there is communication across a central bus. Using the central bus as a shared communication resource implies that operations must be sequential rather than pipelined. Consequently, for many flip-flops in the design the ratio of useful-to-gated clock cycles in the core can be below 10%, and even lower in the peripherals.

Compared to the synchronous baseline, the Philips asynchronous single-rail implementation shows a factor of 4 reduction in power.[13] In addition to avoiding wasted energy in clock tree and clock-gating logic, the energy-only-when-and-where-needed advantages are realized in the following ways:

- Asynchronous peripherals are *demand-driven* rather than *clock-driven*. For example, the interrupt controller in the synchronous design must be clocked every cycle to poll actively for external interrupts. In the asynchronous design, the interrupt controller is idle until an external signal wakes it up.
- The asynchronous design has a distributed control block with lower switching activity than the synchronous control block, because on average only a few handshake components are active, whereas the entire synchronous controller is constantly clocked.

There are two other reported reasons why the asynchronous design has lower power. The first is the use of point-to-point connections which enable frequent transfers to bypass the bus and lower the switching capacitance at the cost of only a small increase in area. The paper does not address how difficult this architectural change would be in the synchronous design. Secondly, the asynchronous design uses latches instead of flip-flops. This advantage will be discussed in detail in Section 4. The combined cost of these power advantages is an area increase of approximately 30% over the synchronous baseline design.[13]

There is, however, one caveat to this comparison: the synchronous power numbers are based on the design running at a lower than maximum clock frequency, to match the performance of the slower asynchronous design. This observation is important because the synchronous design could, in theory, work at this lower clock frequency using a lower supply voltage, and this would substantially reduce the synchronous power and hence reduce the asynchronous power advantage.

The most recent asynchronous 80C51 designs, commercialized by Handshake Solutions as HT80C51 and HT80C51MX, report successful and ongoing performance, area, and power improvements over the 1998 version. We refer the reader to the Handshake Solutions web site for more details.[44] Handshake Solutions has also developed an asynchronous alternative, ARM996HS, to the more complex ARM968E-S microprocessor, reducing power by a factor of 2.8.[99]

2.1.3. Fano Channel Decoders

The energy-only-when-and-where-needed benefits of asynchronous design also extend to designs using dual-rail datapath templates, such as the pre-charge half-buffer (PCHB) templates developed by Andrew Lines.[20] These templates are used in the communication chip by Ozdag et al.[17, 18] which implements the Fano algorithm, a channel decoding algorithm that is a low-power alternative to the gold-standard Viterbi and Turbo algorithms. The Fano algorithm searches the decoding space *sequentially* to find a good but non-optimal solution, and has far lower *average* complexity than the Viterbi and Turbo algorithms. The synchronous baseline design has a regular datapath where the flip-flops possibly change every clock cycle.[19]

The asynchronous alternative is designed to optimize for the case when the received symbols have no errors.

This leads to a two-part top-level chip partitioning, one part focusing on the case with no errors and the other designed to handle errors and transfer control back when no further errors are observed for the duration of a small fixed number of symbols. The two blocks work at different data-dependent frequencies and synchronize only when necessary. The no-error datapath requires only a fraction of the resources of the original synchronous design, and thus consumes far less energy.

Compared to the synchronous baseline, the asynchronous version typically operates at one third the power and runs over two times faster, assuming standard process normalization. The cost of this advantage is a larger area associated with the dual-rail datapath and the integrated fine-grain pipelined control logic. The asynchronous design occupies five times the area compared to the synchronous design, although much of this could be recovered if the asynchronous design used standard memories with asynchronous wrappers.

In architectural optimizations for the average case, it is always important to ask if the synchronous design could also be optimized for the average case to achieve similar power reductions. Indeed, some of the advantages may be possible in the synchronous design by using multiple clock domains and efficient clock gating. But the performance of a synchronous standard-cell implementation of the no-error part would be significantly slower than the asynchronous high-speed domino-logic PCHB ring.

2.2. Bit-Partitioned and Width-Adaptive Micro-Architectures

Distributed control can also facilitate the partitioning of asynchronous datapaths into bit-partitioned architectures. Once partitioned, components associated with the most significant bits can remain idle and thus save substantial energy, if these bits are known to remain unchanged. This is, for instance, the case when the design operates mostly on small numbers. In particular, in *width-adaptive* 1-of-*N* (token-flow) architectures, blocks associated with unused most significant bits do not receive requests (or tokens) and thus remain idle.[10] In single-rail datapaths, this feature can be coupled with dynamic shortening of matched-delay lines to increase performance.[7]

A similar set of techniques collectively known as *value compression* (which includes size, zero, and significance compression) has been explored in synchronous design.[38–40] The idea is similar in that extra bits are added to the data words to identify which bytes have useful data. It has been shown to yield significant power reduction in caches[41–43] and can be used to clock-gate flip-flops associated with unneeded bytes in the datapath.[38,40] Taking advantage of the higher average performance associated with operating on fewer bits is, however, more challenging in synchronous designs, due to the fixed clock cycle. Consequently, value compression techniques can yield better delays per operation in an asynchronous design context. If dynamic voltage scaling is an option, this reduced delay can be exchanged for a reduction in energy, as expressed by the Et^2 energy-delay metric discussed in Section 2.3.1.

The remainder of this section discusses a variety of asynchronous architectures that use these techniques in the context of application-specific and processor-based designs, with either dual-rail or single-rail datapaths.

2.2.1. IFIR Filters

A representative example is the design of a seven-band IFIR filter bank for a digital hearing aid.[8] The asynchronous implementation exploits the fact that typical real-life audio signals are dominated by numerically small samples, and adapts the number range in its operations to the actual need. It does this by slicing the dual-rail datapath and random access memories into two partitions and by using a tagging and overflow detection scheme that activates the most significant slice only when it is needed.

The synchronous baseline design draws 470 μW when processing input data corresponding to a typical sound pressure level less than 50 dB, while the asynchronous alternative draws 85 μW. The largest single source of power reduction, accounting for 30% of the reduction, comes from the bit-partitioned architecture. The other 70% is mainly due to the low-power distributed asynchronous control, one-hot addressing logic, and an efficient add-multiply-accumulate datapath. As in previously discussed designs, the cost of these reductions in power is a larger area. The synchronous design contains 48,000 transistors whereas the asynchronous design contains 70,000 transistors.

An important observation associated with this application is that although the improved average performance due to the reduced-width datapath exceeds the required performance, this excess cannot be translated into lower energy by means of voltage scaling, because the supply voltage for this hearing aid is already close to the minimum voltage appropriate for the process. Comparing basic power needs, as was done above, is therefore appropriate. This is different for the designs discussed next, where voltage scaling is part of the optimization spectrum and for which energy-delay metrics such as Et^2 are more appropriate.[12]

2.2.2. DCT Matrix-Vector Multipliers

Optimizing for both zero and small-valued data also makes sense in discrete-cosine-transform (DCT) applications that include matrix-vector multipliers. Tugsinavisut et al. explored synchronous value compression techniques as well asynchronous bundled-data (i.e., single-rail) pipeline versions thereof.[6] The key difference is that the asynchronous version uses data-dependent delay lines while the synchronous design has a global clock that can

accommodate only worst-case delays. Though the energy per operation is similar for both designs, the asynchronous architecture is on average significantly faster, yielding a 49% improvement in the Et^2 energy-delay metric.[b] Unlike some of the dual-rail designs described earlier, this comes at a marginal cost in area overhead.

2.2.3. Byte-Serial Sensor Network Processor

Width-adaptive data (WAD) representations use a special integer representation consisting of the binary bits 0, 1 and delimiter bits $\underline{0}$, $\underline{1}$.[10] Delimiter bits $\underline{0}$ and $\underline{1}$ terminate a given binary word: any bit positions more significant than the delimiter bit are assumed to have the same (binary) value as the delimiter. In Ref. [11], Fang et al. evaluate the use and energy advantages of WAD representations for register files. They conclude that WAD integer compression saves a considerable amount of energy in the average case, overcoming the (in their case 25%) energy overhead of adding a delimiter bit per (4-bit) word. For parallel architectures they propose using a hybrid approach in which only a subset of the integers use a WAD representation. This improves the energy efficiency but comes at the cost of a more complicated datapath design. For bit-serial architectures, the datapath can uniformly handle WAD integers: instead of sending the entire word through the datapath, only the bits up to and including the delimiter bit need be sent, producing a *length-adaptive data* (LAD) representation.[9] This saves energy but comes at the cost of reduced performance because of the bit-serial operation.

For the BitSNAP sensor network asynchronous processor in Ref. [9], this LAD-based bit-serial approach reduced the datapath energy consumption by 50% over a comparable 16-bit parallel-word asynchronous processor, while still providing performance suited for powering low-energy sensor network nodes. Unfortunately, a direct comparison to an equivalent bit-serial or value-compressed synchronous processor was not within the scope of this work. However, compared to the Philips 80C51 asynchronous microcontroller discussed in Section 2.1.2, BitSNAP consumes substantially less energy per instruction and energy per instruction per bit.[9] This is particularly exciting, once one realizes that WAD and LAD compression techniques are largely orthogonal to—and hence can be used in addition to—the other low power asynchronous techniques described above.

2.3. Energy-Efficient High-Performance Designs

Asynchronous high-performance designs share many of the architectural low-power advantages discussed above in that modules act only when necessary. Unlike in Figure 3, however, many of the high-performance asynchronous design styles use temporary storage in the handshaking channels between the modules rather than in dedicated latch-based or flip-flop-based registers. Moreover, they often use dual-rail or 1-of-N domino logic in the datapath to encode data validity in the data wires themselves, as opposed to sending a separate validity signal as is done in bundled-data or single-rail datapaths. In addition, their dedicated completion detection circuitry determines when data are valid and when they are neutral (i.e., reset), which is used to control sequencing and fine-grained parallel and serial operations.

As an example, consider the asynchronous micro-architecture illustrated in Figure 4. Here, the functional module on the left directly transmits a 4-bit result to the second functional module on the right, using two 1-of-4 encoded sets of wires labeled A[0..3] and A[4..7]. The first two bits are transmitted by raising one of the A[0..3] wires and the remaining two bits are transmitted by raising one of the A[4..7] wires. After the data are consumed, the two raised wires are reset low in preparation for sending the next 4-bit value. The completion detection circuitry consists of two 4-input OR gates followed by a Muller C-element whose output goes high when the 4-bit result contains valid data and low when it returns to neutral.

The completion detection circuitry can operate concurrently with the datapath and plays a central role in helping the CTRL blocks control the precharging and evaluation of the domino logic in each datapath module. In an optimally-balanced asynchronous pipeline, each datapath module is precharged and re-enters evaluate mode just as new data arrive at its inputs. In this way, the completion detection is hidden from the critical path and there is no latency penalty to the computation—except for the

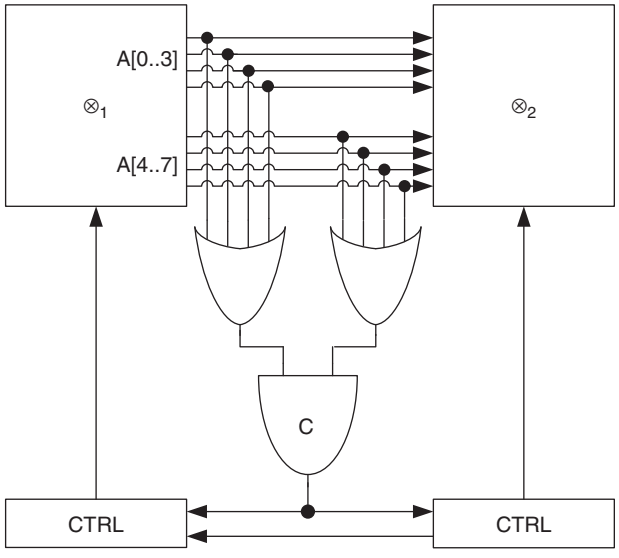

Fig. 4. High-performance asynchronous micro-architecture fragment with two 1-of-4 data bundles and associated completion-sensing-based CTRL.

[b]The use of Et^2 for energy-delay comparisons is motivated in Section 2.3.1.

additional side-capacitance associated with the inputs to the completion detection logic.

Consequently, a high-performance asynchronous pipeline can avoid the data-setup timing margins needed for single-rail latch-based or synchronous flip-flop-based designs. Moreover, in a well-balanced asynchronous pipeline, time borrowing is implicit and fast signals proceed unhindered—free of clock periods set to accommodate the slowest signals. Together, these advantages offer a potentially significant increase in system performance. The cost for such performance is the extra area associated with the 1-of-N data encoding and completion detection. In addition, 1-of-N datapaths typically have higher switching activity than single-rail datapaths, because they require between two to four transitions per $\log_2$N bits. Details of the switching activity depend on whether a two-phase or a four-phase handshake protocol is used and whether an acknowledgement signal is shared among multiple 1-of-N data bundles. Mitigating factors in favor of 1-of-N rail designs (including dual-rail) are the lack of a global clock and the lack of glitching. Glitching can waste significant energy in synchronous and single-rail datapaths. We refer the reader to Section 4 for a more detailed discussion of glitching in implementations that use static logic and single-rail datapaths.

Any increase in system performance can be translated to lower power by using a lower supply voltage, less parallelism, or smaller gates. Moreover, the advantages of self-adapting completion-sensing micro-architectures appear to grow with smaller nano-scale process geometries as on-chip process, voltage and temperature variations increase. The challenge for adopting these design styles is somewhat complicated by the need to support non-standard circuits such as domino logic within predominantly standard-cell-based design flows.

The remainder of this section discusses several asynchronous microprocessors developed by Martin et al.[16] and an asynchronous turbo decoder developed by Golani et al.[23] Both use self-adaptive high-speed completion detection techniques to yield energy-efficient designs.

2.3.1. Martin's Three Generations of Asynchronous Microprocessors

Alain Martin's research group at Caltech produced three generations of asynchronous microprocessors based on different pipelined asynchronous design styles.[16] The MiniMIPS and Lutonium 8051 microprocessors use pipeline pre-charge half-buffer (PCHB) templates[20] to build 1-of-N domino-logic datapath pipelines in the style of Figure 4. These quasi-delay-insensitive circuits offer low latency and high throughput.

To compare and optimize the energy-efficiency of designs across a large range of supply voltages, they adopted the *energy-delay metric* Et^2, where E is the energy per operation, and t the execution time of the operation. This metric is stable over a large range of supply voltages: scaling the voltage by a factor N, scales E by a factor of N^2 and t by $1/N$, thus keeping Et^2 constant. In particular, if design A has a lower Et^2 value than design B at one voltage level, A will also have a lower Et^2 value than B at any other voltage level within normal operating conditions. Moreover, power supply voltage scaling can make circuit A, with the lower Et^2, run faster than B for the same energy per operation, or alternatively, can make A run at the same speed as B but with lower power. The lower Et^2, the better and more energy-efficient the design.

The Lutonium 8051 is particularly interesting because of the macro-architectural and micro-architectural differences with respect to the Philips 80C51 designs discussed in Section 2.1.2. For one, Lutonium uses point-to-point channels instead of a shared bus, and this enables a far more concurrent implementation.[22] Also, unlike the control-oriented design style used by Philips, the 1-of-N PCHB templates used by Lutonium effectively hide the completion detection delays from the critical path, enhancing performance. Finally, Lutonium optimizes its design choices and transistor sizes to minimize Et^2 rather than to minimize power as is done in the Philips approach.

The high performance design of Lutonium gives a remarkably good Et^2. It outperforms its closest competitor, a fast synchronous design, with 25 times lower Et^2. Compared to the best Philips 80C51 implementation (single-rail 2 in Table I), Lutonium is 25 times faster but draws only 19 times more power. Therefore it consumes only nineteen twentyfifths of the energy per operation of the Philips design and completes each operation in one twentyfifth of the time required by the Philips design. Overall, therefore, the Et^2 rating of the Lutonium is approximately 800 times lower than that of the Philips design.[22]

One noted caveat, however, is that continued reductions in feature size and attendant reductions in supply voltage may limit the freedom to exchange throughput against energy, which is at the core of the Et^2 approach.[22] Designs may end up having excess throughput that cannot be converted into energy savings. Consequently, for low-performance applications, low-power design approaches such as the Philips control-oriented single-rail techniques may actually yield lower power.

2.3.2. Turbo Decoders

Block processing is characterized by the need to complete one block of data before a second block can be processed, and is common to many types of algorithms including turbo decoders. Pipelining block processing can improve throughput, but the latency associated with filling and draining the pipeline adds to the block processing delay and limits overall performance. In addition, parallel pipelines can further improve performance; however,

the impact of the pipeline latency grows as the number of bits processed per pipeline per block decreases. The maximum performance is achieved when there are as many pipelines as data words, and is characterized by the processing latency of one word.

Dual-rail domino asynchronous datapaths have lower pipeline latency than equivalent synchronous pipelines. Consequently, they offer higher maximum performance, or alternatively, they can achieve the same performance with fewer parallel pipelines.

Golani et al. quantified such improvements using ultra-high-speed static single-track full-buffer (SSTFB) templates for a tree-based turbo decoder architecture with different block sizes.[23] They compared the throughput per area and the energy per block of an SSTFB-based single-pipeline design to an equivalent synchronous design that requires multiple parallel pipelines to achieve the same overall throughput. Their comparisons show that the asynchronous turbo decoder can offer more than twice the throughput per area for block sizes of 1 K bits or fewer, and can offer lower energy per block for block sizes of 768 bits or fewer. This is particularly useful in low-latency wireless applications that require small block sizes.

3. CONSUME ENERGY WHEN AND WHERE NEEDED—BUFFERING AND COMMUNICATION

Communication fabrics for a system on chip (SoC)—including SoC interconnection blocks (e.g., FIFOs, busses, and crossbars) and more recently proposed networks on chips (NoC)—introduce new constraints and objective functions that must be addressed in both synchronous and asynchronous implementations. The first constraint is that these communication fabrics are geographically distributed across a chip, increasing the difficulty of managing the skew of a global clock. Second, they often require both low latency and high throughput to support the various traffic patterns between chip modules. Third, they often require support for flow control and a latency-insensitive design approach in which pipeline stages can be added or removed even late in the design cycle. Fourth, as the traffic patterns may be random it is desirable to draw power proportional to the traffic density.

Asynchronous macro-architectures and micro-architectures exhibit energy-efficient data buffering and communication in which energy is consumed only when and where needed. Integrated handshaking circumvents the issues of global clock skew, provides a built-in mechanism for flow control, creates a latency-insensitive design approach, and generally draws power that is proportional to the data transfer rate. Moreover, many asynchronous communication fabrics exhibit better latency and throughput than is possible with synchronous alternatives.

This section first describes the basic advantages that asynchronous architectures can exhibit in simple first-in-first-out (FIFO) buffers, then discusses SoC and NoC interfaces, and ends with SoC and NoC communication fabrics.

3.1. Low-Power FIFOs

FIFOs are often needed at the boundaries of SoC or NoC network fabrics and within network routers—e.g., of store-and-forward network architectures. In synchronous design, FIFOs are typically implemented with either shift-registers or register files. Shift-registers are effective for small numbers of cells but the transfer of data between cells is energy inefficient. Register files avoid unnecessary data movement at the cost of large electrical capacitance on bit- and word-lines as well as at the cost of managing read- and write-pointers, both of which limit throughput.

The flexibility of asynchronous design offers a wide range of FIFO alternatives that trade off latency, throughput, and power.[95–98] For instance, an asynchronous *linear FIFO* has the same linear configuration as a shift-register, offers high throughput, and has the same amount of data movement (and latency). In an asynchronous *parallel FIFO*, data are distributed over a number of FIFOs using a demultiplexer, and then re-assembled into a single stream by multiplexing the FIFO outputs. The input side distributes the data elements in a fixed round-robin order over the parallel set of FIFOs, and the output side collects data elements from those FIFOs in exactly the same order. The extra circuitry for demultiplexing and multiplexing costs both area and energy, but for large FIFOs the additional energy consumption is relatively small in comparison to the energy saved in data movement due to the reduced number of FIFO stages each data element moves through. A *tree FIFO* is an extension of the parallel FIFO where the data elements are recursively distributed over a binary tree structure of (linear) FIFOs, and then re-assembled via a mirror-image binary tree into a single output stream.[95, 97] Tree FIFOs have even lower latency than parallel FIFOs, but they also have a higher design complexity and area overhead. The optimal energy configuration depends on the relative costs in energy for demultiplexers, multiplexers, and FIFO stages. An alternative to the tree FIFO is the *square FIFO* which is organized as a square array of FIFOs rather than as a tree.[95, 98]

Non-linear FIFO structures are also feasible in synchronous design, but they are far less practical there because the demultiplexers and multiplexers run at much higher frequencies than the individual FIFO stages, and so synchronous implementations must employ significant clock gating to approximate the reduced data movement. Moreover, the self-timed flow-through nature of asynchronous FIFOs typically yields substantially better latencies than can be achieved in clock-driven synchronous designs.

3.2. SoC and NoC Interfaces—Crossing Clock Boundaries

Systems and networks on chip often include cores with multiple clock domains. Consequently, crossing clock boundaries efficiently becomes a critical issue. When the clock domains are not synchronized, the possibility of metastability appears. Metastability can occur when a storage element, e.g., a latch, captures a changing signal, causing the latch output to stay at an indeterminate value for a prolonged period of time; this effectively propagates an indeterminate signal, "*X*", through the circuit.

A traditional synchronous solution involves using synchronizers at the boundary of two clock domains. These consist of two latches or flip-flops in series controlled by the destination clock. The potentially indeterminate value at the output of the first latch or flip-flop is assumed to resolve to a stable binary value by the time it is sampled by the second latch or flip-flop. The data are often accompanied by an additional validity signal. Only the validity signal needs to go through the synchronizer and the data sampling is controlled by the synchronized validity signal. Alternatively, a dual-ported memory based on register files or FIFOs with writes and reads controlled by the two disparate clocks can be used (e.g., Ref. [81]). These synchronization structures are well-supported by current automated design flows but can yield significant latency overhead, particularly when the SoC or NoC fabric has its own synchronous clock domain because then two synchronizations are needed per core-to-core communication. This is illustrated in Figure 5(a) for an abstract SoC where the synchronous cores and the synchronous network each have their own clock domain. Data transmitted between two synchronous cores suffer the latency penalty twice: once to get on the network and once more to get into the destination core.

With an asynchronous network, as illustrated in Figure 5(b), this latency penalty occurs only once, at the destination core. Data can enter the asynchronous network with very little latency because it is unnecessary to wait for a clock edge.[71] Figure 5 illustrates this reduced latency by using a symbol for synchronous-to-asynchronous interfaces that is approximately half the size of the synchronous-to-synchronous symbol. By omitting the interface symbol, Figure 5 also illustrates the fact that asynchronous-to-asynchronous communication is completely free of synchronization overhead.

The larger size of the synchronous-to-synchronous symbol in Figure 5 is especially appropriate for bidirectional interfaces. A bidirectional synchronous-to-synchronous interface is effectively a pair of synchronous-to-asynchronous and asynchronous-to-synchronous interfaces back-to-back, one for each direction.[71]

Low-latency high-throughput asynchronous-to-synchronous and synchronous-to-synchronous interfaces have been developed by several researchers (e.g., Refs. [71, 72]). These combine efficient asynchronous FIFOs with single-point metastability resolution detectors. Chelsea and Nowick developed high-throughput low-latency interfaces using single-rail data and special metastability detectors to handle the full and empty FIFO cases.[72] Lines developed high-throughput interfaces that arbitrate between the clock and signals for data-ready or space-available, using dual-rail data and pipelined completion detection.[71] Low-latency synchronizers have also been proposed by Chakraborty and Greenstreet but are applicable only when both domains are synchronous.[74] The circuits are based on the insight that the regularity of synchronous clocks permits prediction in advance of safe sampling times. Their results are impressive. A proposed single-stage ripple FIFO can provide nearly two clock periods of skew

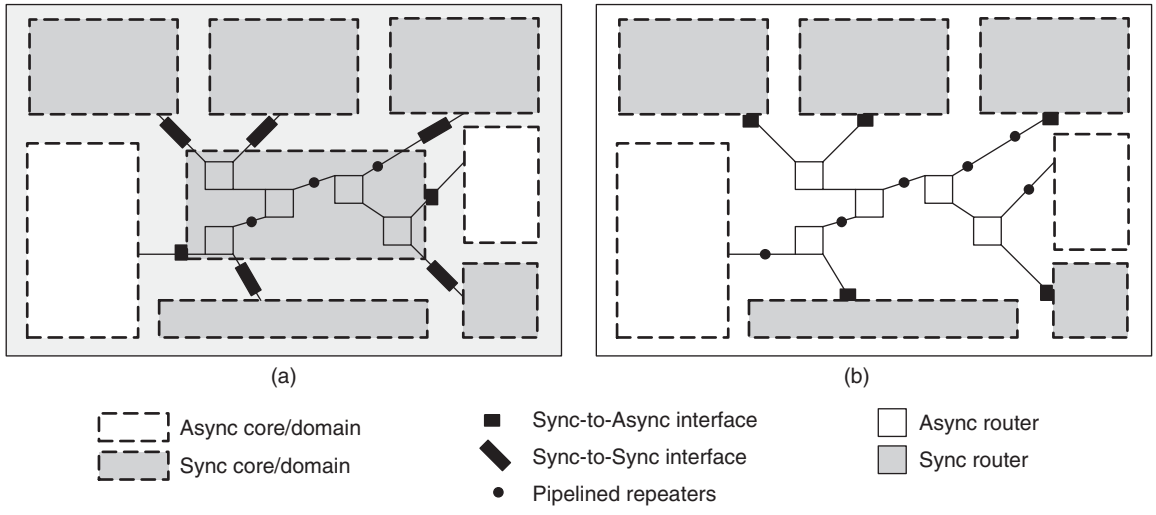

Fig. 5. Tree-based network on chip (a) synchronous (b) asynchronous.

tolerance. More complicated designs support clocks of arbitrary frequencies and achieve latencies that are at most slightly more than one clock period and often less.

An alternative approach to synchronizing globally-asynchronous locally-synchronous (GALS) systems is to use locally generated clocks that stretch when required; these are known as "pausible clocks."[73, 75, 76] In particular, the clock can pause until expected input data have arrived or until the environment is able to consume output data. To accommodate clock-tree delays, inputs and outputs may require FIFOs whose lengths correspond to the maximum number of clock edges in flight through the clock tree. The latency overhead of these FIFOs may be negligible if inputs or outputs occur less frequently than every cycle. The pausible clock has been applied effectively in a variety of areas, including cryptographic security,[77] wireless baseband processing[78] and multi-core DSP processors.[79]

Note that the power and energy consumption of the interfaces between clock domains and the power and energy consumption of the pausible clocks themselves is less important than their impact on low-power and low-energy system design. Compared to clock-gating, locally-generated clocks can conserve energy that would otherwise be wasted in clock generation and clock-tree buffers. More significantly, these locally-generated-clock approaches make designing with many clock domains, including asynchronous domains, practical and efficient. System designs that use multiple clock domains offer significant benefits, because such designs permit each synchronous block to operate at the lowest possible frequency and energy level. By removing roadblocks to combining synchronous and asynchronous designs, these approaches provide the necessary foundation for asynchronous SoC and NoC communication fabrics.

The increasing variability in today's ICs is making traditional fully synchronous solutions less practical, from both a complexity and power stand-point. Macro-cells for crossing clock boundaries are appearing not only in design libraries but also in static timing analysis and formal verification tools (e.g., see Ref. [80]). Industry adoption of at least some of these more advanced GALS interface techniques appears inevitable (e.g., see Ref. [89]).

3.3. SoC and NoC Communication Fabrics

Networks on chip (NoC) are viewed by many as a strategic technology for addressing the increasing communication needs in large-scale multi-core systems on chip (SoC). They offer a scalable communication infrastructure, with better modularity, versatility and predictability than traditional bus systems.[82, 83, 94] Additionally, globally-asynchronous locally-synchronous (GALS) technologies can offer a strategic and versatile solution to the global clock and synchronization problems between the multi-clock core domains of a large SoC.

The communication architecture of an NoC can be general-purpose or application-specific, depending on whether the cores play a homogeneous or heterogeneous role in the target application. Application-specific networks require a more advanced design flow to analyze and tune the network topology and placement to given energy, throughput and latency targets, within reasonable design time and effort.[84] As an example, Figure 5 illustrates an abstract heterogeneous system with 7 cores and a tree-based NoC with 5 routers. To achieve high throughput and avoid long and hence slow wire connections, the ingoing and outgoing channels of the NoC routers can be pipelined using what is referred to as *relay stations*[72] or *pipelined repeaters*.[71]

Below, we discuss four GALS-based NoC styles: Nexus[71, 85] and FAUST[86] use a general-purpose NoC design, CHAIN[88] and pSELF[91, 92] target application-specific NoCs.

3.3.1. Nexus

Nexus by Fulcrum Microsystems[71, 85] has a 16-port, 36-bit asynchronous crossbar interconnect, with routing and arbitration to resolve contention. It handles one-way burst transactions between locally clocked modules. A burst has a variable number of data words terminated by a tail bit, and is routed atomically when the following three conditions hold: sender has data, the receiver has space, and the arbitration is won. In that case, crossbar links are created from the sender channel to the receiver channel. The links are disassembled automatically when the last word of the burst leaves the crossbar, to free the crossbar for the next transaction. Round trips are implemented as split transactions.

All cross-chip communication and routing are implemented using a low-latency, high-throughput quasi-delay-insensitive (QDI) custom design style with four-phase handshaking, low-power 1-of-4 data encoding, and high-speed precharge domino logic (see Section 2.3). QDI designs are particularly attractive for NoC applications because they work robustly over large delay variations, due, for example, to voltage supply droop, temperature changes or crosstalk. As with all asynchronous schemes, the resulting network consumes energy only for transactions actually done.

3.3.2. FAUST

FAUST (Flexible Architecture of Unified System for Telecom) is a NoC-based SoC that supports complex telecom applications that use orthogonal frequency-division multiplexing (OFDM).[86] OFDM is a popular modulation scheme for wideband digital communication and a candidate for 4 G mobile communication. Like Nexus, the asynchronous on-chip network (NoC) in FAUST uses a low-latency (but

standard-cell) QDI design style[87]—in this case to implement a high-throughput 2D-mesh topology with packet switching and wormhole routing. Wormhole routing is similar to the atomic crossbar routing in Nexus in that it keeps the data words, called "flits," in a packet together, possibly spanning multiple NoC routers. This reduces the buffer sizes per router and avoids the need for reordering.

But unlike Nexus, FAUST also handles real-time constraints. The real-time constraints can be derived from the embedded application, and are partitioned into (1) high-priority real-time or low-latency packets, which are mapped to a high-priority virtual router channel, and (2) low-priority best-effort packets, which are mapped to a low-priority virtual router channel. Packets in the high-priority virtual channel have routing priority over those in the low-priority virtual channel. A credit-based flow control technique avoids deadlock and contention at the transport layer.

The asynchronous NoC in FAUST has 20 routers and takes about 15% of the overall SoC chip area, which is comparable to that of a bus-based solution. The NoC's typical power is about 6% of the overall 23-core SoC power required by the application. The major power reduction comes from dynamic voltage scaling in the cores: the clock frequency of each core is set to reach the best trade-off in required performance versus power. Dynamic voltage scaling is discussed in more detail in Section 5.

3.3.3. CHAIN

CHAIN (CHip Area INterconnect)[88] targets heterogeneous low-power systems in which the network is system specific. It uses four-phase handshaking and a QDI data encoding for better timing closure and robustness against crosstalk—as do Nexus and FAUST. The QDI data encoding is typically 1-of-4 with an extra signal to indicate End-Of-Packet and a reverse acknowledgement, making 6 wires per link. Links can be *ganged* to increase the bandwidth. This creates the flexibility to use links with different data widths along different paths, and to trade lower-frequency but higher-bandwidth parallel operations against higher-frequency but lower-bandwidth serial ones. The variable-length packets remain intact by using best-effort wormhole routing (with potential extensions to prioritized routing[90]). The network topology can be set up as a ring, mesh, bus or a tree structure.

CHAIN is now an integrated part of the Silistix design flow CHAINworks™.[47] Intel and Silistix jointly evaluated an early version of CHAINworks™ on the bus-based Intel® PXA27x processor family for cellular and handheld applications,[89] and concluded that the asynchronous NoC design style is well suited for low-power on-chip communication and can greatly reduce the total number of global wires and global timing constraints. They also concluded that the design approach could fit seamlessly into their standard SoC flow, and that it offers a cleaner partitioning and interfacing of bus and peripheral operations for design as well as for validation purposes. They emphasize the importance of asynchronous-to-synchronous interfacing, which turned out to be their biggest design automation tool challenge and their biggest effort in reducing communication latency.

3.3.4. pSELF

Researchers at the University of Utah came to a similar conclusion as reported by Intel and Silistix: the NoC latency bottleneck is in the traditional synchronization interfaces, not in the routing.[92] Their pSELF (phase SELF) network protocol[91, 92] is a variation on the SELF (Synchronous ELastic Flow) protocol.[36, 93] Both protocols use the valid and stall signals typical for *latency-insensitive* or *elastic* pipelines that can hold a variable number of data elements,[37] and they can both be implemented as a strictly asynchronous circuit or as a clocked version.

The latency-insensitive design approach makes it easy to add or remove pipeline stages even late in the design cycle to accommodate delay changes in the computation and communication modules. Traditional clocked systems make such additions much more difficult. SELF supports this flexibility by sending the global clock into a control block for each pipeline stage. This control block enables or disables the latches on the basis of the valid-stall-clock protocol. This reduces the global clock load and provides local clock gating on a stage-by-stage basis. Asynchronous and GALS implementations of pSELF—and likewise of SELF—have no global clock, and so these completely eliminate the global clock distribution network and load. This is particularly beneficial at lower network activities such as are typical for cell phones and PDAs (see, e.g., Refs. [89, 15]).

The asynchronous-to-synchronous synchronization interface for pSELF enjoys over four times lower latency than traditional synchronous-to-synchronous synchronization interfaces, due to both its more efficient synchronization and the flow-through nature of its buffers. This carries over to the total network latency for the core-to-core asynchronous communication in pSELF, which exhibit over four times lower latency than the clocked version in their examples. Thus, pSELF achieves substantial low latency, and provides elasticity and flexibility to reduce energy in the cores as is done in FAUST.

4. REDUCING CAPACITIVE LOAD—LATCHES VERSUS FLIP-FLOPS

Standard-cell synchronous circuits use either latches or flip-flops as their basic storage elements. A typical standard-cell latch is shown in Figure 6(a) and a typical

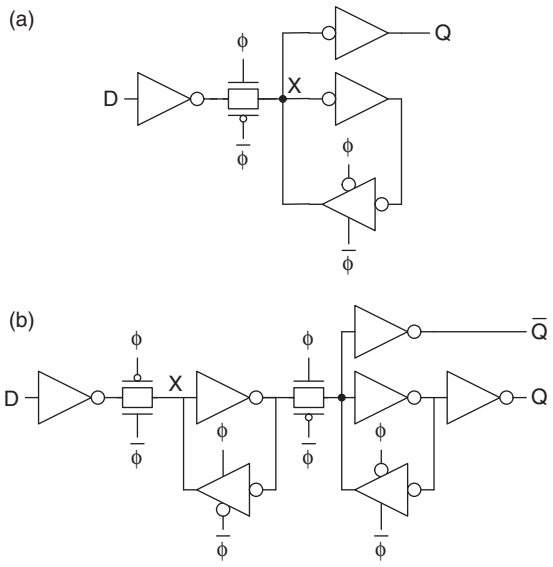

Fig. 6. Typical standard-cell designs for (a) latch and (b) flip-flop. (Courtesy of D. Harris, Harvey Mudd College.)

flip-flop, composed of two back-to-back latches, is shown in Figure 6(b).[34] As is evident from the drawing, latches are often half the size and have half the clock input capacitance of their flip-flop counterparts. Compared to flip-flop based storage, latch-based storage thus has the *potential* for significantly lower area as well as lower energy consumption associated with both the reduced capacitance on the clock as well as the reduced datapath energy consumption associated with the lower complexity of the latches. This is particularly true in high-performance designs in which clocks and latches can consume 70% of the overall energy (e.g., Ref. [33]).

Moreover, latches can permit time borrowing across pipeline stages which can yield smaller, low-power gates along otherwise critical paths.[34] These same time-borrowing properties, however, can also enable latches to propagate glitches across pipeline stages, thus wasting energy.

For low-performance applications with many levels of logic per pipeline stage, glitch energy can be a significant concern. Thus, for these applications a careful analysis of the costs and advantages of latches from a low-power perspective is important.

Latches are used extensively in full-custom synchronous designs and more recently in synchronous NoC designs.[36, 37, 93] However, latches have not gained wide-spread acceptance in application-specific integrated circuit (ASIC) designs, where flip-flop-based standard-cell flows still dominate the marketplace. This lack of wide-spread adoption is likely due to the fact that ASIC design flows are highly automated. Supporting latches in the design requires supporting them at all levels of the design flow, from synthesis to place and route, from clock generation and timing analysis to module- and system-level validation. This is a substantial development effort that requires strong demand from ASIC designers. A positive example of this is the support of pulse-based latches instead of flip-flops, to save area and energy. This advanced technique is used in the IBM Cell processor[32] and explored by Cadence Design Systems.[31] It has many of the advantages of using regular latches but allows designers to use traditional flip-flop-oriented register-transfer level (RTL) specifications, thus easing adoption.

As mentioned in Section 2.3, asynchronous designs come in two flavors, one based on dual-rail or 1-of-*N* datapaths in which the storage elements are integrated into the logic. The other based on single-rail datapaths with separate latches or flip-flops for storage. This section focuses on single-rail approaches that use latches.

4.1. Asynchronous Flows with Normally Open Latches

Micropipelines, originally introduced by Ivan Sutherland,[25] is a broad class of asynchronous pipelines that generally includes single-rail datapaths, delay lines, and pipeline control. As with all asynchronous pipelines, the pipeline is *elastic* in the sense that it can hold a variable number of data elements depending on the rate of data being sent in from and consumed by its environment. The minimum number of data elements it can hold is zero and we say the pipeline is empty. The maximum, also called *capacity*, varies depending on the concurrency in the distributed pipeline control logic. With *half-buffers* the capacity of a linear pipeline is half the number of stages; with *full-buffers* its capacity equals the number of stages. A variety of different control schemes have been proposed that use two-phase request-acknowledge handshaking,[25] four-phase request-acknowledge handshaking,[24] pulse-mode valid-ack interlocked handshaking,[33] or single-track handshaking,[28, 100] most of which use latches for datapath

Open and Closed Latches

Unfortunately, hydraulic and electrical engineers fail to agree on the meaning of "open" and "closed." An open valve permits water to flow whereas an open switch prevents the flow of electricity. Some authors prefer, therefore, to apply the terms "transparent" and "opaque" to the two states of a latch. In this paper, however, we have used the hydraulic convention. An "open" latch is transparent and permits data to flow through, whereas a "closed" latch is opaque and preserves the last value that passed through it.

storage.[c] Overall, the energy consumption in asynchronous pipelines is dominated by the logic and latches in the datapath rather than by the control.

In most of these schemes, the latches are open in the initial, pipeline-empty, state. The latches associated with a particular pipeline stage close only after all logic glitches in the corresponding datapath have resolved and the latch inputs are known to be stable. Meanwhile, glitches can propagate through the entire empty pipeline. The latches for a particular pipeline stage re-open once their data are no longer needed by the subsequent stage. With this in mind, consider the propagation of glitches in a pipeline when it is *full*, i.e., when it is holding the maximum number of data elements possible. Once the output environment consumes data, a *bubble* ripples back through the pipeline, subsequently enabling each stage to first open and then close its latches to capture new data. The amount of glitch propagation in this case is significantly limited by the number of closed latches in the nearly-full pipeline. This implies that, unlike in synchronous latch-based design, glitch propagation is a function of the average number of data elements in the pipeline. If more pipeline stages are empty, the pipeline may exhibit more glitches than when the majority of the stages are full. Generally speaking, the fuller the pipeline, the lower the glitch power. Interestingly, the optimal throughput occurs when the forward and backward propagation of data and bubbles in the pipeline balance each other.[20, 21] Thus there is also a tradeoff between optimal throughput and glitch power that to our knowledge has not yet been analyzed.

Compared to traditional synchronous designs, these latch-based micropipeline styles generally guarantee a significant gap between one stage closing and the previous stage opening,[33] making it easier to satisfy hold times. In ultra-high-speed pipelines, however, this gap can be small[27] and additional min-delay buffers may still be necessary to satisfy the hold-time-related constraints. These buffers do consume additional energy but also balance the propagation delays through the logic, and thus in turn also reduce the amount of glitch energy. Though this is not discussed in Ref. [27] glitch energy is thus to some degree related to the pipeline granularity of the design. In particular, for finer-grain pipelines (e.g., "gate-level pipelines" where the logic in each stage is only a single gate deep), the propagation delays for different bits are quite balanced, thereby causing much less glitching activity. This is an interesting, but perhaps secondary, effect in high-speed single-rail pipelines where the energy consumption is dominated by the latches rather than by the datapath and control logic.

[c]Notably, one proposed technique uses low-power double-edge-triggered flip-flops.[26]

4.2. Asynchronous Flows with Normally Closed Latches

To limit glitch propagation, several flows have been developed with latches that are normally closed. One proposed approach by Chelcea et al. uses self-resetting latches in a micropipeline style datapath.[29] In this case, the latch opens only after data is guaranteed to be stable and closes automatically after some controller delay. Compared to normally-open latch schemes, this comes at a 14.3% performance penalty but the reduced glitching yields a 41.4% average improvement in the combined energy-delay metric Et. The delayed closing of the latch increases the challenges in satisfying hold times but this was not observed to be an issue in Ref. [29]. Another interesting micropipeline-based approach is to use latches that can be re-configured on the fly to be either normally open or normally closed, enabling high performance when needed and low energy consumption otherwise.[30]

Ad Peeters explored the relationship between data-validity schemes and handshaking protocols as well as their impact on energy consumption.[5] Based on this analysis, he proposed the low-power design flow used to design the single-rail DCC and 80C51 microcontroller described in Section 2.1. This design flow is not micro-pipelined based but still uses single-rail datapaths with normally closed latches and a four-phase handshaking protocol with a matched-delay line per datapath. The matched delay is typically half that of the worst-case datapath delay, and is exercised twice during the four-phase handshake cycle that controls the datapath operation. The destination latches are opened after one pass through the delay line, well after the source latches have closed. The destination latches are closed at the end of the second pass. Consequently glitching is limited to the logic surrounding the destination latches.

This way, read and write access to a latch are guaranteed to be mutually exclusive. In some cases when multiple writes occur without an intervening read, unnecessary switching activity occurs in the downstream logic attached to the latch output. For cases where such glitching is substantial, Peeters developed a more complex latch implementation with a read port that propagates the data only when it receives a read request.[5]

5. REDUCING THE SUPPLY VOLTAGE

CMOS circuits, both synchronous and asynchronous, can operate over a very large range of supply voltages. At a lower supply voltage, the circuits run slower and use less energy per operation than at a higher supply voltage. This relationship is exploited in *dynamic voltage scaling*: at low circuit activity or workload, the supply voltage is lowered such that the operation finishes "just in time," and likewise, at a high workload the supply voltage is raised to meet the operation deadline—again, preferably "just in time."

For nominal to threshold supply voltage levels, the operating speed or frequency of the circuit changes approximately linearly with the supply voltage and the energy per operation changes quadratically. Thus, the power has a cubic dependency on the supply voltage.[49] The region where the circuits operate at voltage levels ranging from slightly above the nominal supply voltage to slightly below threshold is striped in Figure 7. This region is an excellent match for asynchronous circuits and the energy-delay optimization metric Et^2 discussed in Section 2.3.1.

The quadratic relationship between energy and supply voltage no longer holds as the supply voltage scales down into the subthreshold region. In the subthreshold region, the circuits operate using leakage currents, and as a result the operating speed of the circuit changes exponentially with the supply voltage. Leakage currents integrate over the longer operating delay until the leakage energy per operation exceeds the active energy, as is shown in Figure 7. The voltage level at which this happens defines the (global) minimum energy point of operation. The fact that this minimum lies in the subthreshold region makes subthreshold operation a worthwhile challenge to pursue when designing ultra-low energy circuits.[52] Asynchronous design techniques can offer additional support, for instance by minimizing the idle time per operation to avoid unnecessary leakage energy.[55]

Note that in both regions—i.e., the "Et^2 operation" region and the subthreshold region—the reduction in energy per operation stems from a lower swing in the (global) supply voltage. It is also possible to generate and use low-swing signals on a more local basis, and gain energy efficiency where it is most effective.[48] The grey-colored "low swing operation" region in Figure 7 gives an indication of the significant gain in energy that can be obtained this way—as example, we show a 10% (of V_{nominal}) swing around a near-threshold reference voltage. Low-swing signaling has been studied and applied, usually in the context of full-custom synchronous design. It is a popular technique for energy-efficient communication over long wires, but the combination with asynchronous design techniques is far less studied.

Below, we outline several representative asynchronous techniques that are used in the Et^2 and subthreshold region as well some that relate to the combination of low-swing signaling and asynchronous design.

5.1. Using Dynamic Voltage Scaling and Multiple Supply Voltages in the "Et^2 Operation" Region

Scaling the operating voltage in the "Et^2 operation" region (see Fig. 7) is easier for asynchronous circuits, because they adapt automatically to the increased or decreased delay caused by the reduced or elevated voltage. This comes with the caveat that tight timing relations, not covered by the delay-insensitive part of the design, must still be validated by characterizing the design for the various process, temperature and voltage corners that the circuit will use—just as is done for the setup and hold timing relations in synchronous circuits. But once these timing relations are validated, there is a great run-time benefit over synchronous circuits of *not* having to coordinate supply voltage with clock speed.

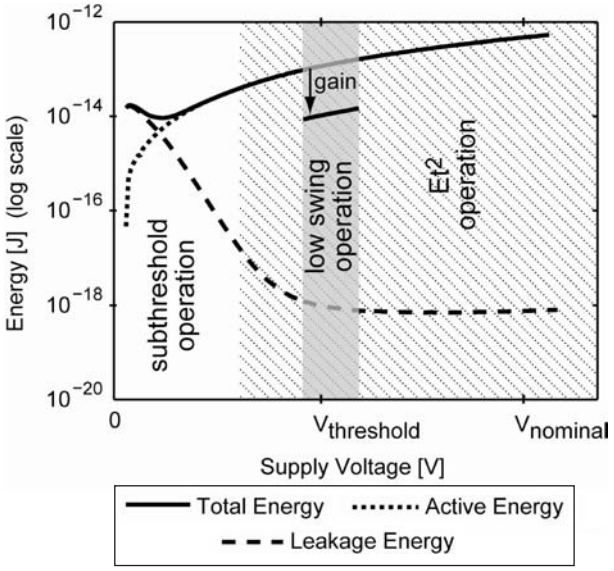

Fig. 7. Scaling of energy per operation versus supply voltage, for the Et^2 operation region from V_{nominal} to sub-$V_{\text{threshold}}$ (striped), low voltage swing (grey overlay), and subthreshold region (white). Note: The graphs in Figure 7 are based on Figure 3 in Ref. [49] and reflect an operation with an activity factor of 2%, simulated for a 0.18 μm CMOS process.

Philips and the Technical University of Denmark explored a system-level solution to adapt the supply voltage for asynchronous circuits to the smallest possible value that still meets the performance requirements, while accounting for process and data dependent variations.[60] Their solution is to insert a first-in-first-out (FIFO) buffer at a performance-critical system interface, and monitor its occupancy to detect if—and to what extent—the system fills or drains the buffer faster or slower than needed. This information is then used to drive a DC–DC converter to down-scale or up-scale the supply voltage.[d] The DCC error corrector discussed in Section 2.1.1 is an example of an application where this solution works particularly well, because the processing time for a code word depends on its error content. Given that 95% of the code words are correct, only a sequence of incorrect code words would temporarily raise the supply voltage above the lowest level.[60]

[d] An alternative to using an adjustable DC–DC converter is to use power switches to select from among a number of power supplies. This alternative solution was proposed in Ref. [52] to overcome the delay and limited efficiency of DC–DC converters for broad voltage ranges, and enable the use of dynamic voltage scaling into the subthreshold region.

As an alternative to—or even in addition to—using dynamic voltage scaling, asynchronous systems can take advantage of multiple supply voltages. This works particularly well in the context of asynchronous networks in multi-core systems on chip.

Researchers at CMU use FIFO buffers in the context of multi-core system on chip (SoC) design, where the buffers act as interfaces between multiple voltage and frequency domains.[62] The cores are synchronous, and can be homogeneous or heterogeneous. Communication between the cores is asynchronous and supported by a network on chip (NoC) communication architecture. CMU developed an optimization strategy that assigns cores to joint or disjoint voltage-frequency islands (VFI) and then assigns static supply and threshold voltage levels to each island to minimize the energy consumption subject to the performance constraints of the target application. This also accommodates dynamic voltage scaling: for applications with large workload variability, the operating voltage and frequency for each island can further be controlled dynamically around the statically assigned values. The team simultaneously partitions the NoC over the set of VFIs, so that the contributions in energy and performance account for both computation and communication. Their partitioning results in 40% (simulated) energy savings for an MPEG2/MP3 video application. Their globally-asynchronous locally-synchronous (GALS) design style works well with multiple voltage-frequency domains, and likely supports a larger configuration space for energy optimization than a clocked NoC would.

The CMU GALS techniques discussed in Refs. [61, 62] can be combined with the other asynchronous buffering and communication techniques described in Section 3.

5.2. Energy Efficient Asynchronous Techniques for Subthreshold Operation

The subthreshold region is where the point of minimum energy per operation lies (see Fig. 7)—which makes it a desirable region of operation for energy-constrained applications such as wireless sensor nodes and medical implants. For a circuit to operate in the subthreshold region means (1) operating with aggravated process variations, because NMOS and PMOS transistor currents are no longer balanced, and (2) operating with reduced robustness, because of a reduction in the transistor on-off current ratio. This is also the region where minor variations in supply voltage give large variations in delay and hence leakage energy. We will next discuss two asynchronous techniques that reduce the fraction of leakage energy by minimizing the idle time per operation.

Jayakumar et al.[54, 53] operate on a linear network of programmable logic array (PLA) modules, with strictly forward data dependencies between the modules. The dynamic NOR-NOR PLAs are of medium size and have constant output delays across all possible data inputs. The predictable delay behavior and the regularity of the PLA modules significantly improve the yield in the presence of process, voltage and temperature variations and make this design style suitable for operation in the subthreshold region. The design predictability and regularity also enable design automation. The network of PLAs in Ref. [54] is clocked and operates as a single combinational unit: all PLAs are precharged simultaneously, and then start evaluating in a domino fashion until all PLAs have evaluated, and only then can the next precharge-evaluate operation begin. As a result, each PLA spends a substantial amount of idle time in either a fully precharged mode (awaiting input data from its predecessors) or a fully evaluated mode (waiting for its successor PLAs to finish their evaluation). The consequence is a shift in the minimum energy point: from around the threshold voltage for shallow networks to 2.5 times the threshold voltage for deep networks. The asynchronous micropipelined version in Ref. [53] differs in two significant ways from the clocked version. For one, it uses handshake signaling, but more importantly it uses latches to store the PLA input data—thus releasing predecessor PLAs from having to hold the data. As a result, the PLA spends just enough idle time in a fully precharged or a fully evaluated mode as is needed to complete the handshake protocol and to latch its data inputs or hand over its data outputs. Simulated examples show a factor of 4 lower energy and a factor of 7 higher throughput over the clocked counterparts, at the cost of 47% more area.

The main caveat in this comparison is that the clocked version might also be pipelined and become more energy-efficient by simply adding flip-flops or latches between the PLA stages. The real advantages of an asynchronous micropipelined network of PLAs come from the use of latches, the local latch control, and the ability to overlap precharge-evaluate operations with asymmetric precharge and evaluate times. Some of these have been discussed in Section 4.

Akgun et al.[55] designed a current-sensing completion detection system that replaces the worst-case delay line in an asynchronous bundled-data or single-rail circuit by a computation-dependent delay line. This work continues in the footsteps of Dean[56] and Lampinen,[57] who pioneered current-sensing completion detection to replace dual-rail circuits by lower-area and lower-latency single-rail circuits. The completion detection system by Akgun et al. can sense currents in the pA to nA range, which makes it suitable for completion sensing of subthreshold operations. It consists of three analog components: a *sensor transistor* senses the current drawn by the computation and converts this to a low-swing voltage signal, which is amplified by an *AC-coupled sense amplifier*, and then translated by a *monostable multivibrator* to a (high) voltage signal with a pulse width proportional to the completion time of the computation. This pulse is used as the matched-delay

bundled-data control signal. The goal of this technique by Akgun et al. is to minimize the idle time per operation so as to avoid unnecessary leakage energy. This tends to be more difficult to achieve in synchronous circuits, especially for data-dependent operations in the subthreshold domain. Simulation results for a 16-bit ripple-carry adder in a standard 0.18 μm process show a high correlation between the generated completion pulse and the actual computation time, and average delay improvements of 16% over the worst-case delay without sensing—which translates to shorter idle times and hence lower energy per operation.

5.3. Low Swing Signaling Techniques

Low-swing on-chip signaling techniques have been used effectively in synchronous design. In particular, they are used for long communication wires in combination with source-synchronous signaling.[50] However, the (always-sensing) clock-restore circuit that recovers the clock signal from the low-swing signaling scheme in Ref. [50] can be used equally well to recover the request signal in an asynchronous single-rail low-swing transaction. The full-swing restored clock or request signal subsequently causes the simple clock-enabled sense amplifiers to sample and amplify the low-swing data signals. This scheme needs an extra reduced power supply at the transmitter end of the wire.

Recent developments in the asynchronous group at Sun Microsystems show that low voltage swings can also be generated via capacitive coupling without the need for an extra power supply.[51] This approach works equally well for synchronous and asynchronous designs. It is less clear how to improve the receiver end of the wire. The key question here is: how low can the voltage swing go and still be reliably detected in the presence of noise and delay variations on the communication wires. In particular: how low a swing can reliably deliver an un-clocked signal, like the clock or request signal in the above example, is still an open question.

Lastly, asynchronous completion detection coupled with low-swing techniques may also be used to reduce the leakage energy in variable delay computations. The general idea is to turn off the voltage or current source and prevent a further increase in swing as soon as the output data have been latched.[58]

6. LEAKAGE REDUCTION

As mentioned in Section 5, leakage power is increasing to such an extent that static power and energy consumption are becoming a significant factor that must be minimized. The voltage scaling techniques discussed in Section 5 reduce both the dynamic and the static power, but other techniques to reduce static power have also been developed, including power gating, dual and multiple threshold voltages, and back-body biasing. This section reviews these techniques and briefly touches on their potential applicability in asynchronous design.

Power gating involves using additional large high-threshold transistors that act as switches to turn off the power supply when the associated module implemented with low-threshold transistors is idle.[63] This multi-threshold CMOS (MTCMOS) style requires special state-holding registers implemented with high-threshold transistors that *remain on* to save key state when the power to the rest of the logic is switched off.[70] In addition, special isolation buffers are necessary to avoid floating inputs at the boundary between voltage islands. The transistors in these special switches and sleep-mode circuits must be sized to account for both IR drop and ground bounce. Lastly, sleep protocols are needed to turn these sleep-mode mechanisms on and off. Power gating is becoming well supported in commercial synchronous design flows and can reduce leakage currents by many orders of magnitude at the cost of significant area and some performance. The techniques can be applied at a fine-grain scale within a module or at a coarse-grain scale within a multi-core SoC or NoC system such as described in Sections 3 and 5.1. How easy it will be to adopt these techniques for asynchronous domains is still an open question.

Dual-threshold techniques have also been very useful in developing low-leakage memory cells for SRAMs (e.g., Refs. [67, 69]). In addition, cell libraries have been developed which include multiple cells of the same functionality with different power supply thresholds, enabling a tradeoff between delay and leakage power. New synthesis and place-and-route tools use multiple objective functions across multiple process corners, to select a proper mix of cell types. Asynchronous circuits that use traditional SRAM structures with asynchronous wrappers can immediately take advantage of these advanced cell designs. Moreover, asynchronous flows that adopt standard-cell libraries and standard synthesis and place-and-route tools can also immediately benefit from these newer libraries.

Progress has also been made in the full-custom domain, for instance with the development of low-leakage techniques for domino logic (e.g., Refs. [64, 65]). Interestingly, the performance and area costs of these techniques can be reduced using a mixture of low-threshold and high-threshold transistors within the logic structure. This can be done such that the high-threshold transistors affect only the non-critical pre-charge delay. The extension of these techniques to asynchronous design styles is an open area of research. New issues associated with asynchronous domino logic include the development of asynchronous sleep protocols and state isolation and preservation with integrated storage.

An alternative to dual- and multiple-threshold cells is back-body biasing via dynamic-threshold CMOS

(DTCMOS) which involves dynamically adjusting the threshold voltage by changing the substrate voltage.[66] This technique has been used effectively in both silicon-on-insulator and bulk CMOS technologies, particularly in SRAM memories (e.g., Refs. [69, 68]). It requires a dynamically-adjustable threshold power supply and associated threshold power grid that distributes this supply to the substrate wells of all targeted transistors. DTCMOS has also been proposed as a solution for reducing delay variations of operations in the subthreshold region.[59] It can also serve to implement or augment the just-in-time computing techniques discussed in Section 5.

7. SUMMARY AND CONCLUSIONS

In summary, many of the power and energy benefits of asynchronous design come from its natural ability to consume energy only when and where needed. This applies for instance to computational blocks that can be readily bit-partitioned to save energy by activating only bits with significant data but it applies also to buffering and communication structures, such as FIFOs, busses, crossbars, and networks of routers. In some applications, the high performance achieved by data-driven asynchronous design yields impressive energy-delay results that are difficult to achieve in synchronous designs.

SoC and NoC communication structures can also take advantage of the low latency, integrated flow control, and reduced idle energy of asynchronous designs. In lower performance domains, the use of power-efficient normally-closed latches leads to low total energy consumption. In the subthreshold region, asynchronous completion detection techniques can help reduce idle times, mitigate increased process variations and limit voltage swings, thereby minimizing leakage currents, increasing yield and facilitating energy-efficient just-in-time computing. The lack of a global clock also facilitates the use of dynamic voltage scaling by removing the need to coordinate adjustments in supply voltage with clock speed. Lastly, many of the techniques for addressing static power should transfer over to asynchronous designs, but this remains an open area of research.

A perceived stumbling block for wide-spread adoption of asynchronous techniques has been the availability of commercially-supported design flows that accommodate asynchronous and GALS approaches and libraries that include advanced circuits used by higher performance asynchronous designs. However, there are many university research projects and a few companies that have already addressed this need or are actively addressing it, largely by adapting synchronous design tools and flows. Their solutions enable the seamless integration of asynchronous design techniques into otherwise synchronous design flows and facilitate the expanded use of asynchronous design in industry (e.g., Refs. [44–47]).

Acknowledgments: We appreciate the thoughtful review of our manuscript by Peter's student Ken Shiring, now with Cadence, Steve Burns, Michael Kishinevsky, Noel Menezes and Mark Schuelein of Intel Corporation, Rachna Goel and Ad Peeters of Handshake Solutions, John Bainbridge of Silistix, and Montek Singh of the University of North Carolina. We also thank Ivan Sutherland for helping us use the terms "energy" and "power" consistently, and for accepting—to his regret—our use of "open" and "closed" for latches.

References

1. J. Sparsø and S. Furber (eds.), Principles of Asynchronous Circuit Design: A Systems Perspective, Kluwer Academic Publishers **(2001)**.
2. A. Peeters and K. van Berkel, Single-rail handshake circuits. *Proceedings of the Second Working Conference on Asynchronous Design Methodologies* **(1995)**, pp. 53–62.
3. K. van Berkel, R. Burgess, J. Kessels, M. Roncken, F. Schalij, and A. Peeters, Asynchronous circuits for low power: A DCC error corrector. *IEEE Design & Test of Computers* 11, 22 **(1994)**.
4. K. van Berkel, R. Burgess, J. Kessels, A. Peeters, M. Roncken, F. Schalij, and R. van de Wiel, A single-rail re-implementation of a DCC error detector using a generic standard-cell library. *Proceedings of the Second Working Conference on Asynchronous Design Methodologies*, May **(1995)**, pp. 72–79.
5. A. M. G. Peeters, Single-Rail Handshake Circuits. Ph.D. Thesis, Eindhoven University of Technology **(1996)**.
6. S. Tugsinavisut, Y. Hong, D. Kim, K. Kim, and P. A. Beerel, Efficient asynchronous bundled-data pipelines for DCT matrix-vector multiplication. *IEEE Transactions on VLSI Systems* 13, 448 **(2005)**.
7. L. S. Nielsen and J. Sparso, A low-power asynchronous data-path for a FIR filter bank. *Proceedings of the International Symposium on Advanced Research in Asynchronous Circuits and Systems (ASYNC)* **(1996)**, pp. 197–207.
8. L. S. Nielsen and J. Sparso, Designing asynchronous circuits for low power: An IFIR filter bank for a digital hearing aid. *Proceedings of the IEEE* 87, 268 **(1999)**.
9. V. Ekanayake, C. Kelly, IV, and R. Manohar, BitSNAP: Dynamic significance compression for a low-energy sensor network asynchronous processor. *Proceedings of the International Symposium on Asynchronous Circuits and Systems (ASYNC)* **(2005)**, pp. 144–154.
10. R. Manohar, Width-adaptive data word architectures. *Proceedings of the 19th Conference on Advanced Research in VLSI* **(2001)**, pp. 112–129.
11. D. Fang and R. Manohar, Non-uniform access asynchronous register files. *Proceedings of the International Symposium on Asynchronous Circuits and Systems (ASYNC)* **(2004)**, pp. 75–85.
12. A. J. Martin, M. Nystrom, and P. I. Penzes, Et^2: A metric for time and energy efficiency of computation, Power-Aware Computing, Kluwer Academic Publishers **(2001)**.
13. H. van Gageldonk, K. van Berkel, A. Peeters, D. Baumann, D. Gloor, and G. Stegmann, An asynchronous low-power 80C51 microcontroller. *Proceedings of the International Symposium on Asynchronous Circuits and Systems (ASYNC)* **(1998)**, pp. 96–107.
14. 80C51-based 8-bit microcontrollers: Data handbook (IC20). Philips Semiconductors **(1997)**.
15. J. Kessels and P. Marston, Designing asynchronous standby circuits for a low-power pager. *Proceedings of the International Symposium on Advanced Research in Asynchronous Circuits and Systems (ASYNC)* **(1997)**, pp. 268–278.

16. A. J. Martin, M. Nystrom, and C. G. Wong, Three generations of asynchronous microprocessors. *IEEE Design and Test of Computers* 20, 9 **(2003)**.

17. R. O. Ozdag and P. A. Beerel, A channel based asynchronous low power high performance standard-cell based sequential decoder implemented with QDI templates. *Proceedings of the International Symposium on Asynchronous Circuits and Systems (ASYNC)* **(2004)**, pp. 187–197.

18. R. O. Ozdag and P. A. Beerel, An asynchronous low-power high-performance sequential decoder implemented with QDI templates. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* 14, 975 **(2006)**.

19. S. K. Singh, P. Thiennviboon, R. O. Ozdag, S. Tugsinavisut, P. A. Beerel, and K. M. Chugg, Algorithm and circuit co-design for a low-power sequential decoder. *Conference Record of the Asilomar Conference on Signals, Systems, and Computers* **(1999)**, pp. 389–394.

20. A. M. Lines, Pipelined asynchronous circuits. Master's Thesis, California Institute of Technology, Record Number CaltechCSTR:1998.cs-tr-95-21 **(1995)**.

21. M. R. Greenstreet and K. Steiglitz, Bubbles can make self-timed pipelines fast. *The Journal of VLSI Signal Processing* 2, 139 **(1990)**.

22. A. J. Martin, M. Nyström, K. Papadantonakis, P. I. Pénzes, P. Prakash, C. G. Wong, J. Chang, K. S. Ko, B. Lee, E. Ou, J. Pugh, E.-V. Talvala, J. T. Tong, and A. Tura, The Lutonium: A sub-nanojoule asynchronous 8051 microcontroller. *Proceedings of the International Symposium on Asynchronous Circuits and Systems (ASYNC)* **(2003)**, pp. 14–23.

23. P. Golani, G. D. Dimou, M. Prakash, and P. A. Beerel, Design of a high-speed asynchronous turbo decoder. *Proceedings of the International Symposium on Asynchronous Circuits and Systems (ASYNC)* **(2007)**, pp. 49–59.

24. S. B. Furber and P. Day, Four-phase micropipeline latch control circuits. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* 4, 247 **(1996)**.

25. I. E. Sutherland, Micropipelines. *Communications of the ACM* 32, 720 **(1989)**.

26. K. Y. Yun, P. A. Beerel, and J. Arceo, High-performance asynchronous pipeline circuits. *Proceeding of the International Symposium on Advanced Research in Asynchronous Circuits and Systems (ASYNC)* **(1996)**, pp. 17–28.

27. M. Singh and S. M. Nowick, MOUSETRAP: High-speed transition-signaling asynchronous pipelines. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* 15, 684 **(2007)**.

28. I. Sutherland and S. Fairbanks, GasP: A minimal FIFO control. *Proceedings of the International Symposium on Asynchronous Circuits and Systems (ASYNC)* **(2001)**, pp. 46–53.

29. T. Chelcea, G. Venkataramani, and S. C. Goldstein, Self-resetting latches for asynchronous micropipelines. *Proceedings of the Design Automation Conference (DAC)* **(2007)**, pp. 986–989.

30. M. Lewis, J. Garside, and L. Brackenbury, Reconfigurable latch controllers for low power asynchronous circuits. *Proceedings of the International Symposium on Advanced Research in Asynchronous Circuits and Systems (ASYNC)* **(1999)**, pp. 27–36.

31. S. Shibatani and A. H. C. Li, Pulse-latch approach reduces dynamic power. *EE-Times* **(2006)**, http://www.eetimes.com/showArticle.jhtml?articleID=190400564.

32. D. Stasiak, R. Chaudhry, D. Cox, S. Posluszny, J. Warnock, S. Weitzel, D. Wendel, and M. Wang, Cell processor low-power design methodology. *IEEE Micro* 25, 71 **(2005)**.

33. S. E. Schuster and P. W. Cook, Low-power synchronous-to-asynchronous-to-synchronous interlocked pipelined CMOS circuits operating at 3.3–4.5 GHz. *IEEE Journal of Solid-State Circuits* 38, 622 **(2003)**.

34. D. Harris, Skew-Tolerant Circuit Design, Morgan Kaufmann Publishers, Inc. **(2001)**.

35. T. Mudge, Power: A first-class architectural design constraint. *IEEE Computer* 34, 52 **(2001)**.

36. J. Cortadella and M. Kishinevsky, Synchronous elastic circuits with early evaluation and token counterflow. *Proceedings of the Design Automation Conference (DAC)* **(2007)**, pp. 416–419.

37. H. M. Jacobson, P. N. Kudva, P. Bose, P. W. Cook, S. E. Schuster, E. G. Mercer, and C. J. Myers, Synchronous interlocked pipelines. *Proceedings of the International Symposium on Asynchronous Circuits and Systems (ASYNC)* **(2002)**, pp. 3–12.

38. R. Canal, A. González, and J. E. Smith, Value compression for efficient computation. Euro-Par, Lecture Notes in Computer Science (LNCS) 3648, Springer-Verlag **(2005)**, pp. 519–529.

39. R. Canal, A. González, and J. E. Smith, Very low power pipelines using significance compression. *Proceedings of the International Symposium on Microarchitecture (MICRO)* **(2000)**, pp. 181–190.

40. D. Brooks and M. Martonosi, Dynamically exploiting narrow width operands to improve processor power and performance. *Proceedings of the International Symposium on High-Performance Computer Architecture (HPCA)* **(1999)**, pp. 13–22.

41. L. Villa, M. Zhang, and K. Asanovic, Dynamic zero compression for cache energy reduction. *Proceedings of the International Symposium on Microarchitecture (MICRO)* **(2000)**, pp. 214–220.

42. N. S. Kim, T. Austin, and T. Mudge, Low-energy data cache using sign compression and cache line bisection. *2nd Annual Workshop on Memory Performance Issues (WMPI)* **(2002)**.

43. M. Ghosh, W. Shi, and H.-H. S. Lee, CoolPression – a hybrid significance compression technique for reducing energy in caches. *Proceedings of the International SoC Conference* **(2004)**, pp. 399–402.

44. Handshake Solutions, www.handshakesolutions.com

45. Fulcrum Microsystems, www.fulcrummicro.com.

46. Achronix Semiconductor, www.achronix.com.

47. Silistix, www.silistix.com.

48. H. Zhang, V. George, and J. M. Rabaey, Low-swing on-chip signaling techniques: Effectiveness and robustness. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* 8, 264 **(2000)**.

49. B. Zhai, D. Blaauw, D. Sylvester, and K. Flautner, Theoretical and practical limits of dynamic voltage scaling. *Proceedings of the Design Automation Conference (DAC)* **(2004)**, pp. 868–873.

50. K. Lee, S.-J. Lee, and H.-J. Yoo, Low-power network-on-chip for high-performance SoC design. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* 14, 148 **(2006)**.

51. R. Ho, I. Ono, F. Liu, R. Hopkins, A. Chow, J. Schauer, and R. Drost, High-speed and low-energy capacitively-driven on-chip wires. *Digest of Technical Papers of the International Solid-State Circuits Conference (ISSCC)* **(2007)**, pp. 412–612.

52. B. H. Calhoun, A. Wang, N. Verma, and A. Chandrakasan, Subthreshold design: The challenges of minimizing circuit energy. *Proceedings of the 2006 International Symposium on Low Power Electronics and Design (ISLPED)* **(2006)**, pp. 366–368.

53. N. Jayakumar, R. Garg, B. Gamache, and S. P. Khatri, A PLA based asynchronous micropipelining approach for subthreshold circuit design. *Proceedings of the Design Automation Conference (DAC)* **(2006)**, pp. 419–424.

54. N. Jayakumar and S. R. Khatri, Minimum energy near-threshold network of PLA based design. *Proceedings of the International Conference on Computer Design (ICCD)* **(2005)**, pp. 399–404.

55. O. C. Akgun, Y. Leblebici, and E. A. Vittoz, Current sensing completion detection for subthreshold asynchronous circuits. *Proceedings of the European Conference on Circuit Theory and Design (ECCTD)* **(2007)**, pp. 376–379.

56. M. E. Dean, D. L. Dill, and M. Horowitz, Self-timed logic using current-sensing completion detection (DSCD). *The Journal of VLSI Signal Processing* 7, 7 **(1994)**.

57. H. Lampinen, P. Perala, and O. Vainio, Design of a self-timed asynchronous parallel FIR filter using CSCD. *Proceedings of the*

International Symposium on Circuits and Systems (ISCAS) 5, V-165 **(2003)**.

58. L. P. Alarcón, T.-T. Liu, M. D. Pierson, and J. M. Rabaey, Exploring very low-energy logic: A case study. *J. Low Power Electronics* 3, 223 **(2007)**.

59. N. Jayakumar and S. P. Khatri, A variation-tolerant sub-threshold design approach. *Proceedings of the Design Automation Conference (DAC)* **(2005)**, pp. 716–719.

60. L. S. Nielsen, C. Niessen, J. Sparsø, and K. van Berkel, Low-power operation using self-timed circuits and adaptive scaling of the supply voltage. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* 2, 391 **(1994)**.

61. K. Niyogi and D. Marculescu, Speed and voltage selection for GALS systems based on voltage-frequency islands. *Proceedings of the Asia and South Pacific Design Automation Conference (ASP-DAC)* 1, 292 **(2005)**.

62. U. Y. Ogras, R. Marculescu, P. Choudhary, and D. Marculescu, Voltage-frequency island partitioning for GALS-based networks-on-chip. *Proceedings of the Design Automation Conference (DAC)* **(2007)**, pp. 110–115.

63. J. T. Kao and A. P. Chandrakasan, Dual-threshold voltage techniques for low-power digital circuits. *IEEE Journal of Solid-State Circuits* 35, 1009 **(2000)**.

64. S. Heo and K. Asanovic, Leakage-biased domino circuits for dynamic fine-grain leakage reduction. *Digest of Technical Papers of the Symposium on VLSI Circuits* 316 **(2002)**.

65. V. Kursun and E. G. Friedman, Sleep switch dual threshold voltage domino logic with reduced standby leakage current. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* 12, 485 **(2004)**.

66. F. Assaderaghi, S. Parke, D. Sinitsky, J. Bokor, P. K. Ko, and C. Hu, A dynamic threshold voltage MOSFET (DTMOS) for very low voltage operation. *IEEE Electron Device Letters* 15, 510 **(1994)**.

67. S. Yang, W. Wolf, W. Wang, N. Vijaykrishnan, and Y. Xie, Low-leakage robust SRAM cell design for sub-100nm technologies. *Proceedings of the Asia and South Pacific Design Automation Conference (ASP-DAC)* **(2005)**, pp. 18–21.

68. A. J. Bhavnagarwala, A. Kapoor, and J. D. Meindl, Dynamic-threshold CMOS SRAM cells for fast, portable applications. *Proceedings of the Annual International Conference on ASIC/SOC* **(2000)**, pp. 359–363.

69. N. Azizi, F. N. Najm, and A. Moshovos, Low-leakage asymmetric-cell SRAM. *IEEE Transactios on Very Large Scale Integration (VLSI) Systems* 11, 701 **(2003)**.

70. M. Keating, D. Flynn, R. Aitken, A. Gibbons, and K. Shi, Low power methodology manual for system-on-chip design, Series on Integrated Circuits and Systems, Springer-Verlag **(2007)**.

71. A. Lines, Nexus: An asynchronous crossbar interconnect for synchronous system-on-chip designs. *Proceedings of the Symposium on High Performance Interconnects* **(2003)**, pp. 2–9.

72. T. Chelcea and S. M. Nowick, Robust interfaces for mixed-timing systems with application to latency-insensitive protocols. *Proceedings of the Design Automation Conference (DAC)* **(2001)**, pp. 21–26.

73. R. Mullins and S. Moore, Demystifying data-driven and pausible clocking schemes. *Proceedings of International Symposium on Asynchronous Circuits and Systems (ASYNC)* **(2007)**, pp. 175–185.

74. A. Chakraborty and M. R. Greenstreet, Efficient self-timed interfaces for crossing clock domains. *Proceeding of the International Symposium on Asynchronous Circuits and Systems (ASYNC)* **(2003)**, pp. 78–88.

75. D. S. Bormann and P. Y. K. Cheung, Asynchronous wrapper for heterogeneous systems. *Proceedings of the International Conference on Computer Design (ICCD)* **(1997)**, pp. 307–314.

76. D. S. Bormann, GALS test chip on 130 nm process. *Proceedings of the Second Workshop on Globally Asynchronous, Locally Synchronous Design (FMGALS)*, Elsevier Electronic Notes in Theoretical Computer Science **(2006)**, Vol. 146, pp. 29–40.

77. F. K. Gurkaynak, S. Oetiker, H. Kaeslin, N. Felber, and W. Fichtner, GALS at ETH Zurich: Success or failure. *Proceedings of the International Symposium on Asynchronous Circuits and Systems (ASYNC)* **(2006)**, pp. 150–159.

78. M. Krstic, E. Grass, and C. Stahl, Request-driven GALS technique for wireless communication system. *Proceedings of the International Symposium on Asynchronous Circuits and Systems (ASYNC)* **(2005)**, pp. 76–85.

79. Y. Zhiyi, M. Meeuwsen, R. Apperson, O. Sattari, M. Lai, J. Webb, E. Work, T. Mohsenin, M. Singh, and B. Baas, An asynchronous array of simple processors for DSP applications. *Digest of Technical Paper of the International Solid-State Circuits Conference (ISSCC)* **(2006)**, pp. 1696–1705.

80. Real Intent, Inc., http://www.realintent.com.

81. E. Beigne and P. Vivet, Design of on-chip and off-chip interfaces for a GALS NoC architecture. *Proceedings of the International Symposium on Asynchronous Circuits and Systems (ASYNC)* **(2006)**, pp. 172–181.

82. W. J. Dally and B. Towles, Route packets, not wires: On-chip interconnection networks. *Proceedings of the Design Automation Conference (DAC)* **(2001)**, pp. 684–689.

83. L. Benini and G. De Micheli, Networks on chips: A new SoC paradigm. *IEEE Computer* 35, 70 **(2002)**.

84. L. Benini, Application specific NoC design. *Proceedings Design Automation and Test in Europe (DATE)* **(2006)**, pp. 1–5.

85. A. Lines, Asynchronous interconnect for synchronous SoC design. *IEEE Micro* 24, 32 **(2004)**.

86. D. Lattard, E. Beigne, C. Bernard, C. Bour, F. Clermidy, Y. Durand, J. Durupt, D. Varreau, P. Vivet, P. Penard, A. Bouttier, and F. Berens, A telecom baseband circuit based on an asynchronous network-on-chip. *Digest of Technical Papers of the International Solid-State Circuits Conference (ISSCC)* **(2007)**, pp. 258–601.

87. E. Beigne, F. Clermidy, P. Vivet, A. Clouard, and M. Renaudin, An asynchronous NOC architecture providing low latency service and its multi-level design framework. *Proceedings of the International Symposium on Asynchronous Circuits and Systems (ASYNC)* **(2005)**, pp. 54–63.

88. J. Bainbridge and S. Furber, Chain: A delay-insensitive chip area interconnect. *IEEE Micro* 22, 16 **(2002)**.

89. A. M. Scott, M. E. Schuelein, M. Roncken, J.-J. Hwan, J. Bainbridge, J. R. Mawer, D. L. Jackson, and A. Bardsley, Asynchronous on-chip communication: Explorations on the Intel® PXA27x processor peripheral bus. *Proceedings of the International Symposium on Asynchronous Circuits and Systems (ASYNC)* **(2007)**, pp. 60–72.

90. T. Felicijan, J. Bainbridge, and S. Furber, An asynchronous low latency arbiter for Quality of Service (QoS) applications. *Proceedings of the International Conference on Microelectronics (ICM)* **(2003)**, pp. 123–126.

91. D. Gebhardt and K. S. Stevens, Elastic flow in an application specific network-on-chip. *Third International Workshop on Formal Methods in Globally Asynchronous Locally Synchronous Design (FMGALS)*, Elsevier Electronic Notes in Theoretical Computer Science **(2007)**.

92. J. You, Y. Xu, H. Han, and K. S. Stevens, Performance evaluation of elastic GALS interfaces and network fabric. *Third International Workshop on Formal Methods in Globally Asynchronous Locally Synchronous Design (FMGALS)*, Elsevier Electronic Notes in Theoretical Computer Science **(2007)**.

93. J. Cortadella, M. Kishinevsky, and B. Grundmann, Synthesis of synchronous elastic architectures. *Proceedings of the Design Automation Conference (DAC)* **(2006)**, pp. 657–662.

94. T. Bjerregaard and S. Mahadevan, A survey of research and practices of network-on-chip. *ACM Computing Surveys (CSUR)* 38, 1 **(2006)**.
95. E. Brunvand, Low latency self-timed flow-through FIFOs. *Proceedings of the Conference on Advanced Research in VLSI* **(1995)**, pp. 76–90.
96. T. Chelcea and S. M. Nowick, Low-latency asynchronous FIFO's using token rings. *Proceedings of the International Symposium on Advanced Research in Asynchronous Circuits and Systems (ASYNC)* **(2000)**, pp. 210–220.
97. K. van Berkel and M. Rem, VLSI programming of asynchronous circuits for low power, Asynchronous Digital Circuit Design, Workshops in Computing, edited by G. M. Birtwistle and A. Davis, Springer-Verlag **(1995)**, pp. 152–210.
98. J. Ebergen, Squaring the FIFO in GasP. *Proceedings of the International Symposium on Asynchronous Circuits and Systems (ASYNC)* **(2001)**, pp. 194–205.
99. A. Bink and R. York, ARM996HS, the first licensable, clockless 32-bit processor core. *IEEE Micro* 27, 58 **(2007)**.
100. K. van Berkel and A. Bink, Single-track handshake signaling with application to micropipelines and handshake circuits. *Proceedings of the International Symposium on Advanced Research in Asynchronous Circuits and Systems (ASYNC)* **(1996)**, pp. 122–133.

Peter A. Beerel

Peter A. Beerel *received his B.S.E. degree in Electrical Engineering from Princeton University, Princeton, NJ, in 1989 and his M.S. and Ph.D. degrees in Electrical Engineering from Stanford University, Stanford, CA, in 1991 and 1994, respectively. He joined the Department of Electrical Engineering—Systems at USC in 1994, where he is currently an Associate Professor. He is also the Faculty Director of Innovation Studies at the USC Stevens Institute for Innovation. He was on leave of absence from USC between June 2002 to September 2004 during which time he was Vice-President of CAD and Verification at Fulcrum. He remains a technical advisor to the company. He has been a member of the technical program committee for the International Symposium on Advanced Research in Asynchronous Circuits and Systems since 1997, was Program Co-chair for ASYNC'98, and was General Co-chair for ASYNC'07. His research interests include a variety of topics in CAD and asynchronous VLSI design. Dr. Beerel was a recipient of an Outstanding Teaching Award in 1997 and the Junior Research Award in 1998, both from USC's School of Engineering. He received a National Science Foundation (NSF) Career Award and a 1995 Zumberge Fellowship. He was also co-winner of the Charles E. Molnar award for two papers published in ASYNC'97 that best bridged theory and practice of asynchronous system design and was a co-recipient of the best paper award in ASYNC'99.*

Marly E. Roncken

Marly E. Roncken *received her M.Sc. degree in Mathematics and Computer Science from the University of Utrecht, The Netherlands, in 1985. From 1985 to 1997, she worked as a researcher at Philips Research Laboratories Eindhoven, The Netherlands, in the area of VLSI Design Automation and Test. At Philips, she was responsible for test operations and test research and development in the VLSI Programming and Silicon Compilation Project for asynchronous circuits (Tangram). In 1997, she joined Strategic CAD Laboratories at Intel Corporation in Hillsboro, OR, where she investigated the testability of high-speed self-timed domino circuits in the context of the RAPPID asynchronous design project. She is currently Intel's Researcher in Residence at the University of California, Berkeley, for the Gigascale Systems Research Center (GSRC). Her latest research focuses on Network on Chip and asynchronous VLSI design for nanometer technologies. Marly Roncken was Program Co-chair for ASYNC'02 and General Co-Chair for ASYNC'07. She is a recipient of the best paper award for the first conference on Asynchronous Circuits and Systems, ASYNC'94, and a co-recipient of the best paper award in ASYNC'99.*