The Future of Formal Methods and GALS Design

Kenneth S. Stevens Daniel Gebhardt Junbok You Yang Xu
Vikas Vij Shomit Das Krishnaji Desai

Electrical and Computer Engineering
University of Utah
Salt Lake City, U.S.A

Abstract

The System-on-Chip era has arrived, and it arrived quickly. Modular composition of components through a shared interconnect is now becoming the standard, rather than the exotic. Asynchronous interconnect fabrics and globally asynchronous locally synchronous (GALS) design has been shown to be potentially advantageous. However, the arduous road to developing asynchronous on-chip communication and interfaces to clocked cores is still nascent. This road of converting to asynchronous networks, and potentially the core intellectual property block as well, will be rocky. Asynchronous circuit design has been employed since the 1950's. However, it is doubtful that its present form will be what we will see 10 years hence. This treatise is intended to provoke debate as it projects what technologies will look like in the future, and discusses, among other aspects, the role of formal verification, education, the CAD industry, and the ever present tradeoff between greed and fear.

Keywords: Asynchronous design, network fabrics, globally asynchronous locally synchronous design, system-on-chip, FIFOs

1 Introduction

From the perspective of the last 20 years it is hard to believe that asynchronous design methodologies were once a common computer design and architecture approach. The last two decades have been a golden era of clocked architectures and circuit design. However, history often repeats itself; often forgotten methods prove useful as technology evolves. Looking forward there appears to be substantial potential for asynchronous circuit and design technologies to emerge once again. This paper explores the current state of asynchronous design and how it might return to become a common design methodology in the semiconductor industry.

Bear in mind that even today up to half of the die area of many of our chips are already comprised of fully asynchronous components. This is particularly true for microprocessors where the memory arrays, such as SRAM used for cache, take up half of the die area and are fully asynchronous components. These asynchronous blocks have no clocked elements, and respond to requests with a faster cycle time

This paper is electronically published in
Electronic Notes in Theoretical Computer Science
URL: `www.elsevier.nl/locate/entcs`

than the clocked systems in which they reside. This allows a zero overhead integration of the asynchronous part into the clocked system. Further, one could even argue that flip-flops and latches are merely circuit realizations of asynchronous state machines, whose constraints are fully integrated into the traditional clocked flows and CAD. If such common asynchronous elements are already part of our design flow, perhaps it is not too unrealistic to expand the asynchronous components in our designs beyond the storage arrays.

This possible asynchronous revival is supported by the authors of the semiconductor technology road-map. This document predicts that a full 20% of designs will be driven by *handshake clocking* in 2012, rising to 40% by 2020 [31]. Handshake clocking is a politically correct way of saying that designs will be asynchronous. Such a steep penetration of this technology would be truly amazing and disruptive. This paper covers a potential evolutionary flow that will result in nearly half the designs being asynchronous by 2020.

2 An Asynchronous Revival

2.1 Scaling giveth and taketh away

A transformation has begun. Novel technology many times does not first manifest itself in established corporations. Such organizations tend to be more dominated by fear than greed. Perhaps this is the correct approach because more disruptive technologies fail than succeed. Thus startups tend to be the place where greed and risk taking can be fostered and properly rewarded. The asynchronous revival seems to be following this pattern. This technology has been nurtured largely at universities, with some continued support from industrial research labs. A recent spate of industry focused work has emerged [41,12,33,10,25,8,1]. Many of these appear to have gained traction and are likely to succeed. These should produce the first "asynchronous millionaires".

Both the demise and potential revival of asynchronous design can most likely be attributed to the same factor: transistor scaling [22]. Circuit design has two primary facets, module design and system integration. Clocked designs and asynchronous designs take opposite approaches to these two facets. Clocked design vastly simplifies the design of the modules, but does little to support system level integration and validation. Module design is challenging for asynchronous systems due to the requirement of implementing hazard-free protocols for each module. However, the protocols vastly enhance the ability to correctly design and verify systems. In general, the system level aspects of a design are the most difficult to optimize and verify. Thus, asynchronous design makes the simple facet of design hard, and the hard facet of design easier. Clocked design is the dual – it simplifies the easy part of design but makes the system level design much more challenging.

Starting 20 years ago, when systems were small, clocked design was clearly the best solution. Module design was the biggest hurdle to be solved. Logic synthesis was a boon to productivity, and was much easier to apply to clocked designs where hazards did not need to be avoided. As transistor counts have continued to exponentially double, the problem of designing efficient *systems* has become a serious challenge. System-on-Chip (SoC) design somewhat simplifies this system

as it introduces modularity. However, important system level integration issues are problematic with clocked design. Each SoC module can have its own frequency, and interfacing them using clocked paradigms is inefficient and awkward. Synchronizers between clock domains cost substantial energy and latency to a design, and all have a probability of failure due to the statistical properties of metastability that occur in these interfaces [17].

Scaling, after basically eliminating asynchronous design, is now the source of an asynchronous revival.

2.2 The Value Proposition

The potential benefits of this revival are clearly displayed in the asynchronous components that are already commonly used in clocked design flows: the memory elements. These are the lowest power parts of a design. Their modular interfaces enable easy integrate into nearly every design based on their protocol and timing requirements. These characteristics are common to nearly all asynchronous design blocks.

One would thus expect, given current technology, that the best place to first apply asynchronous design techniques is in the communication fabric between clocked intellectual property (IP) blocks. This is the concept of GALS: where globally asynchronous communication occurs between locally synchronous IP islands. Unfortunately there are *two* major impediments to successfully implementing GALS based designs. First, this is the *worst* application for asynchronous protocols. Second, interfacing to clocked modules as slaves is very inefficient due to the requirement to synchronize to the local clock.

2.2.1 Asynchronous Communication Costs

Asynchronous protocols all have overhead. These overheads primarily consist of two aspects. First, the protocols all use handshaking and implement flow control. This handshaking is based on the same sequence for every asynchronous protocol: a request is followed by an acknowledgment. The request indicates that new datum is ready for consumption, and the acknowledge notes that the consumer can accept the next datum.

The primary overhead of handshake protocols is the communication of the acknowledgment signal. For example, the function of a network fabric is to transmit data from one location to another on a silicon die. The latency and bandwidth are based on how long it takes to send the signal across the wires and repeaters, and how often a new data item can be launched. Assume a clocked and asynchronous communication system where the data wires and control wires (request and acknowledge signals) have the same latency. A two-cycle handshake communication protocol will reduce the throughput of the network fabric *in half* compared to the clocked system. This is because the propagation of the acknowledgment signal from receiver to sender takes just as long as the data to propagate from the sender to receiver. This overhead is even worse for four-cycle protocols where the return-to-zero phase further degrades throughput. If wire delays are negligible, such as in the IP cores themselves, this overhead can be reduced to a negligible value.

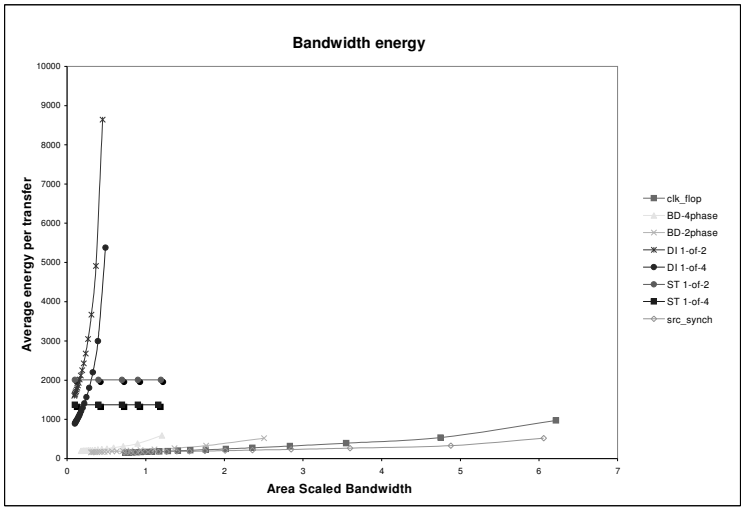

Fig. 1. Bandwidth and energy for clocked and asynchronous protocols across 10,000μm in 65nm design

A recent evaluation and characterization of these costs for many protocols is shown for a 65nm process in Figure 1 [37]. This graph shows area scaled bandwidth for four different classes of communication protocols: asynchronous delay insensitive (DI), single track (ST), and bundled data (BD & source synchronous) protocols, and clocked protocols. This graph shows the protocols under various levels of pipelining for a long $10,000\mu$m wire. The overhead of the protocol limits the range on the horizontal axis, and the energy is plotted on the vertical axis. Clearly, other than the asynchronous source synchronous protocol which removes the backward acknowledge handshake, the asynchronous handshake protocols have substantially lower bandwidth than the clocked protocols.

The horizontal axis shows that communication can potentially be a very inefficient application for asynchronous design, and other methods must be used to compensate for the overhead of handshake communication for network fabrics. For instance, if the flow control of the acknowledgment can be loosened, then more efficient protocols, such as source synchronous, can be implemented. This protocol is actually even more efficient than clocked protocols but requires substantial buffering at the interfaces as the acknowledge signal is delayed or covers data sets.

The second overhead, though potentially less severe, relates the data and control paths. In asynchronous design data must either be encoded as delay insensitive codes, or race paths exist between the handshake and control logic. For DI codes, there is significant overhead associated with the code generation in terms of both energy and performance. This is shown as the high energy profile and limited bandwidth of the delay insensitive protocols of Figure 1. For bundled data a small overhead exists that requires an additional margin to be taken to ensure the silicon race paths are resolved correctly. (For clocked design these are speed paths that can be resolved with the clock frequency.) The margins for bundled data can be somewhat mitigated by using time borrowing in the latches.

Note that these protocols can easily be interchanged in a design and between modular asynchronous block. Various interfacing protocols have been applied to memories, both on and off die. A dual-rail protocol with sense amplifiers is typically applied internally to SRAM arrays. Externally source synchronous protocol

is commonly applied between the memory array and the clocked module, and now between IP cores [20].

2.2.2 Synchronization Costs

The second problem with implementing a GALS based design methodology is the timing rigidity of the clocked design. This of itself produces two problems.

The first problem is that of creating valid data streams for a clocked design. Traditional clocked designs assume data is always valid and requires that the architecture define fixed latencies in terms of clock cycles for every path in the architecture. This approach enables efficient synchronization – but requires that this implicit synchronization information exist in the architecture in the form of constraining relationships between convergent signal paths and their respective delays in terms of the number of clock ticks. This allows synchronization through counting the number of clock cycles. Such an approach works well for small systems but is extremely problematic for large SoC designs and designs where wire delays are significant. The traditional approach to solve this problem in a GALS architecture is to design a system with stoppable clocks. It assumes data is valid on each clock tick for every path. If that is not the case the clock is stalled or stretched until all data is present.

The system level problems presented by cycle accurate data rigidity in clocked design is even too onerous for clocked approaches. Thus for complicated systems, interfaces such as the open core protocol (OCP) [26] are required as wrappers, similar to GALS wrappers, around the clocked designs. However, these normally also require modifications to the IP cores to support the data nondeterminism inherent in the protocols. This removes the requirement for cycle accurate data on every port at a considerable cost.

The second problem with the timing rigidity of clocks is one of synchronization into clocked domains. Synchronization is required when moving data into a clocked domain if the data is not from a the same clock or a related synchronous domain. Synchronization is the process of aligning data entering a clock domain with the local clock edges. This normally is done using special synchronizing flip flops that cost substantial latency and energy. Data synchronization is only necessary for clocked systems. For this reason, embedding asynchronous blocks such as SRAM into a clocked system can occur overhead free. A further drawback to synchronization is reliability. Each synchronizer has a mean time between failure when operating at a fixed response time. Asynchronous systems do not suffer this lack of reliability, but can no longer provide worst case response guarantees.

2.3 Summary

In summary, designing an asynchronous network fabric at first blush appears to address most of the issues with current design: it allows flexibility in frequency, has modular protocols, and should produce lower power results. However, this is the worst application of handshake protocols producing significant overhead. Thus GALS systems are not the clear design win that will easily prove the benefits of asynchronous methods.

The question going forward is how this revival will occur, if there are sufficient benefits for asynchronous design to merit the disruptive changes that would ensue, and if GALS is the correct place to start the revival.

In order to enable adoption of this rather disruptive technology, a significant benefit must be shown. Assume an aggregate improvement of 3× in terms of performance, power, and area. This paper discusses how such a benefit might be achieved with today's technology. Example designs that employ such methods have shown substantial improvements in power, performance, and latency [43,35]. We await to find such examples in the GALS space.

3 Predicting the Way Forward

The remainder of the paper consists of bold speculative predictions of what impact asynchronous and GALS design will have on the semiconductor industry by the year 2020. These predictions are based on the past, current technology and their particular merits and demerits, and assumes the continued presence of silicon and MOS transistors as the primary technology used to implement digital electronics. Making predictions is always a risky endeavor. Technology will evolve significantly between now and then, and may invalidate many of the assumptions. For example, when novel transistors that are not based on MOS technology evolve many of the predictions in this work will either become enhanced or obsolete based on the nature of the new disruptive technology.

3.1 Formal Verification Becomes Pervasive

The first prediction is that formal verification (FV) will move from a niche technology to one that is fundamental to the design flow. While humans have an innate ability to invent and create, they do a poor job of evaluating all possible corner conditions into which concurrent systems can evolve. Such a domain, with its reachability analysis, is the realm of formal verification tools that can prove conformance between an implementation and a specification or prove conditions on a design such as liveness and safety. Such a change to FV-based design will be painful and costly to the design industry, but will also greatly improve productivity and product costs as fewer product recalls will occur.

This change will be driven primarily by two factors. Both of these are a derivative of aggressive process scaling that we have experienced over the last 40 years. The first driver is due to the complexity of the function units that are currently being designed. A good example is the Intel FDIV instruction bug. The complexity is such that it is beyond the scope of model checking or other reachability analysis techniques [15]. Thus systems of higher order logics and theorem proving have been developed to prove the correctness [27]. The complexity of many of the function blocks in our designs will continue to require substantial research and development of FV techniques that can be applied to such design blocks.

The second driver for formal verification into the design flow is the increasingly modular nature of our designs. The focus in this domain is upon formal protocol verification. The increased modularity of our designs have resulted in formal protocols used to efficiently implement and verify communication between blocks

[40]. Asynchronous logic and SoC interface specifications are formal protocols. Any module that conforms to the specific protocol can be directly interconnected. The protocols range from simple to the complex. For instance, handshake communication protocols between pipeline stages can be expressed in a few lines of process logic or as a petri net. On the other hand, the OCP protocol specification is 340 pages long [26].

The correctness of the implementation of the particular protocol, as well as the properties of systems that result from the interconnection of these modules are the two focal points of protocol formal verification. Our work is focused on efficient algorithms using hierarchical verification based on observational equivalence that support nondeterminism so we can verify synchronizers between an asynchronous network and clocked IP cores [21,9,28].

A second aspect of formal verification which must be addressed is the relationship between system behavior and timing. Protocols can range from delay insensitive to timed protocols. In nearly every instance, performance and energy can be greatly increased by making a few timing assumptions [35]. Figure 1 is a good example, showing that making some timing assumptions allows one to move from delay insensitive data encoding to bundled data that results in vastly improved performance and energy of a point-to-point communication channel.

Modeling the timing of events in a protocol are becoming imperative as systems become more complex and optimizations show more value. Initially proving correctness will be to focus as there is significant difficulty in predicting the timing of events in a large system. This will evolve into timing optimization of large modular systems.

Temporal logics have been used for years to verify behavior and timing. However, these systems are very rigid in their ability to address timing. Relative timing is another approach that has more flexibility in timing [38,32]. Some combination of these approaches or new methods will be developed to prove system level correctness of designs.

Verifying asynchronous designs with timing to guarantee the correctness of the circuits suffers from lack of CAD tools, and is a sufficiently different problem that it is likely new algorithms and CAD must be developed. One good example is the the application of relative timing to integrate asynchronous logic and protocols into the traditional clocked ASIC tool flows [39]. Relative timing specifies the relative ordering of two race paths. A unique formal verification engine which supports relative timing employs bisimulation semantics between specification and implementation [36]. Each component in this system uses semi-modular specifications [23,14], and are modeled in CCS process language which supports nondeterminism. A relative timing constraint is applied to the system to be verified in order to remove circuit hazards by pruning the reachability of hazard states. An automatic relative timing identifier based on signal traces, called ARTIST, has been designed, implemented and then embedded into formal verification engine to release the designers from the heavy duty on hand-generated constraints.

Over 100 asynchronous circuits and protocols have been verified and proven to be correct under these RT constraints generated by ARTIST. Figure 2 shows a c-element designed out of NAND gates. ARTIST automatically generates the four

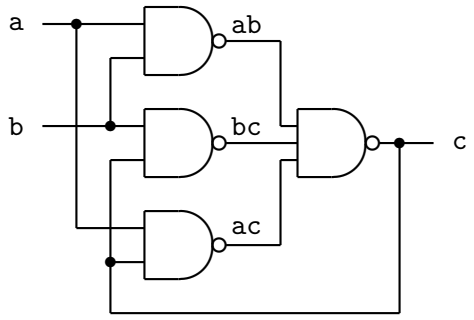

Fig. 2. C-Element implemented with NAND gates

necessary timing constraints based on race paths in the circuit: $c\uparrow \mapsto bc\downarrow \prec a\downarrow$, $c\uparrow \mapsto bc\downarrow \prec b\downarrow$, $c\uparrow \mapsto ac\downarrow \prec a\downarrow$, and $c\uparrow \mapsto ac\downarrow \prec b\downarrow$.

The increased focus on formal verification will result in several positive effects in the industry. This step will help ensure the correctness of all complicated modular systems. It will simplify the transition to asynchronous design and handshake clocking. Hopefully there will also be an improvement in the specifications themselves. The industry can not continue to live with ambiguous English based specifications. The Intel instruction set architecture may be an exception and continue to survive as a text based specification; but here there is a large body of experience that leads to specific well known interpretations of that document. OCP and other protocols will need to be recast in a formal specification language. Unfortunately, herein lies the challenge, as consensus on the correct representation will likely be contentious. There are many options. We particularly like process logics as a formal specification.

3.2 Asynchronous Design Slow Encroachment

Asynchronous design will slowly become integrated into standard design flows, as predicted by the ITRS [31]. We support the bold prediction that by 2020 40% of all designs will be asynchronous or "handshake clocked".

The process of achieving a penetration of handshake clocking in 40% of designs will be slow and arduous. This is primarily due to the difficulties pointed out in Section 2.2. The place to first achieve significant adoption of asynchronous design should be as an asynchronous network fabric in a GALS systems. However, since this is the worst application for asynchronous protocols, the transformation will be fraught with difficulties, challenges, and some false starts.

For any disruptive technology to be adopted there must be a significant advantage in the primary quality metrics of the design. The metrics in the semiconductor industry consist of the trifecta of performance, power, and area. (Throughput and latency are both considered performance metrics.) An aggregate improvement of 3× is sufficient for the cost of a disruptive technology to be developed and matured. Less than that and the adoption may not occur, or will occur at a very slow pace. Asynchronous design has shown such a potential [43,35]. The design of the front end of the Pentium processor showed an aggregate improvement of 10× over the clocked design on the same fabrication line. This included a 2× improvement in energy per instruction and latency, and a 3× improvement in throughput.

While this advantage will not be achievable in all designs, there will be a sig-

nificant set of designs that can be vastly improved with asynchronous design. One may rightly question whether the network fabric is even one of the places where such an achievement is even possible. This presents one of the primary challenges in adopting asynchronous design. If one simply implements an asynchronous design using the same thought process and architectures as a clocked design there will be little benefit, if any, in the process. Thus part of the process of achieving significant penetration in the design space is taking advantage of the asynchronous nature of handshake clocked protocols to obtain advantages in the primary design metrics of power, performance, and area. This is not necessarily apparent to traditional clocked designers, and will require education. Yet given the correct thought process, we claim that even in the network fabric there is significant potential that the 3× target can be achieved. One way to do this for the network fabric will be demonstrated in Section 4.1.

Like most disruptive technologies, adoption will likely start at the low performance end of the spectrum and eventually become adopted all the way up the product line. This seems to be the case with asynchronous design, as startups that appear to have highest likelihood of success are in the low end of the product line. One good example of this is Handshake Solutions which started targeting smart cards, and is moving up the product road-map.

3.3 FIFO Buffering Shifts to Asynchronous Logic

Memory arrays such as SRAM are large asynchronous blocks that have been seamlessly integrated into clocked design. These asynchronous blocks are so completely accepted by most designers that they no longer realize that these arrays use a substantially different clocking methodology. One could also argue that traditional register arrays are also a form of asynchronous design where the clocks become gated on data validity based on the operation and data values.

FIFOs are another important memory array class that is particularly useful for communication operations, such as in the network fabric of a GALS architecture. Recent data indicates that for nearly every data width and buffering capacity for single timing domains the asynchronous buffers tend to show better results on the primary design metrics (performance, power, area) than clocked FIFO designs. This is shown in Figure 3, were the clocked FIFOS EHB and H/T tend to be inferior solutions. More studies need to be performed to validate this when crossing clock boundaries. However, we expect the trend for this type of memory to be consistent with other memories; asynchronous implementations seem to be superior and will eventually become predominant.

3.4 Delay Insensitive Design Will Disappear

Delay insensitive design is based on the theory that if the handshake protocol is extended into the data path, then design correctness becomes a completely independent issue from design performance and timing. As noted earlier, design correctness is such a key issue we predict that formal verification will become a standard part of the design flow. If design correctness can be decoupled from performance, then the flow can be partitioned into two orthogonal steps: one creating the de-

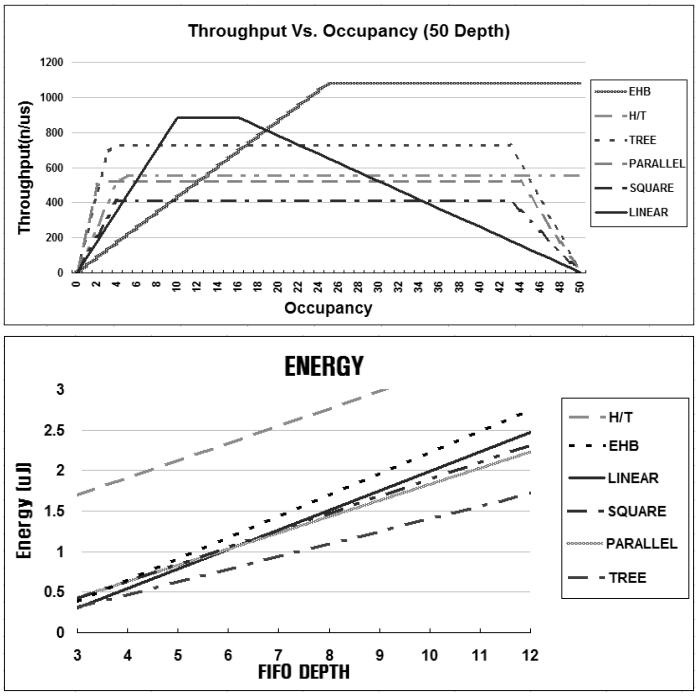

Fig. 3. Throughput and energy for clocked (H/T, EHB) and asynchronous (Tree, Parallel, Linear, Square) FIFO configurations

sign and methodology, and another optimizing performance and power. Thus every current commercial asynchronous company currently employs a delay insensitive design methodology, or started out using such a flow. This is also the case with GALS research [3,4].

Unfortunately the power and performance overheads of delay insensitive flows are so onerous that this design style will fade from existence. Mathematical models comparing different communication protocols show that the delay insensitive family of protocols suffer massive penalties for point-to-point communication links when compared to four cycle and two cycle bundled data protocols [37]. The first order models have been validated by simulation in 65nm process technologies. The DI protocols expend from 3 to 6× the energy of comparable four cycle communication, and up to 8× greater energy per transaction than dual rail protocols as shown in Figure 1. The maximum throughput is also only half that of four-cycle protocols only 30% the throughput of two cycle protocols.

Such huge overheads may make sense, but not if better solutions are available. The biggest problem is one of solving the timing driven synthesis and validation of asynchronous designs. At least two solutions have been found, including one using relative timing, which allow fully asynchronous designs to be timing optimized using traditional clocked CAD flows [39]. Another unpublished solution has been implemented by the successful asynchronous startup Handshake Solutions.

One can argue that process variation, as it continues to grow with scaling, may result in the requirement of turning to delay insensitive designs. While this may be true, the extent of the variation would need to be so great that this would largely spell the doom of the semiconductor industry, and that scaling would halt before this stage were reached.

Due to such a high overhead, one of the few successful asynchronous CAD and chip companies, Handshake Solutions, has evolved away from DI designs and now produces bundled data implementations. This trend will continue industry wide until DI design becomes nothing more than historical interest.

3.5 Data Validity will Invade Core Design

Another significant predicted change will occur in the design of the IP cores as explicit data validity information is introduced into the clocked pipelines. This information was removed early in the architecture of clocked designs as part of the mapping to clocked function units. Synchronization became explicit through cycle counting. However, as complexity of designs increase, this loss of information is becoming a serious impediment to system level design and the overhead of this choice is becoming apparent in many aspects of design.

The notion of data validity is required to cleanly interface IP to communication structures that are shared or have unpredictable latency. Thus interfaces to caches, shared busses, or communication fabrics in an SoC design require data validity aware interfaces. Today these interfaces require custom integration into the core IP of the IP. This is a complicated, time consuming, and error prone task. A much better alternative is to reintroduce data validity into the clocked pipeline.

Data validity can be cleanly introduced into a clocked architecture with mechanical design transformations. This can be achieved using clocked elastic design [6]. This method has several advantages. First, it does not require an error prone projection of an interface protocol such as OCP into the IP core. This mechanism introduces *valid* and *stall* signals at each pipeline interface. The second advantage is that once valid and stall handshakes have been introduced into the clocked pipeline, the interface to the network fabric is vastly simplified [44].

Introducing data validity mechanically into the clocked IP cores opens up a design to a whole family of transformations and optimizations that were never possible beforehand. Probably the first optimization will be to revolutionize how clock gating occurs. Once data validity information from the data path is employed, the efficiency of clock gating in synchronous designs can begin to rival that of asynchronous systems. This will result in substantial reductions in the power of clocked design.

One might assume that this could spell doom for asynchronous design, but instead it will bring about a revolution. Once clocked design utilizes a protocol similar to asynchronous design, interfacing the two methodologies becomes much simpler. This will be no more evident that in the GALS realm. Once every clocked IP core utilized data valid handshakes, the task of integrating the cores to an asynchronous network fabric will be greatly simplified. We will no longer require 300–500 page documents describing interfacing specifications as is currently the norm. The circuits will also operate faster and with lower latency.

An interesting possibility that this transformation will enable will be the translation of clocked IP cores into fully asynchronous design. This approach would be something similar to “desynchronization” that is currently being researched today, but will take advantage of data validity in the pipeline. Interestingly one could

imagine scenarios where such a transformation to the cores, from a power perspective, will have greater impact and motivation than the implementation of a clocked IP with an asynchronous network fabric.

3.6 Pausible Clock Design Will Disappear

One of the primary workhorses of GALS design has been pausible clocking. This style has been a core means of integrating traditional clocked IP into an asynchronous domain. However, the commercial world is now realizing that the IP cores require some sort of interface that does not make the over rigid assumption that correct data will be valid on all interfaces every clock cycle. Today many IP cores are being redesigned to interface with a common network protocol such as OCP. Such a step removes the need for a pausible clock as data at the interfaces now can have arbitrary latencies. This step will mitigate the need to pause the clock, and for the complicated clock wrappers or shells. As elastic clocked design is studied as a serious alternative, this previously fundamental approach, much like delay insensitive design, will also disappear.

3.7 Lightweight Interfaces Replace Complex Protocols

Clocked designs will continue to evolve to further mimic asynchronous designs. Nowhere will this be more apparent than in the interface between SoC modules as the IP inherits valid and stall protocols throughout the pipeline.

Initial interface designs started with very complex protocols in an attempt to shield and protect the clocked assumptions as sacrosanct. This included stoppable clock wrappers of GALS design, and the shells of latency insensitive protocols [19,7]. The next migration was to complex bus and network interfaces such as OCP. As clock designers realize that data valid handshake protocols can relatively easily be integrated into their designs, and that this creates a seamless interface to the rest of the system, these complex wrappers, shells, and interface designs no longer be designed for new IP. This transformation will be like no other because automated algorithms will be implemented to translate clocked design into elastic design. Even legacy designs can be rapidly converted without much non-recurring engineering costs.

Thus, the modularity of interfacing these designs to the network and other IP cores will be a major driving force for the introduction of protocols throughout clocked IP.

3.8 Commercial CAD Evolution for Asynchronous Support

There have been two impediments to the commercialization of asynchronous design. First, the diversity of the designs has been staggering, with little attempt to unify methodology or terminology. This lack of unity has led to many interesting research results, but insufficient collaboration to gel momentum around standard design styles and CAD. The second roadblock has been the need to develop custom CAD for the flow from top to bottom, largely in competition with the clocked CAD.

Just as clocked design is evolving to employ the advantageous aspects of asyn-

chronous design, asynchronous design is likewise evolving to integrate the advantageous aspects of clocked design. DI design will disappear, and bundled data asynchronous designs will become the defacto standard. The data path of such designs will utilize the best synthesis engines, which will come from commercial clocked CAD.

The primary impediment in this domain is related to system timing. However, it is interesting to note that (i) the timing of a bundled data asynchronous design and a clocked design is not fundamentally very different, and (ii) the timing assumptions of the clocked design have been directly integrated into the CAD tools. Thus, the commercial CAD vendors will slowly begin to improve support for asynchronous CAD in their products.

We feel that this will evolve by the asynchronous community first generating sets of constraints in the sdc format sufficient to synthesize and optimize designs. Currently this process is only partially supported in the tools, as commands such as set_data_check do a poor job of timing driven synthesis and optimization. The algorithms of these commands are also very different for each vendor, and at times differ between the tools of a single vendor.

The second aspect where commercial CAD vendors will be instrumental is in run-time optimization. This can take multiple forms. Due to the increased diversity of asynchronous design, we doubt that the vendors will try and internally automate timing as currently done in clocked designs, but that they will more fully support an sdc type format. One of the run-time problems faced in current flows is the bifurcation of the "clock" signal into many thousands of local clock domains with asynchronous design. While this presents a problem with current tools, the locality of the domains and the availability of multiple processing cores open an enticing scenario for efficient algorithm development.

3.9 New Programming Paradigms

The advent of asynchronous design presents some interesting shifts in the efficiency of programming our logic. We expect the design flow to be somewhat modified. First, there will be an additional part of the flow added to design, verify, and characterize asynchronous templates that will implement the asynchronous handshaking protocols. These will be integrated with latches or registers into the design. The data path will then look much like a traditional clocked data path.

Such a flow places a heavier burden on the designer for specifying communication channels between asynchronous templates. This can become very burdensome and error prone in current languages such as Verilog. However, this challenge presents a golden opportunity. The necessity to optimize the communication network is becoming increasingly problematic. This has been predicted to be so fundamental to future design that future designers will focus on the communication structure of the design with little attention to the functional aspect of the design [5]. This evolution to communication centric design should create a shift in chip programming to support such flows. We expect there first to be communication channel based languages, perhaps similar to System-C [18]. Next, we expect a language that will more directly support the optimization of communication [2].

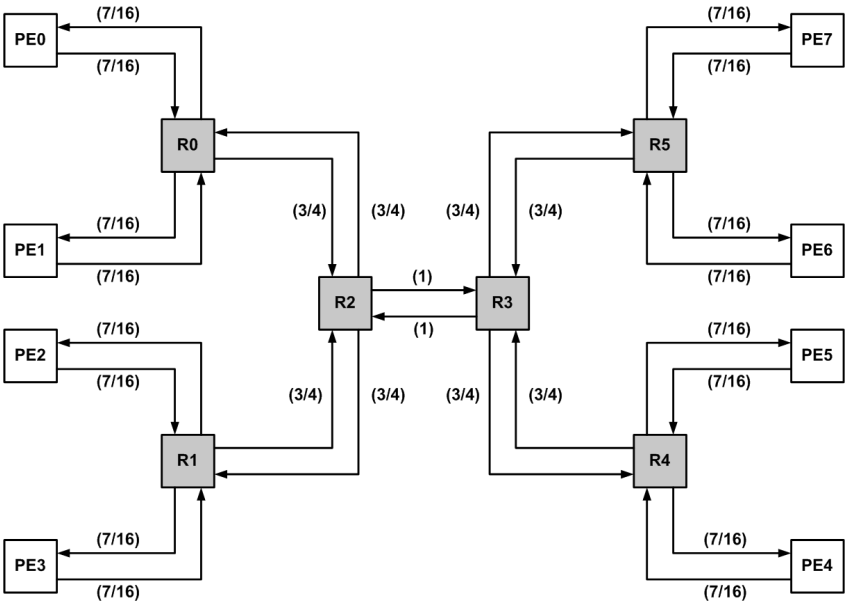

Fig. 4. 8 Node Binary Tree with Respective Bandwidth Requirements

4 Short GALS Example

This paper has outlined the evolution of asynchronous and GALS based technologies, as well as some of the resistance to the changes. Since the communication fabric is one of the key battlegrounds for this evolution, and also one of the most difficult, a short example of how this might occur will be demonstrated.

4.1 Think "Asynk"

Even though asynchronous protocols are inherently less efficient than clocked protocols from a per-link viewpoint, many of their advantages are well-suited to interconnects from a architecture viewpoint, especially with regards to power consumption. SoC traffic can be quite "bursty" and irregular, thus there is a power benefit to not clocking routers which have no data to switch. An asynchronous GALS network has lower power than the "multi-synchronous" approach for a 4G telephony research chip [29]. The cost of the clock is apparent in the synchronous NoC for the Intel 80-core Teraflops chip, where the clock distribution alone uses the highest percentage (33%) of total interconnect power [13], which itself is 28% of each core's total power.

One of the primary asynchronous design advantages is the ability to directly integrate various frequencies and bandwidths without overhead [35]. This is particularly interesting for a network fabric since different links will have different bandwidth requirements based on network connectivity and traffic patterns. Nowhere is this more apparent than in a binary tree topology where the network bisection passes through just a single link. This link, under uniformly distributed communication, would carry half of the bandwidth of the entire network.

One solution to this is to create multi-frequency network links where the cycle time of each link is inversely proportional to the amount of traffic that the link carries [11]. Figure 4 shows an 8 node network topology connected through a binary

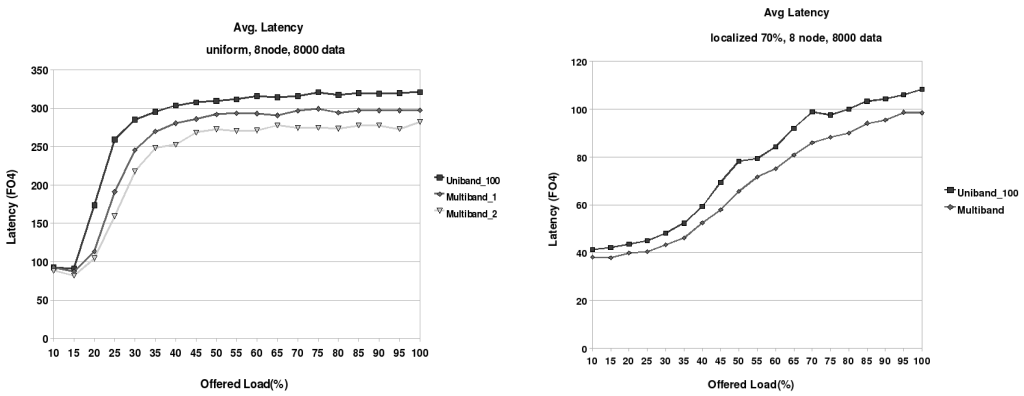

Fig. 5. Average Latency Comparison, Uniform Distribution (left) and 70% localized traffic (right)

Configuration	**Wire Latency**			**Link Cycle Times**		
single bandwidth	100ps	100ps	100ps	680ps	680ps	680ps
multi-bandwidth	145ps	80ps	50ps	860ps	600ps	480ps

Table 1
Cycle times of the single and multiple bandwidth network configurations

tree fabric. The bandwidth requirement on each node is listed in relation to the bandwidth of the center link for uniformly random distributed data.

Two asynchronous network fabrics can be compared to evaluate the benefit of applying multiple frequencies in the links of a network fabric. A multi-bandwidth binary tree topology is compared to a single bandwidth network for the above topology. The wire delay between each link is 100ps in the single frequency implementation, whereas it is scaled from 145ps, to 80ps, to 50ps for the multi-bandwidth case for an identical aggregate wire latency of 500ps. The only difference in these two designs is the physical placement of the nodes. In the single frequency case, the routers are evenly distributed along the path. In the multi-frequency case, the routers are more closely clustered near the center of the network. If the controllers have a logic latency of 280ps, the cycle time will be $280ps + 4 \times 100ps = 680ps$ for each single bandwidth link.

This structure makes the links of the IP cores approximately 25% slower but the center link approximately 30% higher bandwidth than the average link in the single frequency design. As can be seen from Figure 5, this substantially improves the average case latency by about 11%, and delays network congestion from approximately 15% load to 20% offered load.

Figure 5 shows the data in the graph where at the first router node 70% of the traffic is routed to the local node and 30% to the other nodes (rather than 12.5% and 87.5% for uniform random distribution). For this traffic distribution, latency limits can be reached at much higher offered loads. For instance, the average latency of 40 is delayed from 10% to 25% offered load, and the latency of 100, nearly reached at an offered load of 70%, is not reached at 100% load in the multi-bandwidth configuration.

Improvements to the buffering, protocol, and placement of the routers will result

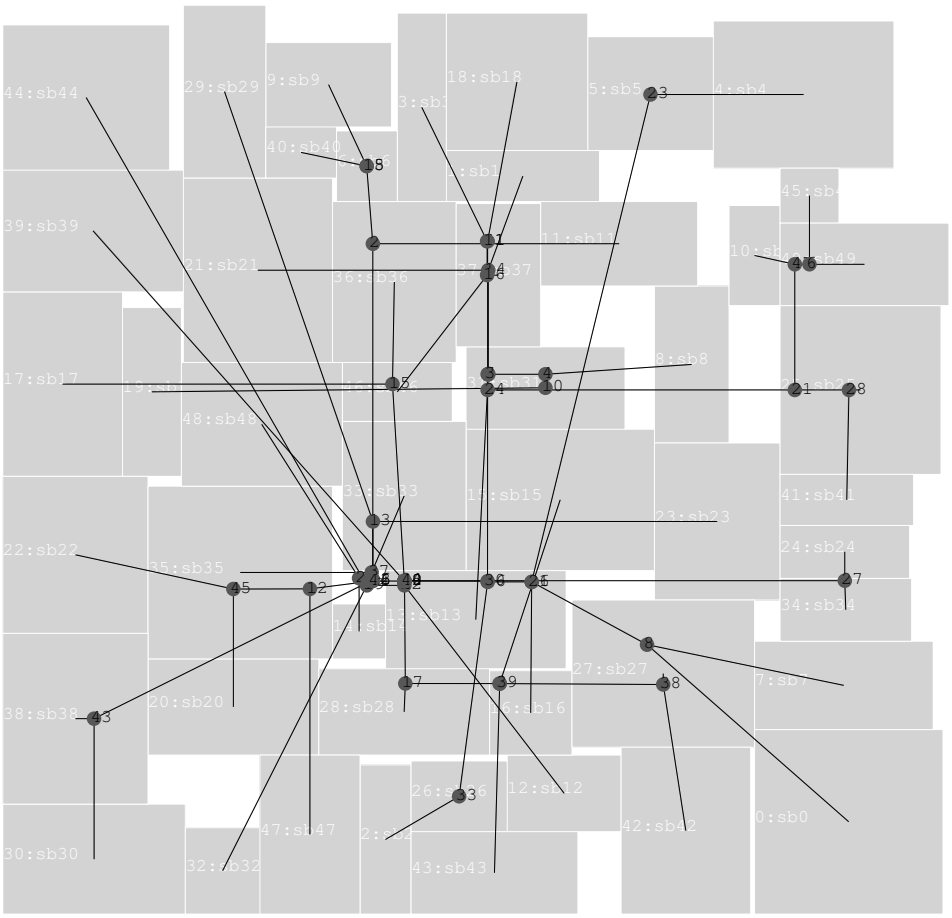

Fig. 6. A floor plan, topology, and router placement for a 50-core SoC.

in improved performance of the multi-frequency networks over the single frequency designs. In particular, note that this configuration only achieves about a 9/16 ratio of central node cycle time to leaf node, rather than the target 7/16 ratio, which would improve the efficiency of the network.

4.2 Application-specific SoCs

While uniform traffic allows a certain academic evaluation of a network, and provides a useful example for illustrating the potentially useful property of link-unique cycle-time, many SoC designs have very irregular traffic patterns. Consider the case of an MPEG 4 decoder, where in the design stage the required bandwidth between each core is known [42]. Much work has been dedicated to constructing an optimized interconnect for this type of SoC [24,16,34]. These methods generate a customized irregular topology and explore the space of router buffer sizes, link widths, and other parameters, resulting in significant power and performance improvements over a regularly-structured NoC design. With smaller process technology, the there is evidence that more numerous and lower-radix routers yield a better solution than larger, more power-hungry routers [30].

Similarly, an asynchronous network fabric also may be optimized for a given SoC. If the routers are sufficiently lightweight, and soft-IP cores are used, the routers

may be located nearly anywhere on on the die floor plan. We use this flexibility in our own network design to move routers such that highly trafficked paths are reduced in wire length and router hop count which will improve power consumption and latency. Our method utilizes a force-directed approach to move routers in ways that reduce path length, and simulated-annealing to explore variants of a tree topology. Figure 6 shows a floor plan of a 50 core design and its asynchronous network topology. Note how some routers are clustered together. This illustrates an advantage of an asynchronous network, where delay of a packet through the routers in a "cluster" is less than that of a similar clocked network. Each hop does not correspond to an entire clock cycle, but can be much faster due to the routers' proximity to each other. In essence, a "cluster" can be thought of as a single large-radix router, but where all the switching logic doesn't need to be active for the majority of traffic (which will only flow through a few of the smaller routers). This may offer an important power benefit, but is in need of further research.

In short, simple and more numerous asynchronous routers offer interesting flexibility and approaches to the current situation where *transistors are fast, wires are slow.*

5 Conclusion

This paper presents a view of what circuit design should be like in the year 2020. GALS design and handshake clocking will become a major commercial design methodology primarily for power, performance and design modularity advantages. This will occur despite the significant technical obstacles. The paper discusses how scaling began to favor clocked design methodologies in the 1980's, and how it now favors asynchronous design methodologies. It compares the overhead and cost of asynchronous design. Significant changes are predicted, including pervasive formal verification methodologies into design flows, that asynchronous design will in fact reach 40% of designs by 2020, and that several technologies will be instrumental to this conversion including FIFO buffering and data validity being integrated into the clocked cores. Delay insensitive methodologies will become nearly nonexistent by 2020 as CAD tools will begin to support bundled data methodologies, and GALS design will no longer use stoppable clocking. Education will be required to build more modular designs. Finally, the first asynchronous millionaires will appear based on successful startups.

References

[1] Achronix Corp.
URL http://www.achronix.com

[2] Amagbegnon, T. P., L. Besnard and P. L. Guernic, *Implementation of the data-flow synchronous language* SIGNAL, in: *Conference on Programming Language Design and Implementation*, ACM Press, 1995.

[3] Bainbridge, J. and S. Furber, *Chain: A Delay-Insensitive Chip Area Interconnect*, IEEE Micro **22** (2002), pp. 16–23.

[4] Campobello, G., M. Castano, C. Ciofi and D. Mangano, *GALS Networks on Chip: A New Solution for Asynchronous Delay-Insensitive Links*, in: *Design, Automation and Test in Europe*, 2006, pp. 160–165.

[5] Coates, W. S., J. K. Lexau, I. W. Jones, S. M. Fairbanks and I. E. Sutherland, *FLEETzero: An Asynchronous Switching Experiment*, in: *7th International Symposium on Asynchronous Circuits and Systems*, 2001, pp. 173–182.

[6] Cortadella, J., M. Kishinevsky and B. Grundmann, *Synthesis of synchronous elastic architectures*, in: *Proceedings of the Digital Automation Conference (DAC06)*, IEEE, 2006, pp. 657–662.

[7] Dobkin, R., R. Ginosar and C. P. Sotiriou, *Data Synchronization Issues in GALS SoCs*, in: *International Symposium on Asynchronous Circuits and Systems*, 2004, pp. 170–179.

[8] Elastix Corp.
URL http://www.elastix-corp.com

[9] Fernandez, J.-C., *An implementation of an efficient algorithm for bisimulation equivalence*, Science of Computer Programming **13** (1990), pp. 219 – 236.

[10] Fulcrum Microelectronics Inc.
URL http://www.fulcrummicro.com

[11] Guz, Z., I. Walter, E. Bolotin, I. Cidon, R. Ginosar and A. Kolodny, *Efficient link capacity and QoS design for network-on-chip*, in: *Proceedings of the Conference on Design, Automation and Test in Europe*, 2006, pp. 9–14.

[12] Handshake Solutions.
URL http://www.handshakesolutions.com

[13] Hoskote, Y., S. Vangal, A. Singh, N. Borkar and S. Borkar, *A 5-ghz mesh interconnect for a teraflops processor*, IEEE Micro **27** (2007), pp. 51–61.

[14] Huffman, D. A., *The Design and Use of Hazard-free Switching Networks*, Journal of the Association for Computing Machinery **4** (1957), pp. 47–62.

[15] *IEEE standard for binary floating-point arithmetic*, Technical report (1985).
URL http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=30711

[16] Jalabert, A., S. Murali, L. Benini and G. D. Micheli, *xpipesCompiler: A tool for instantiating application specific Networks on Chip*, in: *Conference on Design, Automation and Test in Europe (DATE)*, Paris, France, 2004.

[17] Kinniment, D. J., C. E. Dike, K. Heron, G. Russell and A. V. Yakovlev, *Measuring deep metastability and its effect on synchronizer performance*, IEEE Transactions on Very Large Scale Integration (VLSI) Syststems **15** (2007), pp. 1028–1039.

[18] Koch-Hofer, C., M. Renaudin, Y. Thonnart and P. Vivet, *ASC, a SystemC Extension for Modeling Asynchronous Systems, and Its Application to an Asynchronous NoC*, in: *NOCS '07: Proceedings of the First International Symposium on Networks-on-Chip*, 2007, pp. 295–306.

[19] Krstić, M., E. Grass, F. K. Gürkaynak and P. Vivet, *Globally asynchronous, locally synchronous circuits: Overview and outlook*, IEEE Design and Test of Computers **24** (2007), pp. 430–441.

[20] MacWilliams, P. D., W. S. Wu, D. K. Sampath and B. A. Prasad, *Method and apparatus for transferring data in source-synchronous protocol and transferring signals in common clock protocol in multiple agent processing system* (2002), US Patent No. 6336159, Assignee is Intel Corporation.

[21] Milner, R., "Communication and Concurrency," Computer Science, Prentice Hall International, London, 1989.

[22] Moore, G. E., *Cramming more components onto integrated circuits*, Electronics **38** (1965), classic scaling theory paper.

[23] Muller, D. E. and W. S. Bartky, *A theory of asynchronous circuits*, in: *Proceedings of an International Symposium on the Theory of Switching* (1959), pp. 204–243.

[24] Murali, S., P. Meloni, F. Angiolini, D. Atienza, S. Carta, L. Benini, G. de Micheli and L. Raffo, *Designing application-specific networks on chips with floorplan information*, in: *International Conference on Computer-Aided Design (ICCAD)*, 2006, pp. 355–362.

[25] Nanochronous Logic, Inc.
URL http://www.nanochronous.com

[26] *Open core protocol*.
URL http://www.ocpip.org

[27] O'Leary, J., X. Zhao, R. Gerth and C.-J. H. Seger, *Formally Verifying IEEE Compliance of Floating-Point Hardware*, Intel Technology Journal (1999).

[28] Paige, R. and R. Tarjan, *Three partition refinement algorithms*, SIAM Journal of Computation **16** (1987), pp. 973–989.

[29] Panades, I. M., F. Clermidy, P. Vivet and A. Greiner, *Physical implementation of the dspin network-on-chip in the faust architecture*, in: *The International Symposium on Network-on-Chip*, 2008, pp. 139–148.

[30] Pinto, A., L. P. Carloni and A. L. S. Vincentelli, *A methodology for constraint-driven synthesis of on-chip communications*, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (2008).

[31] Semiconductor Industry Association, "The International Technology Roadmap for Semiconductors," 2005 edition edition (2005), http://www.itrs.net/links/2005itrs/design2005.pdf.

[32] Seshia, S. A., R. E. Bryant and K. S. Stevens, *Modeling and verifying circuits using generalized relative timing*, in: *11th International Symposium on Asynchronous Circuits and Systems*, 2005, pp. 98–108.

[33] Silistix, Inc.
URL http://www.silistix.com

[34] Srinivasan, K., K. S. Chatha and G. Konjevod, *Linear-programming-based techniques for synthesis of network-on-chip architectures*, IEEE Trans. Very Large Scale Integr. Syst. **14** (2006), pp. 407–420.

[35] Stevens, K., S. Rotem, R. Ginosar, P. Beerel, C. Myers, K. Yun, R. Kol, C. Dike and M. Roncken, *An Asynchronous Instruction Length Decoder*, IEEE Journal of Solid State Circuits **36** (2001), pp. 217–228.

[36] Stevens, K. S., "Practical Verification and Synthesis of Low Latency Asynchronous Systems," Ph.D. thesis, University of Calgary, Calgary, Alberta, Canada (1994).

[37] Stevens, K. S., *Energy and performance models for clocked and asynchronous communication*, in: *International Symposium on Asynchronous Circuits and Systems*, IEEE, 2003, pp. 56–66.

[38] Stevens, K. S., R. Ginosar and S. Rotem, *Relative Timing*, IEEE Transactions on Very Large Scale Integration (VLSI) Systems **1** (2003), pp. 129–140.

[39] Stevens, K. S., Y. Xu and V. Vij, *Characterization of Asynchronous Templates for Integration into Clocked CAD Flows*, in: *To be published in International Symposium on Asynchronous Circuits and Systems*, IEEE, 2009.

[40] Talpin, J. P., P. L. Guernic, S. Shukla and R. Gupta, *A compositional behavioral modeling framework for embedded systems design and conformance checking*, International Journal of Parallel Programming (IJPP) **33** (2005), pp. 613–643.

[41] Theseus Logic, now merged with Camgian Networks.
URL http://www.camgian.com/2007_06_01.html

[42] Tol, E. B. V. D. and E. G. T. Jaspers, *Mapping of mpeg-4 decoding on a flexible architecture platform*, in: *Media Processors 2002*, 2002, pp. 1–13.

[43] van Berkel, K., R. Burgess, J. L. W. Kessels, A. Peeters, M. Roncken and F. Schalij, *A Fully Asynchronous Low-Power Error Corrector for the DCC Player*, IEEE Journal of Solid-State Circuits **29** (1994), pp. 1429–1439.

[44] You, J., Y. Xu, H. Han and K. S. Stevens, *Performance Evaluation of Elastic GALS Interfaces and Network Fabric*, Electronic Notes in Theoretical Computer Science **200** (2008), pp. 17–32, elsevier.