Design of an Energy-Efficient Asynchronous NoC and Its Optimization Tools for Heterogeneous SoCs

Daniel Gebhardt, *Student Member, IEEE,* Junbok You, and Kenneth S. Stevens, *Senior Member, IEEE*

***Abstract*—The energy usage of on-chip interconnects is a concern for many system-on-chips targeting portable battery-powered devices. We have designed and evaluated a network-on-chip (NoC) for such an application, including tools to optimize for power and communication latency. Our asynchronous (clockless) network operates with efficient two-phase bundled-data links and four-phase routers. The topology and router floorplan is determined by our tool, ANetGen, which optimizes the network for energy and latency using simulated annealing and force-directed placement methods. We compare our solutions against a traditional synchronous NoC as specified by the COSI-2.0 framework and ORION 2.0 router and wire energy models. Traffic is simulated with SystemC functional models, and messages are generated with a "bursty" self-similar *b*-model. Results indicate our asynchronous network was more energy-efficient, lower in area, and provided comparable or superior message latency.**

***Index Terms*—Application-specific, asynchronous design, embedded, GALS, low-power, network-on-chip, system-on-chip.**

I. Introduction

There are few published methods to aid in automating high-level asynchronous network-on-chip (NoC) design for fixed-function system-on-a-chips (SoCs). This paper expands upon our initial work that demonstrated a significant power, area, and latency advantage for an asynchronous network compared to a synchronous network [1]. We now add further important information about the router design and the computer-aided design (CAD) methods used, more operational details of the ANetGen optimization tool, and revised and additional results. Our use of a self-similar traffic generator and the study of the latency effects of bursty traffic is a key value of this NoC research. Unfortunately these types of methods are not more commonly used despite evidence that they should be [2]–[5].

Embedded, energy-constrained SoC designs can be roughly separated into two classes: platform-based and fixed-function (also called application-specific). The former is concerned with being able to perform a wide variety of tasks, many of which cannot be foreseen at design time. The latter is targeted toward a particular function, or a few functions, that have known properties. A fixed-function design might consist of a number of highly specialized cores and memories, and fewer general-purpose processors. The NoC of both these classes should be optimized for minimum energy usage while meeting the predicted performance requirements. However, the application-specific NoC may be more specialized as it has *a priori* knowledge of the communication patterns between cores. This is in contrast to general-purpose interconnects that are often evaluated with traffic patterns such as spatially-uniform, bit-transpose, hotspot, and others. The domain of this paper is the fixed-function, rather than the platform-based SoC.

Some globally-asynchronous locally-synchronous (GALS) interconnect solutions rely on a clock, either with standard synchronous clock distribution, or mesochronous clocks and synchronizers, such as *source-synchronous* methods [6]. However, an asynchronous (async) network has a number of potential advantages over a clocked network in a GALS environment. Standard arguments for asynchronous circuit design include robustness to process/voltage/temperature variation, average-case instead of worst-case performance, and other such points. However, there are also many NoC-specific arguments. In a synchronous NoC, the clock tree for all routers and pipeline buffers can consume significant power as shown in a heterogeneous network [7], and in a large chip multi-processor (CMP) 33% of router power [8]. Many SoC designs have quite bursty and "reactive" traffic. In this case, asynchronous methods are beneficial in that they consume little dynamic power during periods of low traffic without relying on clock gating techniques.

Available bandwidth on each asynchronous link can be independently set, to some extent, by wirelength between routers, link pipeline depth, or by varying the physical wire properties (metal layer, width, and spacing). This is potentially useful when bandwidth requirements on core-to-core paths vary considerably. In contrast, clocked networks commonly use a single frequency for all routers which is wasteful to

Manuscript received July 2, 2010; revised December 21, 2010; accepted March 28, 2011. Date of current version August 19, 2011. This work was supported by the National Science Foundation, under Grant CCF-0702539, and Semiconductor Research Corporation, under Task 1817.001. This paper was recommended by Associate Editor L. P. Carloni.

D. Gebhardt is with the School of Computing, University of Utah, Salt Lake City, UT 84111 USA (e-mail: dan.gebhardt@gmail.com).

J. You is with Texas Instruments, Inc., Dallas, TX 75243 USA (e-mail: j-you@ti.com).

K. S. Stevens is with the Department of Electrical and Computer Engineering, University of Utah, Salt Lake City, UT 84111 USA (e-mail: kstevens@ece.utah.edu).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TCAD.2011.2149870

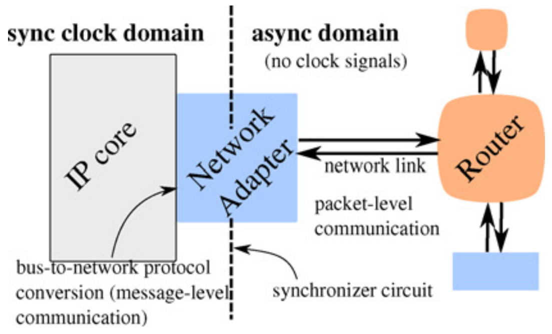


Fig. 1. Component diagram of an asynchronous NoC.

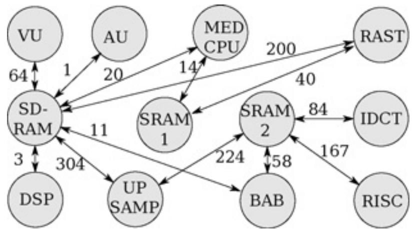


Fig. 2. Example CTG graph based on a MPEG4 SoC. Edge weights are in MBytes/s.

those paths not requiring high bandwidth. A clocked NoC can use discrete "islands" of differing clock speeds to achieve a similar effect, but in a much coarser-grained fashion. Source-synchronous networks do not require a global clock trees and thus yield some benefit of the asynchronous style, as well as easier compatibility with existing design tools. However, they require more complexity to adjust clock rate based on varying sender or receiver speeds, or to take advantage of particular link wire lengths.

Fig. 1 shows the functional components of an async NoC, and how it interfaces with the cores of an SoC. A core has an interface using some type of standard protocol, such as OCP or AMBA AXI. A network adapter converts this protocol to the one used in the particular NoC design, thus implementing the transaction and transport communication layers. It also synchronizes between two timing domains; the intellectual property (IP) core's clock domain and the asynchronous NoC domain. This also enables each core to operate at its own frequency, as is done in GALS. The routers are entirely async and do not use a clock to communicate to each other, or to the network adapters.

Design automation techniques are commonly used to generate a NoC for a specific SoC design. These methods can decrease time of development in commercial products or allow a researcher to explore a larger design space. The NoC solution is chosen based on some metric, usually a function of energy and performance. In the optimization process, potential solutions must be evaluated for quality, and this often requires an abstracted model of the SoC characteristics.

This abstraction can be done at a variety of levels depending upon completeness or availability of the SoC design and NoC components. Ideally, one could simulate the exact functionality of the various cores composing the design, and the NoC would be fully implemented to model the communication. Unfortunately, this method is labor and simulation-time intensive, and not a good choice for early-exploration of the NoC design space. As usual, tradeoffs must be made as function becomes more abstracted. The following properties may be lost or made less precise: causal relationships between various senders and receivers, exact sizes of communication messages, exact temporal and spatial distribution of traffic, latency requirements of specific data segments, and effects of network congestion on core behavior.

A commonly used abstraction in the literature has been titled a communication trace graph (CTG) [9] or a core graph. A *path* describes pairs of source and destination cores, and the particular links and routers a packet traverses. The CTG has a n-tuple of values per path that often includes average expected traffic rate per path and sometimes a latency requirement of a packet. An example CTG based on a MPEG4 decoder is shown in Fig. 2, which we use in our evaluation.

This paper is organized as follows. Section II gives an overview of related work. Our asynchronous router and its design methodology is discussed in Section III. We then explain in Section IV our design-automation tool, ANetGen, and the related traffic generation and simulator tools. The network and tools used for comparison, and the evaluation methods are given in Section V. The results are discussed in Section VI, and we conclude in Section VII.

II. Related Work

Many others have focused on generating or optimizing on-chip interconnects. Regardless of the specific interconnect details, the problems are similar, in that searching the solution space is complex and usually requires heuristics or approximation methods. The COSI framework [10] generates an application-specific NoC and floorplan, taking as input such constraints as core areas and average bandwidth between cores. While it is extensible with new algorithms and components, it does not consider asynchronous network components and, as future work, cites the need for integrating traffic burstiness. For the Xpipes library, a heuristic search determines the topology and router configuration [11]. It uses floorplan information, router energy models, and core communication requirements. The results indicate a significantly reduced power and hop-count versus the best mesh topologies for a variety of SoC designs. It is part of a complete workflow to automatically synthesize a NoC down to chip layout [12]. A linear programming based method is presented in [9]. For the QNoC routers, application-specific optimization is discussed in [13], but it focuses on mapping logical resources of a mesh-style topology rather than physical concerns. A reconfigurable source-synchronous NoC and fabricated 801.11a receiver chip was presented in [6]. Bandwidths and circuit-switched paths spanning multiple routers can be customized at runtime for a particular application, and their design was optimized for low power.

Previous research on asynchronous interconnects is rich, but these designs are either hand-designed for a particular application, or general in design but possibly having over-provisioned resources for a power-constrained SoC. All but one of these existing routers use quasi delay-insensitive protocols between routers, rather than bundled-data. Fulcrum Microsystems created a large asynchronous crossbar to interconnect cores of a SoC [14]. The commercial startup Silistix, based on earlier academic research [15], sells EDA software and circuits that provide an customized asynchronous NoC, but has no published methods for the optimization process. The MANGO router [16] provides both best-effort and guaranteed-service traffic. FAUST [17] is a platform and fabricated chip used in 4G telephony development, and uses an asynchronous mesh-based NoC [18]. The QNoC group has developed an asynchronous router that provides multiple service levels and dynamically allocated virtual channels per level [19]. A mesh-of-trees network was constructed from simple bundled-data routers for a CMP [20] that, same as our work, uses individually controlled latches for buffering. This is similar to the clocked elastic buffer concept [21]. The insertion of link pipeline buffers in an async NoC was explored and compared against a similarly-designed synchronous “elastic” network [22]. Asynchronous link pipelining strategies were explored for application-specific SoCs in [23]. A comparison between the asynchronous network ANOC, and the mesochronous clocked network DSPIN, was performed in [7]. For both designs, a physical layout and functional traffic simulation was done for analysis. While DSPIN had 33% less area and 33% higher bandwidth than ANOC, ANOC had shorter packet latency and at least 37% lower power consumption. DSPIN was also compared against its asynchronous analog, ASPIN [24]. Average power, latency, and saturation threshold are superior in ASPIN with similar die area.

Traffic modeling for NoCs is one of the major outstanding problems in the field [2]. The *b*-model [25] provides a simple method to produce and analyze the burstiness of self-similar traffic with a single parameter. This is in contrast with other bursty generators, which tend to be complex, or rely on the non-self-similar Poisson distribution. The *b*-model has been adapted to study burstiness effects in the Nostrum NoC [26]. Evidence of traffic self-similarity and burstiness in MPEG-2 video applications has been shown [4], [5]. Several analytic models of network performance have been developed for NoC design. A model has been developed to capture spatial and temporal characteristics of traffic for regular, homogeneous NoCs [27]. A generalized analytic router model was developed in [28]. It provides detailed statistics during expected traffic, and is applicable to heterogeneous, irregular networks, but relies on the Poisson arrival process and a synchronously-clocked router.

III. Asynchronous Router Design

A. Overview

The design of the router and link pipeline circuits, and their assembly into a network, is based on simplicity and efficiency. This follows from the theory that a logic function using a simple circuit is more energy-efficient has lower latency than a complex implementation. The cost of a simple design is in sacrifice of performance and features, such as reduced throughput under traffic congestion, quality-of-service guarantees, traffic priority, and reconfigurability. The benefit, however, is a lower energy for a given traffic rate, and depending on the application, could be the more desired choice. If a Pareto front is considered with axes of functionality versus efficiency, this paper aims to fall near the high efficiency region, as shown in Fig. 3.

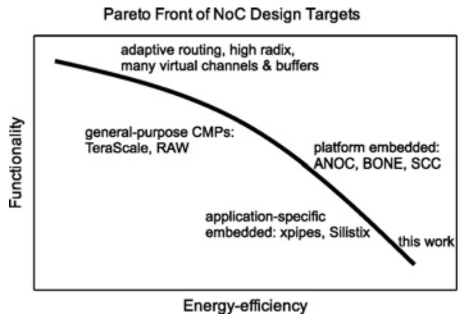


Fig. 3. Pareto front of NoC design styles, their applications, and example implementations.

This design has low area, latency and energy due to the following implementation choices. Individual latches are used for flit buffers instead of flip flops, making a more area-efficient storage structure. Large crossbars consume significant area and power, so simple 2-to-1 MUXs are used. Switch direction is determined by the most significant of the source-route bits, and this bit directly controls the MUXs. This avoids address decoding circuitry and allows simple rotation (only wiring) of the routing bits on the output packet. The packet format is a single flit containing data and routing bits containing source-routing bits in parallel on separate wires with the data bits. These are arranged with bi-directional channels, resulting in a three-ported (radix-3) router. Routers are connected in a tree topology for this paper, described in more detail in Section IV, as this requires the minimum number of routers and logarithmically low number of routing bit wires. However, other possible topologies include a ring or an irregular structure.

The number of required routing bits is determined by the maximum hop count of a network generated for a specific SoC design. The width of each flit must be determined based on required throughput or power and area constraints. This format has the overhead of requiring routing bits with every flit, but series of packets of the same source-to-destination path will not cause the route-bit wires to switch, saving power.

The number of required routing bits is determined by the maximum hop count through a network that is generated for a specific SoC design. The link’s data width is determined based on required throughput, power, or area constraints. The packet format has the overhead of separate routing bit wires on a link. However, it has a number of benefits. It avoids the downside of a multi-flit packet and wormhole switching that can cause blocking at many routers through the network, especially a topology such as this with low path diversity. An application-level transaction will often be composed of a number of packets sent from a source to destination IP block in quick succession. If the path through the network is not being shared with other transactions, this format can take advantage

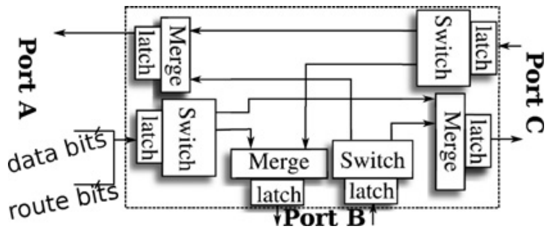


Fig. 4. Architecture of a 3-port asynchronous router.

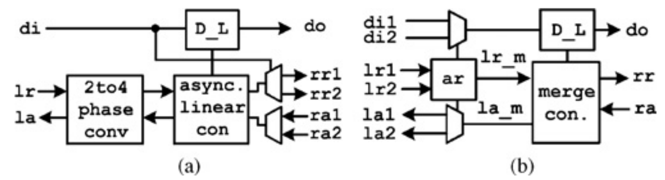


Fig. 5. Design of Switch and Merge modules. (a) Switch. (b) Merge.

of similarity between them, since the routing bits will not need to change state for each packet.

The router design and packet format does not include specific signals for a particular transaction-layer protocol, such as addresses or command types. The network interface at each core implements the transaction and transport layer protocols such that this information is packaged within the data bits of multiple packets. A unique network adapter is needed for a particular transaction protocol, such as OCP or a message-passing interface, the choice of which depends on the how the SoC and IP cores are implemented. The specifics of this packetization process are not implemented in this paper, but the end-to-end latency of a large *message* is used in the evaluation to capture transaction-level performance. The packet format is not fundamentally different than that of the synchronous network used in comparison, which is also fixed-length. Both networks, as a simple example, would need the first packet that begins a transaction to contain an identification code for operation type, such as memory burst-read.

The router is implemented with three components: a switch module, merge module, and a buffer. The switch module determines, based on the flit's route bits, which output port will latch the data available at the switch's input. The merge module arbitrates between two input channels to an output channel, granting access to the first-to-arrive request signal. This effectively alternates between the two input channels, assuming each provides the next packet within an output channel's cycle-time. A router is composed of three switch modules and three merge modules, as shown in Fig. 4. Each switch and merge module has one set of latches providing 1-flit buffers on each input and output port.

B. Router Circuit Design

Asynchronous protocols normally fall into two categories: quasi delay-insensitive (QDI) and bundled-data (BD). Generally, QDI is more robust to variations while BD allows simpler circuits. BD has a lower wire count compared to QDI's common encodings (e.g., 1-of-4 and dual-rail). This is potentially more energy-efficient due to fewer wire repeaters especially with wide links [29]. The choice of 4-phase or 2-phase protocol impacts performance and circuit complexity. The throughput across long links is limited by wire latency, thus a 2-phase protocol achieves almost twice the throughput as a 4-phase protocol (but is still half the bandwidth of a synchronous or source-synchronous link). However, a 4-phase, level-sensitive protocol typically allows more simple circuits. A more detailed description of these protocols and encodings is given in [30]. Most asynchronous NoCs use QDI for the link protocol, and the benefit of BD is worthy of exploration, which is done in this paper.

With these properties in mind, the router was designed to internally operate using a BD 4-phase protocol and BD 2-phase between routers. Within the router, 4-phase is more efficient. It requires less logic than 2-phase BD or QDI and works directly with the level-sensitive 4-phase MUTEX element [31] that arbitrates shared output channels. The BD 2-phase protocol is used on links because wire delay can be a limiting factor in bandwidth, and 2-phase requires half the time-of-flight wire delays as 4-phase.

The block diagrams for the router's switch and merge modules are shown in Fig. 5. The 2-to-4 phase converter (*2to4 phase conv*) adapts the link protocol to the router protocol. The switch does this through signals left request (lr) and left acknowledge (la), and similarly for the merge, right request (rr) and right acknowledge. The phase converter handshakes with a BD 4-phase burst-mode asynchronous controller (*async. linear con*) to pipeline the flit from the input port latches to the output port latches (*D_L*). The linear controller's specification and timing assumptions have been previously studied [32]. The *request* signal is directed to one of the two output channel's merge module (*rr1* or *rr2*) based on the most significant route bit. The route-bits are rotated and passed to the merge module. The routing operation occurs concurrently with the handshake back to the input channel.

The merge module is composed of the arbitration circuit (*ar*) and merge controller shown in Fig. 5(b). The arbiter serializes requests to a shared output channel from two input channels. If both input channels have a steady flow of packets, it alternates between them in a round-robin fashion. The output of the arbiter controls a MUX that selects the appropriate input flit to store in the output latch. The arbiter uses a 4-phase handshake signal to the merge controller via *lr_m* and *la_m*. The MUTEX element is described in other work [30] in more detail, passes only the first-to-arrive request signal to its associated output, until it is de-asserted. The merge controller interfaces with a network link with a 2-phase protocol via *rr* and *ra*, as well as control the output data latch, *D_L*.

A more detailed schematic of the switch and merge modules can be found in previous work [1]. All of the circuits were designed with the static, regular V_{th}, Artisan cell library on IBM's 65 nm 10 sf process except the MUTEX element in the merge module. The MUTEX was designed and characterized as a separate library cell through manual layout and HSPICE simulation. The power and latency of the switch and merge modules could have been significantly improved by using

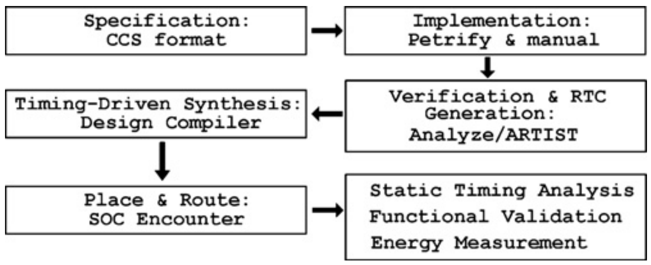

Specification: CCS format
Implementation: Petrify & manual
Verification & RTC Generation: Analyze/ARTIST
Timing-Driven Synthesis: Design Compiler
Place & Route: SOC Encounter
Static Timing Analysis
Functional Validation
Energy Measurement

Fig. 6. Asynchronous circuit design flow.

domino or custom dynamic gates, rather than a traditional static cell library. However, dynamic libraries are rare and so this normally would require custom cells or layout. A traditional static library implementation allows designers to use a commercial static gate cell library and CAD flows, potentially decreasing design time.

The methodology of this asynchronous circuit design used a CAD tool flow similar to [33], and is shown in Fig. 6.

Specification: The process logic of CCS [32], [34] was used to make the specification of circuit modules. The linear controller specification is shown in (1) and the merge controller in (2)

$$\begin{aligned} \text{LEFT} &= \text{lr.`cl.`al.`c2.lr.`la.} \quad \text{LEFT} \\ \text{RIGHT} &= \text{c1.`rr. c2.ra.`rr.ra.RIGHT} \\ \text{LC} &= (\text{LEFT}|\text{RIGHT})\backslash\{\text{c1, c2}\} \end{aligned} \tag{1}$$

$$\begin{aligned} \text{LEFT} &= \text{lr.'c1.'la.'c2.lr.'la.LEFT} \\ \text{RIGHT} &= \text{c1.'rr. c2.ra.} \quad \text{RIGHT} \\ \text{MG\_CON} &= (\text{LEFT}|\text{RIGHT})\backslash\{\text{c1, c2}\}. \end{aligned} \tag{2}$$

Implementation: Circuit implementation of async modules was done with the software tool *Petrify* [35], and manual design for the MUTEX and 2-to-4 phase converter circuits. The input to *Petrify* is a Petri net, equivalent to the process-based specification above. These are presented in Figs. 7 and 8. Relative timing constraints (RTC) are shown as dashed arrows; these enable better timing optimization for asynchronous circuits [32].

Verification and RTC generation: The implemented circuits were verified using the asynchronous formal verification tool, *Analyze* [36]. Another tool, *ARTIST* generated the RTCs that allow the circuit to be proven conformant to its specification, and thus operate correctly [37].

Timing-driven synthesis: The RTCs from *ARTIST* were converted into Synopsys design constraints format. With these, Synopsys design compiler was used to synthesize the async modules, and finally the full router.

Place and route: The synthesized async router was physically placed and routed with Cadence SoC Encounter.

Static timing analysis: The placed and routed designs were timing-verified by static timing analysis with Synopsys PrimeTime. This guaranteed the RT constraints generated by the verification tools were met.

Functional validation: Functionality and performance were validated in the design with ModelSim using back annotated pre-layout and post-layout delays.

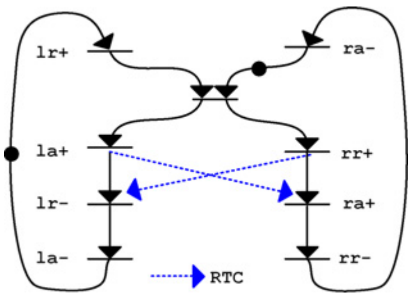


Fig. 7. Petri-net of linear controller.

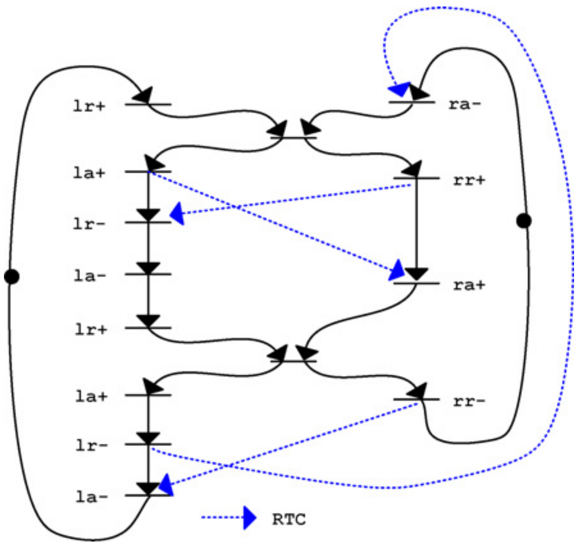


Fig. 8. Petri-net of merge controller.

Energy measurement: Energy was measured using HSPICE simulations of the design's SPICE netlist, including parasitic extraction from Mentor Graphics Calibre PEX.

C. Evaluation

We have constructed a number of routers with varying flit widths, but the system-level results in Section VI use parameterization of 32 bits of data and 8 bits for source-routing. The resulting area is 2828 μm^2, dynamic energy/flit is 1.03 pJ with 25% of bits switching per flit, and leakage power is 0.009 mW.

The area is dominated by the data latches and MUXs used in the merge modules. The controllers (linear controllers in switch modules and merge controllers in merge modules) make a very small contribution to the total area. Dynamic energy is consumed when one data word passes a router from an input port to an output port.

The maximum throughput of the router is 2.12 Gflits/s. This was measured by providing data into the input ports at maximum rate and allowing the output port to communicate with another router with no wire delay. The backward latency of a router is the delay from a request on an incoming channel to the acknowledgment on that channel, completing the handshake of the two-phase protocol. Fast backward latency is desirable because it frees the previous router's output port for another transaction. Forward latency of a router is the delay from a request on an incoming channel of a router to the associated request on an output channel, assuming no contention or stalling in the arbitration circuit. This is determined by the delay to buffer the data, arbitrate control, and switch to the outbound channel. This router design has a 250 ps backward latency and 460 ps forward latency.

Our router's low power and area are due to its simple architecture and the use of latches, rather than flip-flops, for the storage elements. Since much of the area and power of router architectures is from memory elements, this advantage makes a significant difference. Furthermore, the simplicity of the control circuits also contributes to high throughput and low energy. The router employs a bundled data protocol rather than delay insensitive codes which results in fewer wires per channel and efficient use of standard cell libraries. The cost of this is that the circuit timing must be carefully specified and controlled, similar to clocked design, to ensure correct operation. System-level power and performance results using this router in a network are shown in Section VI.

IV. Asynchronous Network Generation

We built a tool, *ANetGen*, that determines the logical topology of routers in the network and where they should be physically placed on the chip's floorplan. ANetGen takes an input format that defines the CTG edges and expected traffic bandwidth, as well as the core dimensions. The core floorplan is specified prior to ANetGen with a separate tool, in this case Parquet [38], that also uses communication bandwidth between cores to optimize for wirelength as well as area. ANetGen's objective function is to minimize wirelength and hop counts for high traffic paths. It does this with a combination of simulated annealing (SA) and force-directed movement techniques.

A. Topology and Placement

Asynchronous circuits have unique properties that can be leveraged to optimize the network. Specifically, the physical path length between endpoints directly affects packet latency, not just the number of routers and pipeline buffers a packet must travel through, assuming an uncongested path. This is in contrast to a synchronous system, where each network element constitutes at least one required clock cycle.

For example, consider three closely-placed routers. If they are synchronous, then a flit's latency will be at least six cycles (one for each router and wire segment). However, if they are asynchronous, the delays of the wires are minimal and thus only three hops of logic delay are required.

Link energy usage can be significant and grows, relatively, with shrinking process technology [39]. With this in mind, the physical placement of routers needs to be determined such that wirelength is minimized, especially on highly trafficked paths. For these experiments, we assume hard IP blocks which have fixed dimensions and network adapter ports. This is in contrast to earlier work [1] which assumed soft-IP. In an actual design flow, the router placement our tool generates will provide input to the hierarchical placer, or floorplacer [40] that will legalize the placement of cells and macros composing each core.

The problem of finding the optimal tree topology is similar to the NP-hard quadratic assignment problem of mapping cores to a mesh topology [41]. For this, we utilized a simulated annealing framework: the ParadisEO C++ Library [42]. In ANetGen's current implementation, it limits topology exploration to forms of a tree, which has a minimal number of three-port routers and deadlock-free routing. This is mostly to explore its potential in an asynchronous network when considering physical design elements. The range of tree-type topologies offers room for exploration by the tool; it can be more balanced or more "linear," and core mapping within a given topology can change. We save a detailed analysis and comparison with other topologies to future work, but this method produces good results, as seen in Section VI.

The ANetGen operation, at a high level, is described as follows, and is shown in Fig. 9. The input, the cores' physical information and communication properties, is read by the tool. It generates an initial topology of a balanced tree and places the routers. Iteratively, the topology is perturbed by swapping the links of two routers or cores topologically near each other. The quality, or "fitness" of the new solution is, at this point, titled partial fitness as it does not take into account the physical placement of the routers. If the partial fitness improves, the routers are placed on the floorplan, and a full fitness is calculated and recorded. After a number of iterations, the system is determined to be "cooled" and the best solution (usually the solution at the end) is saved.

The fitness of a solution is based on two parts: power and router contention probability. Power is estimated from the flit transfer rate over each router, using our circuit energy data, and over each link, using the Orion 2.0 wire models (part of a synchronous router model package) [43]. Router contention is a unitless, relative value calculated by the following formula:

$$Contn. = \sum_{path\ p} \left((HOPS_p)^{k1} \times (BW_p + CRIT_p) \right)$$

where $k1$ is an exponent (currently 1.5) giving geometrically worse (higher) fitness to solutions with many-hop paths. CRIT is a user-supplied measure of path criticality that weights the impact a path has on fitness independent of bandwidth.

When a full fitness must be calculated, the router locations need to be determined for accurate wirelength, and thus energy information. We used a method extended from [44] that uses force-directed movement to provide router locations and link lengths to the SA process. Force vectors are applied to routers that are proportional to: 1) bandwidth requirements; 2) wirelengths of connected links; and 3) total path wirelength from a sender, through the routers, to receiver. The purpose of 1 and 3 is to more heavily favor forces that decrease total distances on high-bandwidth paths, while 2 spaces routers for higher throughput by reducing long-links. The process is iterative, where a router moves a distance proportional to the sum of its force vectors. After some time, movement halts and the placement converges. Because we assume hard IP blocks, we then perform a "de-overlap" of the routers to remove them from IP-block bounds. In brief, the algorithm is as follows. The routers are ordered by bandwidth through each. In order, they are moved to the nearest core edge and not allowed to

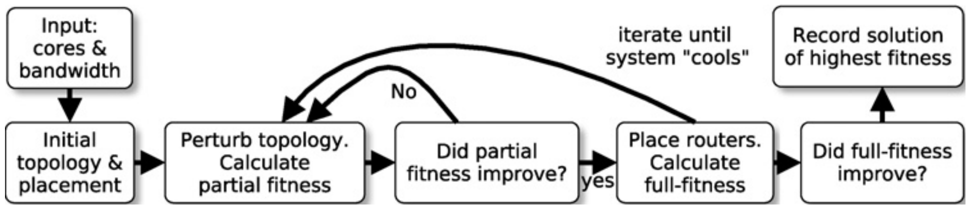

Input: cores & bandwidth
Initial topology & placement
Perturb topology. Calculate partial fitness
Did partial fitness improve?
No
yes
iterate until system "cools"
Place routers. Calculate full-fitness
Did full-fitness improve?
Record solution of highest fitness

Fig. 9. ANetGen operation diagram.

enter any core, and the placement occurs again. This repeats for each router remaining in a core.

B. Self-Similar Traffic Generator

In order to explore the performance characteristics of the network as realistically as possible, we moved away from the commonly used Poisson traffic models and decided instead to use a self-similar model. Self-similar traffic has been seen in NoCs and is a suggested model for research [4], [5]. We implemented the self-similar, *b*-model traffic generator [25], suggested as a key feature in future NoC benchmark sets [3]. The traffic generator outputs chunks of data, or *messages*, that simulate the requests from software to the transaction layer of SoC communication. The self-similar behavior is seen in the times these messages are generated, not necessarily the times the individual packets enter the network (which is dependent upon network bandwidth, congestion, and similar properties). A message is broken down into contiguous packets to be sent over the network as fast as it will allow. Message generation cannot be stalled by the network, but message latency through the network to a destination IP core is delayed by the bandwidth of the network and transient effects of packet contention with other paths. This operation flow and structural partitioning is shown in Fig. 10. The self-similarity is not dependent upon details of how a message is formed into packets, nor the link-level protocol, and can be used with various network implementations. This design has a SystemC transaction level model (TLM) for its interface, and thus it is portable and relatively easy to integrate with other network models, as we did with COSI (described below).

Our model is parameterizable with the following inputs:

1) source and destination cores;
2) a *b*-value in the range [0.5, 1.0) indicating burstiness;
3) simulation duration;
4) average bandwidth, i.e., desired total traffic volume;
5) the smallest time-resolution of the burstiness;
6) message size, e.g., 256 bytes.

Self-similar traffic is generated recursively with an algorithm closely following the original [25]. A known volume of traffic to be sent during a known simulation duration is divided into two parts, each part weighted by the bias, *b*. Each of these is then split, and the process continues for each sub-division of time, until the desired time resolution is reached. Steps of the this process are shown in Fig. 11. There are, however, a number of interesting details to note. The b-model determines the total volume of data to send in each *window* determined by the specified time resolution. Within a window, a message

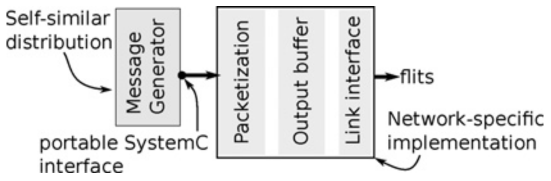

Self-similar distribution
Message Generator
portable SystemC interface
Packetization
Output buffer
Link interface
flits
Network-specific implementation

Fig. 10. Structural partitioning of traffic generation.

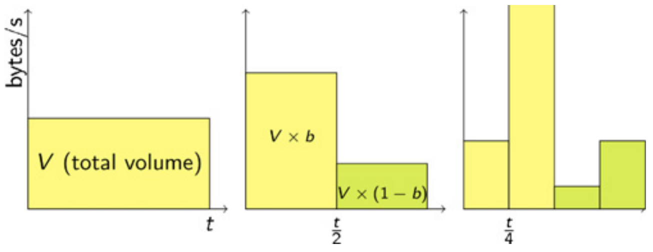


Fig. 11. Generation of a self-similar traffic volume distribution over a simulation's duration.

is probabilistically sent each cycle such that over the time window the proper amount of data is generated. Greater *b*-values for paths that share network resources lead to worse congestion, and thus message latency, if those paths have overlapping "high-volume" periods.

It may be the case that the desired volume of traffic per window exceeds the capacity of the link or output buffer, or the previous window has not finished sending its data yet. In these cases the packets are queued up in an "infinite" buffer. Therefore, the self-similar model's output is the ideal desired data transmissions, but the actual packet-level data distribution is subject to network limitations as is expected.

C. Simulator

Our requirements for asynchronous NoC simulation include modeling details at both a high-level and a low-level. At a high-level, it must allow easy design space exploration such as buffer sizes, message lengths, router radix, and provide statistical reporting in human-readable format. At a low-level, it must model wire delays between routers, logic delays within routers, energy consumption of links and routers, and be relatively easy to automate network setup, simulator execution, and results analysis. Additionally, the ability to substitute an RTL router model for a functional C++ model is important for a more accurate post-circuit-design energy and latency evaluation.

With these considerations in mind we built an asynchronous network simulator using the SystemC library. The following modules were developed: an arbiter, an *inport* to the router, an *outport* from the router, and input and output port FIFOs, and a self-similar traffic generator. The SystemC TLM library is used for inter-router links and traffic generation. We chose this method to allow easier extensibility of the channels if needed, and TLM provides a convenient way to model link and protocol delays.

The traffic generator and router ports use a `simple socket` to receive a `generic payload` transaction object that contains packet and routing information. When a TLM object is received by the *inport*'s socket, a `wait` is performed to model the wire delay. This delay is calculated from an interpolation of HSPICE simulations of various wirelengths in IBM's 65 nm technology. The wire energy per transfer is calculated using the Orion 2.0 model. The router waits an additional time period to model forward logic delay. The flit is written to the FIFO, which triggers the arbiter. Another `wait` models the acknowledgment delay to the sender. It should be noted that these channel delays are specific to the asynchronous protocol used, e.g., two-phase or four-phase, and are easily changed.

Within the arbiter, a `doSwitching` SC_METHOD is called whenever a packet is received by an input FIFO or acknowledged by an output FIFO. The arbitration mechanism is that described in Section III. At each switching operation, the appropriate energy is logged. This energy was measured from transistor-level router simulations.

Each *outport* operates in its own thread, waiting for a packet to be passed to it by the arbiter, or for a TLM response indicating that the channel is free. When there is data in the FIFO and the channel is free, it sends a new TLM `generic payload`. The *outport* also records wire energy of the transmitting link.

V. Network Comparison Methodology

The evaluation of all network instances was done with the SystemC simulators uniquely generated for each. In this section, we present the simulation parameters and benchmarks used in the evaluation.

A. Synchronous Network Generation

The baseline network used for comparison purposes is generated by a research tool called COSI 2.0, a source-code release that incorporates much of the functionality of COSI-NoC (v.1.2) [10]. Similar to ANetGen, COSI takes as input a SoC design abstraction consisting of core dimensions or area, and a set of communication constraints between those cores, which are called *flows*. This is a more generalized concept than the CTG mentioned in Section I, and COSI can consider temporal properties between flows, such as mutual exclusion.

Given these flows, its optimization algorithms try to find the network and floorplan that meets the constraints while minimizing power, based on router and wire models. As output, COSI produces a floorplan, topology, and a SystemC-based simulator. Included with the software release are algorithms for generating a mesh and a min-cut partitioning method (hierarchical star) similar to that of [11]. We modified COSI to incorporate the Orion 2.0 router and wire models, and also made a number of other changes to COSI to improve its operation and result reporting. The SystemC simulator produced by COSI was modified to use our bursty traffic generator previously described.

TABLE I
Average Bandwidths for the ADSTB Design

Sender	Receiver	MBytes/s	Sender	Receiver	MBytes/s
CPU	AudioDec	1	CPU	DDR	3
CPU	Demux	1	CPU	MPEG2	1
DDR	CPU	3	DDR	HDTVEnc	314
DDR	MPEG2	593	Dem1	Demux	31
Dem2	Demux	31	Demux	AudioDec	5
Demux	MPEG2	7	HDTVEnc	DDR	148
MPEG2	DDR	424			

Both the COSI and ANetGen SystemC models do not consider details of a transaction-layer implementation at the network adapters. The COSI routers, same as the async routers, have a fixed packet size with no specific bits allocated for fields such as address, burst size, or interrupt. In both models we assume the network adapter and transaction-level protocol use the data field of a packet to transfer this control information and serialize data over multiple packets if necessary. The end-to-end protocol and network adapter may use, for example, *request* and *grant* packets to establish a cache-line-sized memory transfer in an atomic fashion. Our evaluation methodology does not include the network adapter and protocol overhead, but we assume it to be similar for both the synchronous and asynchronous networks if transactions are greater than a COSI-sized packet (or are small enough to fit in the async packet). We have changed the COSI models to use an 8-bit field in the first flit, instead of a full 32-bit flit, for routing a packet and the rest of the packet designated the *data* field. This reduces its routing overhead to be comparable to the asynchronous design. A comparison between asynchronous and synchronous networks with identical packet formats and similar circuit properties is shown in [22].

B. SoC Designs and NoC Generation

We used two SoC design abstractions of the CTG format described in Section I for our evaluations. One is titled ADSTB and is from the public COSI 2.0 distribution. The other is an MPEG4 decoder originally described by [45] and used in several other NoC research projects. Bandwidth requirements were modified from those originally provided, and are shown in Fig. 2 for MPEG4 and Table I for ADSTB. The burstiness parameter, as it is changed to different values, is set to the same for all paths. The die areas after router placement for the ADSTB and MPEG4 designs were 35.7 mm^2 and 78.7 mm^2, respectively.

We generated a network for each design using the COSI (sync.) and ANetGen tools. We configured COSI to generate a hierarchical star network with $N/3 + 1$ partitions (N is number of cores), chosen based on empirical experimentation for low energy. The floorplanner was constrained to a square aspect ratio outline. The resulting floorplan was also used for ANetGen. The ADSTB topologies generated by COSI and ANetGen are shown in Fig. 12. A floorplan of the MPEG4 design is shown in Fig. 13. We assume, due to the diminutive

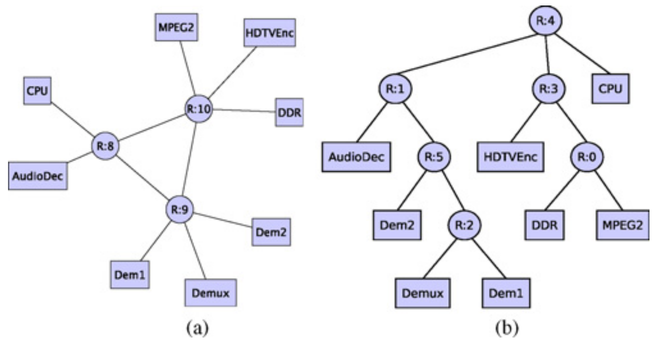


Fig. 12. Topologies of the ADSTB design. (a) COSI-generated. (b) ANetGen-generated.

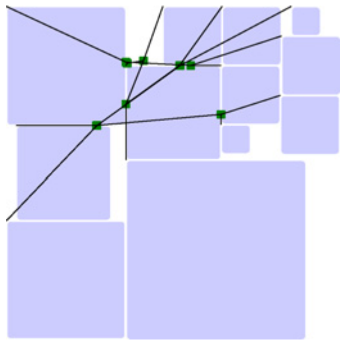


Fig. 13. MPEG4 floorplan. Router locations determined by ANetGen.

size of the routers, that they may be placed between cores, and the core location adjusted slightly to accommodate. Routers are small, dark blocks. Connectivity between a core and router or two routers is shown with a line. Actual wire routing is, of course, Manhattan, as is link length calculation.

C. Simulation Parameters

We instrumented the SystemC router and wire models from COSI and ANetGen to record energy usage, packet latency, and message latency over the course of a simulation. Orion 2.0 is used for the wire energy model in both frameworks, and also for the synchronous router leakage power and switching energy models. Leakage power of routers and wires was calculated using parameters from the IBM process used in the async circuit evaluation rather than the default assumed by Orion. Energy of the async routers comes from circuit simulation described in Section III. We set the router and wire models to assume that 25% of a flit's data bits switch per transfer. For the async models, the actual route bit values were used, thus accurately capturing the benefit of fewer state-changes of these wires due to series of packets with the same source and destination. Additionally, we modified the Orion link model to more accurately estimate the size of the repeater (inverter) needed to drive the initial segment of wire before a larger repeater is used.

We chose parameters for the Orion router model to be comparable to our asynchronous configuration, given the set of parameters available. These are shown in Table III. Clock tree power estimation was excluded from these models. A 32-bit flit width provided adequate bandwidth while keeping power and area low, especially in the synchronous network

TABLE II
MESSAGE LATENCY AND DYNAMIC POWER FOR MPEG4 UNDER VARIOUS PACKET SIZES AT $b = 0.8$

	Packet Size		
	2-flit	4-flit	8-flit
Mean of path medians (ns)	231	223	219
Mean of path maximums (ns)	475	472	485
Dynamic power (mW)	31.4	29.7	28.1

TABLE III
ORION 2.0 MODEL PARAMETERS

Router freq.	2 GHz	Router I/O buff's	2/1 flit
Tech. library	65 nm NVT	Crossbar	Multitree
Voltage	1.0 v	Flit width	32 bits

using the ORION models. The packet size of four-flits was empirically chosen over two-flit and eight-flit packets based on the maximum latency of messages over paths. Table II shows the effect of packet size on message latency, where the values are the mean of all path-specific median and maximum latencies. Message latency, and path-specific latency are defined and presented in Section VI. The power is the dynamic router and wire power, which reduces as packet size increases due to less routing overhead. A router's input buffer size of four-flits, enough for a full packet, was considered, but it greatly increased energy while not showing large improvements to message latency. We chose two-flit buffers as a compromise between energy and latency based on experimentation. The core interface for the asynchronous network was set to operate at 1 GHz, compared to 2 GHz for the COSI network's interfaces. We did this to compensate for the link-level protocol used by the COSI models that is not as efficient as the asynchronous protocol.

We changed the COSI SystemC router models from their default behavior. COSI configures its router models to have weighted arbitration based on expected flow-specific traffic volume. While in some circumstances this may be desirable, it caused extremely long latencies for certain traffic flows, as can be seen in some previous results [1]. For this paper, we changed the switch arbitration such that the incoming packet waiting the longest is chosen for output from among the other contending input packets. This improved latency on many paths and drastically reduced maximum latency.

VI. RESULTS

In this section, we present results that show our asynchronous network is much more energy-efficient, but with comparable latency characteristics.

This paper uses message latency, rather than packet latency, as the primary metric for measuring network performance. We do this because a message better represents a useful chunk of data, and how it moves through the network. A message is composed of a number of packets, and is typically managed in the transaction layer protocol. An OCP burst-write is an example of this. Message latency is defined as from the time the first packet of the message leaves the sending core's output buffer and enters the network, to the time the tail packet leaves the network and enters the destination core. The following results

TABLE IV
POWER CONSUMPTION (MW) OF ALL ROUTERS AND WIRES DURING SIMULATION

	Rtr dyn	Rtr leak	Wire dyn	Wire leak	TOTAL
ADSTB sync	3.05	0.66	4.69	0.64	9.04
ANetGen	0.54	0.06	3.93	0.69	5.22
MPEG4 sync	6.75	1.21	12.72	1.85	22.53
ANetGen	1.39	0.10	11.32	2.07	14.87

were generated with a message size of 256 data bytes. Simulations use three traffic burstiness b-values: $\{0.5, 0.65, 0.8\}$. We assume that packets are not dropped, and that the destination cores do not stall, causing a blocked input port.

A. Power Consumption and Area

Power consumption of the whole network over the course of the simulation is shown in Table IV, broken down into the following areas: dynamic power of routers, leakage power of routers, dynamic power of wires, and leakage power of wires. The simulator recorded dynamic energy during operation, and these power values are the total energy divided by the simulation time. The same amount of data is sent for each network, and the simulation times are equal, thus energy and power are analogous in comparing efficiency. These measurements do not include the power of clock distribution, and assume ideal clock gating at the router. These assumptions are very conservative on our part; based on other designs that have been taken to layout, clock power can be approximately 30% of the total NoC power [46]. The wire power includes that from drivers and repeaters (large inverters) and it is significant.

In both cases, the dynamic power of the asynchronous routers is significantly lower than the synchronous routers. The total network power for the ANetGen solution is about 50% of the synchronous solution. The leakage power of the async routers is much smaller compared to the ORION models thanks to their very low area. Total router areas are $15\,630\,\mu\text{m}^2$ (ANetGen) versus $99\,704\,\mu\text{m}^2$ (COSI) for ADSTB, and $26\,050\,\mu\text{m}^2$ (ANetGen) versus $138\,822\,\mu\text{m}^2$ (COSI) for MPEG4. Wire leakage power is higher for the async networks because they have the overhead of parallel route bits and more wires overall. However, the wire's dynamic power is less in the async network indicating that high-throughput paths have shorter wirelength.

Another comparison is the energy-per-flit each type of router uses, and is shown in Fig. 14. The ANetGen router is between a 2×2 and 3×3 Orion router in functionality, as it has three inputs and three outputs (3×3), but only two output possibilities for each input. The energy value of the ANetGen asynchronous router includes the routing bits, while the Orion energy values do not consider the packet's required header flit energy overhead. The energy-per-flit is much lower for our design than what the Orion models predict for traditional synchronous routers.

In summary, the significant power savings of the asynchronous network is due to both router and wire power reductions. However, these results indicate further power improvements should come from more efficient inter-router channel implementations, as the wiring resources consume the majority of the power. Wirelength minimization, especially when very low-power routers are used, is critical. As such, with our asynchronous design, there is little benefit to be gained from further router energy optimization, and future work will concentrate on wire resources and performance.

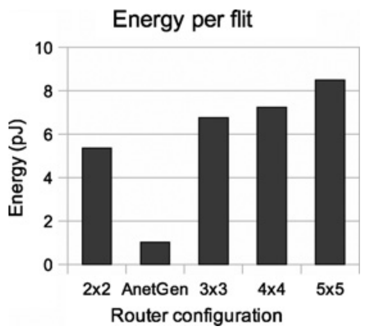


Fig. 14. Energy-per-flit switch. ANetGen radix-3 router and NxN Orion models.

TABLE V
MESSAGE LATENCIES (NS), ABSOLUTE MAXIMUM AND 95% INTERVAL

	Network	Burstiness		
95% less than		**0.5**	**0.65**	**0.8**
ADSTB	Sync.	167	250	264
	ANetGen	78	140	152
MPEG4	Sync.	215	323	514
	ANetGen	144	171	322
Maximum				
ADSTB	Sync.	262	373	373
	ANetGen	195	389	454
MPEG4	Sync.	573	873	1039
	ANetGen	286	1170	2607

B. Message Latency Distribution

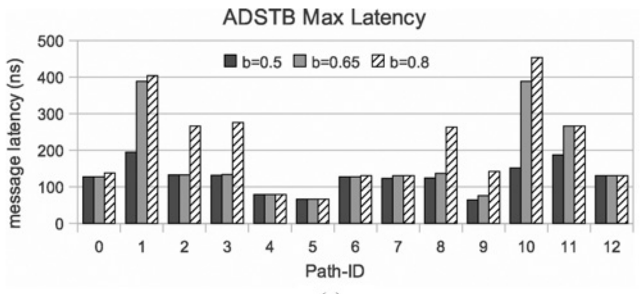
The latency of sending each message was recorded over the course of simulation. An increase in traffic burstiness causes longer periods of contention for switch and link resources. This rising latency is seen in Table V for each benchmark and network. Two types of latency values are shown: the maximum over the course of the simulation and the latency of which 95% of messages were faster than.

The async network sends most messages in less time than the COSI-derived network under all measured conditions. However, under bursty traffic, the higher hop-count and low path diversity of the async network take their toll; the COSI network has lower maximum message latency with both ADSTB and MPEG4. This may be a deciding property between the networks if the SoC needs tighter latency bounds on particular paths.

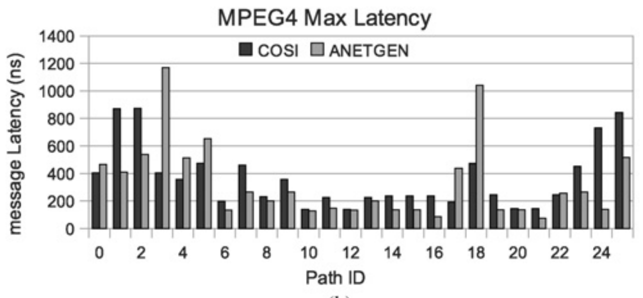
C. Per-Path Message Latency

An understanding of latency and congestion within the network cannot be fully understood by the overall delay alone. Due to the heterogeneity and diverse path properties in an application-specific SoC, there is benefit to analyzing each path through the network separately.

For each path, or pair of communicating cores, Fig. 15 shows the maximum latency seen on each path during simula-



(a)



(b)

Fig. 15. Message latency shown per path, i.e., specific source-to-destination core pairs. (a) Message latency with increasing traffic burstiness in the ANetGen network. (b) Comparing Synchronous and ANetGen networks at b = 0.65.

tion. The first sub-figure indicates that traffic burstiness affects the maximum latency significantly on many paths. For several, this value doubles between $b = 0.5$ and $b = 0.8$. Some paths do not show a change as they do not contend enough with others to be delayed. In the second sub-figure, we compare the maximum latencies of the synchronous network and ANetGen-generated network at a burstiness of 0.65 with the MPEG4 design. The results here are mixed; each network provides lower latency for some paths, but is worse for others. This is do to the differences in topology, available bandwidth, and packet format of the two network types. The async topology is better for those paths that pass through only one or two routers, as an async router has a higher throughput. However, those paths that traverse a large number of async routers can run into more contention. The fewer-hop synchronous design, on those paths, provides lower latency. The async network can also take advantage of short links between certain routers, and not delay a flit an entire cycle. A long link, however, may have less throughput than a synchronous link. The difference in latency between paths within one particular network is due to the topological proximity of sender and receiver, where some paths have fewer hops than others. Both COSI and ANetGen try to map paths carrying more traffic such that they have fewer router hops.

ANetGen has a mechanism for allowing a design-engineer or automation tool to decrease the latency of certain paths, if, for example a sender and receiver communicate rarely, but require low latency when they do. This is a "criticality" weight factored into NoC optimization, and is a unitless value relative in magnitude to average bandwidth. We show the effect of critical-path weighting in Fig. 16. Path ID `1` is from core `babcalc` to `sdram`, with an average traffic rate of 11 MBytes/s which is relatively low for this design. We set

TABLE VI
SOURCE'S OUTPUT BUFFER PACKET DELAY (NS)

	Network	Burstiness		
Median		**0.5**	**0.65**	**0.8**
ADSTB	Sync.	58	170	127 367
	ANetGen	38	55	33 388
MPEG4	Sync.	53.5	108	99 176
	ANetGen	43	62	25 789
Maximum				
ADSTB	Sync.	689	142 066	945 523
	ANetGen	313	34 513	430 515
MPEG4	Sync.	821	135 506	889 027
	ANetGen	433	32 298	416 332

the criticality of this path to 1 (the default), 100 and 1000, generated a new topology and placement with ANetGen for each, and simulated as before. The benefit is an improved worst-case message latency on this path, but it comes at a cost of worse latency on others and approximately a 4.2% dynamic power increase at the `crit = 1000` point.

D. Output Buffer Delay

Another metric of measuring the network performance is the output buffer delay, which is from the time the traffic generator's message is formed into packets and placed in the output buffer to the time the packet enters the network. The buffer entry time is set for each packet by the traffic generator when it pushes an entire message to the buffer at once. Therefore, the last packet of a 64-packet message would have a minimum delay of 64 sender-cycles. The traffic generator operates detached from the network flow control so an infinite buffer is needed to accept its traffic at any time. The network then empties that buffer as quickly as possible. As burstiness increases, the additional delay comes not only from contention within the network, but also from the local traffic generator's attempt to send more data in a shorter time period, possibly exceeding the network's maximum throughput. This grows the buffer more rapidly, increasing delay, even if the network was uncongested. From the results in Table VI, we see that the asynchronous networks consistently have a lower delay for both median and maximum values. This is from the higher throughput available within the async network.

E. Period-Specific Bandwidth

A measure of network performance related to message latency is termed period specific bandwidth (PSB), which is the bandwidth available to a path for some particular period of time. This is in contrast to the average bandwidth that an application produces over a long period. We define a PSB requirement for a source-destination path with a pair of values: $\{V, T\}$, where V is in bytes and T is in seconds. These values would be determined by the SoC application developer. This concept can help in validating an interconnect of, say, a real-time speech recognition SoC, where 18 MBytes must be processed in 0.1 s [47].

For example, the maximum synchronous network message latency recorded in simulation between the *vu* and *sdram* cores of the MPEG4 was 1033 ns at 0.65 burstiness. Suppose this path had a PSB requirement of {256 bytes, 500 ns} (equating

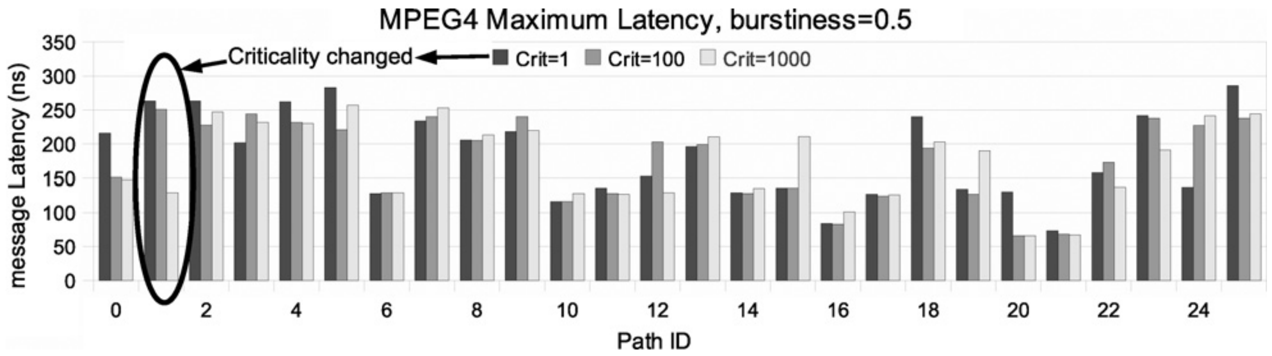


Fig. 16. Message latency resulting from increasing criticality on `Path 1`.

to 512 MBytes/s). This network would be a poor choice because the application would occasionally not receive proper communication throughput, despite the fact that the network did support its average bandwidth of only 64 MBytes/s.

F. Simulation Runtime

These simulations were run on machines with Intel Core i7 and i5 processors. Run-time for the COSI-based synchronous network was 10 min, and that of the asynchronous simulator was 2 min, for a simulation time of 8389 μs. The simulation speed difference is likely due to the asynchronous sim's use of transaction-level-modeling for the links, rather than using a SystemC signal for each physical wire.

VII. Conclusion

In this paper, we presented our asynchronous network design based on simple and efficient circuits and architecture. In order to utilize these effectively, we developed a tool to optimize the network topology and routers' physical placement on the floorplan, based on the communication requirements of an SoC's IP blocks. In addition, our evaluation methods used a self-similar traffic generator that we used to demonstrate how message latency is affected by bursty traffic.

The asynchronous network formed by our tool ANetGen consumed half as much power as the network based on synchronous models. Wires consumed the largest fraction due to repeater energy. Our asynchronous network also had competitive latency, and was superior when considering latency of most messages and output buffer packets. Under very bursty traffic the maximum latency worsened due to the limitations of the topology. The evaluation suggests that the common abstraction of SoC requirements that uses only average bandwidth may not be sufficient. We demonstrated a user-supplied input to ANetGen that improved a specified path's latency at the expense of others, and a slight increase in power.

This paper illustrated that performance metrics can be highly sensitive to the assumed traffic burstiness. The addition of a single-valued burstiness to the tuple representing a network flow's properties should be considered in other NoC evaluations, as well as an appropriate traffic generator. Our SystemC traffic generator may be useful to others wishing to pursue this.

Acknowledgment

The authors would like to thank ARM and IBM for providing the 65 nm library cells and process technology.

References

[1] D. Gebhardt, J. You, and K. S. Stevens, "Comparing energy and latency of asynchronous and synchronous NoCs for embedded SoCs," in *Proc. Int. Symp. Netw.-on-Chips*, May 2010, pp. 115–122.
[2] R. Marculescu, U. Ogras, L.-S. Peh, N. Jerger, and Y. Hoskote, "Outstanding research problems in NoC design: System, microarchitecture, and circuit perspectives," *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.*, vol. 28, no. 1, pp. 3–21, Jan. 2009.
[3] Z. Lu, A. Jantsch, E. Salminen, and C. Grecu, "Network-on-chip benchmarking specification part 2: Micro-benchmark specification," OCP Int. Partnership Assoc., Inc., Beaverton, OR, Tech. Rep., May 2008 [Online]. Available: http://www.ocpip.org/press_release_noc_spec_part_two.php
[4] G. V. Varatkar and R. Marculescu, "On-chip traffic modeling and synthesis for MPEG-2 video applications," *IEEE Trans. Very Large Scale Integr. Syst.*, vol. 12, no. 1, pp. 108–119, Jan. 2004.
[5] P. Bogdan, M. Kas, R. Marculescu, and O. Mutlu, "Quale: A quantum-leap inspired model for non-stationary analysis of NoC traffic in chip multi-processors," in *Proc. Int. Symp. Netw.-on-Chips*, 2010, pp. 241–248.
[6] A. T. Tran, D. N. Truong, and B. Baas, "A reconfigurable source-synchronous on-chip network for GALS many-core platforms," *IEEE Trans. Comput.-Aided Des. Integ. Circuits Syst.*, vol. 29, no. 6, pp. 897–910, Jun. 2010.
[7] I. M. Panades, F. Clermidy, P. Vivet, and A. Greiner, "Physical implementation of the DSPIN network-on-chip in the faust architecture," in *Proc. Int. Symp. Netw.-on-Chips*, 2008, pp. 139–148.
[8] Y. Hoskote, S. Vangal, A. Singh, N. Borkar, and S. Borkar, "A 5-GHz mesh interconnect for a teraflops processor," *IEEE Micro*, vol. 27, no. 5, pp. 51–61, Sep. 2007.
[9] K. Srinivasan, K. S. Chatha, and G. Konjevod, "Linear-programming-based techniques for synthesis of network-on-chip architectures," *IEEE Trans. Very Large Scale Integr. Syst.*, vol. 14, no. 4, pp. 407–420, Apr. 2006.
[10] A. Pinto, L. P. Carloni, and A. L. S. Vincentelli, "A methodology for constraint-driven synthesis of on-chip communications," *IEEE Trans. Comput.-Aided Des.*, vol. 28, no. 3, pp. 364–377, Mar. 2009.
[11] S. Murali, P. Meloni, F. Angiolini, D. Atienza, S. Carta, L. Benini, G. D. Micheli, and L. Raffo, "Designing application-specific networks on chips with floorplan information," in *Proc. Int. Conf. Comput.-Aided Des.*, 2006, pp. 355–362.
[12] D. Atienza, F. Angiolini, S. Murali, A. Pullini, L. Benini, and G. De Micheli, "Network-on-chip design and synthesis outlook," *Integr.-The VLSI J.*, vol. 41, no. 3, pp. 340–359, 2008.
[13] Z. Guz, I. Walter, E. Bolotin, I. Cidon, R. Ginosar, and A. Kolodny, "Network delays and link capacities in application-specific wormhole NoCs," *VLSI Des.*, vol. 2007, no. 90941, p. 15, 2007.
[14] A. Lines, "Asynchronous interconnect for synchronous SoC design," *IEEE Micro*, vol. 24, no. 1, pp. 32–41, Jan.–Feb. 2004.
[15] W. J. Bainbridge and S. B. Furber, "CHAIN: A delay insensitive chip area interconnect," *IEEE Micro*, vol. 22, no. 5, pp. 16–23, Sep.–Oct. 2002.

[16] T. Bjerregaard and J. Sparsø, "A router architecture for connection-oriented service guarantees in the MANGO clockless network-on-chip," in *Proc. Des. Autom. Test Eur.*, 2005, pp. 1226–1231.

[17] 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," in *Proc. Solid-State Circuits Conf. Dig. Tech. Papers*, Feb. 2007, pp. 258–601.

[18] 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," in *Proc. Int. Symp. Asynchronous Circuits Syst.*, 2005, pp. 54–63.

[19] R. R. Dobkin, R. Ginosar, and A. Kolodny, "QNoC asynchronous router," *Integr. VLSI J.*, vol. 42, no. 2, pp. 103–115, 2009.

[20] M. Horak, S. Nowick, M. Carlberg, and U. Vishkin, "A low-overhead asynchronous interconnection network for gals chip multiprocessors," *IEEE Trans. Comput.-Aided Des. Integr. Circuits. Syst.*, vol. 30, no. 4, pp. 494–507, Apr. 2011.

[21] G. Michelogiannakis and W. J. Dally, "Router designs for elastic buffer on-chip networks," in *Proc. Conf. High Perf. Comput. Networking Storage Anal.*, 2009, p. 10.

[22] J. You, D. Gebhardt, and K. S. Stevens, "Bandwidth optimization in asynchronous NoCs by customizing link wire length," in *Proc. IEEE Int. Conf. Comput. Des.*, Oct. 2010, pp. 455–461.

[23] D. Gebhardt, J. You, and K. S. Stevens, "Link pipelining strategies for an application-specific asynchronous NoC," in *Proc. Int. Symp. Netw.-on-Chips*, May 2011, pp. 185–192.

[24] A. Sheibanyrad, I. M. Panades, and A. Greiner, "Systematic comparison between the asynchronous and the multi-synchronous implementations of a network on chip architecture," in *Proc. Des. Autom. Test Eur.*, 2007, pp. 1090–1095.

[25] M. Wang, T. Madhyastha, N. H. Chan, S. Papadimitriou, and C. Faloutsos, "Data mining meets performance evaluation: Fast algorithms for modeling bursty traffic," in *Proc. Int. Conf. Data Eng.*, 2002, pp. 507–516.

[26] R. Thid, I. Sander, and A. Jantsch, "Flexible bus and NoC performance analysis with configurable synthetic workloads," in *Proc. EUROMICRO Conf. Digital Syst. Des.*, 2006, pp. 681–688.

[27] V. Soteriou, H. Wang, and L.-S. Peh, "A statistical traffic model for on-chip interconnection networks," in *Proc. Int. Symp. Modeling Anal. Simul.*, 2006, pp. 104–116.

[28] U. Y. Ogras and R. Marculescu, "Analytical router modeling for networks-on-chip performance analysis," in *Proc. Des. Autom. Test Eur.*, 2007, pp. 1096–1101.

[29] K. S. Stevens, P. Golani, and P. A. Beerel, "Energy and performance models for synchronous and asynchronous communication," *IEEE Trans. Very Large Scale Integr. Syst.*, vol. 19, no. 3, pp. 369–382, Mar. 2011.

[30] J. Sparsø, "Asynchronous circuit design: A tutorial," *Principles of Asynchronous Circuit Design - A Systems Perspective*. Dordrecht, The Netherlands: Kluwer Academic, Dec. 2001, chs. 1–8, pp. 1–152 [Online]. Available: http://www2.imm.dtu.dk/pubdb/p.php?855

[31] C. Seitz, "System timing," in *Introduction to VLSI Systems*, C. A. Mead and L. A. Conway, Eds. Reading, MA: Addison-Wesley, 1980, ch. 7.

[32] K. S. Stevens, R. Ginosar, and S. Rotem, "Relative timing," *IEEE Trans. Very Large Scale Integr. Syst.*, vol. 11, no. 1, pp. 129–140, Jan. 2003.

[33] K. S. Stevens, Y. Xu, and V. Vij, "Characterization of asynchronous templates for integration into clocked CAD flows," in *Proc. Int. Symp. Asynchronous Circuits Syst.*, May 2009, pp. 151–161.

[34] R. Milner, *Communication and Concurrency*. London, U.K.: Prentice-Hall, 1989.

[35] J. Cortadella, M. Kishinevsky, A. Kondratyev, L. Lavagno, and A. Yakovlev, "Petrify: A tool for manipulating concurrent specifications and synthesis of asynchronous controllers," *IEICE Trans. Inform. Syst.*, vol. E80-D, no. 3, pp. 315–325, Mar. 1997.

[36] K. S. Stevens, "Practical verification and synthesis of low latency asynchronous systems," Ph.D. dissertation, Dept. Comput. Sci., Univ. Calgary, Calgary, AB, Canada, Sep. 1994.

[37] Y. Xu and K. S. Stevens, "Automatic synthesis of computation interference constraints for relative timing verification," in *Proc. 26th Int. Conf. Comput. Des.*, Oct. 2009, pp. 16–22.

[38] S. Adya and I. Markov, "Fixed-outline floorplanning: Enabling hierarchical design," *IEEE Trans. Very Large Scale Integr.*, vol. 11, no. 6, pp. 1120–1135, Dec. 2003.

[39] A. Pullini, F. Angiolini, P. Meloni, D. Atienza, S. Murali, L. Raffo, G. D. Micheli, and L. Benini, "NoC design and implementation in 65 nm technology," in *Proc. Int. Symp. Netw. Chip*, May 2007, pp. 273–282.

[40] J. A. Roy, D. A. Papa, S. N. Adya, H. H. Chan, A. N. Ng, J. F. Lu, and I. L. Markov, "Capo: Robust and scalable open-source min-cut floorplacer," in *Proc. Int. Symp. Phys. Des.*, 2005, pp. 224–226.

[41] J. Hu and R. Marculescu, "Energy-aware mapping for tile-based NoC architectures under performance constraints," in *Proc. Asia South Pacific Design Autom. Conf.*, 2003, pp. 233–239.

[42] S. Cahon, N. Melab, and E.-G. Talbi, "Paradiseo: A framework for metaheuristics," in *Proc. Int. Workshop Optimization Frameworks Ind. Applicat.*, Oct. 2005.

[43] A. Kahng, B. Li, L.-S. Peh, and K. Samadi, "Orion 2.0: A fast and accurate NoC power and area model for early-stage design space exploration," in *Proc. DATE*, Apr. 2009, pp. 423–428.

[44] D. Gebhardt and K. S. Stevens, "Elastic flow in an application specific network-on-chip," *Electron. Notes Theor. Comput. Sci.*, vol. 200, no. 1, pp. 3–15, Feb. 2008.

[45] E. B. van der Tol and E. G. T. Jaspers, "Mapping of MPEG-4 decoding on a flexible architecture platform," in *Proc. Media Processors*, 2002, pp. 1–13.

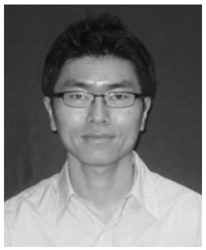
[46] F. Angiolini, P. Meloni, S. M. Carta, L. Raffo, and L. Benini, "A layout-aware analysis of networks-on-chip and traditional interconnects for MPSoCs," *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.*, vol. 26, no. 3, pp. 421–434, Mar. 2007.

[47] B. Mathew, A. Davis, and Z. Fang, "A low-power accelerator for the sphinx 3 speech recognition system," in *Proc. Compilers Architecture Synthesis Embedded Syst.*, 2003, pp. 210–219.

Daniel Gebhardt (S'01) received the B.S. degree in electrical engineering from the University of Portland, Portland, OR, in 2004. He is currently pursuing the Ph.D. degree in computer science from the School of Computing, University of Utah, Salt Lake City.

He was with Hewlett-Packard Corporation, Palo Alto, CA, Synopsys, Inc., Mountain View, CA, and Intel Corporation, Santa Clara, CA, on a variety of software and hardware projects. His current research interests include low-power architectures, design automation techniques, and communication analysis.

Mr. Gebhardt was the Chair of the University of Portland Student IEEE Chapter and received the IEEE Outstanding Student Award in 2004.



Junbok You received the B.S. and M.S. degrees in control and instrumentation engineering from Chung-Ang University, Seoul, Korea, in 1999 and 2001, respectively, and the Ph.D. degree in electrical engineering from the University of Utah, Salt Lake City, in 2011.

He is currently with Texas Instruments, Inc., Dallas, as an Electrical Circuit Design Engineer. He was also with the LG Electronics Digital Media Laboratory, Seoul. His current research interests include asynchronous circuits, network-on-chip, and very large-scale integration architecture and design.

Kenneth S. Stevens (S'83–M'84–SM'99) received the B.A. degree in biology, as well as the B.S., M.S., and Ph.D. degrees in computer science from the University of Utah, Salt Lake City, and the University of Calgary, Calgary, AB, Canada.

He is currently an Associate Professor of electrical and computer engineering with the Department of Electrical and Computer Engineering, University of Utah. He has split time between industry and academia, holding positions with the Fairchild/Schlumberger Laboratory for AI Research, Palo Alto, CA, the Schlumberger Palo Alto Research Laboratory, Palo Alto, and Hewlett Packard Laboratories, Palo Alto, the Air Force Institute of Technology (AF Grad School), Dayton, OH, and Intel's Strategic CAD Laboratories, Hillsboro, OR. He holds several patents, is the co-founder of a software company, and has developed public domain software for the Free Software Foundation. His current research interests include asynchronous circuits, very large scale integration, architecture and design, hardware synthesis and verification, and timing analysis.