21st Century Digital Design Tools

William J. Dally
NVIDIA & Stanford University
2701 San Tomas Expressway
Santa Clara, CA 95050
408-486-2000
bdally@nvidia.com

Chris Malachowsky
NVIDIA
2701 San Tomas Expressway
Santa Clara, CA 95050
408-486-2000
chris@nvidia.com

Stephen W. Keckler
NVIDIA & UT-Austin
2701 San Tomas Expressway
Santa Clara, CA 95050
408-486-2000
skeckler@nvidia.com

ABSTRACT
Most chips today are designed with 20th century CAD tools. These tools, and the abstractions they are based on, were originally intended to handle designs of millions of gates or less. They are not up to the task of handling today's billion-gate designs. The result is months of delay and considerable labor from final RTL to tapeout. Surprises in timing closure, global congestion, and power consumption are common. Even taking an existing design to a new process node is a time-consuming and laborious process.

Twenty-first century CAD tools should be based on higher-level abstractions to enable billion-gate chips to go from final RTL to tapeout in days, not months. Key to attaining this increase in productivity is raising the level of design and using simple, standard interfaces. Designs should be composed from high-level modules – processors, MODEMs, CODECs, memory subsystems, and I/O subsystems – rather than gates and flip-flops. Each module, which we expect to contain 100 thousand to 10 million gates, is easily laid out by today's tools, is placed as a unit, and communicates over a NoC via a standard interface. Restricting modules to standard sizes and aspect ratios further simplifies physical design. We expect even a large chip to contain at most a few thousand such modules and expect the physical design and chip-assembly to take a few days with minimal labor after completion of the module-level design.

Categories and Subject Descriptors
B.7.2 [**Integrated Circuit**]: Design aids – *layout, placement and routing, verification.*

General Terms
Design, Standardization

Keywords
Design automation, NoC, Chiplet, Modularity, Digital design

1. INTRODUCTION
Many designers today are able to realize complex systems with billions of transistors in a few weeks by composing previously designed pieces of intellectual property (IP). The physical design takes only a few hours and changes can be accommodated with a minimum of rework. These designers are working at the board or system level. In contrast, designers working at the chip level take months to complete a design of similar complexity.

The system designer is able to achieve a high level of productivity because the packaging of the components they are composing enforces modularity. Modularity implies information hiding and a fixed, often standard, interface. The system designer typically sees only the specification of a chip they are using. They cannot see or alter the implementation of the chip. Each chip can only interact with the rest of the system over its package pins, a set of fixed and often standard interfaces. The modularity enforced by packaging enables a system designer to use a complex component without incurring a design cost associated with its internal complexity.

In this paper, we suggest that chip designers can achieve productivity comparable to system designers if they adopt a comparable level of modularity. A chip design has no packaging constraints to enforce this modularity found at the system level. The designer must adopt a discipline and a methodology that enforces modularity – and resist the temptation to violate it.

The ideas we describe here are at an early stage. We have not yet implemented any commercial chips using this approach. We provide a number of specific examples to show what might be possible and to give some substance to the proposal. We share these ideas here to encourage others in the design community to help flesh out and then adopt this approach in a quest to build an ecosystem for more productive SoC design. While others have advocated raising the level of abstraction for chip design [1], we propose specific methods for a modular design discipline. The advantages of modular design and information hiding have long been understood in the software engineering community [2]. They apply equally well to hardware design.

We propose two artificial constraints to impose modularity on a design and simplify composition. First, all modules are one of a few standard heights (for example 0.5mm, 1mm, and 2mm) placed in rows, with no global signals routed through a module. Second, modules are restricted to communicate over a network-on-chip (NoC) [3, 4] using standard interfaces. A few flavors of the standard interface would be provided to accommodate a range of bandwidth requirements. Restricting modules to standard heights greatly simplifies the bin-packing problem of module placement. Restricting global signals to dedicated wiring channels enables separate design and verification of modules. Restricting communication to a standard packetized interface enforces information hiding, avoiding the *rat's nest* that can develop when modules are able to see and exploit the internals of other modules.

DAC '13, May 29-June 7, 2013, Austin, TX, USA

The cost of self-imposed modularity is likely to be a chip that is less optimal along some axis than one that violates modularity. It may be slightly larger, slightly slower, or consume slightly more power than a chip that is carefully handcrafted without imposing this discipline. However, this slightly sub-optimal chip will get to market months before its handcrafted cousin. The revenue generated and market position gained by earlier market delivery could easily justify, for instance, consuming a bit more die area. If reclaiming the additional area is important, a handcrafted design can follow the modular design as a cost reduction step.

Semiconductor manufacturers have already shown a willingness to adopt modularity at the expense of optimality in the use of standard cells. Making all cells a standard height and enforcing fully-restored CMOS electrical interface constraints imposes an area penalty. We have seen cases where redesigning a standard-cell unit using full-custom design can save upwards of 50% in area [5]. Yet this is rarely done in practice because full-custom design at the transistor or gate level is simply too costly. Interestingly, recent work indicates that the gap between automated standard cell and custom design methodologies may be closing [6].

The design discipline we are proposing is simply up-leveling the concept of standard cells. Like standard cells, we propose cells with a common height to simplify composition. The difference is that our proposed row height is 0.5mm while a standard cell's height is 6 to 12 wire pitches, about 1µm in a typical 28nm process. A unit of modularity that is 6-12 wire pitches in height was the right abstraction when 10^3 to 10^5 such units fit on an edge of a chip. Now that we are producing 10^8 to 10^9-gate chips, we need to scale up unit modularity to a unit that is at least 0.5mm on a side, resulting in designs that have perhaps a thousand of these new units on a chip.

As we up-level standard cells, the need for a standard interface changes. With standard cells, the standard interface is a fully-restored static CMOS logic signal. Pre-charged signals, dynamic nodes, pass-gate signals, and flip-flop storage nodes are not allowed to be exposed outside the cell. Flip-flop and latch cells typically use several inverters solely to provide this isolation. Enforcing this standard signaling convention ensures that any standard cell can talk to any other standard cell without fear of dynamic charge sharing or other subtle circuit effects. As we up-level to modules in our proposal, the standard interface becomes a packetized transport protocol, namely a NoC interface. This interface ensures that any module can talk to any other module without worrying about subtle protocol effects or timing surprises when integrated at the chip-level.

The remainder of this paper explores the concept of high-level modular design in more detail, focusing on the design elements of chiplets, standard interfaces, NoC, and I/O. We conclude with a case study of an experimental chip that employed some of these principles and a call to arms to the chip design community.

2. CHIPLETS

Figure 1 shows how we envision modules or *chiplets* being placed on a portion of a chip. The figure shows one corner of a chip with I/O chiplets along the periphery and rows of single, double, and quad-height modules. Dedicated wiring tracks are provided between the module rows. To preserve modularity, routing through modules is prohibited.

All modules are a fixed height but have an arbitrary width. As with standard cells, this approach facilitates simple placement in rows while allowing modules that span a wide range of areas. We

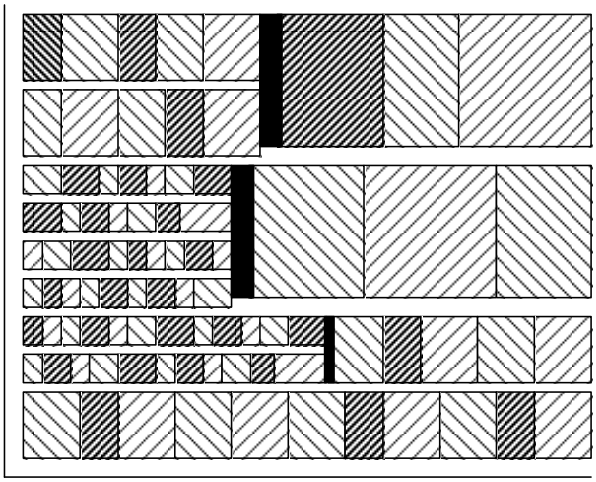

Figure 1: Layout of chiplets on a modular SoC.

envision perhaps three to four module heights to allow a wide range of module sizes without requiring extreme aspect ratios that tend to result in inefficient layout. Chiplet heights can be arranged as a multiple of the power grid so that vias from the power grid to the chiplet are in the same position in each row.

Assuming modules of 0.5, 1, and 2mm in height (single, double, and quad height), the design can accommodate areas from $0.06mm^2$ (0.5mm x 0.125mm) to $16mm^2$ (2mm x 8mm) with aspect ratios no worse than 4:1. In a typical 28nm process, this range of areas would enable chiplets to include 190k to 48M gates. Modules smaller than $0.06mm^2$ (190k gates) can be combined with other small modules into a single module sharing a NoC interface, using conventional design techniques. Modules larger than $16mm^2$ (48M gates) can be placed as a large macro or be decomposed into multiple, smaller modules.

The top 5% of a chiplet's height (25, 50, or 100µm for single, double, and quad-height chiplets) can be reserved for a wiring channel. The NoC logic and its channels are implemented in this area. A 25µm wiring channel for single-height modules provides over 250 wiring tracks per layer, sufficient bandwidth for a row of single-height modules. Should additional bandwidth be required, a row of modules may be omitted and its space used entirely for wiring. NoC connections between wiring channels are accomplished by inserting NoC wiring modules into each row. The connection from each module to the NoC takes place entirely along the upper and/or lower edge of the module using one or more of the fixed interfaces described below.

Providing dedicated wiring area simplifies layout and timing closure, allowing each module's layout to be finalized and timing verified independent of global wiring. It also makes the global wiring more predictable. Timing of the NoC is not dependent on the design of any module.

The center of a module only has connections to power, ground, and external I/O. Chiplets can be powered via global power and ground grids that are distributed on upper layers of metal and supplied via an array of supply balls.

I/O chiplets are modules that include I/O drivers, pads, and signal balls that connect to the package. The presence of the signal balls, and any special supply balls needed to support them may interrupt the power grid and the array of power supply balls. Otherwise an I/O chiplet is no different from any other chiplet. We anticipate that I/O chiplets will include not just the I/O circuit elements, but

also the logic associated with those pads. For example, a memory controller would be included in the chiplet containing SDDR I/O pads, and a PCIe root complex would be contained in the chiplet containing PCIe PHYs. While I/O chiplets will typically be placed along the periphery of the chip, area I/O is also possible. The implications of I/O chiplets are described in more detail in Section 5 below.

As shown in Figure 1, we expect the tallest rows of chiplets to be placed toward the center of the chip so that the larger wiring channels associated with these chiplets are available in the center of the die, where the highest wiring density is expected. At the point where row heights change, a NoC chiplet is inserted (solid black in Figure 1) to route packets between the different channels.

3. STANDARD INTERFACES

One of the motivations of advocating for standard interfaces in our modularized design proposal is to ensure that relatively straightforward automation can fully construct that top-level design ensuring functional correctness, while avoiding timing and routing surprises.

3.1 The Rat's Nest

Perhaps the hardest step of converting an existing design to the modular approach we advocate is eliminating the rat's nest of connections between modules on a modern SoC and replacing this wiring with disciplined communication over standard interfaces. Without the discipline of modularity, it is not unusual to have thousands of connections between modules. A designer working on one module needing to know the status of an internal component of another module is tempted to just reach inside that module and grab the relevant signal. To the designer, this option appears inexpensive as it only requires typing a signal name in Verilog or adding a wire to a module specification. In practice, it is quite costly to both the logical and physical design of the chip.

Violating strict modularity increases the logical design complexity in several ways. Accessing internal signals of modules violates the principle of *information hiding* and in doing so makes module designs fragile and brittle. Changing or verifying a design becomes challenging. Substituting a new module for an old module may be very difficult when several other modules on a SoC depend on internal signals and potentially subtle or non-obvious behavioral properties. Except in rare circumstances, accessing internal signals of a module is simply bad design.

Violating strict modularity also incurs physical design costs, including area and design time. Random wiring between modules is costly in terms of area because it has a low duty factor. A separate wiring track (or tracks for multi bit signals) is allocated for each signal accessed in this manner, even though the typical signal is relevant (both changing and observed) a tiny fraction of the time. Except for signals with very high duty factor, a far more area-efficient approach multiplexes many signals over a shared communication structure, such as a NoC, rather than allocating dedicated wiring to each signal. Random wiring also makes full-chip timing closure more difficult and time consuming, as internal module signals may have little local timing margin and translate into top-level timing violations.

3.2 A Modular Interface

Instead of unstructured module interfaces and access to internal module signals, we propose that modules communicate with other modules by sending packets over a NoC. To illustrate how this would work, we will propose one such packet-based NoC design.

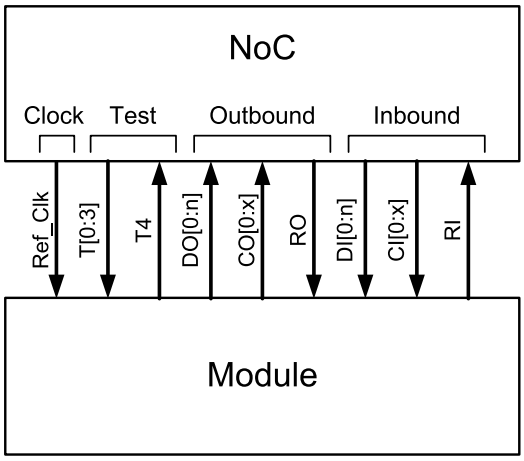

Figure 2: Standard module interface.

In this design, a packet consists of one or more 32-bit *flits*. The destination of the packet is encoded in the first flit. The remainder of the packet is user-defined. Figure 2 shows the standard module interface. The interface consists of separate outbound (CO, RO, DO), inbound (CI, RI, DI), test, and clock signal groups. The inbound and outbound signal groups are further divided into data (DO and DI), control (CO and CI), and flow control (RO and RI). To allow the bandwidth to be configured for modules with different needs, the data fields (DO and DI) can be configured to be 32, 64, 128, or 256 bits wide, enabling modules to send 1, 2, 4, or 8 flits each cycle. The least significant 16 bits of the first flit of a message encodes the message destination. The remainder of the bits is defined by the two endpoints and is not interpreted by the network. The control (CO and CI) signal groups encode how many flits are valid, whether the last valid flit terminates the packet, and on which virtual channel the packet is sent [7]. The flow control lines (RO and RI) flow in the direction opposite to the corresponding data and control lines to signal when the data and control are accepted by the interface.

To send a packet, a module places the first flit(s) on the DO lines and sets the CO lines appropriately. The data and control are held until the RO line is sensed high at the end of a cycle, indicating that the data has been accepted. A packet that is longer than the width of the DO lines is sent over multiple cycles. Flits from different virtual channels can be interleaved on a cycle-by-cycle basis by setting the CO lines appropriately. Modules receive arriving packets using an identical interface but in the opposite direction.

The test interface is comparable to JTAG and allows a tester to connect to and independently test each module via the NoC. The NoC must also include a capability to test the NoC itself and the module to NoC interfaces.

The clock field provides a standard reference clock to the module; we expect that a 1GHz reference will be suitable for most SoCs. The NoC operates globally at this 1GHz clock rate and the module to NoC interface is synchronous with this clock. Each module may operate using an arbitrary clock and arbitrary timing convention but is responsible for generating its local clock (or clocks). It is also responsible for synchronizing its local signals with the NoC interface, for example by using a FIFO synchronizer. A standard synchronizer would be available for inclusion in each module. This approach to timing allows

Table 1: Example NoC communication.

	Source		Destination	
Cycle	**CO**	**DO**	**CI**	**DI**
1	2, No, 3	A, B		
2	1, Yes, 3	C		
…				
i			1, No, 3	A
i+1			1, No, 3	B
i+2			1, Yes, 3	C

complete flexibility in module design (including asynchronous) while providing a global timing reference that can be used to generate a local clock.

Table 1 illustrates how an encoder chiplet with a 2-flit (64-bit) interface sends a 3-flit (96-bit) message to a memory controller with a 1-flit (32-bit interface). On the first cycle, the encoder puts two flits (A and B) on the DO signals and sets CO to indicate 2 valid flits, no termination, and virtual channel 3. The low 16-bits of flit A encode the network address of the memory controller. On the second cycle, the remaining flit (C) is placed on the low DO lines and the CO signals are set to encode 1 flit, packet termination, and virtual channel 3.

A few cycles later, the packet arrives on the CI and DI pins of the memory controller. Three cycles are required for the packet to be delivered at the far end over the 32-bit controller interface. The network converts the packet from 64-bit wide to 32-bit wide at the destination NoC interface.

Providing a variable width interface enables the NoC to efficiently support a wide variety of interface bandwidths ranging from 4GB/s to 32GB/s, with a 1GHz NoC clock. Providing automatic conversion between widths in the network allows any module to talk to any other module regardless of their width. To simplify width conversion, the physical layout of the multi-flit data buses are bit interleaved. If interface bandwidth higher than 32GB/s is required, multiple parallel 32GB/s NoC interfaces can be instantiated on a module.

4. NETWORK-ON-CHIP (NoC)

We envision that the NoC for a particular SoC will be customized based on the anticipated traffic matrix for that NoC and the quality of service (QoS) required for each traffic flow. The traffic matrix may be explicitly specified or may be extracted from simulations of the SoC. Starting with the traffic matrix and the placement of modules, a NoC synthesis tool generates a NoC that provides the required throughput for each flow with a minimum of latency and energy. The tool configures a NoC from a library of channels, routers, NICs, and controllers. The NoC synthesis tool also assigns virtual channels to particular flows to meet QoS requirements of the application.

We expect the NoC channels to be implemented with optimized circuits that employ equalization and low-swing signaling to achieve substantially lower energy and latency than a wire driven by a standard full-swing CMOS gate. The use of low-swing signaling is enabled by dedicated wiring channels, which facilitates control of wire geometry and crosstalk. We expect the performance of these optimized wires to more than offset the increased energy and latency due to the insertion of routers in the communication path.

Some SoCs include isochronous flows that have fixed bandwidth requirements with tight latency constraints. We expect the NoC to handle such flows by provisioning channels with sufficient bandwidth to meet the worst-case requirements of all simultaneous flows and then assigning isochronous flows to a high-priority virtual channel.

Some interactions between modules on a SoC involve simple events. For example, one module may need to know when a queue in another module reaches a threshold. In a traditional rat's nest SoC design, a dedicated wire would be run between the two modules with this indication. On a NoC with standard interfaces, such events are sent as simple single-flit packets on the highest-priority virtual channel. To minimize latency, an event packet can be launched in the same cycle that the event is detected.

5. I/O CHIPLETS

One of the most visibly and structurally obvious holdovers from the early days of IC design is the I/O ring. This ring is the traditional repository of all the I/O buffer and power/ground pads for the device. The I/O padring requires up-leveling in a world that has full subsystem-sized modularity.

One of the obvious problems with the I/O ring structure is that it separates the functional block with the I/O need from the actual I/O pads. This independence provided desirable isolation of all the challenges of high energy and often electrically-hostile external interfaces from the more standardized and less tolerant small-signal domain of the internal logic. However the costs include long wires, non-standard module interfaces, inter-module timing challenges, global wiring congestion, unnaturally constrained floor plans and aspect ratios, and a considerable amount of exposed detailed module specific knowledge and behavior. None of these properties match our goal of design modularity with encapsulated functionality communicating via standard interfaces.

Instead we propose to abandon the global I/O ring structure altogether, in favor of an I/O chiplet architecture which embeds the I/O pads with its interface logic. At the chip level, I/O pads are spread throughout the whole of the device using area I/O, rather than being constrained to the periphery. Within a chiplet, the I/O can still be distributed peripherally. However, the main goal is to make the physical implementation of the module completely self-contained. This approach will require, for instance, that each module design solve its own di/dt and noise issues, provide for its own connections to power and ground resources, and accommodate standardized test infrastructures. While in some respect, we have just moved I/O design challenges from the chip to the chiplet, making chiplets and their I/O truly composable will facilitate much easier chip assembly and full-chip electrical verification.

6. A CASE STUDY

The TRIPS processor was a research prototype chip that employed much of the design style described above [8]. The TRIPS chip was designed to demonstrate distributed processor and memory architectures and included two processors and a distributed non-uniform NUCA cache. As shown in chip floorplan of Figure 3, the chip was implemented using 106 instances of 11 different tiles (chiplets). The diagram shows the tile boundaries, annotated with tile names, along with the outlines of the major RAM structures within each tile. Each processor was composed of 30 instances of five types of tiles, connected via a NoC and a small number of control networks. The five tiles (GT, RT, ET, DT, and IT) represented the major functions of the distributed processor, including global control, register file, execution units,

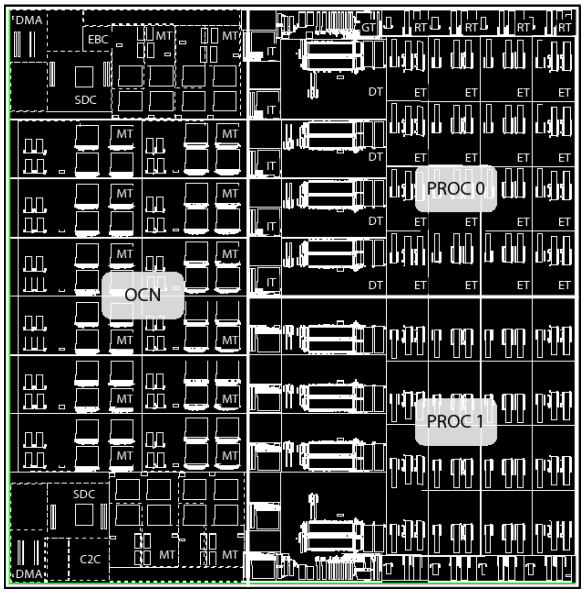

Figure 3: Floorplan of a tiled chiplet architecture.
© 2006 IEEE. Reprinted, with permission, from [8].

data caches, and instruction caches. The tiles ranged in size from 1mm2 to 9mm2 (in a 130nm process) and the design intended to align the tiles in both the X and Y dimensions. The on-chip NUCA memory system on the left side of the chip was composed of 16 level-2 cache tiles, memory controller tiles, DMA engines, and external communication interfaces.

The logical connections between the tiles in each processor included a general purpose NoC for data operand transmission, as well as several control networks to implement the distributed uniprocessor control protocols, such as instruction distribution. Each tile contained a NoC router as well as pipelined channels for the control protocol networks. All tiles connected via abutment with no inter-tile global wiring. The memory system only included a NoC (separate from the operand NoCs in the processors) and the tiles there were intended to connect via abutment.

In light of the design methodology described above, the TRIPS design achieved some of the physical design productivity goals. The tiled design in fact did eliminate all global wiring, making chip assembly trivial. Likewise, global timing closure was easy because of the clean timing interfaces at each tile boundary. In most areas of the chip, the tiles were sufficiently aligned to enable connection via abutment.

However, a number of challenges prevented the design from being as modular as intended. First, the aspect ratios of the tiles were not always well matched in both dimensions. For example, the DT needed to match the width of the GT and the height of the ET, but the fit was not perfect. Second, the tiling approach broke down in the corners of the memory system because space and aspect ratio constraints required the MTs on the top and bottom to be arranged vertically (instead of horizontally) so that the different controllers could fit in a square space, rather than a thin rectangular space. Four different tiles (the two memory controllers, the chip-to-chip router, and the I/O controller) each required a substantial number of chip pins, which far exceeded the size of each of controller. Instead, the pins were spread throughout the chip with top-level wiring connecting the controllers to their I/Os.

Finally, the design intended to perform a single physical design for each type of tile, with the layout and intra-tile wiring replicated during chip assembly. However, because the I/Os were spread throughout the chip, the I/O pattern caused non-uniform obstructions in different physical instances of the same tile. As a result, each tile was actually placed and routed individually, subject to the different I/O blockages within each instance. However, because all of the tile instances were independent, the placement and routing of each block could be performed in parallel. The entire chip physical design including placement, routing, and chip assembly was automated and could be performed in about a day.

While the TRIPS chip demonstrates a number of advantages of raising the level of abstraction for full-chip logical and physical design, it also highlights a number of the challenges, including imperfect chip component tiling and a need for both data and control networks. Arguably, the TRIPS chip is a hard design to tile because each tile contains only a small portion of the design, such as a slice of the level-1 data cache or a slice of the register file. Contemporary SoCs composed of commodity building blocks likely require fewer if any communication channels that could not be transmitted via a general purpose NoC. However, composing a two-dimensional tiled design in which tiles align in both dimensions may still be difficult.

Many of the differences between the TRIPS design and the approach we propose here are due to differences in requirements. Aligning modules in two dimensions and connecting by abutment is an ideal way to achieve modularity for a single design. Aligning uniform height modules in one dimension and routing the NoC in wiring channels external to the modules is better suited to building an ecosystem capable of quickly generating a wide range of SoCs.

7. COMPROMISES

While we expect the modular approach described above to serve the vast majority of SoCs, two areas may require compromises to the proposed methodology.

Out-of-band signals: The vast majority of signaling between modules can be accomplished over the NoC. Even most low-latency event signals can be efficiently packetized, with appropriate use of virtual channels for providing priority and QoS. However, in rare cases modules may need to exchange signals directly, bypassing the NoC. Such signals may arise in things like interrupt signals when partitioning a complex block into multiple tightly coupled chiplets. This practice of sending *out-of-band* signals should be strongly discouraged lest designers resort to it to avoid the effort required to packetize their interfaces. However, when needed, the methodology we describe here can be extended to allow direct connection of signals between modules. By requiring that such signals be sampled by the NoC clock at both the outbound and inbound interfaces and be routed entirely in the wiring channels, module verification/timing closure and global verification/timing closure can still be decoupled.

Captive vs. non-captive signal balls: Making all of the signal balls used by an I/O module *captive* (contained within the boundaries of the module) keeps the effect of these balls local to the module. Modules can be composed in any way without the ball pattern of one module affecting the functionality of another module. However, this restriction becomes costly in area when a module has a large number of balls but only a small area for circuits and logic. The alternative is to allow the module to route signals via a redistribution layer to non-captive balls, risking

disruption to the power and ground grids of any modules placed under the balls.

8. A CALL TO ARMS

We articulate our vision of the future of SoC design with the goal of engaging the broader design community to adopt a similar vision and work with us to build the ecosystem needed to realize it. Four major components are required to make this vision a reality: module placement software, a NoC generator, IP modules, and chiplet verification tools.

Module placement: The module placement software is perhaps the simplest element as it is just a standard-cell placement system scaled up to larger modules and modified to deal with multiple row heights. The inter-module traffic matrix can be used instead of a netlist to derive module affinity. The standard techniques of graph partitioning for coarse placement followed by iterative refinement should yield a good placement. Given the small number of modules being placed (10^3) we expect a good solution to be reached within only a few hours of CPU time.

NoC generator: The NoC generator is perhaps the most complex element of the ecosystem. The NoC generator uses the module placement and the traffic matrix to derive the communication loading of key *cuts* of the SoC. It then determines a topology to provision sufficient communication across each cut at minimum cost. We expect this step to employ a standard topology such as a flattened butterfly as the underlying substrate, and adapt it to provide more bandwidth where needed and less bandwidth where there is little demand. The final step of the NoC generator instantiates library modules for channels, routers, and NICs to realize the selected topology.

IP modules: The most important part of the ecosystem is the IP because a critical mass of it is required to make the ecosystem viable. Chiplets for processors, on-chip memory, off-chip memory controllers, common peripheral interfaces (PCIe, SATA, USB, Ethernet, etc.), and common CODECs and MODEMs are needed to enable SoC designers to quickly assemble chips from a library of modules. IP modules may be soft (synthesizable Verilog) or hard (placed and routed cells). A proper IP module will include the module functionality packaged behind a standard NoC interface of an appropriate width. Hard IP is simply soft IP synthesized for a particular process and with placement, routing, timing closure, and test generation completed. The hard IP is ready to be placed in a modular SoC.

We expect that most of the required IP already exists and simply needs to be tied to a standard interface. For a given user, some IP will be generated in house, some IP will be provided by tool vendors, and some will be licensed by third parties. Our vision is that simpler composition of modules will lower the barrier to creating a vibrant market of standard high-level digital IP chiplets.

Chiplet verification tools: The challenges to widespread adoption of this approach lies in inter-module interconnects and establishing the necessary conventions and standards to allow automation to be responsible for all the heavy lifting. Besides functional connection requirements within the constraints of standardized module heights, formal specifications for ensuring appropriate power/ground resources and test interfaces to each module will be needed and require verification.

New analysis tools will be required to focus on the chiplets to ensure that they are built correctly and follow established rules so that they can be safely re-instantiated in any circumstance. Compositions of chiplets should be *correct by construction.* Chiplet composition tools and appropriate checkers will be needed to ensure that all rules are followed and proper inter-chiplet connections are made. Given the independent nature of these chiplets, new automated test generation and test application methodologies and tools will be required. Such tools would allow the testing requirements of a module to be encapsulated so that those integrating a chiplet can again be isolated from the details of the test, yet be assured a quality result each and every time the module is used.

We have heard both engineers and venture capitalists lament the high cost of designing a SoC today. Estimates are that $50M is required to get a complex digital SoC to first prototype [9]. Some attribute the dearth of fabless semiconductor startups and the slowdown of innovation in the field to this high cost. While some of this cost can be attributed to mask sets and wafer starts, the bulk comes from design and verification. Our vision of a modular SoC built from standard chiplets offers a path toward greatly reducing the non-recurring cost of a SoC. In doing, so we hope that it will spur a new generation of fabless semiconductor startups and encourage more innovation in chip architecture and design.

9. ACKNOWLEDGMENTS

We thank our colleagues at NVIDIA whose work on numerous SoCs has shaped our thoughts on this topic.

10. REFERENCES

[1] Borkar, S. 2009. Design perspectives on 22nm CMOS and beyond. In *Proceedings of the Design Automation Conference*, July 2009, pp. 93-94.

[2] Parnas, D. L. 1972. On the criteria to be used in decomposing systems into modules. *Communications of the ACM*, *15*(12) 1972, pp. 1053-1058.

[3] Dally, W. J. and Towles, B. 2001. Route packets, not wires: on-chip interconnection networks. In *Proceedings of the Design Automation Conference*, June 2001, pp. 684-689.

[4] Dally, W. J. and Towles, B. P. 2003. *Principles and Practices of Interconnection Networks*. Morgan Kaufmann.

[5] Dally, W.J. and Chang, A. 2000. The role of custom design in ASIC chips. In *Proceedings of the Design Automation Conference*, June 2000, pp. 643-647.

[6] Ueno, K. et al. 2007. A design methodology realizing an over GHz synthesizable streaming processing unit. In *Proceedings of the IEEE Symposium on VLSI Circuits*, June 2007, pp. 48-49.

[7] Dally, W. J. 1992. Virtual-channel flow control. *IEEE Transactions on Parallel and Distributed Systems, 3*(2), 1992, pp. 194-205.

[8] Sankaralingam, K., et al. 2006. Distributed microarchitectural protocols in the TRIPS prototype processor. In *Proceedings of the International Symposium on Microarchitecture*, December 2006, pp. 480-491.

[9] Goering, R. (2009) *Are SoC Development Costs Significantly Underestimated*?, http://www.cadence.com/Community/blogs/ii/archive/2009/09/24/are-soc-development-costs-significantly-underestimated.aspx