Network on a Chip: An architecture for billion transistor era

Ahmed Hemani[1], Axel Jantsch[2], Shashi Kumar[5], Adam Postula[6], Johnny Öberg[3], Mikael Millberg[4], Dan Lindqvist[7]

[1,2,3,4]ESDlab, Dept. of Electronics, Royal Institute of Technology, KTH, Electrum, 164 40 Kista, Sweden. ahmed@ele.kth.se

[5]School of Engineering, Jönköping, Sweden (Shashi could you please complete/correct this)

[6]University of Queensland, Brisbane, Australia (Adam could you please complete/correct this)

[7] Ericsson Radio Systems AB, Färögatan 6, 164 40 Kista, Sweden.

Abstract. Looking into the future, when the billion transitor ASICs will become reality, this paper presents Network on a chip (NOC) concept and its associated methodology as solution to the design productivity problem. NOC is a network of computational, storage and I/O resources, interconnected by a network of switches. Resources communcate with each other using addressed data packets routed to their destination by the switch fabric. Arguments are presented to justify that in the billion transistor era, the area and performance penalty would be minimum. A concrete topology for the NOC, a honeycomb structure, is proposed and discussed. A methodology to support NOC is presented. This methodology outlines steps from requirements down to implementation. As an illustration of the concepts, a plausible mapping of an entire basestation on hypothetical NOC is discussed.

1. Introduction

Complexity of VLSI circuits in terms of logic transistors that can be integrated on a chip is increasing at the rate of 58% per annum compounded. Whereas the design productivity growth is increasing at 21% per annum[2]. Closing this gap is the holy grail of the design automation community. Increasing abstraction and increasing team size are the twin weapons used by the design community to bridge this gap. Though this is proving increasingly difficult.

The design productivity gap is caused not just by the increasing number of transistors to cope with but also due to the increasing number of factors that must be considered while designing VLSI circuits. According to The international Technology Roadmap For Semiconductors: 1999 the factors that must be considered by a VLSI designer when we reach the billion transistor mark are: *Functionality, testability, wire delay, power management, embedded software, signal integrity, RF, hybrid chips, packaging and management of physical limits*. This in sharp contrast to just two factors: *functionality and testability* that were considered at the begining of VLSI era.

The thesis of this paper is that the present generation and the emerging design methods and tools and the associated chip architecture/platforms, while providing intermediate solutions, will not survive when we reach the billion transistor mark. We propose a platform and methodology that will address these factors and still allow companies to meet the time to market and make profit.

The cornerstone of the proposed platform is a concept that we call as *Network On a Chip or NOC*. NOC will be composed of computing resources in the form of processor cores and field programmable logic blocks, distributed storage resources, programmable I/O and all these resources interconnected by a switching fabric, allowing any chip resource to communicate with any other chip resource. We further argue that, providing this intra chip communication based on switches will cause little or no overhead in the billion transistor scenario.

The proposed design methodology will allow designer to write applications in concurrent and/or high level language like C++, Java, SDL, MATLAB etc. A compiler will map the application to the platform by partitioning the functionality and mapping them to available resources on the platform and automatically refining the inter resource communication, the latter being the key innovation introduced by this methodology. Present generation of methodologies are stymied by inability to estimate and lookahead in the system design phase when the geometry of the final solution is not known. In case of NOC, this problem does not exist because the platform and its geometry is known to the synthesis/refinement tools.

This is a concept paper that presents ideas and evaluates them qualitatively and analytically. Concrete implementations or results are not presented and would follow as the research progresses and will be published later.

2. Emerging methodologies

Behavioural synthesis is a promising design technology and inspite of the delay in it getting widespread acceptance, it will eventually replace and/or subsume RTL synthesis. Behavioural synthesis as apparent from the figures released by Synopsys is in many cases providing impressive productivity gain. The problem is that Behavioural synthesis is a technology that automates implementation of functions/algorithms, which are small parts of even today's SOCs and will solve an insignificant problem in the billion transistor era. In the next generation methodology, behavioural synthesis tightly coupled with physical design is likely to play a similar role as today's module generators for rapidly generating IPs by the independent IP vendors or silicon vendors that today supply cell libraries.

IPs and their associated design methodology have been in the limelight and promise to provide dramatic productivity improvement by implementing reuse at functional block or subsystem level. We do believe that IPs will be a solution to the design productivity problem in short term but not in long term after the geometries shrink to less than 100 nm.

An inflection point is reached at around 100 nm design and below. More automated detailed design cannot occur until unbuffered, predictable timing synthesis of small blocks is possible. For 100 nm technology, this implies irregular logic blocks must be no larger than about 100K gates. For an ASIC, this implies that approximately 200 to 400 such blocks must be specified and planned before detailed design can start. This is too much complexity for the system design phase and therefore requires the introduction of a new middle layer of design between detailed logic design and system specification/partitioning. Defining the entry into and exit out of this middle layer of the design process will be required.

Platform based design is another promising methodology that actually builds on using IPs. The key concept here is separation of architecture design phase from the function design phase. The developed architecture or platform can then be reused in many designs by systematically deriving the derivative architecture/platform. We believe that the PBD approach is a step in the right direction but it would still fall short of the requirements for the billion transistor era mainly because the engineering cost of doing detailed design of platforms would be too high to be amortised over a few derivative designs.

3. Emerging platforms

If one conceptually thinks of today's SOC architectures in terms of three kinds of resources: computational, storage and I/O, then the interconnectivity among them is a design time decision and remains fixed and cannot be changed once the design is done. In contrast, the computational

functionality can often be changed after design has been done because it is often implemented as software these days. As such, this fixed chip level interconnectivity is an impediment in using the platform in a more general sense.

FPGAs, in contrast have interconnect resources that are reprogrammable and hence more flexible. The problem with FPGAs is that the computational resources typically have fine granularity and thus the demand on interconnect resources is very high and proves to be the bottleneck in performance.

Recognising the growing share of functionality that is implemented as software, FPGAs are adapting. Xilinx for instance offers IBM's power PC embedded cores (see [4]), Altera offers ARC cores (see [5]) and Lucent is offering Motorola cores.

Chamelon(see [3]) has an interesting architecture that offers an embedded ARC core together with reconfigurable computing fabric to form a general purpose platform for a wide range of applications.

All these reconfigurable platforms are inspiration for the proposed Network on a Chip platform. The reconfigurablity there is mainly at the logic level, what we need in the billion transistor era is reconfigurability at the task or process level.

4. Network on a chip platform

FPGAs, we believe are the harbingers of the platforms or architectures to emerge in the billion transistor era. We propose Network on a chip as logical successors of FPGAs and have the following characteristics:

- These platforms/architectures would be generic, like FPGAs, and not tailored to a specific application domain. This is essential to allow amortisation of the huge engineering cost of designing such platforms for a large number of applications. As this is a long term prediction, this argument could be wrong if a particular domain appears to be large enough to allow customising the platform for such a domain.

- Computational resources would be in the form of processor cores. These processor cores, could be general purpose cores like ARM, DSP cores, protocol engines or even blocks of FPGAs for implementating computation in hardware. At present, .18 micron technology is proving to be fast enough to allow baseband DSP algorithms for today's narrowband access technology to be imlemented in software. We believe when we move to .1 mircon and beyond, technology will be allow transition to broadband. Moreover, the verification problem will be so intractable and engineering cost of implementing logic in hardware so high that most system designers will choose to implement logic in software anyway. In other words, implementing logic in software is not only a way to generality of platform, but also a necessary way to live with the verification problem.

- Storage resources would be distributed to avoid the high cost of accessing data across the chip. SRAMS, DRAMS, FLASH should be available in different sizes to fulfill a variety of requirements.

- I/Os would be general purpose and configurable to handle coded or uncoded data and would have some buffering capability.

- The interconnect scheme is in many respects at the heart of the NOC architecture. Each resource, wether it is computational, storage or I/O, will have an address and will be inter-

connected by a network of switches. These resources, will communicate with each other by sending addressed packets of data, routed to their destination by the network of switches.

- Though many topologies are possible, we present here one possibility for further discussion and analysis. The overall organisation is in the form of a honeycomb, as shown in Figure 1. The resources - computational, storage and I/O - are organised as nodes of the hexagon with a local switch at the centre that interconnects these resources. Hexagons at the periphery would be primarily for I/O, whereas the ones in the core would have storage and computational resource.

- Storage resources can be combined to create a larger virtual storage. This would be transparent to the applications; the refinement tool would be responsible for allocating and combining memories to create larger ones if necessary.

- Each resource, located on a hexagonal node being connected to three switches, can reach 12 resources with a single hop. To further improve the connectivity, switches are directly connected to their next nearest neighbours, as shown in Figure 1, allowing any resource to reach 27 additional resource with two hops. As a last measure to further improve connectivity, every alternate switch is directly connected making each resource element reach a lot more elements with minimal number of hops.

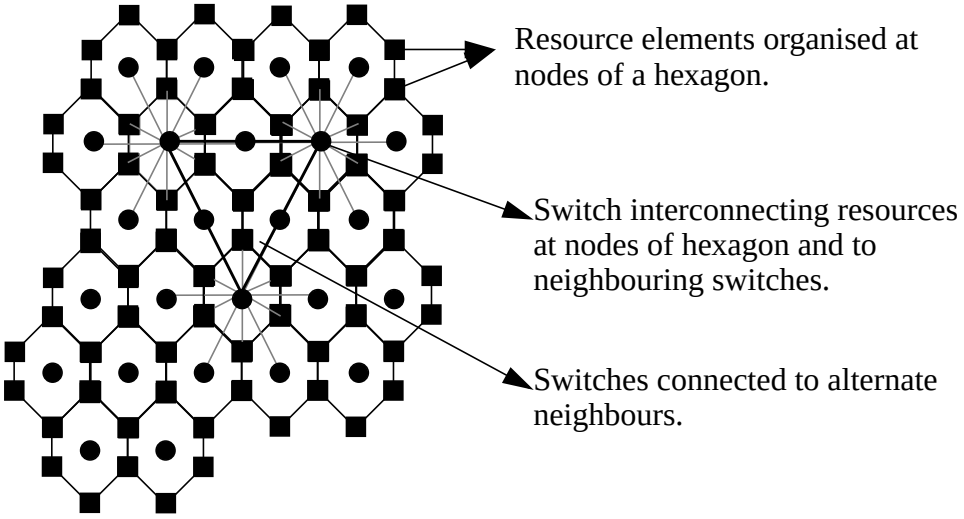

Figure 1. A honeycomb structure for NOC.

5. Methodology

As important as a good architecture is a good methodology and supporting tools, which translates a given system specification into a working system implemented on a NOC platform. In the following we review the major tasks and challanges that must be addressed.

As input we assume a requirements definition, which is an enumeration of the system's purpose, main system functions, the context and environment, and non-functional requirements and constraints. This document is not executable and in a first step we have to derive an executable functional system specification. We must be able to systematically evaluate the functionality in order to unambiguously specify what exactly we want the system to do. Simulation will be the most important tool to do that but in addition formal techniques would be more than welcome to establish certain liveness or safety properties.

In the next step a task graph of concurrent tasks has to be developed. Since the implementation will consist of many concurrent activities and a very parallel architecture, this step is the first and perhaps the most important one towards the implementation. Even though the tasks are not yet bound to specific resources and the final mapping task graph onto the architecture will be a many-to-many mapping, this step basically determines the granularity and the atomic units that eventually will be implemented on specific resources. Moreover, in this step we determine the major communication requirements between tasks.

Once we have the task graph we must map the tasks onto the architecture. This step faces many boundary constraints and even many of its sub-problems are very hard optimization problems. Some tasks will fit better on certain kind of resources than others. So we have to try bind a task to the resource that is best suited to implement it. A data processing kind of tasks will fit best onto a DSP or FPGA, depending on what performance requirements it has, what kind of basic operations it requires and how big it is. Control dominated task will fit on micro processors or FPGAs, depending on if the majority of the operations is at the bit level or the word level. At the same time we have to take the communication requirements between tasks and the memory access of the tasks into account. Tasks which exchange a large amount of data should sit physically close to each other with a high bandwidth interconnection between them. Tasks which need a lot of memory should have access to large storage devices with a reasonable access time. To guarantee the needed communication bandwidth is a particularly tough problem because the interconnect resources are flexible and will be shared among many tasks. Since it is very hard to predict the communication pattern of individual tasks, to estimate the communication pattern of a large numberof interacting tasks seems to be formidable.

When each task has been bound to a specific resource, the code must be generated. If several concurrent tasks are mapped onto the same processor, we have to consider a scheduling policy and some kind of operating system. For FPGA type of resources a traditional hardware design flow will be followed with Verilog or VHDL most likely as intermediate languages. For RISC like micro processors code generation is quite efficient, but for DSP processors we will need better compilers. We must also implement the communication between the tasks and the interfaces in the software and in the FPGAs. Finally, we must generate the code to program the I/O devices. Since the platform has a fixed set of components, it would be possible to have a library of interface components to use for communication refinement, thus considerably easing the task of interface design compared to custom interface synthesis.

All these steps are dependent on each other and for a truly optimal solution have to be considered together. Hence, in practice we will see iterations between these phases. Even more important is the validation of all steps with respect to functionality, performance, power, cost, and noise. To do this efficiently, we must deploy estimation techniques to predict performance figures before we make a design decision, and we must validate the performance after we have finished a design step. It should be emphasized, that both the functionala validation and the estimation of performance and other non-functional properties is greatly alleviated for a purly programmable platform as opposed to a design with custom hardware parts. The reason for this is that the potential design space is significantly reduced.

The methodology for Honey Comb Architecture will also require a specific configurable distributed operating system to manage network resources. Important features of this O.S. will be a capability to configure itself on the available network resources, a network of communicating schedulers for taking care of real time constraints will also be a distinctive feature of the O.S. for Honey Comb architecture.

In summary we can conclude, that the first phases, i.e. the functional specification and task graph development, are architecture independent, but will gain importance in the near future because the size of systems increase in general. The later phases, i.e. mapping, estimation, and validation have to be honeycomb specific but will be in general benefit from a fixed, purely programmable platform like the honycomb, because it resembles the introduction of an abstraction level and separates all design problems into two independent sub-problems: (1) the design, validation and optimization of the platform, and (2) the design, validation and optimization of the application on the platform.

6. Discussion and Analysis

Having presented the overall scheme, let us discuss the NOC's area and performance overhead compared to traditional architecture based on fixed interconnectivity.

Area overhead. This is hard to quantify, especially at this stage when the NOC is a concept. Moreover, NOC being a reprogrammable architecture like FPGAs, different designs will utilise the resources to a different extent. A more relevant question in this context would be the overhead of the switch based interconnect scheme. At this stage, we are not in a position to answer this question concretely. But given the fact that the proposed interconnect scheme allows any resource to reach any other resource, an overhead, we believe, is justified.

Performance overhead. A more concrete answer than the area overhead can be given here. Interconnect delay already dominate the performance equation and Luca Carloni et. al. (see [2]), have proposed a delay insensitive design, which essentially involves pipelining the wires and adding logic to the functional/computational blocks to implement the pipeline control.

According to ITRP99 by the time we reach .1 micron, the delay in long wires would be 100x compared to the fastest switching gate. And with clocks ticking at 10 GHZ, the interconnect delay for long wires would be 10s of cycles.

Taking these facts into account, we believe that switch based interconnect scheme will provide natural pipelining with the added benefit of interconnecting all resources.

Switches being control and memory intensive, the depth of logic in them would be significantly lower compared to that in the computational blocks. This observation leads to three conclusions.

- The first is, if the delay in wire is going to be 100x compared to a gate and if switches imply a small depth of gates, the switches would add an insifnificant amount of delay.
- Secondly, if as proposed by Luca Carloni et. al. (see [2]), delay insensitive design style involving pipelining the wire, is essential for present and future technology, the overhead of switching is comprable to that of the overhead of delay insensitive design style.
- Lastly, maintaining global synchronous assumption is already proving to be difficult and designs involving multiple clock domains are becoming common(need to refer to our and other GALS work). Again based on the fact that the computational blocks would have much greater depth of logic compared to switches, we can think of clocking the switches at a much higher clock compared to computational blocks. In such a case, the latency introduced by the switches, as seen by the computational blocks would be attenuated by the ratio of the switch clock to computation clock.

Fault tolerance. According to ITRP99(see [1]), below 100 nm, soft errors are predicted to be frequent enough to severely impact both semiconductor yields and field level product reliabili-

ty. Having, a switch based interconnectivity and a regular strucutre can significantly alleviate this problem.

7. Example

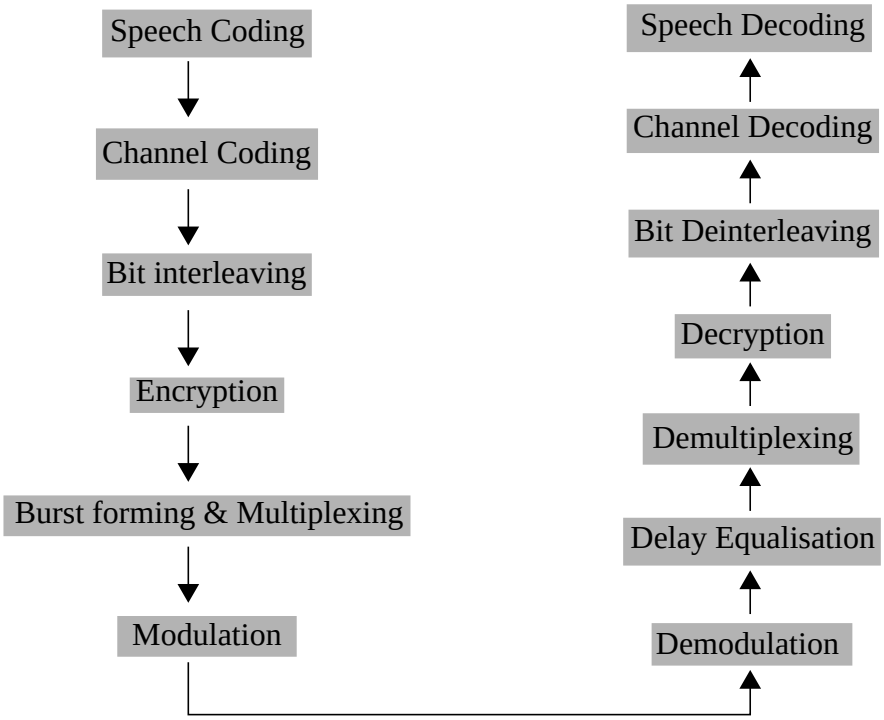

Figure 2. GSM receive and transmit base band processing

To illustrate the feasibility let us consider mapping a GSM bases band processing logic to a hypothetical Network on a chip. Figure 2 shows data flow for base band processing in GSM for both receive and transmit side. Using the next generation platform based GSM base station architecture of Ericsson as the reference, one can say that to handle base band processing, as shown in Figure 2, for one time slot that contains 8 channels requires about 10 million gates. These gates are divided over 22 DSP processor cores, a general purpose embedded processor, relatively small amount of hardware logic, 15 Kilobits of SRAM and 32 Mbyte of DRAM, the latter being common to the whole basestation. The largest basestation can handle 12 timeslots. A billion transistor ASIC can easily integrate the entire base station on a chip with good margin. Performance should not be a problem either, because, even today much of the DSP processing is implemented as software and as technology shrinks the performance gain should scale well with increased computational need for broad band. The Ericsson basestation architecture being proprietary and confidential, we show the feasibility by proposing a honeycomb structure, shown in Figure 3, that could be used to implement a single time slot and this could then be stacked to implement additional time slots. The honeycomb in Figure 3 has 22 computational resources that corresponds to the 22 DSPs used in the Ericsson's architecture to implement the functionality shown in Figure 2 for a single time slot plus some other control and management functionality. If we assume that each storage resource has about 8 kbit SRAM, we have a total of 44 kbit SRAM available, enough storage to accomodate the needs of the Ericsson architecure. As shown in Figure 2, the data flow is linear, thus the resources would not need many hops to connect each other. Some control and mangement functionality, not shown in Figure 2, does

need more inter resource connection that what Figure 2 suggests, but these signal paths are not time critical.

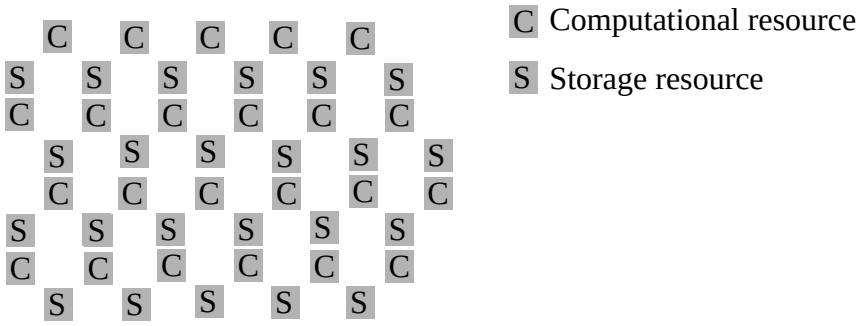

Figure 3. A Honeycomb structure to implement GSM base band processing for one time slot

8. Conclusion & Future work

We have presented what we think would be the right platform and methodology for the SOC design in the billion transistor era. This platform is based on a reconfigurable network of resources: computational, storage and I/O. One possible topology for such a platform, the honeycomb structure, has been proposed and its merits analysed. We have also presented a plausible way of mapping an entire basestaion to such a Network on a chip concept.

Though, we have presnted one possible topology, there are many others, for instance one can consider a complement of the presented honeycomb scheme. The resources can be at the center of hexagon and the switches can be at nodes of hexagon, creating an interconnect rich topology. Another possibility is to go for a mesh of squares, where the nodes of the squares are populated by resources and switches are at the center.

Much of the discussion is based on intuitive arguments backed back-of-the-envelope type calculation. We are working on developing a concrete platform and methodology. A realistic example like base station would be the driver.

9. References

1. *ITRP99: The International Technology Roadmap For Semiconductors: 1999.*
2. Luca P. Carloni, Kenneth L. McMillan, Alexander Saldanha, Alberto L. Sangiovanni-Vincentelli. *A Methodology for Correct-by-Construction Latency Insensitive Design.* Proceedings of ICCAD 1999. San Jose, USA.
3. *CS2000: Reconfigurable Communications Processor.* Family Product Brief. Chamelion Systems Inc. Santa Clara, USA.
4. *Xilinx, IBM ink cores agreement*. By Craig Matsumoto EE Times (07/26/00, 11:23 a.m. EST)
5. *Altera nabs 32-bit MPU cores from Tensilica, ARC* By Michael Santarini EE Times (10/11/99, 10:04 a.m. EST) .
6. Ahmed Hemani, Thomas Meincke, Shashi Kumar, Peeter Ellervee, Johnny Oberg, Thomas Olsson, Peter Nilsson, Dan Lindqvist. *Lowering power consumption in clock by using Globally Asynchronous, Locally Synchronous Design Style*., Proceedings of DAC '99, June 21-25, 1999, New Orleans, USA.
7. A. Hemani, B. Svantesson, P. Ellervee, A. Postula, J. Öberg, A. Jantsch, H: Tenhunen, "*High-Level Synthesis of Control and Memory Intensive Communication Systems*" ASIC conference and Exhibit, Austin, Texas, Sept. 18-22, 1995. pp: 185-191