Design Space Exploration of a Mesochronous Link for Cost-Effective and Flexible GALS NOCs

Daniele Ludovici[§], Alessandro Strano[†], Georgi N. Gaydadjiev[§], Luca Benini[††], Davide Bertozzi[†]
[†] ENDIF, University of Ferrara, 44100 Ferrara, Italy.
[††] DEIS, University of Bologna, 40136 Bologna, Italy.
[§] Computer Engineering Lab., Delft University of Technology, The Netherlands.

Abstract

There is today little doubt on the fact that a high-performance and cost-effective Network-on-Chip can only be designed in 45nm and beyond under a relaxed synchronization assumption. In this direction, this paper focuses on a GALS system where the NoC and its end-nodes have independent clocks (unrelated in frequency and phase) and are synchronized via dual-clock FIFOs at network interfaces. Within the network, we assume mesochronous synchronization implemented with hierarchical clock tree distribution. This paper contributes two essential components of any practical design automation support for network instantiation in the target system. On one hand, it introduces a switch design which greatly reduces the overhead for mesochronous synchronization and can be adapted to meet different layout constraints. On the other hand, the paper illustrates a design space exploration framework of mesochronous links that can direct the selection of synchronization options on a port-by-port basis for all the switches in the NoC, based on timing and layout constraints. A final case study illustrates how a cost-effective GALS NoC can be assembled, placed and routed by exploiting the flexibility of the architecture and the outcomes of the exploration framework, thus proving the viability and effectiveness of the design platform.

1. Introduction

Network-on-chip communication architectures are being widely adopted in System-on-Chip design to ensure scalability and facilitate a component-based approach to large-scale heterogeneous system integration. As technology advances into aggressive nanometer-level scaling, a number of design challenges emerge from technology constraints which require a continuous evolution of NoC implementation strategies at the circuit and architectural level. Synchronization is today definitely among the most critical challenges in the design of a global on-chip communication infrastructure, as emerging variability, signal integrity, power dissipation limits are contributing to a severe break-down of the global synchronicity assumption when a logical structures spans more than a couple of millimeters on die [23]. NoCs typically span the entire chip area and there is now little doubt on the fact that a high-performance and cost-effective NoC can only be designed in 45nm and beyond under a relaxed synchronization assumption [24].

A number of NoC research proposals have tackled the synchronization challenge in recent years, leading to a rich research field which could be coarsely clustered in purely asynchronous vs. globally-asynchronous, locally synchronous (GALS) approaches. This work can be classified in the latter group. More precisely, we assume that the NoC and its end-nodes have independent clocks (unrelated in frequency and phase) and are synchronized via dual-clock FIFOs at the network interface. Within the network, we assume mesochronous synchronization implemented with hierarchical clock tree synthesis [2]. This mixed asynchronous/mesochronous approach is considered very promising as it limits the performance and area overheads associated with purely asynchronous implementations and, more importantly, is fully compatible with the current generation of design implementation tools.

Our contribution is two-fold. First we introduce a new switch design which greatly reduces the overhead in supporting mesochronous synchronization. In a nutshell, we merge the input buffering logic with the mesochronous synchronizer, thereby achieving major savings thanks to the sharing of expensive buffers, at the expenses of a reduction of the link delay budget and phase shift margin. Our design is still fully synthesizable and does not even require ad-hoc library support (e.g. synchronizer cells). Moreover, we can choose on a port-by-port basis if the mesochronous synchronizer will be merged with the input port or be simply added immediately outside the switch. This additional degree of freedom enables us to select the external synchronizer when two switches are connected with a very long link which would not be compatible with the delay and phase constraints of the merged synchronizer.

Our second contribution is a design space exploration of mesochronous NoC links providing selection guidelines of synchronization options on a port-by-port basis for all the switches in the NoC. As a result, target timing and phase margins can be reliably enforced in the NoC while adapting to the layout constraints at hand. Our exploration framework also captures the implications of synthesis directives on the skew tolerance of mesochronous links, so that performance optimization of the NoC design can be performed without loosing control on key synchronization parameters. Our exploration framework is an essential component of any practical design automation support for network instantiation in the target GALS system aiming at a cost-effective implementation and at less design respins. It would be extremely labor-intensive for a designer to instantiate an entire network manually while checking design constraints which determine switch instantiation options. Above all, it would be difficult for him to capture the implications of his design choices across a number of design parameters and layers.

Our network architecture and design flow is finally tested on a complete (from architecture to fully placed and routed layout) NoC design case study, demonstrating the superior area properties of GALS NoCs designed with our platform and the area-performance-skew tolerance trade-off spanned by the possible architecture variants.

2. Related Work

Many systems deploying concepts of the GALS philosophy can be found in literature.

In [25] a many-core heterogeneous computational platform employing GALS compatible circuit switching on-chip network has been presented. [18] presents an example of mesh-connected GALS chip multiprocessor. The work shows that the typical performance penalties of GALS systems (mainly due to additional communication latency) can be hidden by using large FIFO buffers. In [2], the physical implementation of the DSPIN network-on-chip in the FAUST architecture has been presented. In [19] a cost effective solution for asynchronous delay-insensitive on-chip communication is proposed. The solution is based on the Berger coding scheme and allows to obtain a very low wire overhead. [20] proposes a new

978-3-9810801-6-2/DATE10 © 2010 EDAA

Asynchronous Network-On-Chip (NOC) architecture aiming at low latency transfers.

At interfaces level, many GALS wrappers have been proposed in [21], [22], [16], [14], [15] to provide fast and reliable asynchronous communication services.

GALS modularity has been used also along with the concept of voltage-frequency islands (VFIs) which has been recently introduced for achieving fine-grain system-level power management. [13] proposes a design methodology for partitioning a NoC architecture into multiple VFIs. [11] and [12] proposes a complete dynamic voltage and frequency scaling architecture for IP units integration within a GALS NoC.

Examples at industrial level are presented in [10], [9], [17], [5]. In [9] authors examine the use of GALS techniques to address on-chip communication between different synchronous modules on a bus. Issues related to validation, module interfaces and tool flows, while looking at advantages in power savings, timing closure and Time-to-Market/Time-to-Money (TTM) are explored. [17], [5] both suggest to implement the boundary interface with a source-synchronous design style and propose some form of ping-pong buffering to counter timing and metastability concerns.

In this context, we advocate for tight integration of synchronization interfaces into NoC building blocks to cut down on latency, area and power. The feasibility of this challenging yet promising approach to NOC design has been first explored in our previous work in [3]. That work describes architecture design principles of a tightly coupled synchronization architecture and provides switch-level insights on area/power savings. This paper significantly extends that work. First, it provides a wider range of architecture alternatives and even port-level configuration capability (flexibility). Above all, it identifies the distinctive design constraints of the new schemes and performs a design space exploration of mesochronous NoC links and switches to capture how timing margins can be preserved for different combinations of synthesis and layout constraints. Finally, we provide a system-level demonstration of the cost-effectiveness of the proposed GALS NoC.

3. Target GALS architecture

The focus of this work is on the building process of GALS systems, where processing blocks can be separated from each other such that each of them can be clocked by an independent clock domain. This approach is a promising strategy to address global clock design challenges. There are a number of implementation solutions for GALS systems. One method consists of asynchronous clockless handshaking, which uses multiple phases of signal exchange to transfer data. Due to the round-trip signal exchange, the transfer latency between two consecutive data words is high. Besides that, the asynchronous clockless circuits are difficult to verify in traditional CAD flows, and they also demand a comparatively large area and energy requirements [1, 21].

The one addressed in this work consists of implementing the on-chip network as an independent clock domain, and therefore to place circuitry to reliably and efficiently move data across asynchronous clock boundaries between NoC switches and connected network interfaces. These latter are assumed to be part of the clock domain of the IP core that they serve. Dual-clock FIFOs are an effective solution to provide asynchronous boundary communication, especially in throughput-critical interfaces. However, many designers are skeptical about their utilization due to the relevant latency, area and power overhead they incur. Beyond urging research activities aiming at the optimization of dual-clock FIFO architectures, this fact emphasizes the need for their conscious use in GALS systems.

Aware of this, we try to minimize their usage as much as possible by instantiating them only at IP core boundaries, after their respective network interfaces. This choice however moves many chip-wide timing concerns to the on-chip network. In fact, this latter ends up spanning the entire chip and might be difficult to clock due to the growing chip sizes, clock rates, wire delays and parameter variations. There is little doubt on the fact that a high-performance and cost-effective Network-on-Chip can only be designed in 45nm and beyond under a relaxed synchronization assumption.

We find that mesochronous synchronization can relieve the burden of chip-wide clock tree distribution while requiring simpler and more compact synchronization interfaces than dual-clock FIFOs. Hierarchical clock tree synthesis is an effective way of exploiting mesochronous links, as already experimented in [2]. During the first step, a clock tree is synthesized for each network switch with a tightly controlled skew (e.g.,5%). Then, each clock tree is characterized with its input delay, skew and input capacitance. This information is used by the clock tree synthesis (CTS) tool to infer a top clock tree balancing the leaves with a much looser skew constraint (e.g., 30/40%). The ultimate result is a global clock tree which consumes less power then the traditional one generated by enforcing chip-wide skew constraints. For future large multiprocessor systems-on-chip, the use of this methodology can be not just an issue of power efficiency but even of CTS feasibility.

However, power savings with this methodology should not be taken for granted, since it involves some overheads: the transmission of the clock signal across mesochronous links, the mesochronous synchronizers themselves (implementing power-hungry buffering resources) and the increased number of buffer slots needed at link end-nodes to cover the larger round-trip time (associated with the synchronization latency) for correct flow control management.

Our design platform aims at minimizing such overheads through a novel mesochronous architecture taking advantage of the tight integration of the synchronizer into the NoC architecture. Since however these solutions give rise to timing constraints that might not be verified for specific layout conditions, we provide architecture variants for these cases as well, thus coming up with a flexible NoC suitable for many design instances.

3.1 Baseline Switch Architecture

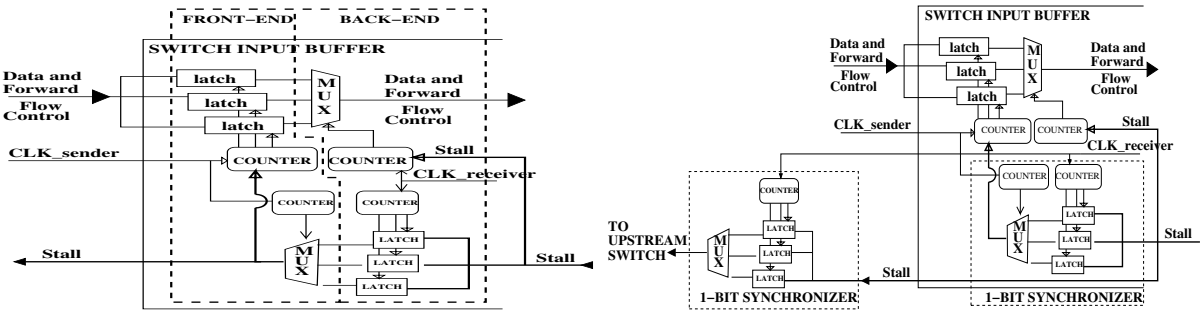
The basic building block of the proposed GALS NoC is the baseline switch architecture of the xpipesLite library [8]. It leverages wormhole switching and source-based routing, with shifting of routing bits in packet headers at each hop. It is an output buffered switch which however implements a retiming and flow control stage also at the input ports. Arbitration is not centralized, i.e., there is a modular round-robin arbiter for each output port serializing conflicting access requests from all input ports. The critical path starts from the switch input buffer, goes through the arbiter, the crossbar selection signals, some combinational logic and finally includes a library setup time for correct sampling at the switch output port.

This switch uses a stall/go flow control protocol [6]. It requires two control wires: one going forward and flagging data availability ("valid") and one going backward and signaling either a condition of buffer filled ("stall") or of buffer free ("go"). The sender only needs two buffer slots to cope with stalls. With this scheme, power is minimized since any congestion issue simply results in no unneeded transitions over the data wires. Moreover, recovery from congestion is instantaneous. In the fully synchronous variant of the switch architecture, the input buffer is used to break the timing path going across the upstream link. It has to manage flow control as well, therefore it must consist of 2 slots.

The basic scheme for mesochronous synchronization consists of placing a mesochronous synchronizer in front of the switch input port (see e.g. [7]). We call this approach *Loosely Coupled Synchronizer*. This way, one more hop is introduced across the link. It should not be ignored that also the backward propagating flow control signal (the stall) should be synchronized before reaching the upstream switch, therefore a 1-bit synchronizer should be placed on this link (in front of the upstream switch) as well, thus further increasing the round-trip latency. By using the simple FIFO-based synchronizers presented in [3, 5], the one-way latency would be 2-cycles, therefore a stall condition would not be notified immediately to the sender and the flits in flight while the receiver is busy might simply get lost. We found that only enhancing the upstream switch input buffer from 2 to 4 slots enables lossless full-bandwidth operation, which leads to a significant 40% power overhead for a 2x2 switch of the xpipesLite architecture [3].

3.2 Tightly Coupled Synchronizer

We found in [3] that the key principle for reducing the latency, area and power overhead of switches with loosely coupled synchronizers is to co-design the synchronizer with the downstream switch input port. The co-optimization consists of using the buffering resources of the switch input stage not only for retiming and flow control as in the synchronous architecture, but also for synchronization purposes, thus practically merging this stage with the synchronizer.



(a) Tightly Coupled Mesochronous Synchronizer. (b) Hybrid architecture

Figure 1. Variants for tight integration of mesochronous synchronizers into the switch architecture.

The basic architecture of a new switch input port is illustrated is Fig.1(a) and denoted *Tightly Coupled Synchronizer*.

The switch port interface is source synchronous: it receives from the upstream switch a regular NoC link (carrying data and forward flow control) and a copy of the transmitter clock signal. Since the latter wire should be routed so to experience the same routing delay as the link, it can be used as a strobe signal for data.

The circuit is composed by a front-end and a back-end. The front-end is driven by the sender clock signal, and strobes the incoming data and flow control wires onto a set of parallel latch banks in a rotating fashion, based on a counter. The back-end of the circuit leverages the local clock, and through a multiplexing logic driven by a counter it selectively forwards stable data from latch outputs to the switch datapath and stable valid signals to the switch arbiter. The rationale is to temporarily store incoming information in one of the front-end latches, using the incoming clock wire to avoid any timing problem related to the clock phase offset. Once the information stored in the latch is stable, it can be used in the target clock domain. The ultimate sampling of this data will occur at the switch output port.

Counters in the front-end and back-end are initialized upon reset. 3 latch banks in the front-end of the synchronizer suffice for short-range (single cycle) mesochronous communication. They allow to keep latched data stable for a time window long enough to enable a unique and safe bootstrap configuration of the counters that proves robust in any phase skew scenario (from -360° to +360°). This avoids the need for a costly phase detector.

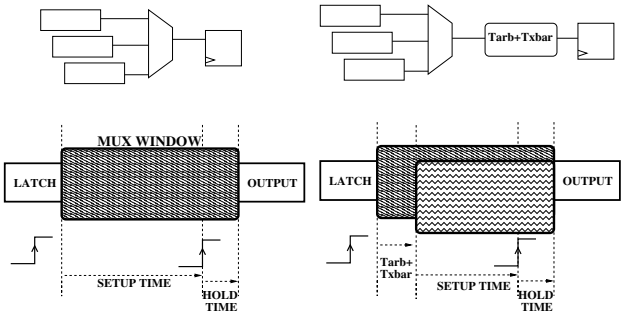
As regards latency, while the fully synchronous switch takes 1 cycle in the upstream link and 1 cycle in the switch itself, the novel tightly integrated switch takes from 1 to 3 clock cycles to cross the same path, depending on the negative or positive skew. Interestingly, the link is always crossed in 1 cycle, therefore there is no need of increasing the buffer size of the switch input port since the round-trip latency is unaffected.

Our architecture adopts a special solution for backward flow control. In fact, since the latching elements of the synchronizer also act as the switch input buffer, the stall/go signal needs to be brought to both front-end and back-end counters in order to freeze synchronizer operation when a stall is asserted. However, while this signal is already in synch with the back-end counter, it should be synchronized with the transmitter clock before driving the front-end counter. This is done in the lower part of Fig.1(a) by synchronizing the stall/go signal with the transmitter clock already available at the switch front-end. The output of this simple 1-bit synchronizer can then be directly sent to the upstream switch where it will be immediately sampled since it is already in synch with the transmitter clock. This scheme leads to a compact and modular implementation without any external synchronizer between NoC building blocks and has therefore to be preferred whenever possible.

3.3 Architecture flexibility

Unfortunately, the way the tightly coupled architecture synchronizes the stall signal incurs a severe constraint on the round trip time. In fact, the transmitter clock used for stall synchronization at the downstream switch has already undergone a link delay T_{link}, and the stall signal itself takes another link delay to go back to the upstream switch:

$$2 \cdot T_{link} + T_{counter} + T_{mux} + T_{setup} \leq T_{clock} \qquad (1)$$



(a) Loosely coupled synch. (b) Tightly coupled synch.

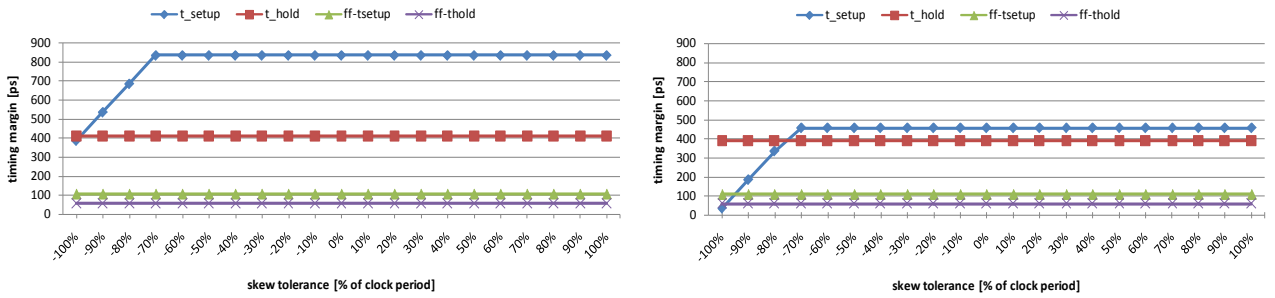
Figure 2. Basic mechanisms affecting skew tolerance.

Whenever this constraint cannot be met, then our architecture proves flexible enough to provide an alternative solution which meets the constraints at a small cost in area and modularity. The solution, denoted as *Hybrid*, is illustrated in Fig.1(b). In practice, the simple 1-bit synchronizer can be replicated in front of the upstream switch, thereby breaking the round-trip dependency of the timing. The hybrid architecture now exposes a new timing path going from the switch arbiter to the upstream switch through the stall signal, which is a one-way path, not a round-trip one. Implications of this path on global system timing will be highlighted in section 5.

Another degree of flexibility of our architecture is that a switch can be assembled out of a mix of synchronous and mesochronous ports. In fact, the output architecture of the tightly coupled synchronizer resembles that of a synchronous switch input buffer, therefore for the switch datapath and control logic it is irrelevant whether the input port is synchronous or mesochronous. A flexible heterogeneous switch architecture can therefore be built, where input ports are either the conventional 2-slot buffer of synchronous switches or the tightly coupled synchronizer. Finally, an external mesochronous synchronizer can also be instantiated in front of the synchronous switch input ports to infer, whenever needed, the loosely coupled synchronization architecture.

4. Design Space Exploration of the Mesochronous Link Architecture

The above architecture provides degrees of freedom for port-level selection of the most suitable synchronization option based on timing and layout constraints. The following design space exploration of a mesochronous link implemented with our architecture will provide the guidelines for such port-level selection, and is therefore an essential enabler for automatic assembly of the target GALS NoC.



(a) Loose Coupling. (b) Tight Coupling.

Figure 3. Setup and hold times for the loosely vs. tightly coupled architectures as a function of clock phase offset.

For the loosely coupled architecture, an external synchronizer will be considered that features the same architecture of the upper part of Fig.1(a). Its multiplexer output directly feeds the input port of a synchronous switch for correct sampling of synchronized data. Also, an external 1-bit synchronizer is needed in the backward flow control link.

4.1 Skew Tolerance

Skew tolerance of our architecture schemes depends on the relative alignment of data arrival time at latch outputs, multiplexer selection window and sampling edge in the receiver clock domain. A few basic definitions help to assess the interaction among these parameters in determining skew tolerance. For the loosely coupled synchronizer, such definitions are pictorially illustrated in Fig.2(a).

During the *mux window*, data at latch outputs is selected for forwarding to the sampling flip flop in the switch input port. Its duration closely follows that of the clock period. Sampling occurs *on the next rising edge of the receiver clock inside the mux window*. We denote the time between the starting point of the mux window and such sampling instant as the *Setup time*. Conversely, after an *Hold time* since the rising edge of the clock the mux window terminates. This is the time required by the back-end counter to switch the multiplexer selection signals.

When we consider the tightly coupled architecture (Fig.2(b)), then the same metrics are taken at the switch output port rather than at the multiplexer output of the synchronizer. Therefore, the starting time of the *mux window* is delayed due to the worst case timing path between the synchronizer output and the switch output port, which includes the arbitration time, crossbar selection time and some more combinational logic delay for header processing. At the same time, the sampling rising edge of the receiver clock remains unaltered, therefore the ultimate effect is a shortening of the *Setup time* for the tightly integrated mesochronous switch architecture.

Fig.3(a) quantifies these timing margins for the loosely coupled switch architecture. Results are referred to a 2x2 switch working at 660 MHz after place&route. x-axis reports negative and positive values of the skew, expressed as percentage of the clock period. Setup and hold times have been experimentally measured by driving the switch under test with a clocked testbench, by inducing phase offset with the switch clock and by monitoring waveforms at the switch. The connecting link between the testbench and the switch is assumed to have zero delay. A positive skew means that the clock at the switch is delayed with respect to the one at the testbench. The figure also compares setup and hold times with the minimum values required by the technology library for correct sampling (denoted *Lib_setup* and *Lib_hold*).

First of all, we observe that both times are well above the library constraints, thus creating some margin against variability. For the whole range of the skew, the hold time stays the same. The result is relevant for positive skew, since its effect is to shift the *mux window* to the right, close to the region where latch output data switches. However, the stability window of the latch output data is long enough to always enable correct sampling of stable data before the point in time where it switches.

In contrast, a negative skew causes the *mux window* to shift to the left, therefore as the negative skew grows (in absolute values) the latch output data ends up switching inside the *mux window*, which corresponds to the knee of the setup time in Fig.3(a). From there on, the switching transient of data becomes closer to the sampling edge of the receiver clock and correct sampling can be guaranteed until the setup time curve equals the *Lib_setup* one. However, even for -100% skew synchronizer operation is correct.

Fig.3(b) illustrates the same results for the tightly coupled mesochronous switch. As anticipated above, the setup time is decreased by 370ps, corresponding to the time for arbitration, crossbar selection and shifting of routing bits. Interestingly, the knee of the setup time occurs for the same value of the negative skew, in that the switching instant of the latch output data enters the *mux window* at exactly the same point in time. The ultimate implication is that the tightly coupled synchronizer cannot work properly with -100% skew, since the crossing point with the *Lib_setup* occurs at around -95%. In practice, we can conclude that a 2x2 switch with tight coupling of the synchronizer on each port consumes 40% less area and power than its loosely coupled counterpart while incurring a 23% degradation of the maximum skew tolerance.

4.2 Interconnect delay

Skew tolerance is usually assessed by measuring the phase offset between the clock at the synchronizer front-end and the one at its back-end. However, the front-end clock is the one of the transmitter (i.e., the upstream switch) which has been sent across the connecting link together with data. Unfortunately, in the hierarchical clock-tree synthesis process of GALS systems it is not possible to precisely control the exact phase of such a front-end clock, but rather that of the clock at the upstream switch [2]. In fact, the synthesis step of the top clock tree in the hierarchical methodology enforces a maximum skew between leaves in different switches. However, this is not the actual phase offset measured between the front-end and the back-end of the synchronizer, which differs for the interconnect delay and even for other contributions such as the clock propagation time and the combinational logic in the output buffer of the upstream switch.

This deviation of the enforced clock phase offset by the actually measured one should be taken into close account when designing the skew tolerance of the synchronizer. The following analysis aims at quantifying this effect.

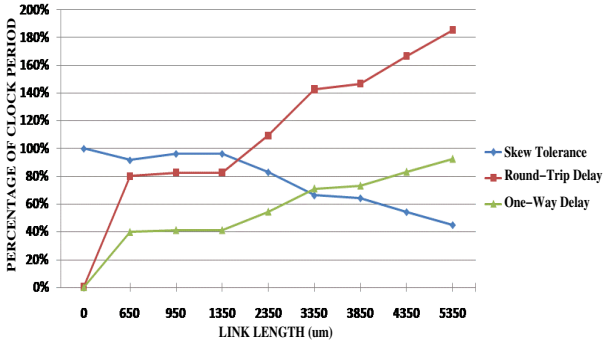


Figure 4. Effect of link length on skew tolerance and feasibility of the tightly coupled synchronizer.

The impact of the interconnect delay on the setup time curves in Fig.3(a) and Fig.3(b) is to shift them to the right. In fact, such a delay implies that new data appears later at the output of synchronizer latches. As a consequence, the knee of the setup time will occur earlier, when moving from a condition of null skew towards more negative skews, hence the maximum tolerated negative skew will be reduced as well. How fast the curve moves to the right and negative skew tolerance is degraded depends on the link length, on the target frequency and on how effectively the physical synthesis tool enforces the timing constraint on that specific link.

Fig.4 shows how skew tolerance is impacted by the length of the upstream link in the *tightly coupled architecture*. It is derived from a post-layout analysis of two connected 2x2 switches synthesized at a target speed of 660 MHz. Skew tolerance of the downstream switch is analyzed. It should be observed that by playing with the input/output delay constraints for switches during the synthesis process, we were able to demand 20% more performance to the switch-to-switch wires to conservatively meet the constraint in equation 1.

Each point represents the percentage of negative skew at which the setup time curve intersects the *Lib_setup* one, i.e., it shows how the maximum tolerable negative skew degrades with link length. Not only interconnect delay, but also clock propagation time and a small combinational logic delay at the output of the upstream switch contribute to degrade negative skew robustness of the downstream switch. The irregular decay of the skew tolerance curve depends on the interconnect delay enforced by the physical synthesis tool for each value of the link length. Skew tolerance remains always above 50% for links of less than 5mm.

However, correct operation of the tightly coupled synchronizer depends not only on skew tolerance, but also on the satisfaction of the constraint in equation 1. The timing path indicated by that equation is displayed in Fig.4 as *Round-Trip Delay*. The intersection of this curve with a $yvalue$ of 100% indicates that from there on equation 1 is violated, and the stall signal cannot reach the upstream switch within one clock cycle. In practice, a link length of 2 mm is the *feasibility range* for the tightly coupled synchronization architecture in this experimental setting. For longer links, our architecture however provides the hybrid solution as an additional degree of freedom, which implies a looser timing constraints to be met, also illustrated in Fig.4. This solution leaves the skew tolerance of the tightly coupled synchronizer unaffected but extends the feasibility range to around 5.5mm at the small area cost of an additional 1-bit synchronizer.

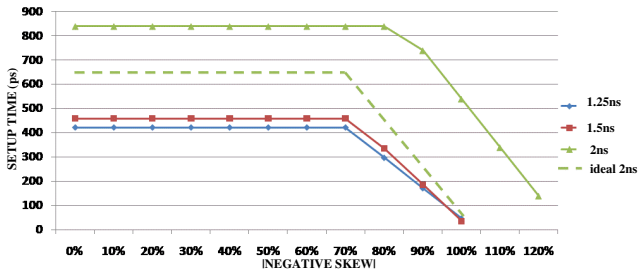


Figure 5. Setup time as a function of negative skew.

4.3 Target frequency

We now extend the above results to the case where the same RTL design (a 2x2 switch) is synthesized for a higher and lower target frequency and observe implications on the timing margins of a tightly coupled mesochronous NoC architecture. Fig.5 shows setup time as a function of negative skew for different target cycle times. The skew is expressed as percentage of the cycle time, and the assumption behind this plot is that as we relax the cycle time also the maximum skew constraint can be proportionally relaxed.

If we assume that by relaxing the target speed all delays scale proportionally in the design, then we would expect that by relaxing the speed from 1.5ns to 2ns the setup time increases by 1.33x (see dashed line). This is not the case, since the real setup time increases much more, as the figure shows. The reason for this is that the arbitration and switching logic inside the switch can be optimized for area as the timing constraint is relaxed only up to a certain point, beyond which no more netlist transformations are feasible. So, the shifting of the *mux window* to the right, illustrated in Fig.2(b) for the tightly coupled architectures, scales ideally only up to a certain point, beyond which we observe a more-than-linear increase of the setup time. This is what happens for the 2ns target period.

Another deviation of the ideal curve from the real one regards the knees. In fact, although the target period increases from 1.25 to 2ns, a given skew percentage on the x-axis actually means a different absolute phase offset for the different cases. Therefore, the switching instant of synchronizer latch outputs should enter the *mux window* at the same percentage skew for all target periods (see dashed line in Fig.5 for a 2ns target).

This is again not the case, indicating that a delay has not scaled proportionally to the clock period. The responsible for this is the time to generate the latch_enable signals in the synchronizer front-end. For a tight 1.25ns constraint, this net was already *non-critical*, therefore by relaxing the timing constraint its delay stays more or less the same. Therefore, the knee appears later for large cycle times, as the real curve for a 2ns target proves. The key take-away from this characterization is that by relaxing the target clock frequency for the same RTL design an improvement of the skew tolerance and of the timing margins can be generally achieved for the tightly coupled architecture. In addition, a larger cycle time provides the physical synthesis tool more margin to enforce the feasibility constraint of equation 1.

4.4 Switch radix

A last degree of freedom that we explored is the switch radix. We assume that for the same given target frequency, the switch radix is increased from 2 to 5. The effect on the timing margins is similar to what happens when we move from a loosely coupled to a tightly coupled mesochronous architecture. In fact, an increase of arbitration and crossbar selection time takes place, which results in a decrease of the setup time. Conversely, the knee occurs for the same amount of negative skew, since the modification concerns only the switch internal architecture, not the time at which latch outputs switch. Overall, by combining the two effects we have an additional reduction of the maximum (negative) skew tolerance, which is equal to the increase in delay of the combinational logic described above. In general, for high radix switches it has to be verified that the reduced setup time is still above the minimum value required by the technology library.

In contrast, synthesis constraints help relieve the above limitation. In fact, for both the 2x2 and the 5x5 switch, the synthesis tool tries to meet the same target cycle time and to exploit the available slack to save area. In practice, the syntheses of both the 2x2 and the 5x5 designs converge with almost no slack. Therefore, control logic with 5 and 2 inputs takes almost the same delay with a large difference in area. In this way, the setup time in the two cases is almost unaffected because of the netlist transformations performed by the synthesis tool in the area-performance plane. In our experiments, a 5x5 switch exhibits a setup time which is only 5% lower than the one in the 2x2 switch. For those NoC architectures where the above netlist optimizations are ineffective or for very high radix switches, it is necessary to verify that the reduced setup time is still above the minimum value required by the technology library.

Another implication of the switch radix concerns timing closure of the *hybrid* synchronization architecture. As already noted while commenting Fig.1(b), the critical path starts in the switch arbiter (which generates the *stall* signal) and includes the propagation of the *stall* signal to the upstream switch. As the switch radix increases, the delay for *stall* generation by the arbiter increases, and might make this timing path critical for the entire NoC. This timing path is compared with those of the tightly and loosely coupled architectures in section 5.

5. Experimental results

It has been demonstrated in [4] that as technology keeps scaling to sub-65nm technologies, it is more and more likely that the critical path moves from the switch internal microarchitecture to the switch–to–switch links due to the reverse scaling of on-chip interconnects. Therefore, it is essential to assess the pressure that each synchronization interfaces puts on the propagation delay of such links. Our first experiment assesses this property. In practice, we consider a 5x5 switch (ideally extracted from the center of a 2D

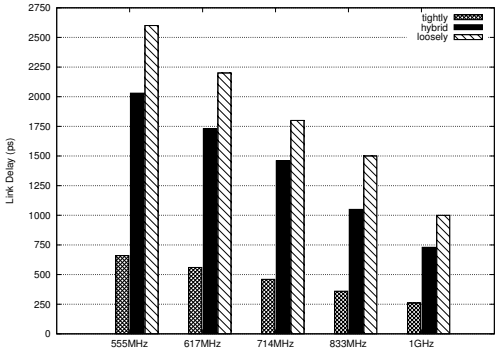


Figure 6. Post P&R operating frequency and relative tolerated link delay for different synchronization schemes.

mesh) after place&route. The switch has been synthesized with a very tight timing constraints (1 GHz), so that after place&route the critical path for all architectures will be certainly in the switch–to–switch link. The switch is connected to a tester injecting clock and data with an increasing delay. The utilized testbench assumes an ideal alignment between clock signal and data as well as no routing skew. This way, we can assess for a certain operating frequency, the relative link delay supported by each architecture. Obviously, a certain link delay corresponds to a relative channel length depending on how the link synthesis policy is chosen. Figure 6 reports various operating speeds of all the architectural solutions and the relative link delay they are able to tolerate. Given a certain working frequency, the tightly coupled architecture supports a smaller link delay with respect to the loosely coupled solution due to the more stringent round trip timing constraint discussed in Section 3.3. Indeed, an increment of either frequency or delay would result in a loss of packets due to the late arrival of the backward propagating signal. However, such supported link delays enable the tightly coupled architecture to sustain a correct communication within a typical range of link length in nanoscale technologies. In fact, our early exploration pointed out that a link delay of 550ps after place&route corresponds to 4mm channel by using a commercial 65nm technology library. Limitations suffered by the tightly coupled architecture can be removed by adopting a hybrid solution (Section 3.3). In fact, by using a small single-bit synchronizer at transmitter end for the backward signal, round trip dependency can be removed thus increasing maximum achievable frequency while keeping area cost as low as that of the tightly coupled architecture. Figure 6 proves that the hybrid architecture can tolerate a link delay similar to that of the loosely coupled solution but at a much lower area cost.

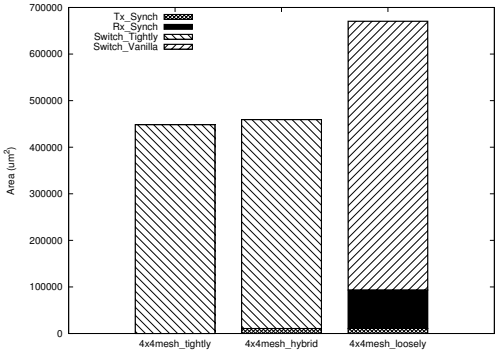


Figure 7. Area of 4x4 2D meshes.

We now prove the ability of our design platform to build a cost-effective GALS NoC. We designed a 4x4 2D mesh topology in the tightly coupled, loosely coupled and hybrid architecture variants. We aimed at the same target speed in order to assess system-level area savings with the tightly coupled approach. The three solutions have been synthesized at the same frequency, particularly, in order to carry out a fair comparison, synthesis have been aligned at the frequency of the slower scheme (tightly coupled). Therefore, loosely coupled and hybrid architectures have been synthesized with relaxed performance constraints with respect their maximum achievable speed. This results in a more area-efficient gate-level netlist. Switches were placed 1 mm apart from each other. We achieved timing closure of the three platforms at 600 MHz. Post-P&R area figures are compared in Fig.7. Clearly, the hybrid and the tightly coupled variants provide a 40% area saving over the loosely coupled one. This latter has to implement 2 more slots in the input buffer, which makes area of the vanilla switches larger than that of the tightly coupled ones. The hybrid architecture limits its area overhead with respect to the tightly coupled one because of the low footprint of 1-bit external synchronizers on the *stall* wires. In contrast, external synchronizers at receiver switches in the loosely coupled architecture consume about 12% of total mesh area.

6. Conclusions

This paper provides a contribution to the automated design and configuration of GALS NoCs and to their timing closure. On one hand, it proposes a compact mesochronous NoC architecture which can greatly reduce the area and latency overhead. On the other hand, it assesses timing margins and phase offset budget for the new solution, and provides an architecture variant for those cases where layout and timing constraints cannot be met. This paper does not just propose a new flexible mesochronous architecture, but explores its design space in the context of on-chip networks. Outcomes of this exploration can be used to direct the selection of synchronization options on a port-by-port basis for all the switches in the NoC. A final case study proves the cost-effectiveness of the NoCs obtained with the proposed design platform.

Acknowledgements

This work has been partially supported by the GALAXY European Project (FP7-ICT-214364), by the Hipeac Network of Excellence (Interconnect Cluster) and by the European Commission in the context of the Scalable computer ARChitectures (SARC) integrated project (FP6 #27648).

7. REFERENCES

[1] B. R. Quinton, M. R. Greenstreet, and S. J.E. Wilton, "Asynchronous IC Interconnect Network Design and Implementation Using a Standard ASIC", Proc. of Int. Conference of ComputerDesign (ICCD), pp.267–274, 2005.
[2] I.M.Panades, F.Clermidy, P.Vivet, A.Greiner, "Physical Implementation of the DSPIN Network-on-Chip in the FAUST Architecture", Int. Symp. on Networks-on-Chip, pp.139-148, 2008.
[3] D.Ludovici, A.Strano, D.Bertozzi, L.Benini, G.N.Gaydadjiev , "Comparing Tightly and Loosely Coupled Mesochronous Synchronizers in a NoC Switch Architecture", Proceedings of the 3rd ACM/IEEE International Symposium on Networks-on-Chip, pp.244-249, 2009.
[4] D.Ludovici, F.Gilabert, S.Medardoni, C.G.Requena, M.E.Gómez, P.López, D.Bertozzi, G.N.Gaydadjiev "Assessing Fat-Tree Topologies for Regular Network-on-Chip Design under Nanoscale Technology Constraints", Proc. of DATE, pp. 562-565, France 2009.
[5] F.Vitullo, N.E.L'Insalata, E.Petri, L.Fanucci, M.Casula, R.Locatelli, M.Coppola, "Low-Complexity Link Microarchitecture for Mesochronous Communication in Networks-on-Chip", IEEE Trans. on Computers, Vol.57, no.9, pp.1196–1201, 2008.
[6] A. Pullini, F. Angiolini, D. Bertozzi, L. Benini, "Fault Tolerance Overhead in Network-on-Chip Flow Control Schemes", Proceedings of 18th Annual Symposium on Integrated Circuits and System Design (SBCCI) 2005, Florianpolis, Brazil, Sep 4-7, 2005, pp. 224-229.
[7] S.Vangal et al.; "An 80-Tile Sub-100-W TeraFLOPS Processor in 65-nm CMOS", IEEE Journal of Solid-State Circuits, Vol.43, Issue 1, pp.29–41, 2008.
[8] S.Stergiou et al., "Xpipes Lite: a Synthesis Oriented Design Library for Networks on Chips", DAC, pp.559-564, 2005.
[9] A.M. Scott, M.E. Schuelein, M. Roncken, J. Hwan, J. Bainbridge, J.R. Mawer, D.L. Jackson, A. Bardsley, "Asynchronous on-Chip Communication: Explorations on the Intel PXA27x Processor Peripheral Bus", Proc. of the 13th IEEE Int. Symp. on Asynchronous Circuits and Systems, pp.60–72, 2007.
[10] M.B.Stensgaard, T.Bjerregaard, J.Sparso, J.H.Pedersen, "A Simple Clockless Network-on-Chip for a Commercial Audio DSP Chip", Proc. of the 9th EUROMICRO Conference on Digital System Design, pp.641–648, 2006.
[11] E.Beigne, F.Clermidy, S.Miermont, P.Vivet, "Dynamic Voltage and Frequency Scaling Architecture for Units Integration within a GALS NoC", Proc. of Int. Symp. on Networks-on-Chip, pp.129–138, 2008.
[12] E.Beigne, F.Clermidy, S.Miermont, Y.Thonnart, A.Valentian, P.Vivet, "A Localized Power Control mixing hopping and Super Cut-Off techniques within a GALS NoC", Int. Conf. on Integrated Circuit Design and Technology and Tutorial, pp.37–42, 2008.
[13] U.Y. Ogras, R.Marculescu, P.Choudhary, D.Marculescu, "Voltage-Frequency Island Partitioning for GALS-based Networks-on-Chip", Proc. IEEE/ACM Design Automation Conf., pp.110–115, 2007.
[14] W.Ning, G.Fen, W.Fei, "Design of a GALS Wrapper for Network on Chip", World Congress on Computer Science and Information Engineering, pp.592–595, 2009.
[15] D.Mangano, G.Falconeri, C.Pistritto, A.Scandurra, "Effective full-duplex Mesochronous Link Architecture for Network-on-Chip Data-Link layer", Proc. of the 10th Euromicro Conference on Digital System Design Architectures, Methods and Tools, pp.519–526, 2007.
[16] Y.Thonnart, E.Beigne, P.Vivet, "Design and Implementation of a GALS Adapter for ANoC Based Architectures", Proc. of the 2009 15th IEEE Symp. on Asynchronous Circuits and Systems, pp.13–22, 2009.
[17] M.Ghoneima, Y.Ismail, M.Khellah, V.De, "Variation-Tolerant and Low-Power Source-Synchronous Multi-Cycle On-Chip Interconnection Scheme", VLSI Design, 2007.
[18] Zhiyi Yu, Bevan M. Baas, "High Performance, Energy Efficiency, and Scalability with GALS Chip Multiprocessors", IEEE Trans. VLSI, vol.17, no.1, pp.66–79, 2009.
[19] G.Campobello, M.Castano, C.Ciofi, D.Mangano, "GALS networks on chip: a new solution for asynchronous delay-insensitive links", Proc. of DATE, pp.160–165, 2006.
[20] E.Beigne, F.Clermidy, P.Vivet, A.Clouard, M.Renaudin, "An Asynchronous NOC Architecture Providing Low Latency Service and Its Multi-Level Design Framework", Proc. of the 11th IEEE Int. Symp. on Asynchronous Circuits and Systems, pp. 54–63, 2005.
[21] E.Beigne, P.Vivet, "Design of on-chip and off-chip interfaces for a GALS NoC architecture", Proc. of the 12th IEEE Int. Symp. on Asynchronous Circuits and Systems, pp.172, 2006.
[22] M.Krstic, E.Grass, C.Stahl, "Request-driven GALS technique for wireless communication system", Proc. of the 11th IEEE Int. Symp. on Asynchronous Circuits and Systems, pp.76–85, 2005.
[23] E.Flamand, "Strategic Directions Towards Multicore Application Specific Computing", Proc. of DATE, page 1266, 2009.
[24] S.Borkar, "Design Perspectives on 22nm CMOS and Beyond", Proc. of DAC 2009.
[25] A.Tran, D.Truong, B.Baas, "A GALS Many-Core Heterogeneous DSP Platform with source-Synchronous On-Chip Interconnection Network", Proc. of Int. Symp. on Networks-on-Chip, 2009.