Comparing Tightly and Loosely Coupled Mesochronous Synchronizers in a NoC Switch Architecture

Daniele Ludovici[§], Alessandro Strano[†], Davide Bertozzi[†], Luca Benini[††], Georgi N. Gaydadjiev[§]
[†] ENDIF, University of Ferrara, 44100 Ferrara, Italy.
[††] DEIS, University of Bologna, 40136 Bologna, Italy.
[§] Computer Engineering Lab., Delft University of Technology, The Netherlands.
email: *d.ludovici@tudelft.nl, alessandro.strano@student.unife.it*
dbertozzi@ing.unife.it, luca.benini@unibo.it, g.n.gaydadjiev@tudelft.nl

Abstract

With the advent of Networks-on-Chip (NoCs), the interest for mesochronous synchronizers is again on the rise due to the intricacies of skew-controlled chip-wide clock tree distribution. Recently proposed schemes agree on a source synchronous design style with some form of ping-pong buffering to counter timing and metastability concerns. However, the integration issues of such synchronizers in a NoC setting are still largely uncovered. Most schemes are in fact placed between communicating switches, thus neglecting the abrupt increase of buffering resources needed at switch input stages. This paper goes a step forward and aims at deep integration of the synchronizer in the switch architecture, thus merging key tasks such as synchronization, buffering and flow control into a unique architecture block. This paper compares the integrated and the loosely coupled solutions from a performance and area viewpoint, while devoting special attention to their robustness with respect to physical design parameters.

I. Introduction

The problem of distributing the global clock in a chip with minimal clock skew is getting difficult to solve due to the reverse scaling of wire delay in nanoscale integrated circuits. At the same time, the design paradigm shift towards on-chip multi-processor architectures is raising the need for easily extensible clock trees and for self-assembly clock tree synthesis strategies like the one used in the Polaris chip [1].

A fully asynchronous approach to global intra-chip communication would eliminate the clock distribution concern and would make designs more modular since timing assumptions are explicit in the hand-shaking protocols. Unfortunately, current design tools and IP libraries heavily rely on the synchronous paradigm instead, thus making intermediate solutions more attractive and affordable in the short run.

An incremental approach with respect to the current design practice consists of mesochronous synchronization. A single clock signal is distributed to the various macrocells in the design with an arbitrary amount of space-dependent time-invariant phase offset (i.e., the skew). Mesochronous synchronization can be viewed not just as an enabler for architecture scalability, but also as a means of eliminating (or relieving) the skew constraints in the clock tree synthesis process, thus resulting in frequency speed-ups, power consumption reductions and fast back-end turnarounds [20].

Unfortunately, mesochronous synchronizers come with their own set of problems. A traditional approach to their design consists of delaying either data or the clock signal so to sample data only when it is stable. This solution requires components often not available in a standard-cell design flow (e.g., configurable digital delay lines) or explicitly targeting full-custom design (e.g., Muller C-elements).

Another issue concerns the synchronization latency, which impacts not just communication performance but has architecture-level implications as well, thus resulting in a more costly system overall. As an example, a latency-insensitive receiver architecture has to be designed.

Flexible synchronizers have been proposed that can support even arbitrary clock frequencies in the communicating domains (e.g., dual-clock FIFOs [3]), however they incur a large area and power overhead. Even custom-tailoring the synchronizer for mesochronous systems does not easily avoid such an overhead, like the scheme in [2], which implies 8 FFs and a mux for each synchronized bitline.

In the direction of providing a low-area and low-latency realization of a mesochronous synchronizer, some recent works suggest a source-synchronous design style combined with some form of ping-pong buffering to counter timing and metastability concerns [6], [7]. While [6] avoids a phase detector but requires the link delay to be less than one clock period, the scheme in [7] can handle slow and long links but requires a phase detector.

Both schemes feature substantial pros (ease of implementation in traditional design flows, minimal complexity), which motivates the further work that has been done in order to more thoroughly investigate the implications of their utilization in the context of on-chip networks (NoCs) [5], [8]. Knowledge of the target application domain may in fact pave the way for large optimizations of the basic synchronizer architecture and circuits and for tuning the low-level details of the design. Unfortunately, such customization works for synchronizers are not as frequent in the open literature as the proposal of new synchronization concept schemes.

As an example, the work in [8] assesses timing margins in a real NoC test case, thus coming up with optimized circuit solutions. Both [5] and [8] address the support for bidirectional communication and for backwards flow control needed for mesochronous signaling in a NoC. In this paper, a further optimized variant of the source synchronous scheme presented in [8] is used as the baseline synchronization scheme for the sake of comparison. In particular, the phase detector is avoided through a careful design of the synchronizer buffer stage and of its behavior at reset.

However, while sharing the same synchronizer design

978-1-4244-4143-3/09/$25.00 ©2009 IEEE

philosophy of both [5] and [8], this paper moves a significant step forward in the direction of a synchronizer customization for NoCs. In fact, all previous solutions envision a loosely coupled synchronizer with the switch architecture. This approach has the clear drawback of implying a larger amount of buffering resources in the upstream and/or downstream switch of the mesochronous link, which depends on the introduced synchronization latency and on the specific flow control strategy employed by the network. The idea behind this work consists of merging the synchronizer architecture with that of the switch input buffer, thus coming up with a compact architecture block taking care of flow control, synchronization and buffering and tightly integrated in the switch architecture itself. Beyond making mesochronous NoC design more modular, this approach results in large area savings for the overall switch due to the multi-purpose buffer design strategy and in a reduction of the communication latency as well. Layout synthesis of the synchronization-enriched switch architecture by means of a commercial flow proves its feasibility in a 65nm technology under different physical design parameters.

II. Related Work

Research on mesochronous synchronization started a long time ago. A well-established solution illustrated in [9] consists of delay-line synchronizers, using a variable delay on the data lines. This delay is computed in such a way to avoid switching in the metastability window of the receiving registers. Variable delay lines make this solution expensive and not always available in standard cell libraries. This is the same problem of the works in [12], [13], which use voltage comparators.

Several periodic synchronizers are illustrated in [16], which avoid metastability by delaying either the data or the clock signal to sample data when the clock is stable. Configurable digital delay lines are again needed and experimented frequency is very low.

The works in [12], [11], [10] achieve mesochronous data synchronization by using Muller C-elements and digital delay lines that are typically designed with a full-custom approach. [10] presents a self-tested self-synchronization method for mesochronous communication. The scheme uses two clocks with a phase shift of 180° and a failure detector is used to select which one to use. In [11] a phase detector in place of a metastability detector is used in the same scheme.

Architectures based on FIFO synchronizers are proposed in [9], [14]. FIFO size in [14] depends on the skew, hence is link-dependent or given in the worst-case. Implementation is also very expensive, as showed in [15].

More recently, an optimized bisynchronous FIFO has been proposed in [3] featuring low-latency and small footprint. It can be adapted to the mesochronous needs while proving able to tolerate skew only up to 50% of the clock period.

Summing up, mesochronous synchronizers presented so far incur few out of several disadvantages: high implementation overhead, use of non-trivial or full-custom components or low skew tolerance. Moreover, very few works are able to assess timing margins with layout awareness.

More recently, the unrelenting pace of technology scaling and the design paradigm shift to NoC-based on-chip multiprocessor systems are bringing physical layer issues to the forefront, especially clock distribution. Solutions able to mitigate them are more urgent, and this has justified a renewed interest in mesochronous synchronization. However, the perspective is not on the concept synchronization scheme any more, but rather on its suitability for the integration into the on-chip network architecture [18]. This involves assessing timing margins in the target domain and dealing with domain-specific concerns such as the need to properly handle flow control.

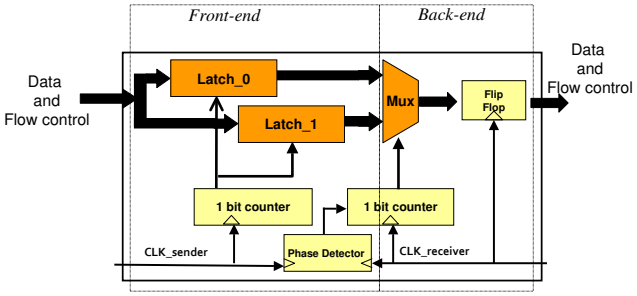


Fig. 1. Baseline synchronizer of [8].

An early mesochronous scheme for the SoCBus NoC was proposed in [17], aiming at compact realization while still lacking of a validation on an NoC test case.

A significant step forward comes from the OCN system [19], which uses a source synchronous scheme. A matched-delay architecture is used to compensate the strobe skew and enable high-speed mesochronous communication. A FIFO synchronizer is used at the receiver side.

Two recent papers [7], [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. The SKIL link [5] can support arbitrarily skewed clock signals by relying on a two-stage buffer structure and can go through a standard design flow. The same work has been taken a step forward in [4] by considering flow control and support to virtual channels in the synchronizer architecture. Similarly, the approach in [7] aims at assuring that the receiver clock reads data when they are stable. However, the focus of this work is still on long generic links, nothing closely related to NoCs. The study of this synchronizer inside of a NoC layout for short-range mesochronous links is reported in [8], where NoC-specific timing margins are assessed and flow control implications thoroughly investigated. A similar scheme is implemented in the mesochronous interfaces of the Polaris chip [1].

This paper takes the same approach to synchronization of [5], [8] (source synchronous data transmission, safe storage of data at the receiver side, sampling in the receiver domain only when data is stable) but improves baseline architectures and circuits by providing a more compact and equally robust solution. However, this is used just as the baseline architecture for the sake of comparison with the novel synchronization structure that we propose in this paper. Our guiding principle consists of tightly integrating the synchronizer module into the switch architecture, so to design a multi-purpose switch input stage taking care of synchronization, buffering and flow control. We view this as a way of keeping architecture overhead when moving from fully synchronous to mesochronous clocking close to the minimum.

III. Baseline Synchronization Architecture

This work moves from the synchronizer architecture presented in [8] and illustrated in Fig.1. The circuit receives as its inputs a bundle of NoC wires representing a regular NoC link, carrying data and/or flow control commands, and a copy of the clock signal of the sender. Since the latter wire experiences the same propagation delay as the data and flow control wires, it can be used as a strobe signal for them. The circuit is composed by a front-end and a back-end. The front-end is driven by the incoming clock signal, and

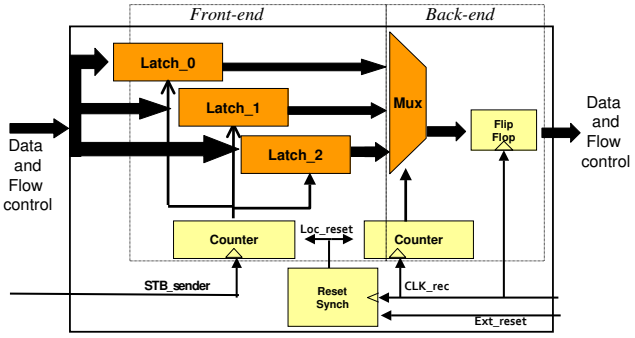


Fig. 2. Baseline loosely coupled synchronizer of this paper.

strobes the incoming data and flow control wires onto a set of parallel latches in a rotating fashion, based on a counter. The back-end of the circuit leverages the local clock, and samples data from one of the latches in the front-end thanks to multiplexing logic which is also based on a counter. 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 read by the target clock domain and sampled by a regular flip-flop.

Counters in the front-end and back-end are initialized upon reset, after observing the actual clock skew among the sender and receiver with a phase detector, so as to establish a proper offset. The phase detector only operates upon the system reset, but given the mesochronous nature of the link, its findings hold equally well during normal operation.

Since few flow control wires are traveling backwards, another similar but much smaller synchronizer needs to be instantiated at the sender to handle them.

A. Optimizations of the baseline synchronizer

We agree with [8] that it is always possible to choose a counter setup so that the sampling clock edge in the back-end captures the output of the latches in a stable condition, even accounting for timing margin to neutralize jitter. Therefore, no more than 2 latches in parallel are needed in the front-end for short-range (i.e., single cycle) mesochronous communication with this scheme.

Other considerations however suggest that a different choice may be desirable at this point. In particular, we feel that by increasing the number of input latches by one more stage it becomes possible to avoid the phase detector (see the new architecture in Fig.2). This would be desirable due to the timing uncertainty or the high area footprint or the non-compliance to a standard cell flow that affects many phase detector implementations. A third latch bank allows to keep latched data stable for a longer time window and to even find a unique and safe bootstrap configuration (i.e., counters initialization) that turns out to be robust in any phase skew scenario. Post-synthesis simulations confirmed that the circuit works properly when sweeping the clock skew from -360° to +360°. Post-layout robustness will be assessed in the experimental section. At regime, the output multiplexer always selects the output of the latch bank preceding the bank which is being enabled by the front-end counter. Rotating operation of both front- and back-end counters preserves this order.

In contrast to [8], the reset architecture is designed, as Fig.2 shows. In most SoCs, the reset signal coming into the chip is an asynchronous input. Therefore, reset de-assertion should be synchronized in the receive clock domain. In fact, if a reset removal to a flip flop occurs close to the active edge of its clock, flip flops can enter a metastable state. We use a brute-force synchronizer (available in some new technology libraries as a standard cell) for reset synchronization with the receiver clock. Now the problem arises about how to reset the front-end. Typically, a reset can be sent by the upstream switch. In our architecture, we prevent metastability in the front-end by delaying the strobe generation in the upstream switch by one clock cycle after reset deassertion. This way, on the first strobe edge, the receiver synchronizer is already reset. This strobe generation delay is compliant with network packet injection delay after reset. The transmitter clock signal is used as the strobe signal in our architecture. Differently than [8], a larger timing margin is enforced for safe input data sampling. In fact, the transmitter clock signal has to be processed at the receiver in order to drive the latch enable signals. In actual layouts, this processing time adds up to the routing skew between data and strobe and to the delay for driving the latch enable high-fanout nets. As a result, the latch enable signal might be activated too late, and the input data signal might have already changed. In order to make the synchronizer more robust to these events, we ensure that input data sampling occurs in the middle of the clock period. In fact, a switching latch enable signal opens the sampling window of the next latch during the rising edge, and closes the same during the falling one. As a result, the latch enable activation has a margin of half clock cycle to occur. Our post-layout simulations prove that this margin is largely met in practice. Finally, in agreement with [8], we computed the minimum size of the input buffer in the downstream switch to be 4 slots (flits). They are required by the stall/go flow control protocol in order to cover the round trip latency and not to drop flits in flight when a stall signal has to be propagated backwards. The original input buffer size is 2 slots, reflecting the requirements of stall/go without synchronization. Please refer to [8] for further details on this.

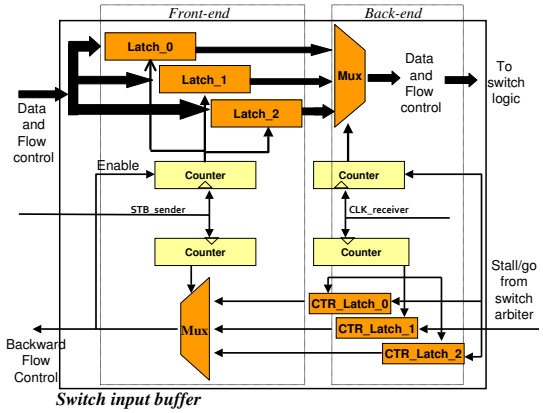


Fig. 3. Proposed tightly coupled synchronizer.

IV. Tightly Integrated Synchronizer Architecture

The previous synchronizer, similarly to SKIL or to the Polaris one, is a module of the NoC architecture, thus loosely coupled with the downstream switch. The loose coupling stems from the fact that the flip flop in the synchronizer back-end belongs directly to the switch input

buffer. The mux output is therefore sampled like any other input in the fully synchronous scenario. However, our early exploration indicates that the area overhead induced in this input buffer as an effect of the added synchronization latency is much larger than the synchronizer area itself.

This indicates that a tighter integration of the synchronizer into the switch input buffer is desirable. In particular, the latch enable signals of the synchronizer front-end could be conditioned with backward-propagating flow control signals, so to exploit input latches as useful buffer stages and not just as an overhead for synchronization. Should this be the case, input data would be at first stored in the latches and then synchronized. This would allow to completely remove the switch input buffer and to replace it with the synchronizer itself. The synchronizer output would then be directly fed to the switch arbitration logic and to the crossbar. The ultimate consequence is that the mesochronous synchronizer becomes the actual switch input stage, with its latching stages acting as both buffering and synchronization stages (see Fig.3). A side benefit is that the latency of the synchronization stage in front of the switch is removed, since now the synchronizer and the switch input buffer coincide. The main change required to make the new architecture come true is to bring flow control signals to the front-end and back-end counters of the synchronizer. This solution would still require 4 slot buffers, i.e., 4 latching banks. However, a further optimization is feasible. The backward-propagating flow control signal (the stall/go signal) could be directly synchronized with the strobe signal in the synchronizer front-end before being propagated to the upstream switch. This would save also the synchronizer at the transmitter side. In fact, the backward-propagating signal would be already in synch with the strobe, which in turn is in synch with the transmitter clock. The ultimate result is the architecture illustrated in Fig.3. We are aware that this latter choice shrinks timing margins for the backward flow control signal, in that it leaves the downstream switch with some generation delay across its synchronizer and also experiences the link propagation delay. This margin will be assessed post-layout in the experimental section, proving the applicability of the scheme and providing simple variants when long links need to be crossed. For this architecture solution, only 3 latching banks are needed in the synchronizer. In practice, only 1 slot buffer more than the fully synchronous input buffer. The tightly coupled synchronizer makes the mesochronous NoC design fully modular like the synchronous one, since no external blocks to the switches have to be instantiated for switch-to-switch communication. Please notice that the reset architecture remains unchanged with respect to Fig.2.

A. Operating principle

In case a *go* signal comes from the switch arbiter, at each clock cycle data are latched in the input buffers of the synchronizer, synchronized with the local clock and propagated to the switch arbiter and crossbar. When a *stall* occurs, the output mux keeps driving the same output until communication can be resumed. While the stall signal gets synchronized with the strobe and reaches the front-end, the front-end latches keep sampling input flits in a rotating way. When the stall signal finally leaves the synchronizer, it will stop the transmission of the upstream switch and the front-end counter operation at the same time. At this point, the situation is frozen. When then a *go* arrives, the output mux becomes operational again. Later, input latches and upstream switch resume their operation again at the same time. Please observe that this mechanism does not waste bandwidth on flow resumption, since the synchronizer backend can immediately start sweeping the output of front-end latches upon receipt of a *go*. Interestingly, flow control logic in the synchronizer is simplified with respect to that of the original switch input buffer. Before, a finite state machine used to generate a stall by monitoring the number of elements in the buffer. When it was equal to one and a stall came from the switch internal logic, than a stall was also generated for the upstream switch. In the new architecture, the synchronizer just synchronizes the stall signal from the switch logic with the transmitter clock and propagates it upstream. This way, large logic is saved. The latency of both the tightly and the loosely coupled synchronizers varies depending on the skew, and ranges from one to two clock cycles.

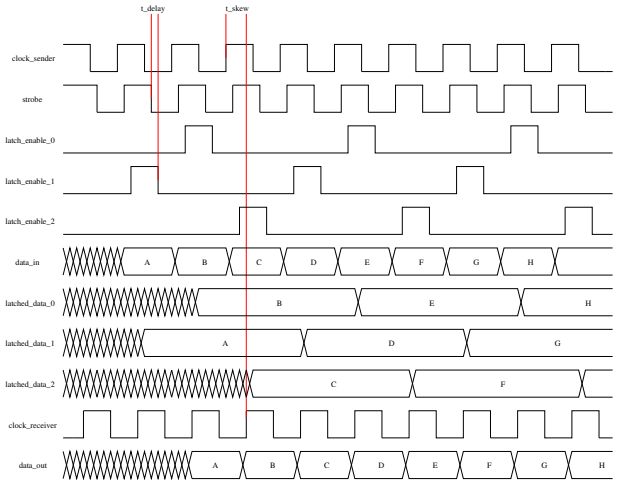


Fig. 4. Waveforms example of the tightly coupled synchronizer.

Fig.4 reports the waveforms showing operation of the tightly coupled synchronizer. A delay from the strobe signal is assumed for the latch enable signals to account for their high-fanout.

V. Experimental Results

The mesochronous synchronizers illustrated so far have been implemented by means of the xpipesLite NoC library [21]. All the analyzes discussed in this work have been carried out by means of a backend synthesis flow leveraging industrial tools. The technology library is a low-power low-Vth 65nm STMicroelectronics library available through the CMP project [22].

A. Comparative latency Analysis

Since the tightly coupled synchronizer not only changes the synchronizer implementation but also affects the entire network architecture, we performed basic tests to capture the macroscopic performance differences implied by the different synchronization architectures. We focus on synchronization latency, since the stall/go mechanism implemented

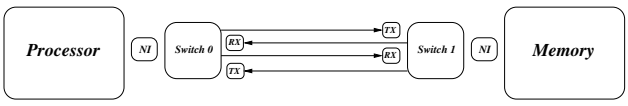
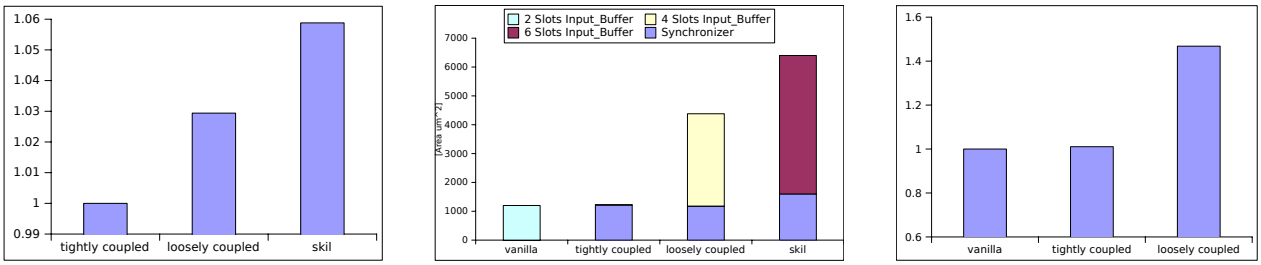


Fig. 6. Test-case platform under analysis.



(a) Execution cycles (b) Input buffer + synch. area breakdown (c) Power consumption

Fig. 5. Performance, area breakdown and power results.

in our synchronizer ensures that a stall-to-go transition of the flow control signal can be immediately propagated to the next stage. Hence, there are no wasted cycles at flow resumption, differently than [8]. Since what matters here is not a network-wide performance analysis, but just to investigate the latency of each scheme, this feature can be more conveniently stimulated and analyzed in a simple ad-hoc experimental test case for fine-grain performance analysis. We opted for a simple processor–NoC–memory topology (see Fig.6). The investigated NoC is comprised of a couple of 2x2 switches respectively connected to the processor and the memory. Furthermore, each network switch is connected to its own RX-, TX-mesochronous part meant for synchronizing received data and flow control signals. For the sake of comparison, the SKIL synchronizer is considered as well. This is another loosely coupled module with the switch architecture. The implementation was entirely derived from [5].

The traffic pattern consists of full-bandwidth read and write transactions, i.e., the target memory never stops the access flow. Of course, the only performance differentiation is seen for read transactions, since they are blocking for the processor core, hence they rely on the network ability to keep latency to a minimum. Performance results could be easily interpreted by means of a simple analytical model (Formula 1). It relates performance results to the intrinsic design characteristics of each synchronizer.

In fact in the best case, SKIL exposes two cycles synchronization overhead plus a further execution cycle for traversing the network switch; whereas our loosely coupled solution only requires one cycle latency in the mesochronous plus one cycle in the network switch. Even better, the tightly coupled mesochronous synchronizer requires the same computational resources of the vanilla switch (i.e., 1 execution cycle). The reason is that the tightly coupled solution seamlessly replaces the input buffer of the network switch thus providing a fast, reliable and robust mechanism for data synchronization.

Summarizing, whenever the system with the tightly coupled mesochronous synchronizer performs a computational task in n cycles, the alternative schemes, i.e., SKIL and the loosely coupled synchronizer respectively require a number of cycles equal to Formula 1, where *latency* is the number of clock cycles of the deployed mesochronous architecture whereas *#transaction* is the number of read operations performed by the processor unit. As depicted in Fig. 5(a) there is a direct impact of the adopted synchronization solution on the overall system performance. While the tightly coupled solution keeps the same performance as the vanilla network switch, a performance drop up to 6% incurs when using the SKIL scheme in spite of the very simple test case.

$$cycles = n + latency \times 2 \times \#transactions \quad (1)$$

B. Post-layout analysis

We went through a commercial backend synthesis flow and refined RTL description of the mesochronous switches (tightly and loosely coupled) up to the physical layout. The following analysis considers a *stall/go* flow control scheme.

	Critical Path		Area (μm^2)	Area overhead
	RX	TX		
SKIL	0.89 ns	0.28 ns	6400	5.33x
loosely coupled	0.68 ns	0.29 ns	4380	3.65x
tightly coupled	0.49 ns		1200	1x

TABLE I. Post-layout report.

Table I reports post-layout critical path as well as area footprint (also Fig. 5(b)) of the investigated mesochronous solutions taken in isolation. The area breakdown in Fig. 5(b) is that of the switch input buffer plus that of the transmitter and receiver synchronizers. Size of the switch input buffer is 4 due to the fact that 2 clock cycles of latency are added in the round-trip from the tx and rx synchronizers (1 cycle each). Therefore, a properly oversized input buffer is needed to handle a stall without incurring any data loss. For the same reason, as deploying a SKIL synchronizer in the link requires 3 clock cycles to traverse the channel, the amount of required buffering resources in the switch input buffer is 6 as shown in Fig. 5(b). For the tightly coupled solution, total area just refers to the multi-purpose switch input buffer (which is also the synchronizer).

Concerning performance, post-layout critical path of the tightly coupled solution is not directly comparable with the SKIL and the loosely coupled one as it is fully integrated in the network switch architecture. Nonetheless, as the mesochronous design per se does not represent the performance bottleneck of the whole NoC architecture (which can run at 1 GHz in all cases), the low area footprint overhead of the tightly coupled mesochrounous solution turns out to be an appealing peculiarity when compared with more area-greedy alternatives as SKIL and the loosely coupled synchronizers.

C. Skew tolerance

This section studies the skew tolerance of the tightly vs. loosely coupled mesochronous synchronizers. In order to estimate the window size where the incoming data can be sampled and held as valid, we performed a complete set of post-layout simulations with 0% $\rightarrow$ 100% different skews for the investigated synchronizers. Fig.7 reports the hold time for the sampling element fed by the synchronizer's multiplexer, since the setup time was found to meet even

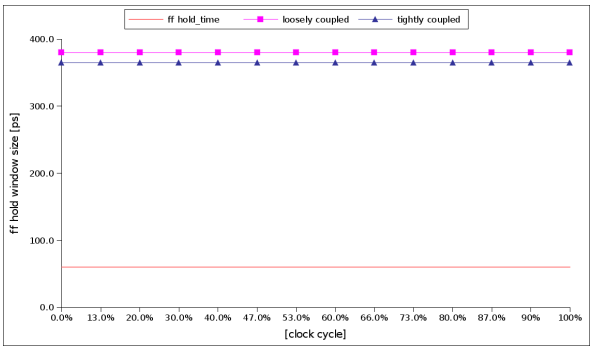


Fig. 7. Skew tolerance of tightly vs. loosely coupled mesochronous synchronizers expressed as the hold time of the sampling element fed by the synchronizer's multiplexer.

larger timing constraints. As the figure shows, both synchronizer solutions turn out to be highly skew-tolerant, in that the hold time remains well above 300 ps. This way, timing margins are guaranteed, since the value of a library cell hold time for the target 65 nm technology is always much lower than 300 ps. In spite of this, it must be emphasized that the loosely coupled solution exhibits a larger window size mainly due to the less complex logic of the counter driving the mux in the backend of the synchronizer.

D. Extension for long links and power analysis

The tightly coupled mesochronous solution is able to absorb clock skew from -360° up to +360° for incoming data signals. They are correctly synchronized despite of long wires whereas backward-propagating flow control signals are more sensible to link length. The reason is that the timing margin since the strobe edge occurs at the transmitter switch until the backward propagating flow control signal comes back is one clock cycle. This condition is difficult to meet for overly long links between two network switches. We assessed 4mm switch–to–switch link length as a feasible distance for the tightly coupled mesochronous scheme after layout on silicon reaching timing closure. We did not experiment for longer links, left for future work. Instead, we propose a simple implementation variant to be used when the above condition fails. In practice, a mixed solution might be used where the tightly coupled synchronizer is utilized for short range communication, while an external synchronizer for the backward-propagating flow control signals is added for long range communication with a negligible area overhead.

Final step of our exploration was to contrast power consumption of the proposed mesochronous schemes. In order to perform a fair evaluation, we compared the tightly coupled mesochronous solution (along with the switch where it is integrated) against to a loosely coupled mesochronous along with the switch it is connected to. For the sake of completeness, the obtained results have been compared with a vanilla network switch. Interestingly, power consumption of such plain solution almost corresponds to the one of the switch with the tightly coupled mesochronous integrated as depicted in Fig 5(c). The reason is that, both switches are very similar as the input buffer of the plain switch has been seamlessly replaced by the integrated mesochronous. Concerning the tightly vs. loosely coupled solution comparison, it must be noticed that the tightly coupled version has been optimized from the power consumption viewpoint. Indeed, in a single module it implements synchronization as well as data storage and flow control management requiring a smaller number of buffering resources when compared to the loosely coupled solution. In fact in the latter, the mesochronous synchronizer requires an additional power greedy input buffer in the switch in order to properly work.

VI. Conclusions

In this paper we address the customization of mesochronous communication concept schemes for the target network-on-chip domain. We move from the consideration that a loosely coupled synchronizer with the switch architecture implies a significant buffering overhead in the switch input buffer, larger than the mesochronous synchronizer itself. As a consequence, we advocate for tight integration of the synchronizer in the switch. In the proposed architecture, the synchronizer latches act as both synchronization, buffering and flow control resources at the same time, completely replacing the pre-existing input buffer. The resulting architecture proves robust to physical design parameters (skew, link length) and features almost no area overhead with respect to the baseline switch used in fully synchronous networks.

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).

References

[1] 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.
[2] P.Caputa and C.Svensson, "An On-Chip Delay- and Skew-Insensitive Multicycle Communication Scheme", Proc. IEEE Int. Conf. Solid-State Circuits, pp.1765–1774, 2006.
[3] I.M.Panades, A.Greiner, "Bi-Synchronous FIFO for Synchronous Circuit Communication Well Suited for Network-on-Chip in GALS Architectures", Int. Symp. on Networks-on-Chip, pp.83–94, 2007.
[4] S.Saponara, F.Vitullo, R.Locatelli, P.Teninge, M.Coppola, L.Fanucci, "LIME: A Low-latency and Low-complexity On-chip Mesochronous Link with Integrated Flow Control", 11th EUROMICRO DSD'08, pp.32–35, 2008.
[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] D.Mangano, R.Locatelli, A.Scandurra, C.Pistritto, M.Coppola, L.Fanucci, F.Vitullo, D.Zandri, "Skew Insensitive Physical Links for Networks-on-Chip", Int. Conf. on Nano-Networks, pp.1–5, 2006.
[7] M.Ghoneima, Y.Ismail, M.Khellah, V.De, "Variation-Tolerant and Low-Power Source-Synchronous Multi-Cycle On-Chip Interconnection Scheme", VLSI Design, 2007.
[8] I.Loi, F.Angiolini, L.Benini, "Developing Mesochronous Synchronizers to Enable 3D NoCs", VLSI Design, 2007.
[9] W.J.Dally, J.W.Poulton, "Digital Systems Engineering", Cambridge University Press, 1998
[10] F.Mu, C.Svensson; "Self-Tested Self-Synchronization Circuit for Mesochronous Clocking", IEEE Trans. on Circuits and Systems II: Analog and Digital Signal Processing, Vol.48, no.2, pp.129–141, 2001.
[11] B.Mesgarzadeh, C.Svensson, A.Alvandpour, "A New Mesochronous Clocking Scheme for Synchronization in SoC", ISCAS, pp.605–609, 2002.
[12] S.Kim, R.Sridhar, "Self-Timed Mesochronous Interconnections for High-Speed VLSI Systems", GLSVLSI, pp.122–128, 1996.
[13] M.R.Greenstreet, "Implementing a STARI chip", ICCD, pp.3, 1995.
[14] A.Edmanand, C.Svensson, "Timing Closure through Globally Synchronous,Timing Portioned Design Methodology", DAC, pp.71–74, 2004.
[15] P.Caput, C.Svensson, "An On-Chip Delay- and Skew-Insensitive Multicycle Communication Scheme", IEEE Conf. Solid-State Circuits, pp.1765–1774, 2006.
[16] Y.Semiat and R.Ginosar, "Timing Measurements of Synchronization Circuits", Int. Symp. on Advanced Research in Asynch. Circuits and Systems, pp.68–77, 2003.
[17] D.Wiklund, "Mesochronous Clocking and Communication in On-Chip Networks", Proc. Swedish System-on-Chip Conf., 2003.
[18] D.Mangano, A.Scandurra, C.Pistritto, "Relieving Physical Issues in New NoC-based SoCs", Int.Conf. on Nano-Networks, 2007.
[19] D.Kim, K.Kim, J.Y.Kim, S.Lee, H.J.Yoo, "Solutions for Real Chip Implementation Issues of NoC and Their Application to Memory-Centric NoC", Int. Symp. on Networks-on-Chips, 2007.
[20] 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, 2008.
[21] S.Stergiou, F.Angiolini, S.Carta, L.Raffo, D.Bertozzi, G.De Micheli, "XPipes Lite: a Synthesis Oriented Design Library for Networks on Chips". Proc. of DATE, pp.1188–1193, 2005.
[22] Circuits Multi-Projects, Multi-Project Circuits; http://cmp.imag.fr