Architecture Design Principles for the Integration of Synchronization Interfaces into Network-on-Chip Switches

Daniele Ludovici
Computer Engineering Lab., TUDelft
2628 CD Delft, The Netherlands
daniele@ce.et.tudelft.nl

Alessandro Strano, Davide Bertozzi
ENDIF, University of Ferrara
44100 Ferrara, Italy
strlsn1@unife.it, dbertozzi@ing.unife.it

ABSTRACT

This paper contributes to the maturity of the GALS NoC design practice by advocating for tight integration of GALS synchronization interfaces into NoC architecture building blocks. At the cost of re-engineering the input/output stages of NoC switches and network interfaces, this approach proves capable of materializing GALS NoCs with the same area and power of their synchronous counterparts, while reducing latency at the clock domain boundary. This design style is experimented in this paper with a mesochronous synchronizer and a dual-clock FIFO, which are tightly coupled with the switches of the xpipesLite NoC architecture.

Categories and Subject Descriptors

B.7.1 [**Hardware**]: Integrated Circuits—*VLSI*

General Terms

Design, Performance

Keywords

Networks-on-Chip, Globally Asynchronous Locally Synchronous, Dual-Clock FIFO, Mesochronous Synchronization

1. INTRODUCTION

Traditional globally synchronous clocking circuits have become increasingly difficult to design with growing chip size, clock rates, relative wire delays and parameter variations. Additionally, high speed global clocks consume a significant portion of system power and lack the flexibility to independently control the clock frequencies of submodules to achieve high energy efficiency. The globally asynchronous locally synchronous (GALS) clocking style [22] separates processing blocks such that each block is clocked by an independent clock domain and is an effective strategy to address global clock design challenges [23].

This new paradigm heavily impacts the architecture of the chip-wide communication infrastructure. There is consensus on the fact that an on-chip interconnection network (NoC) can provide physical scalability due to the tile-based architecture without global wires, distribution of the interconnect fabric across the entire chip and load reduction on inter-processor links [24]. GALS NoC features no global clock distribution, thus simplifying the classical timing closure process that would otherwise require several systematic iterations to converge. Moreover, rigid timing constraints between local clock domains do not need to be enforced.

Although it is an appealing alternative to the common design practice, the GALS NoC paradigm lends itself to a wide range of possible implementations, each with its own trade-offs. The main difference lies in the circuitry to reliably and efficiently move data across clock domain boundaries.

On one hand, *single transaction handshaking* (used in [25, 26]) acknowledges each data word before a subsequent word can be transferred, and a corresponding latency exists for each data transfer and acknowledgment, which can significantly decrease the total data throughput. Beyond requiring hardware designers with a good background on asynchronous design, this approach also currently suffers from the low maturity of tools for electronic design automation.

On the other hand, *source synchronous interfaces* (used in [27]) route the source clock along with the data for correct synchronization at the receiving end. The source transmits data words without individual acknowledgments and it halts transfers when the destination indicates it can no longer accept new data. The technique normally works well with high data rates, since one word per clock cycle can be easily achieved. This latter solution certainly requires only an incremental evolution of current EDA tools to be supported and is therefore an amenable solution in the short run, while the former approach might be a more forward looking solution. Many hardware designers are however skeptical about the source synchronous design style since it requires expensive synchronizers at the clock domain boundary. This has a number of implications on total area and power figures of the NoC.

First, it introduces additional communication latency. As a result, provisions must be normally made since the flow control signal may arrive multiple clock cycles after the destination module decides to halt the source module [19]. The problem can be addressed by reserving space in the destination buffer, so that this technique in general requires a larger buffer memory but should normally sustain higher throughputs. Unfortunately, buffers typically consume almost 50% of total NoC power [16], therefore by requiring additional buffering resources the overhead for the synchronization interface partially offsets the energy efficiency of the GALS paradigm. Second, synchronizers affect the modularity of the NoC architectural template, which typically consists of just two network building blocks: switches and network interfaces.

This paper moves from the viewpoint that most of the above overhead for source synchronous interfaces is due to the conventional way of implementing synchronization interfaces. In fact, synchronizers are typically inserted between two connected network building blocks belonging to different clock domains. In practice, they break the switch-to-switch or the network interface-to-switch connections depending on the decisions about clock domain partitioning. However, there is typically no co-optimization of the synchronizer with the following/preceding NoC sub-module, therefore a significant latency, area and power overhead materializes. This

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
NoCArc '09, December 12, 2009, New York City, New York, USA
Copyright 2009 ACM 978-1-60558-774-5 ...$10.00.

design practice is hereafter denoted as the *loose coupling* of synchronization interfaces with the NoC. In contrast, the objective of this paper is to prove that the *tight integration* of the synchronizer into the NoC switch can result into a novel architecture block taking care of synchronization but also of other tasks such as switch input buffering and flow control. The consequent reuse of buffering resources for different purposes in turn leads to large energy savings that make a GALS NoC affordable at almost the same area and power cost of its synchronous counterpart. Moreover, by moving the synchronizer inside the switch (or the network interface), the communication latency at the clock domain boundary reduces to the ideal synchronization latency, which simplifies flow control and reduces its buffering requirements.

In order to prove the practical viability of this approach, this paper demonstrates how it can be applied to two relevant synchronization scenarios. In the former one, the clock domains use the same clock frequency but have unknown phase offset, while in the latter one they have true independent clocks with distinct frequency and offset. The integration process of the synchronizers for the two cases (a mesochronous synchronizer and a dual-clock FIFO, respectively) into the switch input stage of the xpipesLite [21] NoC architecture is detailed in this paper. Finally, the implications on the critical path, area and power of the switch are quantified on an industrial 65nm technology library.

The remaining of this work is as follows. Previous work is reviewed in Section 2. Section 3 presents our mesochronous synchronizer architecture whereas Section 4 describes our dual-clock FIFO solution. Both sections emphasize the benefit of integrating such synchronizer interfaces into a NoC switch architecture. Section 5 points out results in terms of performance, area overhead and power consumption for the investigated architectures. Finally, conclusions are drawn in Section 6.

2. RELATED WORK

Research efforts have recently focused on the development of cost-effective and modular dual-clock FIFO architectures. The work in [1] illustrates a dual-clock FIFO which is modular and can be easily configured for different NoC requirements (clocked or clockless interfaces, synchronization latency for resolving metastability, FIFO capacity). This design was inspired by the modular FIFO from [2], composed of a ring of stages, where each stage is composed of a storage cell, a put interface and a get interface. The solutions in [1] and [2] differ since [1] uses cells that are available from a typical standard cell library, whereas [2] requires custom, pre-charged cells in the control blocks. Furthermore, [1] separates the FIFO control logic from the synchronizers, allowing the synchronization latency to be chosen according to the clock frequency and reliability requirements.

[3] proposed a synchronizing FIFO design based on read and write pointer comparison using Gray codes. The result of the comparison is synchronized to the sender's and receiver's clocks. Their design uses two binary counters for read and write addresses and a dual-port RAM for data storage which results in a larger and more complicated design for the modest FIFO capacities that are typically needed for NoC design. Furthermore, the design in [3] only provides FIFO capacities that are powers of two.

The design in [7] uses the position of the read and write pointers to determine fullness and emptiness. It has one synchronizer to detect fullness, and per-stage synchronizers to detect emptiness. To prevent errors due to metastability when sampling the pointers, they use two tokens each as the read pointer and write pointer. This makes their empty detector a little more complicated, as it requires comparing two synchronized write pointer bits with two read pointer bits for each stage. In addition, to differentiate between a full and an empty FIFO, they only fill the FIFO up to the second-to-last position, which means that an n stage FIFO can only hold $n-1$ items.

Operation of our dual-clock FIFO architecture resembles that of [1]. From an implementation viewpoint, we were inspired by [7] and [3]. From [7], we borrowed the token ring concept for implementing FSMs, since this is a simple and robust solution in contrast to Grey coding. From [3] we borrowed the idea of performing a comparison between read and write pointers asynchronously and then synchronizing the result in the target domains.

For the particular case where the clocks in two communicating domains have the same frequency but unknown phase offset, then a dual-clock FIFO would be too much of an overhead and custom tailored mesochronous synchronizers are normally more cost-effective. Summing up, mesochronous synchronizers presented so far incur few out of several disadvantages: high implementation overhead [4, 5, 6, 7], use of non-trivial or full-custom components [4, 8, 9, 11, 10, 12] or low skew tolerance [7]. More recently, some works have addressed mesochronous synchronization from the strict viewpoint of suitability for application to a NoC environment [13, 14]. Two papers from industries [15, 17] 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 scheme in [15] has been extended to short-range mesochronous links in [19]. Our previous work in [20] further improves that synchronizer by avoiding the need for a phase detector, by enhancing the timing margin for safe input data sampling and by determining the minimum amount of buffering for correct operation.

This overview of previous work was also the foundation for our design activity of a mesochronous synchronizer and a dual-clock FIFO to be used as an experimental setup in this paper, together with the support of the xpipesLite NoC architecture. Our previous work in [20] was a feasibility study for the tight integration of a mesochronous synchronizer into the NoC architecture. This paper builds on that study by bringing architecture design techniques for synchronizer-switch co-design to maturity and by proving that the advantages of tight integration are not associated with a specific synchronization architecture, but can be applied to the general case of multiple independent clock domains.

3. MESOCHRONOUS SYNCHRONIZER

The loosely coupled mesochronous synchronizer used to prove the effectiveness of the tight NoC integration design principle is 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 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.

With no more than 2 latches in parallel in the front-end of the synchronizer, 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. This holds for short-range (i.e., single cycle) mesochronous communication. However, by increasing the number of input latches by one more stage it becomes possible to avoid a phase detector for counter initialization. See more details in [20]. This would be desirable due to the timing uncertainty, the high area footprint and the non-compliance to a standard cell design flow that affects many phase detector implementations.

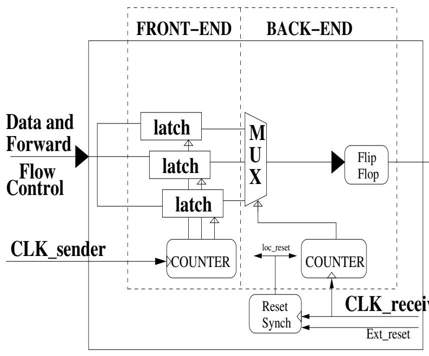


Figure 1: Loosely Coupled Mesochronous Synchronizer.

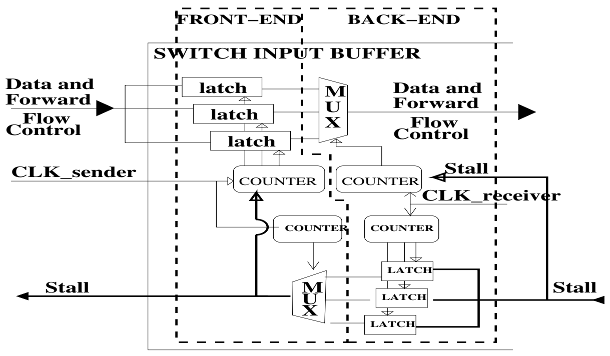


Figure 2: Tightly Coupled Mesochronous Synchronizer.

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 that turns out to be robust in any phase skew scenario from -360° to +360°.

Special care is devoted to enforcing timing margins 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 make the strobe signal transition only on the falling edge of the transmitter clock. This way, we ensure that input data sampling occurs in the middle of the clock period. As a result, the latch enable activation has a margin of half of the clock cycle to occur.

The xpipesLite NoC architecture uses the basic stall/go flow control protocol [28], which implies two control wires: the valid signal propagating together with data lines, and the stall/go signal propagating in the opposite direction. This latter signal therefore needs a similar but smaller synchronizer instantiated in front of the upstream switch. The synchronizers in the forward and in the backward propagating paths add synchronization latency to those paths. As mentioned before, the increased round-trip latency needs to be reflected in an over-provisioning of buffering resources at the input of the downstream switch for correct flow control operation. As result, the buffer size of the switch input stage needs to be increased from 2 (needed for retiming and flow control in the fully synchronous switch) to 4.

3.1 Tight integration into the switch

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 marginal. First of all, we notice that the latch enable signals of the synchronizer front-end could be conditioned with backward-propagating flow control signals (the stall/go in our case), so to exploit input latches as useful buffer stages and not just as an overhead for synchronization. This means that in case of stall driven by the downstream switch, data could be frozen in the synchronizer front-end, while in the current implementation they are propagated all the way to the switch input stage and only there frozen. With the new proposal, data would be at first stored and then synchronized in the receiver clock domain, therefore buffers at the switch input stage would be useless and they would be completely replaced by the synchronizer latches. This is what has been done in the upper part of Fig.2.

The synchronizer output now directly feeds the switch arbitration logic and the crossbar, thus materializing the tight integration concept of the mesochronous synchronizer into the switch architecture. The ultimate consequence is that the mesochronous synchronizer becomes the actual switch input stage, with its latching banks serving both for performance-oriented buffering and synchronization. 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 for the correct operation of the new architecture is to bring the stall/go flow control signal to the front-end and back-end counters of the synchronizer, in order to freeze their operation in case of a stall. While this signal is already in synch with the back-end counter, it should be synchronized with the transmitter clock before entering the front-end counter. The backward-propagating stall/go is then directly synchronized with the transmitter clock available in the front-end by means of a similar but smaller (1-bit) synchronizer.

This paves the way for a relevant optimization. The output of the 1-bit synchronizer could be brought not only to the front-end counter, but also directly to the upstream switch without the need for any further synchronizer since synchronization has already been performed in the downstream switch. The ultimate result is the architecture illustrated in Fig.2. For this architecture solution, only 3 latching banks are needed in the synchronizer front-end, since latency has been reduced to a minimum. 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.

When assessing the timing margins of this architecture, it has to be guaranteed that the stall/go signal reaches the upstream switch before the next rising edge of the clock. Since this signal leaves the downstream switch after the transmitter clock (propagated from the upstream switch) has triggered the front-end counter, this implies that the round-trip delay should be lower than a clock period. Whenever this constraint cannot be met because of the link length/delay, it is always possible to revert to a hybrid solution. The synchronizer for the datapath is still integrated into the switch, while the stall/go signal feeds two 1-bit synchronizers: one at the downstream switch for control of the front-end counter and one at the upstream switch for correct sampling with the transmitter clock. This way, the round-trip dependency is broken at the cost of a minor impact on modularity (i.e., the 1-bit external synchronizer at the upstream switch is still needed).

4. DUAL-CLOCK FIFO ARCHITECTURE

The main goal of our proposed FIFO is to interface two synchronous systems having different clock signals. Each system is synchronous with its own clock signal but can be asynchronous (frequency and/or phase) to the others. The challenge of this architecture is to hide all synchronization issues while respecting the FIFO protocol on each interface. Furthermore, this architecture is scalable and synthesizable

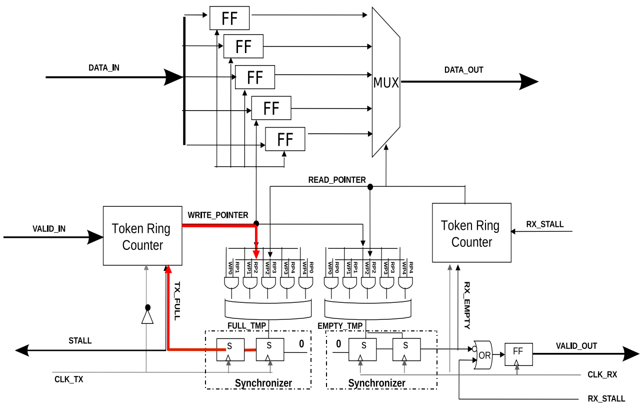


Figure 3: Dual-Clock FIFO Architecture.

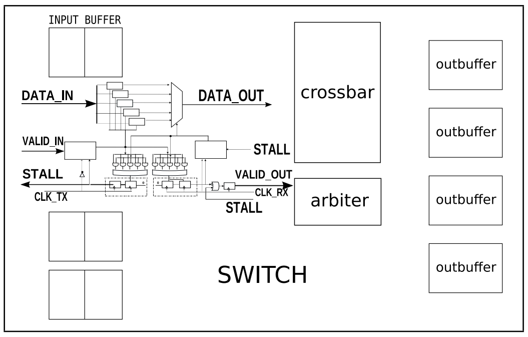


Figure 4: Dual-Clock FIFO integration into the NoC switch architecture.

with a standard cell design flow without using custom cells.

The dual-clock FIFO (see Figure 3) has been designed directly for use in a NoC implementing the stall/go flow control protocol. The queuing and dequeuing of data elements in the FIFO follow the following protocol. The *data_in* is queued into the FIFO, if and only if the *valid_in* signal is true and the *full* signal is false at the falling edge of *clk_Tx*. Again, operation is triggered at the clock falling edge to preserve timing margins. Symmetrically, data is dequeued to *data_out*, if and only if the *RX_stall/go* signal is false (go) and the *empty* signal is false at the rising edge of *Clk_Rx*. The clear partitioning of the sender and receiver interfaces into synchronous and independent interfaces is designed to facilitate timing closure for the modules connected to the FIFO ports.

As shown in Figure 3, the bisynchronous FIFO architecture is composed of 2 token ring counters. In the sender interface, the token ring counter is driven by the *Clk_Tx*, synchronous to incoming data. It generates the *write* pointer indicating the position to be written in the data buffer. In the receiver interface, the token ring counter is driven by the local clock, *Clk_Rx*. It generates the *read* pointer indicating the position to be read in the data buffer. The data buffer contains the data storage of the FIFO, which is parameterizable.

Full and *empty* detectors signal the fullness and the emptiness of the FIFO. In our solution, these detectors perform asynchronous comparisons between the FIFO *write* and *read* pointers that are generated in clock domains that are asynchronous to each other. The asynchronous FIFO pointer comparison technique uses few synchronization flip-flops to build the FIFO.

The Full detector computes the Full signal using the *write* pointer and *read* pointer contents. The Full detector requires N two-input AND gates, one N-input OR gate, where N is the FIFO depth. The detector computes the logic AND operation between the Write and Read pointer and then collects it with an OR gate, obtaining logic value 1 if the FIFO is full, 0 otherwise. The FIFO is considered full when the *write* pointer points to the previous position of the *read* pointer. Viceversa, the FIFO is considered empty when the *write* pointer points to the same position of the *read* pointer.

Assertion of the *empty_tmp* signal is synchronous to the *Clk_Rx*-domain, since *empty_tmp* can only be asserted when the *read* pointer is incremented, but de-assertion of the *empty_tmp* signal happens when the *write* pointer is increased, which is an asynchronous event to *Clk_Rx*. On the contrary, assertion of the *full_tmp* signal is synchronous to the *Clk_Tx*-domain, since *full_tmp* can only be asserted when the *write* pointer is incremented, but de-assertion of *full_tmp* happens when the *read* pointer increments, which is an asynchronous event to *Clk_Tx*. As a consequence, the *full_tmp* and *empty_tmp* signals, coming out of an asynchronous comparison of read and write pointers, need to be synchronized by means of carefully engineered brute force synchronizers.

In particular, when the *read* pointer catches up with the *write* pointer, the *empty_tmp* signal presets the *rx_empty* flip-flops. When a FIFO write takes place, the *write* and *read* pointer contents are different, thus the *empty_tmp* can be deasserted, releasing the preset control of the synchronizer. The *rx_empty* will be deasserted after two rising edges of *Clk_Rx*. Since the deassertion of *empty_tmp* occurs on a falling *Clk_Tx* edge while *rx_empty* is clocked by *Clk_Rx*, the two-flop synchronizer is required to remove metastability associated with the asynchronous deassertion of the preset control of the first flip flop. Also the removal of the preset signal on the second flip-flop can violate the recovery time. But in this case the second flip-flop will not go metastable because the preset to the flip-flop has forced the output high so far and the input to the same flip-flop is also high, which is not subject to a recovery time instability [3].

The *tx_full* signal could be generated in an equivalent way with respect to *rx_empty* but an optimization is required to satisfy proper NoC constraints. In fact, the proposed FIFO is designed to support a STALL/GO flow control protocol. Since *tx_full* signal generated by the two-flop synchronizer coincides with the stall/go signal propagated upstream, it needs to be generated on the rising edge of *Clk_Tx*. In this way, the time since the strobe edge occurs at the transmitter switch until the backward propagating flow control signal comes back should be one clock cycle. In order to meet this timing constraint, the two-flop synchronizer that generates *tx_full* samples on the rising edge of *Clk_Tx* and *full_tmp* does not preset the second *tx_full* flip-flop. In particular, when a FIFO-write operation causes a full condition on the falling edge of *Clk_Tx*, the *full_tmp* signal is consequently asserted and presets the first *tx_full* flip-flop. Therefore, assertion of the *tx_full* signal occurs on the next rising edge of *Clk_Tx*, and it can in turn be safely sampled by the counter on the next falling edge of the same clock. In practice, the token ring counter cannot progress any more since the detection of the full condition. This mechanism relieves the round-trip dependency, since the stall/go signal leaves the FIFO as soon as the rising edge of *Clk_Tx* arrives and samples it, without waiting for the next falling edge.

Finally, when a FIFO-read operation takes place, the *read* pointer is incremented and the *full_tmp* signal is de-asserted, thus releasing the preset control of the first *tx_full* flip-flop. Then, because of the low logic value driving the input port of the first *tx_full* flip-flop, the FIFO will de-assert the *tx_full* signal after two rising edges of *Clk_Tx*. Because the deassertion of *full_tmp* occurs on a rising *Clk_Rx* and because *tx_full* is clocked by the *Clk_Tx*, the two-flop synchronizer is required to remove metastability that could be generated by the first *tx_full* flip-flop.

The *tx_full* assertion process generates a critical timing path. The *full_tmp* critical timing path consists of (1) the $tx\_clk - to - q$ incrementing of the *write* pointer, (2) comparison logic of the *read* pointer with the *write* pointer, (3) presetting the first *tx_full* flip-flop, (4) meeting the setup time of the second *tx_full* flip-flop clocked with *Clk_Tx*. This critical path has to be covered in half clock cycle since *write* pointer is incremented on the falling edge while the *tx_full* flip-flop is sensitive to the rising edge.

As regards the receiver domain, in the bi-synchronous FIFO in isolation the empty assertion crosses a similar critical path but this time it can be properly covered in one clock cycle since the two-flop synchronizer is sensitive to the rising edge like the receiver token ring. Then the *rx_empty* assertion does not represent a critical path.

4.1 Tight integration into the switch

The proposed FIFO synchronizer was then tightly integrated into the switch architecture. In fact, the basic idea is to share the buffering resources of the bi-synch FIFO with those of the switch input stage, thus resulting in a more compact and higher performance realization. As illustrated in Fig.4, the integration does not present any additional problem with respect to that of the mesochronous synchronizer. The data buffer of the dual-clock FIFO is very similar to the architecture of the vanilla switch input stage, therefore there are no major implications on the switch critical path. Moreover, the stall/go signal is provided by the arbiter, which receives the valid signal as an input. In contrast to the mesochronous synchronizer, now the 2 slot input buffer of the vanilla switch has been replaced by a dual-clock FIFO that requires 5 slot buffers, thus leading to a significant overhead. In fact, we proved that in every frequency ratio scenario between sender and receiver, 100% throughput is guaranteed in the presence of a FIFO depth of at least 5.

However, we notice that the xpipesLite architecture is natively output buffered, but nothing prevents us from implementing an input buffered switch with minor effort. The only change consists of exchanging the performance-oriented multi-slot buffer at the output ports with the retiming and flow control stages in the input ports and viceversa. If we assume that a port buffer for performance optimization can consists of at least 5 or 6 slots, then the (at least) 5 slots needed by the dual-clock FIFO come at no overhead with respect to the fully synchronous switch.

5. EXPERIMENTAL RESULTS

This section illustrates the implications (in terms of performance, area occupancy and power consumption) of integrating the previously described synchronizers into a NoC switch architecture. The chosen radix is 5, to reflect switches commonly found in 2D mesh topologies. Post-layout results are obtained with a 65nm STMicroelectronics technology library.

5.1 Mesochronous Synchronizer

As regards latency, while the fully synchronous switch takes 1 cycle in the upstream link and 1 cycle in the switch itself, the novel switch with tightly integrated synchronizers takes from 1 to 3 clock cycles to cross the same path, depending on the negative or positive skew ranging from -100% to +100%, respectively.

In order to estimate the area savings with the tight integration design strategy, we went through a commercial synthesis flow and refined RTL description of the mesochronous switches (tightly and loosely coupled) up to the physical layout. All the systems were synthesized, placed and routed at the same target frequency of 1GHz. In Fig. 5(a), the area footprint of switches is reported along with a breakdown pointing out the contribution of synchronizers and/or input buffers. The tightly and loosely coupled solutions are compared against a vanilla (i.e., fully synchronous) switch; *input_buffer* area for this switch only refers to the area occupancy of a normal 2 slot input buffer. For the loosely coupled solution, a 4 slot buffer is needed to cover the round trip latency, and this is most of the overhead for this solution. As clearly pointed out by the area breakdown in Fig. 5(a), the sum of the transmitter and of the receiver synchronizers is almost equal to that of a 2 slot buffer, i.e., of the input buffer in the vanilla switch. For the tightly coupled solution, *input_buffer/synchronizer* area refers to the multipurpose switch input buffer (which is also the synchronizer). Clearly, there is almost no area overhead when moving from a fully synchronous to a tightly integrated mesochronous switch as they employ similar buffering resources.

From the performance viewpoint, our post-layout synthesis results confirm that the critical path of the switch is not impacted by the replacement of the vanilla input buffer with the tightly integrated mesochronous synchronizer. By experimenting with different switch radix, the critical path deviates only marginally in the two cases, therefore no performance penalty should be expected for the mesochronous switch.

A further step of our exploration was to contrast power consumption of the proposed mesochronous schemes. Our target design is a 5x5 switch in three different variants: the first has a tightly integrated mesochronous synchronizer per input port; the second has a pair of loosely coupled rx- and tx-synchronizers per input port; the last one is a vanilla switch with 5 fully synchronous input ports. Three different traffic patterns have been experimented to carry out the power analysis: *idle*, request for a *random* output port and *parallel* communication flows. Post-layout simulation frequency was 700MHz for all the designs. As showed in Fig. 5(b), in all the cases, the highest power consumption is consumed by the most buffer demanding solution, i.e., the loosely coupled design. Power consumption of the vanilla and tightly coupled designs are similar as expected; this is mainly due to the equivalent buffering resources deployed in both switches.

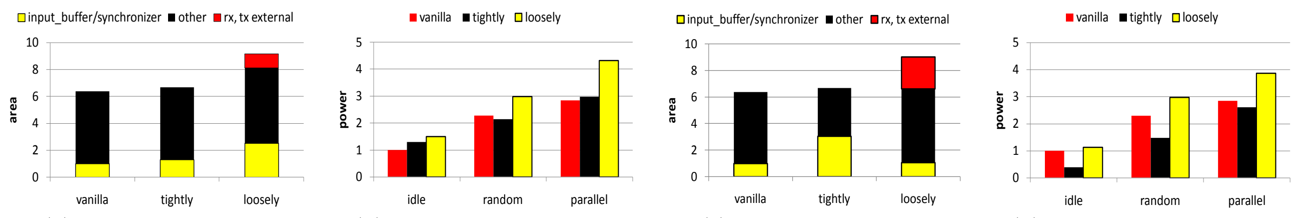
5.2 Dual-Clock FIFO

As the sender and the receiver have different clocks, the latency of the dual-clock FIFO depends on the relation between these two signals. The aforementioned metric can be decomposed in two parameters: the first is a ΔT_{rx} that is the time between the falling edge of the clock sender and the rising edge of the clock receiver. ΔT_{rx} can vary between 0 and 1 clock cycle depending of the offset between the clock signals. The second parameter is the number of clock cycles required by the read pointer to reach the location pointed by the writer. As reported in Table 1, three different scenarios have been analyzed in order to characterize the *crossing latency of the switch with the integrated dual-clock FIFO*. In the first, when the *read* and *write* pointers are adjacent and the writer is preceding the reader, a minimum latency occurs. In this case, the *read* pointer opens the mux window after ΔT_{rx} for the data it is pointing to and the next data (which is the one being currently written) will be read after a further clock cycle. Therefore, the minimum latency to traverse the FIFO synchronizer in this case is $\Delta T_{rx} + 1Clock_{rx}$. In the second case, when the buffer is *empty* and a write operation occurs, a $\Delta T_{rx} + 1Clock_{rx}$ is needed to clear the emptiness condition and a further clock cycle is required to enable the data at the multiplexer output. Finally, when the distance between the pointers is maximum (fullness condition), the required time for the reader to point the current writer position is given by a ΔT_{rx} (offset between the clock signals) plus a contribution which depends on the number of buffer slots preceding the one currently pointed by the writer (which accounts to *BufferDepth-2*). Please note that in all latency results 1 $Clock_{rx}$ cycle has been added to account for the time from the FIFO output to the input of the switch output buffer. In fact, Table 1 reports the overall switch crossing latency.

I°	minimum latency	$\Delta T_{rx} + 2Clock_{rx}$
II°	empty deassertion	$\Delta T_{rx} + 3Clock_{rx}$
III°	maximum latency	$\Delta T_{rx} + Clock_{rx} \times (BufferDepth - 1)$

Table 1: Switch crossing latency.

In order to characterize the integration of a dual-clock FIFO into a NoC switch architecture from the area and power viewpoint, three different systems have been considered. The first is the conventional 5x5 vanilla switch with a 2-slot input buffer and a 6-slot output buffer per port. The second version is a switch where a dual-clock FIFO with 6



(a) Area breakdown. (b) Power consumption. (c) Area breakdown. (d) Power consumption.

Figure 5: Post-layout normalized results of a switch with Mesochronous (a), (b) and Dual-clock FIFO (c), (d) synchronizers.

buffer slots has been tightly integrated into each input port. In order to carry out a fair comparison with the vanilla system, total buffering resources have been kept equal, i.e., the output buffer size in the switch with the FIFO synchronizer has been reduced from 6 to 2 slots. The last configuration is a vanilla switch (2-slot input, 6-slot output) with an external dual-clock FIFO (6 buffer slots) per input port.

To assess area occupancy, all the above switch configurations have been synthesized, placed and routed at the same target frequency of 1GHz. Total area of the tightly coupled system exhibits almost the same area footprint of the vanilla switch. This is a direct consequence of the fact that exactly the same buffering resources have been deployed in a specular fashion (between input and output) in these systems as explained above. As showed in Fig. 5(c), the main difference is in the distribution of the cell area devoted to synchronization. Since the classical 2-slot input buffer has been replaced with a larger 6-slot one in the tightly coupled system, the switch presents a higher *input_buffer/synchronization* area with respect to the vanilla system. The loosely coupled system features the same area overhead (with the same distribution of *input_buffer* and *other* cell area) of the fully synchronous switch plus a further synchronization area due to the external block implementing the dual-clock FIFO.

As for the mesochronous synchronizer, a number of experiments has been carried out to assess the power consumption of a switch integrating dual-clock FIFOs on each input port. The vanilla, tightly and loosely coupled systems have been tested under different traffic patterns: idle, random and parallel. Post-layout simulations have been carried out at 800MHz. All the results were grouped per traffic pattern. As we found out in previous experiments (in Section 5.1) with the switch connected with an external mesochronous synchronizer, the area overhead comes along with a power penalty. In fact, the switch with the external dual-clock FIFO is the most power greedy under all possible traffic patterns, as showed in Fig. 5(d). This is due to a larger amount of buffering resources. From the power viewpoint, there is a substantial benefit when integrating the dual-clock FIFO in the switch architecture, in fact, the tightly coupled design is the most power saving among those under test.

The motivation lies in the inherent clock gating which is implemented by our dual-clock FIFO, which clocks only one bank of flip-flops at a time out of the total input buffer. If the incoming data is not valid, then the token ring circuit does not even switch thereby gating the entire input buffer. Obviously a similar clock gating technique can be applied to the vanilla switch as well, and in fact the key take-away here should be that the dual-clock FIFO integration into the switch does not imply any major power overhead, as long as buffer depths of at least 6 flits are used in all switch variants for performance optimization.

6. CONCLUSIONS

This paper advocates for a tight integration of synchronization interfaces into NoC architecture building blocks, and proves that it is possible to evolve a fully synchronous switch into a GALS counterpart with marginal area and power overhead while preserving the operating speed. This approach holds promise of resulting in cost-effective implementations of GALS NoCs where the source synchronous synchronization paradigm is used for clock domain crossing. The novel architecture implies a different set of timing constraints with respect to those in conventional implementations with external synchronizers. Their definition and enforcement by a commercial backend synthesis toolflow is our ongoing work, aiming at a specialized design flow for the new design strategy of GALS NoCs.

7. REFERENCES

[1] T.Ono, M.Greenstreet, “A Modular Synchronizing FIFO for NOCs”, International Network-on-Chip Symposium, 2009

[2] T.Chelcea, S.M.Nowick, “Robust Interfaces for Mixed-Timing Systems”, IEEE Transactions on Very Large Scale Integration Systems, 12(8): 857-873, 2004.

[3] C.Cummings, P.Alfke, “Simulation and Synthesis Techniques for Asynchronous FIFO Design with Asynchronous Pointer Comparison”, SNUG-2002, San Josè, CA, 2002.

[4] W.J.Dally, J.W.Poulton, “Digital Systems Engineering”, Cambridge University Press, 1998

[5] A.Edmanand, C.Svensson, “Timing Closure through Globally Synchronous,Timing Portioned Design Methodology”, DAC, pp.71–74, 2004.

[6] P.Caput, C.Svensson, “An On-Chip Delay- and Skew-Insensitive Multicycle Communication Scheme”, IEEE Conf. Solid-State Circuits, pp.1765–1774, 2006.

[7] 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.

[8] S.Kim, R.Sridhar, “Self-Timed Mesochronous Interconnections for High-Speed VLSI Systems”, GLSVLSI, pp.122–128, 1996.

[9] M.R.Greenstreet, “Implementing a STARI chip”, ICCD, pp.3, 1995.

[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] Y.Semiat and R.Ginosar, “Timing Measurements of Synchronization Circuits”, Int. Symp. on Advanced Research in Asynch. Circuits and Systems, pp.68–77, 2003.

[13] D.Wiklund, “Mesochronous Clocking and Communication in On-Chip Networks”, Proc. Swedish System-on-Chip Conf., 2003.

[14] 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.

[15] M.Ghoneima, Y.Ismail, M.Khellah, V.De, “Variation-Tolerant and Low-Power Source-Synchronous Multi-Cycle On-Chip Interconnection Scheme”, VLSI Design, 2007.

[16] D. Ludovici, D. Bertozzi, L. Benini and G. N. Gaydadjiev, "Capturing Topology-Level Implications of Link Synthesis Techniques for Nanoscale Networks-on-Chip", Proceedings of the 19th ACM/IEEE Great Lakes Symposium on VLSI (GLSVLSI), pp.125-128, 2009.

[17] 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.

[18] 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.

[19] I.Loi, F.Angiolini, L.Benini, “Developing Mesochronous Synchronizers to Enable 3D NoCs”, VLSI Design, 2007.

[20] 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.

[21] S.Stergiou, F.Angiolini, S.Carta, L.Ra↑o, D.Bertozzi, G.De Micheli, “XPipes Lite: a Synthesis Oriented Design Library for Networks on Chips”. Proc. of DATE, pp.1188–1193, 2005.

[22] D.M.Chapiro, “Globally-Asynchronous Locally-Synchronous Systems”. PhD Dissertation, Stanford University, October 1984.

[23] Z.Yu, B.M.Baas, “High Performance, Energy E¡ciency, and Scalability with GALS Chip Multiprocessors”. IEEE Trans. on VLSI Systems, Vol.17, no. 1, January 2009.

[24] Luca Benini and Giovanni De Micheli, “Networks on chip: a new SoC paradigm”. IEEE Computer, 35(1):70-78, January 2002.

[25] H. Zhang, , V. Prabhu, V. George, M.Wan, M. Benes, A. Abnous, and J. M. Rabaey, “A 1-V heterogeneous recon⌊gurable DSP IC for wireless baseband digital signal processing”. IEEE J. Solid-State Circuits, vol. 35, no. 11, pp. 1697-1704, Nov. 2000.

[26] D. Lattard et al., “A telecom baseband circuit based on an asynchronous network-on-chip”. Proc. ISSCC, Feb. 2007, pp. 258-259.

[27] Z. Yu, M. Meeuwsen, R. Apperson, O. Sattari, M. Lai, J. Webb, E. Work, T. Mohsenin, M. Singh, and B. Baas, “An asynchronous array of simple processors for DSP applications”. in Proc. IEEE Int. Solid-State Circuits Conf. (ISSCC), Feb. 2006, pp. 428-429.

[28] A.Pullini et al., "Fault tolerance overhead in network-on-chip 'ow control schemes", SBCCI, pp.224-229, 2005.