A Library of Dual-Clock FIFOs for Cost-Effective and Flexible MPSoC Design

Alessandro Strano[†], Daniele Ludovici[§], Davide Bertozzi[†]
[†] ENDIF, University of Ferrara, 44100 Ferrara, Italy.
[§] Computer Engineering Lab., Delft University of Technology, The Netherlands.

***Abstract*—Customization of IP blocks in a multi-processor system-on-chip (MPSoC) is the historical approach to the cost-effective implementation of such systems. A recent trend consists of structuring a MPSoC into loosely coupled voltage and frequency islands to meet tight power budgets. In this context, synchronization between islands of synchronicity becomes a major design issue. Dual-clock FIFOs compare favorably with respect to synchronizer-based designs and pausible clocking interfaces from a performance viewpoint, but incur a significant area, power and latency overhead. This paper proposes a library of dual-clock FIFOs for cost-effective MPSoC design, where each architecture variant in the library has been designed to match well-defined operating conditions at the minimum implementation cost. Each FIFO synchronizer is suitable for plug-and-play insertion into the NoC architecture and selection depends on the performance requirements of the synchronization interface at hand. Above all, components of our synchronization library have not been conceived in isolation, but have been tightly co-designed with the switching fabric of the on-chip interconnection network, thus making a conscious use of power-hungry buffering resources and leading to affordable implementations in the resource constrained MPSoC domain.**

I. Introduction

Multi-Processor System-on-Chips are a key technology for the success of a wide range of applications in the embedded systems market spanning from mobile phones to any kind of digital play apparatus. Current semiconductor roadmaps foresee that MPSoCs will be more and more power greedy at every next technology node. The way designers typically tackle this problem is twofold: by customizing all the system-on-chip components for the application domain at hand and by leveraging voltage/frequency partitioning and scaling. In fact, a MPSoC is typically partitioned into independent voltage/frequency islands.

Although customization is an effective way to save power and improve performance of any specific system-on-chip component, it comes with a cost from the design effort viewpoint. First, standard socket interfaces need to be typically designed in order to absorb the communication protocol differences among the components (data width, handshaking protocol, signaling needs). OCP and AXI are relevant examples of such industry-standard interface sockets. Second, component diversity shows up also at a lower level of abstraction, in that each component has to be operated at a different speed which might even range over time as an effect of dynamic voltage and frequency scaling policies.

Therefore, a key enabler for a successful integration of many intellectual property (IP) blocks working at different and varying frequencies is the utilization of synchronization interfaces at their boundaries. In particular, in order to achieve arbitrary frequency decoupling between two different clock domains (let us assume a direct correspondence between an IP block and a clock domain) while retaining high throughput, the utilization of a FIFO-based synchronizer becomes a must [15].

However, if a single general purpose dual-clock FIFO architecture were used for all IP blocks, there would be a high risk of resource over provisioning as different IP cores would certainly have different speed and throughput requirements and all synchronization interfaces should be conservatively configured for the worst case. As an example, buffering of such synchronizer components should be tailored depending on the sender/receiver frequency ratio. Even more, the design-time availability of information about these frequency ratios (e.g., the on-chip interconnect might always run at a higher speed than connected IP cores), combined with knowledge of performance constraints, would lead to high-impact specializations of the dual-clock FIFO architecture and ultimately to large area and power savings.

The main goal of this paper is to present a library of FIFO-based synchronizer components enabling area and power efficient decoupling between distinct clock domains. Each component is conceived for well-characterized operating conditions (throughput, frequency ratios) and is area and power optimized for each of them. Such a library is a key enabler for the design of cost-effective MPSoCs. Please notice that the library we propose is not just derived by tuning the configuration parameters of the baseline synchronizer architecture, but it also includes fully customized architectures for specific operating conditions.

At the same time, design modularity is preserved since all library components are designed for easy plug-and-play into a MPSoC platform. In particular, in this work we assume that the communication infrastructure of our MPSoC is a Network-on-Chip (NoC). There is wide consensus on the fact that such on-chip interconnection networks can provide physical scalability due to the tile-based architecture they enable without using global wires, to the distribution of the interconnect fabric across the entire chip and to the load reduction they feature on inter-processor links [1].

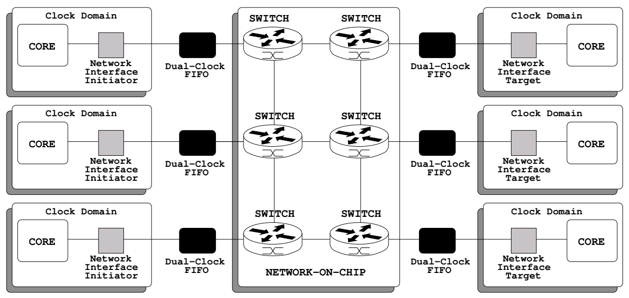


Fig. 1. A Network-on-Chip infrastructure interconnecting multiple frequency domains through dual-clock FIFO interfaces.

Synchronization interfaces, such as dual-clock FIFOs, are typically instantiated as external blocks with respect to the module they are connected with. An example can be viewed in Figure 1, which shows a GALS NoC platform with bi-synch FIFOs at IP core boundaries and the network as a global mesochronous domain [4]. This “loose coupling” of synchronizers with respect to NoC components implies several drawbacks. First, the FIFO module introduces additional communication latency in the intercommunication link. 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 [11]. The problem can be addressed by reserving space in the destination buffer, thus incurring a significant area and power overhead, or by enhancing the dual-clock FIFO with flow control capability.

As an extension of our previous work on mesochronous synchronizers [12], we believe that the aforementioned problem can be tackled by merging the dual-clock FIFO with the switch input buffer, thus coming up with a unique architecture block in charge of buffering, synchronization and flow control and sharing buffering resources for all of these tasks. This paper shows that this design principle, which we denote as “tight coupling” of synchronizer with NoC, can be applied to dual-clock FIFOs in an even more straightforward way than we demonstrated for mesochronous synchronizers in [12]. As a result, all components of our synchronization library have been conceived for tight integration into NoC building blocks.

While developing library components for this target, we also tackled the challenging problem of timing margins. Our dual-clock FIFOs have been designed to adhere to the source synchronous paradigm, where data is sent to the receiver along with the clock of the sender. Safe sampling of data at the receiver side in the presence of routing skew between data and clock is therefore key for the success of the architecture, while ensuring correct timing of the NoC module directly fed by the synchronizer/input buffer. Therefore, we have designed the front-end of our synchronizer library components so to provide good timing margins for the source synchronous link without impacting correct behavior of the NoC switch.

The remaining of this paper is as follows. Previous work is reviewed in Section II. The main architecture variants of dual-clock FIFOs are illustrated in Sections III and VI. Latency analysis is performed in Section IV while throughput analysis is illustrated in Section V. Experimental results characterizing the benefits of NoC-synchronizer merging and the power/area trade-off spanned by synchronization library components are reported in Section VII. Conclusions are drawn in Section VIII.

II. Related Work

Since it is becoming evident that distributing a global clock tree across the entire chip while meeting tight skew constraints is an intricate, expensive if not infeasible task, synchronization interfaces between multiple clock domains are receiving increasing attention. In this context, FIFO interfaces are generally believed to be an effective means of decoupling sender and receiver actions and to maximize throughput. Unfortunately, they come at a significant latency, area and power cost. Therefore, research efforts have recently focused on the development of cost-effective and modular dual-clock FIFO architectures. From the pure fully asynchronous world, there are several examples of FIFOs in literature [16], [17]. Since these designs do not utilize clocks, they are difficult to be adopted when two different clock domains have to be synchronized (e.g., two frequency islands of a GALS system). A reliable data transfer across unrelated asynchronous clock domains is accomplished by [18]. This work is demonstrated in both standard cell and full custom design used in a GALS array processor. In any case, the presented synchronizer does not account for timing implication when integrated in the NoC domain. The work in [2] illustrates a modular and easily configurable dual-clock FIFO for different NoC requirements (clocked or clock-less interfaces, synchronization latency for resolving metastability, FIFO capacity). This design was inspired by the modular FIFO from [3], 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 [2] and [3] differ as [2] uses cells that are available from a typical standard cell library, whereas [3] requires custom, precharged cells in the control blocks. Nevertheless, [2] achieves performance which is comparable or even slightly higher than that reported in [3] when scaling for the different fabrication technologies. Furthermore, [2] separates the FIFO control logic from the synchronizers, allowing the synchronization latency to be chosen according to the clock frequency and reliability requirements. [5] 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

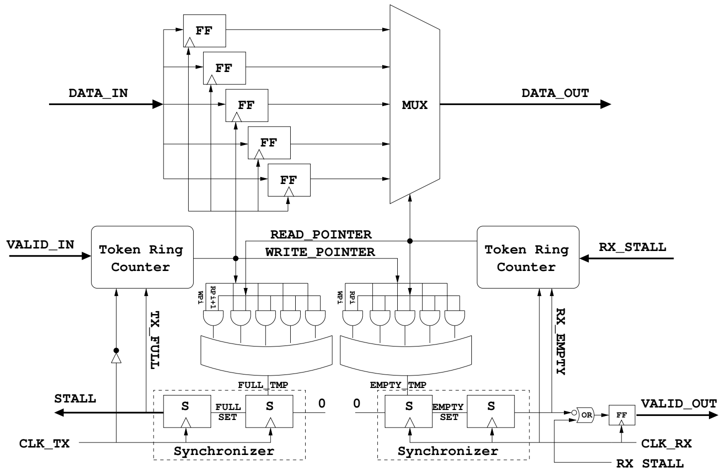


Fig. 2. Dual-Clock FIFO Architecture.

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 [5] only provides FIFO capacities that are powers of two. The design in [8] uses the position of the *read* and *write* pointers to determine fullness and emptiness condition. It employees one synchronizer to detect fullness, and a synchronizer per-stage 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 bit 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 a n stage FIFO can only hold $n-1$ items.

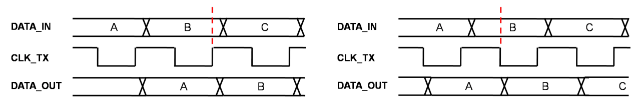
The operating principle of the dual-clock FIFO architecture proposed in this paper resembles that of [2]. From an implementation viewpoint, we were inspired by [8] and [5]. From [8], we borrowed the token ring concept for implementing FSMs, since this is a simple yet robust solution with respect to Grey coding. From [5] we borrowed the idea of performing a comparison between read and write pointers asynchronously and then synchronizing the result in the target domains. With respect to previous work, this paper presents not just a single dual-clock FIFO architecture for use in the general case, but specializes the architecture for the specific operating conditions.

While no previous work addresses the co-design of synchronizers with NoC building blocks, our past work in [12] for the first time proposed to merge switch input buffers with mesochronous synchronizers. Basic principles for this merging process were then more extensively detailed in [9]. This paper extends those efforts by applying the tight synchronizer-NoC coupling design principle to the dual-clock FIFOs of the proposed library, while verifying timing margins in the NoC setting.

III. Baseline Dual-Clock FIFO Architecture

All the components of our synchronization library are derived from a baseline dual-clock FIFO architecture, which is designed to be compatible with a standard cell design flow (i.e., without using custom cells).

The dual-clock FIFO (see Figure 2) is designed directly for use in a NoC setting. The xpipesLite NoC architecture is used as an experimentation platform. The NoC architecture determines the flow control protocol that the FIFO has to implement for correct interfacing with the NoC. xpipesLite implements the stall/go flow control protocol requiring two control wires: one flags data availability while the other one



(a) Sampling on clock rising edge. (b) Sampling on clock falling edge.

Fig. 3.

propagates in the opposite direction and instructs the sender to stop or resume the flit transmission flow.

The circuit in Figure 2 receives as input a bundle of NoC wires representing a regular NoC link, carrying data and flow control commands, and a copy of the clock signal of the sender. Since the latter wire in principle experiences the same propagation delay as the data and flow control wires, it can be used as a strobe signal for them. The bi-synchronous FIFO has a sender and a receiver interface. Each interface has its own clock signal, *Clk_Tx* for the sender and *Clk_Rx* for the receiver.

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 write pointer. In actual layouts, this processing time adds up to the routing skew between data and strobe and to the delay for driving the clock input of the flip-flops. As a result, the strobe 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, each data is sampled approximately in the middle of the clock period (see Figure 3(a) and Figure 3(b) for a pictorial illustration).

Queuing and de-queuing of data elements in the FIFO follow the protocol we describe hereafter. 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*. 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*.

As shown in Figure 2, the bi-synchronous 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. Data buffer contains the data storage of the FIFO, which is parameterizable.

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

The *full_detector* decides the value of the *full* signal depending on the content of *write* and *read* pointer. It requires N 2-input AND gates and 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 the result along with an OR gate, thus 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 de-asserted, releasing the preset control of the synchronizer. The *rx_empty* will be de-asserted after two rising edges of *Clk_Rx*. Since the de-assertion 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 de-assertion 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. However, in this case the second flip-flop will not go to a metastability state 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 [5].

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*. 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 longer 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. Therefore, due to 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*. As the de-assertion of *full_tmp* occurs on a rising edge of *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 (i) the $tx\_clk - to - q$ incrementing of the *write* pointer, (ii) comparison logic of the *read* pointer with the *write* pointer, (iii) presetting the first *tx_full* flip-flop, (iv) 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. Furthermore, the *rx_empty* assertion does not represent a critical path.

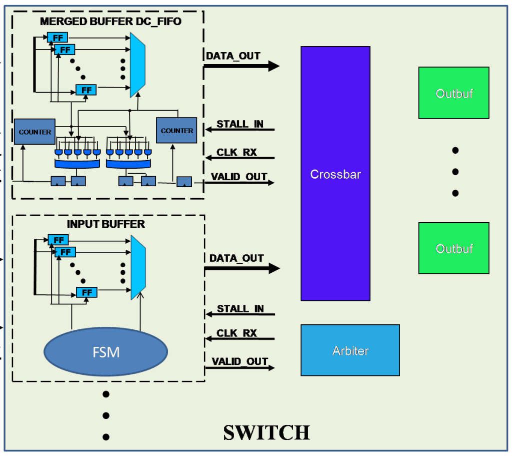


Fig. 4. Dual-Clock FIFO integration into one input port of the NoC switch architecture.

A. Tight Integration into the Switch Architecture

The dual-clock FIFO presented above can be used as a standalone component in a generic instance of the xpipesLite architecture, since it properly implements flow control. However, placing a FIFO in front of a NoC switch (like in Figure 1) implies a latency and buffering overhead for clock domain decoupling.

We found an alternative approach effective for mesochronous synchronizers in [9], [12]: merging the synchronizer with the switch input buffer allows to share buffering resources for multiple tasks, namely switch buffering, flow control and synchronization, thus leading to a cost effective implementation of the synchronization interface. Unfortunately, this tight integration of architecture modules in some cases raises new timing constraints that have to be verified or even enforced with proper architecture-design techniques, as showed in [4].

With this paper we extend the tight coupling design philosophy to dual-clock FIFOs, and prove that the with the architecture illustrated in Section III, merging with the switch input buffer is straightforward (and far easier than for mesochronous synchronizers) and does not fundamentally alter circuit timing.

As illustrated in Figure 4, the data buffer of the dual-clock FIFO is very similar to the architecture of the vanilla switch input stage (natively consisting of sampling elements, a mux and a FSM), therefore experimental results will prove no major implications on the switch critical path (see Section VII-A). Moreover, the stall/go signal is provided by the arbiter, which receives the valid signal $valid\_out$ as an input.

The 2-slot input buffer of the vanilla switch has been replaced by a dual-clock FIFO that requires 5-slot buffers in the case where high performance needs to be guaranteed, thus leading to an area and power overhead. We will prove in Section V 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 slots. Please notice that in any case the latency overhead of the synchronization interface is much reduced, since FIFO synchronizer and switch input buffer and not cascaded anymore.

The buffering overhead cannot be removed, but it is so marginal that in typical use cases it can be completely hidden. In fact, the xpipesLite architecture features input and output buffers, and the size of each of them is individually tunable. Generally, one buffer is kept to its minimum size (2 slots for correct support of flow control), while the other one is oversized to sustain performance and usually set to 6 slots based on past experience on system-level performance analysis [14]. In this scenario, the switch input buffer (merged with the FIFO synchronizer) can be used to implement performance-efficient buffering, while the output buffer can be retained just for retiming purposes. This way, the buffering overhead for synchronization is completely masked and there is fundamentally no area difference between a fully synchronous switch and a switch with synchronization-capable input buffers.

Finally, please observe that Figure 4 emphasizes the possibility to carry out a port-level configuration of the synchronization options to be supported by the switch. One or more input ports could decouple sender/receiver clock domains, while other ports might connect to switches/network interfaces in the same clock domain, thus giving rise to a large number of GALS NoC architecture variants.

IV. Latency Analysis

A. Assertion and de-assertion latency of full/empty

Buffering requirements of the dual-clock FIFO can be limited when the number of clock cycles between the full/empty detection and the write/read suspension can be minimized. To meet this goal, the proposed architecture leverages optimized full/empty brute-force synchronizers having asynchronous preset control. Their runtime operation is detailed hereafter.

Figure 5(a) shows FIFO waveform when the transmitter frequency is higher than the receiver one. Since the receiver is slower than the sender, the dual-clock FIFO will generate periodically full assertion and full de-assertion. During T2 interval, value of the pointer is $WPi = RPi + 1$ and *full_tmp* is asserted, setting *full_set* on falling *tx_clk* edge. During T3, on the rising tx_{clk} edge, the second *tx_full* flip-flop will sample a logical high value and will drive *tx_full* signal high. During T3, on the falling *tx_clk* edge, the token ring counter will receive a high *tx_full* and will interrupt data write. Furthermore, the elapsed clock cycle between the detection of a full assertion and the write suspension is reduced to one clock cycle. During T3, *read_pointer* shifts and *full_tmp* is de-asserted on rising rx_{clk} edge. Consequently, during T4 *full_set* is de-asserted and during T5 also *tx_full* is de-asserted synchronously with tx_{clk}. Finally in T5, token ring counter in the sender domain restarts shifting regularly *write_pointer* on the falling tx_{clk} edge. Therefore, elapsed clock cycles between *full_set* de-assertion and the *write_pointer* shift are reduced to one and a half clock cycles.

Let us now analyze a scenario when the transmitter frequency is lower than the receiver one. Since the *read_pointer* is faster than the *write_pointer*, the FIFO will generate periodically empty assertion and empty de-assertion. Symmetrically, the elapsed clock cycles between the detection of an empty assertion and the read suspension is only one clock cycle. Differently from the previous scenario, the elapsed clock cycles between *empty_set* de-assertion and the *read_pointer* shift is two clock cycles. This is due to the fact that the token ring counter in the receiver domain needs half clock cycle more to sample *rx_empty* due to the intrinsic 2-flop synchronizer latency. As the elapsed time between detection of empty/full and read/write suspension is one clock cycle, no quasi-full and quasi-empty detection need to be implemented and underflow/overflow is avoided by construction. On the contrary, slower empty/full de-assertion does not introduce errors in the FIFO activity but it can impact the FIFO throughput (see Section V).

B. Switch crossing latency

As the sender and the receiver bridged by the dual-clock FIFO have different clocks, the crossing latency of the FIFO depends on frequency ratio and clock phase offset. Latency can be decomposed in two parameters: the first one is ΔT_{rx}, that is the time between the falling edge of the sender clock and the rising edge of the receiver clock. ΔT_{rx} can vary between 0 and 1 clock cycle depending on 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 I, three different scenarios have been analyzed in order to characterize the crossing latency of the switch with the integrated dual-clock FIFO interface.

In the first scenario, when the *read* and *write* pointers are adjacent and the writer is preceding the reader, a minimum

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

TABLE I
SWITCH CROSSING LATENCY.

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 (full condition), the required time for the reader to point the current write 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 I reports the overall switch crossing latency.

V. THROUGHPUT ANALYSIS

We consider our FIFO-based synchronizer to provide 100% throughput when the slowest end of the FIFO (either transmitter or receiver) can push/eject 1 data word per clock cycle.

In case of large FIFO depth, the de-assertion latency of the full signal (needed to notify the writer that further data can be stored) does not impact throughput, in that the reader has enough storage of past data words. This way, the reader avoids a blocking condition due to the delay needed to restart the writer. Similarly, the de-assertion latency of the empty signal does not lead to a blocking of the writer (which should wait till the reader resumes data consumption from the FIFO) due to the large availability of empty buffer slots in a deep FIFO.

By construction, we verified that the proposed dual-clock FIFO architecture guarantees 100% throughput when a FIFO depth of 5 slots is set, regardless of the frequency ratio between sender and receiver.

However, an interesting trade-off exists between throughput and FIFO depth (and consequently area and power footprint). Therefore, we now investigate what happens when the FIFO depth is configured to be lower than 5 to save area. Figure 5(b) shows the system behavior when considering a FIFO depth of 4 and a *tx_clk* frequency similar (but higher) to the *rx_clk* frequency. During T2, value of the pointer is $WPi = RPi + 1$ and *full_tmp* is asserted, thus setting *full_set*. During T3, the token ring will receive a high *tx_full* and will interrupt the data write. Consequently to *read_pointer* shifting, *full_set* and *tx_full* are de-asserted respectively during T4 and T5. During T5, token ring in the sender's domain restarts to shift regularly the *write_pointer*. Since we have a FIFO depth of 4, during T5 the value of the pointer is $WPi = RPi$ and *empty_tmp* is asserted. Although transmitter frequency is higher than the receiver one, the FIFO generates an *empty_tmp* inducing a logic high *rx_empty* and a suspension of the regular read operation. This effect is clearly undesired and causes a reduction of throughput. In this condition, the FIFO is not able to deliver one word per cycle and the throughput is around 50%.

On the contrary, the FIFO will produce different results when considering a lower *rx_clk* frequency. Assuming the frequency ratio scenario showed in Figure 5(c) and a FIFO depth of 4, this time we obtain a throughput of 100% In fact, during T2, the value of the pointer is $WPi = RPi + 1$ and *full_tmp* is asserted. In any case, during T5, when the token ring in the sender's domain restarts to shift regularly the *write_pointer*, the value of the pointer is not $WPi = RPi$ and *empty_tmp* is not asserted.

Therefore, the FIFO throughput directly depends on the relative frequency between domains. Notice that examples could be envisioned by considering *rx_clock* frequency higher than the *tx_clock* one, thus obtaining symmetrical throughput results. Summing up, a FIFO with depth of 4 has 100% throughput if the sender clock cycle time (T_{tx}) and receiver clock cycle time (T_{rx}) meet one of these requirements:

$$3 \times T_{tx} < 2 \times T_{rx} \tag{1}$$

$$2 \times T_{tx} > 3 \times T_{rx} \tag{2}$$

As a result, the proposed FIFO with depth 4 guarantees 100% throughput when the transmitter module works with a frequency 33% lower or 50% higher than the receiver module. In presence of different frequency ratios, throughput is between 50% and 100%.

For the sake of further area and power optimizations, we analyzed the proposed FIFO with a depth of 3 slots. Following previous deductions, this solution guarantees a throughput of 100% in case the transmitter is three times faster or slower than the receiver. In the remaining cases, the throughput is between 50% and 100%. Table II sums up the throughput results as a function of FIFO depth and frequency ratio scenarios.

VI. SPECIALIZED LIBRARY COMPONENTS

The dual-clock FIFO architecture can be specialized in order to reduce latency and area while sustaining throughput. In particular, an architecture specialization can be envisioned when synchronizing data from a sender domain which is permanently faster than the receiver one (Figure 6). In this architecture, the *valid_in* signal is sampled by the *data_buffer* as an additional data wire. This way, *data_in* and *valid_in* cross together the dual-clock FIFO without any additional logic needed to generate *valid_out*. Moreover, data is sampled by *data_buffer* in every possible condition, also when *valid_in* is low. Since an empty assertion can only occur when the sender is slower than the receiver, it is not possible to have an empty condition in our considered scenario. Therefore, the *empty_detector* is not required in this architecture.

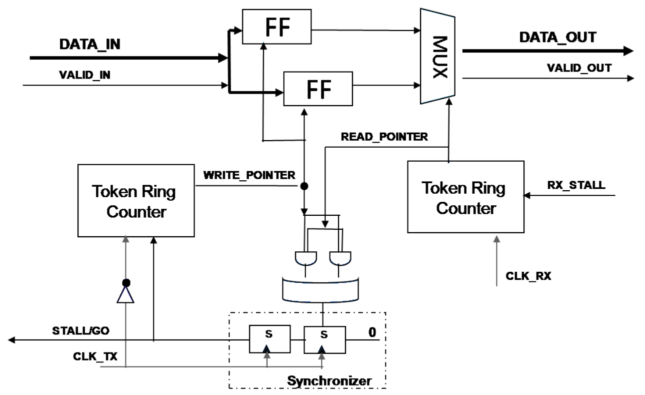
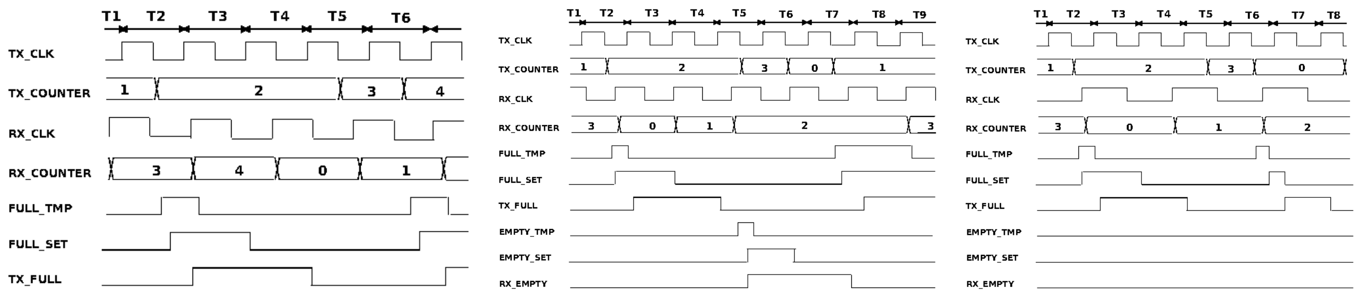


Fig. 6. Specialized Dual-Clock FIFO.

This way, considering a scenario with the sender faster than the receiver and the specialized FIFO with a depth of 4, it is possible to achieve 100% throughput also when the condition in Formula 1 is not met. In fact, since no empty condition has to be detected, this architecture allows to read a safely sampled data in slot i also in case the $write\_pointer$, waiting for full de-assertion, has not shifted from i to $i + 1$ position yet. Therefore, buffer resources are optimized and performance is preserved. In particular, 1 slot buffer is saved while guarantying the same throughput with respect to the architecture of Figure 2. Table III reports area and throughput



(a) FIFO depth of 5, tx faster than rx. (b) FIFO depth of 4, rx and tx freq. are similar. (c) FIFO depth of 4, rx slower than tx.

Fig. 5.

	FREQUENCY SCENARIO	FIFO DEPTH OF 5	FIFO DEPTH OF 4	FIFO DEPTH OF 3
I°	$3 \times f_{tx} > f_{rx}$	100%	100%	100%
II°	$1.5 \times f_{tx} > f_{rx}$	100%	100%	$50\% - 100\%$
III°	$f_{tx} \geqslant f_{rx}$	100%	$50\% - 100\%$	$50\% - 100\%$
IV°	$f_{rx} > f_{tx}$	100%	$50\% - 100\%$	$50\% - 100\%$
V°	$1.5 \times f_{rx} > f_{tx}$	100%	100%	$50\% - 100\%$
VI°	$3 \times f_{rx} > f_{tx}$	100%	100%	100%

TABLE II
DUAL-CLOCK FIFO THROUGHPUT WITH PARAMETERIZED BUFFER DEPTH AS A FUNCTION OF SENDER-RECEIVER FREQUENCY RATIO.

	FREQUENCY SCENARIO	FIFO DEPTH OF 4	FIFO DEPTH OF 3	FIFO DEPTH OF 2
I°	$3 \times f_{tx} > f_{rx}$	100%	100%	100%
II°	$1.5 \times f_{tx} > f_{rx}$	100%	100%	-
III°	$f_{tx} \geqslant f_{rx}$	100%	-	-

TABLE III
THROUGHPUT OF SPECIALIZED DUAL-CLOCK FIFO VARIANTS.

results of the architecture depicted in Figure 6 as a function of buffer depth and frequency ratio of the end-nodes.

Please note that a similar optimization is not feasible when the receiver is permanently faster than the sender domain. The reason is that, the *full_detector* is still required because it is not possible to guarantee the absence of the full condition. In fact, when a contention takes place in the switch block, an input may loose its grant (from the arbiter) and consequently become stalled. In this case, a slower sender may succeed in filling up the dual-clock FIFO buffer, therefore, the *full_detector* is necessary to interrupt the communication.

VII. EXPERIMENTAL RESULTS

Experimental results are structured into three subsections. In the first one, post-place&route results are presented documenting critical path differences (if any) between fully synchronous switches and switches with integrated dual-clock FIFOs at their inputs. In the second one, area and power benefits of the tightly coupled design principle applied to dual-clock FIFOs are quantified, while in the third one the implementation overhead for the different components in the synchronization library is characterized in relative terms. All physical synthesis experiments have been performed with a 65nm STMicroelectronics technology library.

A. Impact of NoC-synchronizer merging on the switch critical path

The switch configurations illustrated in Table IV were synthesized, placed and routed. The first one is a fully synchronous switch with a 2 slot input buffer and a 6 slot output buffer. Moreover, the input buffer of the same switch is merged with a high-throughput dual-clock FIFO, thus augmenting the input buffer storage to 5 slots. Finally, two remaining switch configurations implement 6 slot buffers in each input port: a fully synchronous one and one with merged dual-clock FIFOs. The last column of Table IV reports the resulting maximum operating frequency.

	INPUT BUFFER	OUTPUT BUFFER	FREQUENCY
I°	2 fully synchronous	6 fully synchronous	1.43GHz
II°	5 bi-synch FIFO	6 fully synchronous	1.25GHz
III°	6 fully synchronous	2 fully synchronous	1.2GHz
IIII°	6 bi-synch FIFO	2 fully synchronous	1.2GHz

TABLE IV
2X2 SWITCH CRITICAL PATH.

The first configuration allows the switch to work at the highest frequency of 1.43GHz. In the second configuration, the switch, having the integrated FIFO synchronizer, features a lower frequency (1.25GHz). This result depends on the fact that the integration of the dual-clock FIFO has shifted the critical path from the switch crossbar and arbitration logic to the dual-clock FIFO itself in the input stage (see Figure 2).

In the third configuration, the switch performance decreases to 1.2GHz. This is due to the fact that the delay of the finite state machine in the input buffer is larger depending on the number of buffer slots, and it adds up to the propagation delay through the arbiter and the crossbar selection signals. Interestingly, the equivalent switch configuration with multiple clock domain support achieves the same speed and features the same critical path. As a result, the fourth configuration supports multiple clock domains while guaranteeing the same critical path of the fully synchronous switch having an equivalent overall amount of buffer storage.

The key take-away is that the tightly coupled FIFO synchronizer can determine the critical path in low radix switches (e.g., 2x2) when these latter could afford a speed higher than 1.25GHz with a typical fully synchronous input buffer. However, please notice that most topologies in use for NoC design typically require a larger number of switch I/Os.

In the following analysis we implement a 5x5 switch (used to build up a 2D mesh topology) in 2 different configurations. In particular we compare one fully synchronous switch configuration with one switch configuration integrating the FIFO synchronizers. Input and output buffers have the same size in both designs. Results are illustrate in Table V.

	INPUT BUFFER	OUTPUT BUFFER	FREQUENCY
I°	6 fully synchronous	2 fully synchronous	830Mhz
II°	6 bi-synch FIFO	2 fully synchronous	830Mhz

TABLE V
5X5 SWITCH CRITICAL PATH.

The relevant result here is that, the increased switch radix decreases the maximum operating speed, which ends up falling below the 1.25GHz threshold beyond which the dual-clock FIFO behaves as speed limiter. Below this threshold, the critical path moves somewhere else, hence the support for multiple clock domains does not bring any limitation to the maximum achievable performance.

B. Area and power benefits of NoC-synchronizer merging

Three different designs have been compared. The first one is the conventional 5x5 vanilla (fully synchronous) switch with a 6-slot input buffer and a 2-slot output buffer per port. The second one is a switch where a dual-clock FIFO with 6 buffer slots has been merged with each input port. In order to carry out a fair comparison with the vanilla switch, total buffering resources have been kept equal, i.e., the output buffer size in the switch with the FIFO synchronizers has been reduced from 6 to 2 slots. The last configuration is a vanilla switch (6-slot inputs, 2-slot outputs) with external dual-clock FIFO (6 buffer slots) per input port.

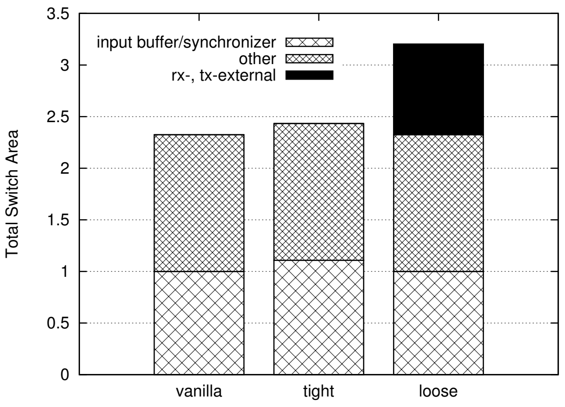
To assess area occupancy, all the above switch configurations have been synthesized, placed and routed at the same target frequency of 1GHz. Total area of the tightly coupled system exhibits almost the same area footprint of the vanilla switch. This is a direct consequence of the fact that exactly the same buffering resources have been deployed in a specular fashion (between input and output).

As showed in Figure 7(a), being the input buffer size of the three systems the same (6 slots), there is a similar amount of cell area devoted to either only buffering (vanilla switch) or buffering and synchronization (tightly and loosely coupled switch). Moreover, the loosely coupled system features the same area overhead (with the same distribution of *input_buffer* and *other* cell area) of the other switches plus a further synchronization area due to the external block implementing the dual-clock FIFO.

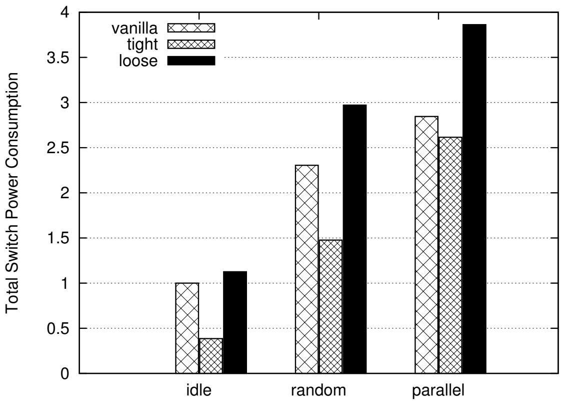
These results point that the merging approach applied to the dual-clock FIFO design achieves up to 24% of area saving with respect to the loosely coupled design methodology.

To assess the power consumption of a switch integrating dual-clock FIFOs on the input ports, the vanilla, tightly and loosely coupled designs have been tested under different traffic patterns: idle (to measure standby power), random (target output port of input packets is randomly selected) and parallel (no switch internal conflicts). Post-layout simulations have been carried out at 800MHz. The switch with the external dual-clock FIFO is the most power greedy under all possible traffic patterns, as showed in Figure 7(b). 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 efficient among those under test and achieves up to 51% power saving (under random traffic).

The motivation lies in the inherent clock gating that 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 thus 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



(a) Area breakdown.



(b) Power consumption.

Fig. 7. Post-layout normalized results of area (a) and power (b) for a switch with a dual-clock FIFO synchronizer.

at least 6 flits are used in all switch variants for performance optimization.

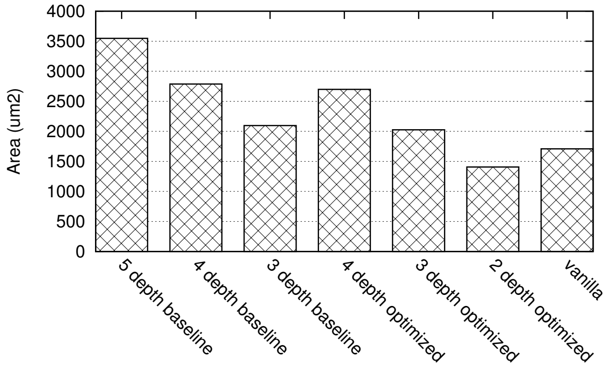
Above all, these results indicate that with the proposed architecture design techniques it is possible to evolve a fully synchronous switch to a switch supporting relaxation of synchronization assumptions with marginal implementation overhead. This is a key enabler for the GALS paradigm in the context of NoC-centric MPSoCs.

C. Characterization of synchronization library components

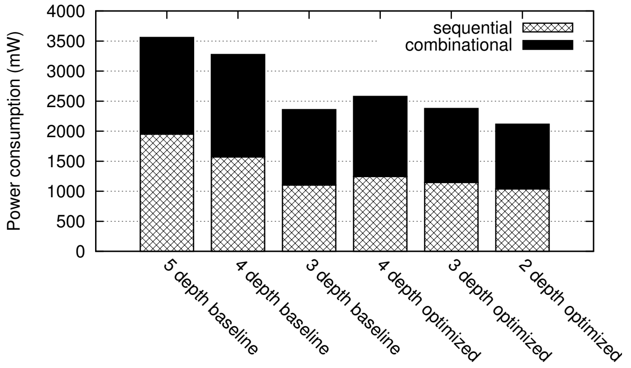
In this experiment a comparison between the baseline dual-clock FIFO architecture and its specialized version has been carried out. By varying buffer depth of each solution, we could span the entire range of components of the proposed synchronization library.

Buffer depth of the baseline architecture ranges from 5 to 3 slots while supporting all possible frequency ratio scenarios. On the other hand, the specialized version requires from 4 to 2 buffer slots and is able to work with all the sender/receiver frequency ratio where the sender is always faster than the receiver. Furthermore, for the sake of comparison, a fully synchronous input buffer (with minimum number of slots, i.e. 2) has been considered.

As reported in Figure 8(a), when the buffer depth is reduced in both the architectures, a corresponding reduction of overall cell area takes place. The reason lies in a reduction of both sequential and logic cell area. In fact, by reducing the buffer size, sequential elements are obviously reduced along with a simplification of the detector and all the combinational logic. It is interesting to note that when considering the two architectures with the same buffer depth (e.g., 4) there is a marginal area reduction of the specialized version. This is



(a) Area.



(b) Power consumption.

Fig. 8. Area (a) and power (b) consumption of baseline and specialized dual-clock FIFO architectures with different buffer depths.

due to the absence of the empty detector which is no longer required. A further consideration stems from a comparison between the vanilla input buffer (2-slots) and the equivalent size specialized dual-clock FIFO. Of note, the dual-clock FIFO is more area efficient than the vanilla input buffer because the FSM taking care of *valid* signals is simplified.

Power consumption has been computed by considering all the architectures operating at the same frequencies that permit all them to correctly work while supporting full throughput communication. Experiments have been performed with a sender four times faster than the receiver. Sender clock frequency has been set to 1GHz. All the architectural variants have been tested under parallel traffic patterns.

As reported in Figure 8(b), a reduction of the buffer depth implies a corresponding reduction of power consumption in both baseline and optimized architectures. Interestingly, by comparing the result of the two architectures with the same buffer depth of 3, it is possible to note a similar power consumption. Conversely, when comparing the baseline and the optimized architectures with a buffer depth of 4, it is clear that a higher consumption takes place for the baseline one. In fact, the baseline dual-clock FIFO with 4 and 5 buffer slots requires a larger token ring counter due to the high fanout net. This is not an issue in the optimized FIFO as there is no empty detector.

VIII. Conclusions

In this paper, the idea of dual-clock FIFOs merged with the input buffers of Network-on-Chip switching fabrics has been proposed and its implementation illustrated in detail. Benefits of this design philosophy have been proved in terms of area, power and latency while demonstrating the marginal impact on switch critical path.

Furthermore, several variants of the aforementioned synchronizer architecture have been developed by varying configuration parameters and/or specializing the architecture for well-defined ranges of operating conditions. All together, these components build up a library of synchronization interfaces that can be instantiated in a plug-and-play fashion so to meet performance requirements and operating conditions (frequency ratio of sender and receiver clock) at the minimum area and power cost. From this viewpoint, the presented library is a key step in the direction of a design automation tool for automatic instantiation of modern MPSoCs.

Moreover, the design techniques of synchronization interfaces presented in this work enable to replace a fully synchronous switch with an augmented variant supporting the relaxation of synchronization assumptions at marginal implementation overhead, thus making GALS technology affordable for NoC-centric MPSoCs.

Acknowledgements

This work has been partially supported by the GALAXY European Project (FP7-ICT-214364) and by the Hipeac Network of Excellence (Interconnect Cluster).

References

[1] L.Benini and G.De Micheli, "Networks on chip: a new SoC paradigm". IEEE Computer, 35(1):70-78, January 2002.
[2] T.Ono, M.Greenstreet, "A Modular Synchronizing FIFO for NOCs", Proceedings of International Symposium on Networks-on-Chip (NOCS), 2009
[3] T.Chelcea, S.M.Nowick, "Robust Interfaces for Mixed-Timing Systems", IEEE Transactions on Very Large Scale Integration Systems, 12(8): 857-873, 2004.
[4] D.Ludovici, A.Strano, G.N.Gaydadjiev, L.Benini, D.Bertozzi, "Design Space Exploration of a Mesochronous Link for Cost-Effective and Flexible GALS NOCs", Proceedings of Design, Automation and Test in Europe (DATE'10), pp. 679–684, Dresden, Germany, 2010.
[5] C.Cummings, P.Alfke, "Simulation and Synthesis Techniques for Asynchronous FIFO Design with Asynchronous Pointer Comparison", SNUG-2002, San Josè, CA, 2002.
[6] A.Edmanand, C.Svensson, "Timing Closure through Globally Synchronous, Timing Portioned Design Methodology", Proceedings of Design and Automation Conference (DAC), pp.71–74, 2004.
[7] P.Caput, C.Svensson, "An On-Chip Delay- and Skew-Insensitive Multicycle Communication Scheme", IEEE Solid-State Circuits Conference (ISSCC), pp.1765–1774, 2006.
[8] I.M.Panades, A.Greiner, "Bi-Synchronous FIFO for Synchronous Circuit Communication Well Suited for Network-on-Chip in GALS Architectures", Proceedings of International Symposium on Networks-on-Chip (NOCS), pp.83–94, 2007.
[9] D.Ludovici, A.Strano, D.Bertozzi "Architecture Design Principles for the Integration of Synchronization Interfaces into Network-on-Chip Switches", Proceedings of 2nd. International Workshop on Network on Chip Architecture (NoCArc), pp.31–36, New York City, NY, 2009.
[10] 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.
[11] I.Loi, F.Angiolini, L.Benini, "Developing Mesochronous Synchronizers to Enable 3D NoCs", Proceedings of International Conference on VLSI Design, 2007.
[12] 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.
[13] S.Stergiou, F.Angiolini, S.Carta, L.Raffo, D.Bertozzi, G.De Micheli, "XPipes Lite: a Synthesis Oriented Design Library for Networks on Chips", Proceedings of Design, Automation and Test in Europe (DATE'05), pp.1188–1193, 2005.
[14] F.Angiolini, L.Benini, P.Meloni, L.Raffo, S.Carta, "Contrasting a NoC and a Traditional Interconnect Fabric with Layout Awareness", Proceedings of Design, Automation and Test in Europe (DATE'06), March 2006.
[15] "Specification of optimized GALS interfaces and application scenarios", GALAXY Project deliverable D3, online at http://www.galaxy-project.org/publ_deliv.html
[16] J.Ebergen, "Squaring the FIFO in GasP", Proceedings of International Symposium on Asynchronous Circuits and Systems, pp.194–205, 2001.
[17] C.E.Molnar, I.W.Jones, W.S.Coates, J.K.Lexau, "A FIFO ring performance experiment", Proceedings of International Symposium on Asynchronous Circuits and Systems, pp.279–289, 1997.
[18] R.Apperson, Z.Yu, M.Meeuwsen, T.Mohsenin, B.Baas, "A scalable dual-clock FIFO for data transfers between arbitrary and haltable clock domains", IEEE Transactions on Very Large Scale Integration (VLSI) Systems, 15(10), pp.1125–1134, 2007.