Hybrid-Timing FIFOs to use on Networks-on-Chip in GALS Architectures

A. SHEIBANYRAD
The University of Pierre et Marie Curie (LIP6)
abbas.sheibanyrad@lip6.fr

A. GREINER
The University of Pierre et Marie Curie (LIP6)
alain.greiner@lip6.fr

Abstract. *This paper presents three high-throughput low-latency FIFOs that can be used as efficient and reliable interfaces between different domains in hybrid-timing systems. These three hardware components have been designed to be used in a Globally Asynchronous Locally Synchronous clusterized Multi-Processor System-on-Chip communicating by a Multi-Synchronous or by a fully Asynchronous Network-on-Chip. The proposed architectures are rather generic and allow the system designer to make various trade-off between latency and robustness, depending on selected synchronizer. These FIFOs have been physically implemented with portable ALLIANCE CMOS standard cell library and the architectures have been evaluated by SPICE simulation for a 90nm CMOS fabrication process.*

1. Introduction

NoCs (Networks-on-Chip) is a new design paradigm for scalable, high-throughput communication infrastructure, in Multi-Processor Systems-on-Chip (MP-SoCs) with billions of transistors. The idea of NoC is dividing a chip into several independent subsystems (clusters) connected together by a global communication architecture.

Because of physical issues in nanometer fabrication processes, it is not anymore possible to distribute a synchronous clock signal on the entire wide chip area. NoCs using Globally Asynchronous Locally Synchronous (GALS) techniques address this difficulty.

Several solutions have been proposed to resolve the problems of clock boundaries and the risk of synchronization failures (metastability). Related solutions [1, 2, 3, 4, 5, 6, 7, 8, and 9] vary according to the constraints that the clock signal must respect in different clusters: mesochronous, plesiochronous, pseudochronous (quasi-synchronous), and heterochronous approaches correspond to various hypothesis regarding the phases and frequencies of clocks signals. Table 1 summarizes these conditions.

Table 1. Timing Dependency Methods

Type	Δ Frequency	Δ Phase
Synchronous	0	0
Pseudochronous	0	Constant
Mesochronous	0	Undefined
Plesiochronous	ε	ε
Heterochronous	Rational	Undefined
Multi-synchronous	Undefined	Undefined
Asynchronous	-	-

In MP-SoC design, a fundamental challenge is the capability of operating under totally independent timing assumptions. Such a multi-synchronous system contains several synchronous subsystems clocked with completely independent clocks. The NoC architecture defines global asynchronous communication infrastructure.

In section 2 we talk about two different NoC approaches to comply with multi-synchronous constraint and we explain how in such architectures we can instantiate some special FIFOs to robustly interface hybrid-timing domains. The main purpose of this article is to present architectures of three needed components: synchronous-to-asynchronous, asynchronous-to-synchronous, and bi-synchronous FIFOs.

We present Synchronous ⇔ Asynchronous FIFOs in section 3. The design of Bi-Synchronous FIFO is described in section 4. And finally we explain hardware implementations in section 5.

2. NoCs and GALS Architectures

To simplify physical implementation, most of NoCs have a two-dimensional mesh topology. Routers (switching modules) are distributed in each subsystem, and are connected to north, south, east, and west neighbors by means of bidirectional point-to-point links. A generic subsystem is connected to NoC through a Network Interface Controller (NIC) which translates local interconnect communication protocol to network protocol.

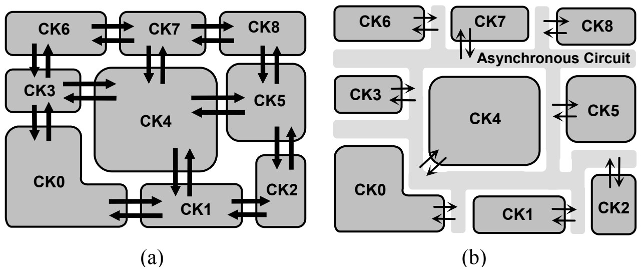

Fig. 1. Multi-Synchronous System in 2D Mesh Topology

In a multi-synchronous system arranged into 2D mesh (Fig. 1a) each router is a synchronous circuit. Router clock signal can be local subsystem clock, or can be a dedicated (possibly faster) mesochronous clock signal. The use of a fast dedicated clock for routers reduces the network latency, but complicates the design, as it creates another clock boundary between routers and subsystems. The possibility of synchronization failure (metastability) between two adjacent routers (or between routers and subsystems if routers use a dedicated clock signal) is the main problem of GALS architectures using multi-synchronous NoCs.

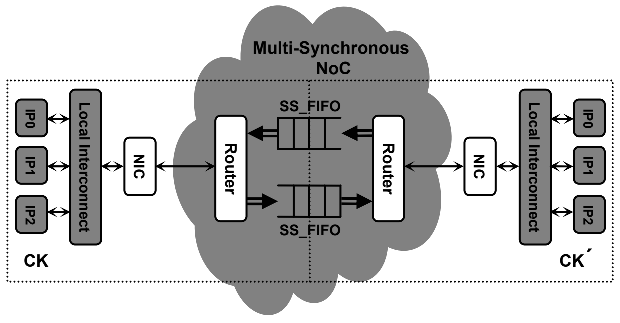

Fig. 2. Bi-synchronous FIFOs between two adjacent clusters

DSPIN [28] is an example of clusterized multi-synchronous NoCs. DSPIN uses bi-synchronous FIFO as robust interfaces between adjacent subsystems. The producer side and consumer side of a bi-synchronous FIFO (SS_FIFOs in Fig. 2) use different clock signals. In [4, 5, 10, and 11] several authors have proposed different types of bi-synchronous FIFOs. This paper presents the architecture of a new fast one.

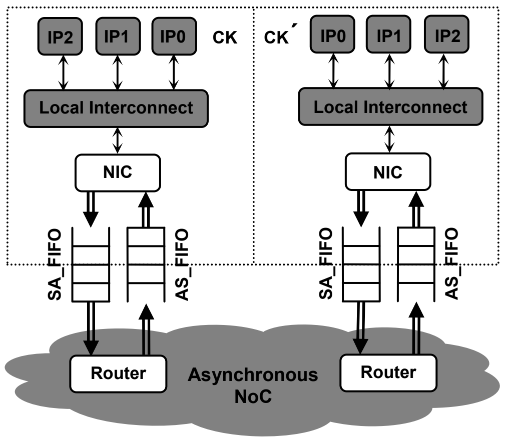

Fig. 3. AtoS and StoA FIFOs as interface between NoC and clusters

We illustrate in Fig. 1b an alternative solution: The global interconnect has a fully asynchronous architecture. This type of NoCs realizes the GALS approach by providing synchronous interfaces to each local subsystem. As an example we can denote *MANGO* architecture presented in [26] which is one of the first asynchronous NoC. *MANGO*'s designers have proposed in [27] an OCP Compliant Network Adapter (NA) interfacing local subsystems to NoC. The synchronization in NA has a minimized overhead.

To robustly interface an asynchronous network to synchronous subsystems on a chip, as presented in Fig. 3, we can use two special types of FIFO: Synchronous-to-Asynchronous FIFO (SA_FIFO) and Asynchronous-to-Synchronous FIFO (AS_FIFO). Using FIFOs to interface hybrid-timing domains couples two fundamental issues which need to be considered in designing such interface: flow control (high level issue) and clock domain resynchronization (low level issue). This coupling reduces the need of hardware synchronizer to the handshake signals that are used for flow control.

The designs of some AS and SA FIFOs are presented in [10, 14, 15, 16, and 17]. We present here the architecture of new Synchronous ⇔ Asynchronous FIFOs which are currently used by *ASPIN* micro-network presented in [29].

Some of published solutions for interfacing hybrid-timing domains are strongly dependent on synchronizer choice. In [15 and 16] designs of various types of synchronizers using pausible clocking methods ([12]) are proposed and in [17] the authors have suggested to generate a stoppable clock for local systems.

In [14] a pipelined synchronizing FIFO is proposed. This FIFO requires that the producer produces data at a constant rate. The latency of this design is proportional to the number of FIFO stages and requires using of a specific synchronizer.

In [10] various FIFOs are used to interface four possible combinations of independent mixed-timing environments. These four FIFOs have the same basic architecture with small differences of simple adaptation to the consumer and the producer interface type. There is at least two weak points in this proposal: the architecture depends on a specific synchronizer (the cascaded Flip-Flops), and use of a more conservative synchronizer (with latency larger than one clock cycle), can decrease the throughput to a value less than one data transfer per cycle. Furthermore, the authors of [10] don't talk about silicon area, but we believe that architectures proposed in the present paper have a smaller foot-print.

3. Synchronous ⇔ Asynchronous FIFOs

The design of Synchronous ⇔ Asynchronous FIFO must satisfy two main requirements: minimizing the probability of metastability, and maximizing the data transfer rate. In this section we illustrate how these two aims are achieved.

3.1. Synchronizer: the Latency/Robustness Trade-Off

Transferring data between different timing domains requires safe synchronization which is the main issue of GALS paradigm. The aim of synchronization is to prevent metastability.

Some authors recommend stretching the clock signal (modifying the cycle time). In these methods, instead of synchronizing asynchronous inputs with the clock, the clock is synchronized with asynchronous inputs. The synchronizer must be able to detect that it will be in the metastable situation and it stretches the clock cycle of the local system until the probability of metastability is zero. For more than one asynchronous input, the clock must be stretched until all synchronizers insure that the metastable states won't occur. Consequently, as it is said in [7 and 18], these solutions are not well suited for high speed designs with IP cores having large clock buffer delays.

Some others suggested modifying the Flip-Flop design to avoid propagation of metastable state values ([19]). Although the output of such circuit has well defined values (VDD or VSS) and undesirable values are prevented to propagate, this does not solve the problem: the precise duration of the metastable state remains unpredictable. The transition of Flip-Flop outputs is asynchronous compared with the clock signal of next Flip-Flop....

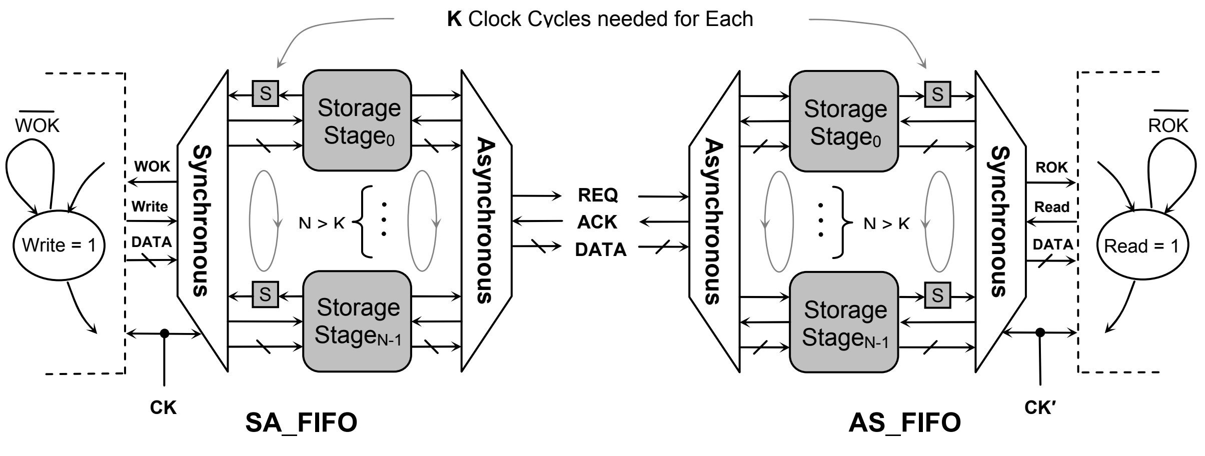

Fig. 4. Synchronous ⇔ Asynchronous FIFOs

The metastability in multi-synchronous systems can not be totally suppressed, but as it is explained in [21], the synchronization failure probability (typically expressed in terms of Mean Time Between Failures or MTBF) can be bounded to an acceptable value by a carefully designed synchronizer ([22 and 23]). The simplest and safest solution is to use several cascaded Flip-Flops. According to [21], with two cascaded Flip-Flops with 200 MHz clock frequency and 20 MHz input data rate, for the 0.18 μm technology MTBF can be estimated to about 10^{204} years. For three consecutive Flip-Flops in the same condition, MTBF will be 10^{420} years!

Increasing synchronization delay is a penalty for obtaining extra safety: When synchronization latency is not an issue, MTBF can be improved by using conservative synchronizers. We believe that the synchronizer choice must be a design decision depending on the application requirements. The general architecture of our FIFOs will support trade-off between latency and robustness.

3.2. General Architecture

Fig. 4 shows the general architecture of the two FIFOs converting asynchronous Four-Phase Bundled-Data protocol to synchronous FIFO protocol and vice versa. The Four-Phase Bundled-Data asynchronous protocol is a sequence of REQ+, ACK+, REQ-, and ACK- events, where REQ and ACK are the asynchronous flow control signals. Data is valid when REQ is positive. The high level of ACK indicates that the request of data communication is accepted.

In synchronous FIFO protocol, the producer and the consumer share the same clock signal. The protocol uses two handshake signals: ROK (correspondingly WOK) and READ (correspondingly WRITE). ROK signal (not empty) is set by the producer at each cycle where there is a valid data to be transferred. READ signal is set by the consumer at each cycle where the consumer wants to consume a data on next clock edge. Both ROK and READ signals are state signals generated by Moore FSMs.

We call AS_FIFO, the asynchronous-to-synchronous FIFO, and SA_FIFO, the synchronous-to-asynchronous FIFO. The task of protocol converting is performed by storage stages of FIFOs. Signals which have a risk of metastability (and must use a synchronizer) are handshake signals transmitted from asynchronous side to synchronous side.

As it is said in previous section, the synchronizer design is a trade-off between robustness (i.e. low probability of metastability) and latency (measured as a number of cycles of synchronous domain). If synchronization cost is K clock cycles, for a throughput of one data transfer per cycle, FIFO must have at least K+1 stages. In such a pipelined design, the effect of synchronization latency is different in the two FIFO types.

In asynchronous-to-synchronous FIFO (AS_FIFO), the synchronizer latency is visible only when FIFO is empty. In synchronous-to-asynchronous FIFO (SA_FIFO), it is visible when FIFO is full. The latency between the arrival of data to an empty AS_FIFO and its availability on output (typically named FIFO Latency) is about K clock cycles. For a full SA_FIFO, the latency between data consumption and informing the availability of an empty stage on the other side is about K clock cycles. For a data burst these latencies are just the initial latencies.

3.3. Detailed Architecture

Fig. 5a and 5c show the internal architecture of SA_FIFO and AS_FIFO, with a depth of 2 storage stages. Clearly, these two architectures can be generalized to an n-stage FIFO.

We present in Fig. 5b and 5d the FSMs of synchronous side controllers. These controllers are Mealy Finite State Machines. State W_i means that the next WRITE event will be done to stage i. Similarly, state R_i indicates that data will be read in stage i at the next READ event. Consequently, WOK and ROK signals depend on both FSM state and asynchronous stage content (signals WOK_i or ROK_i). A synchronous hazard free command is generated ($WRITE_i$ or $READ_i$) when there is a synchronous request (WRITE or READ signals) and the current asynchronous stage is ready to accept. ROK_i means that stage i is not empty and WOK_i means that stage i is not full. Positive edge of $Write_i$ indicates to ith asynchronous stage of SA_FIFO that the synchronous data must be written. Positive edge of $Read_i$ informs the ith asynchronous stage of AS_FIFO that the stage must be freed, and finally positive edge of $wasRead_i$ means that the synchronous consumer has read data and the stage can change its value.

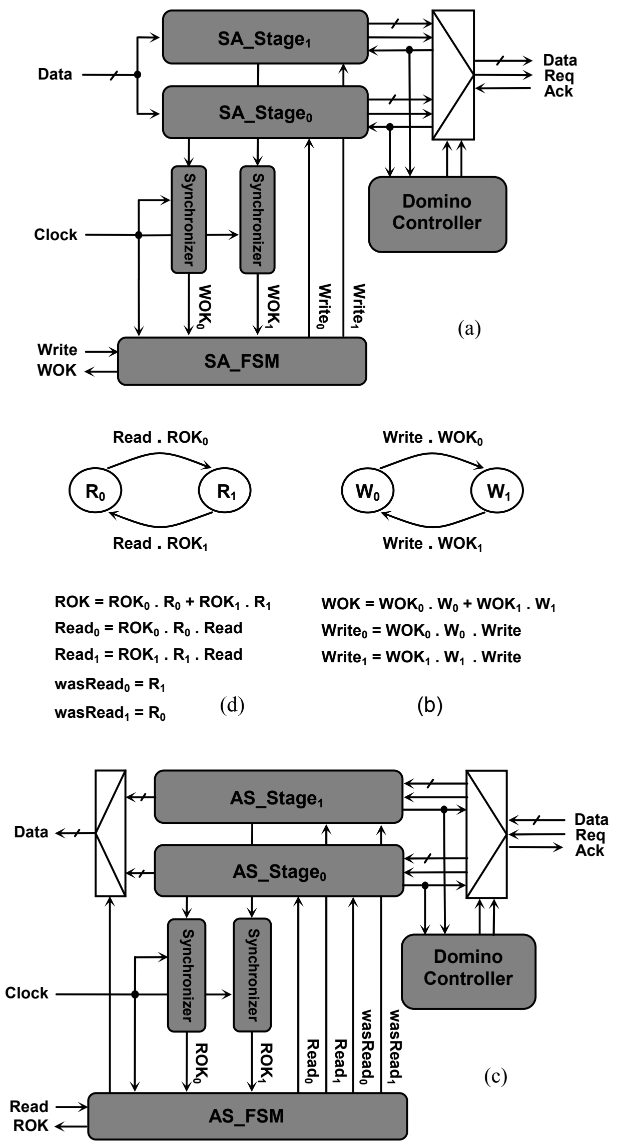

Fig. 5. SA_FIFO and AS_FIFO

As seen in Fig. 5b, $Write_i$ depends on Write, WOK_i and W_i. All these signals are synchronous and will be asserted on the rising edge of the clock. But the Stage considers $Write_i$ as an asynchronous signal and immediately on its rising edge input data will be written. There may be a constraint: The value of data should not stay unstable much after the activation of Write. Although in a general synchronous circuit this constraint is not illogic, to satisfy the conservator designers $Write_i$ can be equal to W_{i+1} (like $wasRead_i$ in Fig. 5d). With this modification the Stage will be ordered to write on the rising edge of the clock; when data is certainly stable. In this case for having a maximum throughput of one data transfer per cycle, the minimum number of stages should be K+2 where K is the synchronizer latency.

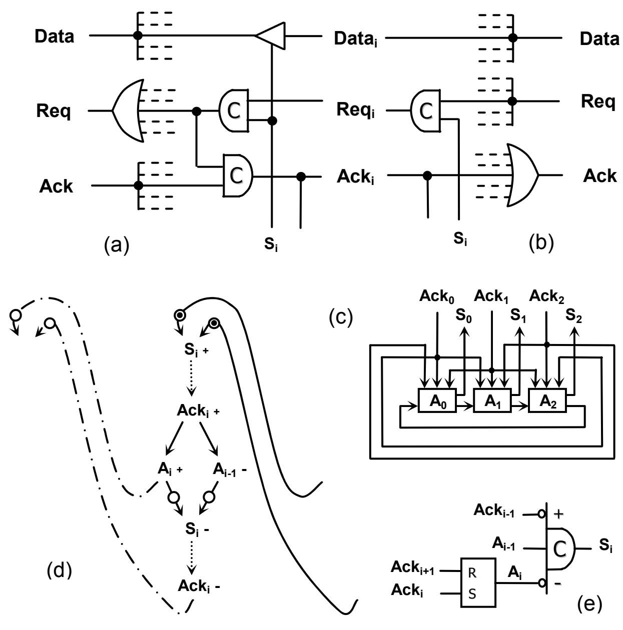

Fig. 6. Asynchronous MUX, DEMUX, and Domino Controller

Asynchronous side of the design includes an asynchronous multiplexer and an asynchronous controller in SA_FIFO. It includes an asynchronous demultiplexer and an asynchronous controller in AS_FIFO. The design of asynchronous multiplexer and demultiplexer using four-phase bundled-data protocol ([13]) are respectively shown in Fig. 6a and Fig. 6b. These circuits need to handshake with their controller module generating the Select signals (S_i). This handshaking brings out with the sequence of S_i+, Ack_i+, S_i-, and Ack_i-. After Ack_i- indicating the end of current four-phase sequence, controller can select another set to multiplexing or demultiplexing.

The asynchronous controller used in AS_FIFO and SA_FIFO is named Domino Controller. It is an asynchronous One-Hot counter providing required handshake protocol for asynchronous multiplexer and demultiplexer. For instance, the block diagram of a 3-bit Domino Controller is illustrated in Fig. 6c. Each cell of i has 2 outputs S_i and A_i (ith bit of the counter) and 4 inputs Ack_{i-1}, Ack_i, Ack_{i+1}, and A_{i-1}. The one bit is moved from cell to cell in a ring topology. At the initial state, A_2 and S_0 are 1 and the other outputs are 0. High value of S_0 means that the first asynchronous event will be performed in stage$_0$.

The functionality of Domino Controller is explained by the cell STG (Signal Transition Graph) demonstrated in Fig. 6d. The circuit implementation of the STG is presented in Fig. 6e. Ack_i+ means S_i+ is seen. So, the one bit that is in the previous cell (i-1) can be transferred to current cell (i). The handshake protocol continues by S_i- when the one transferring is ended.

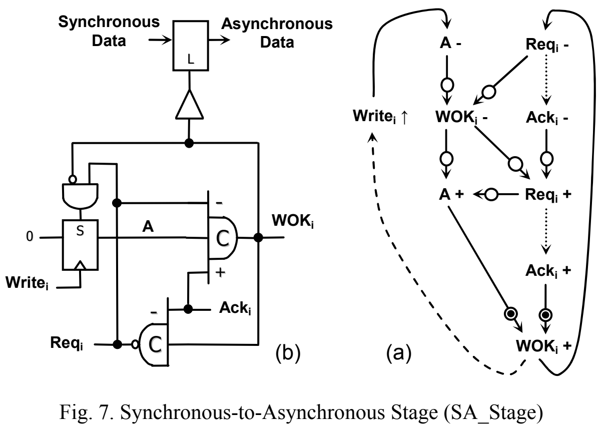

Fig. 7. Synchronous-to-Asynchronous Stage (SA_Stage)

As we said before, the pipelined stages in AS_FIFO and SA_FIFO have two main functionalities: storing data and converting communication protocol. As it is demonstrated in Fig. 7b and Fig. 8b (the schematics of SA_FIFO and AS_FIFO stage circuits) data storage is done by the latches sampling on high value of WOK_i and of L. The transition to 0 of WOK_i means that this stage contains valid data and no more writing is permitted. So data sampling must be ended at this time. When the value of L on the rising edge of $wasRead_i$ (showing that the content of stage has been read) is changed to 1, a new data can be written.

The operation of SA_FIFO and AS_FIFO storage stages are analyzed as two STG in Fig. 7a and Fig. 8a. Dotted lines are the asynchronous side transitions and dashed lines are that of the synchronous side. According to the synchronous protocol base, the synchronous side transitions should be considered on the edges. Regarding to these two STG, on the rising edge of $Write_i$, $Read_i$, and of $wasRead_i$ respectively, A, ROK_i, and C must go to the low position. In the circuits implementation three D Flip-Flops which have a constant value of 0 as input data, generate A, ROK_i, and C. These Flip-Flops will asynchronously be set when their S input (Set) signal is 1.

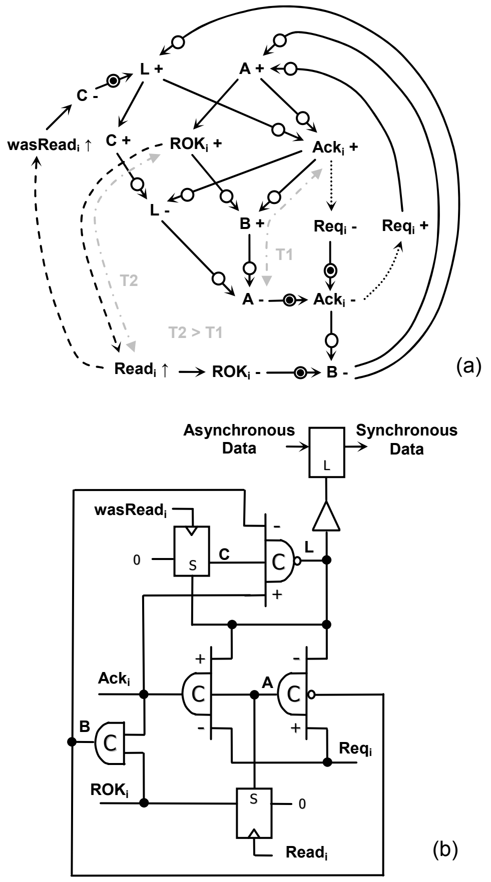

Fig. 8. Asynchronous-to-Synchronous Stage (AS_Stage)

The circuit implementation of AS_FIFO stage shown in Fig. 8b has a time constraint: before the rising edge of $Read_i$ where ROK_i- must be done, the value of A should return to 0. While A (as a set signal of Flip-Flop) has high value, ROK_i (as an output signal of Flip-Flop) is hold at 1. The transition of ROK_i+ causes $Read_i$ to rise. Regarding to AS_FIFO architecture (Fig. 5) the time between ROK_i+ and the rising edge of $Read_i$ (T2) is more than K clock cycles where K is the synchronizer latency. In the other side, A- happens after Ack_i+ occurring simultaneous with ROK_i+, by propagation delay of two gates (T1). Evidently a two-gate propagation delay is less than the latency of a robust synchronizer. The latency of a two cascaded Flip-Flops is one clock cycle. But really it is true that if a designer uses a miraculous synchronizer (!) which has very low latency (less than two-gate propagation delays), this time constraint might express a bother of functionality for the design.

3.4. Improved Architecture of AS_FIFO

As said, the architecture of AS_FIFO presented in the previous subsection has a time constraints. To resolve this restriction, we propose a new STG of AS_Stage shown in Fig. 9a. The circuit implementation, demonstrated in Fig. 9b, is simpler than the previous design.

In this design, the stage will be informed to be freed at the moment it is authorized to accept a new content. $Read_i$ is removed and on the rising edge of $wasRead_i$ the i^{th} stage will be freed (ROK_i goes to Low) and a new data is permitted to be written (L goes to High) to the latches of the stage.

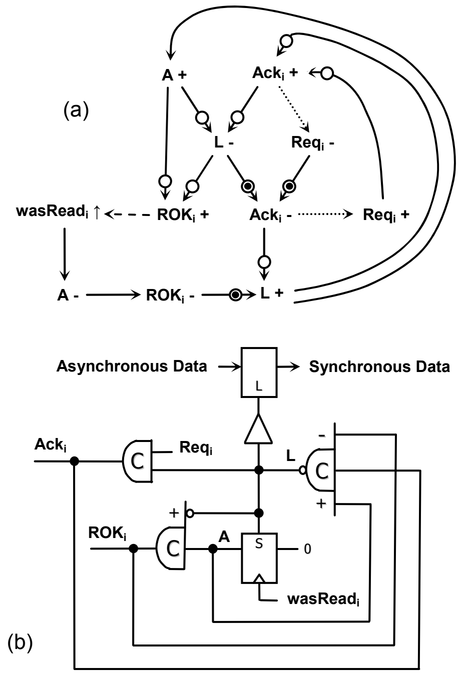

Fig. 9. Improved AS_Stage

Since the stage will be freed after the rising edge of the clock, and not like the previous proposed architecture before the rising edge where data will be read by synchronous side, we need one more clock cycle to perform a read event to the stage. Consequently in order to have maximum throughput of one data transferring per cycle, this new AS_FIFO must have at least K+2 stages; One stage more than the previous architecture.

4. Bi-Synchronous FIFO

In the case of two hybrid clocked domains a bi-synchronous FIFO could be used as interface between these two regions. In Fig. 10 the architecture of a Bi-Synchronous FIFO is presented. The FIFO is called SS_FIFO.

The design is based on the architecture principles of SA_FIFO and AS_FIFO. It can be imagined that SS_FIFO is a SA_FIFO and an AS_FIFO which are merged together: The asynchronous sides of SA_FIFO and AS_FIFO (asynchronous multiplexer, demultiplexer, and Domino controllers) are removed and these two FIFOs are connected together stage by stage.

In fact each storage stage of SS_FIFO is composed of one SA_Stage and one AS_Stage. So each SS_FIFO storage stage is able to store two data words. This feature reduces the need of synchronizer. In reality for each two storage place (each stage) we need one synchronizer on the producer side and one synchronizer on the consumer side. The synchronizer need is divided by two.

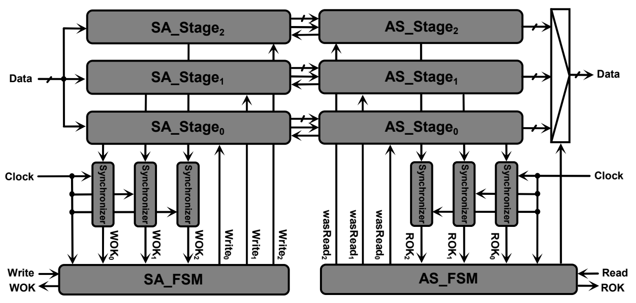

Fig. 10. Bi-Synchronous FIFO (SS_FIFO)

In addition, the complexity of the two FSMs and the output multiplexer is reduced. For an n-place FIFO instead of n states we need n/2 states, for each of two FSMs of SA_FSM and AS_FSM. And in place of an n-to-1 multiplexer we need an n/2-to-1. As a result, the silicon area of an n-place SS_FIFO is much smaller than an n-place AS_FIFO or SA_FIFO. The cost is a little increase in the FIFO latency, as data must pass through two stages to be appeared on the output port.

5. Implementation

We developed a generic FIFO generator, using *Stratus* hardware description language of *Coriolis* platform ([25]). This generator creates both a netlist of standard cells and a physical layout. The two parameters are depth of FIFO (FIFO's places) and number of data bits.

In this implementation the synchronizer uses two cascaded Flip-Flops and its latency is one clock cycle. In order to reach the maximum throughput of one data transfer per cycle, 2-Stage AS_FIFO and 2-Stage SA_FIFO use the constrained architectures.

Table 2. Simulation Results

FIFO	Surface	Min Latency	Max Latency	Max Throughput
2-Place SA_FIFO	1422 μm^2	177 pS		2.39 GEvents/S
3-Place SA_FIFO	2054 μm^2	207 pS		2.36 GEvents/S
8-Place SA_FIFO	5215 μm^2	219 pS		2.22 GEvents/S
2-Place AS_FIFO	1452 μm^2	271 pS + T	271 pS + 2T	1.50 GEvents/S
3-Place AS_FIFO	2011 μm^2	247 pS + T	247 pS + 2T	2.61 GEvents/S
8-Place AS_FIFO	5107 μm^2	263 pS + T	263 pS + 2T	2.89 GEvents/S
6-Place SS_FIFO	2940 μm^2	362 pS + T	362 pS + 2T	2.61 GEvents/S
8-Place SS_FIFO	3869 μm^2	366 pS + T	366 pS + 2T	4.60 GEvents/S

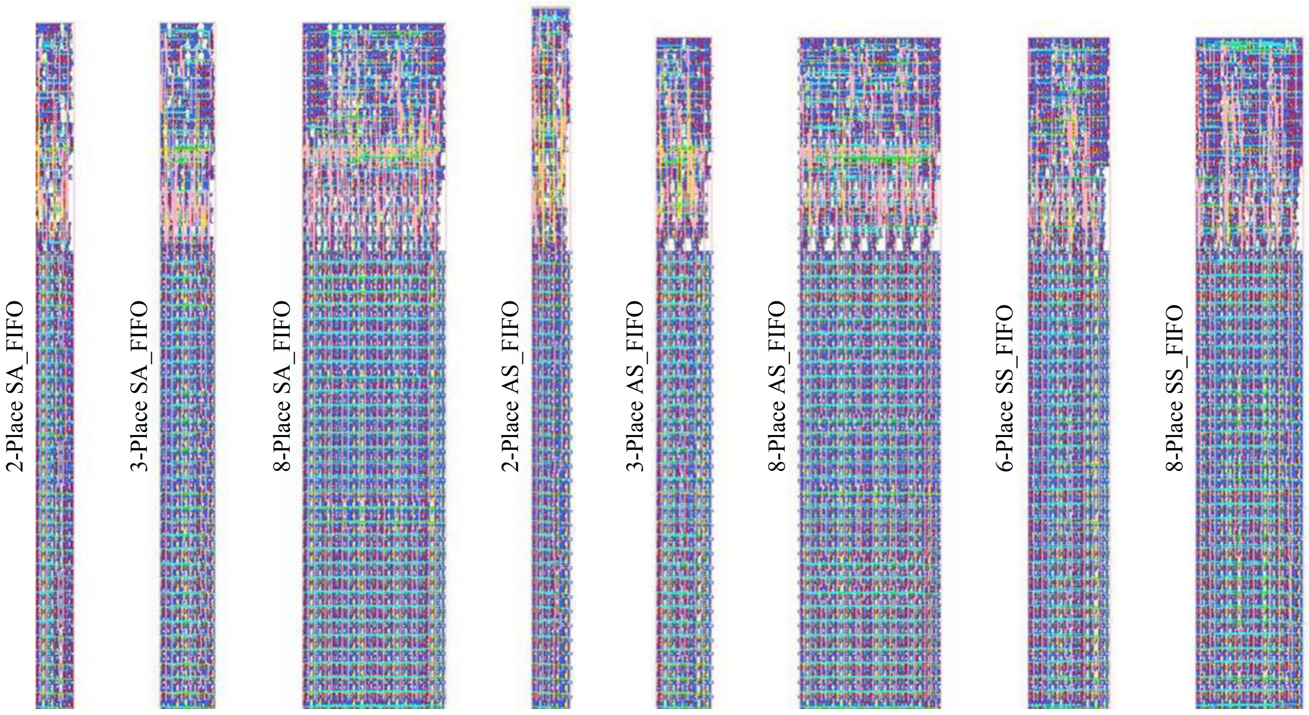

Fig. 11. Physical Layouts

As a standard cell library, we used portable *ALLIANCE* CMOS standard cell library ([24]). The physical layouts of some 32-bit FIFOs are presented in Fig. 11. The silicon areas of these examples are represented in Table 2. These values are normalized to GPLVT ST-Microelectronic library surfaces in 90 nm fabrication process.

From the physical layout, we extracted SPICE models of the FIFOs, using *ALLIANCE* CAD Tools ([20]). The target fabrication process is the ST-Microelectronics 90 nm GPLVT transistors in typical conditions. Electrical simulation under *Eldo* proved that the aim of getting to the maximum throughput of one event (data transfer) per cycle is attained, and these low-area high-throughput FIFOs have low initial latencies.

Due to relation between the asynchronous event entrance time and the consumer clock phase, AS_FIFO has various latencies with a difference of about one clock cycle. Caused by skew relation between the consumer and the producer clocks, SS_FIFO has different latencies too. The simulation results are presented in Table 2. In this Table, T is the consumer clock cycle time.

The SA_FIFO throughput value is related to asynchronous handshake protocol. The throughput of AS_FIFOs with more than 3 stages is limited on asynchronous side components too. But in the case of 2-Place (2-Stage) and 3-Place AS_FIFO there are the other constraints: regarding to Fig. 8 in 2-Place AS_FIFO, Ack_i+ and Req_{i+1}+ must be happened in the same clock cycle if maximum throughput of one data word transferring per cycle is required. Fig. 9 shows that the throughput of one event per cycle for 3-Place AS_FIFO is attained if ROK_i- and ROK_i+ are occurred in the same clock. This constraint should also be respected in 6-Place (3-Stage) SS_FIFO.

Due to the inability of 2-Place AS_FIFO to reach the maximum throughput (comparing 1.5 GEvents/Sec with 2.61 of 3-Place AS_FIFO), in order to sustain the throughput, one could opt for 3-Place AS_FIFO. Its area (2011 μm^2) is not negligible, but it should not be forgotten that these components in addition of robustly interfacing have another advantage: providing a storage place with a FIFO behavior. As we know, in order to obtain minimum overhead of data communication between two different timing domains, having a FIFO in the interface is not eliminable. So, we suppose that using a FIFO with the storage place of more than three may also be reasonable!

Finally, as a quick comparison, the minimum latency of a Mixed-Clock FIFO presented in [10] is 0.5 T_P + 2.5 T_C and its maximum value is 0.5 T_P + 3 T_C where T_P is the producer clock cycle time and T_C is that of the consumer. The max throughput of an 8-bit 4-place Mixed-Clock FIFO is 549 MHz. This evaluation has been given for 0.6 µm HP CMOS technology. The same evaluations for Async-Async, Async-Sync, and Sync-Async FIFOs are respectively 423, 421, and 454.

6. Conclusion

Three new FIFO architectures for interfacing asynchronous NoCs and synchronous subsystems or two adjacent multi-synchronous routers, in MP-SoCs have been presented. The synchronizer used in the architectures can be arbitrarily chosen by the system designer, supporting various trade-off between latency and robustness. The FIFOs can achieve the maximal throughput of one word per cycle, even if the selected synchronizer has a large latency. The designs have been physically implemented with portable *ALLIANCE* CMOS standard cell library. The throughputs and latencies have been proved by SPICE simulation from the extracted layout.

References

[1] Nilsson E., Öberg J., "*Reducing power and latency in 2-D mesh NoCs using globally pseudochronous locally synchronous clocking*," 2nd IEEE/ACM/IFIP international Conference on Hardware/Software Codesign and System Synthesis (Stockholm, Sweden, September 08 - 10, 2004)

[2] L.R. Dennison, W.J. Dally, D. Xanthopoulos, "*Low-latency plesiochronous data retiming*," arvlsi, p. 304, 16th Conference on Advanced Research in VLSI (ARVLSI'95), 1995

[3] W.K. Stewart, S.A. Ward, "*A Solution to a Special Case of the Synchronization Problem*," IEEE Transactions on Computers, vol. 37, no. 1, pp. 123-125, Jan., 1988

[4] Ajanta Chakraborty, Mark R. Greenstreet, "*Efficient Self-Timed Interfaces for Crossing Clock Domains*," async, p. 78, 9th IEEE International Symposium on Asynchronous Circuits and Systems (ASYNC'03), 2003

[5] Yaron Semiat, Ran Ginosar, "*Timing Measurements of Synchronization Circuits*" async, p. 68, 9th IEEE International Symposium on Asynchronous Circuits and Systems (ASYNC'03), 2003

[6] Ran Ginosar, Rakefet Kol, "*adaptive Synchronization*," iccd, p. 188, IEEE International Conference on Computer Design (ICCD'98), 1998

[7] Joycee Mekie, Supratik Chakraborty, D.K. Sharma, Girish Venkataramani, P. S. Thiagarajan, "*Interface Design for Rationally Clocked GALS Systems*," async, pp. 160-171, 12th IEEE International Symposium on Asynchronous Circuits and Systems (ASYNC'06), 2006

[8] U. Frank, R. Ginosar, "*A Predictive Synchronizer for Periodic Clock Domains*," PATMOS 2004

[9] L.F.G. Sarmenta, G.A. Pratt, S.A. Ward, "*Rational clocking [digital systems design]*," iccd, p. 271, IEEE International Conference on Computer Design (ICCD'95), 1995

[10] T. Chelcea, S. M. Nowick, "*Robust Interfaces for Mixed-Timing Systems*," IEEE Trans. on Very Large Scale Integr. Syst. vol. 12, issue 8, pp. 857-873, Aug, 2004

[11] J. Jex, C. Dike, K. Self, "*Fully asynchronous interface with programmable metastability settling time synchronizer*," US Patent 5 598 113, 1997

[12] Kenneth Y. Yun, Ryan P. Donohue, "*Pausible Clocking: A First Step Toward Heterogeneous Systems*," iccd, p. 118, IEEE International Conference on Computer Design (ICCD'96), 1996

[13] Jens Sparsoe, Steve Furber, "*Principles of Asynchronous Circuit Design – A Systems Perspective*," Kluwer Academic Publishers, 2001

[14] Jakov N. Seizovic., "*Pipeline synchronization*," International Symposium on Advanced Research in Asynchronous Circuits and Systems, pages 87--96, November 1994

[15] Simon Moore, George Taylor, Peter Robinson, Robert Mullins, "*Point to Point GALS Interconnect*," async, p.69, 8th International Symposium on Asynchronous Circuits and Systems (ASYNC'02), 2002

[16] David S. Bormann, Peter Y. K. Cheung, "*Asynchronous Wrapper for Heterogeneous Systems*," iccd, p. 307, IEEE International Conference on Computer Design (ICCD'97), 1997

[17] E. Sjogren, C. J. Myers, "*Interfacing Synchronous and Asynchronous Modules Within a High-Speed Pipeline*," arvlsi, p.47, 17th Conference on Advanced Research in VLSI (ARVLSI '97), 1997

[18] Rostislav (Reuven) Dobkin, Ran Ginosar, Christos P. Sotiriou, "*Data Synchronization Issues in GALS SoCs*," async, pp. 170-180, 10th IEEE International Symposium on Asynchronous Circuits and Systems (ASYNC'04), 2004

[19] S. Ghahremani, "*Metastable Protected Latch*," US Patent 6 072346, 2000

[20] Greiner A., F. Pêcheux, "*ALLIANCE. A Complete Set of CAD Tools for Teaching VLSI Design*," 3rd Eurochip Workshop on VLSI Design Training, pp. 230-37, Grenoble, France, 1992

[21] Ran Ginosar, "*Fourteen Ways to Fool Your Synchronizer*," async, p. 89, 9th IEEE International Symposium on Asynchronous Circuits and Systems (ASYNC'03), 2003

[22] C. Dike, e. Burton, "*Miller and Noise Effects in A synchronizing flip-flop*," IEEE J. Solid-state circuits, 34(6), pp. 849-855, 1999

[23] D.J. Kinniment, A. Bystrov, A.V. Yakovlev, "*Synchronization Circuit Performance*," IEEE Journal of Solid-State Circuits, 37(2), p. 202-209, 2002

[24] http://www-asim.lip6.fr/recherche/alliance/

[25] http://www-asim.lip6.fr/recherche/coriolis/

[26] Tobias Bjerregaard, Jens Sparsø, "*A Router Architecture for Connection-Oriented Service Guarantees in the MANGO Clockless Network-on-Chip*," Proceedings of the Design, Automation and Test in Europe Conference, IEEE, March 2005

[27] Tobias Bjerregaard, Shankar Mahadevan, Rasmus Olsen, Jens Sparsø, "*An OCP Compliant Network Adapter for GALS-based SoC Design Using the MANGO Network-on-Chip*," Proceedings of the International Symposium on System-on-Chip, IEEE, November 2005

[28] I. Miro Panades, A. Greiner, A. Sheibanyrad, "*A Low Cost Network-on-Chip with Guaranteed Service Well Suited to the GALS Approach*", Nano-Net 2006

[29] A. Sheibanyrad, I. Moro Panades, A. Greiner, "Systematic Comparison between the Asynchronous and the Multi-Synchronous Implementations of a Network on Chip Architecture", DATE 2007