Advances in Asynchronous logic: from Principles to GALS & NoC, Recent Industry Applications, and Commercial CAD tools

Alex Yakovlev
Newcastle University
Newcastle, UK
Alex.Yakovlev@newcastle.ac.uk

Pascal Vivet
CEA, LETI, Minatec Campus
Grenoble, France
Pascal.Vivet@cea.fr

Marc Renaudin
TIEMPO
Montbonnot, France
Marc.Renaudin@tiempo-ic.com

Abstract— The growing variability and complexity of advanced CMOS technologies makes the physical design of clocked logic in large Systems-on-Chip more and more challenging. Asynchronous logic has been studied for many years and become an attractive solution for a broad range of applications, from massively parallel multi-media systems to systems with ultra-low power & low-noise constraints, like cryptography, energy autonomous systems, and sensor-network nodes. The objective of this embedded tutorial is to give a comprehensive and recent overview of asynchronous logic. The tutorial will cover the basic principles and advantages of asynchronous logic, some insights on new research challenges, and will present the GALS scheme as an intermediate design style with recent results in asynchronous Network-on-Chip for future Many Core architectures. Regarding industrial acceptance, recent asynchronous logic applications within the microelectronics industry will be presented, with a main focus on the commercial CAD tools available today.

Keywords—component; asynchronous design, handshake circuits, GALS, CAD flow

I. INTRODUCTION

The growing variability and complexity of advanced CMOS technologies makes the physical design of clocked logic in multi-core Systems-on-Chip (SoCs) extremely challenging and costly. The main issues for such systems, which carry on scaling with the Moore's law, are concerned with achieving timing closure in the face of PVT variations, IR drops on power lines, synchronization issues for different clock domains and so on. On the other end of the spectrum of CMOS are many problems in the realm of more-than-Moore, i.e. developments in mixed signal systems such as systems with energy harvesting sources, RFID etc., where power and timing conditions are harsh and require logic to be robust to them. Clearly, IC designers begin to realize that reliance on the use of a single global clock no longer guarantees economic and reliable solutions. The question is therefore arising, whether it is worth to switch to clockless or asynchronous systems, and if so, what types of asynchronous logic design techniques would be most appropriate.

Asynchronous logic is not a new paradigm, it has been studied for many years and become an attractive solution for a broad range of applications, from massively parallel multi-media systems to systems with ultra-low power and low-noise constraints, like cryptography, energy autonomous systems, and sensor-network nodes. While in the past asynchronous logic design methods were developed in relative isolation, they tended to follow fairly purist approaches, such where an entire system would be seen to be built without clocking, using delay-insensitive or speed-independent circuit theory. The reality has proven that those approaches, albeit theoretically elegant, haven't been picked up by industry for various reasons, and most prominently for the lack of connections with real-life industrial design and test practices, starting from the description languages, flexible design tools and ending with the accepted validation procedures. More recent views of the asynchronous design experts are much more in line with real-life, and this is what our position in this tutorial, whose objective is to give a comprehensive and recent overview of asynchronous logic and its modern day status. One of such present day trends for introducing asynchrony into system design is through a more evolutionary approach, called "globally asynchronous, locally synchronous" (GALS) design, which uses asynchronous principles to build interfaces between locally clocked data-processing islands, to combine the advantages of both aspects.

The tutorial is organized as follows. Section II will discuss the main principles and advantages of asynchronous logic. Section III will present the GALS scheme as an intermediate design style. Section IV will follow with recent results in asynchronous Networks-on-Chip for future Many Core architectures. Section V will focus on a modern industrial design flow from Tiempo with its own synthesis tools. Section VI will cover recent applications and CAD perspectives within the microelectronics industry.

978-3-9815370-0-0/DATE13/©2013 EDAA

II. ASYNCHRONOUS DESIGN PRINCIPLES

A. *Design Principles*

This section will outline the key principles lying behind most of the existing asynchronous design methods.

Asynchronous handshaking. When data is passed between logic blocks two aspects of timing are most important. One of them concerns the conditions that determine the moment of time when the sender knows that the receiver has received the previous item, so it can send the next item. The other aspect is for the receiver to know that the data on the information lines is valid as opposed to being in transit (some bits have changed while other bits are still changing their state). In synchronous circuits both aspects are resolved with the help of the clock pulse. In asynchronous circuits, the situation is different. The first aspect is resolved by the use of handshake protocols, where two signals, request and acknowledgement are used to transmit switching events between the sender and receiver, as shown in Fig.1. These req-ack events form communication tokens a useful abstraction for asynchronous system design. Three main types of signaling are often used for handshakes [1]: (i) four-phase, or level-based, or Return-to-Zero (RTZ); (ii) two-phase, or transition-based, or non-Return-to-Zero (NRZ), and (iii) pulse-based. The advantage of the four-phased method is a relative simplicity of the implementation logic, while the two-phase signaling has less communication overhead than the four-phase protocol with more complex logic to implement. The pulse-based approach combines the advantages of the previous two schemes, however at the cost of requiring extra care with the formation of pulses, as some of the transitions on the wires are unacknowledged.

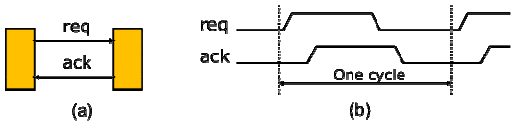

Figure 1. Handshake signaling: req-ack pair (a); four-phase (return to zero, RTZ) protocol (b).

Delay-insensitive encoding. The second aspect of timing is concerned with the validity of data on the information lines. Without having a clock, the validity of the data should be derived from the causal relations between data lines [1]. A simple way, the so called bundled data method, uses the assumption that the data value must transition strictly before the edge on the request signal. In this way, the validity tag is provided by the request signal, which is analogous to the clock. While the bundled data approach often enables reuse of datapaths from synchronous implementations (hence benefit from using results of commercial logic synthesis tools), the timing constraints between data and control may undermine the robustness of circuits, for example under severe PVT variation. Hence the use of delay-insensitive (DI) codes, where the validity of data is directly embedded in its encoding. The simplest form of DI encoding is dual-rail. It relies on the use of two wires for each bit and when combined with a four-phase signaling protocol also requires a NULL state (spacer) when both wires are in the zero state, as shown in Fig 2, where the transmission of a sequence of logical 1 and 0 is shown. Other DI codes can be formed, for example, those based on m-of-n codes (having valid codewords with exactly m bits that are equal to 1 out of the total n wires) [2]. Two examples of DI m-of-n codes are shown that are commonly used in recent NoCs: 1-of-4 (0001=> 00, 0010=>01, 0100=>10, 1000=>11); 2-of-7 (1100000, 1010000, …, 0000011 – in total 21 combinations, which can encode 4 bits of data plus 5 control tokens).

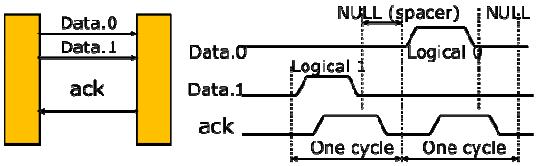

Figure 2. Data encoding in dual rail

Completion detection. In the absence of the clock, the deriving of the validity of data on inputs as well as signaling the completion of transients inside logic blocks can be done by using special pieces of hardware dedicated solely to completion detection. For bundled data representation the completion is indicated by a special matched delay, whose value must be greater than the worst-case delay in the single-rail logic block. For DI signaling, such as dual-rail, special circuitry is added to propagate information about the transient completion, as illustrated in the implementation of a completion detection tree shown in Fig. 3. This tree consists of a row of OR gates followed by a multi-input C-element, which can be constructed either as a tree of 2-input C-elements or using trees of simple gates for multi-input AND and OR and a 2-input C-element at the end. The need for completion logic at the output of the dual-rail logic, as opposed to using 'shortcuts' via matched delays, effectively supports the notion of causal acknowledgement, described in the following paragraph.

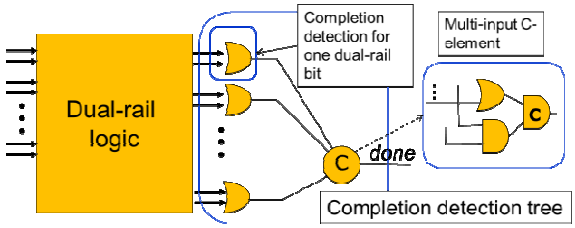

Figure 3. Completion detection for dual-rail logic

Causal acknowledgement. In synchronous circuits or when using the bundled data method the correctness of timing conditions is determined by assumptions about relative delays (in case of clock, between the worst case paths and clock period). For asynchronous operation that relies on the explicit indication of the transients between the circuit gates, that is called causal acknowledgement. According to this principle, every transition on inputs or the output of each gate is acknowledged, or indicated, by some other signal transition. To illustrate this effect, consider the circuit shown in Fig. 4(a). The circuit diagram shows a speed-independent or quasi-delay-insensitive (QDI) implementation of the behavior of a two-way C-element, whose specification is given in the signal transition graph shown on the right. The refined behavior of this circuit, at the level of all gates, is shown in Fig. 4(b), which clearly

shows that the cause-effect relation is guaranteed for all transitions regardless of the delays of all gates. Signal transition graphs (STGs) are generally used to specify asynchronous control logic [3]. There are tools to perform synthesis of control logic from STGs, the most widely used such tool is Petrify [3].

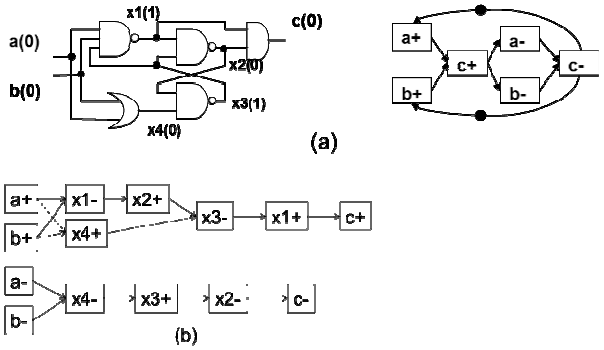

Figure 4. C-element implementation in simple gates with full indication of signal transitions and its signal tranistion graph (a); its refined signal-transition graph (b)

Full indication and early evaluation. Related to the notion of causal acknowledgement and indicatability is the property of causality between signals inside the circuit, which has its implication on the implementation of logic operators. For example, when using dual-rail encoding one can think of a strongly-indicating QDI implementation of a two input logical AND, shown on the left of Fig. 5 (this implementation is also known as DIMS [2], for "directly indicating min-term synthesis") and a weakly-indicating implementation, sometimes also known as an implementation "with early propagation" (shown on the right). The tradeoffs between robustness and efficiency in these two options for two-input AND are quite obvious.

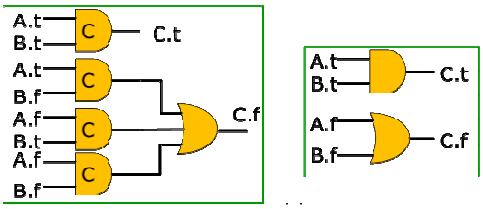

Figure 5. Strongly indicating and weakly indicating ("with early propagation") implementationss of logical AND.

Time comparison. The last important principle is about the need for explicit logic to perform the so-called "time comparison", which exhibits itself either in the form of synchronization or arbitration. A synchronizer is a device which interfaces a circuit with its own clock to an input whose signal transitions are produced outside the clock domain, and thus being asynchronous to the circuit. Using a simple flip-flop as a synchronizer is prone to failures because when its data input (used as an asynchronous request or other validity tag) and clock input change close to each other, the set-up condition for the flip-flop is violated and its output may enter a metastable state, between 0 and 1, and stay there longer than the clock period. This may cause a malfunctioning in the circuit where the metstable state may be interpreted as 0 in one place and 1 in another. More about metastability, synchronization and the design of synchronizers can be found in [4].

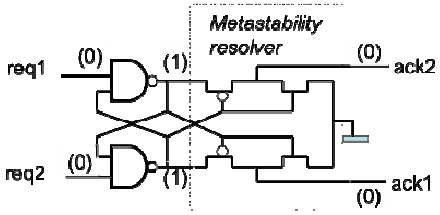

Figure 6. Two-way arbiter (mutex) with metastability resolver

Another form of time comparison is for asynchronous arbitration [4], which is for example required for resolving various forms of conflict in systems, including arbitration for common resources between mutually asynchronous clients. A two-way arbiter (also called mutex) is illustrated in Fig. 6. It can be used to determine the order of arrival between two mutually concurrent requests req1 and req2. The circuit issues only one grant ack1 or ack2 at a time. Since the circuit uses an SR latch the digital part of the mutex may enter a metastable state. However, the analogue part, called metastability resolver, prevents its propagation to the ack outputs. Only when the latch leaves the metastable state one of the outputs ack1 or ack2 goes to 1.Two way mutexes are used in building GALS interfaces and complex arbiters in NoC routers.

B. Main advantages and drawbacks

Researchers have always been excited by asynchronous design and motivated by their ability to work on average not worst case delays; lower power consumption (automatic fine-grain "clock" gating; automatic instantaneous stand-by at arbitrary granularity in time and function; distributed localized control; more architectural options/freedom; more freedom to scale the supply voltage); modularity; lower EMI and smoother Idd (the local "clocks" tend to tick at random points in time); low sensitivity to PVT variations (due to inherent indicatability); secure chips (white noise current spectrum).

So why hasn't asynchronous logic been adopted in the past? Some of the reasons are due to overheads (area, speed, power) in control and handshaking, dual-rail and completion detection costs. However, these are less of an issue now due to the counterarguments in favour. The most prominent now are actually the reasons associated with the design and test issues. Namely: the variety of styles and variants to go and one can easily get confused which is better; lack of practical CAD tools as the tools tend to be quite specific to particular design styles and design niches; complexity of timing and performance models; difficulty with sign-off (for particular frequency requirements). Last but not least, the hardness to test asynchronous circuits using conventional testing methods and equipment is another obstacle.

Despite these drawbacks, the situation is gradually changing and as the following sections demonstrate, the design methods and tools gradually mature and industry is certainly much less averse to this discipline because the pluses are gradually outweighing minuses.

III. GLOBALLY ASYNCHRONOUS LOCALLY SYNCHRONOUS

A. *GALS Design Main Principles*

The Globally Asynchronous Locally Synchronous (GALS) scheme has been introduced as early as 1984, by Daniel Chapiro. A GALS system (Figure 7.) consists of a number of complex digital blocks operating synchronously. Those blocks are usually developed using standard synchronous CAD tools and design flow. However, the operation of the blocks is not mutually synchronized, hence the name “locally synchronous”. These locally synchronous blocks communicate with each other asynchronously: at top level, the system is asynchronous.

In the literature, the “GALS” term stands for any design style between fully synchronous design using unrelated clock frequencies – the GALS term is often used by the industry today to implement multi-synchronous large SoCs – to various forms of asynchronous communication schemes between the synchronous islands. The communication topologies can be anything from point-to-point communications to structured interconnects, like bus hierarchy, rings, or Network-on-Chip. For asynchronous style, a common approach is to add a so called "asynchronous wrapper" that provides an interface from the synchronous to the asynchronous environment to every locally synchronous block.

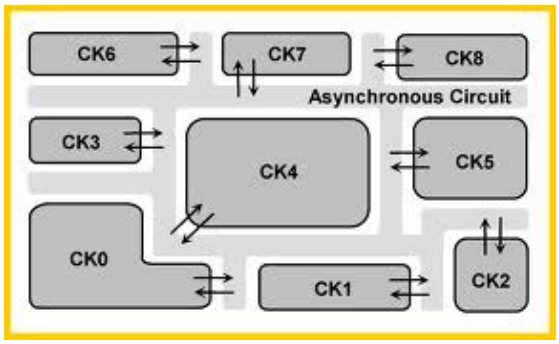

Figure 7. GALS architecture template

The main advantage of GALS is to provide modular and scalable architecture. GALS is a way to structure system level communication, such as those in Network-on-Chips. Regarding physical design and timing optimization, global clocking is not feasible anymore due to long wires and high frequencies. GALS is a way to provide structured pipelined interconnect: top level long wire delays are handled through asynchronous signalling and pipelines, with high level protocol semantics. As a result, local synchronous islands are smaller IPs, thus allowing smaller and independent clock trees, with smaller clock skews. GALS is also a natural enabler for Low Power. By fully decoupling communication from computation, power management techniques such as Dynamic Voltage and Frequency Scaling, DVFS) can be applied to GALS where synchronous IPs are independent voltage and frequency islands. Lastly, GALS design brings also Low Noise. By decoupling the frequency domains, the GALS architecture will generate smaller noise with a set of smaller multi-harmonics instead of a single large-noise harmonic. This can be beneficial for crypto applications for instance.

Nevertheless, all these often advocated advantages of a GALS template must no hide various issues and drawbacks. When designing a GALS system, the main issue is the design of reliable GALS interfaces in order to handle the so-called metastability problem, which may occur between the synchronous and asynchronous logic domains [5]. The GALS design style can be classified into two main forms: (i) a full clock cycle is required to wait for metastability to resolve; such a technique is adopted in multi-synchronous GALS design by using a two flip-flop synchronizer, likewise using mixed synchronization FIFOs ; ii) a single flip-flop but with a delayed clock; this is the initial unsynchronous machine from Chapiro which has evolved in the pausible or stretchable clock concepts [8]. In both cases, the GALS interfacing logic must provide high performance with high throughput and low latency. The two following sections present the design principles of these two design styles.

B. *FIFO based synchronization*

For SoC level synchronization, a two flip-flop (2FF) synchronizer is simple to design, but a back pressure protocol is required to handle sporadic and bursty traffic with associated full/empty states. Then, a simple 2FF synchronizer would present a large performance overhead with 4 clock cycles throughput (at least) for any token round trip; which is unacceptable cost at system level. The classical solution is to use a synchronization FIFO to hide synchronization and allow parallel read and write.

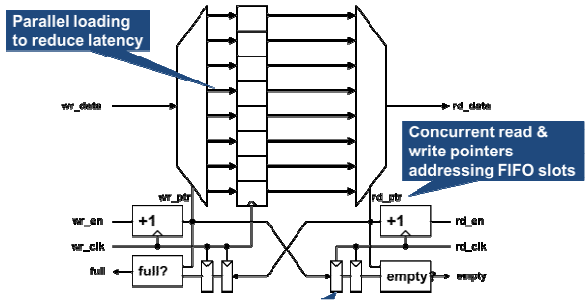

Figure 8. Dual Clock FIFO principles

As presented in Figure 8. , a dual clock FIFO allows concurrent read and write using independent read and write clock domains, and synchronization of read/write FIFO counters with respective opposite clocks to compute the empty/full FIFO status. Since the FIFO pointers cross timing domains, it is required to use an adequate encoding to ensure proper synchronization and detection of FIFO empty/full states. For this, Gray encoding is classically used in multi-synchronous FIFOs. With a Hamming distance of one, the Gray-encoded FIFO counters only change one digit by read or write iteration, thus providing correct multi-bit synchronization. Gray encoding leads to complex logic, since counter increment need translation from/to binary, and is mostly limited to 2^N FIFO sizes, with an N-bit Gray counter.

For mixed-mode asynchronous/synchronous FIFOs to be used in a GALS scheme, similar FIFO strategy applies but one FIFO side is transformed or wrapped to behave with an asynchronous protocol; and different pointer encoding can be used to replace costly Gray codes. Various FIFO encoding and

architecture have been studied in the recent years. Most of these FIFOs use token based encoding instead of Gray code. Chelcea [6] has proposed mix-mode synchronization FIFOs, using a 1-hot token-based encoding implemented using precharge logic, providing efficient but - partly - full custom design. Sheibanyrad [11] has proposed a bubble encoding for FIFO design targetting a GALS NoC architecture. More recently, Thonnart [7] has proposed to use Johnson encoding, providing also a Hamming distance of 1 as for Gray code, a less dense code but not restricted to 2^N values, allowing to have efficient small FIFOs (depth 5 or 6).

C. *Pausible Clock Design*

Another solution for cross-timing domain resynchronization is to use the so-called pausible clocking. The basic idea is to stretch the clock cycle when a transfer occurs between synchronous and asynchronous domains until the transfer is complete, to avoid any metastable state and allow safe data latching. As presented in Figure 9. , the locally synchronous block is clocked by a local clock generator and communicates with external world using asynchronous handshake channels (implemented usually by the bundle-data 4-phase protocol). The IP is wrapped by specific port controllers which request the clock generator to suspend the next clock edge until incoming/outgoing transfer is completed, to allow safe data transfer.

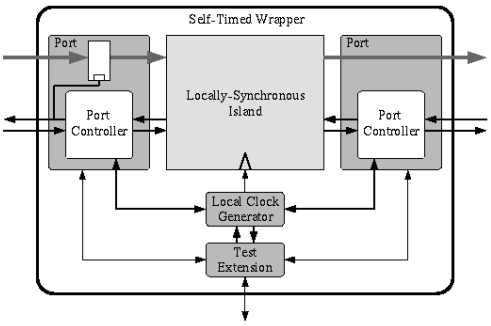

Figure 9. IP Self Timed Wrapper [9]

Various pausible clock schemes have been proposed in the literature [8]. A comprehensive GALS design flow based on pausible clocking has been developed by ETH Lab with various port controllers (poll type, demand type), various interconnection topologies, an associated test architecture, and a complete design flow [9]. Nevertheless, such an approach suffers from various design issues. The port controllers are specified as asynchronous finite state machines (AFSM), using a burst-mode description. The port controllers are implemented using an AFSM synthesis tool, but care must be taken to avoid any logic remapping during logic synthesis and place & route, to guarantee hazard-free asynchronous logic. A more tricky issue concerns clock tree insertion time. In preliminary works, the clock tree insertion time of the synchronous IP was limited to match a delay race between clock pause path and data latching in the port controller being clocked by the generated clock. That was leading to small clock tree insertion time (half the clock period), thus limiting the size of the synchronous IP. This clock tree limitation has been overcome in recent work, but with more tricky design, and still providing a low data transfer throughput (less than 0.5 word per clock cycle) [10]. Pausible clocking exhibits also other system level limitations: (i) the Local Clock Generator design is a low-cost feature to offer Dynamic Frequency Scaling (DFS), but since it is based on a programmable delay line that can be implemented in full std-cell, or could be more sophisticated, it is usually still inaccurate to guarantee a target frequency; (ii) lastly, by pausing the clock edge for each data transfer, the transfer throughput is limited, thus reducing the system performance by not ensuring a stable and regular clock generation.

D. *Recent GALS designs and Conclusion*

The most accomplished GALS circuit design techniques using Pausible Clock, with associated circuit results, have been developed by ETH and IHP labs. The ETH lab have developed various circuits [9], targeting a crypto-application with the AES algorithm, and also studied various topologies (ring, bus, crossbar). Thanks to clock tree partitioning, compared to pure synchronous design, the GALS version presents lower EMI, which was an advantage for the crypto-application, while presenting similar performances (speed, area, power).

More recently, the IHP lab have developed [10] a complete design methodology, and a baseband OFDM TX circuit, with 6 GALS IP blocks and 16 GALS links, fabricated and measured in a 40nm CMOS technology. A fair comparison with a synchronous version of the same OFDM BB TX design exhibits a 5% area gain and 6% power consumption reduction for the GALS version running at 160MHz. These gains are obtained thanks to GALS partitioning, smaller clock tree, and gain in clock tree power. Nevertheless, the proposed design techniques still require lots of expertise for too minor gains.

	Pausable Clocking	**FIFO-based**
Area overhead	Low	Medium – High
Latency	Low	High
Throughput	Lowered wrt. Clock Pause rate	High
Power Consumption	Low	High
Additional Cells	Mutex, Delayline, Muller-C	Empty/Full flag
Advantages	No Metastability	Simple Solution, Throughput
Disadvantages	Local Clock generators, Throughput	Area Overhead, Latency

Figure 10. GALS design technique comparison

Finally, when comparing the two main GALS design techniques (Figure 10.), pausible clocking is more adapted for low power, low performance application, and requires more design expertise, while FIFO based synchronization will target high performance application with guaranteed throughput, but presenting higher latency and area costs. In both cases, in order to hide the inherent design complexity regarding timing analysis and associated verification, these GALS interface blocks (pausible clock generator & port controller, or re-synchronization FIFO) can be designed as hard macros that can be easily reused at system level. In conclusion, the main benefits of GALS will not be only obtained from circuit design techniques but most of all at system level thanks to modular and scalable GALS design, allowing local DVFS for further power consumption system level optimization [14].

IV. ASYNCHRONOUS NETWORK-ON-CHIP

A. *Main principles*

Networks-on-Chip (NoC) have been introduced as an alternative to more traditional bus-based architectures, to give a modular and scalable communication architecture based on packet switching. With a clear separation between computation and communication, the NoC architecture perfectly fits the GALS paradigm (Figure 11.), where NoC units are implemented as synchronous units, with independent clock domains, while the NoC infrastructure can be implemented in fully asynchronous logic.

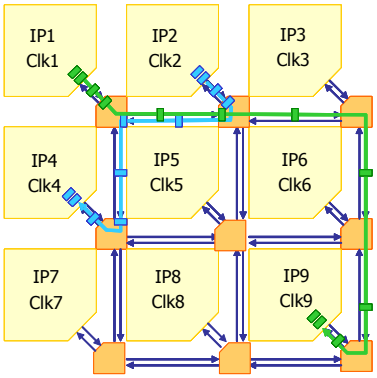

Figure 11. GALS Network-on-Chip template

Regarding clocking strategy, the NoC may be implemented using synchronous logic, but due to timing constraints for long wire communication, single-clock synchronous design is not feasible (clock skew on large topologies, margins, etc.), a certain level of de-synchronization is often used. This can be done using standard multi-synchronous design, source-synchronous design, or with meso-chronous clocking. Such synchronous NoC implementations are feasible, but present large latency overhead, due to synchronization cost at each router hop, even when using optimized mesochronous design [11], when compared to a fully asynchronous NoC.

A more elegant solution consists in implementing NoC routers and links using asynchronous logic and handshake channels. Many asynchronous NoC have been proposed in the literature, using various asynchronous design techniques:

- Bundle-data asynchronous logic, like in MANGO, QNOC, which use standard 4-phase protocols, or more recently [12] using a two-phase Mousetrap protocol, to increase link throughput ;
- Quasi-Delay-Insensitive (QDI) asynchronous logic, like HERMES-A using dual rail, ANOC using 1-of-4 DI code for lower power, or CHAIN using more complex 2-of-7 DI code for code density and additional power reduction, to provide robust NoC routers and links insensitive to timing variations ;
- Mixed design (internal router using bundle data logic while long NoC links are implemented using QDI logic), like in ASPIN [11], to benefit of both advantages : smaller logic in the routers, robustness on the long NoC links, but requiring additional conversion logic.

B. *NoC Building Block Design*

An asynchronous NoC is composed of asynchronous routers, aimed to route and arbitrate packets in the topology, and NoC GALS interfaces that are responsible for bridging the timing domains between the asynchronous router and the synchronous units. These NoC GALS interfaces can be either implemented using a FIFO-based synchronizer [7] or using a pausible clock GALS interface [14]. Due to performance constraints, FIFO-based GALS interfaces will be preferred for asynchronous NOC, as explained in section III.

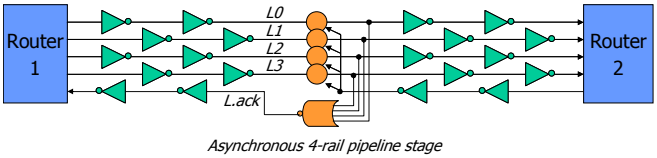

Figure 12. Asynchronous NoC link pipeline stage

Lastly, the NoC is composed of asynchronous long wire NoC links that can be implemented using various encodings: bundle-data, which requires timing margin constraints to be met on long wires, or using QDI asynchronous logic for design robustness and easier physical design. In order to increase NoC link performances, it is often required to pipeline the long link wires. Compared to synchronous retiming in pipelines, pipeline retiming can also be done using asynchronous logic, yet more easily and efficiently. A long NoC link shown in Figure 12. is pipelined; it uses a 4-phase 1-of-4 QDI encoding, by adding a half-buffer pipeline stage (C-elements and a NOR gate). As a result, the wire length is divided by two per stage, the throughput is multiplied by 2, with *a cycle time being 4 times this new wire length due to 4 phase protocol*, while the forward latency is preserved: the inverters being replaced by C-element. This can be done as often as needed according to the NoC link length and properly optimized using place & route tools [16]. Compared to synchronous retiming, asynchronous pipeline does not add extra clock cycles along the NoC links.

C. *Opportunities for Power Reduction Design Techniques*

In asynchronous Network-on-Chip, as for GALS, the main system level advantages will come from additional power reduction design techniques. In the NoC routers, a possibility is to benefit from the robustness and locality of asynchronous logic to automatically detect NoC traffic activity and use this information to power down and save leakage when NoC routers are Idle [14].

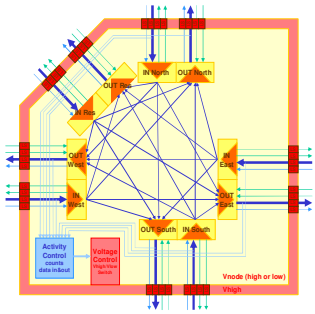

Figure 13. NoC router with Activity Detection

Activity detection on the incoming and outgoing NoC links is performed using channel monitors. The channel monitor drives the voltage regulator, which is composed of a simple power switch, offering a normal mode and a low power mode (Figure 13.). This is a fully autonomous and robust detection mechanism, without additional software control and with minimal latency cost (2 ns wake-up). In Idle mode, the leakage reduction is a factor of 4, while the router is still functional providing a low power mode (1/4 the throughput, 1/2 the energy). Such a scheme may be implemented using synchronous logic in a NOC router, but would not be as efficient, due to extra cost for clocking and wake-up control.

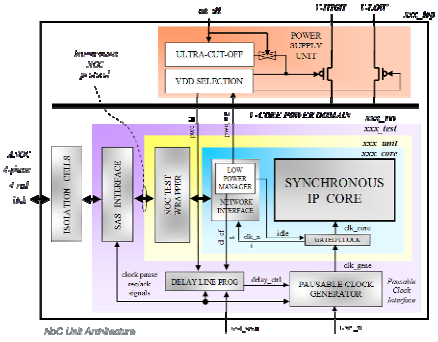

Figure 14. GALS NoC unit template for DVFS [14]

For NoC units, as discussed in section III for any generic GALS scheme, the main idea is to implement independent frequency and power domain synchronous islands, to achieve Dynamic Voltage and Frequency Scaling (DVFS). As presented in Figure 14. , a given synchronous IP core of the NoC architecture can be wrapped in a template architecture, with its Network-Interface for handling NoC traffic, a Local Power Manager to control, by software, the various Power Modes, a Test wrapper for testability, a GALS interface for interfacing synchronous and asynchronous domains, a Local Clock Generator for DFS, and finally a Power supply units providing voltage selection and leakage control for DVS. The GALS NoC template allows decoupling of the various architecture services, such as NoC communication, power management, clock generation, re-synchronization stage, all locally at IP level. In many-core architectures, such a DVFS template can then be exploited for power optimization according to application constraints [14][15].

D. Asynchronous NoC results

A synchronous vs. asynchronous comparison of the same NoC in a 65nm technology is presented in Figure 15. [13]. ANOC achieves low power (* *measured on a real application*), for half the latency, for the same throughput, but with some area cost due to QDI logic. A recent Design Flow based on standard-cells and using Place & Route tools can achieve more aggressive performances for QDI asynchronous logic [16].

	A-NOC	S-NOC
Area	0.17 mm2	0.094 mm2
Frequency	550 MFlit/s	480 MHz
Latency	2.3 ns	4.2 ns
Average Power*	11.2 mW	82 mW

Figure 15. Asynchronous vs. Synchronous NoC router comparison

V. INTRODUCTION TO TIEMPO ASYNCHRONOUS CIRCUIT SYNTHESIS AND DESIGN FLOW

The Tiempo asynchronous circuit design flow is based on a unique and specific synthesis tool called ACC (Asynchronous Circuit Compiler). ACC performs fully-automated synthesis of asynchronous circuits starting from a standard hardware description language, SystemVerilog. Indeed, TLM-like descriptions in SystemVerilog are transformed into gate-level netlists in Verilog. ACC maps the circuits on standard cell libraries augmented with asynchronous cells.

In terms of flow (Figure 16.), ACC is the only different tool, and it is made compliant and interoperable to standard design flows based on industry-standard tools. SystemVerilog models can be simulated with any HDL simulators, enabling the verification of mixed asynchronous/synchronous designs. ACC provides a set of sync-async and async-sync interfaces to ease mixing synchronous and asynchronous designs into a single SoC. The implementation of the asynchronous netlists is supported by industry-standard P&R, STA and LVS/DRC checker tools.

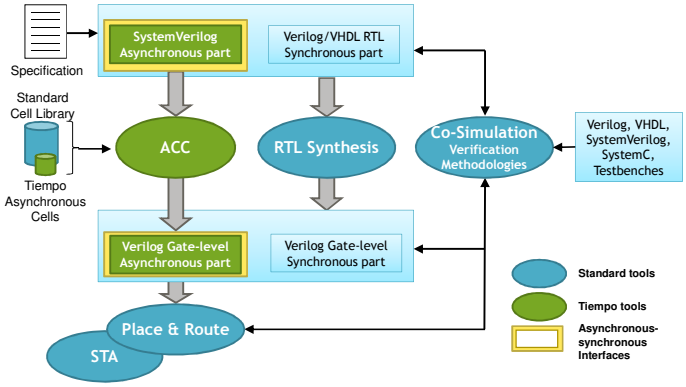

Figure 16. Tiempo asynchronous Circuits Design Flow.

A. Tiempo Design Flow Description

1) Modeling

Asynchronous circuit models are written in un-timed Transaction Level Modeling (TLM), using the standard IEEE-1800 SystemVerilog language. Using this format provides a seamless integration of Tiempo clockless technology into verification platforms like Synopsys VCS™, Mentor Graphics Questa™, Cadence NCsim™ or others, and allows the designer to access any required debug tools to find out cause of possible design mistakes.

Channels represent the basic medium for communication between asynchronous design entities and processes. A channel allows point-to-point communication between two processes, where each communication through a channel involves a token exchange between the two processes, via a “handshake”. Handshaking protocols can be of two kinds: push or pull [1]. Each mode defines which one of the two processes involved in the token exchange is the initiator of the handshake: the one writing data to the channel or the one reading them. Channels are modeled as SystemVerilog interfaces. Tiempo developed SystemVerilog definition files and macros to predefine an interface (i.e., channel type) for each of the SystemVerilog data

types (bit, byte, integer, etc…) as well as for user-defined types (e.g. typedef or enumerated type). Channel communications are modeled as read and write operations using methods automatically created with each channel type. Those operations can be blocking or non-blocking.

Tiempo defined the language to provide the designers with the key tools necessary to build efficient asynchronous circuits for their applications. Indeed, the language is such that it enables the designers to adopt the right architecture styles for their applications, including pipelined, data-flow, parallel, sequential, etc… Hence, design objects such as modules or components are available to model concurrency and hierarchy (SystemVerilog Modules), asynchronous communicating processes are available to model procedural and concurrent behaviors (SystemVerilog Processes), and communicating channels are available to make modules and processes communicate and synchronize between each other (SystemVerilog Interfaces).

Further information with regards to Tiempo SystemVerilog coding style can be found in [22], in which a simple FSM-ALU structure and its testbench are detailed.

2) Synthesis

The ACC synthesis tool is seamlessly integrated into standard design flows. Indeed, ACC interoperability with commercial CAD tools is made effective using the standard file formats most of the tools are using to exchange information between each other (Figure 17.). In addition, ACC provides a standard TCL interface using industry-standard names for the different commands.

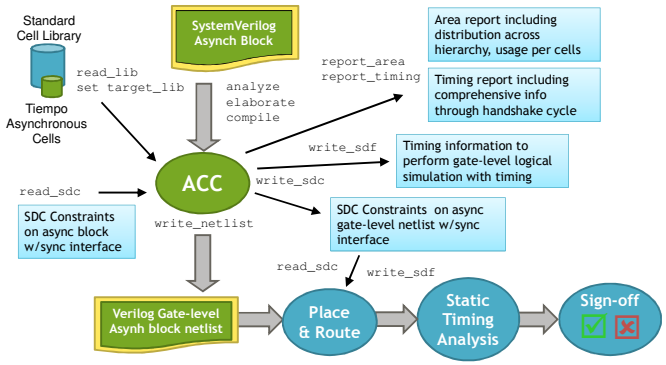

Figure 17. ACC interaction with the design flow tools.

The input to the tool is as follows. The HDL description is in SystemVerilog. The targeted technology is specified using the standard library format (.lib file). A set of standard cells commonly used for synchronous circuits is provided to the tool as well as a set of asynchronous cells, necessary for the synthesis of efficient delay insensitive circuits. Finally, design constraints are specified using the standard design constraint format (.sdc file). In addition to the standard sdc language, some commands have been added in order to accurately and efficiently constrain asynchronous circuits. As an example, it is very common in an asynchronous design to have many parts running at very different speeds. The new commands provide a means to control the constraints applied to different modules and/or handshaking communications.

At the output, ACC generates the circuit netlist in Verilog format (.v file). ACC also generates timing information using the standard delay format (.sdf file). Last, but not least, the tool generates an sdc file, compiling the input design constraints and the constraints generated by the synthesis process.

3) Place and route

Standard place and route tools can be used to implement the netlist generated by ACC. Since there is no clock at all there is no need to perform a clock tree synthesis and the associated timing closure. However, timing driven place and route has to be done in order to respect a given performance target. This step is made possible using the "sdc" constraint file generated by ACC. Timing constraints are expressed as a set set_max_delay commands annotating the handshaking cycles of the design [23]. This enables the P&R tool to accurately optimize the asynchronous logic paths involved in the handshaking communications, taking into account the local timing characteristics of the asynchronous netlist. This step is for asynchronous circuits what optimizing the critical paths between flip-flops is for a synchronous circuit. The only exception here is that there are no functional issues since the asynchronous design is delay insensitive, whereas in a synchronous circuit the timing constraints must be fulfilled, not only to respect the timing performance target, but also to guarantee the functionally correct behavior.

4) Verification

For functional verification, transaction viewers of standard simulators can be used to give a high level view of the communication involved in the asynchronous design, through those channels [22]. The transaction view of the system is particularly convenient as it hides the low-level implementation of those channels, and gathers only the necessary information in a convenient format. Using this higher level view, one can represent the channel state (whether active or inactive), the start and end time of each channel operation, and finally the relevant attributes characterizing it, such as the occurrence of the operation or the token value at a given time. As an example, illustrated in Figure 18. , Mentor Graphics Questa™ transactions support the simulation and debugging of an asynchronous design by detailing the sequences of token exchanges between the different system modules in the design, whether they are in the control-path or data-path, providing the necessary fields for proper data monitoring and verification, and giving a clear picture of the capacity and utilization of the different channels in the system.

For timing, dynamic verification is performed at the gate level running a standard simulator using the Verilog netlist along with the delay information provided by the "sdf" file. Such a logical simulation with timing enables an accurate analysis of the timing behavior of asynchronous circuits and their dependency with respect to data. Static timing verification is done running standard sign-off tools like Synopsys PrimeTime™, using the Verilog netlist, before or after placement and routing, the "sdc" constraint file, and the back-annotation "sdf" or "spef" files.

Physical verifications after place and route are performed using standard Extraction, LVS and DRC tools.

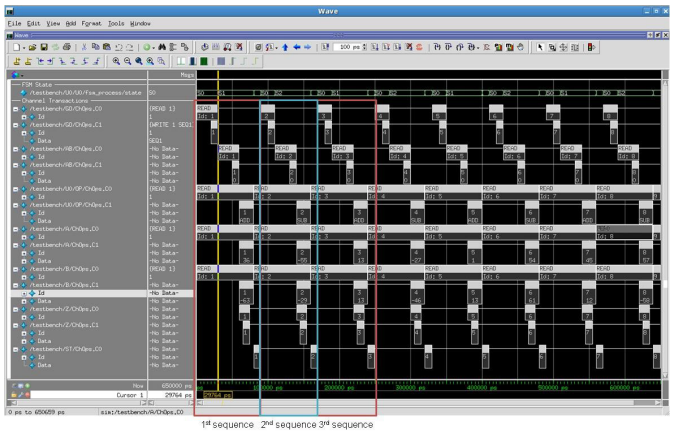

Figure 18. Typical view of tokens flowing in a viewer [22].

Regarding formal verification, there is no tool yet enabling checking the equivalence between the SystemVerilog model and the Verilog netlist. Today, extensive dynamic simulations are performed, using a unique test-bench and checking for code and gate coverage, in order to show the functional correctness.

Layout equivalency checking (LEC) is possible using standard tools, to verify netlists before and after place and route.

B. Design Examples

Tiempo's asynchronous delay insensitive technology is successfully applied to designing integrated circuits implementing secured transactions such as smartcard for banking or ticketing, ePassport, or standalone circuit for DRM or NFC secure elements. Tiempo products deliver a new security paradigm against hardware attacks as well as exceptional performance in ultra-low and/or variable power environments (secured contactless transactions). Hence, the TESIC platform developed by Tiempo is a fully delay insensitive integrated circuit including a microcontroller, three crypto-processor cores, and several dedicated blocks (Figure 19.). An FPGA emulation of TESIC is also available for software prototyping, thanks to the ACC tool, which not only targets ASIC but also FPGA technologies.

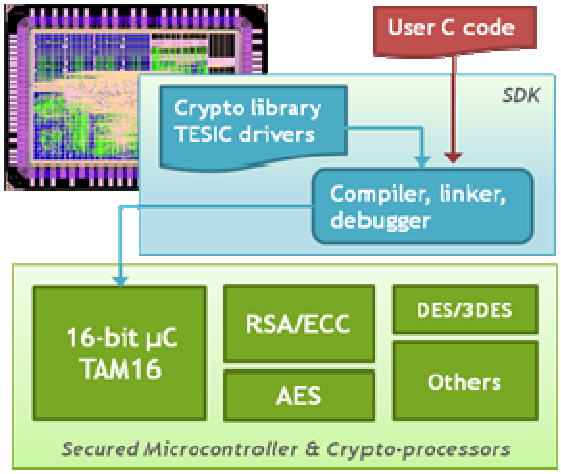

Figure 19. TESIC platform for secured transactions.

Finally, the most recent major application of Tiempo design technology and flow is the design of variability-tolerant circuits on advanced processes (32 nm to 14 nm). The circuits generated by ACC have the fundamental property of being delay insensitive. This feature makes them robust with respect to any effect impacting the timing (such as process, voltage or temperature variation), and easy to design since process/ library timing information can be inaccurate or even absent (no timing closure design phase needed). Moreover, it enables them to run at maximum speed under a wide range of operating conditions, hence accurately monitoring the switching speed of devices and gate networks. All these key and unprecedented features have been proven by the design, fabrication and characterization on a 32 nm process of Tiempo unique monitoring chip (Fig. 20) that allows faster advanced process performance characterization [24].

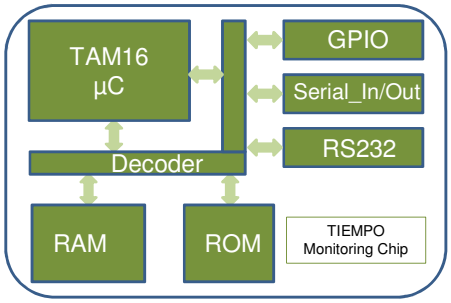

Figure 20. MTAM16, Tiempo monitoring chip for advanced processes.

Tiempo is the first company to commercialize a complete flow enabling the design of delay insensitive asynchronous circuits. Its unique synthesis tool called ACC, extensively uses standard languages and formats in order to make it compliant and interoperable with existing commercial CAD tools. It is successfully applied to the design of products in the domains of secured integrated circuits and variability-tolerant circuits fabricated using advanced processes.

VI. Recent Industry Examples and CAD Perspectives

A. Asynchronous Design in the Industry

Fulcrum, now part of INTEL, is designing Gigabit Ethernet Crossbars. Fulcrum is targeting high speed design, based on QDI precharge logic, the so-called PCHB template [17]. This aggressive asynchronous design style achieves high performance by using deep logic pipelining, specific precharge cells, and full custom layout. An automated asynchronous flow has been developed [18]. For more relaxed part of the design, synchronous design is used, thus using a GALS scheme.

Achronix, is designing fast FPGAs. As any FPGA, the architecture template is based on interconnected LUTs, but the FPGA is implemented using high speed asynchronous QDI precharge logic. As any FPGA, it can map synchronous RTL, using ad-hoc FPGA mapping tools. As for Fulcrum, the main asynchronous logic advantage is high performance and ease of full custom design thanks to QDI logic robustness.

Handshake Solutions, (*initially Philips technology is now back in NXP*), developed a complete flow, with the Haste language, associated synthesis tool, and test methodology, targeting bundle-data handshake circuits. The flow has been extensively used for various markets: pagers, automotive, smartcard, e-

passport, game consoles, etc. The main advantage of asynchronous logic is regarding low power and low noise.

Octasic, is designing DSP for Audio and Multimedia applications. The DSP architecture is based on parallel synchronous sub-units, working in an interlocked manner, where synchronization is implemented using asynchronous bundle-data logic. The main advantage of asynchronous logic is regarding low power and modular and scalable architecture.

Tiempo, has a complete design flow presented above to design variability-tolerant QDI asynchronous circuits. Variability may come from the energy source as it is the case for contactless secured platforms for banking, ticketing, or passport applications. In this field QDI circuits' properties to counter hardware attacks are also exploited. Variability may also come from the fabrication process, especially advanced processes, where Tiempo circuits show extremely high robustness with respect to process, temperature and voltage variations, and run at maximum speed, thus enabling process performance monitoring.

To sum up, several companies have adopted asynchronous design methodology (some disappeared: *Theseus, Silistix, Elastix*). Companies never claim asynchronous logic as an objective, only a way to achieve differentiation: QDI logic for high performance, low power under variability constraints, especially process and voltage (e.g. subthreshold designs), and bundle data for low-power/low-noise.

B. CAD tool status and perspectives

One of the main limitations of asynchronous design concerned the lack of asynchronous CAD tools. In previous years, many CAD tools have been developed, such as Petrify [3], Minimalist [1] or others, but these tools only address low complexity asynchronous controllers, limited to tens or hundreds of gates. In order to address system level design, a language-based approach must be adopted for asynchronous design, to achieve productivity similar to RTL abstraction for synchronous design. Various languages for asynchronous logic have been proposed, which fit the asynchronous handshake channel semantics, like CSP/CHP, Tangram/Haste, Balsa [1]. Associated asynchronous design styles and corresponding synthesis tools have been developed. Nevertheless, these languages are too specific, and their adoption by industry is difficult. The ***Tiempo*** design flow, as presented section V, being based on a standard *System Verilog* language extended with an asynchronous channel *System Verilog* library is a good trade-off, as input to a dedicated asynchronous synthesis flow. A similar approach with Verilog was developed in [18].

Regarding cell libraries, asynchronous design requires specific cells, such as C-element, arbitration cells (Section II), or more specific cells (like pre-charge logic cells). C-element can be designed from a basic foundry library. But for efficient design, it is mandatory to develop a specific cell library (about 50 cells usually), that will then be supported by other tools (Timing Analysis, Place & Route) [16] [18]. Better support of these specific cells by foundry libraries and by standard tools would be beneficial to wider usage of asynchronous logic.

For performance analysis, even if current timing analysis tools can be used at a cell or pipeline level, performance analysis and optimization of asynchronous logic is still an advanced research topic, lacking industrial CAD tools [19][20].

VII. CONCLUSION AND PERSPECTIVES

Asynchronous logic design has a long history of research innovation. Current developments in nanoscale technologies start to open doors and give way for asynchronous logic into industrial design practice, where several companies have already demonstrated its advantages in real products and services. This tutorial shows that such routes are likely to lead further via GALS and NoC architectural paradigms. Other paradigms will certainly be emerging in the near future with the rise of interest in energy-harvesting electronics, wireless sensor networks and mixed-criticality systems. Due to the lack of space, we could not address all of them here. To facilitate the progress in design more research efforts are required in developing CAD tools. Tiempo and its unique QDI-based synthesis flow, closely linked with the accepted languages and tools, is a strong motivating factor.

REFERENCES

[1] J. Sparsø and S.B. Furber, editors. "Principles of Asynchronous Circuit Design", Kluwer Academic Publishers, 2001.

[2] T. Verhoeff. "Delay-Insensitive Codes - an Overview". Distributed Computing, 3(1):1-8, 1988.

[3] J. Cortadella, M. Kishinevsky, A. Kondratyev, L. Lavagno, and A. Yakovlev. Logic Synthesis of Asynchronous Controllers and Interfaces. Springer-Verlag, 2002.

[4] D.J. Kinniment, "Synchronization and Arbitration in Digital Systems", Wiley and Sons, 2007

[5] R. Ginosar, "Fourteen Ways to Fool Your Synchronizer", Proceedings of ASYNC'2003, pp. 89-96, May 2003.

[6] T. Chelcea and S.M. Nowick, ''Robust Interfaces for Mixed-Timing Systems,'' IEEE Trans. VLSI Systems, vol. 12, no. 8, Aug. 2004, pp. 857-873.

[7] Y. Thonnart, E. Beigné, P. Vivet, "Design and Implementation of a GALS Adapter for ANoC based Architectures", Proceedings ASYNC'2009, pp. 13-22.

[8] R. Mullins, S. Moore, "Demystifying Data-Driven and Pausible Clocking Schemes", in Proc. of ASYNC'07, pp 175-184, March 2007.

[9] F. Gürkaynak, S. Oetiker, H. Kaeslin, N. Felber, W. Fichtner, "GALS at ETH Zurich: Sucess or Failure ?", Proc of ASYNC'2006, pp 150-159.

[10] Xin Fan, M. Krstic, E. Grass, B. Sanders, C. Heer, "Exploring pausible clocking based GALS design for 40-nm system integration", DATE'2012, pp. 1118 – 1121.

[11] A. Sheibanyrad & all, "Multisynchronous and Fully Asynchronous NoCs for GALS Architectures", IEEE Design & Test of Computers 2009, pp. 572 – 580.

[12] M.N. Horak, S. Nowick, M. Carlberg, U. Vishkin, "A Low-Overhead Asynchronous Interconnection Network for GALS Chip Multiprocessors", NOCS' 2010, pp. 43 50.

[13] Y. Thonnart, P. Vivet, F. Clermidy, "A Fully Asynchronous Low-Power Framework for GALS NoC Integration", Proceedings of DATE'2010, March 2010.

[14] E. Beigné et al., "An Asynchronous Power Aware and Adaptive NoC based Circuit", IEEE Journal Of Solid State Circuits, April 2009, vol.44, pp.1167-1177.

[15] Herbert, S.; Marculescu, D., "Analysis of dynamic voltage/frequency scaling in chip-multiprocessors", ISLPED'2007, pp. 38-43.

[16] Y. Thonnart, E. Beigné, P. Vivet, "A Pseudo-Synchronous Implementation Flow for WCHB QDI Asynchronous Circuits", Proceedings of ASYNC'2012, Mai 2012.

[17] A.M. Lines, "Pipelined asynchronous circuits", Master's thesis, California Institute of Technology, 1995.

[18] P.A. Beerel, G.D. Dimou, A.M. Lines, "Proteus: An ASIC Flow for GHz Asynchronous Designs," IEEE Design & Test of Computers, vol.28, no.5, pp.36-51, 2011.

[19] M. Najibi, P. A. Beerel, "Performance Bounds of Asynchronous Circuits with Mode-Based Conditional Behavior ", Proc. of ASYNC'2012.

[20] J. Hansen, M. Singh, "A Fast Hierarchical Approach to Resource Sharing in Pipelined Asynchronous Systems ", Proc. of ASYNC'2012.

[21] Tiempo White Paper #1, "Tiempo asynchronous Technology introduction", http://www.tiempo-ic.com/company/technology.html.

[22] Tiempo White Paper #2, "Introduction to SystemVerilog Asynchronous Modeling", http://www.tiempo-ic.com/company/technology.html.

[23] N. Leblond, "How transactions viewing accelerates debug of asynchronous SystemVerilog designs", Verification Horizons, Feb. 2011, Vol. 7, N°. 1.

[24] N. Leblond, "How to Reach High Performance with Tiempo Clockless De-signs Using PrimeTime and ICC", SyNopsys User Group 2010, Austin, Texas.

[25] Marc Renaudin, "Asynchronous Design Improves Performance Process Monitoring", Published in Issue of Chip Design Magazine, September 2012.