FACULDADE DE INFORMÁTICA

PUCRS - Brazil

http://www.inf.pucrs.br

Power-Efficient Clockless Intrachip Communication Design with an Integrated High to Low Level Flow based on the Balsa Framework

Matheus Moreira, Felipe Magalhães,
Matheus Gibiluka, Fabiano Hessel, Ney Calazans

TECHNICAL REPORT SERIES

Number 075

September, 2013

Power-Efficient Clockless Intrachip Communication Design with an Integrated High to Low Level Flow based on the Balsa Framework

Matheus Moreira, Felipe Magalhães, Matheus Gibiluka, Fabiano Hessel, Ney Calazans

Pontifícia Universidade Católica do Rio Grande do Sul (GAPH-FACIN-PUCRS)
Av. Ipiranga, 6681 - Prédio 32, Sala 726 - 90619-900 - Porto Alegre – RS – BRAZIL
{matheus.moreira, felipe.magalhaes, matheus.gibiluka}@acad.pucrs.br, {fabiano.hessel, ney.calazans}@pucrs.br

Abstract

The downscaling of silicon technology and the capacity to build entire systems on a chip have made intrachip communication a relevant research topic. Besides, technology challenges point to a growth in the use of non-synchronous networks on chip, either through the use of globally asynchronous, locally synchronous (GALS) or clockless approaches. However, clockless circuit design with current automation tools is challenging, due to the fact that most of these tools focus on synchronous (clocked) design styles. This paper proposes an alternative flow to design clockless (also called asynchronous) intrachip network routers. The flow starts with high level descriptions based on the Balsa language and its associated synthesis methods. Logic synthesis results are linked to physical design in a 65nm CMOS technology and a specific standard cell library supporting several clockless design templates coupled to the Cadence Encounter environment. Given the practical utilization rates of real world intrachip network routers, the reported results show that a power economy in excess of 70% can be achieved compared to synchronous routers. Additionally, the coupling of a high level asynchronous design environment like Balsa to an adequate design flow and cell library enables unprecedented design space exploration for clockless intrachip communication modules.

Keywords: clockless design, asynchronous design, network on chip, intrachip networks, globally asynchronous locally synchronous, Balsa, design flow.

1. Introduction

A digital system is called *synchronous* when a single control signal coordinates all events occurring within it. This signal is usually named *clock*. Any digital system that does not fit in this definition is called here *non-synchronous*. A non-synchronous system may contain any number of (local) clocks or no clock at all. The latter are classically designated as *asynchronous* or *clockless* systems.

Current semiconductor technology nodes such as 90nm and below allow the implementation of systems-on-chip (SoCs) comprising dozens of intellectual property cores (IP cores or simply IPs) interconnected through some communication architecture. A straightforward approach to build such SoCs is to interconnect its set of IP cores through intrachip buses such as the IBM CoreConnect or ARM Amba architectures [1]. However, intrinsic bus architectures' limitations furthered research on intrachip networks, also called networks on chip (NoCs) research in recent years, to reach unprecedented levels of scalability and communication parallelism [2] [3]. Indeed, NoCs have already enabled effective intrachip communication architectures as discussed, for example, in [4].

Modern SoCs rely heavily on IP reuse, where IPs may employ particular standards and/or protocols and present varying design constraints. Often, IPs' requirements determine the use of specific communication protocols and/or operating frequencies. This renders the design of SoCs with multiple frequency domains far more natural than fully synchronous SoCs. Systems with multiple frequency domains that communicate with each other through some synchronization mechanism are classified as globally asynchronous locally synchronous (GALS) [5]. Expectations [6] are that scalability, technology migration needs and robustness to effects like soft errors and crosstalk will growingly impair the construction of synchronous SoCs.

Synchronization interfaces must be used at specific points of a GALS SoC [7], to allow distinct clock domains to communicate. Clearly, NoCs are natural candidates to include such synchronization interfaces, and this is an active research area. These NoCs can themselves be fully synchronous, GALS or clockless components, depending on their router design and on the router-to-router and router-to-IP interfaces design. Pontes et al. [8] present a review of non-synchronous NoCs employed in GALS SoCs and propose the NoC/SoC classification reproduced in Table 1. This work addresses the class of NoCs/SoCs corresponding to the last line of this table.

Table 1
NoC and SoC classifications as a function of router type and asynchronous interface types. Legend: R=Router, IP=IP Core [8].

R-to-R Interface	R-to-IP Interface	NOC type	SOC type
Sync	Sync	Sync	Sync
Sync	Async	Sync	GALS
Async	Sync	GALS	GALS
Async	Async	GALS	GALS
Async	Async	Async	GALS

We consider that the first two lines of Table 1 are approaches not scalable for future technology nodes, due to global clock distribution constraints and other issues, like excessive electromagnetic emissions. The third and fourth lines are potentially scalable approaches, but in large NoCs they may require that data packets cross a significant number of clock domains, leading to possibly hard to solve latency problems and over-constrained designs.

Other reasons can justify the choice of fully asynchronous NoC architectures. First, according to the last Edition of the International Technology Roadmap for Semiconductors (ITRS), the use of asynchronous logic must increase in future complex systems [6]. Consider for example the amount of global signaling used in new designs: the ITRS predictions state that an average of 20% of this signaling was expected to use asynchronous logic in 2012, but this steadily increases to 54% by 2026. Thus, dominating asynchronous design techniques seems ineluctable for future SoC designers.

Second, according e.g. to Beerel et al. [9], clockless design may potentially bring four kinds of benefits: (*i*) increase performance, (*ii*) reduce power, (*iii*) increase modularity and ease of design, and (*iv*) reduce electromagnetic interference. This work describes a case study NoC router design and the associated design flow, where the use of the asynchronous paradigm leads to significant power reductions for typical real world NoC traffic scenarios.

Regardless of all potential benefits achievable by using asynchronous design, automation for non-synchronous circuits design is still insufficient and mainstream commercial tools still focus almost exclusively on the synchronous design style [9]. One fact that corroborates this situation is the number of research works proposed in the last decade that address the adaptation of synchronous design flows to implement clockless circuits, e.g. [10], [11] and [12]. Some change is taking place in this arena though, as commercial firms like Tiempo propose the use of standard HDL languages like System Verilog to produce asynchronous circuits [13].

Besides automation tools, most asynchronous circuits may greatly benefit from the use of specific components, which are typically absent in e.g. commercial standard cell libraries.

Without specific electronic design automation (EDA) tools and the support new cell libraries to further semi-custom approaches, the design of non-synchronous modules and systems is often classified as a difficult task that requires special training of a design team and access to specific resources and tools.

This work shows that the use of currently available open source tools and libraries can already enable designs that are competitive with synchronous design, as shown here by trading increased area for reduced power. Also, by coupling high level languages to efficient physical level resources, this work shows it is possible to perform design space exploration for asynchronous modules, at least in the domain of intrachip communication architectures.

Albeit synchronous routers are conceptually easier to implement, they may prevent the achievement of the benefits expected from circuits designed without a global clock, namely low power, average case performance and robustness to process and operating conditions. Moreover, as technology nodes evolve, the complexity to design synchronous circuits increases substantially, because timing and power constraints become harder to meet. Some previous works, such as those described in [14], [15] and [16], already showed highly efficient asynchronous NoC implementations and used it in real life products, but relied in a mostly handcrafted NoC design.

This work describes the design of an asynchronous router, called Balsa Based Router (abbreviated to BaBaRouter) to support the design of GALS SoCs. It proposes a new design flow for asynchronous NoC routers design and implementation. The Balsa language allowed the description of this fully asynchronous NoC router with reduced complexity. The circuit is an asynchronous version of the well-known Hermes NoC router [17]. It was synthesized for the STMicroelectronics (STM) 65nm CMOS technology using commercial backend tools from Cadence, the Balsa Framework and ASCEnD [18], a specific standard cell library fully compatible with the Balsa framework and the native standard cell library of the chosen technology. The BaBaRouter was synthesized until the layout level, and validated by timing simulation. The Cadence framework allowed its layout generation and post-layout timing simulation. Additionally, this document compares the BaBaRouter to an equivalent version of the synchronous Hermes router in the same technology, designed through a conventional design flow using the same Cadence framework.

The rest of the paper is organized into seven sections. Section 2 describes some basic concepts about asynchronous circuits. Next, Section 3 provides an overview of the design resources necessary to develop the ideas and support the proposed implementations, which comprise the Balsa framework and language, the ASCEnD library and the original synchronous Hermes NoC router. Section 4 discusses related work. Section 5 scrutinizes the proposed asynchronous router architecture and Section 6 describes the design flows adopted for BaBaRouter and the Hermes router. Section 7 presents the validation environment, compares synchronous and asynchronous routers and discusses the obtained results. Finally, Section 8 draws some conclusions and directions for further work.

2. Clockless Circuits

Clockless circuits employ explicit handshaking among their components to synchronize, communicate and operate [19]. Characterizing a clockless design requires the choice of: (*i*) a delay model, (*ii*) a code to represent information, (*iii*) a handshake protocol, and (*iv*) a set of basic components. These are explored in the rest of this Section.

Clockless circuits can be classified according to several criteria. One important criterion is based on the delays of wires and gates. The most robust and restrictive delay model is the delay insensitive (DI) model [19], which operates correctly regardless of gate and wire delay values. Unfortunately, this class is too restrictive. The addition of an assumption on wire delays in some carefully selected forks enables to define the quasi-delay-insensitive (QDI) circuit class. Here, signal transitions occur at the same time only at each end point of the mentioned forks, which are called isochronic forks. Usually, the set of basic components of a clockless design is created to incorporate all isochronic forks needed to guarantee delay insensitivity. Thus, the design process may ignore delays altogether, just like the synchronous design process does.

There are different ways to encode information to adequately support delay models in components that communicate through handshake protocols. The use of regular binary encoding of data usually implies the use of separate request-

acknowledge control signals, in what is called a *bundled data channel* [9]. While this makes design straightforward for those used to synchronous techniques, timing restrictions between control and data signals need to be guaranteed at every handshake point, making design of large circuits hard to scale.

As an alternative, DI codes [19] [20] [21] are robust to wire delay variations, because the request signal is embedded within the data signal. An example is the class of *m-of-n codes* [21]. Given **n** and **m**, with **m<n**, an m-of-n code consists in the set of all n-bit code words with Hamming weight (i.e. the number of bits in 1 in the code word) equal to m. For example, the well-known one-hot code is an example of 1-of-n code. The use of m-of-n codes coupled to a protocol that establishes how valid codes succeed one another in a data flow allows obtaining communication with absolute insensitivity to delay variations in individual wires.

Handshake protocols can be either *2-phase* or *4-phase* [19] both illustrated in Fig. 1. Usually, 2-phase protocols operate faster, but require more hardware than 4-phase protocols. The latter require that after each data transmission wires return to a fixed logic state, the so-called *spacer*, selected among the invalid code words in the chosen code. While a 4-phase protocol increases the time to propagate values, it reduces control complexity.

One approach to achieve delay insensitivity consists in representing each data bit in a circuit by a pair of wires, in what is called dual-rail (DR) encoding (where each bit is represented using a 1-of-2 code). Let **d.t** (data true or 1) and **d.f** (data false or 0) be the names of two wires representing a single data bit. One example of 2-phase handshake using a 1-bit DR code appears in Fig. 1(a). Here, a transition in wire **d.f** signals a **logical 0** value, which is recognized by a transition in the signal acknowledge (**ack**). A transition in **d.t** signals a **logical 1** value, which is again acknowledged by a transition in **ack**. Several other 2-phase protocol implementations exist [9] [19]. Note the protocol requires that **d.t** and **d.f** never transition at the same time, and that a subsequent transition in a wire can only occur after a transition in the **ack** signal. Also, in this 2-phase protocol, data encoding varies in time. Transitions on specific wires represent data, in a clearly glitch-sensitive scheme. This requires careful logic design and because of this may incur in significant hardware overhead.

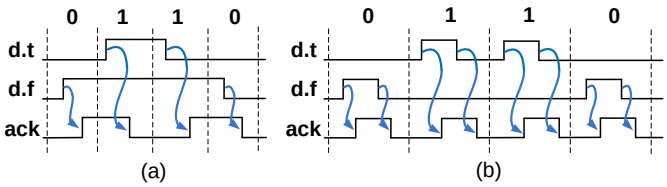
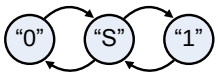


Fig. 1. Handhsake protocols types: (a) 2-phase (b) 4-phase.

Fig. 1(b) shows an example 4-phase protocol using DR code. Logical levels in wires uniquely identify data bit values. Again, let **d.t** and **d.f** be the names of two wires representing one bit of some DR code. Valid bit values are always valid code words of the 1-of-2 code ("**01**" for bit 0 and "**10**" for 1). However, after a value is acknowledged, all wires must return to a predefined value, here the **all-0s** spacer. The spacer itself is an invalid DR code word. This protocol is accordingly denominated *return to zero* or RTZ. In Fig. 1(b), the first communicated data value is a logical 0, encoded by **d.t=0** and **d.f=1**. After the value is acknowledged by a low-to-high transition in the **ack** signal, a spacer is issued, in this case **d.t=0** and **d.f=0**. Next, the **ack** signal switches to 0, signaling reception of the spacer, and a new transmission may occur. Any 4-phase protocol requires spacers when using m-of-n codes. Fig. 2(a) shows the RTZ conventions and Fig. 2(b) shows the valid transitions of a 4-phase DR encoded value.

Value	*d.t*	*d.f*
Spacer	0	0
Logical 0	0	1
Logical 1	1	0
Invalid	1	1

(a)



(b)

Fig. 2. Example of (a) 4-phase DR encoding and (b) 4-phase DR values transition.

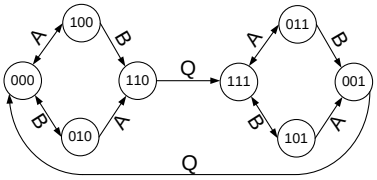
Synchronous design is based on a well-known template formed by the interspersion of combinational blocks of logic gates among a set of registers controlled by the clock. Logic gates and registers are the components of the template, and any interconnection of these where any feedback path mandatorily passes through at least one register is a valid synchronous circuit. Multiple clockless design templates have been proposed [9], each with a proper set of components and rules to interconnect them.

In fact, most clockless design templates employ a set of components distinct from those used in the synchronous template. These templates can greatly benefit from the availability of basic components other than ordinary logic gates and flip-flops available in current standard cell libraries, which constitutes a motivation to extend existing synchronous cell libraries. Such components include e.g. special registers, event fork, join and merge devices, as well as metastability filters. Although most of these may be built from logic gates, this is often inefficient.

A fundamental device that enables to build most of these elements more effectively is the C-element. The importance of C-elements is that they may help in the synchronization of independent events. Fig. 3(a) depicts the truth table and Fig. 3(b) shows a transition diagram for an ordinary 2-input C-element.

A	B	Q_i
0	0	0
0	1	Q_{i-1}
1	0	Q_{i-1}
1	1	1

(a)



(b)

Fig. 3. Simple 2-input C-Element specification: (a) truth table (b) asynchronous state transition diagram.

A C-element output switches only when all inputs have the same logical value. When inputs A and B are equal, output Q assumes this same value. However, when inputs are different, the output keeps its previous logic value. The asynchronous state transition diagram of Fig. 3(b) for the C-element has ver-

tices containing values of inputs and output in the order ABQ_i.

Multiple clockless templates use C-Elements to implement their logic blocks. For instance, Fig. 4 shows an example of a QDI 4-phase DR register (also called a half-buffer [9]). This register requires only resettable C-Elements for event sequencing [22] [23]. Not shown, but usually present in these components is the validity detector, that provides an acknowledge signal to feed the handshake with the previous stage. For the half-buffer, the detector uses an OR gate for each pair of data wires. These bit validity signals are synchronized with simple C-Elements.

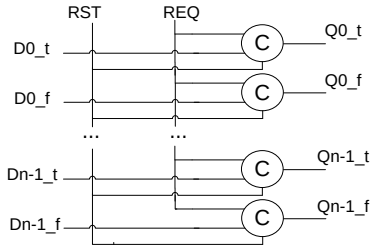


Fig. 4. The structure of an N-input QDI 4-phase DR half-buffer.

To implement Boolean functions without losing the delay insensitivity property, different component schemes can be employed for asynchronous circuits in general and specifically for QDI 4-phase DR circuits. One of the most used, due to its simplicity, is the delay insensitive minterm synthesis (DIMS) [9]. In this approach, all minterms of the input variables are generated by C-elements and are then combined to perform a given function. This is similar to two-level logic implementations used e.g. in programmable logic arrays (PLAs). As an example, Fig. 5 shows a QDI 4-phase DR DIMS half adder.

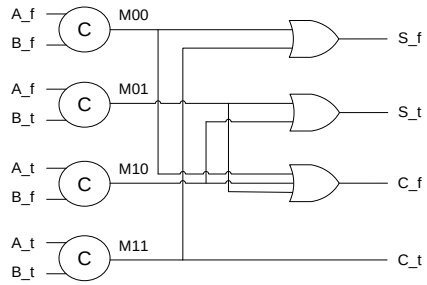


Fig. 5. QDI 4-phase DR DIMS half adder.

All valid code minterms of A and B are generated by the C-elements and the outputs are computed using OR gates as in PLAs. S_f must be at logical 1, identifying a logical 0 value in the sum, only when both A and B inputs have the same value, which means that either M00 or M11 will be at logical 1. When inputs have different values, identifying a logical 1 value in the sum, S_t will be at logical 1. The carry signal is computed in a similar way. Although DIMS implies a large amount of hardware to compute each minterm, it is widely used in asynchronous design for its simplicity and robustness.

3. Design Resources

The implementations proposed here capitalize on three open source design resources, each discussed in the next Sections: (*i*) the Balsa framework and language; (*ii*) The ASCEnD asynchronous cell library, and (*iii*) The Hermes NoC, more specifically, the specification of its router.

3.1. The Balsa Framework and Language

With the growing interest for asynchronous circuits, different research tools have been proposed to automate the process of designing asynchronous circuits. Among these, Balsa stands as a comprehensive, open source environment [19] [24]. The tool was designed and is maintained by the Advanced Processor Technologies Group from the University of Manchester. Version 4.0 of the Balsa framework was released in June, 2010, at http://apt.cs.manchester.ac.uk/projects/tools/balsa/.

Balsa is both a language to describe asynchronous circuits and a framework to simulate and synthesize them. The compilation of a Balsa description is transparent, since language constructs are directly mapped into handshake components [19]. In this way, it is relatively easy for the designer to visualize the circuit-level architecture of a Balsa description. Moreover, Balsa description modifications reflect in predictable changes in the resulting circuit, which means that the designer has a clear control of the generated hardware.

Balsa requires the designer to furnish the order of handshake events through the Balsa language operators. These events imply the communication between handshake components (data exchange or control only) and can be specified as occurring sequentially or concurrently. Moreover, handshake components are transparent and are derived from higher abstraction language constructs, such as `if/else`", `case`, `select` and `arbitrate`. The two latter are very important for asynchronous circuits; they allow the designer to control and arbitrate events without a discrete notion of time. Therefore, one of the main advantages of using Balsa to describe and implement asynchronous circuits is that communication control between handshake components is abstracted. Due to the fact that all the complexity of describing control signals is transparent, the designer just needs to describe circuit behavior through a data flow approach. The tool automatically generates handshake components and required control circuits.

After translating Balsa descriptions into a network of handshake components, different technologies can be used to map this to a circuit. This work uses an in house standard cell library designed to support asynchronous circuits, fully compatible with the Balsa framework. In this way, mapped netlists generated by Balsa can be imported into back-end commercial tools for physical implementation. Last but not least, Balsa allows the designer to choose a design style for the generated circuit. The available choices are based on the data encoding, and can currently be: bundled-data, DR and 1-of-4.

3.2. The ASCEnD Asynchronous Cell Library

ASCEnD is a freely available standard cell library for supporting asynchronous ICs semi-custom designs. The library has its components designed at the layout level and can be easily ported to different CMOS technologies, as it is supported by a very modular design flow with automated tools.

Currently, ASCEnD is available for the 65nm STMicroelec-

tronics CMOS technology (ASCEnD-ST65) and is being ported to the IBM 130nm CMOS technology available through the MOSIS service. A total of 504 C-elements compose the library. It contains mostly C-elements with varying driving strengths (speed to charge/discharge a load), functionalities and topologies. There are components in the library useful for both high speed and low power design. ASCEnD-ST65 components' design and tradeoffs are discussed in detail in references [18] and [25].

ASCEnD also contains 4 versions of metastability filters. Each implementation employs different transistor sizes and, consequently, presents distinct driving strengths. Together with a commercial standard cell library containing basic combinational logic gates and flip-flops, the library can be used to implement a large class of asynchronous circuits' templates. For the designs proposed in this work, only C-Elements are required other than conventional standard cells, the latter being available in the standard cell library provided with the STM technology design kit.

3.3. The Hermes NoC and Router

This work uses as base NoC architecture Hermes [17], a well known open source, packet switched NoC framework provided by the research group of the authors. The block diagram of the Hermes NoC router appears at Fig. 6. Hermes is an open architecture with a set of tools available to facilitate generation and simulation of NoC instances.

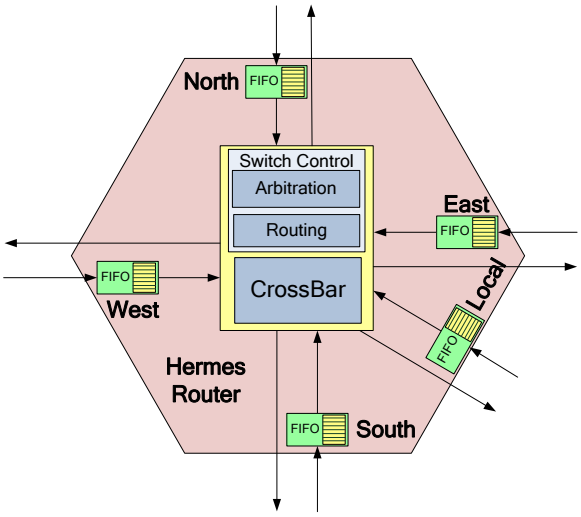


Fig. 6. Hermes router block diagram [17].

Hermes assumes the use of 2D mesh topologies and the Hermes router is composed by three main module types: input buffers, a shared logic used for routing, called *switch control*, and a crossbar. Input buffers are first-in-first-out (FIFO) memories, with a local logic that ensures flow control. Basic versions of this NoC employ the straightforward XY algorithm (others are available) to route packets over the network. The scheduling of the routing requests uses a round robin algorithm for arbitrating access priority to the routing logic. Each router may have up to five bidirectional ports (North, South, East, West and Local) that include an input buffer and one of the crossbar outputs. The priority of a port to access the routing logic depends on the last port that was granted access to it. The priority scheme is intended to avoid starvation, by ensuring that all input requests will be eventually granted. Input buffers (one for each port, totalizing up to five buffers per router) are responsible for storing pending packets waiting for arbitration or transmission. The switch control block, composed by arbitration and routing logic units, is responsible for routing incoming packets and to manage their access to an appropriate output port through the crossbar.

The Hermes NoC was proposed to support a subset of the Open Systems Interconnection (OSI) reference model. Only Transport, Network, Data Link and Physical layers are defined. However, only the three last layers are implemented within the router itself. The upper layer (Transport) is a responsibility left to the network interfaces of IP cores attached to the NoC. The bottom layer of the stack is the Physical layer, responsible for connecting network interfaces and switches to their neighbors. The Physical layer also defines aspects of transmission and physical characteristics of the network, such as flit width.

The intermediate layer (Data Link) is responsible for creating links between nodes. Logical connections enable packet transmission. The transmission protocol employed in the Hermes router in this work is handshake. The choice of using this protocol, instead of the more common credit-based flow control also available for Hermes, was made to provide a fair comparison between the synchronous and asynchronous router implementations.

The Network layer provides a logical addressing scheme and delivers end-to-end packets. Hermes uses packet switching, which means that this layer also fulfills the task of routing packets through the network. Up to five simultaneous routing connections can be established in each router. A good understanding of the Hermes NoC router is crucial to grasp how BaBaRouter is implemented.

4. Related Work

As far as the authors could verify, six other fully asynchronous NoC routers are described in the literature: Asynchronous QNoC [26], MANGO [27], A-NoC [28], ASPIN [29] and Hermes-A/Hermes-AA [30] [31], the latter proposed by the research group of the authors.

The BaBaRouter design employs a fully QDI asynchronous template. The only revised work that pledges the same approach is A-NoC [28]. The use of a QDI template, even if it increases design complexity somehow, is justified by the fact that it enables overall delay insensitivity, leading to designs that are robust to process, voltage and temperature variations. MANGO, asynchronous QNoC and A-NoC claim support to quality of service through the use of virtual channels and/or special circuits. A-NoC is the most developed of the proposals and presents the best overall performance. It has been successfully used to build at least two complete integrated circuits [32]. Hermes-AA presents adaptive routing schemes to deal with unpredictable application dynamic behaviors, allowing packets to take different paths through the network every time congestion is detected. Albeit currently BaBaRouter employs

only an XY routing algorithm, adaptive algorithms can also be included in the router.

All revised NoCs report the use of synchronous design flows and tools for implementing asynchronous routers. This generates a greater workload, where the focus of the work becomes the adaptation of the circuit, in order to guarantee asynchronous characteristics using tools conceived for synchronous design. Besides, asynchronous modules are handcrafted. This means that to implement handshake elements, asynchronous control logic or even asynchronous components such as C-elements, manual design is omnipresent.

Since Hermes-A and Hermes-AA are very similar we restrict attention to the latter. The development of Hermes-AA [31] demonstrated that adapting typical tools and standard cell libraries to design asynchronous circuits is challenging. Hermes-AA functional description took months to be implemented and validated. Its logic synthesis capitalized on lots of manual labor to generate a functional netlist. In contrast, BaBaRouter first functional version was described and validated in the Balsa Framework in roughly one week, and Balsa automatically generates a functional logic netlist. Both designs require special standard cells in their synthesis, which are available e.g. in ASCEnD. The complexity of physical synthesis for both Hermes-AA and BaBaRouter are approximately the same. This is because they were implemented in the same technology and through the same backend tools. However this is not a critical part for asynchronous circuits design.

The main three differences between the work presented herein and those discusses above are: (*i*) the use of a high level language and framework for design entry and synthesis, (*ii*) the availability of a library of cells to support clockless design and (*iii*) an automated connection between the high level framework and the physical synthesis framework. This enables easy design space exploration for asynchronous circuits, a feature absent in previous works.

5. The BaBaRouter Architecture

The BaBaRouter was described using the Balsa language. The router can be implemented for different asynchronous templates, due to the fact that a Balsa description abstracts data encoding methods, as well as communication protocols. Thus specific design template decisions are made during synthesis and mapping to a specific technology. Moreover, as the Hermes router, BaBaRouter is parameterizable. In fact, the latter has exactly the same functionality and overall structure as the Hermes router.

As Fig. 7 shows, BaBaRouter has four fundamental block types: (*i*) the input FIFO buffers and (*ii*) the input control (IN CTRL), which together are equivalent to an input buffer of Hermes, (*iii*) the switch control and (*iv*) the crossbar. All blocks are built with handshake components and each block is itself a handshake component. Consequently, blocks make use of handshake channels to communicate and synchronize. All channels in Fig. 7 abstract existing request and acknowledge control signals. Also, five input and five output ports compose the router, East (0), West (1), North (2), South (3) and Local (4), each with input and output interfaces. Again, ports 0-3 interconnect routers and port 4 interconnect router and IP.

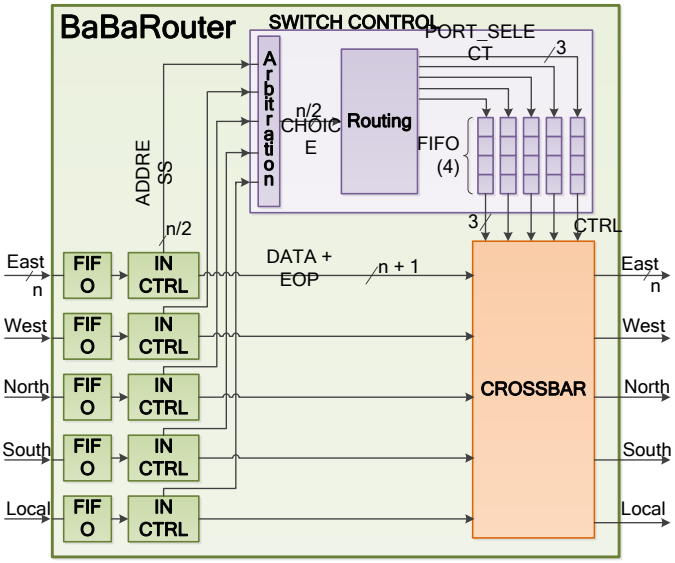


Fig. 7. BaBaRouter implementation block diagram. Communication and synchronization takes place through handshake channels. Control signals are implicit in the Figure. The flit size is denoted by n, which is used to specify the adopted internal channel widths.

5.1. The Input FIFO and IN CTRL Modules

The input FIFO is a basic structure composed by sequentially connected registers. Width and depth of the FIFOs are a choice of the designer. Neighbor registers handshake to each other as data flows through the FIFO. The width of router data channels is the same as the router flit width. Throughout this work we assume the use of 8-bit flits and 8-flit deep FIFOs but these values are both parameterizable at design time.

The IN CTRL block is responsible for controlling packet information. A packet in BaBaRouter follows the Hermes format, as depicted in Fig. 8

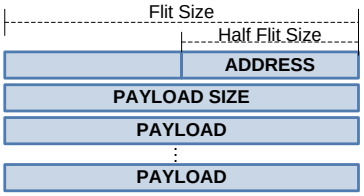


Fig. 8. BaBaRouter packet structure.

The first flit has the address in its lower half, followed by the payload size, in the second flit, trailed by the payload, i.e. all following flits. When the IN CTRL of a given port detects a new communication, it sends the address to the switch control through a (n/2)-bit channel, where n is the flit width. Then, it sends the data, together with an end of package (EOP) signal of value '0', to the crossbar, through a (n+1)-bit channel. The next flit is the payload size. IN CTRL sets an internal register to the value contained in this flit, resets its internal counter and propagates the flit to the crossbar with the '0' EOP value. Next, IN CTRL increments its counter for each new flit received and propagates the flit to the crossbar with EOP='0'. When it detects the last flit, by comparing its counter to the

internal register that keeps the packet size, it signals EOP='1'.

5.2. The Switch Control Module

The switch control is responsible for calculating the destination of the packet received in a given input port using the address received from IN CTRL and the internal router address. All input ports share the switch control module to route packets. Therefore, requests from these ports to the switch must be arbitrated. This is required to solve conflicts. For instance, if two ports receive a new package at the same time and the delays of each path to the switch control are equivalent, two requests to use the switch control block arrive at the same time. To choose what packet will be routed first, the switch control arbitrates requests through the Arbitration block. This can be achieved safely at high level using the **`arbitrate`** Balsa construct, which forces the design to become robust to metastability.

Next, the Routing block computes the path the arbitrated packet must follow. Currently, packets are routed using the XY algorithm. Other routing algorithms can be easily adapted, due to the high modularity inherent to asynchronous circuits and in particular to Balsa descriptions and the BaBaRouter's architecture. The Routing module produces a 3-bit code for choosing one of the five output ports. The switch control has a 3-bit 4-position output FIFO for each router output port. This decides to what output port the communication from some input port is propagated. For instance, consider that a router with address "11" receives a new packet in the East port with destination "11". The switch control will compute that the packet must follow to the Local output port. Then, it will write in the local output FIFO the value of the East router input (0). In other words, it will inform that the Local output port needs to be reserved for the East input port. Each new request to the Local output port will be queued in the output FIFO. The FIFO will be full when all input ports, but the Local, request the local output port. Note that it is impossible for such FIFOs that any routing request is pending to enter the full FIFO. This justifies the fixed size of the FIFOs and guarantees that the switch control alone will never cause communication to stall.

FIFOs in the switch control outputs are useful to avoid starvation while ensuring fair service to all ports, when multiple inputs request the same output port. A same input port is not allowed to request the same output port twice consecutively. This approach is indeed fairer than the Round Robin arbiter of Hermes. The approach provides fair arbitration of priorities and costs little hardware. Implementing the original Hermes Round Robin in an asynchronous style would be a much more area-consuming and complex task.

5.3. The Crossbar

The crossbar binds an output port to an input port. This is done with the information generated by the switch control module. When the switch control routes a packet to an output port, it signals to the crossbar through the CTRL channels which input port must be bound to a given output port. The crossbar then binds the ports and propagates the packets received from the IN CTRL until an EOP='1' is received. Then, the port can be bound to a new input port.

Only when the whole packet is transferred, the crossbar finishes the communication with the switch control for a given output port, generating an acknowledge signal. Thus, a new communication for any of the remaining output ports can take place at any time. BaBaRouter can have data flowing from different inputs to different output ports concurrently. The router saturation point is when all input ports are granted to communicate with different output ports. In this case, five paths are simultaneously established. Fig. 9 shows the simplified crossbar block diagram.

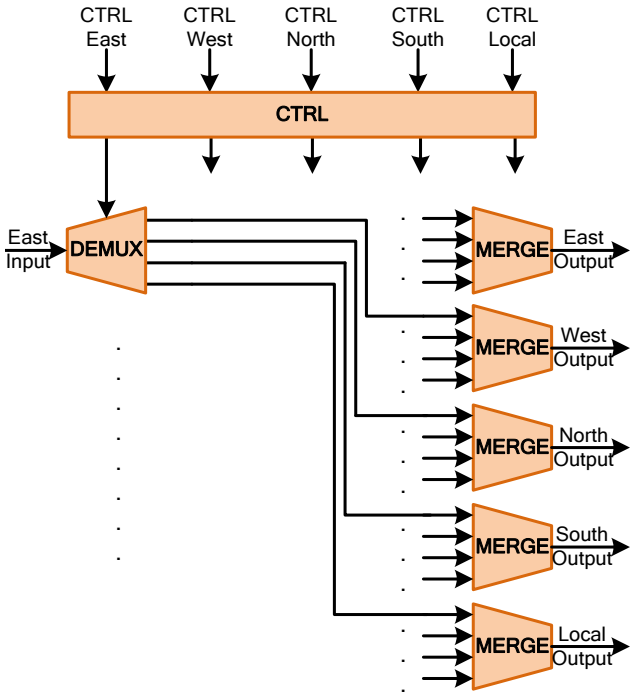


Fig. 9. BaBaRouter Crossbar simplified block diagram.

The crossbar consumes a large amount of hardware because for each input port four channels must exist, one for each possible output port. The choice of what channel to use can be viewed as a demultiplexer (DEMUX) where the control input derives from information coming from the crossbar CTRL block. This block receives data from the switch control along with EOP signals (not shown in Fig. 9) and binds each input port to an output port, producing control signals to the demultiplexers in each input. Channels destined to a same output port are merged to the actual output port. This was the best approach that the authors could find for a Balsa description of a crossbar, to guarantee concurrency of communication for different input/output port pairs.

5.4. Dataflow in BaBaRouter

As Fig. 10 shows, when the first flit of a packet is received in an input of the router, it is propagated through the input FIFO and reaches the IN CTRL, which feeds the switch control and the crossbar. The switch control decides the path that the packet must follow and associates an output port to the input port. This can be done concurrently for all output ports, as the switch control generates routing information.

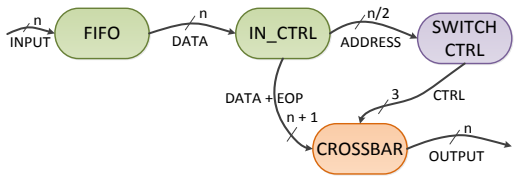


Fig. 10. BaBaRouter dataflow for the first flit of each packet.

As Fig. 11 shows, when the following flits are received in the input port, they follow directly to the output port until the last flit is received. This is due to the fact that all active CTRL channels of Fig. 7 are locked until the respective IN CTRL block signals EOP='1'. When the crossbar detects this situation, it sends the last flit to the output and clears the associated CTRL channel by issuing an acknowledge signal to it. From this point on, that output port may be used for a new packet transmission.

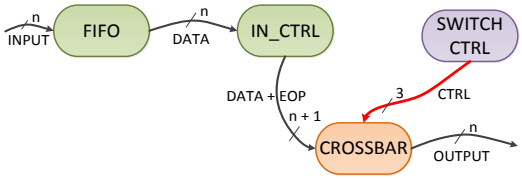


Fig. 11. BaBaRouter dataflow for flits after the first.

6. Routers Design Flow

The BaBaRouter and the Hermes Router were both implemented in the STM 65nm CMOS technology, to compare the obtained circuits. This Section describes the design flows used to produce these functionally equivalent routers.

6.1. The BaBaRouter Implementation Flow

Fig. 12 illustrates the design flow adopted for BaBaRouter. The functional behavior of this router was initially described in the Balsa language and compiled into handshake components through the Balsa compiler (Balsa-c), producing a Breeze Netlist [24]. This netlist contains handshake components only, and can be simulated in the Balsa Framework to validate its functional behavior (with the Balsa Simulator).

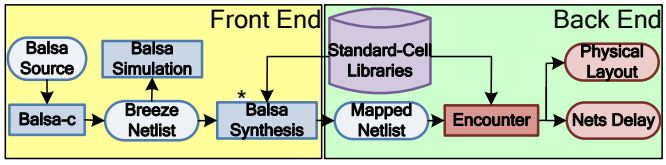


Fig. 12. The BaBaRouter design flow.

After extensive simulation of different random traffic scenarios, the design can be mapped to a specific technology. As mentioned before, this work employs the ASCEnD standard cell library and the core standard cell library of the 65nm technology. Both were made compatible with the Balsa framework for synthesizing the router in a QDI, four-phase DR style, through the use of in-house tools. This synthesis produces a netlist of gates, which is described in Verilog and is fully compatible with commercial back-end tools.

The generated mapped netlist is imported into the Cadence Framework (Encounter) to create the semi-custom physical layout. After place and route, the circuit is extracted and the delay of internal nets is annotated and exported to a standard delay format (SDF) file. This file is the source to conduct timing simulation, for validating the correct behavior of the physical implementation. After extracted, the generated circuit is again extensively simulated.

The focal point of the asynchronous design process is to obtain the Mapped Netlist description compatible with the employed physical synthesis process. The Balsa Front End allows design capture, simulation at the level of communicating processes, automatic synthesis and technology mapping. Design capture and simulation are independent of the final choice for an asynchronous template, which is left for the Balsa synthesis step (marked with * in Fig. 12). In this way, designers have more freedom to explore the design space, because the designed is not constrained by a choice for asynchronous template. This is in contrast to the use of structural design approaches such as those revised in Section 4, where the asynchronous template must be chosen at the design capture step.

Currently, Balsa Synthesis supports three asynchronous templates: (*i*) bundled-data; (*ii*) DI dual-rail; and (*iii*) DI 1-of-4. A set of around a dozen synthesis options is available to further personalize each of these templates. For example mapping may target Xilinx FPGAs or the ASIC Cadence design flow. More relevant to ASIC design space exploration, the logic used to implement basic gates may be the already mentioned DIMS we assume, the Null Convention Logic (NCL) proposed by the Theseus Logic Inc. or a form of balanced logic, used to produce asynchronous circuits robust to power attacks. Several other options are available.

To enable efficient technology mapping, the Mapped Netlist should be straightforward to implement during physical synthesis. This is achieved by associating the technology core library to the ASCEnD library of asynchronous components. Currently, ASCEnD contains much more functional components than those required by any Balsa Mapped Netlist.

It is useful to study the Balsa description of some blocks of the design, to assess the usefulness of the Front End. Fig. 13 shows the description of the BaBaRouter input FIFO and the register it employs. The first procedure (**`Register`**, in line 3) is the description of an asynchronous register with a parameter **`width`**, that defines how many bits it can store, an input **`i`** and an output **`o`**. The procedure also uses a variable **`x`** (line 8), that stores the data. **`Register`** input and output are just handshake channels; the storage hardware itself is denoted by the declared variable. The description of the **`Register`** behavior comprises lines 10 to 13, delimited by the reserved words **`begin`** and **`end`**. The semicolon in line 11 denotes that commands in lines 11 and 12 are sequential. Concurrent commands are also explicitly specified with the two vertical bars operator (||, as noted e.g. in line 29). The **`Register`** procedure consists of an endless loop (line 10) that waits for a handshake request in input channel **`i`** and as it occurs, stores the input data in variable **`x`** (line 11). Next, it requests a communication in the output channel **`o`** to propagate the data stored in **`x`** (line 12). At the end of the loop (in line 13), input **`i`** waits for

a new request. The semicolon of line 11 is responsible for the sequential semantics of the communications **i →x** and **x →o**.

The **Register** design is used as a module to build an asynchronous **fifo**, the next procedure in Fig. 13. The later consists in a parameterizable number (**depth**) of interconnected registers. The **width** of the register instance is also parameterizable.

```
1  import [balsa.types.basic]
2
3  procedure Register (
4          parameter width : type;
5          input i : width;
6          output o : width
7  ) is
8          variable x : width
9  begin
10         loop
11                 i -> x;
12                 o <- x
13         end
14 end
15
16 procedure fifo (
17         parameter depth : cardinal;
18         parameter width : type;
19         input i : width;
20         output o : width
21 ) is
22         procedure reg is Register(width)
23 begin
24         if depth = 1 then
25                 reg(i,o)
26         | depth >= 2 then
27                 local array 1 .. depth-1 of channel c : width
28                 begin
29                         reg(i,c[1]) ||
30                         reg(c[depth-1],o) ||
31                         for || i in 1 .. depth-2 then
32                                 reg(c[i], c[i+1])
33                         end
34                 end
35         else print error, "zero length fifo specified"
36         end
37 end
```

Fig. 13. The Balsa description of the BaBaRouter `fifo`.

The **fifo** interface consists in one input channel **i** and one output channel **o**, which share the same data **width** as the (internal) registers. The internal declaration of procedure **reg** binds to the **Register** module parameterized by the **fifo** width (line 22). This language construction is similar to the declaration of a component in VHDL. The **fifo** behavioral description (lines 24 to 36) tests the **depth**, using a construction with semantics similar to that of a conditional *generate* command in VHDL. If the depth is 1 (line 24), a single register is instantiated (line 25). Otherwise (line 26), the description instantiates one register for the first position (line 29), one register for the last position (line 30) and depth-2 registers for the intermediate positions (lines 31 to 33), using a language construction similar to a *for generate* in VHDL. Note that all registers are configured to operate in parallel, through the concurrency operator (||) at the end of each line and in the *for* construction of line 31. In this way, there is no sequential operation at the **fifo** level, only inside each register. Communication and flow control are implied in the handshake and are independent of the specific asynchronous design style of the implementation.

The graphical representation of the resulting Breeze Netlist for the **fifo** of Fig. 13 parameterized with **width**=8 bits and **depth**=8 registers, appears in Fig. 14. The wire fork nodes (noted **W^**) are used to enable the operation of registers. Loops (*), sequencing (;), fetch (→) and the local variables **x**, that can each store 8 bits of data, are according to the **Register** description. The synchronization components (**.(s)**) are required to adapt the output channel of each previous buffer of the **fifo**, because output channels and input channels of consecutive registers are merged in a single channel. Data transmission starts through the **fifo** input **i**, in the rightmost part of Fig. 14, which feeds the input channel (→) of the first register (**reg#1**). As soon as there is data available in this channel, it is stored in the local variable **x**, which writes the data in **reg#1** output channel (->), once the latter is free to receive it. This process goes on through the registers until the data reaches the last register, which writes this data in the **fifo** output channel **o**, at the left of Fig. 14. As each **Register** propagates its data to the next register, it is free to receive new data. This model mapped to handshake components, or more specifically to Breeze components, can be simulated independently of the asynchronous design template to use. This guarantees design space exploration is possible before deciding for a template.

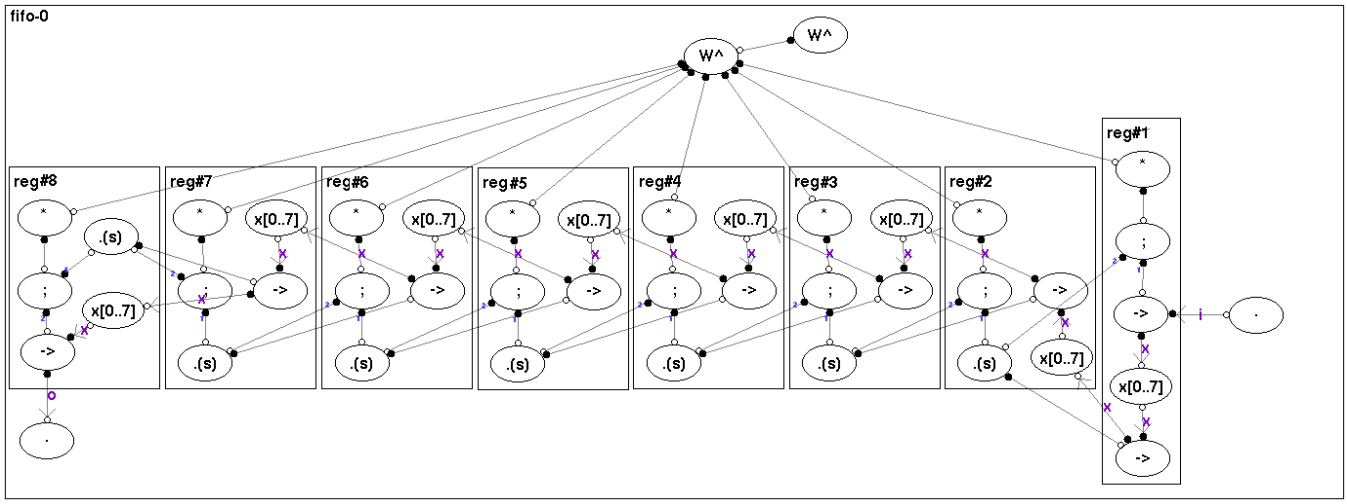


Fig. 14. The Breeze Netlist graphical representation for the BaBaRouter `fifo`.

In the environment proposed here, after validating the design at this high level, it can be synthesized to a target technology where Breeze components are automatically mapped to standard cells. Moreover, as it can be seen from Fig. 13 and Fig. 14, changes in Balsa the description lead to predictable changes in the Breeze netlist, which means predictable changes in the resulting hardware after the synthesis.

6.2. The Hermes Implementation Flow

For the implementation of the Hermes Router, a classical synchronous design flow based on tools from Cadence was employed, as Fig. 15 shows. The router description in VHDL was automatically produced by Atlas [33]. Using RTL Compiler, the design was elaborated and mapped to the basic standard cell library of the STM 65nm technology.

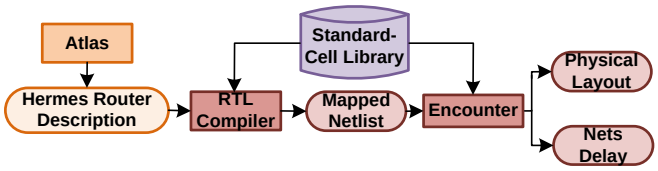


Fig. 15. The Hermes Router design flow.

The Mapped Netlist was used in Encounter to generate the physical layout of the circuit. Similarly to the design flow adopted for BaBaRouter, after place and route, the circuit was extracted and the delay of its internal nets annotated. The correct operation of the circuit was validated through timing simulation.

The maximum operating frequency achieved after synthesizing Hermes was of 1.25GHz. This is due to the fact that its project is highly constrained. The internal logic of the router is sensitive to rising and falling clock edges and the generated critical path comprises registers sensitive to alternate clock edges. In this way each phase of the clock needs to be long enough to respect constraints. This resulted in a rather long clock period, 0.8ns.

7. Clockless versus Synchronous

After validating the physical design of both routers, the obtained results were compared. A simulation scenario was defined in order to evaluate the circuits and conduct power analysis. All results were obtained for the STM 65nm CMOS technology, using typical fabrication process parameters and operating conditions of 1V for supply voltage and 25°C of room temperature.

7.1. Physical Design

Both routers were implemented using the same device types (general purpose transistors with typical threshold voltage). Moreover, both designs use 7 metal layers and a core density of 95%. Table 2 shows physical information on the layouts.

Clearly, BaBaRouter occupies much more area than the Hermes router, roughly 4.4 times. This is expected, due to the fact that EDA tools for synchronous designs are much more developed than the ones for asynchronous projects. Moreover, the standard cell library employed in the design of Hermes is a complete set of typical cells, with a large variety of complex logic standard cells that can be used by the physical design tools. This results in a much more optimized netlist.

In contrast, Balsa still lacks some functionality in its synthesis approach. For example, the environment is not able to automatically choose adequate cell driving strength parameters (although these area available in ASCEnD [25]), which would potentially lead to more optimized designs. In other words, Balsa synthesis forces that all cells of a same functionality have the same driving strength (i.e. the same capacity for charging or discharging an output load). In this way, some cells can be underutilized. In some places of the design weaker (thus smaller) cells could be employed. Furthermore, the library employed for the synthesis consists of C-Elements and basic gates only. Thus, generating complex asynchronous components requires more cells. For example, since the circuit employs DR encoding and the choice of logic blocks for Balsa synthesis in this case is to use the DIMS scheme to guarantee delay insensitivity, DR OR and AND gates must be constructed using C-elements and ordinary OR and AND gates. A richer standard cell library could certainly generate smaller circuits. This is left as future work. As for the input and output interface signals, BaBaRouter employs DR encoding, where each data bit is represented in two wires. This doubles the number of data wires used by Hermes. However, some control signals required by Hermes are not required by BaBaRouter, such as the request signal (for handshaking communication), which is encoded in its DR data signals. In this way, the total number of IO pins is not exactly the double.

Table 2
Comparison of BaBaRouter and Hermes Router physical implementations.

	BaBaRouter	Hermes Router
Standard Cells	10,007	1,822
Standard Cells Area	42,797 µm²	9,625.2 µm²
Standard Cells – Physical Cells Area	40,506.9 µm²	9,174.88 µm²
IO Pins	173	102
Total Wire Length	214.960 mm	46.196 mm

From the standard cell area of BaBaRouter design, subtracting physical cells (fillers and tap cells), 22,376.56µm² are required by C-elements. This means that over 55% of the required circuit area is designated for such cells, testifying the impact of the C-element in asynchronous designs. There are different ways to implement the functionality of a C-element in a standard cell, as [34] shows, each with its advantages and drawbacks. Therefore, by choosing different C-element implementations, area, power and speed tradeoffs could be obtained. Once more, if Balsa could select cells considering electrical behavior and area consumption for a same functionality, a more optimized circuit could be obtained, which is not the case currently.

7.2. Simulation Scenario

As explained in Section 6, after physical synthesis the circuit is extracted from the generated layout and delays of its internal nets are annotated. These were the source to conduct

timing simulations of the routers. To do so, a scenario where the routers are operating at the highest possible throughput was described and simulated. Fig. 16 describes the scenario, which consists in simulating the central router of a 3x3 2D mesh NoC, with packets flowing in all its input and output ports. The specified targets for each input bind each input to a distinct output. As explained in Section 0, the maximum operational frequency for Hermes was of 1.25GHz. However, due to the choice of using handshake control flow, each flit takes at least two clock cycles to be transmitted. Therefore, each port may have a maximum throughput of 625Mflits/s. Since the router may have up to five concurrent communications, the maximum overall throughput of the Hermes router is 3.125Gflits/s.

For BaBaRouter, the maximum throughput cannot of course be measured in the same way. It is taken as the average delay to transmit a flit in timing simulation. The conducted timing simulations showed that the router is capable of transmitting, in average, one flit each 3.164ns. This means that the throughput of BaBaRouter is 316Mflit/s per port or 1.58Gflits for the whole router. Results show that, in best case situations, it can achieve a maximum of 1.750Gflits/s.

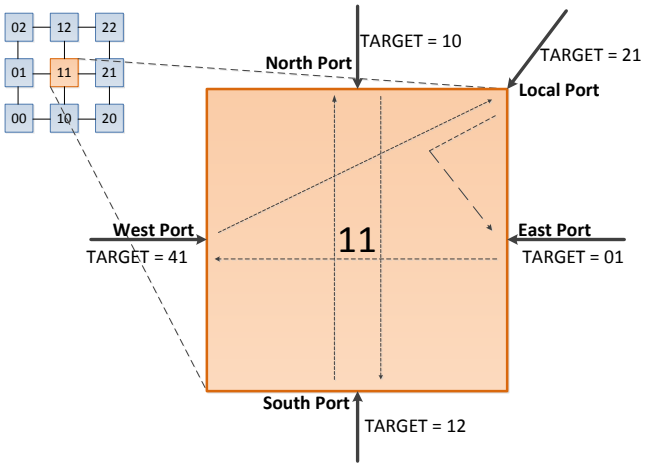


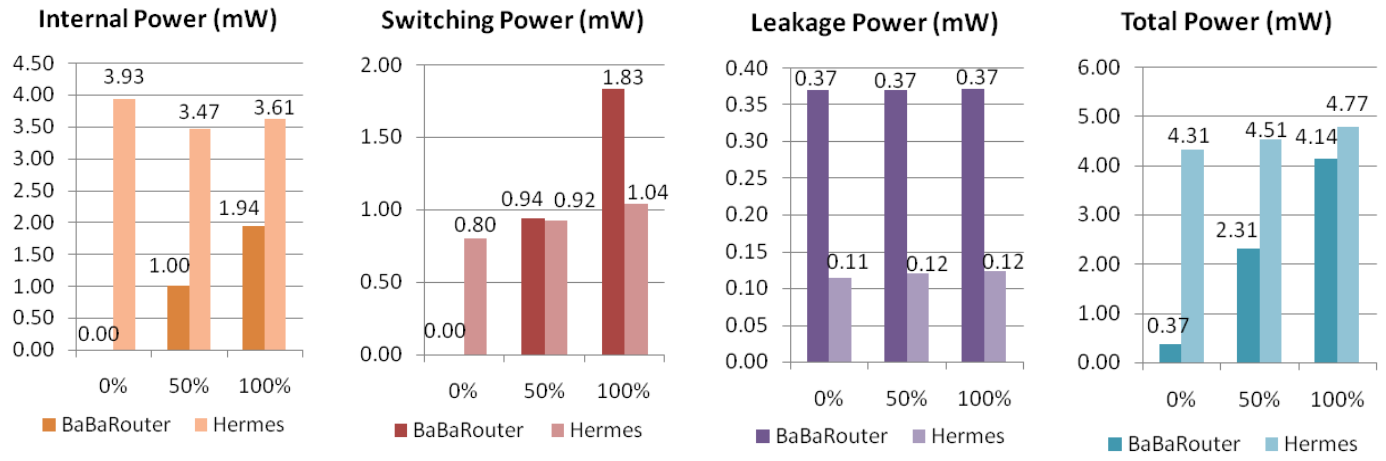
Fig. 16. Scenario to validate functionality of the routers.

Considering the average delay of BaBaRouter, it displays a throughput that is roughly 50% of that of the Hermes router. Albeit not desired, it was expected. This is due to the fact that, as explained before, Balsa cannot select different driving strengths cells for a same functionality. Thus, cells end up underutilized. This means that, in order for the circuit to operate faster, faster cells should be employed in these cases.

7.3. Power Analysis

For the power analysis, four simulation scenarios were generated. In the first, routers were left idle (0%). In the second, routers are transmitting data 50% of the time. The rest of the time routers remain idle. For the third scenario, routers operate sending data whenever they can for the whole simulation time (a 100% load). A last scenario consists in a more realistic operating condition for NoCs where only 20% of the time there is traffic crossing the router [35]. For BaBaRouter, data is asynchronously inserted in the inputs at the maximum possible rate. For Hermes, a 625MHz clock was employed, to have a fair power consumption comparison, since this is equivalent to the average throughput of BaBaRouter. This normalizes the throughput for both routers. Each scenario was simulated for 100µs and the switching activity of the routers' nets was annotated and exported to a toggle count format (TCF) file. This file was the source to conduct the power analysis.

Fig. 17 shows the obtained power results for the first three scenarios. The charts show the internal power (a), switching power (b), leakage power (c) and total power (d) consumption for scenarios, with loads 0%, 50% and 100%. As Fig. 17(d) shows, when idle, BaBaRouter consumes less than 9% of the total power of Hermes. This is a characteristic of asynchronous circuits. Inactive parts of the circuit consume only the static power, due to leakage currents. However, this power consumption is very small. In contrast, in a synchronous circuit, even when idle, each clock cycle activates all registers in the circuit, causing excessive power consumption.



(a) **(b)** **(c)** **(d)**

Fig. 17. Power results and comparison for BaBaRouter and Hermes, internal power (a), switching power (b), leakage power (c) and total power (d). Three scenarios were evaluated, routers always idle (0%), routers transmitting data 50% of the time (50%) and routers always transmitting data (100%).

When routers are sending data 50% of the time, BaBaRouter still consumes around 50% less power than Hermes. This is due to the fact that it presents lower internal power consumption. As for the switching power, the consumption is slightly higher. This is due to the fact that the circuit has a bigger path from an input to the output to which it is bound, requiring more capacitances to be charged or discharged.

For the simulation where routers operate at the maximum data injection rate, Hermes consumes roughly 15% more power than BaBaRouter. This shows that even with the bigger area required by the latter, asynchronous circuits are well suited for low power applications. All the conducted simulations showed that BaBaRouter is more power efficient than the synthesized Hermes in terms of total power consumption.

As for the leakage power consumption, which can be constraining in newer technologies [36], BaBaRouter consumes roughly 3.2 times more than Hermes. However, asynchronous components like C-elements are responsible for only 22% of this consumption. The remaining is due to ordinary logic gates.

A more realistic comparison of Hermes and BaBaRouter power appears in Fig. 18. In this scenario, routers are active 20% of the time. The obtained results display solid savings for the BaBaRouter. Internal power consumption is reduced by almost 90%. This excessive consumption in the Hermes router is due to the clock, taking into account clock tree buffers, inverters and sink registers. As for the switching power, the savings are reduced, but still substantial, over 50%. Leakage power, in turn is naturally bigger for the asynchronous design. This is due to the bigger area required by BaBaRouter. Moreover, leakage power represents over 30% of the total power consumption of this router. In Hermes, this proportion is of only 3%. In this way, further optimizations in asynchronous design can lead to area reductions and, consequently, leakage and total power reductions.

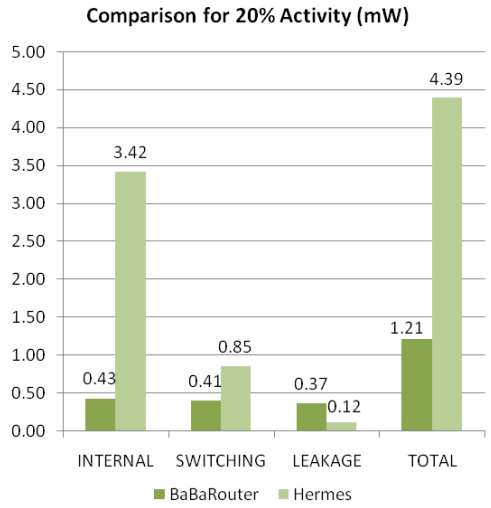


Fig. 18. Power results and comparison for BaBaRouter and Hermes, internal, switching, leakage and total power for a 20% activation scenario.

The power consumption distribution for BaBaRouter modules is presented in Fig. 19. In the chart, inputs are assumed as the combination of the input FIFO and the IN CTRL. The block that consumes more power is the crossbar (40%), while each port consumes around 12% and the switch control less than 0.5%. In this way, the most critical part of the router to be optimized is the Crossbar and the ports architecture.

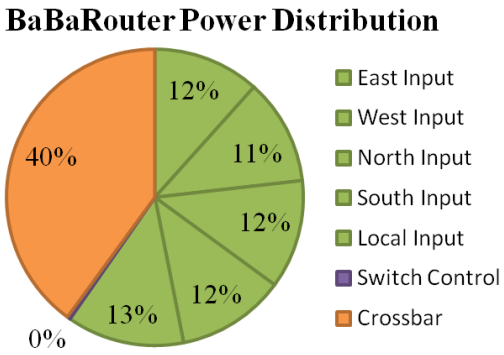


Fig. 19. Power distribution in BaBaRouter.

As for the area distribution of BaBaRouter, as Fig. 20 shows, each input port requires 10% of the total area, the switch control 6% and the Crossbar 44%. This justifies the excessive power consumption of the latter. On the other hand, albeit the switch control requires 6% of the total area of the router, it corresponds to less than 1% of the total power consumption. This is due to the fact that only the first flit of each packet activates this block. The remaining flits will not propagate through it. In this way, it is idle most of the time.

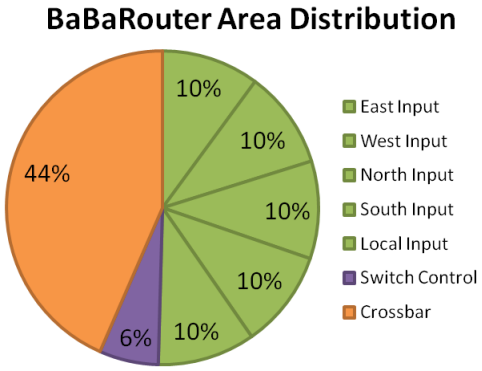


Fig. 20. Area distribution in BaBaRouter.

Fig. 21 shows the power distribution in Hermes for the three evaluated scenarios. As the charts show, the clock tree, required by synchronous approaches, consumes up to 50% of the total power of the circuit. Moreover, sequential cells are also responsible for a big part of the power consumption. This is due to the fact that registers are always being activated, even when and where they are not required. In a fully asynchronous circuit such as BaBaRouter, registers are activated only when and where required. That is the reason for BaBaRouter to consume less power.

Due to their nature, asynchronous circuits tolerate wider variations in power supply voltage [37]. Lower voltages reduce the operating speed and, consequently, power consumption. Therefore, for applications that require low throughput, lower voltages could be applied to BaBaRouter in a straightforward manner, resulting in lower power consumption. For synchronous routers, this technique is not easily ap-

plied, due to timing constraints imposed by the use of a clock signal to control the circuit and its dependency on the supply voltage.

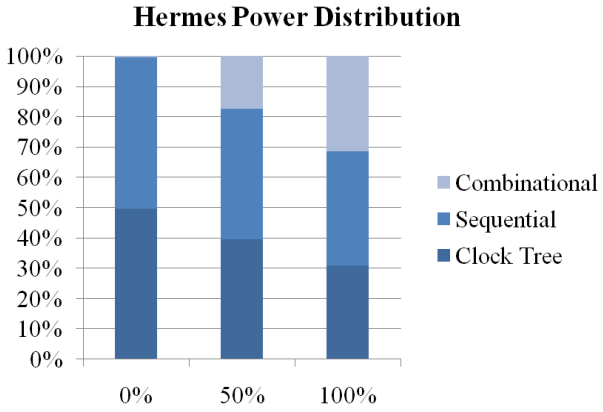


Fig. 21. Power distribution in Hermes router.

Finally, BaBaRouter supports GALS or fully asynchronous SoCs, while Hermes supports only fully synchronous systems. If synchronizer interfaces are added to Hermes to support non synchronous SoCs, power, operating frequency and area figures could be further compromised.

8. Conclusions

This work presented the use of an automated flow to implement an asynchronous QDI DR NoC router. The router was described in a language specifically designed for asynchronous circuits, Balsa. As far as the authors could verify, this is the first asynchronous NoC router to be implemented using a high level design approach.

Results show that by using Balsa to implement asynchronous intrachip networks, substantial power savings can be obtained. When compared to the Hermes synchronous NoC router, BaBaRouter required less power for all simulated scenarios. In realistic scenarios, savings of over 70% in total power consumption were demonstrated. In other words, the designed router may present a solution for low power requirements in NoC-based embedded SoCs. Also, the implemented router offers in average higher throughput and lower power consumption than Hermes-AA, a previous asynchronous NoC router designed through a structural VHDL approach. Not only better performance figures were obtained, but a lower complexity method to implement asynchronous routers was validated, which guarantees opportunities for design space exploration.

Although it does incur an area overhead, the BaBaRouter is naturally tolerant to process and operational variations when compared to a synchronous NoC router. This is due to natural characteristics of asynchronous circuits and is advantageous in current technologies, where small fabrication variations may compromise a whole chip. In addition, BaBaRouter presents a more straightforward possibility of implementing techniques to reduce power consumption, e.g. by varying power supply voltage. Further studies are under way to evaluate this and other optimization possibilities.

Finally, if Hermes was to provide all the listed advantages presented by BaBaRouter, area, operating frequency and power figures would be compromised. In this way, results are in agreement with the ITRS predictions and point that advances in EDA tools for supporting the asynchronous paradigm can improve the quality of implementations similar to BaBaRouter, which would be helpful for coping with the emerging CMOS technology problems. Future work includes the validation in silicon of BaBaRouter. Moreover, different routing algorithms can easily be added to the router. Also other asynchronous templates can be used to re-implement the router, including bundled-data and 1-of-4 encoding, combined with 2-phase or 4-phase protocols. In addition, the authors envisage the construction of an automatic generator for BaBaRouter, along with a high level simulation and evaluation environment.

9. References

[1] M. Mitic, M. Stojcev, A Survey of Three System-on-Chip Buses: AMBA, CoreConnect and Wishbone, in: XLI International Scientific Conference on Information, Communication and Energy Systems and Technologies (ICEST'06), 2006. 4p. Available at http://es.elfak.ni.ac.rs/Papers/ICEST '06.pdf

[2] A. Hemani, A. Janstch, S. Kumar, A. Postula, J. Berg, M. Millberg, D. Lindqvist, Network on a chip: An architecture for billion transistor era, in 18th NORCHIP Conference, 2000.

[3] P. Guerrier, A. Greiner, A Generic Architecture for On-chip Packet-switched Interconnections, in Design and Test in Europe (DATE'00), 2000.

[4] M. Kistler, M. Perrone, F. Petrini, Cell Multiprocessor Communication Network: Built for Speed, IEEE Micro 26(3) (May-June 2006) 10-23.

[5] D. Chapiro, Globally Asynchronous Locally Synchronous Systems, PhD Thesis, Stanford University, October 1984, 134p.

[6] International Technology Roadmap for Semiconductors, 2011 Edition - Design Chapter, available at http://www.itrs.net/Links/2011ITRS/2011Chapters/2011Design.pdf, 2011, 52p.

[7] R. Ginosar, Metastability and Synchronizers, Design & Test of Computers 28(5) (Sep-Oct 2011) 23-35.

[8] J. Pontes, M. Moreira, R. Soares, N. Calazans, Hermes-GLP: A GALS Network on Chip Router with Power Control Techniques, in IEEE Computer Society Annual Symposium on VLSI Design (ISVLSI'08), 2008, pp. 347-352.

[9] P. Beerel, R. Ozdag M. Ferretti, A Designer's Guide to Asynchronous VLSI, Cambridge University Press, Cambridge, 2010, 337 p.

[10] A. Kondratyev, K. Lwin, Design of asynchronous circuits by synchronous CAD tools, IEEE Design & Test of Computers 19(4) (Jul-Aug 2002) 107-117.

[11] J. Cortadella, A. Kondratyev, L. Lavagno, C. Sotiriou Desynchronization: Synthesis of Asynchronous Circuits from Synchronous Specifications, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems 25(10), (Oct 2006) 1904-1921.

[12] Y. Thonnart, E. Beigné, P. Vivet, A Pseudo-Synchronous Implementation Flow for WCHB QDI Asynchronous Cir-

cuits, in 18th IEEE International Symposium on Asynchronous Circuits and Systems (ASYNC'12), 2012, pp. 73-80.

[13] M. Renaudin, A. Fonkoua. Tiempo Asynchronous Circuits System Verilog Modeling Language, in: 18th IEEE International Symposium on Asynchronous Circuits and Systems (ASYNC'12), 2012, pp. 105-112.

[14] F. Darve, A. Sheibanyrad, P. Vivet, F. Petrot, Physical Implementation of an Asynchronous 3D-NoC Router Using Serial Vertical Links, in 2011 IEEE Computer Society Annual Symposium on VLSI (ISVLSI'11), 2011, pp. 25-30.

[15] F. Clermidy, C. Bernard, R. Lemaire, J. Martin, I. Miro-Panades, Y. Thonnart, P. Vivet, N. Wehn, MAGALI: A Network-on-Chip based multi-core system-on-chip for MIMO 4G SDR, in 2010 IEEE International Conference on IC Design and Technology (ICICDT'10), 2010, pp. 74-77.

[16] E. Beigné, F. Clermidy, P. Vivet, A. Clouard, M. Renaudin, An asynchronous NOC architecture providing low latency service and its multi-level design framework, in 11th IEEE International Symposium on Asynchronous Circuits and Systems (ASYNC'05), 2005, pp. 54-63.

[17] F. Moraes, N. Calazans, A. Mello, L. Möller, L. Ost, HERMES: an Infrastructure for Low Area Overhead Packet-switching Networks on Chip. Integration, the VLSI Journal 38(1) (October 2004) 69-93.

[18] M. Moreira, B. Oliveira, J. Pontes, N. Calazans, A 65nm Standard Cell Set and Flow Dedicated to Automated Asynchronous Circuits Design, in: 24th IEEE International SOC Conference (SOCC'11), 2011, pp. 99-104.

[19] J. Sparsø, S. Furber, Principles of Asynchronous Circuit Design – A Systems Perspective, Kluwer Academic Publishers, Boston, 2001, 354p.

[20] M. Agyekum, S. Nowick, An error-correcting unordered code and hardware support for robust asynchronous global communication, in: Design, Automation and Test in Europe (DATE'10), 2010, pp. 765-770.

[21] T. Verhoeff, Delay-insensitive codes- an overview, Distributed Computing, 3(1), 1988, pp. 1-8.

[22] A. J. Martin, Formal program transformations for VLSI circuit synthesis, Formal Development of Programs and Proofs, E. W. Dijkstra, Editor, Addison-Wesley, 1989, pp. 59-80.

[23] E. Yahya, M. Renaudin, QDI latches characteristics and asynchronous linear-pipeline performance analysis, TIMA Technical Report TR 06/06-03, 2006, 11 p.

[24] D. Edwards, A. Bardsley, L. Janin, L. Plana, W. Toms, Balsa: A Tutorial Guide, Version 3.5, APT Group, University of Manchester, 2006, 157p.

[25] M. Moreira, B. Oliveira, J. Pontes, F. Moraes, N. Calazans, Adapting a C-Element Design Flow for Low Power, in: IEEE International Conference on Electronics, Circuits and Systems (ICECS'11), 2011, pp. 45-48.

[26] R. Dobkin, R. Ginosar, A. Kolodny, QNoC asynchronous router, Integration the VLSI Journal 42(2) (February 2009) 103-115.

[27] T. Bjerregaard, J. Sparsø, A router architecture for connection-oriented service guarantees in the MANGO clockless network-on-chip, in: Design, Automation and Test in Europe (DATE'05), 2005, pp. 1226-1231.

[28] Y. Thonnart, E. Beigné, P. Vivet, Design and Implementation of a GALS Adapter for ANoC Based Architectures, in: 14th IEEE International Symposium on Asynchronous Circuits and Systems (ASYNC'09), 2009, pp. 13-22.

[29] A. Sheibanyrad, I. Miro-Panades, A. Greiner, Multisynchronous and Fully Asynchronous NoCs for GALS Architectures, IEEE Design and Test of Computers 25(6) (November-December 2008) 572 - 580.

[30] J. Pontes, M. Moreira, F. Moraes, N. Calazans, Hermes-A - An Asynchronous NoC Router with Distributed Routing, in: International Workshop on Power and Timing Modeling, Optimization and Simulation (PATMOS'10), LNCS vol. 6448, 2010. pp. 150-159.

[31] J. Pontes, M. Moreira, F. Moraes, N. Calazans, Hermes-AA: A 65nm asynchronous NoC router with adaptive routing, in: IEEE International SOC Conference (SOCC'10), 2010, pp. 493-498.

[32] Y. Thonnart, P. Vivet, F. Clermidy, A Fully Asynchronous Low-Power Framework for GALS NoC Integration, Design, Automation, and Test Europe (DATE'10), pp. 33-38, 2010.

[33] L. Ost, A. Mello, J. Palma, F. Moraes, N. Calazans, MAIA – a framework for networks on chip generation and verification, in: Asia and South Pacific Design Automation Conference (ASPDAC'05), January 2005, pp. 49-52.

[34] M. Moreira, B. Oliveira, J. Pontes, F. Moraes, N. Calazans, Impact of C-Elements in Asynchronous Circuits, in: International Symposium on Quality Electronic Design (ISQED'12), 2012, pp. 438-444.

[35] L. Tedesco, A. Mello, N. Calazans, F. Moraes, Traffic Generation and Performance Evaluation for Mesh-based NoCs, in: 18th Symposium on Integrated Circuits and Systems Design (SBCCI'05), 2005, pp. 184-189.

[36] N. Ekekwe, Power dissipation and interconnected noise challenges in nanometer CMOS technologies, IEEE Potentials 29(3) (May 2010) 26-31.

[37] L. Cristófoli, A. Henglez, J. Benfica, L. Bolzani, F. Vargas, A. Atienza, F. Silva, On the Comparison of Synchronous Versus Asynchronous Circuits under the Scope of Conducted Power-Supply Noise, in: Asia Pacific Symposium on Electromagnetic Compatibility (APEMC'10), 2010, pp. 1047-1050.