Exploring Asynchronous End-to-End Communication Through a Synchronous NoC

Iaçanã I. Weber
UFSM – Santa Maria, Brazil
iacanaw@gmail.com

Leonardo L. de Oliveira
UFSM – Santa Maria, Brazil
leonardo@mail.ufsm.br

Fernando Gehm Moraes
PUCRS – Porto Alegre, Brazil
fernando.moraes@pucrs.br

Everton A. Carara
UFSM – Santa Maria, Brazil
carara@ufsm.br

***Abstract*— SoC design will require asynchronous techniques as the large parameter variations across the chip impose challenges to control delays in clock networks and other global signals efficiently. Today's state-of-the-art SoCs contain an ever-growing number of IP cores, each of which is immersed in its independent clock domain. When the IP core population increases and the minimum transistor size shrinks, it is challenging to keep all the different clock domains synchronized across the die while maintaining the considerable data bandwidth between them. One of the most promising solutions to this interconnect issue is the Globally Asynchronous, Locally Synchronous approach. Our goal is to provide an interconnection architecture which supports asynchronous communication between IPs working on its frequency. Instead of designing a fully asynchronous Network-on-Chip (NoC), our idea is to improve an existing synchronous NoC to support end-to-end asynchronous communication.**

Keywords*— *SoC, NoC, GALS, Asynchronous.

I. INTRODUCTION

The advances in semiconductor technology enabled the integration of an increasing number of IP (Intellectual Property) blocks in a single System-on-Chip (SoC) [1]. Networks-on-Chip (NoCs) have been used to connect those cores, performing the role of communication infrastructure. Mesh is the most common NoC topology, where the routers are placed at cross points of shared links to perform multiplexing of different data flows on the links.

Ideally, it would be desirable to have a completely synchronous SoC clocked by the same clock signal. Unfortunately, one of the main problems in today's ASICs is to distribute a skew-free synchronous clock efficiently over the whole chip [2]. The usage of a fully asynchronous system implementation would eliminate the clock distribution and the problems associated with it (e.g., dynamic power, clock tree design) [3][4]. However, such approach is not attractive yet since the existing design tools, and IPs libraries are focused on the synchronous paradigm.

As a solution for the global clock problem, we could approach the design paradigm, named globally asynchronous, locally synchronous (GALS) [5][6], where each IP has its own clock. This paradigm is based on the fact that while it may not be practicable to distribute a global clock signal with acceptable skew, it is still achievable to have a clock within a limited area, so-called clock domain [7]. When the GALS paradigm is used, the communication between two clock domains requires an appropriate synchronizer for Clock-Domain Crossing (CDC) to avoid metastability. The simplest method required to perform synchronization is a simple two-flip-flop synchronizer [8]. Despite a simple and cheap approach, this synchronizer introduces two-cycle latency to the path.

Employing the GALS paradigm in NoC designs can be accomplished, for example, by using routers in the same clock domain of their respective IPs (mesochronous) [6]. The problem with this approach is the need of successive synchronizations starting at the source router all the way up to the target one. Such approach increases the communication latency between IPs due to synchronization issues and possible slow-clock routers in the path.

Another approach is to treat the whole NoC as a communication IP operating in its own clock domain. This way taking advantage of the well-established synchronous designing tools. Although it could work fine for small dimensions NoCs, in larger ones the clock domain would have to operate at low frequency due to the difficulty to distribute the clock over the whole network efficiently. NoCs operating at a low frequency with regard to the IPs may become a system bottleneck.

Our proposal is a fully synchronous NoC capable of connecting two different IPs directly (as wires), each one in its own clock domain, providing an asynchronous communication interface based on a handshake protocol. Once the communication path is established, flits are allowed to bypass all router buffers belonging to the path between the source and target IPs. We achieve such functionality by exploiting the capacity to bypass the router input buffers, turning a packet switching router in a hybrid packet/circuit switching router.

In this work, we make the following contributions:

- First, we present a synchronous NoC, called Arke as a reference to the messenger of the Titans during the Titanomachy. It is a parametrizable 2D/3D mesh NoC, described in synthesizable VHDL, supporting traditional packet switching along with buffer bypass based circuit switching;
- Second, we present our network interface (NI) employing an asynchronous circular FIFO to maximize the end-to-end throughput;
- Lastly, we present results comparing synchronous and

978-1-5386-7431-4/18/$31.00 ©2018 IEEE

asynchronous communication performance. The synchronous communication uses bisynchronous FIFO NIs, commonly used for GALS SoCs, and the asynchronous one uses the NIs employing asynchronous circular FIFOs.

The paper is organized as follows: we first describe the related works in Section II. Then we present the Arke NoC along with the modifications to support the asynchronous communication in Section III. NI with its asynchronous circular FIFO is introduced in Section IV. Section V brings the comparing results and Section VI conclusions and future works.

II. Related Works

The buffer bypass approach has been explored aiming packet latency reduction, which can be drastically achieved avoiding the temporary flit storage in routers [9][10][11]. It can provide to the designers some advantages of asynchronous design style in a design predominantly synchronous, bringing together the best of both worlds. NoC based GALS systems are well suited to take advantage of buffer bypass. The following works are representative researches employing such idea.

A. Asynchronous Bypass Channels [9]

In this work, the Authors propose a new router microarchitecture for multi-synchronous NoCs which avoids synchronization overhead at the intermediate nodes via an asynchronous bypass channel (ABC). They also propose a novel 2D grid-based topology called double chain and a routing algorithm that leverage the advantages of the bypass channel. The goal is that a packet should not be required to latch at the intermediate routers. The ABC design offers an asynchronous bypass path which bypasses the router buffer as well as synchronization at the intermediate nodes. Thus the packet flits traverse intermediate nodes without getting latched, reaching their destination with a latency approaching the wire delay of the connecting links. Experiments comparing the ABC approach against conventional synchronous NoCs with similar resources demonstrate an improvement up to 26% of performance at low loads and increases in saturation throughput by up to 50%. There are some limitations in the project, for example, when a packet needs to make a direction change (e.g., east to north), it must be latched, increasing the overall latency. Also, since there is no connection establishment, the depth of the router buffers must be overcommitted by at least four more than the maximum packet length to ensure no flit get dropped in the transmission. The ABC router design is source-synchronous, hence each port transmit a local clock bit (source clock) along with data bits.

B. SMART [10][11]

Aiming to allow propagation of signals across multiple-mm within a cycle, this work proposes a novel low-swing link circuit using clockless repeaters. They replace conventional links in the network by SMART (Single-cycle Multi-hop Asynchronous Repeated Traversal) links. They also present a tool flow to perform reconfiguration of network routers during runtime, enabling different applications to run on virtually tailored topologies. The authors implemented a 4x4 SMART NoC and evaluated it with a suit of SoC applications. Compared to a conventional state-of-the-art NoC, the SMART NoC reduces network latency by 60.1% on average due to the bypassing of the complete router pipeline. It is required prior knowledge of the application communication to configure the paths which will use the SMART links. The programmer must perform the pre-configuration during the initial cycles of the application, through memory mapped registers. The network needs to be emptied while setting the registers. Besides, if packet collisions occur, it is necessary to store packets in the buffers inside the router temporarily. The main drawback of SMART NoC is scalability since data must be transmitted in one clock cycle. The bypass path is limited to few routers (11 hops in 45nm technology). In [12] the authors propose the SCEPTER NoC (Single-Cycle Express Path Traversal for Efficient Routing) as a bufferless version of SMART.

C. ReNoC [13]

The Reconfigurable NoC (ReNoC) has as the main goal to enable reconfiguration of the network topology. It is composed by conventional NoC routers wrapped by topology switches. Such topology switches are used to connect routers into a given logical topology and thereby allowing different application-specific logical topologies to be configured on top of the same physical architecture. According to the implemented logical topology, some routers are bypassed by topology switches. The authors compare the ReNoC power consumption against traditional mesh NoCs. ReNoC achieves up to 56% of power reduction when implementing application-specific logical topologies. The reconfiguration is static, and the logical topology must be configured such that the delay of the longest logical link does not exceed the clock period.

D. Bypass Router [14]

The SMART NoC routers employ dedicated broadcast wires to transmit SMART-hop setup requests (SSRs), incurring large wire and energy overheads. They are used to set up bypasses paths. The Bypass Router presents an alternative method to set up the bypass paths to reduce the energy consumption and area overhead when compared to SMART routers. Despite the obtained gains in energy and area, it suffers the same scalability drawback concerning the limited number of bypassed routers.

Our proposal aims to support NoC-based GALS systems through asynchronous channels. Since the Arke NoC establishes connections at runtime, no prior knowledge of the application behavior is required to set the asynchronous paths. In addition, the routers only need to temporarily store the header flit, since the payload flits are transmitted through the asynchronous channels that bypass the buffers. Once the asynchronous path is established, the NoC clock domain no longer interferes in the end-to-end communication throughput. The path becomes a dedicated link between the source and the target IPs. Due to the asynchronous handshake protocol employed in the IP-to-IP communication, the NoC clock period does not limit the number of hops in a path, which makes the approach scalable to systems with dozens of IPs. To the best of our knowledge, this is the first work exploring traditional

asynchronous handshake through a fully synchronous NoC, closing the gap between the two design styles. We decouple computation and communication by applying synchronous design for data processing and asynchronous design for end-to-end data transfer. Since it is a parameterizable NoC described in technology-independent synthesizable VHDL, it has a great portability potential between projects. Finally, it can take advantage of the well-established synthesis tools for synchronous projects even employing an asynchronous handshake protocol.

III. ARKE

Arke is a mesh NoC, derived from Hermes [15]. As key differential points, Arke supports 3D topology, no bounded packet size, lower latency and smaller area overhead. Also, and the most important feature, is its ability to bypass router input buffers to establish asynchronous IP-to-IP connections.

This NoC is built from multiple instantiations of a basic block, the router, shown in Fig. 1. The router can have up to seven bi-directional ports in 3D topologies: East, South, West, North, Up, Down and Local. Each port has an input buffer for temporary flits storage during packet switching communications. The Local port is connected to the local IP, while the other ports are connected to neighboring routers. In addition to the buffers, each router has a centralized control logic (*Switch Control*) that implements the routing algorithm and arbitrates new internal connections between the input and output ports. To establish those internal connections, the router has a *Crossbar* controlled by the *Switch Control*.

Arke adopts wormhole packet switching. Each packet has a header flit containing the communication type and the destination address. The payload follows the header and the last of those flits is signaled by the control signal *eop* (end-of-packet). After transmitting the last flit, the routers release the internal crossbar connection.

It supports two types of communication (synchronous and asynchronous). The packet header informs which one will be used. In synchronous communications, wormhole packet switching is employed, and the flow control is implemented using the stall-and-go protocol. This protocol consists of a buffer at the receiver and a return line (*stall_go*) to the transmitter to inform if there is available space. The transmitter interprets such information as a semaphore; if it is "go" (*stall_go* = '1') data can be sent, if it is "stall" (*stall_go* = '0') data must wait until there is space in the receiver buffer. Data valid present in *data_out* is signalized by the transmitter through *tx* (*tx* = '1'). The East port interface can be seen in Fig. 1.

The second communication type Arke supports, and the focus of this work is the asynchronous one. This communication type employs a circuit switching connection which has three phases: (*i*) establishment, (*ii*) transmission and (*iii*) closure.

The establishment of the asynchronous connection occurs as follows: the source IP injects the packet header into the local router, which temporarily stores it waiting for the routing process and crossbar configuration. Once the crossbar is configured, the data and control signals multiplexers are set to bypass state (dotted signals in the Fig. 1). As a consequence, the source IP is now connected directly to the next router in the path. This process repeats until the source, and the target IPs get directly connected. As soon as the connection is established, the source and target IPs are connected without any NoC clock domain influence, and the transmission phase begins.

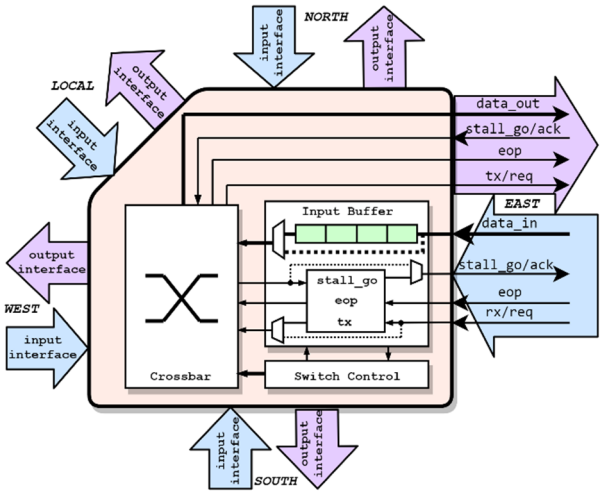

Fig. 1 – Arke 2D Router. Dotted signals are the bypass paths, used for asynchronous communications.

During transmission phase, the IPs communicate via 2-phase (Non-Return-to-Zero) bundle-data protocol. The typical *request* and *acknowledgment* signals involved in such asynchronous protocol are performed by *tx* and *stall_go* respectively (Fig. 1). Thus, to perform a transmission, the source IP first makes the flit available (*data_in*), followed by a transition in the request signal (*req*). The target IP answers transitioning the acknowledgment signal (*ack*). After receiving the acknowledgement, the source IP is free to send a new flit, repeating the protocol. It is important to note that by changing the communication type from synchronous to asynchronous, the control signals change their functions: *tx* becomes request (*req*) and *stall_go* assumes the acknowledge (*ack*) function, so that, it was not necessary to make any change in the router interface in order to support the asynchronous communication.

The connection closure phase starts with the last packet flit transmission. As can be seen in Fig. 1, the *eop* signal is not bypassed, which means that it must be synchronized in the NoC clock domain and transmitted synchronously. Upon receiving the *eop* signal the router starts the process of disconnecting the established path. The connection closure is made in the same way as the establishment, storing the last packet flit temporally at each router in the path until it reaches the target IP. As the flit is forwarded to the next router, the internal crossbar connection is released.

We faced hazard problems while developing the Arke's asynchronous communication support. When performing post logic synthesis simulations with delay, we detected static hazards on the *req* and *ack* signals. The sources of hazards were the bypass multiplexers (Fig. 1) and the crossbar output multiplexers. In asynchronous design, hazards are harmful and must be avoided. To solve the problem, we designed hazard-

free multiplexers for the bypass multiplexers, as they have only two data inputs. In the crossbar multiplexers, we ensured the change of only one bit at time when selecting a new data input (almost one-hot encoding).

IV. Network Interface (NI)

In GALS systems, the IPs can operate in their own clock domain. Consequently, the signals synchronization between different clock domains requires the addition of synchronizers between NoC and IPs [8]. Bisynchronous FIFOs are commonly used for this purpose [16] and placed at the network interface. The main drawback of such approach is the bottleneck created when the communicating IPs operate in higher frequencies than the NoC. Since the packets are forwarded in the NoC's clock domain, it becomes the communication bottleneck, limiting the end-to-end throughput.

The NI we propose employs an asynchronous circular FIFO instead of bisynchronous FIFOs, allowing IPs to communicate without the need to transfer the communication to NoC's clock domain. In this way, the asynchronous communication throughput is bounded by the slower communicating IP. The NI operation extends the Arke asynchronous communication protocol, described in the previous section, to the IPs. Our asynchronous circular FIFO is based on MOUSETRAP pipeline stages [17] to latch data, combined with Johnson counters as writing and reading pointers. The write control bits are used as MOUSETRAP request latches. Fig. 2 shows the modules inside the NI related to the asynchronous data transmission.

Different from the bisynchronous FIFO approach, our proposal requires buffering only at the receiver's end. Data and its corresponding request are latched while waiting for the target IP reading. The dotted area in the asynchronous circular FIFO highlights a MOUSETRAP stage, consisting only the data latch since the request latch was incorporated in the write control. The latch is a standard level-sensitive D-type. Johnson counters are used as read and write pointers (read and write controls on Fig. 2). The write counter is triggered by transitions on the *req* signal generated remotely by the transmitter whilst transitions on the *ack* signal, generated locally by the receiver, triggers the read counter. The XNOR operation between the read and write pointers are used to indicate the storage status (*slots_free* signal). Free slots are indicated by '1' (transparent latch) and occupied ones indicated by '0' (opaque latch). Based on the read control, the multiplexer selects which slot is the next one to be read by target IP.

The transmitter writes directly into the receiver's FIFO transitioning the *req* signal. To know the receiver's FIFO occupation and avoid overwrites, it has the same two Johnson counters. As occurs at the receiver's end, the read counter is triggered remotely by the receiver through the *ack* signal, and the write counter is triggered locally through the *req* signal.

The FIFO depth is seven flits. This depth was chosen so that in a scenario with two IPs working at the same clock frequency, but in distinct clock domains, could communicate at a rate of one flit per cycle. The FIFO's main function is actually to maximize the communication throughput keeping an uninterrupted flow of flits. Since the FIFO's control signals (*req*, *ack* and *slots_free*) are asynchronous, they must be synchronized at the IPs clock domain. Such synchronization is based on the traditional pair of flip-flops. To cover the entire round-trip latency caused by the synchronization, there must be seven positions in the FIFO, which consists in: one source IP writing cycle, two cycles for request synchronization at target IP (*slots_free*), one target IP cycle for reading and another two cycles for acknowledgment synchronization at source IP (*slots_free*). Thus, considering that both communicating IPs are on the same frequency, when source IP is writing in the seventh FIFO position, the information that the first one was read will be arriving and in the next cycle and it will be able to write there again. That is the reason there is an individual status signal for each FIFO slot (*slots_free*). Note that, the source IP does not need to wait for the acknowledgment from each flit before request the next one. The read and write controls keep the information about the target's FIFO occupation.

The presented asynchronous infrastructure covers only the

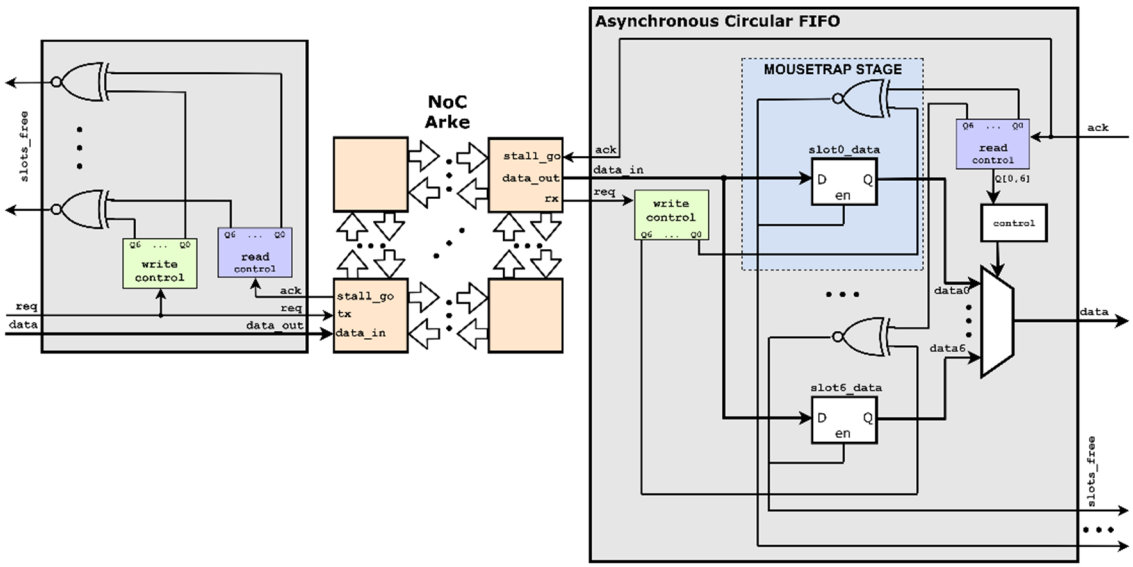

Fig. 2 – Asynchronous circular FIFO employed on Network Interface.

data transmission phase. The NI performs the connection establishment and closure phases synchronously operating in the same NoC clock domain.

V. Experiment Results

This section presents evaluations performed using Arke's synchronous and asynchronous communication. The goal is to demonstrate the potential of Arke asynchronous bypass channels combined with the asynchronous circular FIFO in the NI for GALS systems.

All simulations were carried out with the resulting netlist of the Arke logic synthesis. We used Cadence tools with IBM 180nm standard cells library for the logic synthesis. The obtained mapped file with delays for the cells (SDF - standard delay file) provides important delays related to the asynchronous communication path.

The first evaluation (Fig. 3) compares the performance of the two communication types (synchronous and asynchronous) in a zero load NoC. The scenario comprises a source IP sending one packet to a target IP. The goal is to observe how the packet latency increases as a function of the packet size (from 16 fits to 512 flits) in the two communication types varying only the IPs frequency. The NoC frequency is fixed and used as reference to compute packet latency. The IPs frequency ranges from one to seven times the NoC frequency. It is important to mention that each IP injects one flit per clock cycle, in this way increasing the injection rate when the frequency is increased.

Each line in Fig. 3 represents the packet latency considering the IPs pair operating in a frequency relative to the NoC frequency. For example, the line *IPs 7x Async* presents the packet latency variation considering the IPs pair operating in a frequency corresponding to seven times the NoC frequency and using the asynchronous protocol. The synchronous communication employs NIs with bisynchronous FIFOs while the asynchronous one employs NIs with asynchronous circular FIFOs. As expected, the packet latency on synchronous communication (dotted line) does not change as the IPs pair frequency increases due to the bottleneck generated by NoC. That is why there is only one line for the synchronous communication. Regarding the asynchronous communication (continuous lines), when the IPs pair operates at the same frequency as the NoC, the packet latency is always larger than the synchronous communication. As the graphic shows, there is a constant difference in the latency concerning the synchronous communication, due to the connection establishment and closure phases. However, as the IPs pair frequency increases, the latency overhead related to the connection management is redeemed and the asynchronous communication overcomes the synchronous one. The asynchronous communication presents higher packet latency only for small packet sizes. To clarify the intersection point between the asynchronous and synchronous communication, let's analyze the graphic showed in Fig. 4. This graphic covers only small packets (maximum of 128 flits) and IPs pair frequency with a maximum variation of two times the NoC frequency. In this graphic one can see the minimum packet size where the asynchronous communication starts to take advantage against the synchronous one for different IPs pair frequencies.

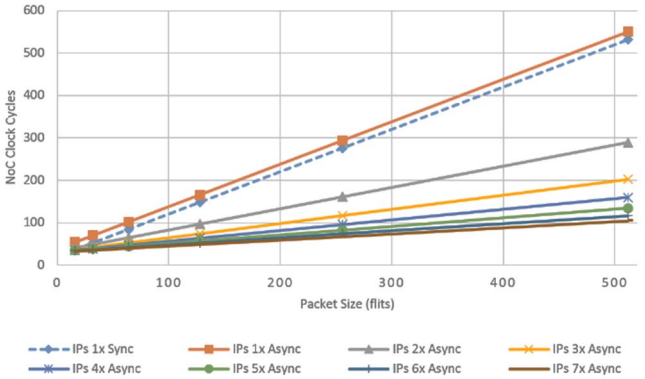

Fig. 3 – Asynchronous vs synchronous communication, covering packets from 16 flit to 512 flits and IPs pair frequencies ranging from the NoC frequency up to seven times it.

The second evaluation compares the performance of both communication types in a scenario comprising several communicating IPs. It consists in a 4x4 NoC connecting 16 IPs. Each IP injects 200 packets, totalizing 3200 packets. The packet sizes range from 18 flits to 512 flits and targets random destinations. The IPs frequency ranges from one to five times the NoC frequency. In this scenario it is computed the total time in NoC clock cycles to transmit all 3200 packets for different IPs frequencies. The dotted line in Fig. 5 presents the results for the synchronous communication whilst the continuous line presents the results for the asynchronous communication. As expected, the total time to finish the synchronous communication transmissions does not change when increasing the IPs frequency. In the other hand, the asynchronous transmissions take more time to finish only when the IPs and the NoC are in the same frequency due to the latency overhead of the connection establishment and closure. It is important to mention that for each packet asynchronously transmitted, there is connection establishment and closure. The performance may be far increased using burst transmissions, keeping the connection active for a set of packets.

Besides the performance evaluations, we also obtained area results for the Router and for the Network Interface. They are

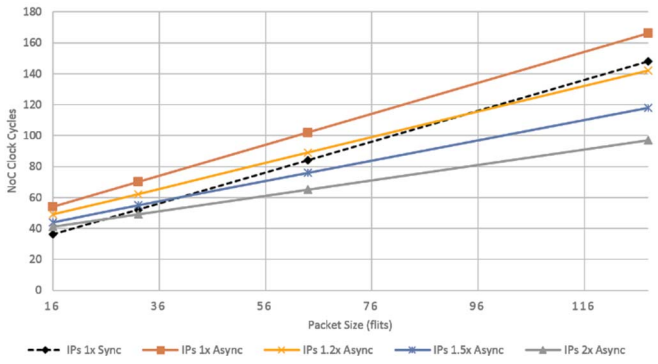

Fig. 4 - Asynchronous vs synchronous communication, covering small packets and IPs pair frequencies ranging from the NoC frequency up to two times it.

presented in Table 1.

Table 1 - Area results (IBM 180nm standard cells library).

Entity	Sync. Area (μm^2)	Async. Area (μm^2)
Router	86,025	115,051
Network Interface	47,860	27,448

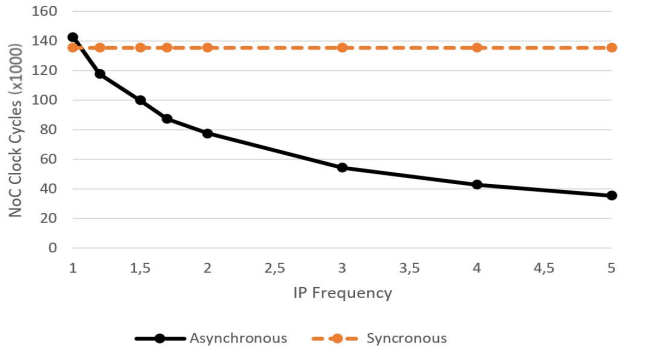

Fig. 5 - Asynchronous vs synchronous communication. Total time to transmit 3200 packets considering different IPs frequencies.

The synchronous router refers to the Arke Router without the support of asynchronous communication. The synchronous NI employs bisynchronous FIFOs while the asynchronous one employs asynchronous circular FIFO. The router area overhead reaches 33.74% due to the addition of circuitry to avoid hazards and support to asynchronous bypass channels. To compare the router size to a processor IP, we synthesized an ARM implementation (Storm Core [18]) comprising only data and control path using IBM 180nm standard cells library. The reported area was 610,173 µm^2. The new Arke router represents 18% of the ARM area whilst the original one represents 14%. Such difference reduces even more depending on the considered processor cache size.

On the other hand, the NI employing asynchronous circular FIFO presented an area reduction of 42.65% when compared to the traditional bisynchronous NI. The bisynchronous NI requires two buffers: one to cross the clock domain from source IP to NoC and another one to cross the clock domain from NoC to target IP. The proposed NI requires only one at the target to maximize the communication throughput.

VI. Conclusions and Future Works

In this paper, we presented the synchronous NoC Arke supporting end-to-end asynchronous communication and an NI employing an asynchronous circular FIFO. The proposal is well suited to support GALS systems, providing high communication throughput to IPs operating at frequencies higher than the NoC. Despite the increased router area, the overhead is negligible when compared to a processor IP. Conversely, our NI employing the asynchronous circular FIFO presented a significant area reduction when compared to the traditional bisynchronous NI.

Future works include exploration of power and clock gating techniques, as we observed that once the asynchronous channel is established, the bypassed router buffers are idle and it could generate significant gains in terms of power/energy saving. Also, different from previous works, we aim to explore the capability of establish a communication between synchronous and asynchronous IPs.

ACKNOWLEDGEMENTS

Author Fernando Gehm Moraes is supported by FAPERGS (17/2551-196-1) and CNPq (302531/2016-5), Brazilian funding agencies.

VII. References

[1] Goossens, H.; Dielissen, J.; Radulescu, A. “AEthereal network on chip: concepts, architectures, and implementations”. IEEE Design & Test of Computers, vol.22, 2005, pp. 414 - 421.

[2] Tatas, K.; Siozios, K.; Soudris, D.; Jantsch, A. “Designing 2D and 3D NOC Archietectures”, Springer Science, 2014.

[3] Bertozzi, D.; Miorandi, G.; Tala, M; Nowick, S. M. “Cost-Effective and Flexible Asynchronous Interconnect Technology for GALS Networks-on-Chip”. In: NGCAS, 2017.

[4] Russell, P.; Döge, J.; Hoppe, C.; et al. “Implementation of an asynchronous bundled-data router for a GALS NoC in the context of a VSoC”. In: DDECS, 2017.

[5] Krstic, M.; Grass, E.; Fan, X. “Asynchronous and GALS Design -Overview and Perspectives”. In: NGCAS, 2017.

[6] Ax, J.; Kucza, N.; Vohrmann, M.; et al. “Comparing Synchronous, Mesochronous and Asynchronous NoCs for GALS Based MPSoCs”. In: MCSoC, 2017.

[7] Jantsch, A. “System modelling, models of computation and their applications”. Compendium for the KTH course 2B1429, January 2001.

[8] Ginosar, R. “Metastability and Synchronizers: A Tutorial”. IEEE Design & Test of Computers, vol.28, pp. 23-35, 2011.

[9] Jain, T. N. K.; Ramakrishna, M.; Gratz, P. V.; et al. “Asynchronous Bypass Channels for Multi-Synchronous NoCs: A Router Microarchitecture, Topology, and Routing Algorithm”. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, vol.30, pp.1663-1676, 2011.

[10] Krishna, T.; Chen, C. H. O.; Kwon, W. C.; Peh, L. S. “SMART: Single-Cycle Multihop Traversals Over a Shared Network on Chip”. IEEE Micro, vol.34, pp.43-56, 2014.

[11] Chen, X.; Jha, N. K. “Reducing Wire and Energy Overheads of the SMART NoC Using a Setup Request Network”. IEEE Transactions on Very Large Scale Integration Systems, vol. 24, pp. 3013-3026, 2016.

[12] Daya, B. K.; Peh, L. S.; Chandrakasan, A. P. “Towards High-Performance Bufferless NoCs with SCEPTER”. IEEE Computer Architecture Letters, vol.15, pp.62-65, 2016.

[13] Stensgaard, M. B.; Sparsø, J. “ReNoC: A Network-on-Chip Architecture with Reconfigurable Topology”, In: NoCS, 2008.

[14] Noghondar, A. F.; Reshadi, M. “A low-cost and latency bypass channel-based on-chip network”. The Journal of Supercomputing, vol.71, pp.3770-3786, 2015.

[15] Moraes, F.G.; Calazans, N.; Mello, A. “HERMES: an Infrastructure for Low Area Overhead Packet-switching”. Integration, the VLSI Journal, vol.38, pp.69-93, 2004.

[16] Panades, I. M.; Greiner, A. “Bi-Synchronous FIFO for Synchronous Circuit Communication Well Suited for Network-on-Chip in GALS Architectures”. In: NoCS, 2007.

[17] Singh, M.; Nowick, S. M. “MOUSETRAP: High-Speed Transition-Signaling Asynchronous Pipelines”. IEEE Transactions on Very Large Scale Integration (VLSI) Systems, vol.15, pp.684-698, 2007.

[18] Nolting, A. “STORM CORE (ARM7 compatible)”. [Online]. Available: https://opencores.org/project,storm_core. [Acessed in 05 Apr 2018].