BAT-Hermes: A Transition-Signaling Bundled-Data NoC Router

Matheus Gibiluka, Matheus Trevisan Moreira, Fernando Gehm Moraes, Ney Laert Vilar Calazans
PUCRS University, Computer Science Department, Porto Alegre, Brazil
{matheus.gibiluka, matheus.moreira}@acad.pucrs.br, {fernando.moraes, ney.calazans}@pucrs.br

***Abstract*—Networks on chip (NoCs) are efficient infrastructures to enable communication among the large number of IPs that compose modern systems on chip (SoCs). However, even if recent technologies allow the construction of such complex systems, they increase the cost and effort of designing a correct and efficient chip-wide clock delivery network for a fully synchronous system. Asynchronous NoCs are an attractive alternative, as they allow each IP to operate on an independent clock domain, eliminating the need for a global clock network and producing globally asynchronous locally synchronous (GALS) systems. This paper details the design of a low power NoC router that is asynchronous and employs a transition-signaling bundled-data design style. Results show reductions of 27% in energy consumption when compared to a similar synchronous router.**

I. Introduction

Today, SoCs with dozens of IP cores are viable and commonly found in commercial products. Interconnecting large amounts of cores requires the careful design of specific communication architectures that fulfill bandwidth, quality of service and power budgets demanded by SoCs. There is already several alternative NoC architectures that are useful in many fields of application and which fulfill widely varied requirements. These may use regular or irregular topologies NoCs, employ deterministic or adaptive routing, be synchronous, asynchronous or a mix of these, etc. As technology nodes advance, changes in the behavior of silicon devices impose serious restrictions for designers of complex systems. Contrary to what occur with processors, memories or I/O controllers, NoCs tend to have a chip-wide reach, and this exacerbates their sensitivity to parameters that tend to degrade as silicon technology advances. In this scenario, asynchronous NoC design appears as a trend to deal with such issues.

Asynchronous design can be based in different templates, each with different timing assumptions. Among these, the most employed are based either on quasi-delay-insensitive (QDI) [1] or bundled-data (BD) [2] families. QDI circuits are more robust against process, voltage and temperature (PVT) variations, but impose large power and area overheads. BD requires more careful design, but can be much more area and power efficient. A BD circuit is somewhat similar to a synchronous circuit. However, instead of using a global signal to control sequencing of events, synchronization occurs locally between each pair of communicating entities, using handshaking protocols. Such protocols can be either 2- or 4-phase and can employ different control circuits, as discussed in [3]. Among these, we highlight the Mousetrap template [3], which relies on conventional standard cells for the implementation of control blocks and employs 2-phase handshaking. These characteristics lead to reduced switching activity, lower cycle time and can be made compatible with synchronous flows and libraries.

This work introduces BAT-Hermes, a low-power NoC router capable of interconnecting IPs within distinct clock domains. When compared to a synchronous router of similar characteristics, BAT-Hermes shows average reductions of 27% in energy consumption. The use of single-rail data encoding potentially leads to low power overheads, when compared to other asynchronous approaches like QDI. Moreover, use of a transition-signaling protocol reduces latency, by eliminating return phases in the handshake, which improves circuit performance when compared to level-signaling approaches, at the cost of more complex controllers, which translates to some additional cost in area.

978-1-4799-8332-2/15/$31.00 ©2015 IEEE

II. Related Work

A survey of related work revealed only one NoC router that implements a transition-signaling handshake protocol: the design proposed by Ghiribaldi et al. in [4]. This router features wormhole switching, 32-bit flits, and dimension-ordered routing. The design was synthesized using a low power standard-Vt 40nm cell library, achieving an average throughput of 20.6Gbytes/s per router. In this design, MUTEXes and C-elements required to implement arbiters rely on ordinary standard cells. All remaining related work on bundled-data routers adopt level-signaling protocols. The reason behind this is that level-signaling significantly reduces design complexity, when compared to transition-signaling protocols [5].

The ASPIN NoC [6], proposed by Sheibanyrad et al. uses both BD and dual-rail data encoding, the latter being used only for long wires. Each router features wormhole packet switching, 32-bit flit size, and uses the XY routing algorithm. Synthesis targets a 90nm low voltage threshold technology, resulting on a maximum throughput of 4.423Gbytes/s per router. There is no information on physical design mapping in [6]. Another work, proposed by Bjerregaard and Sparso in [7], presents the MANGO NoC, which employs virtual channels to provide connection-oriented guaranteed services and connectionless best effort routing. The router implements wormhole packet switching, 32-bit flit size, and XY routing algorithm. The NoC was synthesized using 0.12μm CMOS standard cell technology, yielding an average throughput of 15.4GBytes/s per router under typical conditions. In [8], Dobkin et al. proposed asynchronous QNoC, supporting quality-of-service in four distinct service levels using virtual channels. The routers support XY routing and wormhole packet switching with 8-bit flit size. The NoC was synthesized using a 0.18μm CMOS standard cell library, resulting in an average throughput of 1.1GBytes/sec per router.

Related works employ different design styles and CMOS technologies, making it difficult to draw a fair comparison. Moreover, with respect to BAT-Hermes, only [4] features the same design style. In fact, BAT-Hermes is quite similar to the proposal in [4], with the main difference between the two being the packet control technique. The router proposed here employs a packet control logic based on a flit counter mechanism, to keep compatibility with its synchronous counterpart. The drawback is that it adds considerable control overhead with regard to the simpler end-of-packet (EOP) flag approach of [4]. A transition-signaling FIFO architecture is also proposed in [4], but the authors do not detail it, or even explain if the router actually uses it, which can justify the reported small area.

III. ARCHITECTURE

The starting point for BAT-Hermes was the architecture of the synchronous router called YeAH! [9], which employs a fully distributed control logic and a functionality similar to the Hermes router [10]. In this design, each router port has input and output interfaces (II and OI). The II of each router port is directly connected to all OIs of the remaining ports. Routing and arbitration tasks are distributed across the router and assigned to IIs and OIs, respectively. This is different from centralized routers like the original Hermes, where a single control block performs both tasks for all ports. The distributed approach results in a modular design that avoids bottlenecks in the control unit and improves performance on congested networks. More importantly, this is relevant in conjunction with asynchronous design, as it reduces the complexity of control blocks, alleviating overheads. Just like YeAH!, BAT-Hermes features wormhole packet switching, XY routing algorithm, and parameterizable flit size and buffer depth. The router was designed to work on a 2D mesh topology and can have up to 5 ports - one for communication with an IP and the remaining for communication with other routers.

The II of BAT-Hermes is responsible for buffering and routing tasks. As Figure 1 shows, the II consists in an input buffer (FIFO), the FIFO control unit (Control), and the Routing Control. When the input buffer outputs a header flit (the first flit of a packet), Control activates Routing Control, which requests the usage of the chosen OI through a transition on *req_outport_o*. Once the request is granted via the incoming *ack_outport_i* signal, the packet is transmitted through handshakes on *req_data_o* and *ack_data_i*. Using these two pairs of control signals allows concentrating all routing and arbitration overhead on the first handshake, reducing the latency of data transfers. After using the first flit, Control loads the payload size (the second flit of each packet) in a register and decrements it as the payload (the remaining flits) is transferred. When this register reaches zero, *last_flit_o* switches, indicating that the last flit of the packet is about to be transferred, and Control then waits for the next packet to arrive.

The buffer implements a circular FIFO as proposed in [4]. Since the *req_data_o* signal is shared among all OIs, and given the transition-signaling nature of the design that expects a transition on the acknowledge signal as requests are completed, the *ack_data_i* signals coming from the OIs can be on different phases during execution. Hence, an XOR gate is needed to phase-match the *ack_data_i* signals to *req_data_o*. Since each transition at the input of the XOR results on an output transition, this gate is used throughout the design as a phase-matcher for merging signal buses that are guaranteed by design to switch only one signal at a time. Differently from [4], the proposed router employs transition-signaling communication for all control signals. Additionally, to maintain compatibility with the environments used in the design and analysis of the YeAH! router, at the expense of area and performance overhead, BAT-Hermes uses a flit counter instead of begin-of-packet (BOP) and end-of-packet (EOP) flags to identify packet limits. A BOP/EOP strategy could potentially simplify the control circuit, resulting in smaller area and higher throughput. However, additional sideband signals between routers and IPs would be required to implement the BOP/EOP flags.

The OI handles arbitration between requests from IIs. Figure 2 depicts its architecture, which comprises an Arbiter, a multiplexer (MUX) and an output port control module (Outport Control) connected to each II interfacing to this OI. The arbiter ensures that only one II has access to the specific OI at a time. Ghiribaldi's et al. design [4] was reproduced

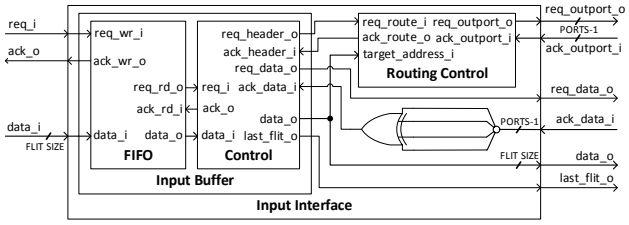

Fig. 1. Architecture of the Input Interface.

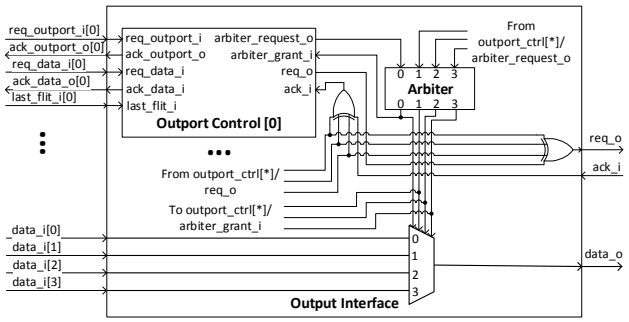

Fig. 2. Simplified architecture of the Output Interface.

using MUTEX cells from the ASCEnD library [11], [12]. Outport Control handles arbiter requests and last flit detection (*last_flit_i*). It also performs phase matching in cases where signals *req_data_i* and *ack_data_o* are out-of-phase. This phase matching is needed due to the shared *req_data_i* signal, similarly to what happens in the control signals of the II.

IV. DESIGN

A. The Input Interface

The II comprises three blocks: FIFO, Control and Routing Control. Details of the first are in [4]. Figure 3 details the (Input Buffer) Control circuit, which implements the logic that detects each packet's first and last flits. Control contains two blocks: a flit counter and the control logic for selecting which handshake signal sends each flit to the OI (*req_outport_o* for the first flit and *req_data_o* for the remaining flits of the packet). Flip-flops *FF1* and *FF2*, register *REG2*, and the adjoining logic compose the flit counter circuit (block *i*). The data stored in these is updated every time a new flit enters the circuit - i.e. a *req_i* transition propagates through *LT1*. Note that *LT1* is transparent whenever new information can be transmitted through Control - that is, the II is not waiting for an acknowledgement of any OI. As soon as a transition in *req_i* propagates through *LT1*, the latch becomes opaque because it triggers the XOR gate that feeds the *LT1* enable signal. *REG2* holds the number of flits remaining in the packet currently being transmitted. Its value is initialized with the payload size, read from the second packet flit, and decremented after sending each flit. Before sending the last packet flit, the value stored in the flit counter is 1. Once the last flit of the packet enters Control, *FF1* asserts its output, generating a rising edge on the signal connected to the clock pin of *FF3*, which in turn switches the value of *last_flit_o*, indicating the last flit will follow. When the next flit (a header) arrives, the value stored in *FF2* propagates to the output of *FF2*, changing the MUX selection to initialize *REG2* with the payload size when the following flit arrives.

The second block, composed of latches *LT2* and *LT3*, flip-flop *FF4* and the adjacent logic routes the request of the header flit to the *req_header_o* signal, and the remaining flits to *req_data_o* signal. The value stored in *FF4* indicates if the current flit is the first of the packet and controls the enabling

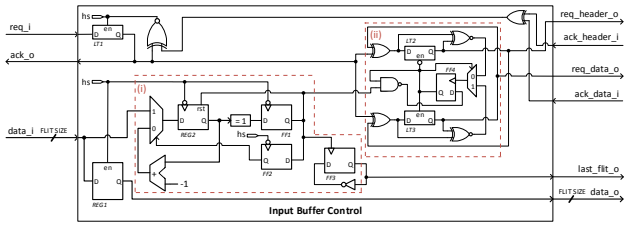

Fig. 3. Input Buffer Control used at the Input Interface.

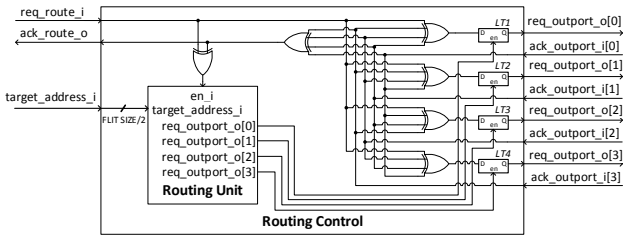

Fig. 4. Routing Control used at the Input Interface.

of *LT2* and *LT3* to select between header and data requests. The value of this register is determined as follows: *i)* If the current output of *FF4* is de-asserted (req_header_o handshake), the next value will be asserted (req_data_o handshake); *ii)* If the current output of *FF4* is asserted and the current flit is the last (*FF1* asserted), the next value will be de-asserted; iii) otherwise, it keeps the value asserted. *LT1* and *REG1* implement a pipeline stage to allow the buffer to fetch a new flit while Control is busy. When a *req_i* request arrives and Control is idle, the request and its data are immediately stored in *LT1* and *REG1*, respectively. As soon as the latches become opaque, an *ack_o* acknowledge is issued, liberating one position on the buffer. Routing Control is responsible for requesting the use of OIs. Figure 4 shows this circuit that comprises a Routing Unit and a set of latches and XOR gates, to keep phase coherence. A combinational circuit accomplishes the routing task, comparing the value of signal *target_address_i* to the current router address. This enables flexibility to adopt different routing algorithms. Once the computation is ready, the target OI latch is enabled, generating a request through the respective *req_outport_o*. Note that this module can be optimized by removing absent IIs and OIs, e.g. on corners and edges of a mesh network.

B. The Output Interface

The building blocks of the OI are an Arbiter, a MUX, and a set of Outport Control modules (each connected to corresponding IIs, up to four per router, per OI). BAT-Hermes employs the Arbiter proposed in [4], but implements it with custom MUTEX cells. This is a very important difference, as our approach enables a more realistic analysis. MUTEX cells are used to guarantee that only one requester will be granted access to the output interface in the event of several simultaneous requests. More importantly, these cells guarantee that no metastability will be propagated to the control path in the event of *eodem tempore* requests. Actually, it is not clear how a standard-cell implementation of a MUTEX avoids the injection of metastability in the control path, which can jeopardize correct functionality of the design. Unlike the handshake between the other components of BAT-Hermes, the Arbiter communicates using a level-signaling protocol. This was mandatory, since no transition-sensitive MUTEX is available in the used cell library. In fact, authors could find no such gate in the literature.

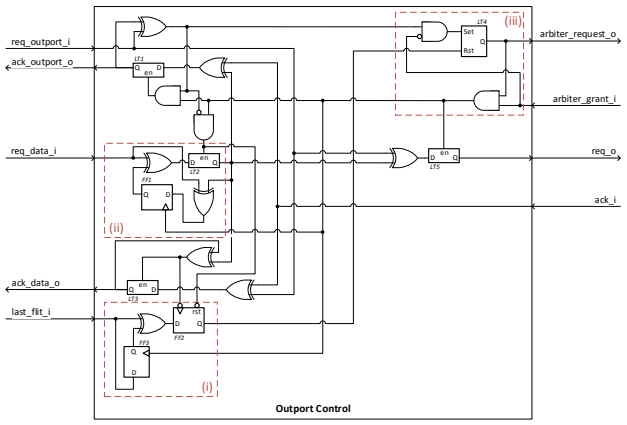

Fig. 5. Outport Control used at the Output Interface.

The Outport Control block controls interactions between IIs, the Arbiter, and the output channel. The circuit, detailed in Figure 5, is split in three main blocks: *i)* last flit detector (*FF2*, *FF3*, and the adjoining XOR gate); *ii)* a programmable phase matcher for the shared signal *req_data_i* (*FF1*, *LT2*, and the connected XOR gates); *iii)* the arbiter handshake control (*LT4* and the adjoining AND gates). The remaining latches and XOR gates serve to phase match request and acknowledge signals and keep these stable when Outport Control is idle. The last flit detector (block *i*) compares the initial value of *last_flit_i* stored in *FF3* to the current one, when acknowledging the transmission of each data flit. When a transition is detected, *FF2* asserts its output and the Outport Control relinquishes the OI. Since *req_data_i* is shared across all OI, it is necessary to guarantee that its initial phase at the input of *LT2* is the same as the signal stored in the latch - if not, the phase mismatch will be mistakenly interpreted as a request to transfer data. At the beginning of each packet transfer, the programmable phase-matcher (block *ii*) configures *FF1* to invert the phase of *req_data_i* when it does not match the value stored in *LT2*. The arbiter handshake control (block *iii*) requests the use of the OI when a header flit is received, by setting *LT4*, and relinquishes it when the last flit of the packet is received, by resetting the same latch.

C. Router Synthesis

BAT-Hermes timing constraints were manually extracted and used as input to a semi-automated synthesis flow based on the Synopsys synthesis framework. Timing constraints were automatically fulfilled in the physical synthesis using *set_min_delay* constraints to adjust the minimal delay of control lines, similar to [4]. However, in our design the ASCEnD cell library [11] provides asynchronous cells, we need not use gates from the conventional cell library to create such functionality. This enables more realistic analysis, because asynchronous cell construction with standard gates can introduce hazards in the circuit and should be avoided. The design targeted the STMicroelectronics 65nm standard Vt cell library and the router was synthesized employing 16-bit flit size and buffer depth of 8. The total gates area for the physical design was 30,013μm^2. Based on post-layout simulations we computed an average throughput of 3.85GB/s when the router simultaneously drives five connections. Given that timing constraints were extracted manually, we could not leverage all degrees of optimizations performed by the synthesis tool on synchronous designs. Additionally, the methodology used to set the delay line values do not optimize delay margins fully, resulting in performance penalties.

V. Experiments and Discussion

To analyze BAT-Hermes trade-offs, the YeAH! router also underwent synthesis and layout generation. This router employs a 16-bit flit size and an 8-flit buffer depth. It uses a traditional synchronous synthesis flow targeting the same technology. Preliminary results show that YeAH! achieves maximum throughput higher than BAT-Hermes. This is a consequence of the high degree of optimization allowed by synchronous tools. Also, this demonstrates that for high-speed applications, this router is a better option than BAT-Hermes, as for such applications the focus is in the maximum obtainable frequency. However, for low power applications, power efficiency must be the constraining metric, as high-speed designs are usually not the most power efficient. For synthesizing a design that would enable us to assess power efficiency trade-offs, we iteratively optimized the clock period of the router during the synthesis process until we obtained a slack margin smaller than 100ps for the critical path. This allowed the synthesis tool to map the design targeting the highest frequency it could with low driving strength gates, ensuring a reasonable area/delay trade-off. This means that allowing a big slack leads to underuse the driving capability of the logic gates. The resulting clock frequency was 667MHz.

From the layout we extracted the physical netlist and annotated nets and gates delays to an SDF file. With this file we conducted timing simulation to extract the switching activity. Design Compiler used the latter to compute power characteristics. Circuits were evaluated on the typical corner, operating at 25°C, standard Vt, and 1V supply. Evaluated performance metrics are: forward latency, maximum throughput, and power consumption. Forward latency is the average time it takes for a flit to traverse an initially idle router with empty buffers (thus, with no contention). Maximum throughput computation relies on the average cycle time of the router multiplied by the flit size and the number of simultaneously supported connections. For the synchronous design, the average cycle time is the clock period; on the asynchronous circuit, this value is measured based as the time between successive acknowledgements on the output interface. Power figures were measured in two scenarios: best and worst cases. In both test cases, five packets with 4,096 flits were simultaneously sent to/from all ports at the maximum possible rate - that is, one flit per clock cycle for YeAH!, and on BAT-Hermes a new flit issued immediately upon receiving an acknowledgment for the previous flit. In best case scenarios all flits were zero, to reduce circuit switching activity; in worst case ones, each flit was the inverse of the previous one, to increase switching activity.

Table I summarizes throughput and power results. Average forward latency of BAT-Hermes is 4.54ns; for YeAH!, it is 3 clock cycles: 4.5ns. YeAH! has a total cell area of 17,182μm^2 and BAT-Hermes has a cell area of 30,013μm^2, an overhead of 75%. The table also presents a measure of power efficiency in mJ/GB, from dividing the average power by the average throughput. This metric correlates performance and energy consumption, displaying the energy required to transfer a Gigabyte of data, enabling the assessment of the power efficiency of each design. These characteristics reflect the scenario used for the power measurement: five simultaneous connections with no contention and flit injection at the maximum possible rate. For both, the traffic best and worst cases, BAT-Hermes yields a better energy efficiency than its synchronous counterpart. This means that to transfer the same amount of data BAT-Hermes requires up to 27% less energy than YeAH!.

The area disparity between BAT-Hermes and YeAH! can be explained by the synthesis process of the former and by the different buffering strategies. Since commercial EDA tools do not support asynchronous circuits, it is necessary to manually define timing constraints and the circuit cannot leverage the timing optimizations supported in synchronous designs, which results in a design with larger area and timing overheads. Additionally, the more complex control circuit based on the packet size increases the number of timing constraints in the controller, requiring additional delay lines – which translates to area overhead. Regarding power efficiency, it is notable the advantage of the asynchronous design employing the transition-signaling handshake protocol. There is a clear trade-off between performance, area and power between the evaluated routers. YeAH! is clearly the best option for high-speed applications, as EDA frameworks allow better design space exploration for fully synchronous designs. The asynchronous nature of BAT-Hermes displays considerable improvement in power efficiency, at a higher cost in area. This is particularly important for contemporary technologies, as power constraints in modern VLSI applications are getting increasingly tighter.

TABLE I. Throughput, power and energy comparison.

Router	Throughput	Average Power		Energy per GB	
		Best	Worst	Best	Worst
YeAH!	6.2 GB/s	7.05 mW	9.12 mW	1.14 mJ/GB	1.47 mJ/GB
BAT-Hermes	3.85 GB/s	3.18 mW	4.15 mW	0.83 mJ/GB	1.08 mJ/GB

VI. Conclusions

This work focused on the design of a low power asynchronous transition-signaling BD NoC router. The router was synthesized to layout using commercial EDA tools and validated through post-synthesis simulation. Experimental results show reduction of energy consumption in the order of 27%, when compared to a fully synchronous router with similar characteristics. As future work we will explore optimizations in the synthesis process employed to enable timing constraints optimization and improve area and timing.

References

[1] A. J. Martin and M. Nystrom, "Asynchronous Techniques for System-on-chip Design," *Proceedings of the IEEE*, vol. 94, no. 6, pp. 1089–1120, 2006.

[2] K. Stevens *et al.*, "The Future of Formal Methods and GALS Design," *Electronic Notes in Theoretical Computer Science*, vol. 245, pp. 115–134, 2009.

[3] S. Nowick and M. Singh, "High-performance asynchronous pipelines: An overview," *IEEE Design & Test of Computers*, vol. 28, no. 5, pp. 8–22, Sep 2011.

[4] A. Ghiribaldi *et al.*, "A Transition-signaling Bundled data NoC Switch Architecture for Cost-effective GALS Multicore Systems," in *Design & Test in Europe (DATE)*, pp. 332–337.

[5] P. Beerel *et al.*, *A Designer's Guide to Asynchronous VLSI*. Cambridge University Press, 2010.

[6] A. Sheibanyrad *et al.*, "Multisynchronous and Fully Asynchronous NoCs for GALS Architectures," *IEEE Design Test of Computers*, vol. 25, no. 6, pp. 572–580, Nov 2008.

[7] T. Bjerregaard and J. Sparso, "A router architecture for connection-oriented service guarantees in the mango clockless network-on-chip," in *Design, Automation and Test in Europe (DATE)*, pp. 1226–1231.

[8] R. Dobkin *et al.*, "QNoC Asynchronous Router," *Integration the VLSI Journal*, vol. 42, no. 2, pp. 103–115, Feb. 2009.

[9] M. Moreira *et al.*, "The YeAH! NoC Router," Faculty of Informatics, PUCRS, Tech. Rep. 083, Dec. 2014.

[10] F. Moraes *et al.*, "HERMES: An Infrastructure for Low Area Overhead Packet-switching Networks on Chip," *Integration the VLSI Journal*, vol. 38, no. 1, pp. 69–93, Oct. 2004.

[11] M. Moreira *et al.*, "A 65nm Standard Cell Set and Flow dedicated to Automated Asynchronous Circuits Design," in *IEEE International System-on-Chip Conference (SOCC)*, 2011, pp. 99–104.

[12] ——, "Adapting a C-element Design Flow for Low Power," in *IEEE International Conference on Electronics, Circuits, and Systems (ICECS)*, 2011, pp. 45–48.