Journal of
Low Power Electronics and Applications

MDPI

Article

Ultra Low Energy FDSOI Asynchronous Reconfiguration Network for Adaptive Circuits †

Soundous Chairat *, Edith Beigne, Ivan Miro-Panades and Marc Belleville

University Grenoble Alpes, CEA, Leti, F-38000 Grenoble, France; edith.beigne@cea.fr (E.B.); ivan.miro-panades@cea.fr (I.M.-P.); marc.belleville@cea.fr (M.B.)

* Correspondence: soundous.chairat@cea.fr; Tel.: +33-(0)4-38-78-26-21

† This paper is an extended version of our paper published inS. Chairat, E. Beigne, F. Berthier, I. Miro-Panades and M. Belleville, "Ultra low energy FDSOI asynchronous reconfiguration network for an IoT wireless sensor network node," 2016 IEEE SOI-3D-Subthreshold Microelectronics Technology Unified Conference (S3S), Burlingame, CA, USA, 2016, pp. 1–3.

Academic Editors: David Bol and Steven A. Vitale
Received: 2 March 2017; Accepted: 4 May 2017; Published: 11 May 2017

Abstract: This paper introduces a plug-and-play on-chip asynchronous communication network aimed at the dynamic reconfiguration of a low-power adaptive circuit such as an internet of things (IoT) system. By using a separate communication network, we can address both digital and analog blocks at a lower configuration cost, increasing the overall system power efficiency. As reconfiguration only occurs according to specific events and has to be automatically in stand-by most of the time, our design is fully asynchronous using handshake protocols. The paper presents the circuit's architecture, performance results, and an example of the reconfiguration of frequency locked loops (FLL) to validate our work. We obtain an overall energy per bit of 0.07 pJ/bit for one stage, in a 28 nm Fully Depleted Silicon On Insulator (FDSOI) technology at 0.6 V and a 1.1 ns/bit latency per stage.

Keywords: on-chip communication network; adaptive blocks; asynchronous design

1. Introduction

The complexity of system-on-chip (SoC) is ever growing, while the constraints on power consumption are increasingly tightened. To address this problem, reconfigurable blocks are integrated in SoCs to allow a maximum trade-off between power and performance. These blocks can adjust their performance based on the available energy budget, on the application requirements, or on the environmental conditions [1]. Furthermore, at system level, it is interesting to get some information back from these blocks to adjust their performances. A complex SoC usually has both analog and digital blocks in several power domains. For example, a wireless sensor node can integrate reconfigurable blocks of various complexities, such as receivers [2], analog-to-digital converters (ADCs) [3], image sensors [4], and digital blocks like clock and power generators. When implementing several of these adaptive blocks on the same circuit, the problem of how to reconfigure them efficiently and with a minimum energy loss arises.

Looking at the state-of-the-art of on-chip communication networks, we can distinguish two types of networks: networks to send and receive functional data and networks dedicated to reconfiguration. Data networks such as the AMBA (ARM) [5], the Wishbone (OpenCores) [6], and the CoreConnect (IBM) [7] are buses used in SoCs to efficiently transport large amounts of data at high speed. The frame can accommodate up to 256 bits for the data bus, and 32 bits for the address bus, with a high throughput. The aim is for the data to be transferred as fast as possible, without strict energy constraints. In this case, the higher the throughput, the better. Another type of data network is the network-on-chip (NoC) [8], which has emerged in response to the increase of integration in system-on-chip (SoC),

as well as the shrinkage of technology. The SoC is split into different islands, with each island accessible through an interface connected to the data network, which allows more efficient routing of the data. The NoC paradigm has introduced new energy-efficient networks with lower latencies and better performances overall for transferring data in SoCs. An interesting category of NoCs is the asynchronous NoCs (ANoCs) [9], which are mainly implemented in globally asynchronous locally synchronous (GALS) [10,11] architectures—especially for sub-micron technologies. Thanks to the nature of asynchronous logic, these networks can bypass problems caused by the clocking distribution in an SoC, such as clock skew. Moreover, it makes access to multiple power islands much easier. This approach has proved efficient, and demonstrated that asynchronous design can be adequate to implement communication networks [12]—especially to reduce clocking problems in SoC. However, due to their complexity and the size of the SoC or multicore itself, ANoCs have not been employed for low-complexity networks.

Dedicated networks or performance-enhancing networks are usually added on top of a functional network and are mostly used in multi-core systems, with their own dedicated processor [13,14]. They are used to collect performance-related data such as temperature from intellectual properties (IPs) in order to monitor the circuit performance and adjust them when needed. This type of network has been successfully implemented in many SoCs. However, each network had to be tailored to the SoC and its specificities. The use of dedicated networks has proven to be efficient with a minimum of area overhead [15].

Since configuration data can be sent both at runtime and when the circuit is in idle mode, using the normal data network to transfer it is not efficient. When the configuration data is sent at runtime using the intrinsic data network, the transfer of functional data is put on hold while reconfiguration data is sent. When reconfiguration happens at idle times, the circuit needs to be powered back up to transfer the data, which is not energy efficient. Using a data network for reconfiguration can be costly energy-wise.

In this work, we target different applications (e.g., internet of things (IoT) systems) based on ultra-low-power mixed signal circuits having constraints far apart from the previous multicores or SoCs, not requiring intricate interfaces and links. Therefore, our proposed reconfiguration communication network is very low power (pJ/bit), and the circuitry's complexity is kept at a minimum (<1800 gates for the interface) to allow the fine grain tuning of small elementary blocks. We propose a network that can dynamically reconfigure any adaptive block, have a fast wake up and an automatic sleep mode thanks to the use of asynchronous logic. Moreover, its deployment is extremely easy due to its asynchronous implementation. There are no clock distribution problems, and power domain crossing is no longer an issue. Moreover, it allows for an easy plug-and-play approach, since the network's interface can be connected to different types of adaptive blocks, regardless of their architecture or which power domain it will be in. Indeed, quasi-delay-insensitive (QDI) asynchronous logic is always functional and insensitive to delay variations due to voltage domains, temperature, or process changing. To test our network, four digital frequency locked loops (FLLs) [16] are embedded in a realistic test case. The network is implemented in a 28 nm Fully Depleted Silicon On Insulator (FDSOI) technology at 0.6 V. The obtained energy per bit is of 0.07 pJ/bit per stage, and the latency is 1.1 ns/bit.

This paper is structured as follows. In Section 2, we discuss the necessity of having adaptive blocks in a mixed signal circuit, especially for IoT application. In Section 3, we present the network's general architecture. Section 4 gives an overview of the asynchronous logic used in designing the circuit and details the architecture of the network and its components. Finally, in Section 5, we present the performance results of our circuit.

2. Adaptive Blocks for Mixed Signal Circuits

Today's integrated systems and designers are facing the increasingly difficult task of designing systems that have to handle very tight sets of specifications, or even multiple sets of specifications, required for adequate performance. Given the numerous variables encountered by the designers

during the design process, including—but not limited to—the variations in the fabrication process, temperature, and transmission media, it becomes very challenging to implement systems that can meet those tight specifications and yet keep high levels of integration and preserve cost-effectiveness.

Three main categories of variation exist: variation at circuit level [17], including process, supply voltage, and temperature variations; variation at system level, where the specifications of a circuit need to be changeable (for example, a transceiver needs to automatically handle multiple standards [18]); and variations in the transmission media, which is at the communication and environment level category [19].

To handle these variations—preferably automatically—adaptation is the key. Adaptation is a very broad technique that can be applied to many fields in the area of analog and mixed signal circuit design [20,21]. The adaptation is based on a Sense and React operation. The value of a parameter is probed, and compared to a reference value. If there is a difference between the two values, the circuit Reacts to change the value of the parameter, in order to minimize or eliminate the variation. Circuits and systems that employ adaptation techniques are called adaptive circuits and systems. These circuits have many uses—especially in IoT systems, where typical constraints are multi-standards wireless communication, various and changing environments, and ultra-low energy consumption leading to a suppression of all over-specifications.

Our proposed circuit deals with the transfer of the sense and react data. The network is responsible for sending the Sense commands and the React configuration data to the adaptive blocks, and collects the sensed values from adaptive blocks and transmits it to an external microcontroller, for instance.

3. General Architecture

The proposed asynchronous service network (ASN) is shown in Figure 1, and is composed of two main blocks: a serial interface controller (SIC) and the interfaces that are directly connected to the adaptive blocks (ABs). The serial interface controller (described in Section 4.3) receives the configuration data from a microcontroller (which is not shown), and passes them to the interfaces described in Section 4.2. The communication protocol is serial, and the interfaces are connected in a daisy chain topology to reduce the number of metal wires needed and have the least intrusive deployment of the network.

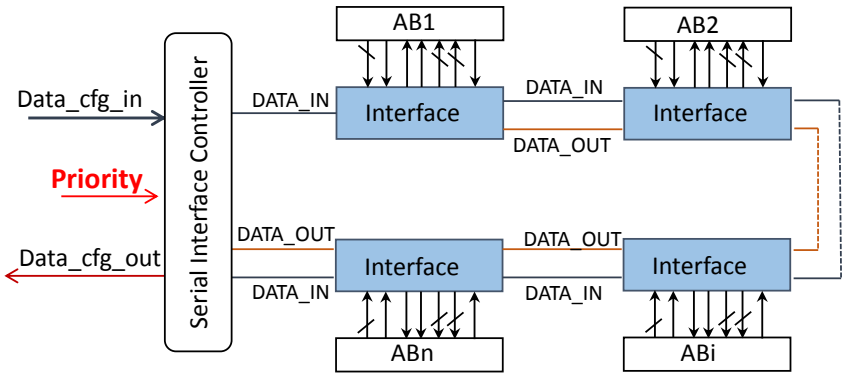


Figure 1. Proposed communication network. AB: adaptive block.

3.1. Frame and Topology

The aim of the network is to achieve a good power efficiency at the least possible area cost. As such, the frame needs to be as compact and small as possible, and the chosen topology has to be area effective. In the following paragraphs, the choice of topology and frame is explained.

3.1.1. Frame Structure

The frame associated with this construct is straightforward (Table 1), and contains only the necessary data: a priority bit, a bypass bit, the addresses of the adaptive blocks and of its internal registers, the Read/Write bit, and the configuration data (when relevant). Usually, in a serial network, start and stop bits are needed as well as acknowledgement bits to verify that the data has reached its destination. However, because our network is based on asynchronous logic, these bits are not necessary, as synchronization is automatically handled. The priority bit allows us to signal to the interface that the incoming configuration data is a priority and has to be processed immediately. We also chose to add a bypass bit to the frame. If a new frame reaches the interface it is intended to, and if this interface is still busy, the bypass bit is set to "1" by the interface, and the frame continues through the network. This bypass bit tells the following interfaces that they do not need to process the incoming data, which allows a gain in latency and energy. After reaching the SIC, this latter knows it has to send the frame again in the network after switching the bypass bit to "0". The return frame received after a read operation has only the address of the block, the address of the register, and the data.

Table 1. Structure of the frame used.

a Data to the interface					
Pr	**Bp**	**addr_bloc**	**addr_reg**	**rw**	**Data**
1 bit	1 bit	4 bits	4 to 8 bits	1	8 to 32 bits
b Data from the interface					
addr_bloc	**addr_reg**	**Data**			
4 bits	4 to 8 bits	8 to 32 bits			

3.1.2. Asynchronous Service Network Topology

The bus topology represents a good latency/complexity trade-off, and is already used for many networks. However, an asynchronous channel can only be read by one block at a time. To send asynchronous data through a bus to several blocks, the data needs to be duplicated (Figure 2) as many times as there are blocks to ensure a correct handshake protocol. Thus, we chose to implement a daisy chain topology as a better solution. Only one asynchronous channel will run through all interfaces, so the wire count will be diminished significantly. In a bus topology and for a four-block network, there will be 65% more wires than in a daisy chain topology. Moreover, when using a daisy chain topology, if an interface is busy and cannot accept the new configuration data (because the previous ones were not processed yet), the data will only have to continue through the network until it reaches the SIC and be sent again.

On the other hand, using a daisy chain topology implies a greater latency, because the current block has to finish analysing the data before sending it to the next block. However, since the network is small and does not contain many blocks, the main constraint is the wire count and not the latency of the network.

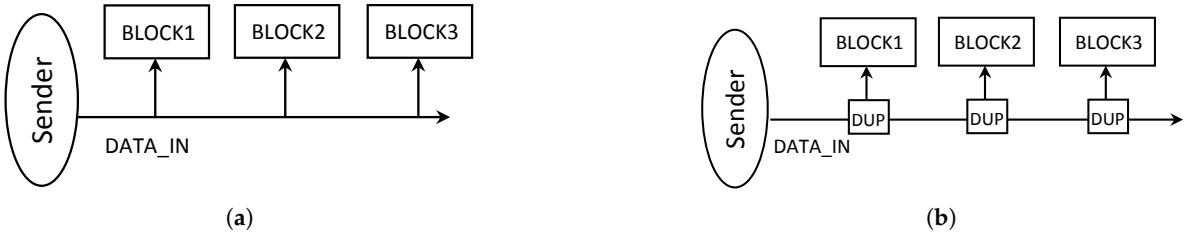


Figure 2. Bus topology for synchronous and asynchronous networks. (**a**) Normal bus topology; (**b**) Asynchronous bus topology.

4. Network's Micro-Architecture

As mentioned previously, we used asynchronous logic [22] to implement the asynchronous service network. It allows us to achieve a plug-and-play implementation and avoid power domain crossing problems common to many SoC circuits [23], since we no longer have to worry about timing problems. It also enables us to have a network with a fast wake-up and an automatic stand-by mode without implementing gated clock logic. In the next section, a small overview of asynchronous logic is given, followed by the detailed architecture of both the serial interface controller and the interfaces.

Asynchronous design is a design methodology for digital circuits where no clock is used. Instead, a handshaking protocol is used to signify when it is correct to process data. When the sender is ready to send data to the next block (the receiver), a request signal (Req) is sent to the receiver, which in turn sends an acknowledgement *(Ack)* to the sender to signify that it is ready to receive the data (Figure 3). Once the data is sent, a new acknowledgement signals that the data was correctly received. Unless the *Ack* is sent, the sender cannot receive any data from any other block. This blocking communication scheme ensures that the data in the sender is not lost or replaced, and that the appropriate data has been sent correctly.

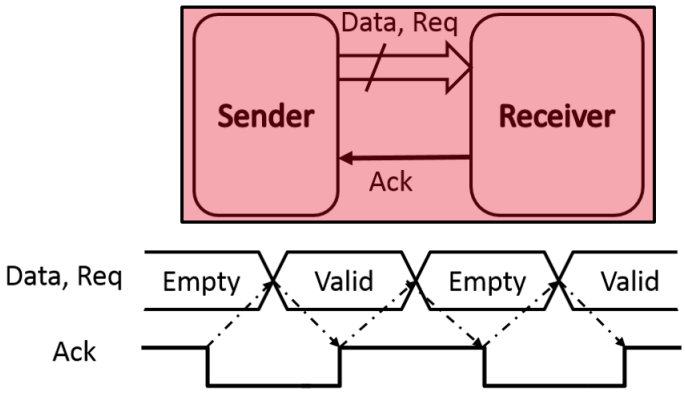
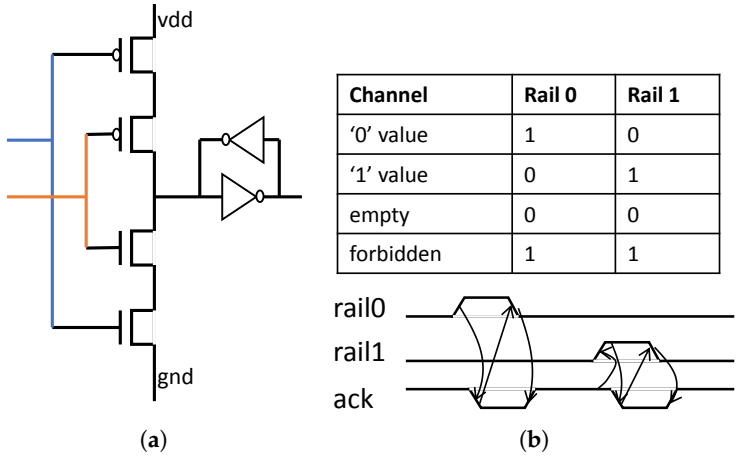


Figure 3. Four-phase dual rail protocol.

4.1. Asynchronous Design

In this design, we are using a four-phase dual rail protocol, where the data request is encoded within the data (i.e., each wire is duplicated to carry the request with it—Figure 4b). For instance, when we send a bit "0" through the channel, only rail0 is activated. Once the data has been received, the *Ack* signal goes to zero, and waits for the rail0 signal to go back to zero to have the empty value (when both rail0 and rail1 are at "0"). Once the *Ack* signal is active again, we can send new valid data. This protocol is possible thanks to the Muller gate (also called C-gate), which manages the acknowledgement (Figure 4a) [24]. This gate switches when both inputs are coincident, therefore acting as a *Rendez-vous* for the inputs. In an asynchronous circuit, it allows synchronization of the events and ensures the correct acknowledgement.

Both the SIC and the interfaces are designed using asynchronous logic, and each interface is connected to the synchronous reconfigurable block via a custom asynchronous-to-synchronous interface. The following paragraphs introduce the architecture of the interface and the SIC.



Channel	Rail 0	Rail 1
'0' value	1	0
'1' value	0	1
empty	0	0
forbidden	1	1

Figure 4. Asynchronous design basics. (**a**) Two-input Muller gate; (**b**) Dual Rail protocol.

4.2. Network's Interface Architecture

Each adaptive block is connected to the communication network through an interface. The interface architecture is shown in Figure 5a, and has less than 1800 gates. The interface has a first input channel connected to the network to receive the configuration data *DATA_IN*. A second input channel is connected to the adaptive blocks to receive data after a read operation *DATA_R_IN*, and a third input is connected to the previous interface to pass along the read data through the network *P_BLOC_IN*. Figure 6 shows how two interfaces are connected. The interface is composed of a comparator block that compares the address of the interface, a register block to keep the data, and a serial-to-parallel converter in order to convert the data sent to the adaptive block. When the interface receives the configuration frame, it first compares the address of the block in the frame to its own address. If the addresses match, the interface stores the data and then sends a request to the adaptive block to write the data into its registers. Once the operation is finished, the network is inactive until the next frame is received. If it does not match, then the data is passed along to the next interface. However, if the addresses match but the previous configuration data was not yet sent to the adaptive block, then the data is sent back through the network to go back to the SIC and be sent again at a later date.

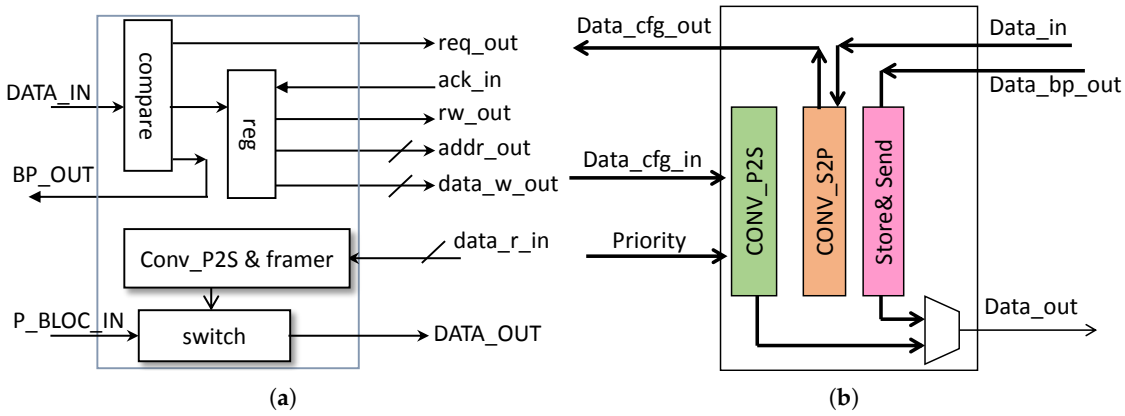


Figure 5. Serial interface controller (SIC) and interfaces architecture; (**a**) Interface architecture; (**b**) SIC architecture.

The adaptive blocks can be either asynchronous or synchronous. When the adaptive block is synchronous, an asynchronous-to-synchronous interface is needed. The interface converts the

dual rail encoding to a single rail encoding (Figure 7a) and vice-versa (Figure 7b) when needed. It also synchronizes the exchange of data using C-Muller gates and acknowledgement signals (*Ack_w* and *Ack_r*).

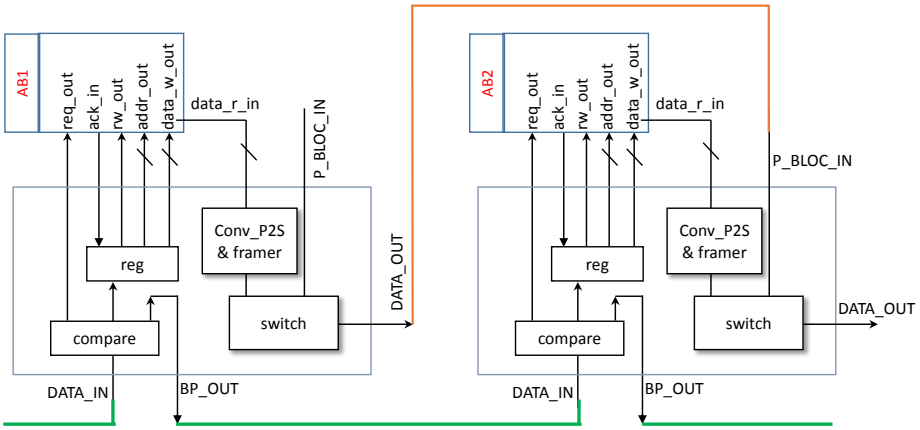


Figure 6. Two daisy-chained interfaces.

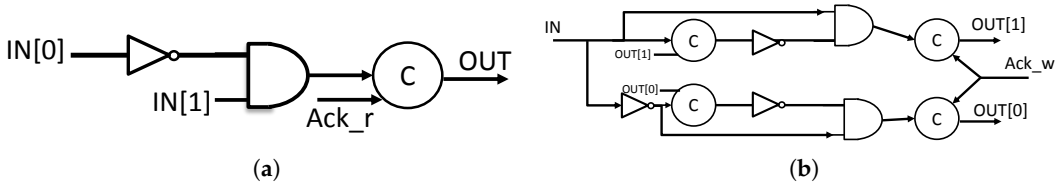


Figure 7. Asynchronous-to-synchronous conversion basic blocks; (**a**) Dual rail-to-wire encoding; (**b**) Wire-to-dual rail encoding.

4.3. The Serial Interface Controller Architecture

Similar to the interface, the SIC is asynchronous (Figure 5b). The SIC has two main roles: the first is to process the configuration data (either the data sent to the adaptive blocks or received from the adaptive block) and the bypassed data, and the second is to handle the priority data. The SIC is composed of three blocks. The first is a parallel-to-serial converter (CONV_P2S) which receives the parallel data from the microcontroller, adds the bypass bit to the frame, then sends it to the network serially. The second block is the serial-to-parallel converter (CONV_S2P) which receives serial data from the network (the result of a read operation), converts it to parallel data, and sends it to the microcontroller. The third block is the Store&Send block, which receives the bypassed serial data from the network and sends it back to the network. The SIC has three input channels: two to receive data from the interface (one for the read data and the other for the bypassed data) and one for the frame coming from the microcontroller. The parallel data received from the microcontroller simply replicates the frame structures without the bypass bit, which is added by the SIC. The two types of frames it receives from the interfaces are the bypassed data and the sense data sent after a read operation of the adaptive blocks. The first type of frame is stored to be sent back to the network, and the second frame is sent to the microcontroller after converting it to parallel data. Upon receiving the bypassed data, the SIC checks if any data is being sent to the network. If that is the case, the SIC waits for the data to be sent, then sends the bypassed data; if not, then the bypassed data is sent without delay. The same is true for the regular configuration data. Once the SIC receives configuration data from the microcontroller, it checks an internal priority flag first (set by the PRIORITY input). If the priority bit of the frame is at "1", it sends the data through the network; if not, the data is disregarded, and the SIC waits until the correct high priority frame arrives to send it through the network, after which, the priority flag is reset. When the priority flag is at "0", the SIC simply sends the data to the network.

5. Test Case and Associated Service Network

In the following paragraph, we implement the asynchronous service network in a reconfigurable circuit based on reconfigurable frequency locked loop (FLL) as shown in Figure 8, in a 28 nm FDSOI technology. This approach allows us to validate our work and to analyze the network's performances. In the following, we will present the results of two possible implementations of our circuit (serial and hybrid), and report and analyze the obtained results in area, power, throughput, and latency. Both networks were fully Placed and Routed on this technology as seen on Figure 9. In the following, simulation results are given from post back-end parasitic extraction. The serial network architecture was also implemented on silicon, and Table 2 presents the results in latency and throughput, for both the post back-end simulations and silicon measurements.

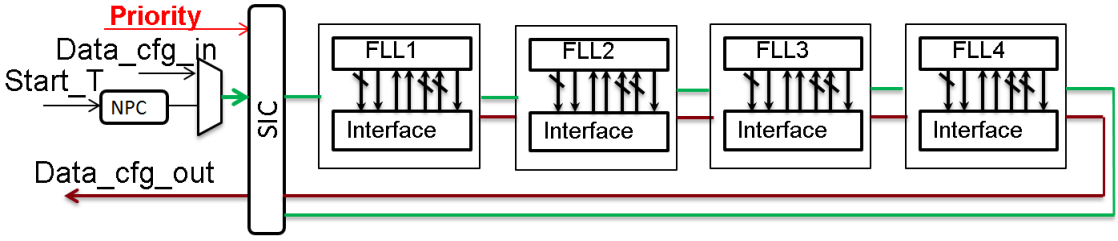


Figure 8. Communication network connected to four frequency locked loop (FLLs) for reconfiguration and performance estimation.

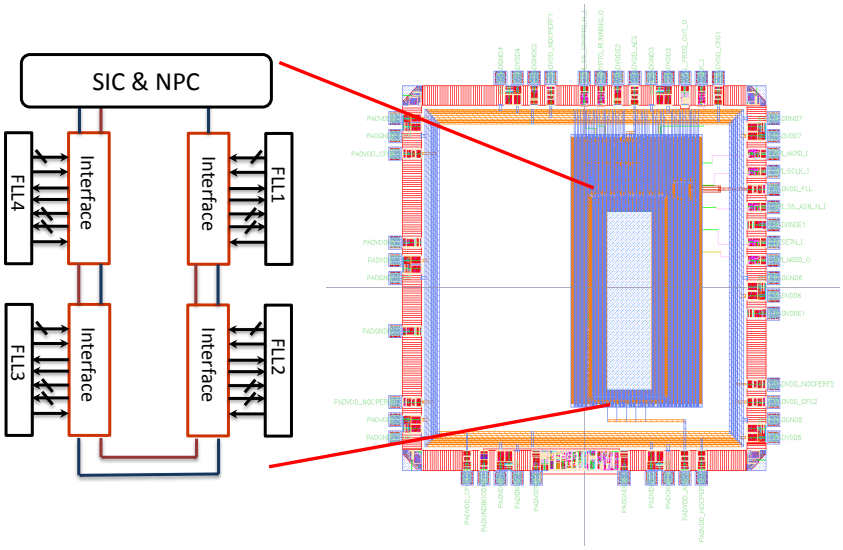


Figure 9. View of the the fully placed and routed network. NPC: network performance characterisation.

Table 2. Serial implementation performance results post back-end and on silicon @ 0.6 V.

	Post Back-end	Silicon Results
Latency Interface 1	20 ns/bit	20 ns/bit
Latency Interface 2	22 ns/bit	23 ns/bit
Latency Interface 3	24 ns/bit	25 ns/bit
Latency Interface 4	26 ns/bit	27 ns/bit
Throughput	37 Mbits/s (800 kflits/s)	37.7 Mbits/s (820 kflits/s)

5.1. Frequency-Locked-Loop and Network's Performance Characterisation Blocks Design

Digital FLLs [16] are used in many circuits to generate a stable clock, and are also used in dynamic voltage and frequency scaling (DVFS) and adaptive voltage and frequency scaling (AVFS) digital

architectures for power management. FLL's main advantages are a fast frequency reconfiguration and a very low area. For our circuit, we chose to reconfigure four FLLs and extrapolate the results to a greater number. We chose to work with FLLs because frequency is one of the main parameters we can change to reconfigure a circuit. For test purposes, we used a serial peripheral interface bus (SPI) to send the data to our network through a bisynchronous First-in First-out memory (FIFO), which acts as a synchronous-to-asynchronous interface (not shown). To estimate the throughput and latency, a small asynchronous block called network performance characterisation (NPC) for performance estimation was designed and integrated in our circuit, along with a test module (TM) comprised of two counters: a fast counter that can go up to 2 GHz and a normal counter, as well as an FLL that provides the frequency to the counters. The test module is synchronous, while the NPC is asynchronous. When a test needs to be done, the TM receives a signal to start the counter. Once the counter starts, the TM sends an asynchronous signal to the NPC to start the measurement. When the measurement is completed, the NPC sends an asynchronous signal to the test module to stop the counting.

The NPC itself contains a testing finite-state machine (FSM) and a multiplexer (MUX). The MUX is used to separate the data we send to the SIC for a normal operation and the data we send for a measurement. To accommodate the new setup, we added a bit to the frame we send from the microcontroller that will specify which operation we want to conduct (measurement or normal), and expanded the ADDR_BLOC from 4 bits to 8 bits when we want to make a measurement. When we want to measure a latency or the throughput, we simply have to set the measurement bit to "1", and then the MUX will pass the incoming message to the testing FSM. The four first bits of the ADDR_BLOC specify which type of measurement we wish to conduct. Then, depending on that, an event is sent to the SIC and interfaces to warn that the incoming data are test data. We then send the rest of the frame to the network that operates normally. For example, to calculate the write latency of the second adaptive block, we first send the correct frame, with the measurement bit switched to "1". Once the FSM gets the frame, it checks to see what kind of measurement it needs to do. Since it is a write latency measurement of the second block, the FSM sends a signal to the corresponding interface (in this case, the second interface) to warn it that the incoming data are for measurement. The FSM then sends the data to the SIC, which sends it to the corresponding interface, and the counter starts counting. The data is processed once it reaches the corresponding interface, but instead of being sent to the adaptive block, the data is disregarded and a signal is sent back to the FSM to stop the counter. We can then get the value of the counter, and have a correct latency measurement. For the throughput, we send a hundred frames to the network.

The latency, throughput, and power consumption were calculated post back-end with sdf back-annotation. Furthermore, to be as representative as possible, each interface was positioned a distance away from the next one (Figure 9) to simulate a realistic communication implementation. The UTBB FDSOI 28 nm technology was chosen, as it corresponds to today's ultra-low-power platform [25], and the simulations were conducted at 0.6 V. The proposed network and FLL's configuration scheme are represented in Figure 8. The chosen frame contains 47 bits, as shown in Table 3.

Table 3. Test case's frame.

Pr	bp	addr_bloc	addr_reg	rw	Data
1 bit	1 bit	4 bits	8 bits	1 bit	32 bits

5.2. Results of Fully Serial Implementation

Thanks to the network performance characterisation block, we were able to accurately determine the latency and throughput of the network. Table 2 reports the implementation results regarding the latency and the throughput, both post back-end and on silicon. The latency to reach each one of the four interfaces considers the SIC latency as well as the link and the time it takes to bypass a previous interface. The post back-end simulation results and silicon measurements are close. Table 4 gives a

more detailed partition of the latency, but only for post back-end simulations. As can be seen, the latency of the interface is important, and is due to two main reasons. The first is technology related: the high Vt (low leakage) cells used negatively impact the latency. The second reason is architecture related: a counter is integrated in each interface. When the interface receives the serial data, it does not know how many bits to expect, and which bit corresponds to what. To help with that, a counter was added to the interface to count the incoming bits. However, because the counter is asynchronous, between each bit, we need to wait for the counter to increment, then send the acknowledgement, which increases the latency. However, the throughput simulated in this case is 800 kflits/s (37 Mbits/s), which is more than enough for the targeted applications. Moreover, since the majority of the latency in our implementation comes from the counter, the latency is reduced considerably for smaller frames (smaller frames means a smaller counter). For a 26-bit frame (four bits address register and 16 bits data), we can reduce the latency by a third.

To estimate the cost of a reconfiguration, we need to consider the contribution of the SIC and every interface the frame has to go through before it reaches its correct destination. As can be seen in Table 4, the energy per bit used by the SIC remains the same for every reconfiguration; we only need 0.03 pJ/bit every time we need to configure a FLL. The interfaces contribute in two ways: when a frame simply goes through an interface, or when it is the interface of the intended FLL. In both cases, the contribution is 0.92 pJ/bit from each interface. In this case, because the latency is very important, the energy per bit used is also high. However, when configuration requires less bits, we estimate that the energy per bit used is lowered by half with a smaller counter.

Thus, for small frame size and to have a minimum of metal wire impact, a completely asynchronous serial network is good. However, for larger frames, another configuration network was devised, as discussed below.

Table 4. Serial and hybrid implementation performance and wiring results.

		Serial Implem	**Hybrid Implem**
SIC	energy	0.03 pJ/bit	0.01 pJ/bit
	latency	0.11 ns/bit	0.09 ns/bit
	leakage	282.5 nW	300 nW
Interface	energy	0.92 pJ/bit	0.04 pJ/bit
	latency	17 ns/bit	0.90 ns/bit
	leakage	217 nW	73.6 nW
Link	energy	0.04 pJ/bit	0.02 pJ/bit
	latency	0.70 ns/bit	0.10 ns/bit
	leakage	6 nW	35.1 nW
nbr of wires		6	24

Results of the Hybrid Implementation

This second implementation is intended to avoid the increase in latency created by the counter in the serial implementation. For the hybrid implementation, the frame shown in Table 3 is split into six flits, where each flit is composed of 8 bits of data and one bit which indicates the end of frame (eof) as shown in Table 5. Each part is then sent in parallel throughout the network. In the case of a read operation, only two flits are sent by the SIC, since the read/write bit acts as the eof bit. For a write operation, up to 6 flits can be sent, depending on the data size (8/16/24/32 bits). In this implementation, instead of having a one-bit channel for the configuration data connecting each interface, a nine-bit channel is used, which translates to nineteen wires in total (eighteen to encode the data in a double rail QDI logic and one for the acknowledgement). A 2-bit channel (five wires) is used this time to send the sense data. The first bit contains the actual sense data, and the second bit serves as an end of frame bit: When at "1", the SIC will know that he had reached the end of the frame; otherwise, it needs to continue receiving data. In total, the number of wires in this network is twenty-four.

Table 5. Hybrid frame structure. eof: end of frame.

1st Flit		2nd Flit		3rd–6th Flit	
addr_bloc	eof	addr_reg	rw	data	eof
8 bits	1 bit	8 bits	1 bit	8 bits	1 bit

This time, 54 bits of data were sent through the network (including the end of frame bits). As can be seen in Table 4, the second implementation's energy consumption is extremely low, as is the latency. This is because we no longer need a counter. The parallelization, coupled with the use of an end of frame bit resulted in a decrease in latency, which also positively impacted the energy per bit used. We also calculated a throughput of 98.7 Mflits/s (0.88 Gbits/s), which is quite good—especially at 0.6 V.

As mentioned below, the decrease of latency favourably impacted the energy needed per bit. We only need 0.04 pJ/bit for each interface for a write operation. In the case of a read operation, the energy per bit needed is the same, but distributed differently. In total, the energy per bit needed for one interface, the SIC, and one link is of 0.07 pJ/bit.

6. Conclusions

This paper presented a plug-and-play dynamic reconfiguration network for adaptive blocks in a mixed signal circuit. Two different versions of the circuit were implemented in a 28 nm FDSOI technology, and all simulations were done post-back-end with SDF-annotation at 0.6 V. The networks are both designed using asynchronous logic in a daisy chain topology where the first implementation is serial and the second is partially parallel. The second implementation proved to be better in terms of latency and energy, as we only need 1.1 ns/bit and 0.07 pJ/bit for one stage. However, for small frame size and to have a minimum of metal wire impact, a completely serial network is preferable.

Acknowledgments: This work has been performed in the THINGS2DO project (JTI Contract Number 621221), co-funded by grants from France and the ECSEL Joint Undertaking.

Author Contributions: Soundous Chairat designed, implemented and tested the communication networks, supervised by Edith Beigne and Marc Belleville; Edith Beigne and Ivan Miro-Panades worked on the FLL design and implementation. Soundous Chairat wrote the paper, with contributions from the other authors.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Hinkelmann, H.; Zipf, P.; Glesner, M. Design Concepts for a Dynamically Reconfigurable Wireless Sensor Node. In Proceedings of the First NASA/ESA Conference on Adaptive Hardware and Systems (AHS), Istanbul, Turkey, 15–18 June 2006.
2. Taris, T.; Mabrouki, A.; Kraimia, H.; Deval, Y.; Begueret, J.B. Reconfigurable Ultra Low Power LNA for 2.4 GHz Wireless Sensor Networks, Electronics. In Proceedings of the 2010 17th IEEE International Conference on Circuits, and Systems (ICECS), Athens, Greece, 11–14 December, 2010.
3. Weltin-Wu, C.; Tsividis, Y. An event-driven clockless level-crossing ADC with signal-dependent adaptive resolution. *J. Solid State Circ. (JSSC)* **2013**, *48*, 2180–2190.
4. Bernard, T.; Zavidovique, B.; Devos, F. A programmable artificial retina. *J. Solid State Circ. (JSSC)* **1993**, *28*, 789–798.
5. AMBA Bus Specifications. Available online: http://www.arm.com/products/system-ip/amba-specifications.php (accessed on 6 May 2017).
6. Wishbone Specifications. Available online: http://cdn.opencores.org/downloads/wbspecb3.pdf (accessed on 6 May 2017).
7. Core Connect Bus Specifications. Available online: https://www-01.ibm.com/chips/techlib/techlib.nsf/products/CoreConnect_Bus_Architecture (accessed on 6 May 2017).
8. Jantsch, A.; Tenhunen, H. *Networks on Chip*; Kluwer Academic Publishers: Dordrecht, The Netherlands, 2003.

9. Amde, M.; Felicijan, T.; Efthymiou, A.; Edwards, D.; Lavagno, L. Asynchronous on-chip networks. *IEEE Proc. Comput. Dig. Tech.* **2005**, *152*, 273–283.
10. Krstic, M.; Grass, E.; Gurkaynak, F.K.; Vivet, P. Globally Asynchronous Locally Synchronous Circuits: Overview and Outlook. *IEEE Des. Test Comput.* **2007**, *24*, 430–441.
11. Beigne, E.; Clermidy, F.; Vivet, P.; Clouard, A.; Renaudin, M. An asynchronous NOC architecture providing low latency service and its multi-level design framework. In Proceedings of the 11th IEEE International Symposium on Asynchronous Circuits and Systems (ASYNC), New York, NY, USA, 14–16 March 2005.
12. Bainbridge, J.; Furber, S. Chain: A delay-insensitive chip area interconnect. *IEEE Micro* **2002**, *22*, 16–23.
13. Zhao, J.; Madduri, S.; Vadlamani, R.; Burleson, W.; Tessier, R. A Dedicated Monitoring Infrastructure for Multicore Processors. *IEEE Trans. Very Larg. Scale Integr. (VLSI) Syst.* **2011**, *19*, 1011–1022.
14. Datta, B.; Burleson, W. Low-power process-variation tolerant on-chip thermal monitoring using track and hold based thermal sensors. In Proceedings of the 19th ACM Great Lakes symposium on VLSI, Boston Area, MA, USA, 10–12 May 2009.
15. McGowen, R.; Poirier, C.; Bostak, C.; Ignowski, J.; Millican, M.; Parks, W.; Naffziger, S. Power and temperature control on a 90 nm Itanium family processor. *J. Solid State Circ. (JSSC)* **2006**, *41*, 229–237.
16. Miro-Panades, I.; Beigné, E.; Thonnart, Y.; Alacoque, L.; Vivet, P.; Lesecq, S.; Puschini, D.; Molnos, A.; Thabet, F.; Tain, B.; et al. A Fine-Grain Variation-Aware Dynamic Vdd-Hopping AVFS Architecture on a 32 nm GALS MPSoC. *J. Solid State Circ. (JSSC)* **2014**, *49*, 1475–1486.
17. Vincent, L.; Maurine, P.; Lesecq, S.; Beigne, E. Embedding statistical tests for on-chip dynamic voltage and temperature monitoring. In Proceedings of the 49th Annual Design Automation Conference (DAC '12), New York, NY, USA, 3–7 June 2012 .
18. Bachmann, C.; van Schaik, G.-J.; Busze, B.; Konijnenburg, M.; Zhang, Y.; Stuyt, J.; Ashouei, M.; Dolmans, G.; Gemmeke, T.; de Groot, H. 10.6 A 0.74 V 200 uW multi-standard transceiver digital baseband in 40 nm LP-CMOS for 2.4GHz Bluetooth Smart/ZigBee/IEEE 802.15.6 personal area networks. In Proceedings of the IEEE International Solid-State Circuits Conference Digest of Technical Papers (ISSCC), San Francisco, CA, USA, 9–13 February 2014.
19. Didioui, A.; Bernier, C.; Tran, L.Q.V.; Sentieys, O. EnvAdapt: An Energy-Aware Simulation Framework for Power-Scalable Transceivers for Wireless Sensor Networks. In Proceedings of the 20th European Wireless Conference, Barcelona, Spain, 14–16 May 2014.
20. Akgul, Y.; Puschini, D.; Lesecq, S.; Beigné, E.; Miro-Panades, I.; Benoit, P.; Torres, L. Power management through DVFS and dynamic body biasing in FD-SOI circuits. In Proceedings of the 2014 51st ACM/EDAC/IEEE Design Automation Conference (DAC), San Francisco, CA, USA, 1–5 June 2014.
21. Thonnart, Y.; Beigne, E.; Valentian, A.; Vivet, P. Automatic Power Regulation Based on an Asynchronous Activity Detection and its Application to ANOC Node Leakage Reduction. In Proceedings of the 14th IEEE International Symposium on Asynchronous Circuits and Systems (ASYNC), Newcastle upon Tyne, UK, 7–10 April 2008.
22. Beigne, E.; Vivet, P.; Thonnart, Y.; Christmann, J.F.; Clermidy, F. Asynchronous Circuit Designs for the Internet of Everything: A Methodology for Ultralow-Power Circuits with GALS Architecture. *IEEE Solid-State Circ. Mag.* **2016**, *8*, 39–47.
23. Cummings, C.E. Clock Domain Crossing (CDC) Design & Verification Techniques Using System Verilog. Available online: http://www.sunburst-design.com/papers/CummingsSNUG2008Boston_CDC.pdf (accessed on 6 May 2017).
24. Sparso, J. *Principles of Asynchronous Circuit Design—A Systems Perspective*; Sparso, J., Furber, S., Eds.; Kluwer Academic Publishers: Dordrecht, The Netherlands , 2001.
25. Magarshack, P.; Flatresse, P.; Cesana, G. UTBB FD-SOI: A process/design symbiosis for breakthrough energy-efficiency. In Proceedings of the Design, Automation & Test in Europe Conference & Exhibition, Grenoble, France, 18–22 March 2013; pp. 952–957.

© 2017 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (http://creativecommons.org/licenses/by/4.0/).