A Scalable System Bus for Low-end Performance SoC

JUAN PABLO ROMERO GALINDO

UNIVERSIDAD INDUSTRIAL DE SANTANDER
FACULTAD DE INGENIERÍAS FISICOMECÁNICAS
ESCUELA DE INGENIERÍA ELÉCTRICA, ELECTRÓNICA Y DE TELECOMUNICACIONES
BUCARAMANGA
2020

A Scalable System Bus for Low-end Performance SoC

JUAN PABLO ROMERO GALINDO

Trabajo de grado presentado como requisito para optar al título de Magister en Ingeniería de Electrónica

Director
ÉLKIM FELIPE ROA FUENTES
INGENIERO ELECTRICISTA. PhD.

UNIVERSIDAD INDUSTRIAL DE SANTANDER
FACULTAD DE INGENIERÍAS FISICOMECÁNICAS
ESCUELA DE INGENIERÍA ELÉCTRICA, ELECTRÓNICA Y DE TELECOMUNICACIONES
BUCARAMANGA
2020

Acknowledgments

In this section, all the people and institutions involved in the realization of the project will be mentioned.

- My family.
  For accompanying and supporting me throughout the study process.
- Élkim Roa.
  Direction of the work and director of the integrated systems research group Onchip. Research group where this project is developed. Contributions in the revision and edition of this document
- Universidad Industrial de Santander (UIS) and Integrated Systems Research Group (Onchip).
  For providing the software tools and equipment necessary for the execution of this project.
- All Onchip team that was part of the digital and analog design of the microcontroller project.
- Miyi Torres.
  For your review and edition of this document.

Content

LIST OF TABLES

LIST OF FIGURES

Resumen

TÍTULO: Bus de Periférico y de Sistema Energéticamente Eficiente para Aplicaciones de SoC de Bajo Consumo. *

AUTOR: Juan Pablo Romero Galindo †

PALABRAS CLAVES: Bus, sistemas en chip, bus de sistema, comunicación dentro del bus.

DESCRIPCIÓN:

Hoy en día, un SoC integra una gran cantidad de módulos dentro de un único circuito integrado, por lo cual es necesario implementar un sistema de comunicación robusto para comunicar cada componente del chip. Los buses son una solución conveniente para la conexión entre los módulos, arbitrar la comunicación y controlar el tiempo en el que se transfiere la información a lo largo del SoC. Aunque el bus es un componente esencial en las aplicaciones de SoC, Hay una falta de literatura que especifique problemas relacionados a este tema. Este trabajo destaca los problemas de tiempo, medido en ciclos de reloj, relacionados con la comunicación ineficiente entre un maestro y un periférico en propuestas de buses estándar. Este trabajo presenta un protocolo de bus alternativo que permite la comunicación directa entre maestros y esclavos vinculados al dominio de bus de periféricos y de sistema, en aplicaciones de bajo consumo de energía. Como resultado de implementar el bus propuesto dentro de un SoC, se presenta una reducción de 5 veces el número de ciclos gastados para realizar transacciones múltiples en comparación con otras propuestas como TileLink y AHB-Lite / APB.

---

*Trabajo de Investigación.

†Facultad de Ingenierías Fisicomecánicas. Escuela de Ingenierías Eléctrica, Electrónica y de Telecomunicaciones. Director: Élkim Felipe Roa Fuentes.

Abstract

TITLE: A Scalable System Bus for Low-end Performance SoC [‡]

AUTHOR: Juan Pablo Romero Galindo [§]

KEYWORDS: Buses,

DESCRIPTION: system-on-a-chip, scalable buses, peripheral bus, low power on-chip communication

Nowadays, an SoC integrates a large number of modules within a single die, which requires implementing a robust communication system to link the whole chip. Buses are a convenient solution for the connection of modules, arbitrating communication, timing, and transferring information along the SoC. Although the bus is an essential component in SoC applications, there is a lack of accurate literature about the topic. This paper spotlights the energy issues related to inefficient communication between a master and a time-constrained peripheral in standard bus approaches. Here we introduce an alternative bus protocol to allow direct communication among masters and slaves linked to the peripheral and system domains in low-energy applications. As a result of implementing the proposed bus within an SoC, we present a 5X clock cycle reduction for multiple transactions when compared to TileLink and AHB-Lite/APB approaches.

[‡] Research Work

[§] Facultad de Ingenierías Fisicomecánicas. Escuela de Ingenierías Eléctrica, Electrónica y de Telecomunicaciones. Advisor: Élkim Felipe Roa Fuentes.

INTRODUCTION

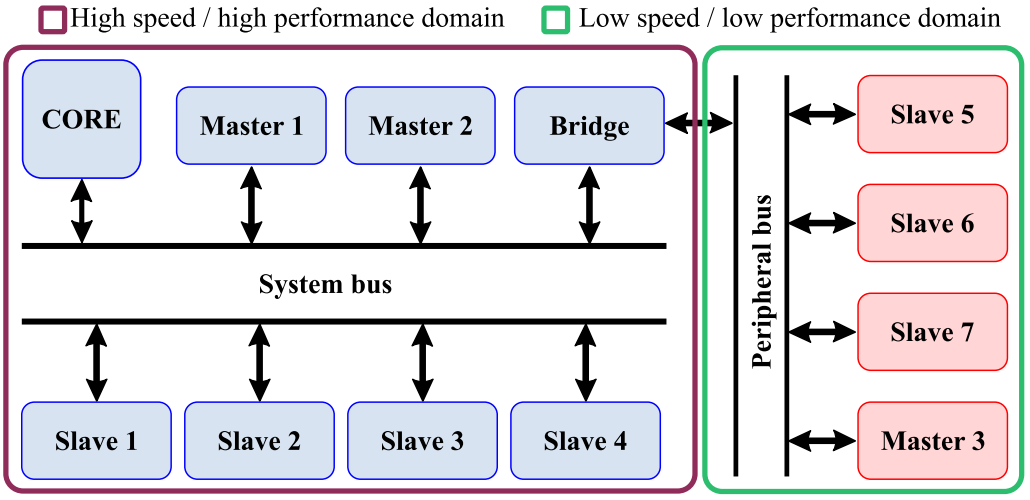
Several technological devices have become indispensable on a daily basis. As a result, the market has reacted by offering a wide variety of *gadgets* with fast generational renewal. Now companies want to quickly launch products, to be able to stand out in the market. To meet demand, the integrated circuit industry has had to reduce the design time and production cost.

Processors, microcontrollers, and systems on a chip (SoCs) are essential parts of consumer electronics. These devices are generally structured by several subsystems, such as cores, digital, analog, or mixed-signal peripherals, and one or more modules in charge of the on-chip network communication.

The increasing number of subsystems integrated into a modern SoC entails the need for implementing energy-efficient and high-yield on-chip communication links. Currently, the most common approach to interface the modules of an SoC relies on bus interconnects, as depicted in Fig. 1. The block diagram shows a common distribution of an SoC, highlighting two main performance domains controlled by a specialized bus, adapted according to the requirements of each domain. As illustrates, the main aim of the peripheral and system buses is to manage the on-chip communication. Therefore, the buses' performance directly influences several relevant features of an SoC such as area, power consumption, and speed (4).

Despite the relevance of buses regarding the yield of an SoC, literature is limited in real approaches and reported works within this field. A small number of reported works are associated with real implementations that show complete interaction among masters and slaves, as well as issues and limitations regarding

Figure 1. Simplified block diagram of an SoC, using two different buses to manage communication.



system communication. The lack of available documentation is mainly associated with the extensive use of commercial AMBA buses from Arm. These regular buses implementations include protocol for high, medium and low performance applications. AXI has been widely employed for high-speed interconnections and peripherals, given its capability to make read and write transactions at the same time (5). Another AMBA bus example is AHB, which is a bus interface suitable for high-performance system designs with moderate power consumption. Finally, APB of AMBA also appears within the most common Arm buses, and is defined as a low-cost interface optimized for minimal power consumption and low interface complexity. Besides licensing problems of the AMBA buses, their specification constrains some applications that require direct access to peripherals (6).

Existent bus-based approaches for low-power applications exhibit some shortcomings for direct peripheral communication. For instance, Fig. 5 depicts a common SoC structure using AHB-Lite as the system bus and APB as the peripheral bus. In the example, it is not possible for a slave connected to APB to start a transaction with a slave connected to the AHB-Lite. This constraint appears

due to the definition of the APB protocol itself. APB bus limitations include few control signals to manage multi-master transactions, poor versatility regarding operating modes and 3 states within its handshake in contrast to 2 states of AHB. Specifically, the states-number constraint leads to the development of elaborated synchronization interfaces between APB as a master and AHB as a slave, thus complicating the implementation and deteriorating the system efficiency. This situation may occur when an ADC needs to send data directly to RAM, ROM, or an output port, leading to an undesired feature that limits energy saving and speed of SoCs.

This work introduces the scalable system bus approach as an efficient solution to enhance interconnect protocols. Instead of interconnecting bridges, a scalable system bus offers a smooth data translation between a master placed on the secondary bus and a slave on the system bus. Compared to commercial and open standard interconnects, the proposed scheme performs with a fivefold clock cycle efficiency. Moreover, this work contrasts three verification approaches for the bus and finally shows supplementary contributions realized all along the time of the master.

1. PROBLEM STATEMENT

There is a wide variety of processors and SoCs, which can be categorized according to their performance level. These can range from high-performance features such as server processors to energy-efficient applications such as the Internet of Things (IoT). This last group of applications is the interest focus of this project.

The bus within an SoC is in charge of communicating the processing circuits with the peripherals. For that, it is necessary to implement a bus in SoC design. Some of the main aspects to take into account for the employment of a bus are the communication protocol to implement and his architecture.

The problem for which there are few open-source bus proposals and literature reports at hand resides in the difficulty of having fully tested processors available to connect a bus. This leads to complications to verify the proposed buses together a whole system (processor-bus-peripherals), which leads to unreliable proposals. Thanks to the proposed RISC-V instruction set by Andrew Waterman and Krste Asanovic (7), the difficulty of having a processor has been greatly reduced. Now, it is possible to implement processors open-source based on the RISC-V instruction set, removing the barrier of having agreements with large processor design companies such as ARM or Intel, to have a processor to connect the bus.

This research shows a study of the buses that are currently available, differentiating the advantages and characteristics of the buses used in low-consumption applications. Besides, as a result of the research, this report shows a proposal of a bus system, which includes peripheral and system bus, characterized for his

scalability, compatibility with processors based on the RISC-V, easy operation, and low power consumption.

Finally, The bus behavior was validate for 3 methods; first, performing multiple simulations using the Cadence simulation tool (SimVision) and exposing it to diverse operating conditions that verify its correct operation in the different scenarios where it can be compromised, second, implementing in an FPGA the proposed bus within of an SoC, third, a first approach to formal verification.

1.1. CURRENT PROPOSED BUSES

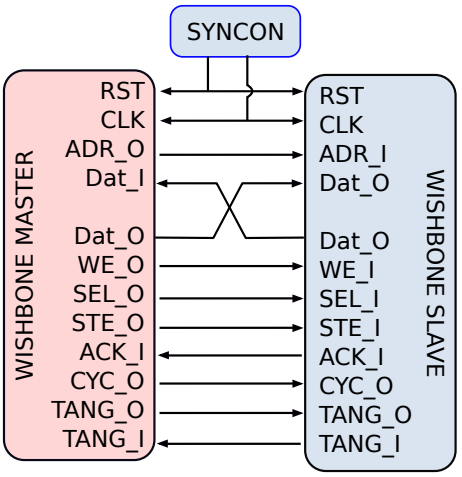
Using a standard bus is important in order to increases the probability of success when a project is integrated for the first time. Design or integrate a circuit together with another intellectual property (IP) module or facilitate the integration of macro projects. In addition, a standard bus ensures performance under various operating conditions and ease modular component design in an SoC (8).

1.1.1. Wishbone The Wishbone architecture offers flexible integration and easy adaptation for specific applications. This was developed as an interface that facilitates structured design methodologies in projects with large numbers of people.

The most outstanding feature that Wishbone offers is the data size modularization of the bus, different interconnection modes, and the support of transfers in a two clock cycle. Figure 2 shows one of the types of interconnection described in the specifications given by Wishbone (1).

1.1.2. On-Chip Peripheral Bus On-Chip Peripheral Bus (OPB) developed by IBM, is a bus designed to connect peripherals, describing a comfortable point-to-

Figure 2. Basic Wishbone architecture. *Figure adapted from* (1).



point communication for on-chip peripherals. OPB implements the management of different levels of bus hierarchy. Low power or low performance peripherals (such as clocks, meters, slaves, and other internal peripherals) are linked on the OPB bus. A bridge is provided between the processor and the local bus (PLB) and OPB to enable the transfer of data from the PLB master to and from OPB slaves. OPB also provides support for different types of peripherals, these can be 8-bit, 16-bit and 32-bit. Transfers between slaves and masters can take at least two clock cycles. Furthermore, OPB supports multiple OPB master buses and protocols with sequential addressing (2).

Figure 3 shows the architecture proposed by IMB, through its OPB bus.

1.1.3. STBus STBus is a set of protocols, interfaces, and architectures developed by STMicroelectronics, which specify the interconnection of subsystems. This offers versatility in configuration based on performance, architecture, and implementation.

STBus is a bus dedicated to consumer microcontrollers. Currently, STBus is not only a characterization of protocols and interfaces, but it is also focused on

Figure 3. Basic OPB architecture. *Figure adapted from* (2)

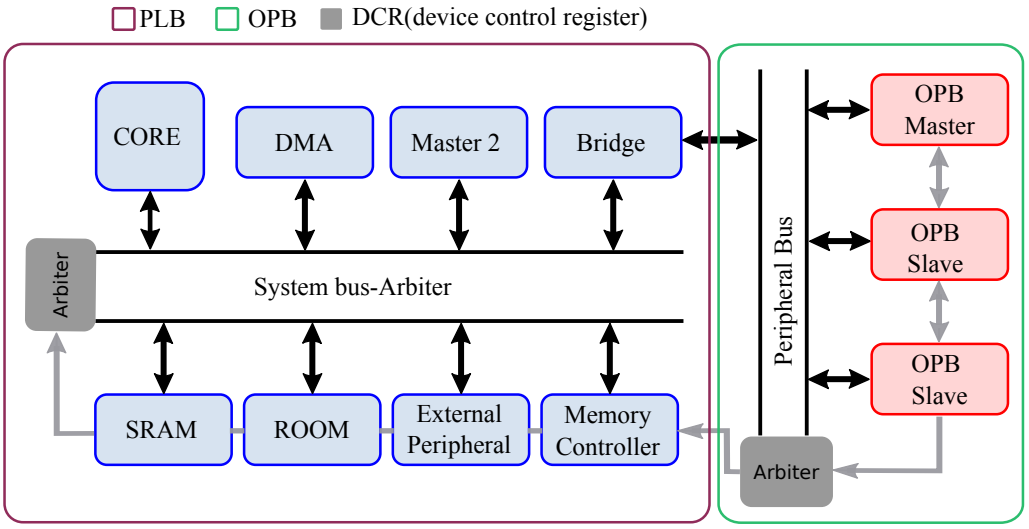
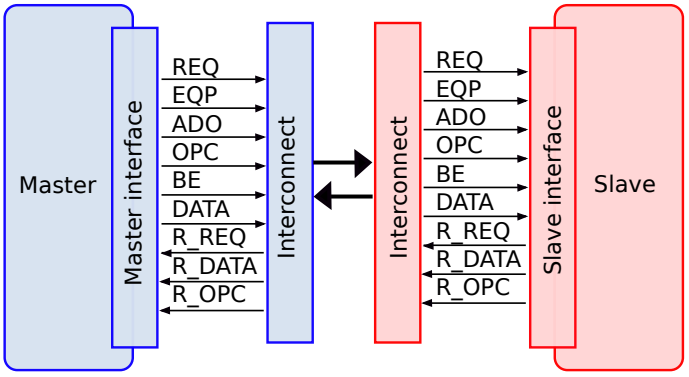


Figure 4. Basic STBus architecture. *Figure adapted from* (3).



allowing the development of ecosystems that include tools at the level of design and exploration of architectures, design in silicon, physical implementation and verification (3). Figure 4 shows the main signals used in the STBus protocol, together with a typical bus configuration described within its specifications.

1.1.4. MBUS MBus is an ultra low power system, originally designed by the University of Michigan. The Mbus interconnect is designed for general-purpose interfaces. The description of its protocol covers a wide functionality of IP cores. Among its main features are the support of transfer in two clock cycle, modulariza-

tion of data size, interrupt vector, control of cache operations, support of various types of connection between IP cores, such as point-to-point and interconnection by bus medium (9).

1.1.5. Tilelink TileLink is a scalable standard based on the management of priority channels. This approach provides a connection between multiple masters with coherent memory-mapped access to memory and other slave devices, as well as low-latency and high-throughput transfers. The TileLink protocol is designed mainly for its use in SoCs, as it enables the interconnection of general-purpose multiprocessors and modules with different levels of complexity. However, it was developed mainly for medium and high-performance systems, cache management, and memory transactions, overlooking low-power and low-area applications.

In addition to prevalent commercial Arm implementations, TileLink, an alternative bus specification proposed by U. C. Berkeley and SiFive, is gaining academic attraction. This protocol is structured as an interconnector of three different levels of links. Each link has different privileges, including basic read and write memory transactions and the handling of cache blocks and special operations. Within the three levels of links, two of them are appropriate for low-power applications: TileLink Uncached Lightweight (TL-UL) and TileLink Uncached Heavyweight (TL-UH). TL-UL is the simplest level of TileLink, supporting only simple read and write memory operations without having the possibility to perform burst transactions. TL-UH inherits the capabilities of TL-UL and provides additional operations such as burst messages, atomic operations, and hint operations (10).

Table 1. Comparison of protocol buses for low-power applications

	Open Standard	**Number of Ch.**	**Burst Mode**	**Pipeline**	**Scalable**
AXI4-lite	No	5	No	No	No
AHB-Lite	No	3	Yes	Yes	No
APB	No	3	No	No	No
TileLink	Yes	5	Yes	Yes	Yes
This Work	**Yes**	**3**	**Yes**	**Yes**	**Yes**

1.2. SOME PROTOCOL BUSES FOR LOW-POWER APPLICATIONS

Different approaches have been proposed to manage on-chip communication. Arm, one of the world's leading suppliers of silicon IP and custom SoCs, offers AMBA family with different bus specifications for a wide variety of SoCs. Table 1 presents some of the most popular AMBA buses for low-power applications and summarizes some of their most important features.

AXI4-lite is an AMBA family common protocol. A relevant characteristic of AXI4-lite is having five independent communication channels that allow multiple outstanding transactions, and out-of-order completion of transactions. Nonetheless, since there are other bus proposals with fewer communication channels, the implementation of AXI4-lite may entail a larger final chip area for some applications. Another AXI4-lite protocol drawback is the lack of burst transaction specifications, which limits throughput, unlike other buses.

AHB-Lite is another one of the most employed protocol buses in low-power systems. This approach defines the interface among modules such as masters, slaves, and interconnects. Main features of the AHB-Lite include the support of burst transactions, locked sequences, and the implementation of a low level of protection. APB bus, defined within AMBA, is a low-cost peripheral interface optimized for low-power and low-complexity interfaces. Commonly, the AHB bus is

implemented next to the APB bus so that the AHB-Lite is responsible for handling some peripherals through the APB bus. However, due to differences in their handshakes, the interaction between these buses is not optimal, demanding the addition of a bridge to synchronize signals and protocols.

In the public domain, there are two interconnect standards: Wishbone (1) and TileLink (10). Wishbone establishes common standard interfaces for data exchange among modules within integrated circuits. Likewise, Wishbone became a popular protocol to communicate independent SoC projects. However, the last revision of this protocol, published ten years ago, lacks features to satisfy the current requirements of low-power SoCs.

Some previous works have preferred to look for alternative ways to improve communication within SoCs, avoiding proposing new buses. One of the mentioned works proposes the addition of a DMA block between peripherals and memory (11). However, the addition of a new module like a DMA involves increasing the complexity of the system and affects the final area of the chip negatively. Pullini *et al.* (12) presents an example of the described area issue, where the size of the DMA may exceed the total area occupied by the bus and the interconnect of the SoC.

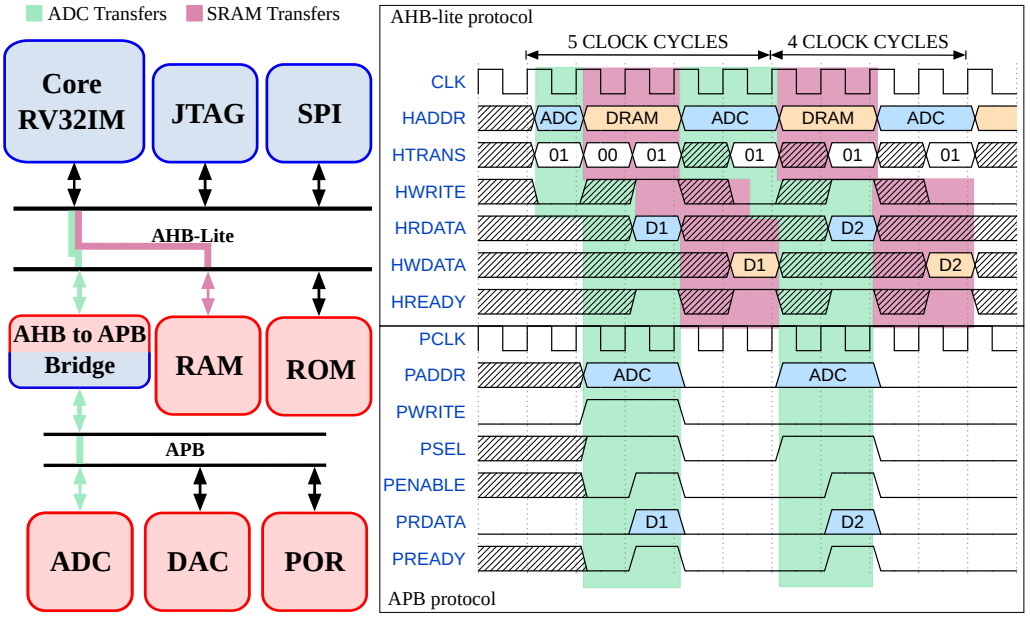
2. CONDOR AND COLIBRI: A PROPOSAL FOR SYSTEM AND PERIPHERAL BUSES

This chapter introduces a scalable bus scheme composed of a system bus and a peripheral bus, called *Condor* and *Colibri*, respectively. The proposed scalable bus system offers the possibility of communicating modules from the system bus with modules in the peripheral bus and vice versa, flexibility design, and low complex handshake. Also, here is described a particular bus problem in some current low-performance SoCs. AHB-lite and APB is used to exemplify the issue, for later provide a solution using Condor and Colibri. Then, is showed the synthesis of *Condor* and *Colibri* in a 180nm CMOS standard technology, and integrated within an SoC along with a RISC-V based core, a 2kB RAM and a 10-bit ADC. Finally, in order to compare the performance of the proposed bus with conventional approaches, we also assessed conventional TileLink and AHB-Lite & APB bus implementations. As a result, the proposed buses system exhibits a 5X clock cycle reduction when sending data from a peripheral module to another module at the system bus.

2.1. CURRENT BUS SCHEMES ISSUES

Most SoCs implement two or more buses to manage on-chip communication. Peripherals usually include a specialized bus that interconnects them as slaves to the system bus. Given its characteristics, the ADC is an example of a module that might be connected as a peripheral on the secondary bus. The ADC, as an active peripheral, continuously updates conversion data at a constant rate and sends it to the SRAM memory or to an output port. Existent bus schemes for low-

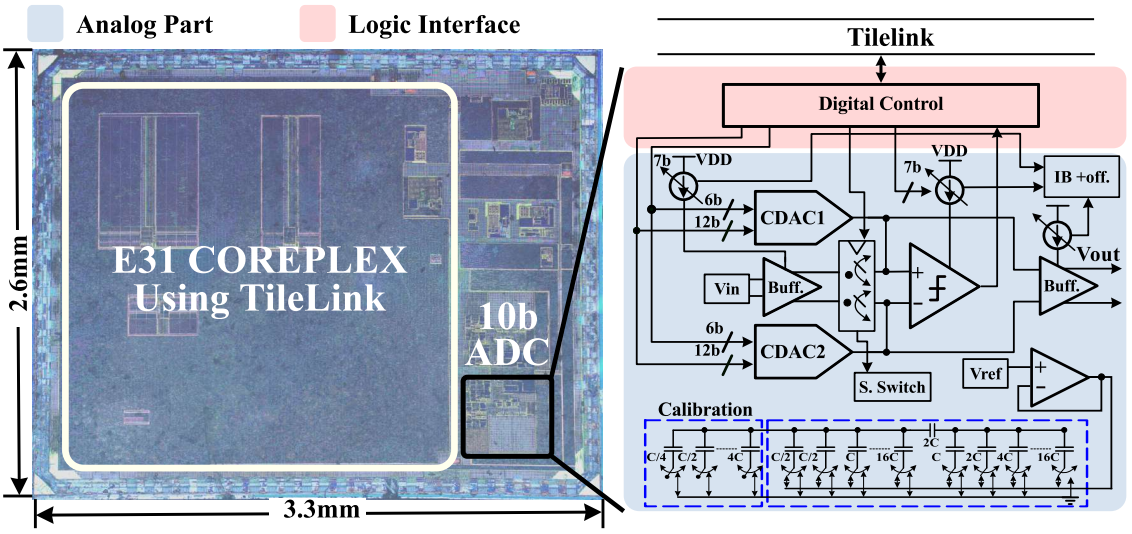
Figure 5. ADC transactions to RAM using APB and AHB-Lite protocols.



power consumption perform the ADC-SRAM transaction inefficiently. Figure 5 shows an SoC implementing conventional interconnection through AHB-Lite and APB buses. Using this configuration entails spending two transactions to send a packet of data from ADC to RAM. Regarding performance, five clock cycles are required. At the beginning of the transaction, the core is a master, starts the transaction and requests data from the ADC, spending one clock cycle. After this operation, two clock cycles are needed to reach the APB access state status. Finally, the core sends the data to the RAM within two clock cycles, using five clock cycles in total. In subsequent transactions, the clock cycles are reduced to four due to the AHB-Lite pipeline. The timing diagram in Figure 5 plots the procedure mentioned above.

Figure 6 shows the implementation of a 10-bit ADC within a full SoC in a 180nm CMOS standard technology. Through the implementation of this peripheral, we realized that maintaining constant communication between peripherals

Figure 6. 10-bit ADC block diagram and die micrograph.



and the SRAM does not require the inclusion of a DMA block into the system. This approach avoids the final silicon area increment disclosed in (12). Additionally, as data processing is not necessary for applications of only data acquisition, it is possible to disable the core for these transactions. Therefore, having direct memory communication through the bus allows the core to turn off, reducing power consumption in applications where data sampling performs in sleep mode. In this operating state, the SoC might perform simple tasks such as periodically carrying out conversions of the ADC and storing them in the SRAM blocks.

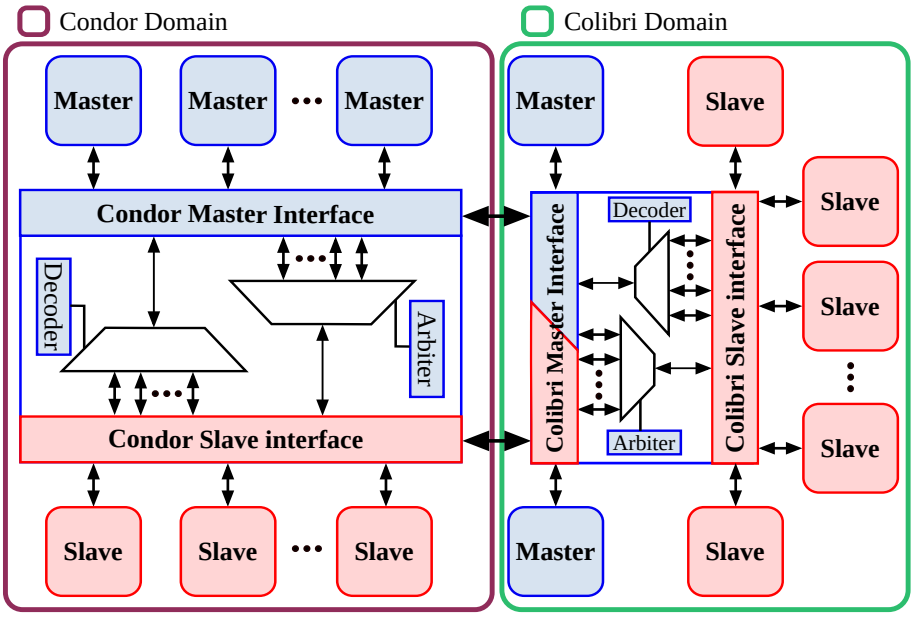
2.2. CONDOR AND COLIBRI

This work introduces an alternative solution for communication among masters connected to the peripheral bus and slaves linked to the system bus. Figure 7 depicts a simplified block diagram of our scalable system bus. *Condor* is the name of the system bus, which controls the peripherals that require high-bandwidth operation. *Colibri* is a scaled version of the *Condor* bus, which uses a minimum

number of *Condor* signals to establish communication and complete the handshake with the system bus. *Condor* is also the bus in charge of communicating the low-bandwidth peripherals. Since the two buses have similar protocols and handshakes, it is possible to establish a direct exchange of transactions between *Colibri* and *Condor* buses. As shown in Figure 7, *Colibri* has one master interface that performs two functions. The first function is working as a slave of *Condor* and as a master for *Colibri*'s slaves and the second function is being responsible for connecting the *Colibri*'s masters to *Condor* master interface. This interface is capable of supporting multi-master transactions through the addition of a basic arbiter. Relevant features of the proposed buses such as the critical path or maximum frequency are limited, as normally occurs in conventional approaches, by the hardware description performed. Additionally, as *Colibri* and *Condor* are independent buses, they can work with different clock frequencies by adding a clock synchronizer between them. For instance, if Colibri has a lower clock speed than Condor, the behavior is the same, adding wait states within each read or write transaction. Common applications have peripheral buses running on lower speed clocks seeking for energy savings and better peripheral performance.

Figure 8 shows the list of signals in the proposed buses system (*Condor* in red and *Colibri* in blue). The signal naming convention follows AMBA specification in order to maintain the correspondence. *Condor* comprises more signals because it is responsible for several functions such as configuring the transactions' size, adding waiting states to previously initialized transactions or selecting among non-sequential, sequential and busy operating modes. On the other hand, *Colibri* has two different structures, the first one is shown on the left side of Figure 8, defined with a minimum number of signals to perform basic read and write transactions with low-performance peripherals. The second structure is presented on the right

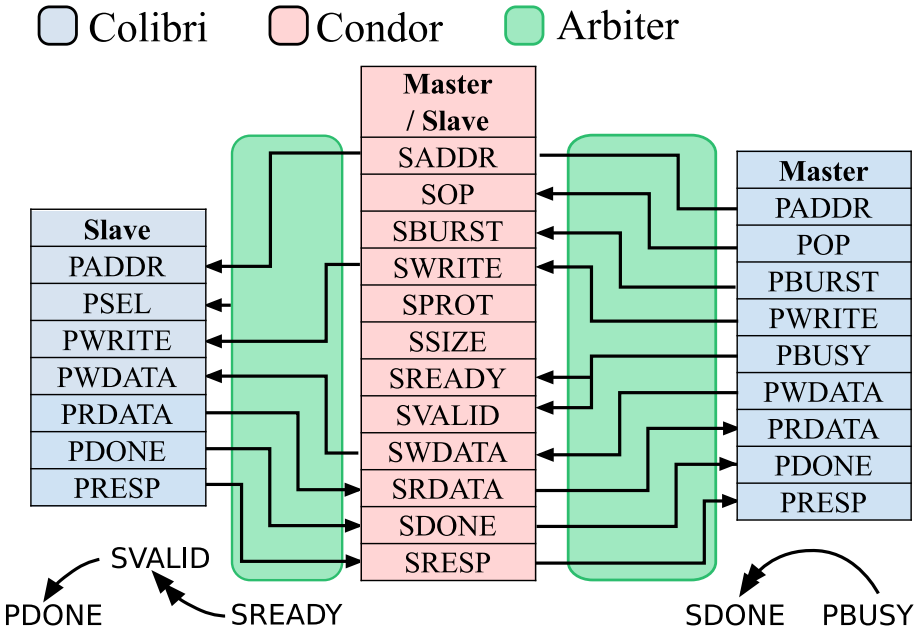
Figure 7. Connections of proposed bus scheme between the system and the peripheral bus.



side of Figure 8. In this case, the signals and the protocol are designed to connect masters in the peripheral bus, which require sequential or non-sequential transactions and being a master of *Colibri* or *Condor* buses. Both the *Colibri* protocol and its signals list are based on the *Condor* bus, enabling more fluid and efficient communication among buses. At the bottom, Figure 8 also shows the transaction handshake signal dependencies. In the case of *Colibri* slaves, *SVALID* and *SREADY* must assert the signals before changing *SDONE* to indicate that valid data is available. *SDONE* may also indicate wait state transactions. Regarding *Colibri* masters, it must wait for *PBUSY* to assert before changing *SDONE*.

Figure 10 provides a description of the arbitration implemented in *Condor* and *Colibri* buses. The priority designation for each master is the main advantage of the proposed buses approach over conventional implementations. This priority is related to the bits assigned to the *Priority_M signal*. *Priority_M* is the result of the master concatenation *Sready* signal. The LSB bit is assigned to the master

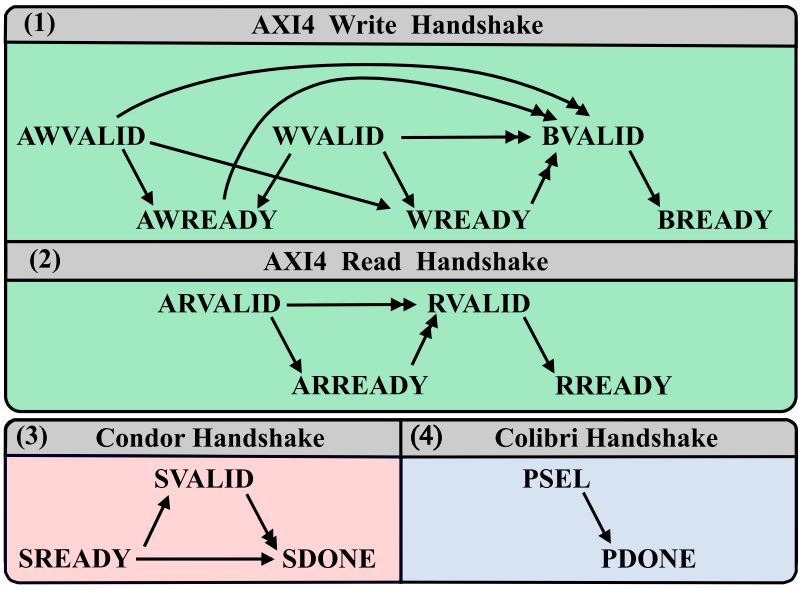
Figure 8. List of signals, handshake and scalability of the proposed buses system.



with the lowest priority and the MSB to the master with the highest priority. In the diagram of Figure 10, *Master 1* was assigned to the MSB and had the highest priority, in contrast to the *Master n*, which has the lowest priority. When a new master transaction is requested, the *Priority_M* signal changes. Once the change is detected, the first hot bit of *Priority_M* associated to a master will take control of the bus. The change of the current master will follow after *SDONE* slave signal asserts. As the arbiter only adds one clock cycle delay for each change of master, its contribution to the total number of clock cycles per transaction is negligible. However, in multiple and individual transactions, an additional clock cycle implies a higher impact on the total number of cycles per transaction.

The handshake is another relevant feature when assessing the performance of a buses approach. For comparison purposes, Figure 9 exhibits a signaling handshake representation for AXI4 and the proposed buses *Condor* and *Colibri*. The illustration shows the control signals among masters and slaves to establish

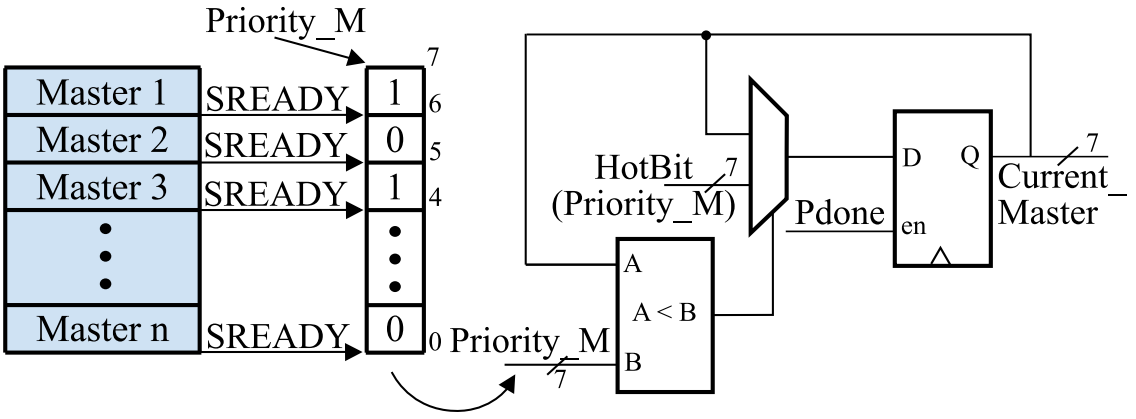
Figure 9. (1) AXI4 write transaction handshake dependencies. (2) AXI4 read transaction handshake dependencies. (3) *Condor* transaction handshake dependencies. (4) *Colibri* transaction handshake dependencies.



communication, which is represented with arrows. Single-headed arrows point towards signals that may assert before or after the prior signal is asserted. In the double-headed case, the arrows point to signals that must settle only after the settlement of the previous signal. Figure 9 (1) summarizes the AXI4 write transaction handshake dependence, which has a considerable amount of signals to maintain control of all steps in write transactions. For read transactions, Figure 9 (2) depicts the AXI4 handshake signal dependence. In contrast, AHB has a simple handshake where its control signals just indicate the start and the end of a transaction. For *Condor* and *Colibri*, the transaction handshake dependence is even simpler as indicated in Figure 9 (3) and Figure 9 (4), respectively.

As a result of having a scalable bus managing on-chip communication, it is now possible to perform bidirectional communication between domains. Therefore, masters located in the *Colibri* domain can communicate directly through the bus with a slave in the *Condor* domain, as shown in Figure 7. For the particular

Figure 10. Master priority assignation of the bus arbiter.



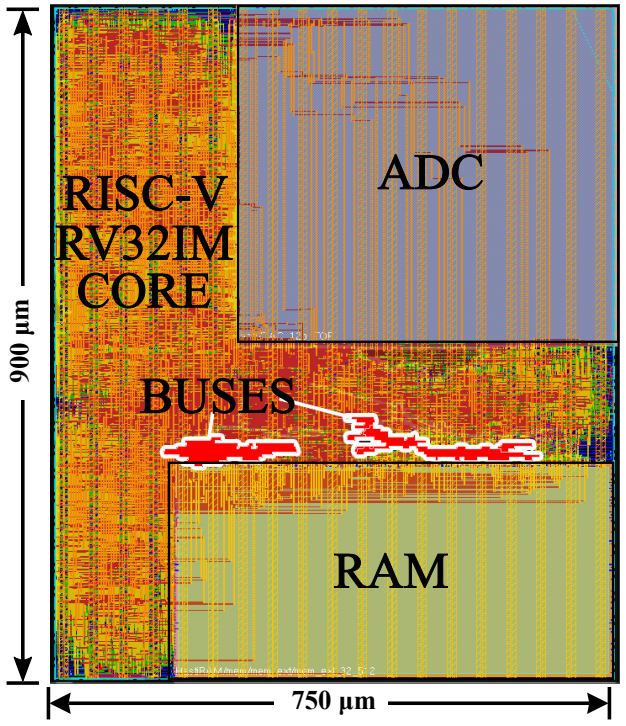
case of the ADC, it may be placed in the low-bandwidth domain and communicate directly through the bus with the SRAM on the main bus. The described strategy optimizes the information exchange, decreasing clock cycles spent in the transaction, and reducing dynamic power consumption.

2.3. SOC IMPLEMENTATION AND RESULTS

The proposed bus scheme described in Figure 7 was implemented and synthesized in a 180-nm CMOS standard technology node. To assess the performance of the proposed scheme, we implemented the SoC of Figure 11, composed of our system and peripheral buses. The SoC includes a 10-bit ADC capable of operating at up to 10MS/s, a 32-bit RV32IM RISC-V ISA core and a 2 kB RAM (7). The ADC connected to the *Colibri* bus is able to establish communication with any slave connected to the *Condor* bus.

Table 2 summarizes the contributions of each SoC block to the total area in Figure 11. The largest components of the system are the core, with a contribution of 41%, and the ADC, which occupies 32% of the area. The equivalent area of the buses is approximately 2% of the total chip area. Table 2 also offers a

Figure 11. SoC layout including a 32-bit RV32IM RISC-V ISA core, a RAM, an ADC and the proposed buses system.



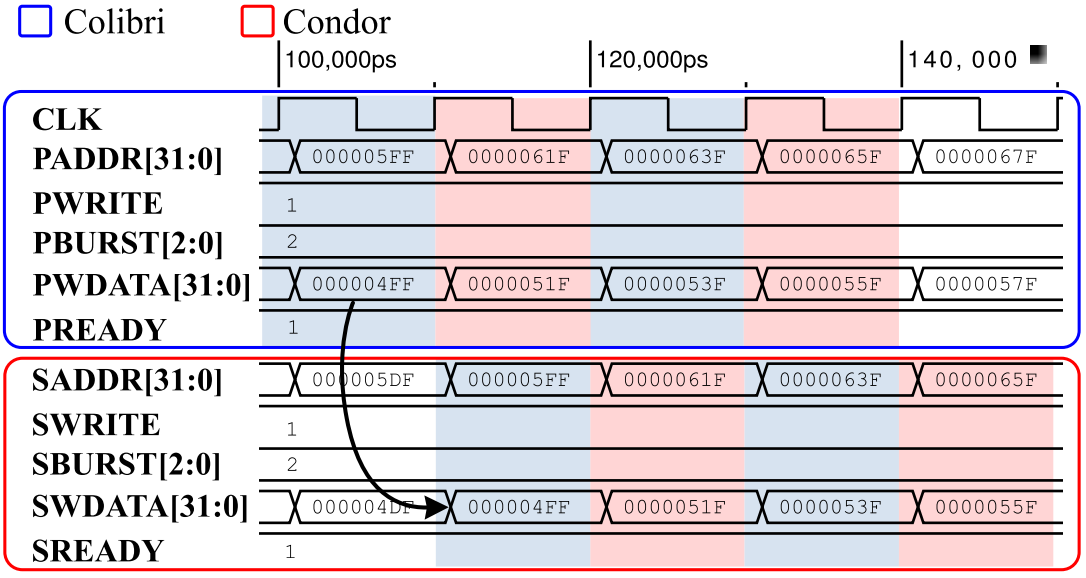
contrast concerning energy consumption, which allows appreciating that the proposed buses use an energy-per-cycle comparable to the RAM's or ADC's digital control consumption. This value is equivalent to just 4% of the total consumption of the SoC.

Figure 12 illustrates a segment of 100-burst-transactions test between *Colibri* and *Condor*. The diagram shows a transaction initialized from a master in *Colibri* (peripheral bus) toward a slave in *Condor* (system bus). Figure12 also shows the pipeline behavior of the communication system and some signals that share protocols. A successful reduction in the number of clock cycles and dynamic power required to complete the transaction can also be noticed. Contrasting with the conventional protocol represented in Figure 5, where AHB-Lite and APB buses execute four transactions to communicate an ADC with a RAM, the proposed

Table 2. Power, timing and area breakout of the SoC.

Core	**Energy** (pJ/cycle)	**Area** (μm^2)
RISC-V RV32IM	299.61	297976
RAM	9.56	179289
10bit ADC	8.71	230462
Colibri	0.19	583
Condor	14.30	8609
Total	332.07	721888

Figure 12. Burst mode timing diagram of the proposed buses.

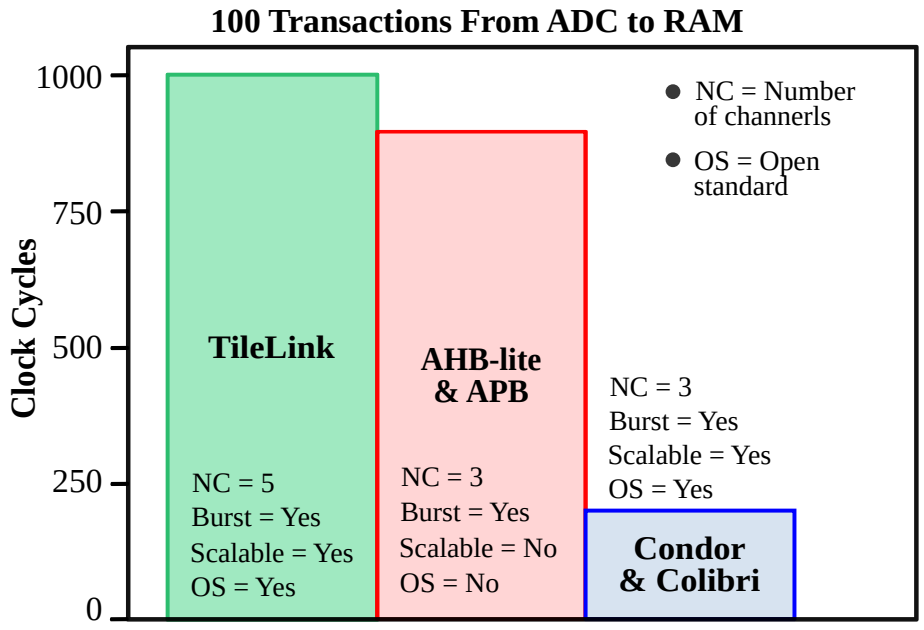


approach allows completing the same communication in just one transaction.

Due to implementation and application differences, comparing the proposed buses with other reported works in terms of energy and performance is problematic. Most related works focus mainly on processor-to-memory communication enhancement. Seeking for a fair comparison, both TileLink and AHB-Lite were integrated into SoCs with different sizes and features in contrast to the SoC presented in Figure 11, whereby we report simulation of multiple transactions using TileLink, AHB-Lite, and the proposed system. Each of them make transactions continuously in their corresponding protocols. Figure 13 summarizes the results

of a communication test implemented in different SoCs. We compared the number of clock cycles spent on sending a 100 data set from a slave connected to the peripheral bus, toward another slave linked to the system bus. The test set up consisted of performing 100 transactions, neglecting the clock cycles spent on loading the program into memory, and including the clock cycles employed in the initialization of the registers. Initially, we performed the TileLink measurements by using the E31 coreplex core (shown in Figure 6), which spent 1000 clock cycles to complete all of the transactions. We applied the same analysis on another SoC with AHB-Lite as the system bus and APB as the peripheral bus, which utilized 900 clock cycles in the process. Finally, to assess the proposed bus performance, we accomplished a simulation with continuous data in burst mode. As illustrated in Figure 12, we carried out 100 transactions between a master connected to *Colibri* bus and a slave attached to *Condor* bus. This test revealed the use of 202 clock cycles to complete the transactions, indicating that the proposed buses approach presents better speed and energy performance than TileLink or AHB-APB implementations.

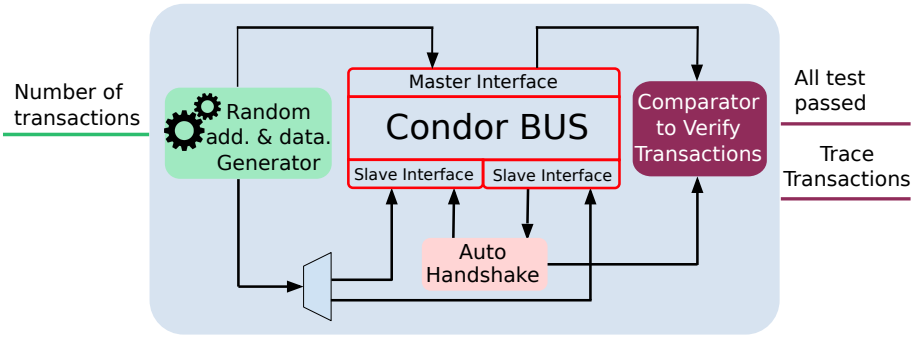
Figure 13. Clock cycles comparison between TileLink, AHB-Lite/APB and the proposed bus.



3. APPLIED BUS VERIFICATION TECHNIQUES

Up to this point, the thesis has presented Condor and Colibri buses. Now, this chapter describes the verification process performed for the proposed bus system. The design was described in Chisel 3 language, and this was synthesized to generate a Verilog file. The resulting Verilog was tested using different methodologies and test benches to check and cover most of the possible cases using System Verilog. In the first part of the chapter describes the simulation-based process, which is one of the most common verification practices. The second procedure presentes the FPGA verification, implementing the source code in a simple demonstration. Finally, the first approach of formal verification is described.

Figure 14. Simulation-based setup of Condor system bus.



3.1. SIMULATION-BASED VERIFICATION

Simulation-based is part of a common practice in hardware and software verification. It consists in defining a series of initial conditions and temporal stimuli to excite the design, emulating the correct behavior of the circuit in order to verify its accurate operation. Due to the complexity of the circuit, several test benches

Figure 15. Terminal result test of 100 Condor transactions.

```
At time t =            31300000. (Write transaction). Current value of the data was =0182cd17, the expected was 0182cd17. MATCH!
At time t =            31700000. (Write transaction). Current value of the data was =011be14e, the expected was 011be14e. MATCH!
At time t =            32100000.(Read transaction). Current value of the data was =026e5391, the expected was 026e5391. MATCH!
At time t =            32500000.(Read transaction). Current value of the data was =03a7777c, the expected was 03a7777c. MATCH!
At time t =            32900000. (Write transaction). Current value of the data was =033bf053, the expected was 033bf053. MATCH!
At time t =            33300000. (Write transaction). Current value of the data was =00346f94, the expected was 00346f94. MATCH!
At time t =            33700000. (Write transaction). Current value of the data was =002d9f64, the expected was 002d9f64. MATCH!
At time t =            34100000.(Read transaction). Current value of the data was =03d0d924, the expected was 03d0d924. MATCH!
At time t =            34500000. (Write transaction). Current value of the data was =022647d8, the expected was 022647d8. MATCH!
At time t =            34900000.(Read transaction). Current value of the data was =01a416bf, the expected was 01a416bf. MATCH!
At time t =            35200000.Transfer out of mapa range
At time t =            35500000.Transfer out of mapa range
At time t =            35900000. (Write transaction). Current value of the data was =006d182c, the expected was 006d182c. MATCH!
At time t =            36300000. (Write transaction). Current value of the data was =00d0975d, the expected was 00d0975d. MATCH!
At time t =            36700000.(Read transaction). Current value of the data was =00b0b52d, the expected was 00b0b52d. MATCH!
At time t =            37000000.Transfer out of mapa range
At time t =            37300000.Transfer out of mapa range
         100 transactions made;          53 write and          18 read transactions.
          29 Transactions out of slave address memory limits.
PASS, all transactions was performed.
```

were developed to verify the different functionalities of the bus and excite the circuit in various scenarios. The system bus was the first part of the proposal tested. Since the system bus is the most critical and essential part within the bus system, the test bench was carefully designed, with the purpose of covering a large number of cases, where the bus could be involved in a real implementation. A pseudo-random function was used to set the read or write transactions, decide the peripheral in which the communication will be executed, and establish the data sent in each transaction.

Figure 16. A sample timeline test of 100 Condor transactions.

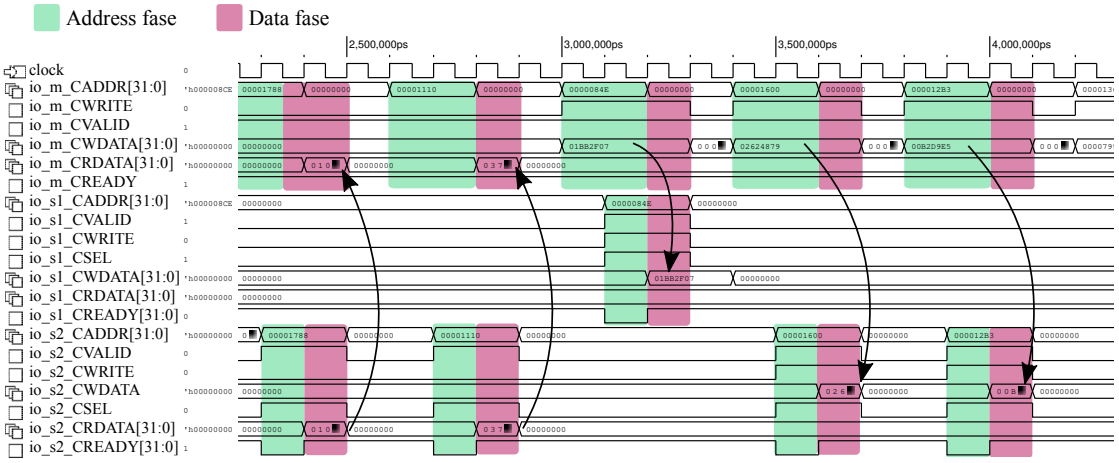
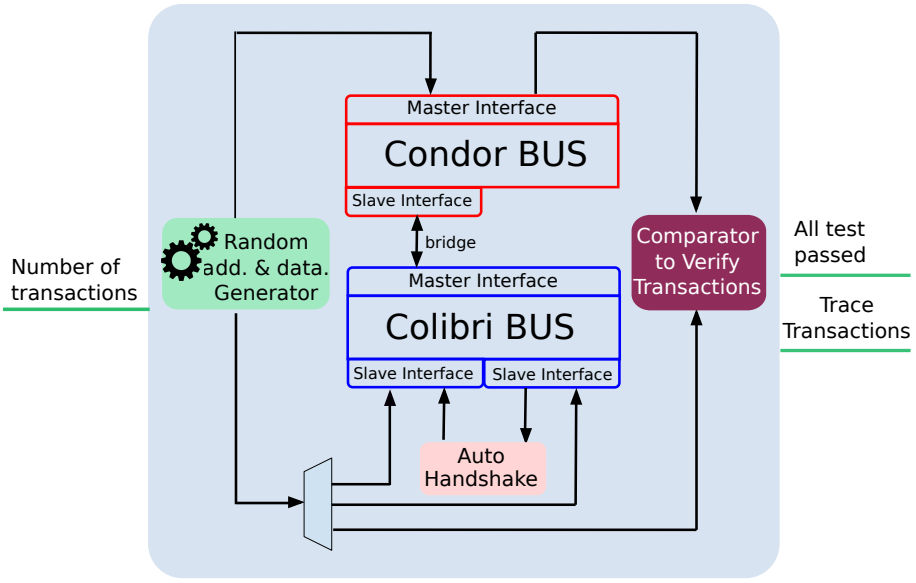


Figure 14 represents the setup of the testbench developed to test the Condor

Bus. Condor in Figure 14 is represented in red, and it is generated with 2 slaves and 1 master. The other blocks are tasks used to automatize repetitive processes like read and write transactions, generate new data to use in future operations, and check all transactions performed. The latter blocks are represented in rose, green and purple respectively in Figure 14. The transaction begins asserting CWRITE, CADDR, CRDATA, and CWDATA signals from the pseudo-random function, also the CREQ, CVALID, CSIZE, COP, and CBURST are defined as a part of one of the tasks detail in the test bench.

Figure 17. Simulation-based setup of Condor and Colibri system bus.

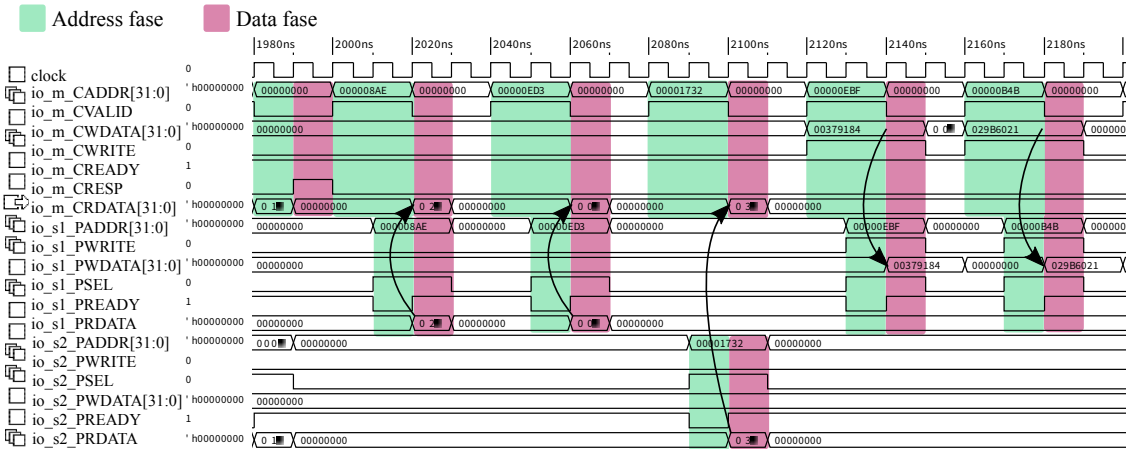


Once started the transactions, the arbiter of the bus selects the correct slave, and establishes the communication between the simulated master and slave. Subsequently, the auto-handshake function, responds with CRESP, CRDTA, and CREADY signals following the protocol. Finally, the comparator checks if the transaction was successful and reports the total number of transactions performed, the number of read or write transactions, out memory bus map transactions, malfunction cases of the bus, and the time and type of transaction in which

Figure 18. Terminal result test of 100 Condor and Colibri transactions.

```
At time t =            31000000.Transfer out of mapa range
At time t =            31400000.(Read transaction). Current value of the data was =006b2944, the expected was 006b2944. MATCH!
At time t =            31800000. (Write transaction). Current value of the data was =01419a8e, the expected was 01419a8e. MATCH!
At time t =            32200000.(Read transaction). Current value of the data was =021e7cf8, the expected was 021e7cf8. MATCH!
At time t =            32600000. (Write transaction). Current value of the data was =0133f64e, the expected was 0133f64e. MATCH!
At time t =            33000000. (Write transaction). Current value of the data was =028952e1, the expected was 028952e1. MATCH!
At time t =            33400000. (Write transaction). Current value of the data was =0167cc67, the expected was 0167cc67. MATCH!
At time t =            33800000. (Write transaction). Current value of the data was =02f43c12, the expected was 02f43c12. MATCH!
At time t =            34100000.Transfer out of mapa range
At time t =            34500000.(Read transaction). Current value of the data was =0011249c, the expected was 0011249c. MATCH!
At time t =            34900000. (Write transaction). Current value of the data was =00df1ced, the expected was 00df1ced. MATCH!
At time t =            35300000.(Read transaction). Current value of the data was =0087e37d, the expected was 0087e37d. MATCH!
At time t =            35600000.Transfer out of mapa range
At time t =            36000000. (Write transaction). Current value of the data was =00c5cad5, the expected was 00c5cad5. MATCH!
At time t =            36400000.(Read transaction). Current value of the data was =0077f4a5, the expected was 0077f4a5. MATCH!
At time t =            36800000.(Read transaction). Current value of the data was =002037b4, the expected was 002037b4. MATCH!
At time t =            37200000.(Read transaction). Current value of the data was =011ae6f6, the expected was 011ae6f6. MATCH!
At time t =            37500000.Transfer out of mapa range
At time t =            37900000. (Write transaction). Current value of the data was =02b0ec02, the expected was 02b0ec02. MATCH!
At time t =            38300000. (Write transaction). Current value of the data was =039cad34, the expected was 039cad34. MATCH!
        100 transactions made;          47 write and          34 read transactions.
         19 Transactions out of slave address memory limits.
PASS, all transactions was performed.
```

Figure 19. A sample of timeline test of 100 Condor and Colibri transactions.



it had an incorrect function, as shown in Figure 15. Additionally, the timeline simulation was checked to review the correct function of the test bench performed and the handshake, as Figure 16 present.

Using the same methodology, two additional test benches were performed. One of them to test the Colibri bus working with the system bus, as shown in Figure 17.

The setup developed for the second test bench is composed of the Condor bus with one master and slave interface, the bridge in charge of connecting Condor and Colibri bus, and the Colibri bus with one master interface and two slaves as

Figure 20. Simulation-based setup of Condor with multi-master bus.

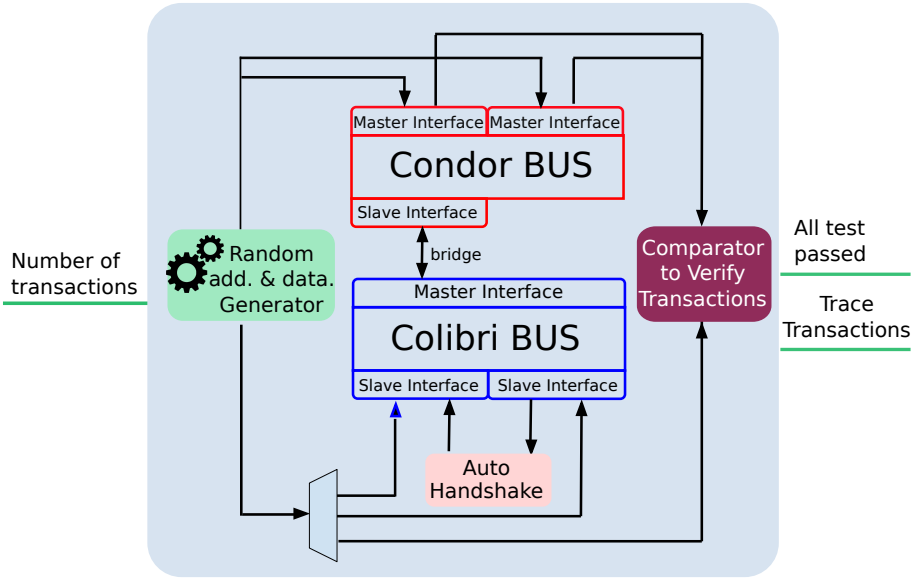


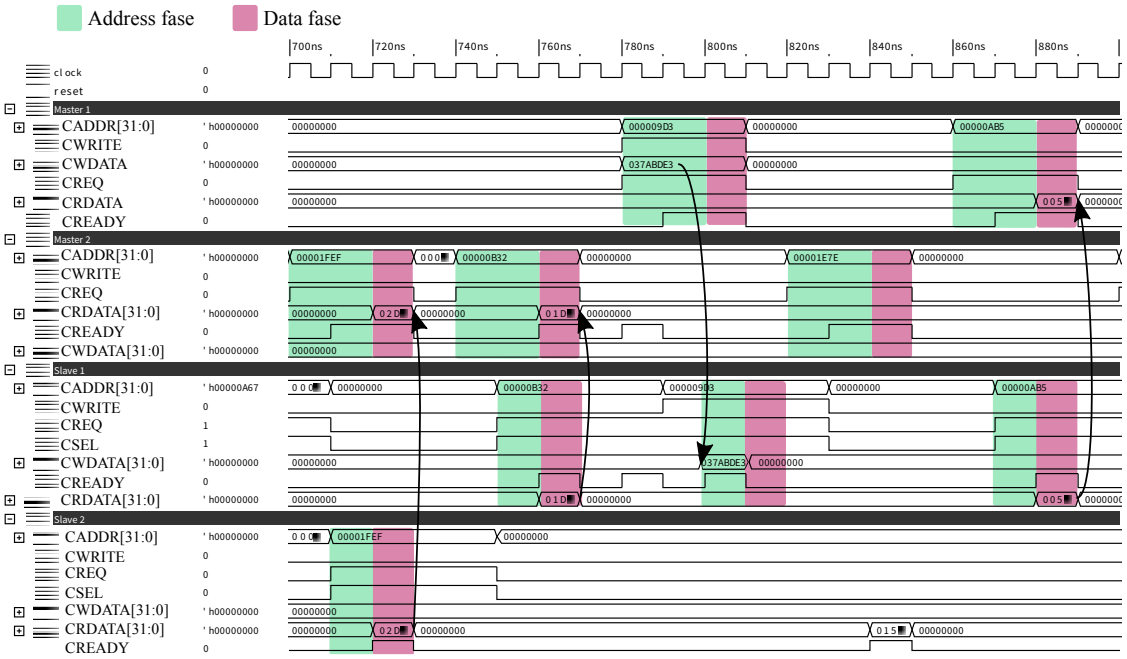
Figure 21. Terminal result test of 100 Condor transactions.

```
At time t =            31000000.Transfer out of mapa range
At time t =            31400000.(Read transaction). Current value of the data was =006b2944, the expected was 006b2944. MATCH!
At time t =            31800000. (Write transaction). Current value of the data was =01419a8e, the expected was 01419a8e. MATCH!
At time t =            32200000.(Read transaction). Current value of the data was =021e7cf8, the expected was 021e7cf8. MATCH!
At time t =            32600000. (Write transaction). Current value of the data was =0133f64e, the expected was 0133f64e. MATCH!
At time t =            33000000. (Write transaction). Current value of the data was =028952e1, the expected was 028952e1. MATCH!
At time t =            33400000. (Write transaction). Current value of the data was =0167cc67, the expected was 0167cc67. MATCH!
At time t =            33800000. (Write transaction). Current value of the data was =02f43c12, the expected was 02f43c12. MATCH!
At time t =            34100000.Transfer out of mapa range
At time t =            34500000.(Read transaction). Current value of the data was =0011249c, the expected was 0011249c. MATCH!
At time t =            34900000. (Write transaction). Current value of the data was =00df1ced, the expected was 00df1ced. MATCH!
At time t =            35300000.(Read transaction). Current value of the data was =0087e37d, the expected was 0087e37d. MATCH!
At time t =            35600000.Transfer out of mapa range
At time t =            36000000. (Write transaction). Current value of the data was =00c5cad5, the expected was 00c5cad5. MATCH!
At time t =            36400000.(Read transaction). Current value of the data was =0077f4a5, the expected was 0077f4a5. MATCH!
At time t =            36800000.(Read transaction). Current value of the data was =002037b4, the expected was 002037b4. MATCH!
At time t =            37200000.(Read transaction). Current value of the data was =011ae6f6, the expected was 011ae6f6. MATCH!
At time t =            37500000.Transfer out of mapa range
At time t =            37900000. (Write transaction). Current value of the data was =02b0ec02, the expected was 02b0ec02. MATCH!
At time t =            38300000. (Write transaction). Current value of the data was =039cad34, the expected was 039cad34. MATCH!
         100 transactions made;          47 write and          34 read transactions.
          19 Transactions out of slave address memory limits.
PASS, all transactions was performed.
```

depicted in Figure 17. Summary reporting of the transaction performed shown in Figure 18 works similarly to the first test described above. To be sure of its correct operation, an inspection was performed on the simulator, checking its operation in the time diagram as shown in Figure 19. From the setup represented in Figure 17 it was possible to scrutinize 100 pseudo-random transactions.

Finally, the multi-master arbitration function of the bus was validated using the setup shown in Figure 20, which is similar to the test presented in Figure 17,

Figure 22. A sample of timeline test of 100 Condor transactions.



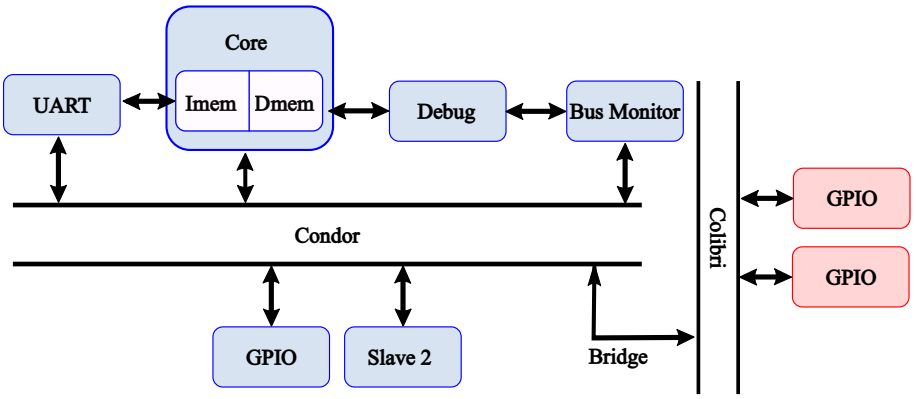
with the difference of having two different master interfaces connected to verify the management of the bus, when one or more masters are connected to the system. The test was composed for 200 transactions, initialized using a pseudo-random function. Figure 21 reports the results of applying this test. The bus is operating correctly as Figure 22 presents.

3.2. FPGA Emulation

Figure 23 is a simplified block diagram of a microcontroller implemented. The proposed system and peripheral buses were integrated within an FPGA to validate the correct performance within a system. The SoC was synthesized in an Artix-7 35T Arty FPGA (13), it has the capability to be programmed in C code, using the RISC-V GNU Compiler (14). One of the C code used to validate the bus is shown in Figure 24, the code was written to perform multiple read and write transactions

through the bus.

Figure 23. Simplified blog diagram integrated in an FPGA.



In the first part of the code shown in Figure 24, between lines 5 and 9, the variable *a* is loaded with the word *deadbeaf* byte by byte, performing 4 independent transactions as Figure 25 illustrates. Figure 25 plots the protocol operation of the system and peripheral bus in a time diagram, also, summarizes some of the most relevant protocol signals of Condor master, Colibri slave, and SoC output signals. Figure 25 plots the core behavior sending the data through the Condor, bridge, and Colibri bus to the peripherals. The write data is assigned to CWDATA in Condor bus and PWDATA signals in Colibri bus and finally, the word *deadbeaf* is loaded in $rego\_0$.

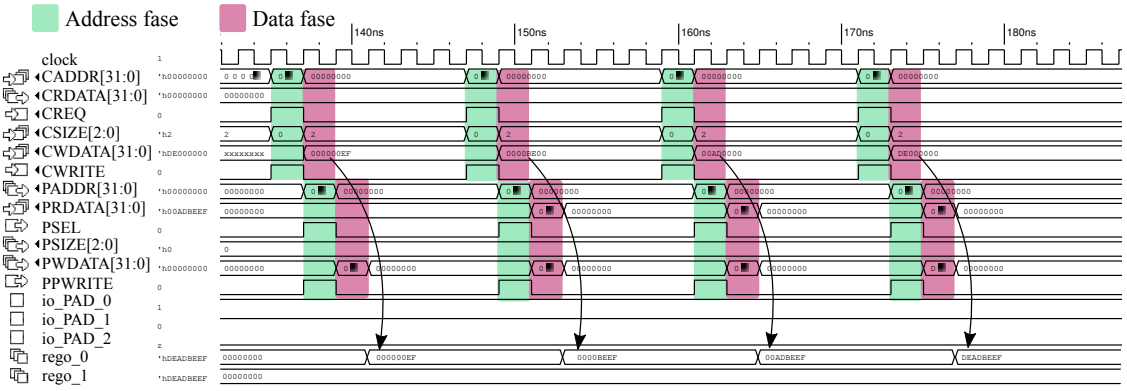
After writing the data, the C program reads the previously assigned data to later assign it to the *b* variable. The read transaction is in a *for* loop, it has a longer execution time due to the compiler have to translate the *for* function as adds, comparisons, and reads instructions. The extra time can be evidenced in Figure 26. After executed the *for* loop, $rego\_1$ loads the *deadbeaf* word.

Subsequently, to verify read and write transactions, a blink in a GPIO was programmed, this was connected to a LED as shown in Figure 28. The blink begins if the comparison between *a* and *b* results equal. Figure 27 shows the

Figure 24. Programming in C to test the behavioral bus.

```
1   #define GPIOBASE 0x209C
2
3   void main(int argc, char** argv) {
4   int data=0;
5   char * a = 0x2144;  //address GPIO 1
6   a[0]=0xEF;
7   a[1]=0xBE;
8   a[2]=0xAD;
9   a[3]=0xDE;
10  char * b = 0x2144+4;//address GPIO 2
11  for(int i=0; i< 4 ;i++){
12    b[i]=a[i];
13  }
14   if( (*a) == (*b) ){
15    pinMode( 0,OUTPUT);
16    while(1){
17      digitalWrite( 0  ,  data&0x1 );
18      data = !data;
19    }
20  }
21  else {
22    pinMode( 0,OUTPUT);
23   digitalWrite( 0  ,  data&0x1 );
24  }
```

Figure 25. Write transfer generated in c cod lines from 5 to 9.



blink in a GPIO in a time diagram. As a result of the C code tested, the behavior of the bus works correctly, performing a blink in the LED as expected (Figure28).

3.3. FORMAL VERIFICATION

Formal verification is a recent paradigm of verification and circuit design. It consists on verifying a design detailing some requirements (properties) and stimu-

Figure 26. Read transfer generated in c code lines from 10 to 13.

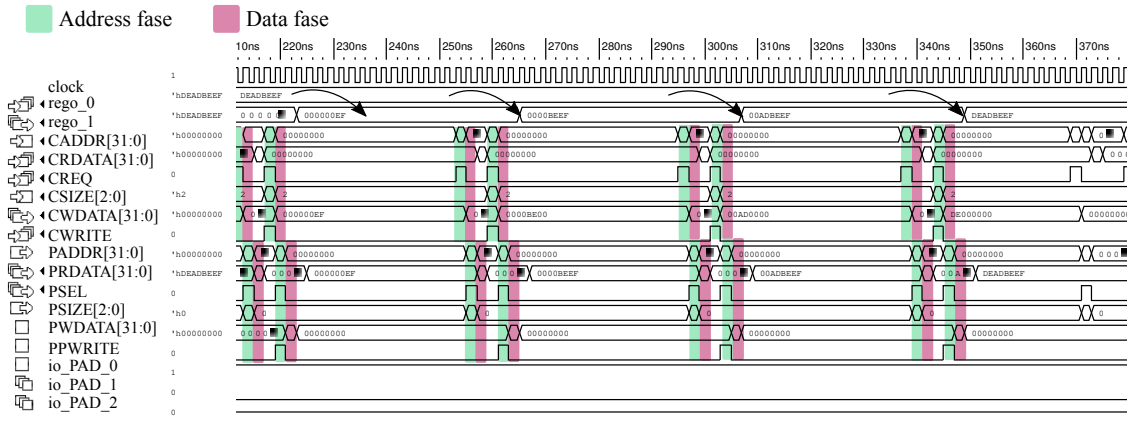


Figure 27. Blink generated in c cod lines from 14 to 24.

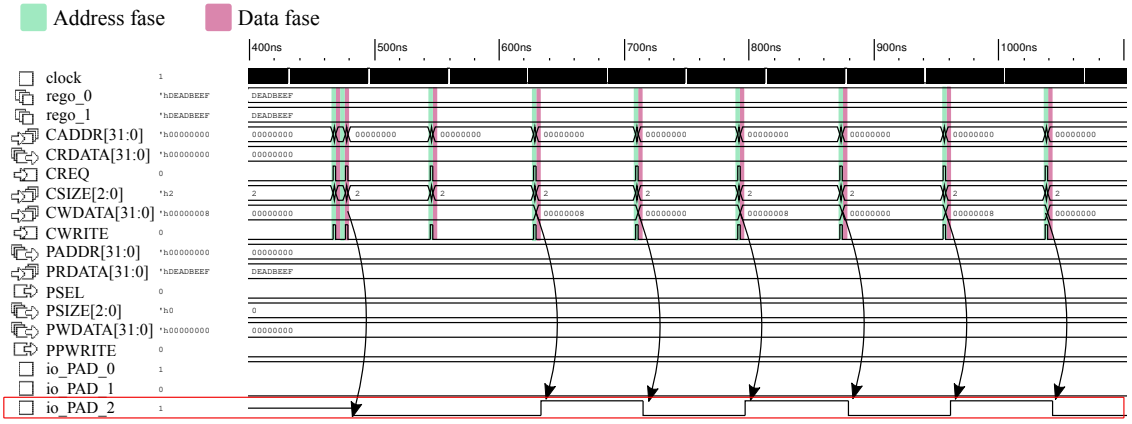


Figure 28. Photograph of Blink running on the FPGA.

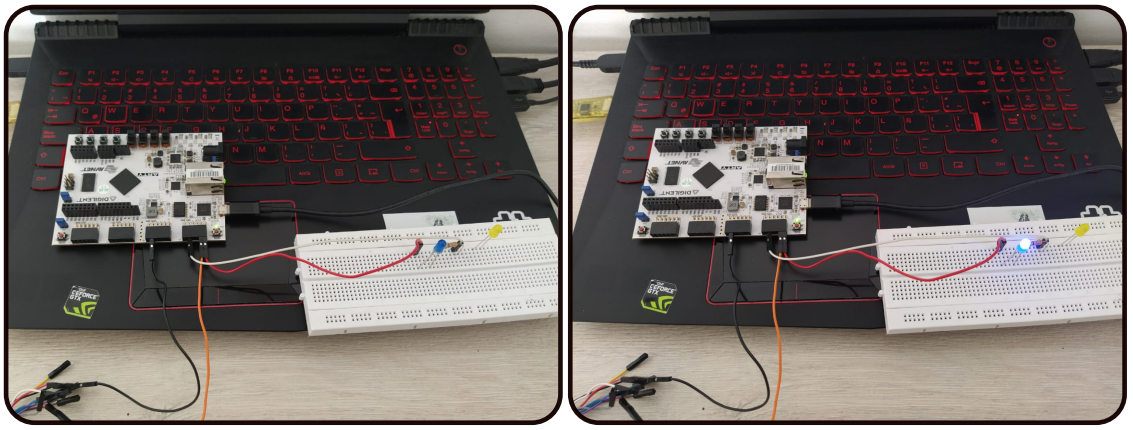
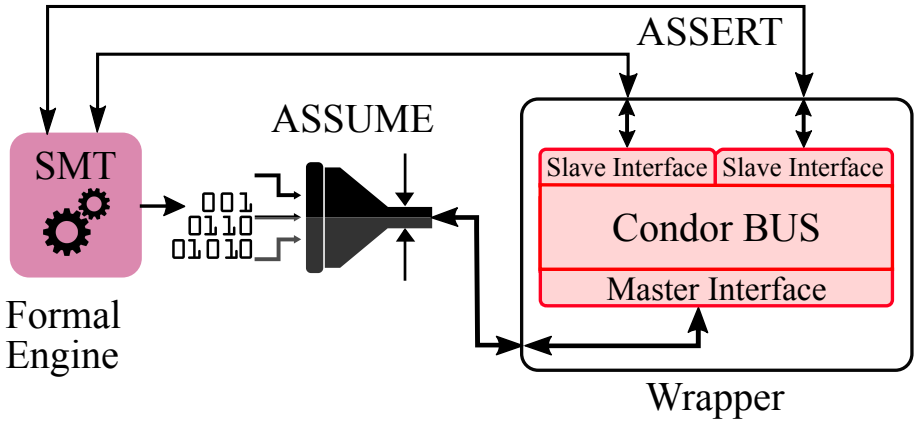


Figure 29. Formal verification setup, including Condor bus.



lating the circuit using clever mathematical techniques. Within the advantages of formal verification are the complete coverage (possible design behaviors analyzed) depending on the constrains. This option of verification offers the possibility to find and simulate errors in corner cases, and understand infinite behaviors thanks to the power of mathematical reasoning. Thereby, asking and answering questions about model behavior is allowed over unbounded periods of time (15).

Formal Verification is a novel way to verify hardware design, that depicts a new alternative in the building and verification of digital circuits. With the purpose of testing the bus more rigorously, the first steps in formal verification were taken into account. For that, a setup depicted in Figure 29 was made. Figure 29 depicts a basic configuration used to test Condor bus. The setup includes a Condor bus with a single master and 2 slaves, and the set of asserts and formal assumes to verify the implementation of the protocol.

Since the full formal verification of the proposed bus is an extensive task, it is beyond the objectives of the project. Thus only some examples of formal verification properties were tested.

In the table 3 are enumerated a list of some basic properties of the bus. Three

of these were translated into its equivalent formal rule (assumes and asserts). The setup to verify formally the bus was made, and the full process of applying a formal rule of formal verification is described afterward.

Table 3. Condor and Colibri protocol properties list.

Nº	**Protocol properties list**
1	The transactions have two phases: the first one establishes the parameters of the transactions. In the second phase, the slave sends the data if it is a read transaction or receives the data if it is a write transaction simultaneously with CREADY in high.
2	CSEL in high indicates the slave the communication will be established with.
3	When receiving or sending the data, the slave responds with CREADY in high and CRESP in low if the transaction was done properly, otherwise the slave responds with both CRESP and CREADY in high to indicate an error.
4	The master requests to occupy the bus asserting the CREQ signal in high. The signal to start the transaction by the master is given by the CVALID signal.
5	To extend the time of the transaction, the CREADY signal is lowered to 0 and the data phase of the next transaction is maintained, following the pipeline.
6	More than one transaction is never performed at the same time.
7	The data phase can only be extended by a CREADY of previous transaction.
8	The address phase of some transactions occurs during the data phase of the previous a transaction. Both phases overlap completely in a pipeline transaction.

9	Only the slave with which the transfer is proceeding must receive the information corresponding to the data phase.
10	The master can assert CVALID and CRESP signals in the same clock cycle.
11	The bus will be occupied by the master without the possibility of change until it sets the CREQ signal to 0.
12	If the address does not correspond to any slave, the master must receive CREADY and RESP signals in high indicating that an error occurs.
13	The master should not receive read data during write transactions.
14	The slave must not send data during read transactions.
15	The data phase must hold at least one clock cycle.
16	Transactions are handled in the same order in which they are requested.

Formal verification is a non common verification technique, the procedure involves different tools from simulation-based. The methodology to apply formal verification is explained below.

To employ the formal verification process, it is necessary to detail some files in advance. One of them is the verilog file of the module to apply this verification strategy. The file is generated from the scale specifying outputs, inputs, and internal signals as ports (bundles in scale) to be used in the formal verification process. In this case, a Condor bus with 1 master and 2 slaves was specified. Further, the SymbiYosys environment must be described. This configuration file

Figure 30. Sby code description to configure the engine and setup.

```
[options]
mode bmc
depth 10

[engines]
smtbmc

[script]
read -formal wrapper.sv
read -formal BUS_formal.v
prep -top wrapper

[files]
BUS_formal.v
wrapper.sv
```

Figure 31. Formal verification code to check CSEL behavior.

```
1  if (!reset ) begin   // selector  test
2    assert (((!io_s1_csel ) | | ( !io_s2_csel ));
3    if($past(!reset))begin
4      if($past((io_m1_caddr > addr1 )&&(io_m1_caddr < (addr1+size1-2'h1)))&&$past(io_m1_creq))
5        assert (io_s1_csel == 1);
6      else if($past((io_m1_caddr > addr2 )&&(io_m1_caddr < (addr2+size2-2'h1)))&&$past(io_m1_creq))
7        assert (io_s2_csel == 1);
8    end
9  end
```

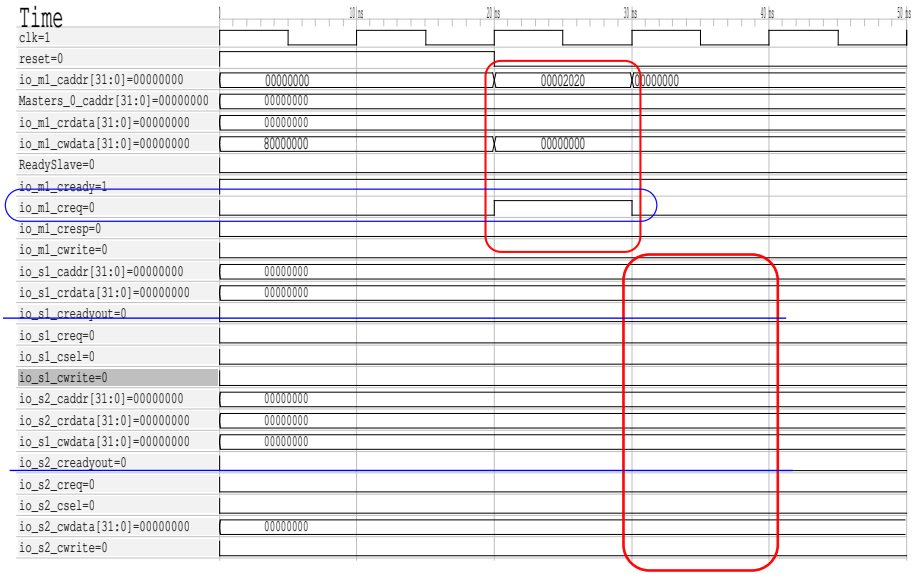
has included the operation modes, number of checking steps, the mathematical engines, and the files which will be verified, as shown in Figure 30. Finally, the main file to build is a SystemVerilog file. This file includes the instances of verilog file and the description of formal rules using assumptions and assertions. The interaction between the components of the formal verification process is depicted in Figure 29.

The first formal verification test performed was the prove of item 2 on the table 3. The $CSEL$ is one of the most significant signals in the protocol, it indicates with which slave the communication is being established. First, it was necessary

to describe a time of reset in high as a assume rule, thereby to ensure that the registers define an initial state. Regarding the CSEL signal two aspects were proven: first, just one CSEL signal has to be asserted at the same time. It was defined as a *assert ((!io_s2_csel)|| (!io_s1_csel))*. The only state when the condition false is that *io_slave1_csel* add *io_slave2_csel* signals are both raised at the same time. The second aspect proved about CSEL signal was that CSEL signal activation corresponds to the slave assigned in the addressing map. For this case, the condition was described as *$past((io_m1_caddr* $>$ *addr1)* && *(io_m1_caddr* $<$ *(addr1+size1-2'h1)))* && *$past(io_m1_creq)*. This previous expression is the condition to discriminate between the slave address. The $pass$ is used to emphasize that the expression is sensitive to a previous response. Figure 31 shows the final formal description of CSEL behavior.

After running the first formal test, a problem was detected as depicted in Figure 32. The formal tool declares a valid address in $CADDR$, which corresponds to slave 1, the correct behavior should have changed the IO_S1_CSEL to high in the next clock cycle and transmit all signals from master to slave. However, no signals were sent. It was because only the $IO\_S1\_CREADYOUT$ and $IO\_S2\_CREADYOUT$ signals in high were taken into account. It was expected that the slaves would initiate the transaction by setting $CREADYOUT$ signal; however, it is not part of the protocol and the slave is not forced to do it. The problem was not detected in simulation-based verification and this problem will not affect the communication in the actual system, nevertheless, verify that the bus works in cases like this makes the bus more robust. After the error was detected, the solution was patched in the bus code and was verified with formal verification.

Figure 32. Formal verification Time diagram using Gtkwave.



4. SUMMARY

4.1. CONCLUSIONS

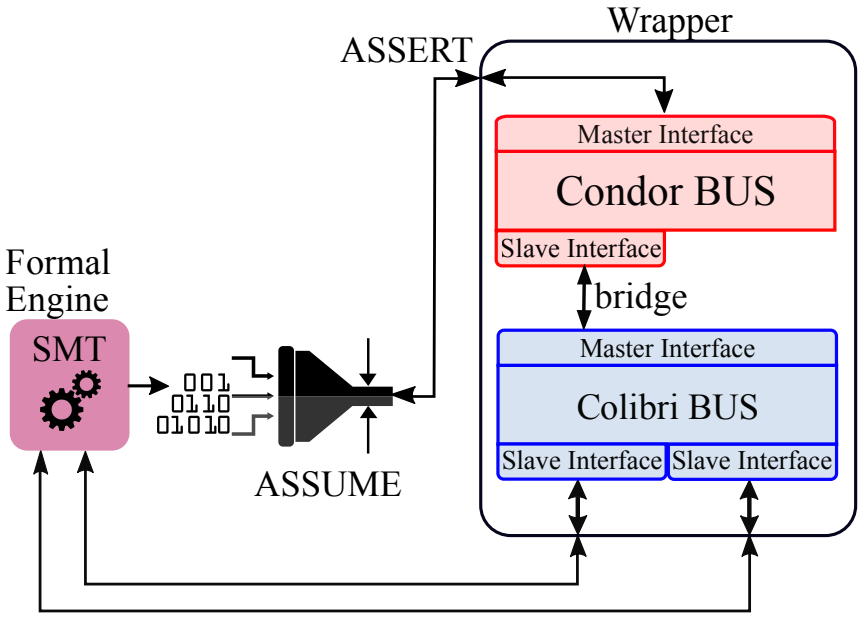
In this work, we introduced a scalable bus scheme composed of a system bus and a peripheral bus, called *Condor* and *Colibri* respectively. The proposed buses system offers the possibility of communicating modules from the system bus with modules in the peripheral bus and vice versa. *Condor* and *Colibri* were synthesized in a 180nm CMOS standard technology and integrated within an SoC along with a RISC-V based core, a 2kB RAM and a 10-bit ADC. The energy consumption of the bus system was 14.49pJ/cycle while the area was 91.92μm^2, equivalent to 4% of the total area of the SoC. In order to compare the performance of the proposed bus with conventional approaches, we also assessed conventional TileLink and AHB-Lite & APB buses implementations. As a result, the proposed buses system exhibits a 5X clock cycle reduction when sending data from a peripheral module to another module at the system bus. The proposed bus was tested using three verification methods; simulation-based verification, FPGA emulation and formal verification. About this last one, the methodology was described and a list of rules was defined in table 3.

Future microcontrollers in development by the OnChip research group would use this work in order to ease communication among peripherals and the core. Results indicate a potential solution for different SOC implementations, paving the way to broader applications. This work is a contribution to the growth of the open hardware/standard, featuring an alternative system bus and a peripheral bus for low-performance applications

4.2. FUTURE WORK

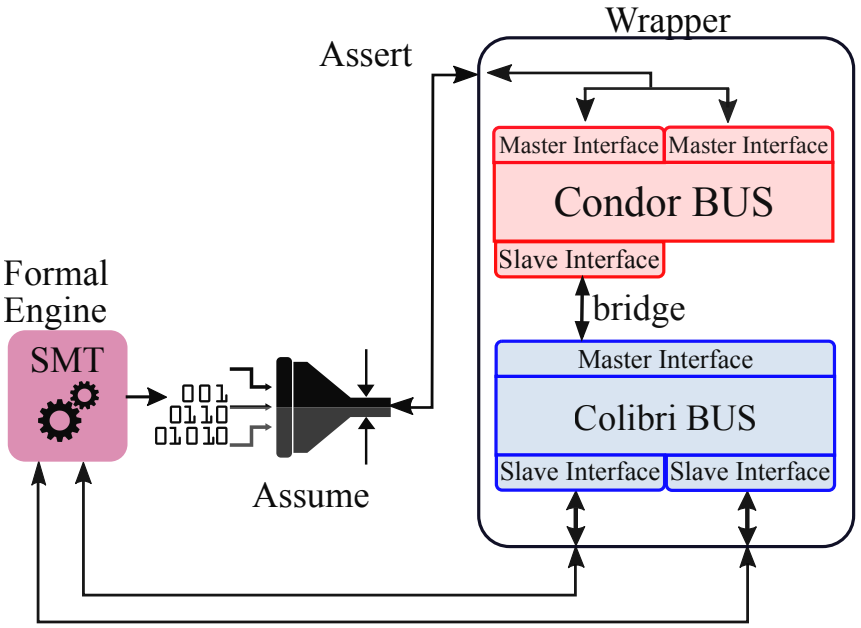
Integrated systems research group OnChip, associated to the Universidad Industrial de Santander, works on proposing new solutions of high-speed interfaces, novel IP solutions, and security circuits. These works are usually integrated with SoCs to be measured and report the results. The OnChip group has a 28nm tapeout planned for October 2020, where Condor and Colibri will be part of the SoC, as a system and peripheral bus respectively. Having trust as a priority, and to reduce the probability of failure, it is recommended for future works to apply formal verification to the bus, using as a reference the remaining items presented in table 3.

Figure 33. Formal verification setup, including Condor and Colibri buses.



Condor and Colibri have similar behaviors. Errors found in Condor using formal verification could be extrapolated to the peripheral bus, Colibri, and apply the corrections on both buses. However, it is recommended to use formal verification to test both buses working at the same time. Figure 33 depicts the advised

Figure 34. Formal verification setup, including multi-master Condor and Colibri bus.



system configuration using Condor and Colibri to test both buses through formal verification.

Figure 34 illustrates another relevant test to verify formally. In this case, the multi-master feature is assessed. This aspect is pertinent due to the possibility of modern microcontrollers and SoCs including more than one master connected to the system bus.

4.3. CONTRIBUTIONS

As outcomes of this master's research, a journal and an article were published in two IEEE Computer Hardware Science conferences. One of these as the main author and the second as a co-author. Moreover, a portable device for applying temperature variation to chips on boards was made.

4.3.1. Related Work The article titled ”Energy Efficient Peripheral and System Buses for Low-Area and Low-Power SoC Applications” was submitted in IEEE International Symposium on Circuits and Systems (ISCAS) conference. The paper was accepted for oral presentation at ISCAS 2020, a conference of IEEE Circuits and Systems (CAS) Society; it was also highlighted as one of the best papers in the System on Chip, Network on Chip, & Multi-core System I session. Thanks to the positive evaluation of the reviewers, the paper was promoted to be submitted a paper extended version in the journal Transactions on Circuits and Systems II (TCAS-II). Finally, the paper was published in TCAS-II (16).

4.3.2. Co-related Work In parallel with the thesis work, I was involved in different assignments within OnChip group projects. One of these was Tucan, a project developed with SiFive. Tucan is an SoC with a RISC-V RV32IMAC based processor developed by SiFive and analog IPs developed by OnChip. My contribution in Tucan was related to the design of the digital interfaces between analog IPs with the digital control managed through the Tilelink bus. The results of this work were published in the IEEE Integrated Circuits Conferences 2020 (CICC), one of the most relevant international conferences of engineering & computer science. The paper titled ”An Energy-Efficient RISC-V RV32IMAC microcontroller for Periodical-Driven Sensing Applications” shows the energy advantages of having several power states in order to reduce the power consumption of the SoC (17). This paper presents different techniques of power management in an SoC focused for low-performance tasks. The document presents the performance analysis, features and measures of the different operation modes.

Furthermore, I led the design and construction of a portable device to vary the temperature of a chip. The device was built to perform temperature variation tests

on integrated circuits performed by the OnChip research group. Some details and features of this device are shown in the appendix.

BIBLIOGRAPHY

ARM,AMBA AXI and ACE Protocol Specification, https://static.docs.arm.com//
/ihi0022/g/IHI0022G amba axi protocol spec.pdf, July 2019.

ARM,AMBA Specification, https://www.arm.com/products/silicon-ip-system
/embedded-system-design/amba-specifications,May 1999.

A. Waterman and K. Asanovic, The RISC-V Instruction Set Manual Volume I: BaseUser-level ISA, 2019.

A. Pullini, D. Rossi, I. Loi, G. Tagliavini, and L. Benini, Mr.Wolf: An Energy-Precision Scalable Parallel Ultra Low Power SoC for IoT Edge Processing,IEEE Journal of Solid-State Circuits, vol. 54, no. 7, pp. 1970–1981, July 2019.

A. Waterman and K. Asanovic, “The RISC-V Instruction Set Manual Volume I: BaseUser-level ISA,” 2019.

C. Duran, M. Wachs, L. E. Rueda G., E. Roa, and K. Asanovic, An energy-efficient risc-v rv32imac microcontroller for periodical-driven sensing applications, in 2020 IEEE Custom Integrated Circuits Conference (CICC), 2020 , pp. 1–4.

Electrow, “W1209 High-precision Digital Thermostat Incubator Temperature Con-

troller, https://www.elecrow.com/ w1209-high-precision-digital-thermostat -incubator-temperature-controller.html

Erik Seligman, Tom Schubert and M V Achutha Kiran Kumar, Formal Verification An Essential Toolkit for Modern VLSI Design. Morgan kaufmann, 2015.

IBM, “On-Chip Peripheral Bus Architecture Specifications.” IBM, April 2001.

J. Romero, N. Cuevas, and E. Roa, Energy efficient peripheral and system buses for low-area and low-power soc applications, IEEE Transactions on Circuits and Systems II: Express Briefs, vol. 67, no. 5, pp. 866–870, 2020.

J.P Romero Galindo, Design and Implementation of AXI4-Lite to APB Bridge Based on Advenced Microcontroller Bus Architecture (AMBA) 4.0 Using Verilog. Tesis de grado, Universidad Industrial de Santander, Santander, Colombia, diciembre 15, 2016.

M. Caldari, M. Conti, M. Coppola, P. Crippa, S. Orcioni, L. Pieralisi, and C. Turchetti, “System-level Power Analysis Methodology Applied to the AMBA AHB Bus [SoC Applications],” in DATE, March 2003, pp. 32–37.

Pat Pannuto, Yoonmyung Lee et. al, “MBus Specification,” June 14, 2015.

P. D. Schiavone, D. Rossi, A. Pullini, A. Di Mauro, F. Conti, and L. Benini, Quentin:

an Ultra-Low-Power PULPissimo SoC in 22nm FDX, in 2018 IEEE SOI-3D-Subthreshold Microelectronics Technology Unified Conference (S3S),
Oct 2018, pp. 1–3.

RISC-V, “RISC-V GNU Compiler Toolchain.” [Online]. Available:
https://github.com/riscv/riscv-gnu-toolchain.

SiFIve, SiFive TileLink Specification, https://www.sifive.com/documentation/ tilelink, December 2018.

STmicroelectronic, STBus Communication System Concepts and Definitions. STmicroelectronic, October 2012.

Xilinx, “Artix-7 35T Arty FPGA Evaluation Kit.” [Online]. Available:
https://www.xilinx.com/produ cts/boards-and-kits/arty.html#documentation.

Appendix

Complementary work was made during the Master's period, as an integral part of the student's development and training. This chapter shows a cooler-warm prototype to increase or decrease the temperature in a controlled way for SoCs.

The OnChip group has as a main field of work the design of microelectronic circuits. Members of the OnChip group implement, propose, or improve IPs for a high-speed interface, security circuits, or SoCs. The most relevant designs are silicon integrated. After the circuits have been fabricated, the circuit specs have to be tested. A relevant variable to consider when testing is how the response of the circuit is modified by temperature variations.

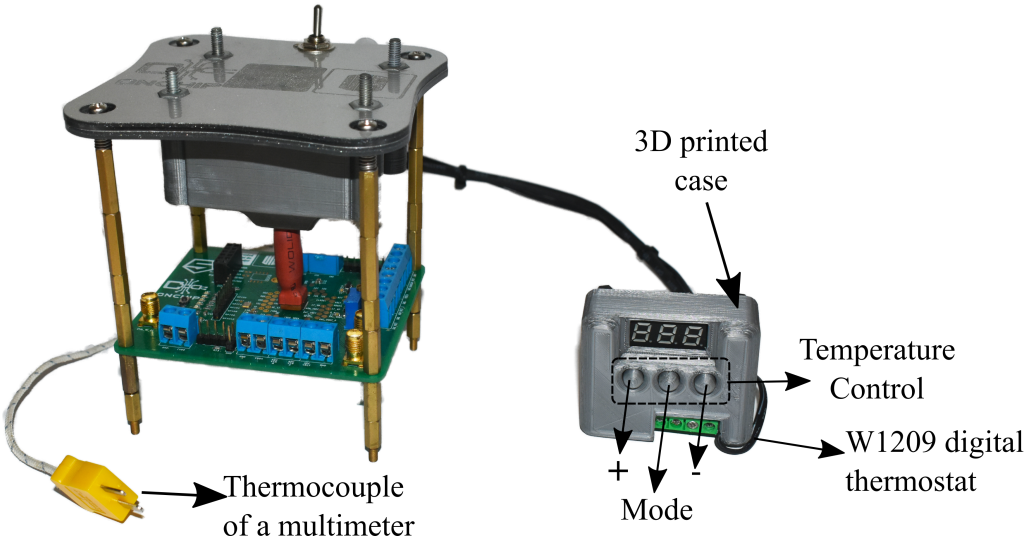
For circuits expected to be fabricated on a large industrial scale, temperature testing is essential to guarantee the right operation of the circuit in complex situations. As for example, cold or hot environments or when functioning nearby temperature variable systems.

Current commercial devices to control and modify the temperature variations on circuits are expensive. For this reason, to build a cheaper and portable alternative was necessary. Figure 35 shows the final prototype made. This is composed of a power source, a bottle of water to supply the liquid cooling, hoses to transport the liquid, two fans, a metal header to transmit the temperature of the system and temperature control, and a case to secure the system. The case holds together the system with the chip to enable the temperature variation.

The cooling system has two sensors to measure temperature. The first is a thermocouple of a multimeter and the second is a W1209 digital thermostat (18),

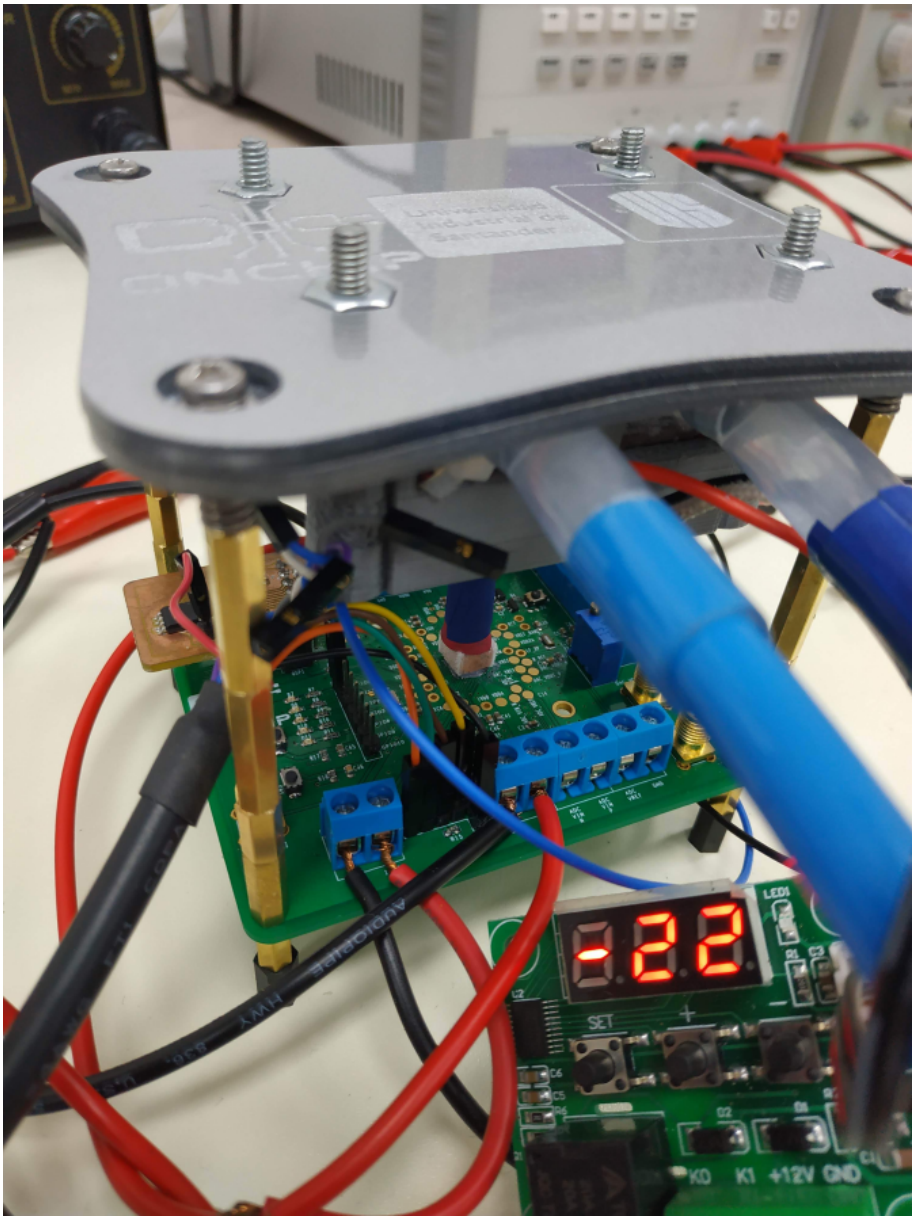
Figure 35. Portable device part list.

Figure 36. Cooler temperature sensors.



as shown in Figure 36. The system can vary the temperature between -15 and 50 degrees Celsius. Figure 37 shows the system operating at its maximum cooling power, -22 degrees Celsius. This measurement corresponds to the temperature near the Peltier cell, but the temperature at the contact point between the mechanism and the chip is around 7 degrees more (-15 degrees). This heat is produced

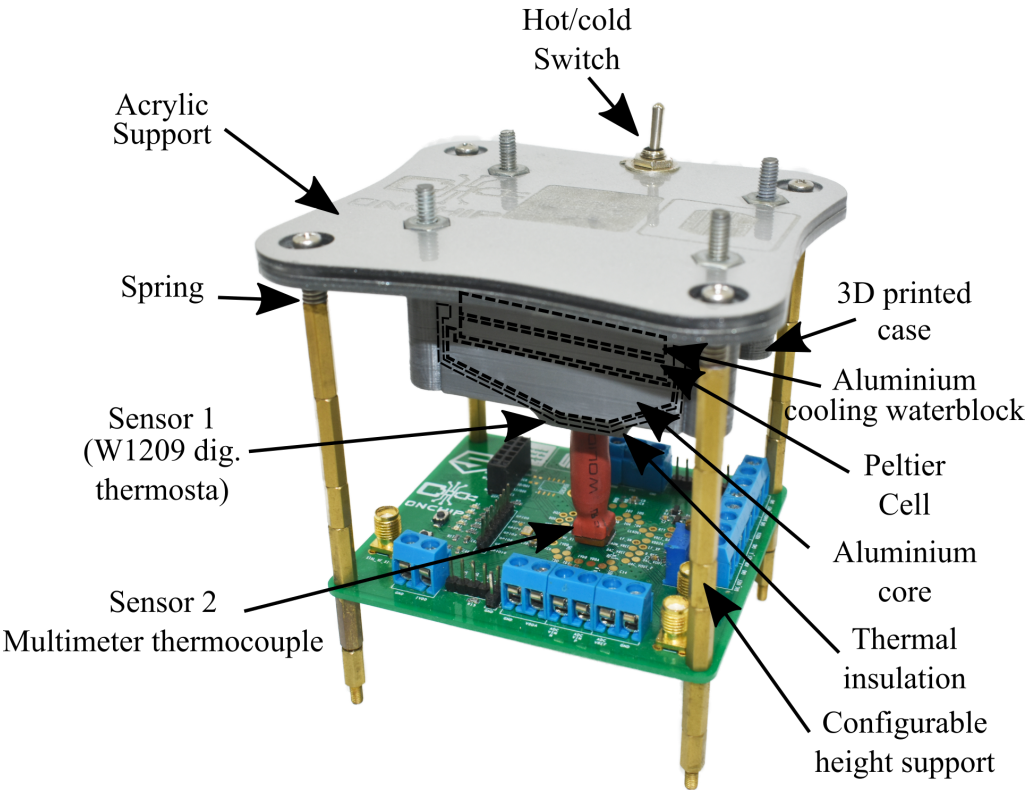
Figure 37. Cooler implementation in an OnChip develop board.



by the temperature exchange between the Peltier cell and the metal that makes contact with the chip. The Peltier cell offers the most accurate measurement because the sensor is located closer to the contact surface.

Figure 38 displays the inclusion of a spring. This small piece plays a significant role in the design of the system. There is a screw in each corner of support. The

Figure 38. Adjustment system between the cooler and the board.



springs push the system from the PCB, creating a resistance to the pressure generated by the weight of the device. The fine height adjustment is made by adjusting the screws.