IEIE Transactions on Smart Processing and Computing, vol. 5, no. 1, February 2016
http://dx.doi.org/10.5573/IEIESPC.2016.5.1.55

Hybrid Multi–System-on-Chip Architecture as a Rapid Development Approach for a High-Flexibility System

Rachmad Vidya Wicaksana Putra and Trio Adiono

Integrated Circuits Laboratory, Microelectronics Center, Institut Teknologi Bandung / Bandung, Indonesia
rachmad@pme.itb.ac.id, tadiono@stei.itb.ac.id

***** Corresponding Author: Trio Adiono

Received February 20, 2016; Accepted February 25, 2016; Published February 29, 2016

** Extended from a Conference: Preliminary results of this paper were presented at the ICEIC 2016. This present paper has been accepted by the editorial board through the regular reviewing process that confirms the original contribution.*

Abstract**:** In this paper, we propose a hybrid multi–system-on-chip (H-MSoC) architecture that provides a high-flexibility system in a rapid development time. The H-MSoC approach provides a flexible system-on-chip (SoC) architecture that is easy to configure for physical- and application-layer development. The physical- and application-layer aspects are dynamically designed and modified; hence, it is important to consider a design methodology that supports rapid SoC development. Physical layer development refers to intellectual property cores or other modular hardware (HW) development, while application layer development refers to user interface or application software (SW) development. H-MSoC is built from multi-SoC architectures in which each SoC is localized and specified based on its development focus, either physical or application (hybrid). Physical HW development SoC is referred to as physical-SoC (Phy-SoC) and application SW development SoC is referred to as application-SoC (App-SoC). Phy-SoC and App-SoC are connected to each other via Ethernet. Ethernet was chosen because of its flexibility, high speed, and easy configuration. For prototyping, we used a LEON3 SoC as the Phy-SoC and a ZYNQ-7000 SoC as the App-SoC. The proposed design was proven in real-time tests and achieved good performance.

Keywords**:** H-MSoC, High-flexibility system, Rapid development, Physical-SoC, Application-SoC

1. Introduction

System-on-chip (SoC) is an integrated circuit (IC) that integrates many electronic components into a single complete system. This technology has driven many developments in electronics and in a broad spectrum of applications, especially in the embedded systems world. Many devices and gadgets for many levels of applications were basically established from SoC technology. Thus, we can conclude that SoC technology has been indispensable to daily life.

Most SoC developments are driven by evolution in applications [1]. Each SoC design methodology faces unique challenges, with solutions based on the purpose of the application [2]. A lot of research into SoC technology has proven this [3-7]. Meanwhile, concerns about SoC technology are always the same: high performance speed, a small footprint, low power consumption, high flexibility and configurability, and a short time-to-market [8]. Thus, much of the research has been conducted in order to answer those challenges.

Traditionally, SoC design has focused on the computational aspect of modular hardware (HW) design or intellectual property (IP) components. Those modular hardware components will be plug-and-play based on the bus interface standard. This method will tend to make the area occupation of the SoC larger and larger, along with the increasing number of hardware components. According to Marculescu et al. [9], as the number of SoC components increases, architecture design of the communications aspect will dominate, defining several important parameters in the SoC, such as area occupation size, performance quality, and energy consumption. Furthermore, with today's technology scaling, inter-resources connections in the SoC can cause severe on-chip problems, such as unpredictable delays, high power consumption, and synchronization errors.

One of most popular solutions addressing the on-chip inter-resources connection problem is the network-on-chip (NoC) architecture. It was developed to achieve efficient on-chip interconnection among many resources (e.g. modular hardware components, such as the processor and IP core) [10]. With this technology, the internal resources become efficiently connected in a network-like concept with a router-like mechanism. This technology requires high design complexity and precise design to accommodate the good routing mechanism. Thus, NoC's inter-resources communications crash can be avoided. However, because of its high complexity, the development time may be more prolonged than in the traditional design.

Another idea for overcoming on-chip inter-resources connection problems is the multi-SoC architecture. By using this concept, two or more SoCs are integrated to become a single system. This idea was proposed by using an asynchronous bridge [11]. Homogeneous or heterogeneous SoCs can be implemented by using this development approach. But the design of an asynchronous bridge is not simple, because the type of SoC will define the unique challenges to designing a compatible asynchronous bridge. Moreover, design quality of the asynchronous bridge will contribute to defining the system's complexity and delay.

From a study of the literature, we can summarize the three important issues in SoC design. First, the SoC has to be easy to configure in order to cope with its various applications. Thus, the SoC architecture needs to be flexible enough. Second, the on-chip inter-resources connections have to be efficient. A single processor will not be able to utilize all of the resources of an entire chip at the same time [5]. Thus, we have to keep both simplicity and efficiency in the on-chip connections. Third, SoC development time needs to be short enough to meet time-to-market constraints. These three issues are our research targets.

In order to achieve those three targets, we propose an architecture that uses a multi-SoC concept to separate and localize each single SoC to handle a specific development purpose, either physical layer development or application layer development. Physical layer development refers to intellectual property cores or other modular hardware (HW) development, while application layer development refers to user interface (UI) or application software (SW) development. They are connected to each other via the Ethernet protocol. A specific SoC for physical layer development is referred to as physical-SoC (Phy-SoC), whereas a specific SoC for application layer development is referred to as application-SoC (App-SoC). Because this approach uses a multi-SoC concept and a separate focus on physical–application layer development, we called the design approach a hybrid multi–system-on-chip (H-MSoC) architecture. By using the H-MSoC approach, we can achieve a SoC design that has high flexibility for many applications in an embedded system, efficient inter-

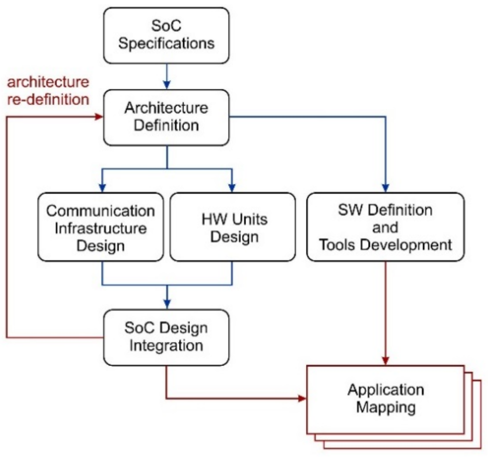


Fig. 1. Standard SoC design methodology [14].

resources connections in each SoC and between SoCs, and a short development time-to-market.

For prototype evaluation, we used a LEON3 SoC as the Phy-SoC and a ZYNQ-7000 SoC as the App-SoC. The LEON3 SoC was chosen because of its powerful computation capabilities and extensive documentation. Hence, it is suitable for use as a physical HW development platform. The ZYNQ-7000 SoC was chosen because of its flexibility and compatibility in executing user applications. Hence, it is suitable for use as an application SW development platform. Meanwhile, the Ethernet protocol was chosen for the inter-SoC communication scheme because of its high speed, flexibility, and easy configuration.

This paper presents several sections. Following the introduction about the research background, related research and the research targets is an overview of related works. A discussion on the proposed H-MSoC architecture is followed by a discussion on the proposed design methodology, which is followed by evaluation of results and analysis. The final three sections are the conclusion, the acknowledgement, and the references.

2. Related Works

System-on-chip is an IC that contains and integrates many electronic components to become a complete system. It has to ensure that each component and the whole integrated system work properly [2]. The SoC design approach uses a platform-based methodology; hence, the SoC developer only needs to focus on the creation of specific IP block functionality and its embedded program [12, 13]. This is the strong point of a platform-based methodology. The SoC developer does not bother to design all components from scratch.

There are two categories of SoC design methodology: standard SoC and communication-centric SoC [14]. These categories are based on the on-chip communications scheme. The standard SoC methodology is the common way to design a traditional SoC. Meanwhile, a communications-centric SoC methodology was developed to comply with NoC requirements. Since the NoC architec-

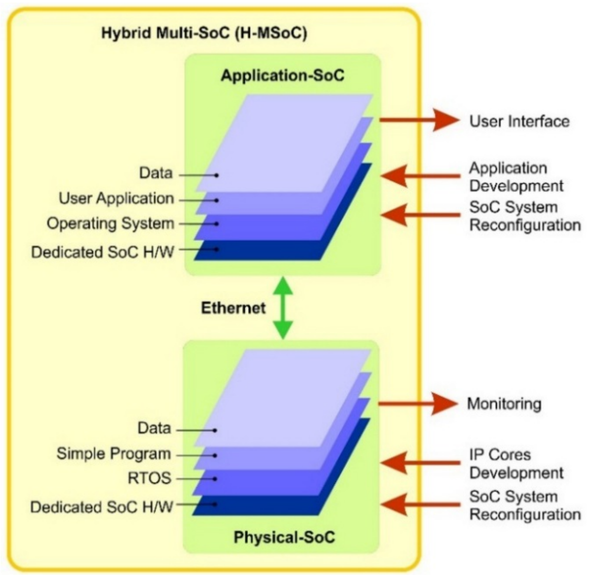


Fig. 2. H-MSoC architecture hierarchy.

ture is not the basic platform of this research, we can adapt the standard SoC to develop the H-MSoC design methodology. The standard SoC design methodology is shown in Fig. 1.

In the standard SoC design methodology, specification of the SoC comes first. Then, it needs to be integrated into the architectural design definition. From this definition, three design aspects can be explored: HW unit design, communications scheme infrastructure, and SW definition and tools development. HW units and the communications scheme are integrated into the SoC HW platform. It needs to be checked in order to ensure that its functionalities comply with the architecture definition. If the SoC HW platform is ready, the SW programs and applications can be implemented on it.

3. Proposed H-MSoC System Architecture

The hybrid multi–system-on-chip (H-MSoC) architecture is formed from Phy-SoC and App-SoC architectures, which are connected to each other via Ethernet, as illustrated in Fig. 2. The focus of Phy-SoC is IP core development, SoC HW configuration, and processing monitor. Meanwhile, the focus of App-SoC is application and the user interface development. Because of its separation and localization approach, both homogeneous and heterogeneous SoC platforms can be used.

Actually, the hierarchical designs of those SoCs are the same. The differences are in the choice of SoC HW architecture, operating system (OS), application program, and types of data. By judicious choice and design from those aspects, each SoC can be designed optimally and efficiently, yet at the same time, can attain high flexibility and a broad spectrum of applications.

For Phy-SoC design, the SoC HW platform should be easily configured in the HW design approach. It essentially supports a real-time operating system (RTOS) and simple programming, because they are sufficient for executing and running the instructions in the Phy-SoC. For App-SoC design, the SoC HW platform should be easily configured, supporting the higher level OS and high-level programming, because they are compatible with application and UI development. Furthermore, each SoC can be reconfigured anytime. Because of the different focus, Phy-SoC and App-SoC can be designed separately. Each SoC is independent. Thus, arguably, the H-MSoC architecture offers rapid development time in achieving high-flexibility functions.

4. Design Methodology and Prototyping

The proposed design methodology's flowchart can be seen in Fig. 3. First, we have to define the H-MSoC specifications and architecture, which can then be broken down into two major designs: the Phy-SoC and App-SoC design methodologies. Since we use the standard SoC design method, each SoC will be treated in a similar manner. Three parts of the SoC are designed: the HW unit, the application SW, and the communications infrastructure. Afterwards, if all SoCs are ready, they will be connected to each other using the Ethernet protocol. If all tests and evaluations show good performance, the H-MSoC architecture is proven.

4.1 Physical-SoC (Phy-SoC)

For Phy-SoC design, we built an embedded system with LEON3-based SoC HW as the reference for the architecture platform. The real-time operating system (RTOS) eCos is embedded in the Phy-SoC HW. We chose the eCos RTOS because it is freely licensed and feasibly configured for precise application requirements [15]. In the higher layers, simple programs run and compute the data.

For this Phy-SoC prototype, the Stratix II EP2S180 DSP development board was used. The focus in the Phy-SoC is IP core or modular HW development. Thus, in this prototype, we have two HW design scenarios. First, we added an existing IP core to prove that the SoC can be reconfigured with the existing IP cores. Second, we added a fully customized IP core to prove that the SoC can be reconfigured with new IP cores, even if designed from scratch. For the first scenario, a physical WiMAX design (PHY_WMX) was added with peripheral support [16]. For the second, a custom ROM design (RCH_ROM) was created and added into the Phy-SoC. The final LEON3-based Phy-SoC architecture is illustrated in Fig. 4.

4.2 Application-SoC (App-SoC)

For the App-SoC design, we built an embedded system with a ZYNQ-7000–based SoC HW as a reference architecture platform. The basic HW architecture of the ZYNQ-7000–based SoC was adopted from the Boot Partition Kit provided by Xillybus [17]. For the operating system, we chose Xillinux, which is also from Xillybus. It

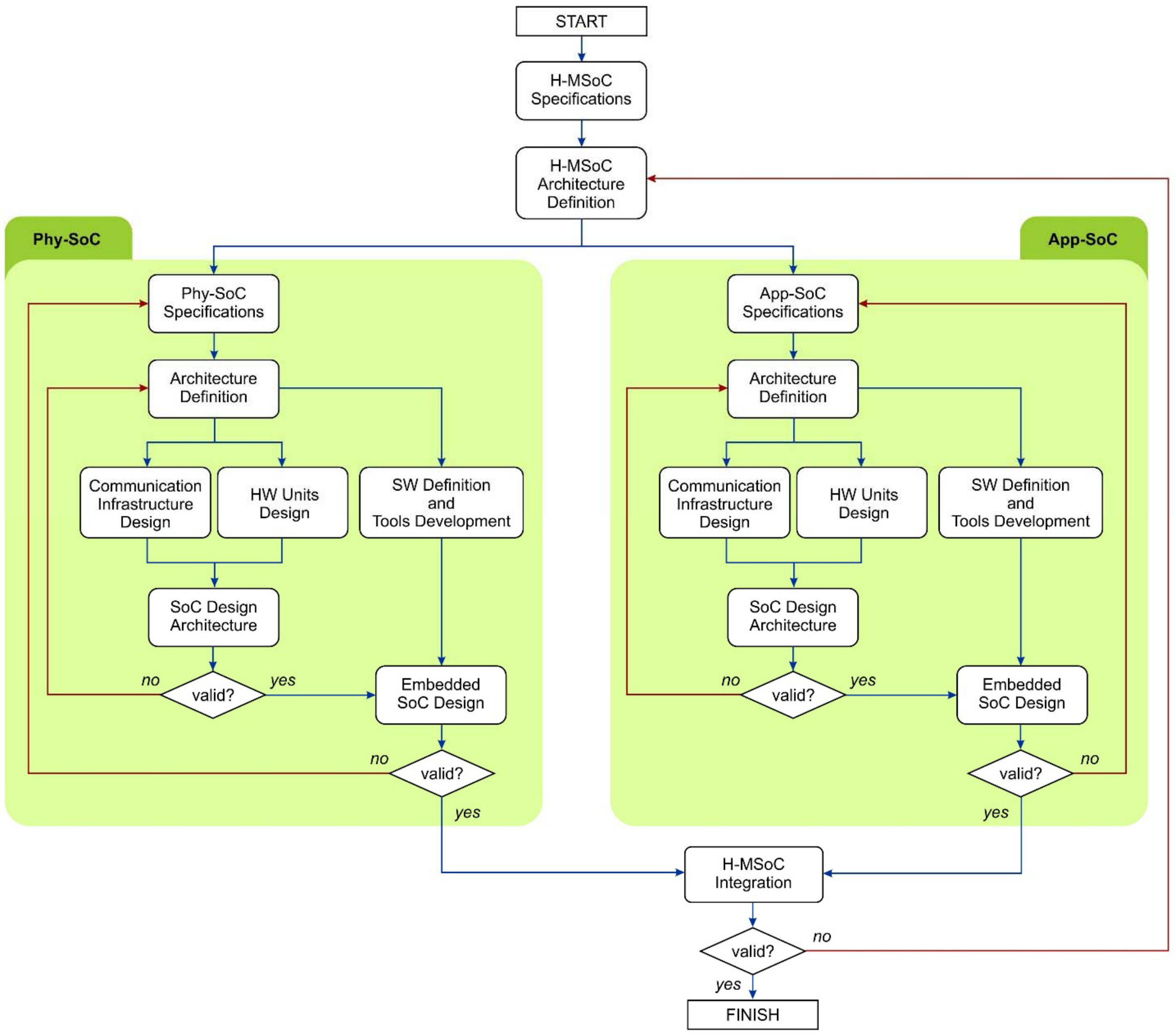


Fig. 3. H-MSoC design methodology.

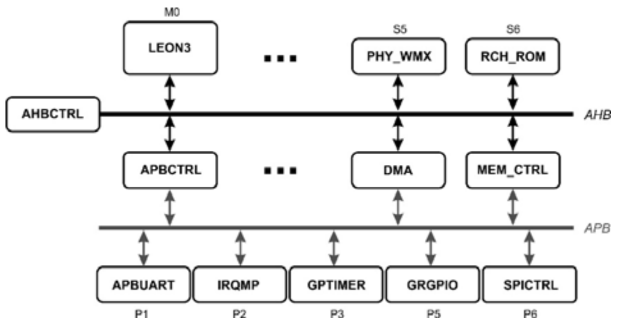


Fig. 4. LEON3 based Phy-SoC Architecture

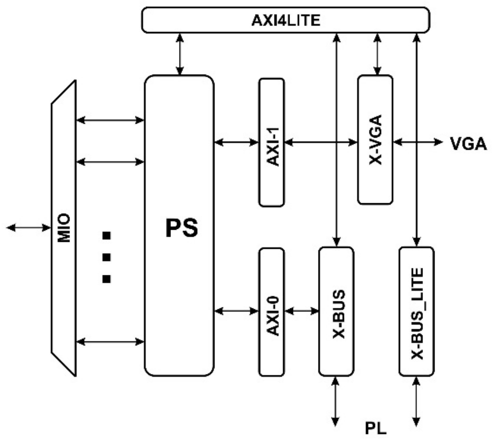


Fig. 5. ZYNQ-7000 based App-SoC Architecture.

is a Linux-based OS distribution for Zedboard, Zybo, Microzed, and SocKit [17]. In the higher layers, we can create functional programs or install existing applications to run and compute the data. For this App-SoC prototype, we used the Zybo development board. The final ZYNQ-7000–based App-SoC architecture is illustrated in Fig. 5.

The focus in the App-SoC is application or user interface development. Thus, in this prototype, we needed to use a high-functionality OS (a Linux distribution) and existing programs that run on the OS in order to prove that the SoC is capable of supporting a high-level OS and many UI-based applications. Hopefully, it can complement the

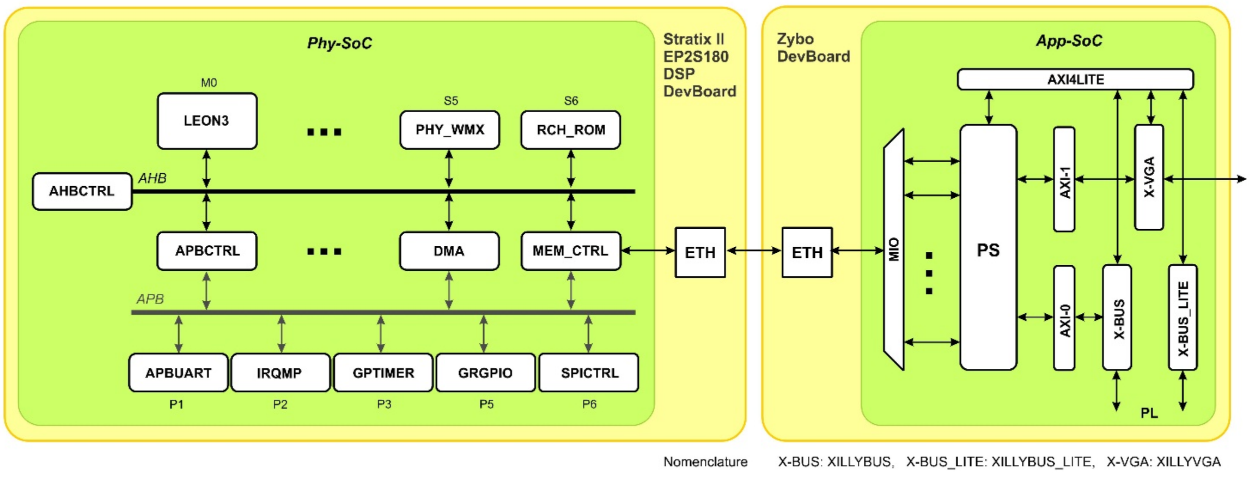


Fig. 6. Detailed H-MSoC architecture prototype.

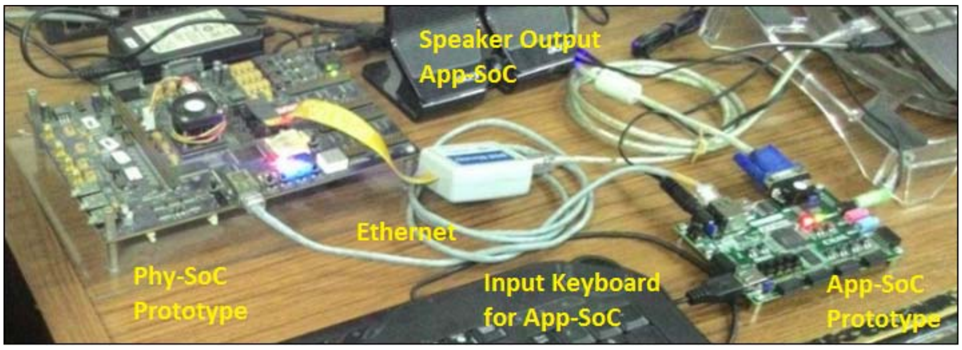


Fig. 7. Actual H-MSoC prototype.

Phy-SoC. Disadvantages of the Phy-SoC can be overcome by the advantages of the App-SoC, and vice versa.

4.3 Design Integration into H-MSoC

After the Phy-SoC and App-SoC are proven to work properly, design integration is the next step. We call this integrated design a hybrid multi-SoC. The integration infrastructure is established by using the Ethernet protocol, which was chosen because of its flexible configuration, high speed, and mature concept worldwide. Fig. 6 illustrates the detailed design integration in the H-MSoC architecture prototype, and Fig. 7 is the actual prototype snapshot.

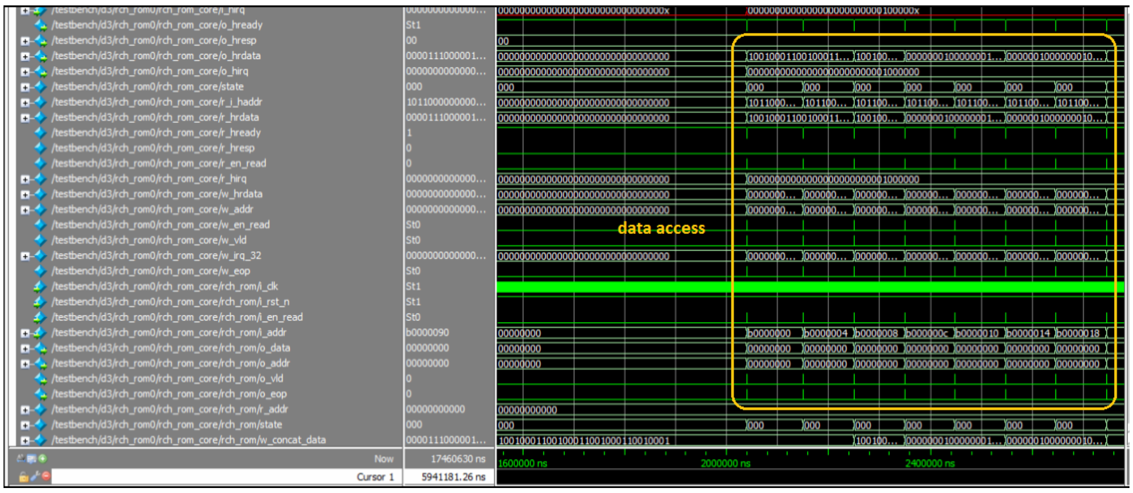
For inter-SoC connection, each SoC uses an ETH PHY. In the prototype, we use an SMSC LAN91C111 for the LEON3 SoC and a Realtek RTL8211E for the ZYNQ-7000 SoC. For Phy-SoC, ETH PHY is connected directly to a memory controller. Thus, every single piece of data transferred or received via Ethernet has to pass through the memory controller. Here, we only do socket programming to transmit and receive the data. Meanwhile, for App-SoC, ETH PHY is connected to multiplexed input/output (MIO), which is connected directly to the processing system (PS). Specifically, we use MIO16 up to MIO27 to connect Ethernet to the PS. Hence, we only need to install OS and Ethernet drivers, and thus, the Ethernet is ready to transmit and receive data.

5. Evaluation of Results and Analysis

5.1 Physical-SoC Evaluation

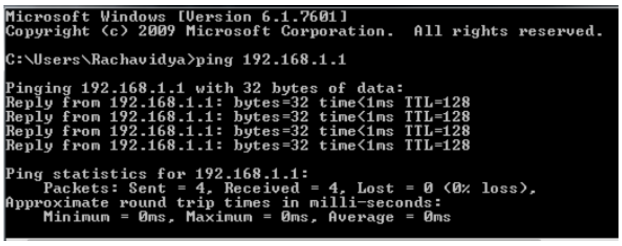
For functional evaluation, we created a test program to evaluate signaling performance inside the Phy-SoC. We mainly evaluated signaling from three aspects: (1) data access from the IP core, (2) data transfer over the Advanced Microcontroller Bus Architecture (AMBA), the Advanced High-performance Bus (AHB) and the Advanced Peripheral Bus (APB), and (3) interrupt performance. From the evaluation, we have proven that the Phy-SoC works successfully. Functional simulation results are presented in Fig. 8. We can clearly see that data access is done in the IP core. The data can also be delivered via AMBA, AHB, and APB in order to evaluate the interface performance between the IP cores and buses.

For communications evaluation, a network ping test and data transfer between Phy-SoC and a laptop were observed. “Transferred data” is a collection of customized data stored in the customized ROM (RCH_ROM). Data transfer is done by using User Datagram Protocol (UDP).



```
/testbench/d3/rch_rom0/rch_rom_core/i_hirq                    0000000000000...
/testbench/d3/rch_rom0/rch_rom_core/o_hready                  St1
/testbench/d3/rch_rom0/rch_rom_core/o_hresp                   00
/testbench/d3/rch_rom0/rch_rom_core/o_hrdata                  0000111000001...
/testbench/d3/rch_rom0/rch_rom_core/o_hirq                    0000000000000...
/testbench/d3/rch_rom0/rch_rom_core/state                     000
/testbench/d3/rch_rom0/rch_rom_core/r_i_haddr                 1011000000000...
/testbench/d3/rch_rom0/rch_rom_core/r_hrdata                  0000111000001...
/testbench/d3/rch_rom0/rch_rom_core/r_hready                  1
/testbench/d3/rch_rom0/rch_rom_core/r_hresp                   0
/testbench/d3/rch_rom0/rch_rom_core/r_en_read                 0
/testbench/d3/rch_rom0/rch_rom_core/r_hirq                    0000000000000...
/testbench/d3/rch_rom0/rch_rom_core/w_hrdata                  0000000000000...
/testbench/d3/rch_rom0/rch_rom_core/w_addr                    0000000000000...
/testbench/d3/rch_rom0/rch_rom_core/w_en_read                 St0
/testbench/d3/rch_rom0/rch_rom_core/w_vld                     St0
/testbench/d3/rch_rom0/rch_rom_core/w_irq_32                  0000000000000...
/testbench/d3/rch_rom0/rch_rom_core/w_eop                     St0
/testbench/d3/rch_rom0/rch_rom_core/rch_rom/i_clk             St1
/testbench/d3/rch_rom0/rch_rom_core/rch_rom/i_rst_n           St1
/testbench/d3/rch_rom0/rch_rom_core/rch_rom/i_en_read         St0
/testbench/d3/rch_rom0/rch_rom_core/rch_rom/i_addr            b0000090
/testbench/d3/rch_rom0/rch_rom_core/rch_rom/o_data            00000000
/testbench/d3/rch_rom0/rch_rom_core/rch_rom/o_addr            00000000
/testbench/d3/rch_rom0/rch_rom_core/rch_rom/o_vld             0
/testbench/d3/rch_rom0/rch_rom_core/rch_rom/o_eop             0
/testbench/d3/rch_rom0/rch_rom_core/rch_rom/r_addr            00000000000
/testbench/d3/rch_rom0/rch_rom_core/rch_rom/state             000
/testbench/d3/rch_rom0/rch_rom_core/rch_rom/w_concat_data     0000111000001...
                                                     Now      17460630 ns
                                                 Cursor 1     5941181.26 ns
```

Fig. 8. Functional simulation result.



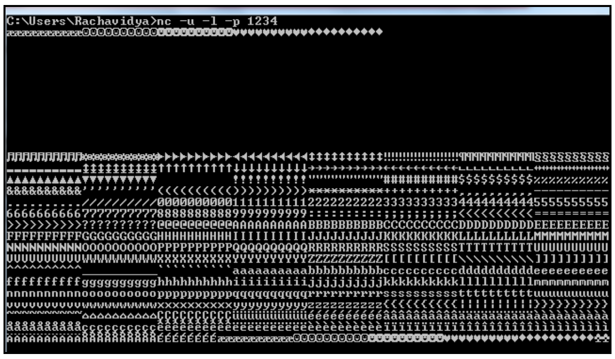
```
Microsoft Windows [Version 6.1.7601]
Copyright (c) 2009 Microsoft Corporation.  All rights reserved.

C:\Users\Rachavidya>ping 192.168.1.1

Pinging 192.168.1.1 with 32 bytes of data:
Reply from 192.168.1.1: bytes=32 time<1ms TTL=128
Reply from 192.168.1.1: bytes=32 time<1ms TTL=128
Reply from 192.168.1.1: bytes=32 time<1ms TTL=128
Reply from 192.168.1.1: bytes=32 time<1ms TTL=128

Ping statistics for 192.168.1.1:
    Packets: Sent = 4, Received = 4, Lost = 0 (0% loss),
Approximate round trip times in milli-seconds:
    Minimum = 0ms, Maximum = 0ms, Average = 0ms
```

Fig. 9. Network ping result.



```
C:\Users\Rachavidya>nc -u -l -p 1234
☺☺☺☺☺☺☺☺☺☺☻☻☻☻☻☻☻☻☻☻♥♥♥♥♥♥♥♥♥♥♦♦♦♦♦♦♦♦♦♦







♫♫♫♫♫♫♫♫♫♫☼☼☼☼☼☼☼☼☼☼►►►►►►►►►►◄◄◄◄◄◄◄◄◄◄↕↕↕↕↕↕↕↕↕↕‼‼‼‼‼‼‼‼‼‼¶¶¶¶¶¶¶¶¶¶§§§§§§§§§§
▬▬▬▬▬▬▬▬▬▬↨↨↨↨↨↨↨↨↨↨↑↑↑↑↑↑↑↑↑↑↓↓↓↓↓↓↓↓↓↓→→→→→→→→→→←←←←←←←←←←∟∟∟∟∟∟∟∟∟∟↔↔↔↔↔↔↔↔↔↔
▲▲▲▲▲▲▲▲▲▲▼▼▼▼▼▼▼▼▼▼          !!!!!!!!!!""""""""""##########$$$$$$$$$$%%%%%%%%%%
&&&&&&&&&&''''''''''(((((((((())))))))))**********++++++++++,,,,,,,,,,----------
..........//////////00000000001111111111222222222233333333334444444444555555555 5
6666666666777777777788888888889999999999::::::::::;;;;;;;;;;<<<<<<<<<<==========
>>>>>>>>>>??????????@@@@@@@@@@AAAAAAAAAABBBBBBBBBBCCCCCCCCCCDDDDDDDDDDEEEEEEEEEE
FFFFFFFFFFGGGGGGGGGGHHHHHHHHHHIIIIIIIIIIJJJJJJJJJJKKKKKKKKKKLLLLLLLLLLMMMMMMMMMM
NNNNNNNNNNOOOOOOOOOOPPPPPPPPPPQQQQQQQQQQRRRRRRRRRRSSSSSSSSSSTTTTTTTTTTUUUUUUUUUU
VVVVVVVVVVWWWWWWWWWWXXXXXXXXXXYYYYYYYYYYZZZZZZZZZZ[[[[[[[[[[\\\\\\\\\\]]]]]]]]]]
^^^^^^^^^^__________``````````aaaaaaaaaabbbbbbbbbbccccccccccddddddddddeeeeeeeeee
ffffffffffgggggggggghhhhhhhhhhiiiiiiiiiijjjjjjjjjjkkkkkkkkkkllllllllllmmmmmmmmmm
nnnnnnnnnnooooooooooppppppppppqqqqqqqqqqrrrrrrrrrrssssssssssttttttttttuuuuuuuuuu
vvvvvvvvvvwwwwwwwwwwxxxxxxxxxxyyyyyyyyyyzzzzzzzzzz{{{{{{{{{{||||||||||}}}}}}}}}}
~~~~~~~~~~⌂⌂⌂⌂⌂⌂⌂⌂⌂⌂ÇÇÇÇÇÇÇÇÇÇüüüüüüüüüüééééééééééââââââââââääääääääääàààààààààà
ååååååååååççççççççççêêêêêêêêêêëëëëëëëëëëèèèèèèèèèèïïïïïïïïïïîîîîîîîîîîìììììììììì
ÄÄÄÄÄÄÄÄÄÄÅÅÅÅÅÅÅÅÅÅÉÉÉÉÉÉÉÉ☺☺☺☺☺☺☺☺☺☺☻☻☻☻☻☻☻☻☻☻♥♥♥♥♥♥♥♥♥♥♦♦♦♦♦♦♦♦♦♦♣
```

Fig. 10. Data transfer result.

UDP was chosen because of its simple mechanism, which is suitable for simple data transfer evaluation. Test results show that the communications module in the Phy-SoC works properly, which means the Phy-SoC can communicate and transfer data with outside systems. Thus, the Phy-SoC is ready to be integrated into the H-MSoC. The network ping test and received data in the laptop are shown in Figs. 9 and 10, respectively.

5.2 Application-SoC Evaluation

For App-SoC evaluation, network ping and video streaming evaluations were conducted. The App-SoC responds to the network ping with a response time

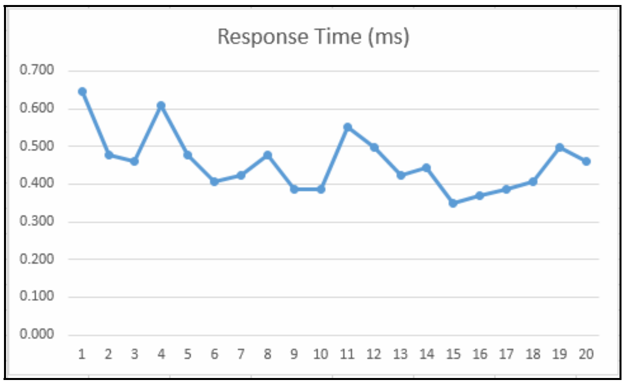


Fig. 11. Network ping response time.

parameter. The results can be seen in Fig. 11. We can see that response times on network ping are handled by App-SoC in 0.3-0.7 ms. Since the App-SoC is supposed to be the one receiving data from the Phy-SoC and it is insensitive to latency, the performance result is sufficient.

For video streaming evaluation, we sent a video from the laptop to the App-SoC and evaluated its output, both sound and display. For a video tester source, we chose the MP4 video format with the following specifications: 322x240 pixels per frame, a frame rate of 25 fps, a data rate of 217 kbps, and total bit rate of 313 kbps. As shown in Fig. 12, display results for video streaming are good. The video is played smoothly, and the audio is clear. This evaluation ensures that the App-SoC is working properly and can be developed further as an application SW development platform. It also means the App-SoC is ready to be connected to the Phy-SoC in order to form the H-MSoC system.

5.3 H-MSoC Evaluation

H-MSoC evaluation was conducted by observing data transfer activity and communications between Phy-SoC and App-SoC. The H-MSoC connection setup is shown in

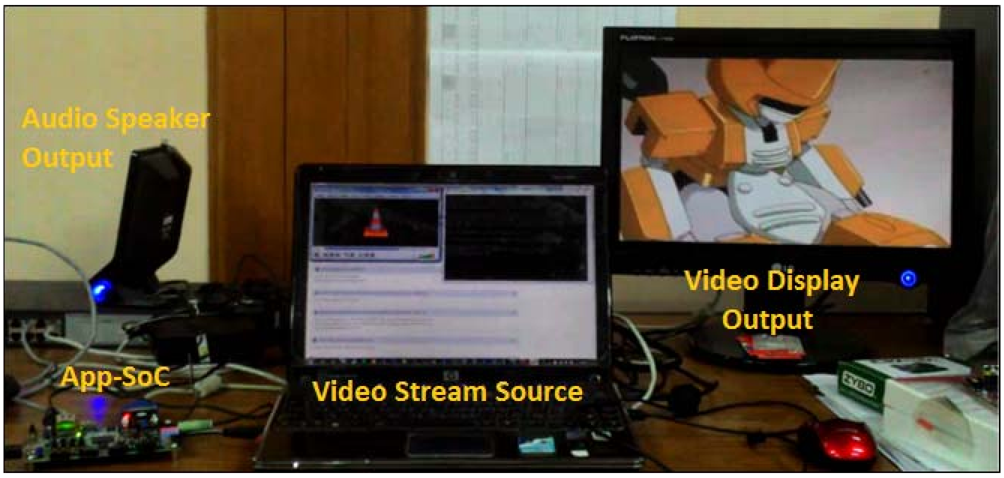


Fig. 12. Video streaming result with clear audio and display.

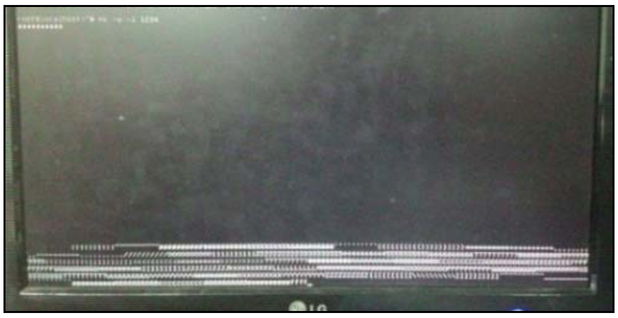


Fig. 13. Display of tranceived data between Phy-SoC and App-SoC.

Fig. 7. A Stratix II EP2S180 DSP development board contains the Phy-SoC design; meanwhile, a Zybo development board contains the App-SoC. Evaluation results in Fig. 13 show that data transfer is successful. Phy-SoC and App-SoC are proven to work together properly to establish collaborative communications as a single H-MSoC. This evaluation leads us to two points about the H-MSoC development approach. First, Phy-SoC and App-SoC are proven to work together properly and to collaborate with each other for internal communications in the H-MSoC system. Second, the H-MSoC architecture is quickly and easily configured, from both the physical HW and application SW development perspectives.

6. Conclusion

In this paper, we propose an alternative SoC architecture that can provide a high-flexibility system in a short development time. It is called a hybrid multi-SoC (H-MSoC) architecture approach. The evaluation results have proven that the proposed H-MSoC architecture and its design methodology can attain the research targets. (1) The H-MSoC architecture is easy to configure and capable of forming a high-flexibility system; (2) on-chip connections are efficient because of SoC separation and localization based on its development layer, either physical HW development or application SW development; and (3) development time is short enough to meet time-to-market constraints.

Acknowledgement

This research was supported by the Indonesia Endowment Fund for Education (LPDP) Scholarship, from the Ministry of Finance, Republic of Indonesia.

References

[1] D. Bertozzi and L. Benini, "Xpipes: a network-on-chip architecture for gigascale systems-on-chip," *IEEE Circuits Syst. Mag.*, vol. 4, pp. 18-31, September 2004. Article (CrossRef Link)

[2] R. Abdel-Khalek and V. Bertacco, "SoCGuard: a runtime verification solution for the functional correctness of SoCs," *Proc. of IEEE/IFIP VLSI System on Chip Conf.*, pp. 49-54, September 2010. Article (CrossRef Link)

[3] H.J. Stolberg, et al., "HiBRID-SoC: a multi-core system-on-chip architecture for multimedia signal processing applications," *Proc. of Design, Automation and Test in Europe Conf. and Exhibition*, pp. 8-13, March 2003. Article (CrossRef Link)

[4] H.J. Stolberg, et al., "HiBRID-SoC: a multi-core SoC architecture for multimedia signal processing," *Proc. of IEEE Workshop on Signal Process. Syst.*, pp. 189-194, August 2003. Article (CrossRef Link)

[5] M. Berekovic, et al., "HiBRID-SoC: a multi-core architecture for image and video applications," *Proc. of Int. Conf. on Image Process.*, pp.III 101-104, September 2003. Article (CrossRef Link)

[6] L. Friebe, et al., "HiBRID-SoC: a system-on-chip architecture with two multimedia DSPs and a RISC core," *Proc. of Int. SoC Conf.*, pp.85-88, September

2003. Article (CrossRef Link)

[7] V.C. Srinivas, et al., "A VLSI system-on-a-chip (SoC) for digital communications," *Proc. of IFIP Int. Conf. on Wireless and Optical Commun. Networks*, pp. 148-152, April 2006. Article (CrossRef Link)

[8] C.C. Yang, et al., "A novel methodology for multi-project system-on-a-chip," *Proc. of IEEE Int. SoC Conf.*, pp. 308-311, September 2011. Article (CrossRef Link)

[9] R. Marculescu, et al., "Outstanding research problems in NoC design: system, microarchitecture, and circuit perspectives," *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Syst.*, vol. 28, pp. 3-21, January 2009. Article (CrossRef Link)

[10] S. Kumar, et al., "A network on chip architecture and design methodology," *Proc. of the IEEE Comput. Soc. Annu. Symp. on VLSI*, pp. 105-112, April 2002. Article (CrossRef Link)

[11] D. Kliem and S.O. Voight, "An asynchronous bus bridge for partitioned multi-SoC architectures on FPGAs," *Proc. of Int. Conf. on Field Programmable Logic and Applicat.*, pp. 1-4, September 2013. Article (CrossRef Link)

[12] C.M. Huang, et al., "Programmable system-on-chip (SoC) for silicon prototyping," *Proc. of IEEE Int. Symp. on Ind. Electron.*, pp. 1976-1981, July 2008. Article (CrossRef Link)

[13] C.M. Huang, et al., "Implementation and prototyping of a complex multi-project system-on-a-chip," *Proc. of IEEE Int. Symp. on Circuits and Syst.*, pp. 2321-2324, May 2009. Article (CrossRef Link)

[14] R. Lemaire, et al., "A flexible modeling environment for a NoC-based multicore architecture," *Proc. of the IEEE Int. High Level Design Validation and Test Workshop*, pp. 140-147, November 2012. Article (CrossRef Link)

[15] eCos. (2015, April 8). eCos Home Page: Introduction [Online]. Available: Article (CrossRef Link)

[16] T. Adiono, et al., "Real-time WiMAX system on chip design, implementation and field test," *Proc. of the IEEJ Int. Analog VLSI Workshop*, pp. 1-5, November 2012.

[17] Xillybus. (2015, April 8). Xillinux: A Linux distribution for Zedboard, ZyBo, MicroZed and SocKit [Online]. Available: Article (CrossRef Link)

Rachmad Vidya Wicaksana Putra received his B.Sc degree in Electrical Engineering major from Institut Teknologi Bandung (ITB), Indonesia, in 2012. He received his M.Sc degree also from ITB in Microelectronics major with a distinction Cum Laude in 2015. He received an Indonesia Endowment Fund for Education (LPDP) Scholarship for his master study and potentially for his Ph.D. program. He was involved in several projects and researches, such as Heart Rate Sensing, Military Digital Radar Display, and Software Defined Radio developments. Currently, he is managing several number of researches in Microelectronics Center ITB, such as RF Sensor and Monitoring System, Internet-of-Things for Smart Home Applications, etc. He is also preparing to start his PhD program at KTH Royal Institute of Technology, Sweden, this year on Neuromorphic Chip Design. His interests are mainly about VLSI, Integrated Circuits and Systems, Neuromorphic, Hardware Arithmetic, Embedded Systems, Digital Signal Processing, and Internet-of-Things.

Trio Adiono received B.Eng. degree in Electrical Engineering major and M.Eng. degree in Microelectronics from Institut Teknologi Bandung (ITB), Indonesia, in 1994 and 1996, respectively. He obtained his Ph.D. degree in VLSI Design from Tokyo Institute of Technology, Japan, in 2002. From 2002 to 2004, he was a research fellow of the Japan Society for the Promotion of Science (JSPS) in Tokyo Institute of Technology. In 2005, he was a visiting scholar at MESA+, Twente University, the Netherlands. He received the "Second Japan Intellectual Property (IP) Award" in 2000 from Nikkei BP for his research on "Low Bit-rate Video Communication LSI Design". He also holds a Japanese Patent on "High Quality Video Compression System". Currently, he is a lecturer at the School of Electrical Engineering and Informatics ITB, a Head of the Microelectronics Center and IC Design Laboratory, Institut Teknologi Bandung. He has co-founded several start-up companies in Japan and Indonesia. He currently serves as a chair of the IEEE SSCS Indonesia Chapter. His research interests include VLSI, Signal and Image Processing, Smart Card, Electronics Solution Design and Integration.