GENERIC ARCHITECTURE PLATFORM FOR MULTIPROCESSOR SYSTEM-ON-CHIP DESIGN

A. Baghdadi, N-E. Zergainoh, D. Lyonnard,
A.A. Jerraya

1. Introduction

This chapter presents a generic architecture model for multiprocessor embedded system-on-chip design. The use of this model as a template in a system design environment allows for efficient generation of multiprocessor architectures. The key characteristics of this model are its great modularity, flexibility and scalability which make it reusable for a large class of application. In addition, it allows for reusability of pre-designed blocks and automatic architecture generation. This chapter focuses on the definition of the architecture model. The feasibility and effectiveness of this architecture model are illustrated by a significant demonstration example.

Current trends in system design methods are towards codesign of mixed hardware/software system targeting multiprocessor system-on-chip. This direction is imposed by the ever increasing system on chip complexity. Additionally time-to-market pressure requires to shorten design time through raising the abstraction level of the specification entry. One of the most important issues in multiprocessor design is the target architecture. The rigidity of the target architecture may lead for a very restraint application field. So in order to build an efficient multiprocessor design flow able to generate complex system architecture from high level specification we need a generic architecture model that can be used as a template by the synthesis process. Modularity, flexibility and scalability are required to have an efficient multiprocessor design flow. What we mean by efficient here is, first of all, feasible and applicable to large application fields. Modularity is needed to master complexity. It allows the design of the different modules separately and provides an assembling scheme. The most common way to achieve modularity is to separate the inter sub-system communication from the behavior when partitioning a system. Modularity allows for reuse of existing modules. Flexibility is required in order to avoid early

The original version of this chapter was revised: The copyright line was incorrect. This has been corrected. The Erratum to this chapter is available at DOI: [10.1007/978-0-387-35409-5_23](https://doi.org/10.1007/978-0-387-35409-5_23)

decisions. It allows the designer to decide quite late in the design process on which technology will be used for the design of each module. When combined with modularity, flexibility allows to change the implementation for a given module at any stage of the design process. For instance a software module may be converted into a hardware module for performance reasons. Scalability allows to adapt the same architecture model for applications of different complexity scale. For instance increasing the number of processors or communication buses.

1.1. Related work

Most of the existing works target monoprocessor architectures, and the most used model in this class is the single CPU, single ASIC target architecture. Even though this architecture is a special and limited example of a distributed system, it is relevant in the area of embedded systems [12]. In this class of work we can cite LYCOS [11], COSYMA [13], CoWare [16] and PMOSS [6]. Another design systems, such as Vulcan [8], TOSCA [1] and COBRA [10], can support more than one ASIC. Several research groups tried to target multiprocessor architectures; we can cite POLIS [2], Chinook [4], SpecSyn [7] and the work led by Wolf and Ti-Yen [17]. In the POLIS [2] system, the target architecture is a system consisting of general-purpose processors combined with a few ASICs and possible other components such as DSPs. The target architecture in the SpecSyn [7] system is a heterogeneous multiprocessor with any number of processors, coprocessors, ASIP or FPGA, communicating through multiple buses. Besides these academic research projects, there were also several industrial trails of open standards and design methodologies [3][15][18][19][20][21] that try to deal with the more and more complex system on chip designs. However, we believe that in all above works, target architectures still lack generic aspects and thus only tackle a restricted application field. In fact, most of the above mentioned systems restrict the kind of components used and/or the communication network to few proprietary and/or specific models designed to be plugged together.

1.2. Contribution

Our long-term objective is the definition of an efficient multiprocessor SoC design environment applicable to large application fields and able to generate multiprocessor architectures. The main contribution here is the definition of a modular, flexible, and scalable architecture model –MFSAM– that may be used for an efficient multiprocessor SoC design flow and handles a large class of applications. The main futures of this model are:

1. As stated above, the model allows for modularity, flexibility and scalability.
2. The model allows for automatic generation of architectures. This point needs some harmonization with the input model and the targeting algorithms.

The model is made of a set of processors communicating through a communication network (Figure 1).

Figure 1. Generic architectural model

A processor may be hardware, software or an IP. The communication network may be of any complexity; it may be made of a single bus or a network with complex protocols. Processors are linked to the common network through communication interfaces. The scalability of this architecture depends on the scalability of the chosen communication network. Modularity is ensured by the use of specific interfaces to link processors to the communication network. This gives the possibility of designing each part of the application separately; even we can include pre-designed modules (IPs). The generic assembling scheme of our model increases largely its modularity. This separation between processor and communication network through specific interfaces also provides high flexibility. In fact, if we change the technology implementing a given module (processor) the only part of the architecture that needs to be changed is the interface of the corresponding module. This chapter focuses on the definition of the architecture model and its use within an architecture generation flow.

2. Architecture in System Design

In this section, we give a brief analysis of system architectures encountered in electronic systems [5][14]. Both embedded system architectures and general-purpose computer architectures are made of three elements: basic components, communication network, and organization scheme.

2.1. Basic components

It consists of basic elements that can be assembled for realizing an architecture implementing a given application. In a hardware/software codesign context, system design components may be classified in three categories: software, hardware, and communication components.

2.2. Communication network

The communication network constitutes the hardware links that support the communication primitives between components. The direct way to connect the components of a system is to have a dedicated communication link between every two communicating components. Between this fully connected network and the bus, there are a wide range of interconnection networks. These networks make a major factor to differentiate modern multiprocessor architectures. Interconnection networks are built up of switching elements and a topology. The topology is the pattern in which the individual switches are connected to other components, like processors, memories, I/O devices, and other switches.

2.3. Organization scheme

It is defined as the method of composition or assembling of the basic components to construct the system architecture. A system architecture can vary from a simple controller to a massively parallel machine. Here what is interesting is the roles played by the different basic elements at the global control level of the system. Thus, we can classify system architectures into two categories: monprocessor architectures and multiprocessor architectures. In addition, communication network and programming model play an essential role in the classification of system architectures. A monprocessor architecture consists of one CPU and one or more ASICs. This scheme follows a master–slave synchronization scheme where the top controller acts as a main processor in charge of coordinating the activities of the other components which are acting as co-processors: although very useful in several application domains, the single processor architecture can only provide a restricted performance capability because of the lack of true parallelism.

A multiprocessor architecture allows more flexibility and improved performances thanks to the distribution of computation among processors. However, it is much more difficult to handle due the parallelism.

3. MFS Architecture Model

The new architecture model that we propose here allows for modularity, flexibility and scalability. This model is based on the analysis results of existing architectures in system design (done in section II). We chose from that huge design space the most appropriate elements that fit our needs. These needs are –in addition to modularity, flexibility and scalability– the adaptability for multiprocessor SoC design and the possibility of an automatic generation of the final architecture. Also, we are mostly interested by architectures suitable for embedded systems on chip which is very relevant in many potential application fields. In this section, we will develop the new architecture model we propose.

3.1. Components

The components of our architecture model belong to the three essential categories: software, hardware, and communication components. It consists of software processors, hardware blocks, memories, and communication interfaces.

3.1.1. Software processors. In this category we can include off-the-shelf microprocessors, microcontrollers, DSPs, and application specific processors (i.e. ASIPs). The attributes requested by our architecture model for this category of components can be verified by most of them. The processor should be able to communicate through an external memory bus (synchronous or asynchronous). It should also be able to handle external interrupts. Additional I/O ports and internal peripherals may be useful but not necessary. The processor-memory interface and interrupts are essential for building the processor-network communication interface.

3.1.2. Hardware blocks. To add a pre-designed hardware block (i.e. IP) following our architecture model it must be provided with its communication interface adapted the to the network. The reason for that is that there is no normalized bus interface for such components.

3.1.3. Memory. Memories are essential components in system design. They are integrated in our architecture model as pre-designed blocks. Thus, to use a memory block only its access scheme and access timing are required. A memory controller with an address decoder must be added to adapt the processor bus or network bus to the memory (local and global memories).

3.1.4. Communication Interfaces. Communication interfaces provide bridges between previous components and the communication network. For software processors, these interfaces can be generated according to both: the processor attributes (e.g. memory bus, interrupts and I/O ports) and communication network. For hardware blocks, communication interfaces are often built in or provided with.

3.2. Communication Network

Although a wide range of communication architecture exists, only few ones are suitable in embedded systems on chip design methods. At a high abstraction level, communications are established through abstract channels, however at the physical level these abstract channels should be transformed in physical wires. The more commonly supported communication network is the point-to-point network. The great advantage of this network that it already leads for an automatic architecture generation from system level specification. Such a network achieves great performances but has a high cost. A more sophisticated network that can also satisfy same conditions as the previous one is the hierarchical buses network. However, it is more difficult to be supported by system design methods. These two networks meet quite well the increasing complexity in embedded systems on chip.

3.3. Organization scheme

As figure 1 shows, the proposed architecture model has a multiprocessor organization scheme. The model is made of a set of components communicating

through a communication network. The control is distributed among the different components that constitute the architecture. Off course, the components and communication network specifications are those mentioned above.

4. Architecture Generation Flow Based on MFSAM

In this section, we present the overall flow for architecture generation based on the MFSAM presented in section 3. As we stated before, the final aim of this work is to build an efficient multiprocessor SoC design tool able to generate a complex application specific architecture, and that is for a large class of applications. The modularity, flexibility and scalability of the proposed model lead for wide application domain, but require to fix a large number of parameters at the design of each application. Thus, in order to make practical the use of the proposed model as template for multiprocessor SoC design tool we chose to drive from it a set of architecture platforms, each of which target a set of applications (i.e. an application filed), see figure 2. This specialization of the architecture model should also assist the designer in the implementation choices of the application.

Figure 2. Architecture platforms based on MFSAM

In addition, as all of these architecture platforms are derived from the same architecture model, the same architecture generation flow and generation tool can support all of them. The architecture generation tool will have at its entry an architecture platform (depending on the application field) and the application specific parameters that will configure the platform. These application specific parameters may be the results of a system level synthesis tool (e.g. a hardware/ software codesign tool). The architecture generation flow is shown figure 3.

Figure 3. MFSAM-based architecture generation flow

Further illustrations about the implementation and efficiency of this flow will be given in the next section through a demonstration example.

5. Demonstration Example

In this example we show the feasibility and prove the efficiency of the proposed architecture model. A multiprocessor architecture platform based on MFSAM is proposed. Then, several application-specific architectures are generated, and that is for two application examples. The validation of the generated architectures was done by cosimulation. The results analysis shows the effectiveness of the proposed architecture model.

5.1. A multiprocessor architecture platform based on MFSAM

As we already dispose of the development kits of the two software processors ARM7 and MC68000, we considered an architecture platform based on these two processors. The architecture platform constitutes of the following components: two types of software processors (ARM7 and MC68000), local memories and communication interfaces. The communication network is a point-to-point network. The block diagram of this architecture platform is given figure 4.

Figure 4. A multiprocessor architecture platform

The free parameters of this architecture platform are the number of software processors, the number of I/O channels for each software processor, the interconnections between them and the interconnections with external systems. These parameters show the scalability of the platform and therefor enable the design of application specific architectures of different scales. In fact, as we mentioned in section IV, the application specific architecture is generated thanks to an architecture generation tool. This tool is the subject of other work in our research group. Actually, communication interfaces are almost automatically generated as hardware blocs according to the processor attributes and to the application parameters (communication channels). In order to validate the generated architecture, we need a cycle accurate executable architecture that can run the application. To that end, we used a cosimulation approach [9]. In this approach software processors are replaced by cycle accurate ISSs (ISS + BFM). All other parts of the architecture are modeled

in VHDL RTL and executed by a VHDL simulator (e.g. VSS). The cosimulation tool ensures the interconnection and synchronization of the running simulators for coherent execution of the overall system.

5.2. Application examples

Many applications can be mapped on the architecture platform presented above. Of course performance and cost aspects must be taken into account. We have used this platform to implement two applications: a Packet Routing Switch and an IS-95 CDMA mobile station. As mentioned in the section IV, to map an application to an architecture platform we have to fix the application specific parameters. Then, the architecture generation tool takes in charge the configuration of the platform and the generation of the application specific architecture.

5.2.1. Packet Routing Switch. It constitutes a powerful solution for large frame or cell switching systems. The version we present here is a simplified one; it consists of two input controllers and two output controllers. Each of the controllers handle one communication channel, and the communication links between input and output controllers is configured by an external signal to be direct or switched. We chose two architectures to implement this switch, one with four processors (two ARM7 and two MC68000), and the other with only two processors (one ARM7 and one MC68000). The configuration of the architecture platform with these parameters led for the two architectures shown in figure 5.

Figure 5. Two implementations of the Packet Routing Switch

5.2.2. IS-95 CDMA mobile station. In an IS-95 CDMA cellular phone system, the mobile station contains a call processor, a QCELP vocoder, and two modems (receiver and transmitter). Many architectural solutions are conceivable, we chose a one with four processors. The chosen architectural solution is shown figure 6 (we did not implement the call processor).

Figure 6. Implementation of IS-95 CDMA mobile station

The difference between this architecture and the one with four processors presented in figure 5 is the I/O for each processor, the interconnection between processors, and the external I/O.

As we stated in the V.1, these architectures have been validated by cosimulation at RT level. Figure 7 shows the cosimulation architecture of the 4-processors architecture implementing the Packet Routing Switch shown figure 5.

Figure 7. Cosimulation architecture of the packet routing switch

5.3 Analysis of the results

Many other applications of different scales can be mapped on this single architecture platform. For this example, the architecture generation was done manually and it took about 1 day to generate one architecture. However, when the architecture generation tool will be accomplished, this time will drop significantly and it will be reduced to the time to capture the application specific parameters, i.e. few minutes. This example illustrates the feasibility and the efficiency of our architecture model. With this model, multiprocessor architectures become much easier to handle. We illustrated how generation of application specific architecture can become simple and very quick. Note that the architecture model we propose in this chapter is far more generic than the architecture platform we presented in this example. This leads obviously for a huge application field. Other kinds of software processors (and DSP cores) can be integrated and used in the same manner. This shows the great flexibility and modularity of the proposed architecture model.

The modularity of our architecture model appears in the organization scheme, which consists of separated modules communicating through a communication network. It separates the behavior from the inter sub-system communication. In addition, each module can be designed separately, an assembling scheme is provided to efficiently connect them and to enable the reuse of existing modules. This assembling scheme is quite structured and permits easily the reconfiguration of the architecture. Thus, technology choice can be done late in the design process which leads to a great flexibility. The scalability of our architecture model is also achieved thanks to the assembling scheme. It depends on the scalability of the chosen communication network. This scalability allows to adapt the proposed architecture model for applications of different complexity scale. For instance increasing the number of processors or communication buses.

It is worth noting that we consider the software processor cores without their peripheral components. This fact makes our approach more generic and optimized as various issues emerge when considering these peripheral components (software targeting, communication interfaces...etc).

Performance aspects of the architectures generated according to the proposed architecture model are subject of our running work.

6. Conclusions

In this chapter, we presented a generic architecture model for multiprocessor embedded system-on-chip design. The proposed model is modular, flexible and scalable. It permits an efficient generation of multiprocessor architectures for embedded systems on chip. This work forms a promised step towards the definition of an efficient multiprocessor SoC design environment applicable to a large application domain. The key point when defining such an environment is to define the architecture model that will fix the class of architectures handled by the system. The model allows for automatic architecture generation. This chapter focused on the definition of the architecture model. The feasibility and effectiveness of this architecture model was illustrated by a significant demonstration example.

References

- [1] S. Antoniazzi, A. Balboni, W. Fornaciari, and D. Sciuto. "Hardware/Software codesign for embedded telecom systems". In proceedings of ICCD'94, October 1994.
- [2] F. Balarin, E. Sentovich, M. Chiodo, P. Giusto, H. Hsieh, B. Tabbara, A. Jurecska, L. Lavagno, C. Passerone, K. Suzuki, A. Sangiovanni-Vincentelli. *Hardware-Software Co-design of Embedded Systems – The POLIS Approach*. Kluwer Academic Publishers, 1997.
- [3] R.A. Bergamaschi, and W.R. Lee. "Designing Systems-on-Chip Using Cores". In the 37th Design Automation Conference, June 2000.
- [4] P.H. Chou, R.B. Ortega, and G. Borriello. "The Chinook hardware/software co-synthesis system". In the 8th International Symposium on System Synthesis, September 1995.
- [5] D. Culler and J.P. Singh with A. Gupta. "Parallel Computer Architecture: A Hardware/Software Approach". Morgan Kaufmann Publishers, August 1998.
- [6] G. H.J. Eikerling, W. Hardt, J. Gerlach, and W. Rosenstiel. "A Methodology for Rapid Analysis and Optimization of Embedded Systems" International IEEE Symposium and Workshop on ECBS, D-Friedrichshafen, pp. 252-259, March 1996.
- [7] D.D. Gajski, F. Vahid, and S. Narayan. "A system-design methodology: Executable-specification refinement". In the European Conference on Design Automation, February 1994.
- [8] R.K. Gupta, C. Coelho, and G. De Micheli. "Synthesis and simulation of digital systems containing interacting hardware and software components". In the 29th Design Automation Conference, June 1992.
- [9] F. Hessel, Ph. Le Marrec, C. Valderrama, M. Romdhani, and A.A. Jerraya. "MCI - Multilanguage Distributed Co-Simulation Tool". Chapter in "Distributed and Parallel Embedded Systems", DIPES'98 organized by IFIP WG10.3/WG10.5, edited by F. Rammig, Kluwer Academic Publishers, 1999.
- [10] G. Koch, U. Weinmann, U. Kebschull, and W. Rosenstiel. "System prototyping in the COBRA project". *Journal on Microprocessors and Microsystems*, Elsevier, vol.20, no.3, May 1996.
- [11] J. Madsen, J. Grode, P.V. Knudsen, M.E. Petersen and A. Haxthausen. "LYCOS: the Lyngby Co-Synthesis System". *Design Automation for Embedded Systems*, vol.2, n.2, 1997.
- [12] G. De Micheli. "Computer-aided hardware-software codesign". *IEEE Micro*, 14(4):10-16, August 1994.
- [13] A. Österling, Th. Benner, R. Ernst, D. Herrmann, Th. Scholz, and W. Ye. "Hardware/Software Co-Design: Principles and Practice". Chapter: "The COSYMA System". Kluwer Academic Publishers, 1997.
- [14] D.A. Patterson and, J.L. Hennessy. "Computer Organization and Design: The Hardware/Software Interface". Second Edition, Morgan Kaufmann Publishers, 1997.
- [15] A. Rincon, W. Lee and M. Slatery. "The Changing Landscape of System-on-a-Chip Design". *IBM MicroNews*, 3rd Quarter 1999, Vol.5, No.3, IBM Microelectronics.
- [16] K. Van Rompaey, D. Verkest, I. Bolsens and H. De Man. "CoWare – A Design Environment for Heterogeneous Hardware/Software Systems". In *EURO-DAC'96*, September 16-20, 1996.
- [17] T.Y. Yen and W. Wolf. "Communication synthesis for distributed embedded systems". In *International Conference on Computer Aided Design*. IEEE Computer Society Press, 1995.
- [18] "AMBA Specification Overview", ARM, <http://www.arm.com/Pro+Peripherals/AMBA>
- [19] "Blue Logic Technology", IBM, <http://www.chips.ibm.com/bluelogic>
- [20] "TheCore ConnectTM Bus Architecture" IBM, 1999, http://www.chips.ibm.com:80/news/1999/990923/pdf/corecon128_wp.pdf
- [21] VSI AllianceTM Architecture Document, Version 1.0, VSI Alliance, 1997, <http://www.vsia.com/>