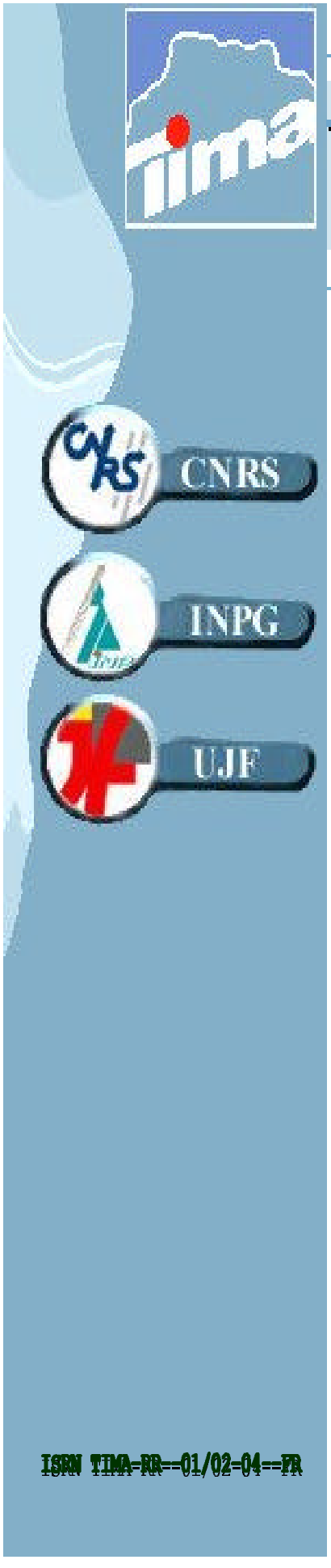

The TIMA Lab. Research Reports Collection.

ISSN 1292-8062

Automatic Generation of Application-Specific Architectures for Heterogeneous Multiprocessor System-on-Chip

D. LYONNARD[1], S,. YOO1, A. BAGHDADI[1], A.A. JERRAYA[1]

1 - TIMA Laboratory, 46 avenue Félix Viallet, 38 000 Grenoble France

ISSN TIMA-RR--01/02-04--FR

TIMA Laboratory,46 avenue Félix Viallet,

38 000 Grenoble, France

Automatic Generation of Application-Specific Architectures for Heterogeneous Multiprocessor System-on-Chip

Damien Lyonnard Sungjoo Yoo Amer Baghdadi Ahmed A. Jerraya
SLS Group, TIMA Laboratory
46 Avenue Félix Viallet,
38031 Grenoble, France
{Damien.Lyonnard,Sungjoo.Yoo,Amer.Baghdadi,Ahmed.Jerraya}@imag.fr

Abstract

We present a design flow for the generation of application-specific multiprocessor architectures. In the flow, architectural parameters are first extracted from a high-level system specification. Parameters are used to instantiate architectural components, such as processors, memory modules and communication networks. The flow includes the automatic generation of communication coprocessor that adapts the processor to the communication network in an application-specific way. Experiments with two system examples show the effectiveness of the presented design flow.

1. Introduction.

To accommodate the ever increasing performance requirements of application domains such as xDSL, networking, wireless, game applications, etc., multiprocessor SoCs are more and more required. Such multiprocessor systems should be specific to each of the application domains in the following aspects:

- Kinds of (application-specific) processor: μPs, DSPs, ASIPs, and coprocessors (DCT, Viterbi decoder, etc.).
- Communication components: memory (e.g. multi-bank memory [8][19], special stream buffers [10]), peripherals [20], etc.
- Communication networks: shared bus [15][18][23], circuit switching [16], packet switching [9], etc.

As the multiprocessor architectures require **heterogeneous processors** (for application-specific optimization), **high-performance complex communication networks** [9] [17] and **sophisticated communication protocols** [24] (e.g. broadcasting, multi-master buses, etc.), architecture generation demands significant design efforts. Unfortunately, for the architecture generation, current design practices allow only limited automation [3][26] and, for most parts, designers resort to manual architecture design, which is time-consuming and error-prone. Such a manual process can hardly afford design space exploration in heterogeneous multiprocessor architectures design.

In our work, we aim at automating the architecture generation step. To do that, **the architecture is generated from a high-level system description** thereby freeing the designer from the interface details required for the architecture design. Thus, the automation enables the designer to focus on more valuable design decisions such as processor/communication component allocation and behavior/communication mapping/scheduling in multiprocessor architectures. In our work, the architecture generation is based on **instantiation of generic multiprocessor architecture templates** and on the **automatic generation of communication coprocessors**.

This paper is organized as follows. In Section 2, we give a review of architecture generation for multiprocessor systems. We explain generic architecture templates and communication coprocessors in Section 3. In Section 4, we introduce our design flow and describe the high-level system specification for architecture generation. In Section 5, we address the architecture generation flow. In Section 6, we give experimental results. The design flow is evaluated in Section 7 before concluding in Section 8.

2. Previous Work.

The main difference between **classical** multiprocessor architectures [5] and multiprocessor SoC architectures is that the multiprocessor SoC architectures have specific applications while the classical architectures have general purposes. This means that the two kinds of architecture are significantly different. In SoC multiprocessor architecture, since the specific application has tight design constraints (e.g. low area and power consumption and high performance), application-specific optimization of the architecture is necessary. Thus, we have to use various kinds of processors (to use a processor specific to the application, e.g. usage of a DSP for voice processing) and the communication networks **can not have regular structures** (from one application to another) to meet the application-specific requirements of communication (e.g. circuit switch network in multimedia applications and CAN bus in automotive applications).

The multiprocessor architecture is also different from the conventional μP/coprocessors architecture as illustrated in Figure 1, where communication between processors is based on master (μP)/ slave (coprocessors) relationship. Interfaces of coprocessors are

DAC 2001, June 18-22, 2001, Las Vegas, Nevada, USA.

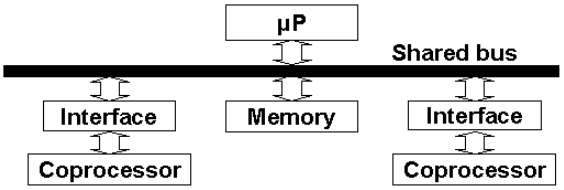

Figure 1. A conventional μP/coprocessors architecture.

generally made of memory-mapped registers and can generate interrupts to the processor. Multiprocessor architectures introduce two new difficulties:

(1) The architecture may include more than one (application-specific, i.e. heterogeneous) master processor.

(2) The inter-processor communication may require more sophisticated networks than a simple shared bus.

To design such an architecture, in [26], a method is presented to generate multiprocessor architectures using point-to-point module interconnection and a *rendez-vous* protocol. In [12], [18], and [23], system-bus based architectures for IP integration are addressed. In [23], a multi-master system bus based on TDMA arbitration is presented. Custom multiprocessor architectures [1][16] enable high communication bandwidth and application-specific processor usage. Specifically, in terms of computation, they have application-specific computation modules (e.g. DSPs, ASIPs, etc.). In terms of communication media, they have fixed networks (e.g. shared bus [1][16]) which are not scalable (e.g. the shared bus is not scalable in terms of communication bandwidth). In general, the architectures used in conventional methods of multiprocessor SoC design [26][18][23][12] and custom multiprocessor architectures [1][16] are not flexible enough to meet the requirements of different application domains (e.g. only point-to-point or shared bus communication is supported.) and not scalable enough to meet different computation needs and different complexity of various applications. In this paper, we present generic architecture templates that are flexible and scalable.

In the case of classical multiprocessor architectures, usage of **communication coprocessor** is used to free the processor from communication tasks [14]. The complexity of communication coprocessor ranges from a simple DMA controller [11], a superscalar RISC processor [14], dual processors [22], even to a processor identical to the one for computation (Intel Paragon [21]).

In recent methods of embedded systems architecture construction [4][6][18][26], processor templates [4][26], more generally, wrappers [6][18] are used to integrate processors and IP blocks into the architectures. In most of those methods, wrappers are manually designed to adapt processors (or IP blocks) to standardized communication protocol (e.g. rendez-vous in [26]) or shared on-chip bus protocols [6][18].

Compared to the communication coprocessors of classical architectures, those of SoC architectures, i.e. wrappers should be optimized for both processors and application protocols/interconnections. In this paper, we present a method of architecture generation based on automatic generation of application-specific wrappers.

3. Architecture Templates.

We use generic multiprocessor architecture templates defined in [2] and exemplified in Figure 2. The architecture template consists of four types of elements: processors (including application-specific coprocessors), communication coprocessors, IP components (memory modules, bus bridges, peripherals, etc.), and communication network(s). In the architecture templates, the interconnection of processors and communication components to the

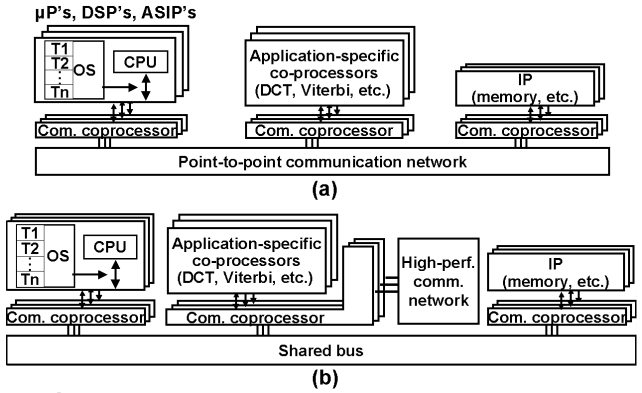

Figure 2. Generic multiprocessor architecture templates.

communication network(s) is accomplished by instantiating communication coprocessors between them.

Architecture templates are parameterized for the four types of elements. Thus, **architecture generation is customizing the architecture elements with parameters**. A generated architecture can be considered as an instance of a generic multiprocessor architecture template.

The main parameters of each type of architectural element are as follows.

(1) Processor parameters:

- Processor types (Pentium, ARM7, 68000, DCT, Viterbi decoder, etc.)
- Number of processors for each processor type
- Memory size and memory map
- Local bus configuration parameters (e.g. data/address multiplexing)

(2) Parameters of communication coprocessor

- Interconnection(s) of processor (or IP component) with other processors or IP components
- Communication protocol and parameters for each interconnection: e.g. master/slave relation[1], data types of transferred data, fifo size, etc.
- For each channel, allocated address bank(s) and interrupt usage/level.

(3) Parameters of IP component

- Size of data storage, allocated address bank(s), etc.

(4) Parameters of communication network

- Dynamic or static interconnections among processors and IP components via the communication network(s).

Figure 2 shows generic multiprocessor architectures. The first one (Figure 2 (a)) may include several heterogeneous processors (general purpose processors and/or application-specific processors), and a set of specific IP devices (shared memories, FIFOs, etc.). A fully connected point-to-point network is used as communication network. The second example of generic multiprocessor architecture (Figure 2 (b)) contains a system-bus instead of the point-to-point network of Figure 2 (a).

Figure 3 shows the general model of the communication coprocessor presented in this paper. The communication coprocessor consists of two parts: processor adapter, i.e. processor-specific

[1] We use the same definitions of master/slave ports as in [4] where a master port is defined to be a port that initiates the communication and a slave port is defined to be a port of which the communication is initiated only by its corresponding master port.

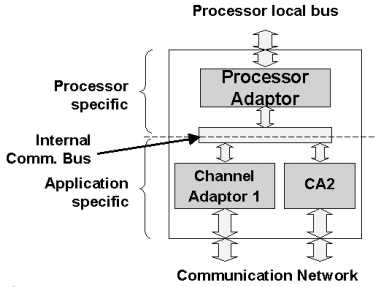

Figure 3. Communication coprocessor.

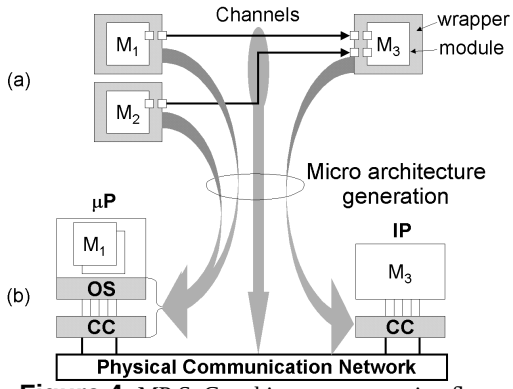

Figure 4. MP SoC architecture generation flow.

part and channel adapters, i.e. application-specific part. The two parts are interconnected through an internal communication bus. This model allows to: (1) adapt the heterogeneous features of processor to the communication network, and (2) enable easy implementation of different communication protocols such as multi-master bus or broadcasting. The details of communication coprocessors will be given in Section 5.

4. System Specification and Refinement in our MP SoC Design Flow.

We represent the system with a hierarchical network of **modules**. A module consists of **behavior** and **port(s)**, i.e. behavior and communication are separately described. Modules are connected with each other by connecting their ports via **communication channels** (in short, channels). We use a **wrapper** when the module has different communication behavior (e.g. different communication protocol) than the communication channel(s).

To represent system communication, we use three abstraction levels of communication: system level, macro-architecture level, and micro-architecture level. At the system level, modules communicate with each other exchanging **messages** over the system level channel. There is no specific communication protocol for system level channels. Thus, system level channels provide only two kinds of functions, **send** and **receive** for the ports to access them. At the macro-architecture level, each channel is given its own communication protocol (e.g. FIFO, handshake, etc.) and parameters (e.g. FIFO size). Macro-architecture level channels provide protocol-specific functions (e.g. fifo_available, fifo_write, etc.) for the ports to access them. At the micro-architecture level, communication is represented at cycle accurate level.

Figure 4 shows a simple view of multiprocessor SoC architecture generation. In our flow, architecture generation means implementing wrappers and communication channels from a **macro-architecture** down to a **micro-architecture**. As shown in Figure 4 (a), in a macro-architecture, modules (with wrappers) are connected with each other via channels. After the architecture generation, a micro-architecture (shown in Figure 4 (b)) is implemented where wrappers in the macro-architecture are implemented in the forms of hardware modules (communication coprocessors (CCs)) and/or software modules (operating systems (OSs)) depending on whether the module is mapped on a processor or on a HW component (e.g. HW IP). In this paper, we focus on the generation of communication coprocessors. For the details of OS generation, the readers are referred to [7].

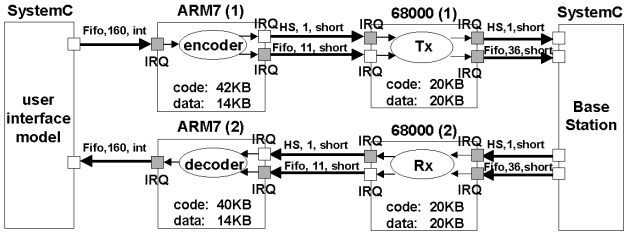

Figure 5. A macro-architecture specification of IS-95 system.

Compared with [3], we present a more extensive method for multiprocessor architecture generation in the three points mentioned in Section 1: (1) generic architecture templates explained in Section 3, (2) architecture generation from the macro-architecture specification will be explained in the rest of this section, and (3) architecture generation based on the generation of application-specific communication coprocessors will be detailed in Section 5.

We generate a micro-architecture from a macro-architecture specification. Note that, at the macro-architecture level, allocation of processors and mapping of behavior and communication have been done. Figure 5 shows a macro-architecture specification of an IS-95 CDMA cellular phone system [25][27]. In the figure, rectangles represent modules, ovals in them represent their behavior parts, and small squares represent their ports. Arrows represent channels. In the case of Figure 5, since wrappers are not yet required, they are not shown in the figure.

The IS-95 system consists of voice encoder and decoder, CDMA modem transmitter (Tx) and receiver (Rx), and two simulation models of base station and user interface. In this case, we assume that an architecture template consisting of two ARM7 processors, two 68000 processors and point-to-point communication network with handshake protocol, is used. Figure 5 shows that encoder and decoder are mapped on two ARM7 processors (ARM7 (1) and (2), respectively) and Tx and Rx are mapped on two 68000 processors (68000 (1) and (2), respectively). In this case, since we use a point-to-point network in the architecture template, each macro-architecture channel in Figure 5 corresponds to a channel at the micro-architecture level. The macro-architecture channel has parameters such as communication protocols. In this example, FIFO and Handshake are assigned to the macro-architecture channels. Protocols of micro-architecture channels in the point-to-point network (in this case, handshake) is not shown in the figure. The parameters of communication protocol, e.g. size of data storage, and data type are also specified. Each port has also parameters such as master/slave, interrupt usage, interrupt levels, and allocated addresses. In Figure 5, we denote master ports with shaded squares and slave ports with blank squares. The interrupt levels and allocated addresses are not shown in the figure due to the space limit.

Modules	CPU	Memory Size	Comm. Channels	Comm. Protocols	Channel Size	Port Type	Allocated Address
Encoder	ARM7 40 MHz	ROM: 42KB RAM: 14KB	UIF ⇒ Enc	FIFO	160, int	Master, IRQ	0x7000
			Enc ⇒ Tx	FIFO	11, short	Master, IRQ	0x7004
			Enc ⇒ Tx	Handshake	1 short	Slave, IRQ	0x7008
Decoder	ARM7 40 MHz	ROM: 40KB RAM: 14KB	Dec ⇒ UIF	FIFO	160, int	Master, IRQ	0x7000
			Rx ⇒ Dec	FIFO	11, short	Master, IRQ	0x7004
			Rx ⇒ Dec	Handshake	1, short	Slave, IRQ	0x7008
Tx	M68000 20 MHz	ROM: 20KB RAM: 20KB	Enc ⇒ Tx	FIFO	11, short	Slave, Level 3	0x9000
			Enc ⇒ Tx	Handshake	1, short	Master, Level 4	0x9002
			Tx ⇒ HWM	FIFO	36, short	Master, Level 5	0x9004
			Tx ⇒ HWM	Handshake	1, short	Master, Level 6	0x9006
Rx	M68000 20 MHz	ROM: 20KB RAM: 20KB	Rx ⇒ Dec	FIFO	11, short	Slave, Level 3	0x9000
			Rx ⇒ Dec	Handshake	1, short	Master, Level 4	0x9002
			HWM ⇒ Tx	FIFO	36, short	Master, Level 5	0x9004
			HWM ⇒ Tx	Handshake	1, short	Master, Level 6	0x9006
UIF model	(SystemC)	-	UIF ⇒ Enc	FIFO	160, int	Slave	-
			Dec ⇒ UIF	FIFO	160, int	Slave	-
Base Station	(SystemC)	-	Tx ⇒ HWM	FIFO	36, short	Slave	-
			Tx ⇒ HWM	Handshake	1, short	Slave	-
			HWM ⇒ Rx	FIFO	36, short	Slave	-
			HWM ⇒ Rx	Handshake	1, short	Slave	-

Figure 6. Parameters extracted from the IS-95 specification.

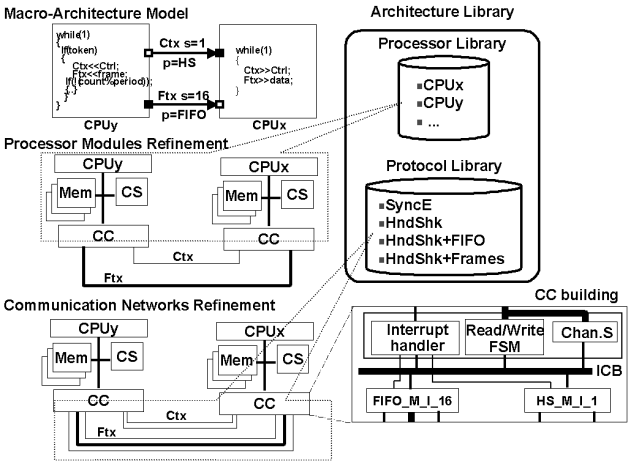

Figure 7. Architecture generation flow.

To instantiate a micro-architecture from the architecture template, parameters are extracted from the macro-architecture specification. Figure 6 shows an example of parameters extracted from the specification of the IS-95 system in Figure 5. The parameters are results of design decisions made by the designer or by automatic tools. The generation of such parameters is beyond the scope of this paper.

5. Automatic Generation of Application Specific MP SoC Architectures.

5.1 Detailed Flow of Architecture Generation.

Figure 7 shows the detailed architecture generation flow for the case of IS-95 system (shown in Figure 5). For the generation of the micro-architecture, we use a **processor library** and a **protocol library**.

The former consists of a list of processor local template architectures which consist of four types of elements: processor cores, local buses, local IP components (e.g. local memory, address decoder, coprocessors, etc.) and processor adapters. Each element has simulation and synthesis/implementation models: e.g. for a processor, an instruction set simulator as simulation model and a hard core like a layout macro as implementation model. Simulation models are used to construct co-simulation models and synthesis/implementation models are used to construct the multiprocessor architecture for synthesis. To allow application-specific instantiation of simulation and synthesis/ implementation models, the models have generic parameters (e.g. memory size).

The latter consists of a list of channel adapters. Each channel adapter has simulation, estimation, and synthesis models that are parameterized (by the channel parameters, e.g. direction, storage size and data type) as the elements in the processor library. The estimation models enable the performance/cost estimation of communication protocol implementation in terms of HW area, power consumption, run-time, utilization, etc.

In Figure 7, we show, in the macro-architecture specification, two modules mapped on two processors, CPUx and CPUy. In the figure, two communication protocols, FIFO and Handshake, are assigned to two communication channels. The sizes of data for the protocols (1 and 11, respectively) are also shown. More details of specification and parameters extraction are given in Section 4.

As mentioned in Section 3, a micro-architecture is generated from an architecture template by setting the parameters for the four types of elements: **processor local architectures**, **communication coprocessors**, **IP components** and **communication network(s)**. To instantiate a processor local architecture from the processor library, parameters such as size of local memory and channel-access addresses are used. The communication coprocessor adapts the processor to the communication network and it implements the protocols of communication channels. Thus, to instantiate communication coprocessors, both processor and protocol libraries are used.

The **processor adapter** in the communication coprocessor is selected from the processor library and configured with the corresponding architecture parameters (allocated addresses, number of interrupts, and their levels). It performs (1) channel access detection by address decoding (by a Read/Write FSM exemplified inside of a communication coprocessor of Figure 7), (2) channel selection, and (3) interrupt management. The address decoder in the processor adapter is configured by a look-up table generated from the parameters for allocated addresses (e.g. 0x7000, 0x7004, and 0x7008 for ARM7 as shown in Figure 6). The processor adapter enables a channel adapter by sending to it an enable signal when the channel adapter, i.e. its channel is accessed by the processor. Since enable signals are used, the internal communication bus (ICB) of the communication coprocessor, shown in Figure 3 and 7, does not have address lines, which gives an easy and area-efficient implementation of communication coprocessors. The interrupt controller is configured from the architecture parameters related with interrupts (e.g. the number of interrupts, their levels).

The **channel adapter** implements (1) the communication protocol of the macro-architecture level channel and (2) the protocol of the connected communication network at the micro-architecture level. There are two types of channel adapter depending on the direction of channel communication: input/output channel adapter. For each channel in the macro-architecture level specification, a pair of channel adapters are selected from the protocol library with the parameter of the communication protocol (type of communication protocol) and they are configured with the architecture parameters (e.g. input/output, master/slave, data type, buffer size, interrupt usage).

5.2 Communication Coprocessors Generation.

Figure 8 shows examples of processor local architectures and communication coprocessors (two shaded rectangles) instantiated for

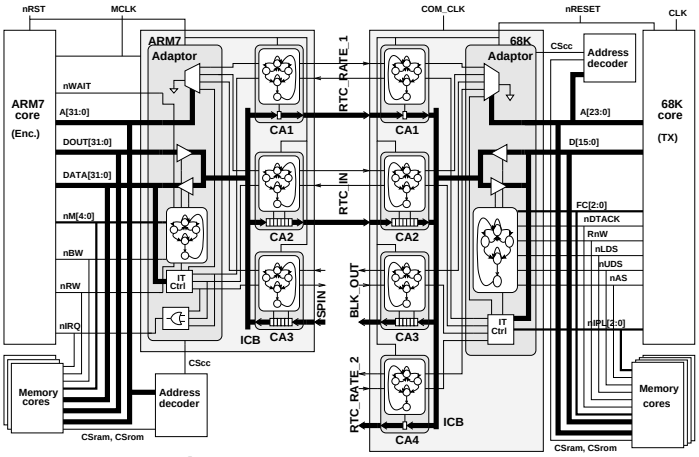

Figure 8. A generated architecture.

the ARM7 processor of encoder and the 68000 processor of Tx with the parameters of Figure 6. In the example, the local architecture consists of a processor core (ARM7 or 68000), several memory cores, and an address decoder. A processor adapter and channel adapters (in each communication processor) are connected via the internal communication bus (ICB).

In Figure 8, channel adapters are instantiated with the parameters of Figure 6. For instance, in the communication coprocessor of the ARM7 processor, output channel adapter 2 (CA2 in Figure 8) is instantiated with the following parameters of Figure 6.

- Channel interconnection: Enc $\Rightarrow$ Tx (point to point)
- Communication protocol: FIFO (macro-architecture protocol) and handshake (the protocol of communication network in the micro-architecture)
- Channel size: 11, `short` (16 bit width)
- Port type: Master, IRQ
- Allocated address: 0x7004 on ARM7

The channel interconnection parameter directs the interconnection of CA2 with its counterpart in the communication coprocessor of 68000 processor. The channel adapter, CA2 implements (1) the communication protocol of macro-architecture channel, in this case, FIFO and (2) that of connected communication network, in this case, the handshake protocol of the point-to-point network. The channel size of FIFO, 11 data items of data type `short`, determines the size of data storage in the output channel adapter, CA2 and the width of data signal (16 bit width) between the two connected channel adapters. To connect two channel adapters via the point-to-point communication network with handshake protocol, two control signals (Req and Ack, they are not shown in the figure due to the space limit) of handshake protocol are used between the two channel adapters as well as one data signal (in this case, 16 bit data) for the data transmission of FIFO as shown in Figure 8.

The port type (in this case, Master and IRQ) determines the usage of interrupt. The interrupt request signal of the channel adapter is connected to the interrupt controller of the processor adapter. The allocated address (in this case, 0x7004 on ARM7) is used to configure the address decoder (in the processor adapter) to enable CA2 when the address, 0x7004 is accessed by the ARM7 processor. The instantiation of module adapters and channel adapters works in the same way for the other communication coprocessors.

6. Experiments.

In the experiments, we use two system examples: a packet routing switch system, and the IS-95 system.

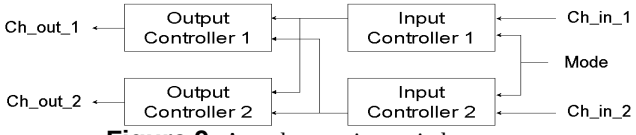

Figure 9. A packet routing switch system.

6.1 Packet Routing Switch System.

The packet routing switch system [13] constitutes large-frame or cell-switching systems. Figure 9 shows the block diagram. It consists of two input controllers (ICs) and two output controllers (OCs). Each of the controllers handles one communication channel. The communication links between input and output controllers are configured by an external signal (Mode in the figure).

We generate two target architectures: one consisting of two ARM7 processors and two 68000 processors and the other consisting of one ARM7 processor and one 68000 processor. We use point-to-point interconnection in the architectures. For two cases of architectures, we generate four and two communication coprocessors, respectively. Further details related to this case study are available in [2].

Cycle-accurate co-simulations of the architectures allowed to measure the latency of communication coprocessor: 2 (4) clock cycles for 68000 write (read) operations[2] and 4 (4) clock cycles for ARM7 write (read) operations.

HW synthesis of the communication coprocessors gives 5,156 gates (for four communication coprocessors specified with generated VHDL-RTL models, except six FIFOs[3] in the case of two ARM7's + two 68000's) and 3,376 gates (for two communication coprocessors except six FIFOs in the case of one ARM7 + one 68000) in terms of HW area cost (AMS 0.6 μm CMOS). In both cases, communication coprocessors (except the FIFOs) take less than 5% overhead in total system area.[4]

6.2 IS-95 System.

We measured the system design time to design the IS-95 system. Table 1 shows the design time for four design steps required to implement the system on a four-processor target architecture template. In the system design, we applied the presented architecture generation flow to the architecture design step and we performed **manual** design for the other three design steps. The parameter extraction step includes such efforts as the compilation of application codes to estimate the size of required memory as well as the parameter extraction from the architecture-level specification explained in Section 4. In the architecture design, we use both processor and protocol libraries to construct the communication coprocessors. In the table, the Software Coding step represents system call insertion in the C code of the IS-95 system and Co-simulation Setup represents the building of cycle-accurate co-simulation models.[5] This experiment shows that a multiprocessor system can be designed within a few days/man with our architecture generation flow.

7. Evaluation.

Compared with conventional wrapper approaches [6][18][24], the main differences of our communication coprocessor are (1) **de-**

[2] The processor and its CC are sharing the same clock signal.

[3] The system has six FIFOs at the outputs of input/output controllers.

[4] We estimate that the ARM7 processor takes about 40K gates, the 68000 processor takes about 10K gates, and the local memory of processors takes about 10K gates.

[5] Currently, we are also working on the development of an automatic tool to generate co-simulation models. With the tool, the design time of the co-simulation setup step could be reduced down to less than one hour.

Table 1. Design time in IS-95 system design.

Design Step	Design Time
Parameter Extraction	$\sim$ 2 hr x 4
Architecture Design	$<$ 1 hr
Software Coding	$\sim$ 4 hr x 4
Co-simulation Setup	$\sim$ 8 hr ($<$ 1 hr)
Total	$<$ 33 hr (26 hr)

composition of wrapper functionalities into software and hardware parts and decomposition of HW wrapper functionalities into processor adapter and channel adapters and (2) **the integration of communication protocol implementation into the wrapper**. Note that since we use wrappers, our approach benefits also from wrapper-based design: modular design and easy integration. Such a decomposition of functionalities enables automatic generation of wrapper in software and hardware parts. The integration of communication protocol implementation into the wrapper gives also advantages:
(1) It enables HW/SW trade-off of communication implementation. By integrating the implementation of communication protocol into the communication coprocessor design, the designer can perform trade-off in communication protocol implementation. For instance, he/she can perform trade-off in a FIFO channel implementation from full HW FIFO, mixed HW/SW FIFO, and to full SW FIFO.
(2) Modular and scalable implementation of broadcasting and multi-master communication networks: since each communication coprocessor can implement channel adapters for master/slave ports, broadcasting and/or multi-master bus can be easily implemented by connecting the channel adapters to a broadcasting or multi-master bus. Thanks to the decomposition of communication coprocessor into processor adapter and channel adapters, such an integration is also scalable to any sizes of broadcasting/multi-master networks.

8. Conclusion.

In this paper, we presented a design flow for application-specific multiprocessor architecture generation. In the flow, our contribution is (1) generic multiprocessor architecture templates, (2) architecture generation from an architecture-level specification, and (3) communication coprocessors optimized to the application and the generated architecture. Experiments show that the run-time of communication coprocessor is sufficiently fast and the area overhead of communication coprocessor can be negligible. The experiments show also that multiprocessor architecture design can be achieved in a very short design cycle (in our case, one or two weeks/man for four-processor architecture design).

9. References.

[1] B. Ackland et al. A Single-Chip 1.6 Billion 16-b MAC/s Multiprocessor DSP. *Custom Integrated Circuits Conference*, 1999.

[2] A. Baghdadi, D. Lyonnard, N. Zergainoh, and A. Jerraya. An efficient architecture model for systematic design of application-specific multiprocessor soc. *Proc. Design Automation and Test in Europe*, pages 55–62, Mar. 2001.

[3] R. A. Bergamaschi and W. R. Lee. Designing Systems-on-Chip Using Cores. *Proc. Design Automation Conf.*, June 2000.

[4] Coware, Inc. N2C. *available at http://www.coware.com/coware N2C.html*.

[5] D. Culler, J. P. Singh, and A. Gupta. *Parallel Computer Architecture: A Hardware/Software Approach*. Morgan Kaufmann Publishers, Aug. 1998.

[6] D. D. Gajski, J. Zhu, R. Dömer, A. Gerstlauer, and S. Zhao. *SpecC: Specification Language and Methodology*. Kluwer Academic Publishers., 2000.

[7] L. Gauthier, S. Yoo, and A. Jerraya. Automatic generation and targeting of application specific operating systems and embedded system software. *Proc. Design Automation and Test in Europe*, pages 679–685, Mar. 2001.

[8] T. Gleerup et al. Memory Architecture for Efficient Utilization of SDRAM: A Case Study of the Computation/Memory Access Trade Off. *Proc. Int'l Workshop on Hardware-Software Codesign*, pages 51–55, May 2000.

[9] P. Guerrier and A. Greiner. A Generic Architecture for On-Chip Packet-Switched Interconnections. *Proc. Design Automation and Test in Europe*, 2000.

[10] F. Harmsze, A. Timmer, and J. van Meerbergen. Memory Arbitration and Cache Management in Stream-Based Systems. *Proc. Design Automation and Test in Europe*, Mar. 2000.

[11] M. Homewood and M. McLaren. Meiko CS-2 Interconnect Elan-Elite Design. *Proc. of Hot Interconnects*, pages 2.1.1–4, Aug. 1993.

[12] IBM. The CoreConnect^{TM} Bus Architecture. *available at http://www.chips.ibm.com/ product/ coreconnect/ docs/ crcon_wp.pdf*, 1999.

[13] IBM, Inc. 28.4G Packet Routing Switch. *Networking Technology Datasheets, available at http://www.chips.ibm.com/ product/ coreconnect/ docs/ crcon_wp.pdf*.

[14] J. Kuskin et al. The Stanford FLASH Multiprocessor. *Proc. of the 21st Int'l Symposium on Computer Architecture*, 1994.

[15] K. Lahiri, A. Raghunathan, G. Lakshminarayana, and S. Dey. Communication Architecture Tuners: A Methodology for the Design of High-Performance Communication Architectures for System-on-Chips. *Proc. Design Automation Conf.*, June 2000.

[16] J. A. J. Leijten et al. PROPHID: A Heterogeneous Multi-Processor Architecture for Multimedia. *Proc. Int'l Conference on Computer Design*, 1997.

[17] J. A. J. Leijten et al. Stream Communication between Real-Time Tasks in a High-Performance Multiprocessor. *Proc. Design Automation and Test in Europe*, 1998.

[18] C. K. Lennard, P. Schaumont, G. de Jong, A. Haverinen, and P. Hardee. Standards for System-Level Design: Practical Reality or Solution in Search of a Question? *Proc. Design Automation and Test in Europe*, Mar. 2000.

[19] H. Lin and W. Wolf. Co Design of Interleaved Memory Systems. *Proc. Int'l Workshop on Hardware-Software Codesign*, pages 46–50, May 2000.

[20] R. Lysecky, F. Vahid, and T. Givargis. Experiments with the Peripheral Virtual Component Interface. *Proc. Int'l Symposium on System Synthesis*, Sept. 2000.

[21] T. Mattson et al. An Overview of the Intel TFLOPS Supercomputer. *Intel Technology Journal*, 1st quarter 1998.

[22] R. S. Nikhil, G. M. Papadopoulos, and Arvind. *T: A Multithreaded Massively Parallel Architecture. *Proc. of the 19th Int'l Symposium on Computer Architecture*, pages 156–167, May 1992.

[23] Sonics, Inc. Sonics μNetworks, Technical Overview. *Networking Technology Datasheets, available at http://www.sonicsinc.com/Documents/Overview.pdf*, June 2000.

[24] Synopsys, Inc. SystemC, Version 2.0. *available at http://www.systemc.org/*.

[25] TIA/EIA-95A. Mobile Station-Base Station Compatibility Standard for Dual-Mode Wideband Spread Spectrum Cellular Systems. 1995.

[26] S. Vercauteren, B. Lin, and H. D. Man. Constructing Application-Specific Heterogeneous Embedded Architectures from Custom HW/SW Applications. *Proc. Design Automation Conf.*, June 1996.

[27] S. Yoo, J. Lee, J. Jung, K. Rha, Y. Cho, and K. Choi. Fast Prototyping of an IS-95 CDMA Cellular Phone: a Case Study. *Proc. the 6th Conference of Asia Pacific Chip Design Languages*, pages 61–66, Oct. 1999.