Topic: Network and communication system

Automatic Generation of Communication Architectures

Dongwan Shin, Andreas Gerstlauer,
Rainer Dömer and Daniel Gajski

Center for Embedded Computer Systems
University of California, Irvine
Irvine, CA 92697-3425, USA
(949) 824-8919

{dongwans,gerstl,doemer,gajski}@cecs.uci.edu

AUTOMATIC GENERATION OF COMMUNICATION ARCHITECTURES

Dongwan Shin
Andreas Gerstlauer
Rainer Dömer
Daniel D. Gajski
Center for Embedded Computer Systems
Information and Computer Science
University of California, Irvine
{dongwans,gerstl, deomer, gajski}@ics.uci.edu

Abstract In this paper, we propose automatic generation of bus-based communication architectures from an abstract model reflecting only the communication topology. Tasks include protocol selection for each bus, master/slave assignment for each component, interrupt handling and addressing for synchronization between components, and arbitration to resolve multiple accesses on a bus. We present a set of experimental results demonstrating how the proposed approach works on typical system designs. Experimental results show the benefits of our methodology and demonstrate the effectiveness of automatic model generation for communication design.

1. Introduction

With ever increasing SoC complexities, design of system communication structures is becoming an in increasingly important factor and bottleneck. Together with time-to-market pressures, communication design requires extensive design space exploration in a short amount of time. Typically, designers use models of a system to validate and evaluate different designs. Traditionally, these models are manually written, which is a tedious, error-prone task, and time-consuming task, severely limiting exploration opportunities.

In order to tackle these problems, we propose a communication design flow with automatic generation of communication models from a virtual architecture model. Figure 1 shows the communication design flow [5]. Design starts with the architecture model which reflects the structure

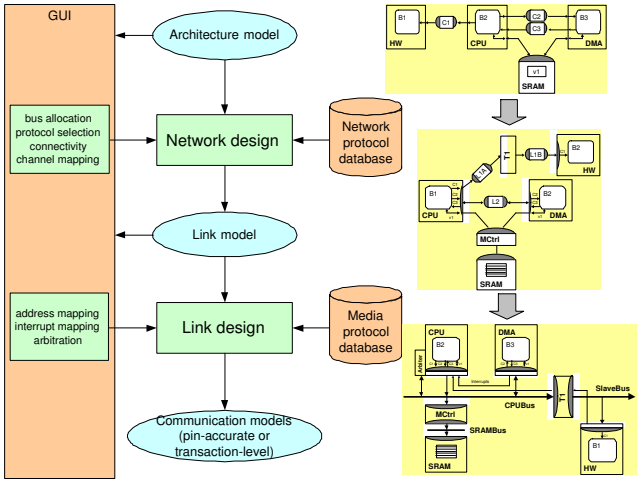


Figure 1. Communication design flow.

of processing components/elements (PEs), but where communication is done abstractly on a message-passing level. Communication design is then divided into two tasks: *network design* and *link design*.

During network design, the topology of the communication architecture is defined and abstract message passing channels between processors are mapped into communication between adjacent stations of the communication architecture. The network topology of communication stations connected by logical link channels is defined, bridges and other communication elements are allocated as necessary, and abstract message passing channels are routed over sets of logical link channels. The result of the network design step is a link model of the system which represents the topology of the communication architecture and in which stations communicate via untyped, logical links.

During link design, logical links between adjacent stations are then grouped and implemented over an actual communication medium (e.g. system busses). For each group of links to be implemented over a single, shared medium, a communication protocol is selected and parameters such as addresses and interrupts for synchronization are assigned to each logical link.

As a result of the communication design process, a pin-accurate or transaction-level communication model of the system is generated. Communication models are fully structural where components are connected via busses and communicate in a timing-accurate manner based on media protocol timing specifications.

In this paper, we concentrate on the link design task and we will present our approach to speeding up the link design process by enabling automatic model refinement. The rest of the paper is organized as follows. Section 2 gives an overview of related work. Section 3 shows our refinement-based link design flow and Section 4 looks at the individual tasks of link refinement. Finally, we present experimental results in Section 5 and wind up with a summary and conclusion.

2. Related work

Recently, system-level design languages have been proposed as vehicles for so-called transaction-level modeling (TLM) for communication abstraction [4] [7]. However, TLM proposals so far focus on simulation only and they lack the path to vertical integration of models for implementation and synthesis.

There are several approaches dealing with automatic generation of communication architectures [2] [3]. These approaches, however, are usually based on target architecture templates and limited in their support for general architectures and applications. Furthermore, most of the work has been done in optimizing communication architectures for specific designs [6] [8]. Finally, approaches that deal with optimization and automatic decision making for communication synthesis [11] [9] are usually lacking support for generating implementations for those decisions.

In previous work [1], we proposed an automatic communication refinement flow. In this paper we extend this work to support more general architectures with networks of interconnected busses, realistic communication mechanisms and advanced synchronization primitives.

3. Link design

Link design implements the functionality of link layer, media access layer and protocol layer and inlines them into corresponding components. The link layer defines the type of a communication station (e.g. master/slave on a bus) for each of its incoming or outgoing links. It is also responsible for implementing synchronization between communication stations, e.g. vai interrupts or by polling in case of interrupt sharing.

The media access layer is responsible for slicing blocks of bytes into bus words. Furthermore, it resolves simultaneous bus accesses of components through arbitration. Depending on the arbitration scheme chosen, additional arbitrater components are introduced into the system as part of the media access layer.

Finally, the protocol layer is responsible for driving and sampling the external pins according to the protocol timing diagrams and thereby matching the transmission timing on the sender and receiver sides.

3.1 Inputs and Outputs

Link design starts from a link model, which represents the topology of the communication architecture. Components on the top level of the design communicate with each other via logical link channels. Each channel provides *send/receive* methods for enable data transactions with message passing semantics.

During the design process, the user provides a set of design decisions such as protocol selection for each bus, master/slave assignment for components, address and interrupt assignment for logical links, and arbitration scheme and bus access priorities.

With these inputs, the link refinement tool produces an output communication model that reflects the bus architecture of the system. In the output model, the top level of the design consists of system components connected by wires of the system busses. The components themselves are refined down to bus-functional models that communicate via ports.

3.2 Databases

Link design is supported by a media database that consists of a database of bus protocols and a database of associated bus-functional compoment models.

3.2.1 Bus database. The bus database contains models of busses including associated protocols. Bus models in the bus database consist of a stack of two layers: protocol layer and media access layer. At the bottom of the stack, the protocol layer is connected to the actual bus wires and it implements the primitives defined by the bus protocol for data transfers, synchronization and arbitration. On top of the protocol layer, the media access layer provides an abstraction of external communication into data links and memory accesses by using and combining bus primitives to regulate media accesses and slice abstract data into bus words.

Each protocol layer can have two separate sides with different implementations for bus masters and bus slaves. Each layer provides a protocol implementation for one single component connected to the bus. Protocol layer models connect to the bus wires through ports of the model and pins of the component. Layers are stacked on top of each

other and connect via interfaces where the media access layer calls the methods of the protocol layer beneath it.

3.2.2 Bus-functional component database. For components with fixed, pre-defined interfaces and communication functionality, the component database has to contain a bus functional model of the component. A bus functional component model accurately describes the component interface at the pin level and it provides a simulation model of communication aspects of the component.

For programmable components with flexible computation behavior but fixed, pre-defined interfaces and communication functionality, a bus functional model with at least two layers has to be provided in the database: a top level bus functional layer describing the component pin interface on the outside and an internal, empty hardware abstraction layer (HAL) describing the interface for accessing the bus medium from the software on the inside. In addition, the HAL has to provide templates of interrupt handlers for each external interrupt line of the processor.

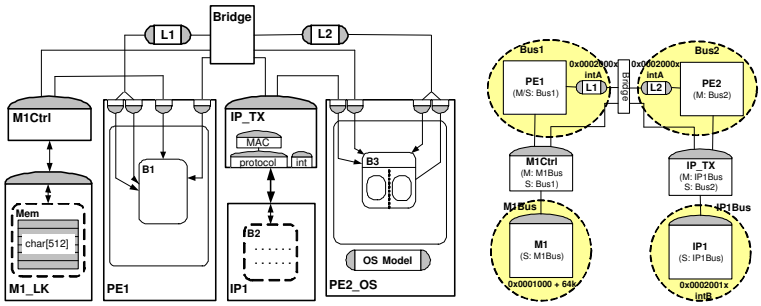
4. Link refinement

Link refinement is the process of transforming the input link model into a communication model based on the user-supplied decisions. The refinement process can be divided into five major steps, namely, channel grouping, bus functional model instantiation, synchronziation synthesis, arbiter/interrupt controller insertion, and bus wiring. be further divided into sub-steps.

In the following, we will outline transformations for link refinement. More details about this process can be found in [10]. We will use a simple example (Figure 2(a)) where 2 PEs (*PE1* and *PE2_OS*), 1 IP (*IP1*), a shared memory (*M1_LK*) and a bridge (*Bridge*) are allocated. They are communicating using message passing channels *L1* and *L2*. The design decision for link design are made as shown in Figure 2(b). For example, the channel *L1* is assigned to interrupt *intA* and address *0x00020000*.

4.1 Channel grouping

The first task of link refinement is channel grouping which combines different links mapped onto a bus. Message passing channels between components will be grouped into transactions over a single, shared bus and unique bus addresses will be assigned to each link and each memory interface or slave register mapped onto the bus.



(a) Link model. (b) Design decisions

Figure 2. An example of link model and design decision.

4.2 Bus-functional component instantiation

As a next step, bus functional models for components with fixed, predefined bus interfaces (e.g. programmable processors, IPs, bridges and system memories) are taken out of the bus-functional component database and instantiated in the design.

Bus functional models for programmable components have to include a definition of the interrupt capabilities of the component. The top level bus functional shell defines the interrupt pins available at the physical component interface and the hardware abstraction layer (HAL) model provides corresponding empty interrupt handler templates. During link refinement, interrupt lines from slaves are connected to the interrupt pins of programmable components and interrupt handlers in the HAL are generated by filling the templates. Finally, interrupt tasks triggered by the HAL interrupt handlers are generated in the operating system of the processors.

4.3 Synchronization synthesis

In order to preserve the semantics of the original input model, synchronization between components has to be introduced whenever necessary. The link layer is responsible for implementing synchronization through interrupts and/or polling. Link layers have different implementations depending on the type of station (master/slave). Methods on the master side wait for interrupt from slaves before invoking media access layer methods to perform the actual data transfer. On the slave side, a slave will send an interrupt to notify the master about any data transfer request. In case of memory or register (memory-mapped I/O) accesses, slave components are assumed to be always ready and no extra synchronization is necessary.

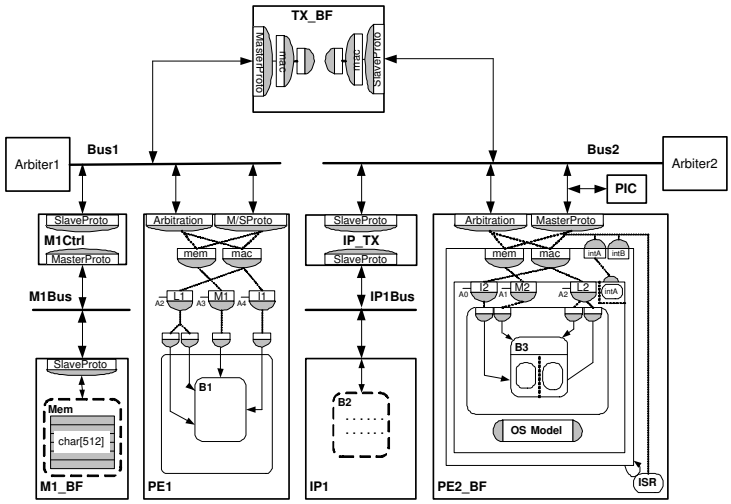


Figure 3. Communication model example.

4.4 Arbiter/Interrupt controller insertion

If multiple master components are connected to a bus, arbitration becomes necessary to resolve conflicting accesses of bus masters. The arbitration mechanism will be instantiated from the bus database as part of the bus protocol master implementation. All masters are assigned additional arbitration ports connected to the arbiter on the bus. The arbiter will be instantiated at the top level of a design together with the arbitration wires. Based on design decisions, we generate a priority-based or round-robin arbitrater component.

If a master communicates with more than one slave, it will require an interrupt controller to handle synchronization requests from multiple slaves. For each slave on a bus, an interrupt port is created and connected to the corresponding interrupt wire on the bus. Finally, an interrupt controller is generated and inserted into the bus master component.

4.5 Bus wiring

After all bus-functional models of processing and communication elements are generated and/or inserted from the database, components at the top of the design need to be connected to each other through bus wires. Bus-functional component models define the bus ports of each station. Connections between port and busses are defined through the

Table 1. Design decisions for link design.

Examples		Traffic (bytes)	Channel (num.)	Medium (master/slave)
JPEG	A1	2244	6	DSP Bus (DSP/IP)
	A2	3420	13	DSP Bus (DSP/(IP,HW))
Vocoder	A1	46944	12	DSP Bus (DSP/HW)
	A2	140832	36	DSP Bus (DSP/(2 HWs)
	A3	154524	42	DSP Bus (DSP/(3 HWs)
	A4	57160	29	2 DSP Bus (2 DSPs/2 HWs)
MP3	A1	0	0	CF Bus (CF)
	A2	169747	66	CF Bus (CF/4 HWs) 4 Handshake Bus (4 HWs)
Baseband	A1	178500	19	CF Bus ((CF,DMA)/(IP,DMA,MEM) DSP Bus (DSP/(5 HWs)

port mapping. Finally, interrupt and arbitration lines are connected based on the priorities selected by the user.

As a result, the final communication model of the design is generated. Figure 3 shows the communication model for the example from Figure 2. Logical link channels from the link model have been inlined into the connected components. Media access and protocol layer channel adapters are taken out of the bus database, inserted into the bus functional model of the corresponding components and connected to the logical link adapters. Additional communication elements such as interrupt controllers (*PIC*) and arbiters (*Arbiter1* and *Arbiter2*) are inserted into the design. Inside programmable components (*PE2*), interrupt service routines (*ISR*) and interrupt handling methods (*intA* and *intB*) are generated and inserted for synchronization with other system components (*PE1* and *IP1*).

5. Experimental results

Based on the described methodology and algorithms, we developed a link refinement tool for automatic generation of communication models. We performed experiments using four industrial strength examples: a JPEG encoder (*JPEG*), a voice codec (*vocoder*), an MP3 decoder (*MP3*) and a baseband platform (*Baseband*) which combines a JPEG encoder with a voice codec. For each example, we implemented several different architectures. Table 1 shows the total traffic, the number of logical link channels and the allocated architecture each.

Table 2. Experiment results of link refinement.

Examples		Lines of Code			Tool	Man.	Gain
		Link	BF	Mod. ($ins. + del. - DB$)	(sec)	(hr)	
JPEG	A1	3464	5250	351 (1867 − 1618 + 102)	0.10	35.1	421
	A2	3755	5655	303 (1975 − 1768 + 96)	0.10	30.3	364
Vocoder	A1	10980	11740	341 (794 − 487 + 34)	0.31	34.1	409
	A2	11415	12205	405 (841 − 487 + 51)	0.33	40.5	486
	A3	12276	13096	489 (897 − 487 + 77)	0.39	48.9	587
	A4	14033	15220	757 (1309 − 674 + 122)	0.84	75.7	908
MP3	A1	29959	31666	375 (1822 − 1584 + 137)	0.44	37.5	450
	A2	33905	36361	1198 (2724 − 1818 + 292)	1.06	119.8	1437
Baseband	A1	20227	23027	1288 (3150 − 2212 + 350)	1.02	128.8	1545

Table 2 shows the results of link refinement. Overall model complexities are given in terms of code size using lines of code (LOC) as a metric. Results show significant differences in complexity between input and generated output models due to extra implementation detail added between abstraction levels. To quantify the actual refinement effort, the number of modified lines is calculated as the sum of lines inserted and lines deleted whereas code coming from database models is excluded. We assume that a person can modify 10 LOC/hour. Thus, manual refinement would require several hundred man-hours for reasonably complex designs. Automatic refinement, on the other hand, completes in the order of seconds. In order to compute the productivity gain, we assume that design decisions (address/interrupt assignment, arbitration) for link refinement can be done in 5 minutes. Results show that a productivity gain of around 1000 times can be expected using the presented approach and automatic model refinement.

6. Conclusions

In this paper, we presented a methodology to automatically generate communication models from a representation of the communication topology and abstract communication channels going across. During this link design process, logical links between adjacent components are grouped and implemented over a system bus and link, MAC and protocol layers are implemented at the interfaces of components.

Using several industrial-strength examples, the feasibility and benefits of the approach have been demonstrated. Huge productivity gains can be obtained using automatic link refinement. Our main contribution in the paper is the automation of a time consuming and error prone

process to achieve better designer productivity, thus enabling designers to explore a large part of the design space in a shorter amount of time.

References

[1] S. Abdi, D. Shin, and D. D. Gajski. Automatic communication refinement in system-level design. In *Proceedings of the Design Automation Conference*, pages 300–305, June 2003.

[2] I. Bolsens, H. D. Man, B. Lin, K. V. Rompay, S. Vercauteren, and D. Verkest. Hardware/Software co-design of the digital telecommunication systems. *Proceedings of the IEEE*, March 1997.

[3] W. O. Cesario, A. Baghdadi, L. Gauthier, D. Lyonnard, G. Nicolescu, Y. Paviot, S. Yoo, A. A. Jerraya, and M. Diaz-Nava. Component-baed design approach for multicore SoCs. In *Proceedings of the Design Automation Conference*, pages 789–794, June 2002.

[4] M. Coppola, S. Curaba, M. Grammatikakis, and G. Maruccia. IPSIM: SystemC 3.0 enhancements for communication refinement. In *Proceedings of the Design Automation and Test Conference in Europe*, pages 106–111, March 2003.

[5] A. Gerstlauer, D. Shin, R. Dömer, and D. D. Gajski. System-level communication modeling for Network-on-Chip synthesis. In *Proceedings of Asian South Pacific Design Automation Conference*, pages 45–48, January 2005.

[6] G. Gogniat, M. Auguin, L. Bianco, and A. Pegatoquet. Communication synthesis and HW/SW integration for embedded system design. In *Proceedings of the International Workshop on Hardware-Software Codesign*, pages 49–53, March 1998.

[7] T. Grötker, S. Liao, G. Martin, and S. Swan. *System Design with SystemC*. Kluwer Academic Publishers, March 2002.

[8] P. Knudsen and J. Madsen. Integrating communication protocol selection with partitioning Hardware/Software codesign. In *Proceedings of the International Symposium on System Synthesis*, pages 111–116, December 1998.

[9] R. B. Ortega and G. Borriello. Communication synthesis for distributed embedded systems. In *Proceedings of the International Conference on Computer-Aided Design*, pages 437–444, November 1998.

[10] D. Shin, A. Gerstlauer, and D. D. Gajski. Communication link synthesis for SoC. Technical Report CECS-TR-04-16, Center for Embedded Computer Systems, University of California, Irvine, June 2004.

[11] T.-Y. Yen and W. Wolf. Communication synthesis for distributed embedded systems. In *Proceedings of the International Conference on Computer-Aided Design*, pages 288–294, November 1995.