Mixed-Level Cosimulation for Fine Gradual Refinement of Communication in SoC Design

Gabriela Nicolescu Sungjoo Yoo Ahmed A. Jerraya

SLS Group, TIMA Laboratory
46 Avenue Félix Viallet, 38031 Grenoble, France
{Gabriela.Nicolescu,Sungjoo.Yoo,Ahmed.Jerraya}@imag.fr

Abstract

In this paper, we propose a method of mixed-level cosimulation that enables fine gradual refinement of SoC communication from protocol-neutral communication to protocol-fixed communication. For fine granularity in refinement, the method enables the designer to perform **channel-by-channel refinement** and **inter-level channel refinement**. Thus, the designer can perform more extensive design space exploration in communication refinement. We show the effectiveness of the proposed method in a case study of communication refinement in an IS-95 CDMA cellular phone system design.

1 Introduction

One of benefits from the separation between behavior and communication in SoC design [1][2][3] is the fact that the designer can design the system in a modular way. The module interfaces and communication channels can be refined separately from the refinement of module behavior. Together with recently introduced high abstraction levels of communication (e.g. Functional Interface (FI) [2], Remote Procedural Call (RPC), (Abstract) Channels [4][5], and Virtual Component Interface (VCI) [2], etc.), the separation enables **gradual communication refinement**.

In gradual communication refinement, system communication can be refined from a high abstraction level (e.g. FI or RPC) to a low level (e.g. VCI or cycle-accurate models) separately from the system behavior [2]. For the validation of each refinement step in gradual refinement, mixed-abstraction-level cosimulation (in short, **mixed-level cosimulation**) plays a key role. In this paper, we focus on mixed-level cosimulation in communication refinement from protocol-neutral communication to protocol-fixed communication.

Refining communication protocols is one of crucial tasks in communication refinement since they can have significant impact on system runtime, power consumption and resource usage. Related to the refinement, there is a huge design space with candidate communication protocol types (FIFO, handshake, etc) [5], protocol-specific parameters (FIFO size, blocking or non-blocking read/write, etc) [6], and different implementation styles of module interface behavior [7][8]. Thus, for extensive design space exploration, fine granularity is very important in gradual communication refinement.

To achieve fine granularity in communication refinement, the two following types of refinement are crucial.

- Inter-level channel refinement
- Channel-by-channel refinement

In inter-level channel refinement, the designer can refine the module interface and the communication channel, separately. One case of inter-level channel refinement is integrating, into high-level communication models, IP blocks that have fixed module interfaces, i.e. fixed communication protocols (we call such IP blocks **protocol-fixed IP blocks**). There is another case that a system testbench at a high abstraction level (i.e. without a specific communication protocol) is connected with a communication channel having a fixed protocol. In channel-by-channel refinement, the inter-level channel refinement can be performed on a communication channel basis. Thus, the designer can focus on the communication refinement of the sub-systems, and evaluate the partial communication refinement before refining all the communication channels.

In this paper, we propose a mixed-level cosimulation method that supports the two above-mentioned types of granularity in communication refinement. This paper is organized as follows. We give a short overview of related work in Section 2. As a preliminary, in Section 3, we explain communication refinement from protocol-neutral to protocol-fixed communication. We propose a mixed-level cosimulation method in Section 4. In Section 5, we present a case study of applying the proposed method. In Section 6, we give the conclusion.

2 Related Work

For the abstraction levels of communication, Virtual Socket Interface Alliance (VSIA) presents Functional Interface (FI) and Virtual Component Interface (VCI) [2]. FI provides a limited and well-defined set of transactions such as read, write, sense, and emit to define a common interface between behavior and communication. FI does not assume any specific communication protocol. VCI provides a set of logical signals with a flexible and extendable protocol at the cycle accurate level. To integrate modules with VCI into a specific bus protocol, a bus protocol specific wrapper should be designed. In [9], based on a system design language, SpecC, a methodology of communication refinement from protocol-neutral (channel), protocol-fixed (virtual bus), to cycle-accurate communication is presented.

The concept of **abstract channel** is based on protocol-fixed communication [5][4]. In [5], communication and behavior are separated by the channel services provided by protocol-fixed communication channels. In [4], FIFO channels and their services can be simulated by a cycle-based simulation engine. In [3][4], Remote procedure call (RPC) is also introduced. It can be used as one of high level communication protocols or as a primitive function to implement a specific communication protocol.

In [10], several mixed-level cosimulation methods are introduced. Among them, bus functional models (BFM's) are widely used in most cosimulation environments [11]. BFM transforms a functional memory access (e.g. reading a variable named A or located at a specific address, for instance, at `0x1000`) into cycle-

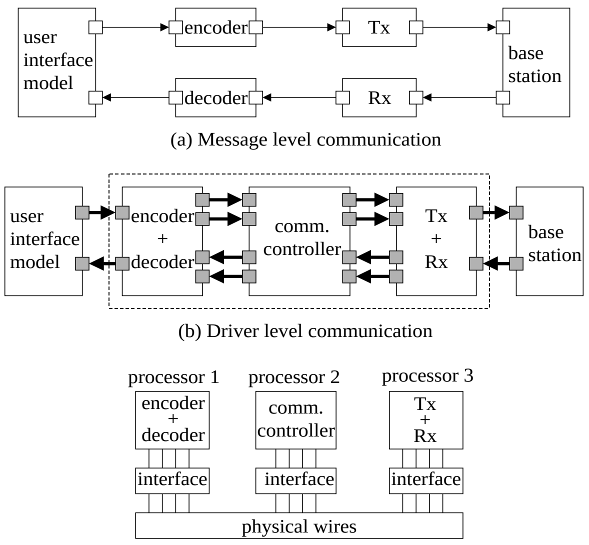


Figure 1: Examples of abstraction levels of communication model.

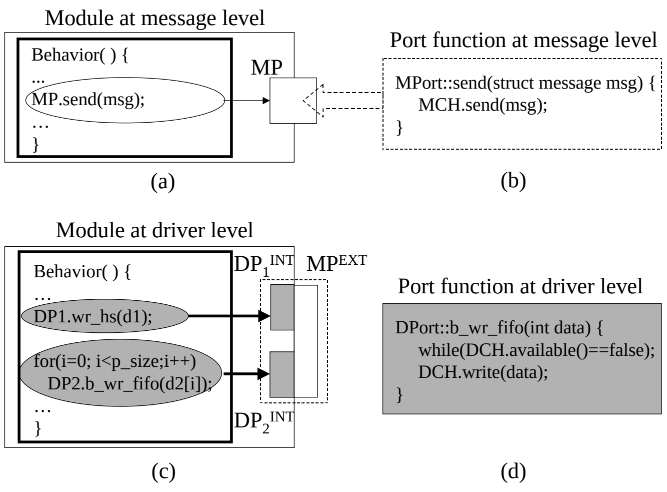


Figure 2: An example of module interface refinement.

accurate memory accesses (e.g. cycle-accurate events on processor pins). For mixed-level cosimulation between RPC and cycle-accurate level, a bus cycle accurate shell (BCASH) is proposed in [4][3]. In [12], a case study of mixed-level cosimulation between abstract channel and cycle-accurate model is presented.

In most of previous methods of communication refinement, (1) validation of refined communication is done after the whole communication refinement from one level to another level has been finished (e.g. after obtaining all the cycle-accurate models of the communication part or after fixing the communication protocols of all the communication channels) [9][6] or (2) mixed-level cosimulation is performed only between cycle-accurate communication level and a high abstraction level [3][4]. Compared with previous mixed-level cosimulation methods, the proposed method has a significant contribution in that (1) it enables much finer gradual communication refinement by supporting mixed-level cosimulation during inter-level channel refinement and channel-by-channel refinement and (2) it supports mixed-level cosimulation between protocol-neutral and protocol-fixed communication levels.

3 Preliminary: Refinement from Protocol-Neutral to Protocol-Fixed Communication

For communication refinement, we use three abstraction levels of communication: **message level** (protocol-neutral communication level, in short ML), **driver level** (protocol-fixed communication level, in short DL), and **register transfer level** (cycle-accurate level). Figure 1 exemplifies the three abstraction levels of communication applied to the communication refinement of an IS-95 CDMA cellular phone system [13][14]. The system consists of voice encoder/decoder modules and CDMA modem transmitter/receiver (Tx/Rx) modules. In the figure, as the system testbench, a user interface model (for speaker and microphone) and a base station simulation model are also shown.

At each abstraction level, we represent the system with a hierarchical network of **modules** as shown in Figure 1. A module consists of **behavior** and **port(s)**. Modules are connected with each other by connecting their ports via **communication channels**. The behavioral part of the module calls **port functions** to communicate with other modules. In the viewpoint of behavior, ports encapsulate communication details (i.e. communication protocols). In this paper, we focus on communication refinement from ML to DL.

At ML, modules communicate with each other exchanging **messages** over ML channels. Note that the message does not have any specific data type, i.e. it has a generic data type. There is no specific communication protocol for the channel. Thus, the channel provides two kinds of **channel access function** (in short, **channel function**), **send** and **receive** for ML ports to exchange messages via the ML channel. Ports also support the same functions for the behavioral part to use them to communicate over the ML channel. At DL, each DL channel has its own communication protocol (e.g. FIFO, handshake, etc) and the parameters (e.g. FIFO size) of the communication protocol are determined. The data transferred via the DL channel have fixed data types (e.g. `int`, `float`, etc).

We define **module interface** as the port and the communication behavior (in the behavioral part of module) related to the port (by calling the port functions). Figure 2 (a) shows an example of module interface at ML. In the figure, the module has an ML port called **MP**. In the behavioral part of the module (**Behavior()**), a blank oval, where the ML port function (**MP.send(msg)**) is called, corresponds to the communication behavior in the behavioral part. In this case, the module interface corresponds to the communication behavior in the behavioral part and the port, **MP**.

In communication refinement from ML to DL, the ML channel and module interface can be refined, separately. We call the two kinds of refinement, **channel refinement** and **module interface refinement**, respectively.

In channel refinement, the ML channel can be split into several DL channels (**channel partitioning**). A communication protocol is assigned to each of DL channels and its protocol parameters are fixed. By channel partitioning, a single message can be transferred over several DL channels. The channel partitioning is up to the designer (probably from the trade-off between system performance and resource usage with performance/cost models of channel implementation [6]). In the case of integrating protocol-fixed IP blocks, channel refinement may require the implementation of a protocol converter(s) since the protocol-fixed IP blocks connected with each other can have already different communication protocols. We call the protocol converter the **communication controller**. Figure 1 (b) shows an example of communication controller assuming that voice encoder/decoder and modem transmitter/receiver are protocol-fixed IP blocks.

In module interface refinement, depending on the channel partitioning or the interface of protocol-fixed IP blocks, an ML port can be refined to several DL ports. Protocol-specific functions (e.g.

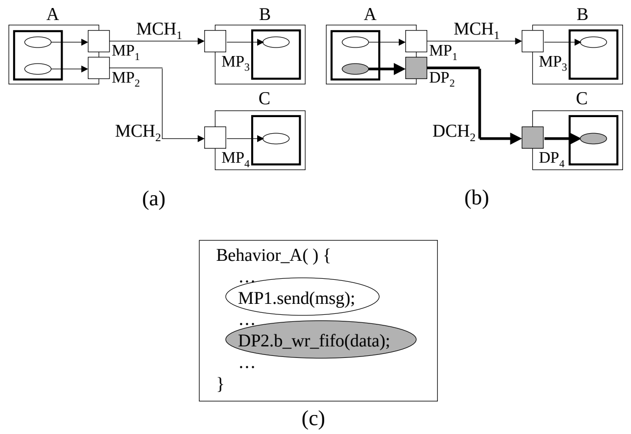


Figure 3: An example of channel-by-channel communication refinement.

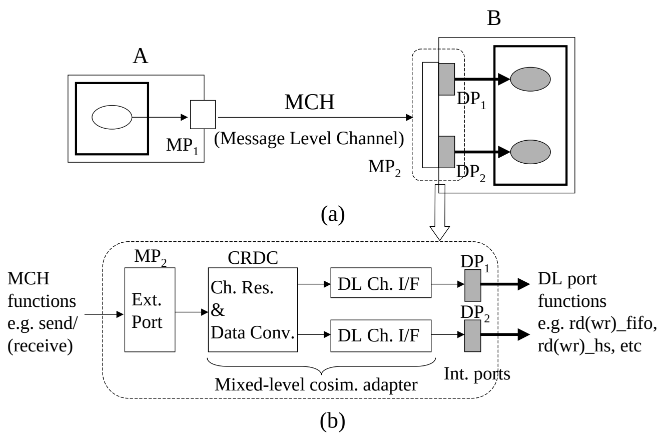


Figure 4: Mixed-level cosimulation in the case of module interface refinement.

blocking write operation to a FIFO channel) are assigned to each DL port. Figure 2 shows an example of module interface refinement. Figure 2 (c) shows a result of module interface refinement from an ML module interface in Figure 2 (a). In this case, the ML port is refined to two DL ports and the message (**msg**, in Figure 2 (a)) is split into two types of data, d1 (a single data item of `int` type) and d2 (a data array of `int` type). We denote the DL communication behavior in the behavioral part of module with shaded ovals as shown in Figure 2 (c).

In Figure 2 (c), assume that only the module interface is refined while the connected ML channel is not yet refined. In this case, from the viewpoint of the connected ML channel, the module should have an ML port for the channel to be connected to the module. From the viewpoint of the DL module interface, the module should have two DL ports. To compromise two conflicting requirements, we introduce a concept called **internal** and **external ports**. In Figure 2 (c), the dashed rectangle includes two **internal ports** (in this case, two DL ports, $\mathrm{DP}_1^{\mathrm{INT}}$ and $\mathrm{DP}_2^{\mathrm{INT}}$) and one **external port** (one ML port, $\mathrm{MP}^{\mathrm{EXT}}$). In this case, internal ports provide the behavioral part of module with DL port functions. The external port accesses the ML channel via ML channel functions. Note that, in this case, the internal ports have already been refined by the designer. For mixed-level cosimulation, we insert a **mixed-level cosimulation adapter** between internal and external ports (to be explained in Section 4).

Examples of an ML port function (MPort::send()) and a DL port function (DPort::b_wr_fifo(), a blocking write function to the FIFO) are shown in Figure 2 (b) and (d). Compared with the ML port function, the DL port function has arguments of fixed data type (in this case, `int` type). Note that the ML/DL port function accesses the ML/DL channel (MCH/DCH) via **ML/DL channel functions** (in this case, MCH.send() for ML and DCH.available() and DCH.write() for DL). In Figure 2 (in Figure 1 also), we denote ML ports (channels) with blank rectangles (thin arrows) and DL ports (channels) with shaded rectangles (thick arrows).

4 Mixed-level Cosimulation between Message Level and Driver Level

In this section, we first explain mixed-level cosimulation for the case of channel-by-channel communication refinement and then we present methods for the case of inter-level channel refinement.

4.1 Mixed-level cosimulation for channel-by-channel refinement

Figure 3 shows an example of channel-by-channel refinement. Figure 3 (a) shows a system representation at ML. We assume that the designer refines, from ML to DL, an ML channel (MCH_2), two ML ports (MP_2 and MP_4) connected to the channel, and communication behavior (two blank ovals in modules A and C) related to the ML ports. Figure 3 (b) exemplifies the result of refinement. In the figure, a new DL channel (DCH_2), new DL ports (DP_2 and DP_4) and DL communication behavior of modules B and C (two shaded ovals) are shown. In the example, note that the communication between modules A and B has not been refined yet and it still remains at ML.

Figure 3 (c) exemplifies the communication behavior of module A (in Figure 3 (b)) after the refinement. In the figure, the behavioral part of module A has communication at two abstraction levels: one (**MP1.send(msg)**) at ML and the other (**DP2.b_wr_fifo(data)**) at DL. Thus, to simulate the case of Figure 3 (b) where ML and DL communication levels co-exist, mixed-level cosimulation is required. However, in this case, as far as both ML and DL communication levels can be simulated in a single cosimulation environment, there is no new functionality required to implement mixed-level cosimulation **since there is no communication between ML and DL over a single channel**.

In practice, to our knowledge, when writing the paper, since there is no simulation environment that supports both abstraction levels of communication (ML and DL) simultaneously, a hybrid cosimulation environment that consists of two different simulation environments is required. In our case study in Section 5, we present our implementation with SDL [15] (for ML) and SystemC [4] (for DL) simulation environments.

4.2 Mixed-level cosimulation for inter-level channel refinement

In general, inter-level channel refinement has two cases: (1) the module interface is refined first before the ML channel is refined (e.g. integrating protocol-fixed IP blocks) or (2) the channel is refined first before the module interface (e.g. connecting a simulation testbench without a specific protocol to a DL channel).

4.2.1 Case of module interface refinement first

Figure 4 shows an example of module interface refinement and the implementation of mixed-level cosimulation. An ML port of module B is assumed to be refined into two DL ports (DP_1 and DP_2) to-

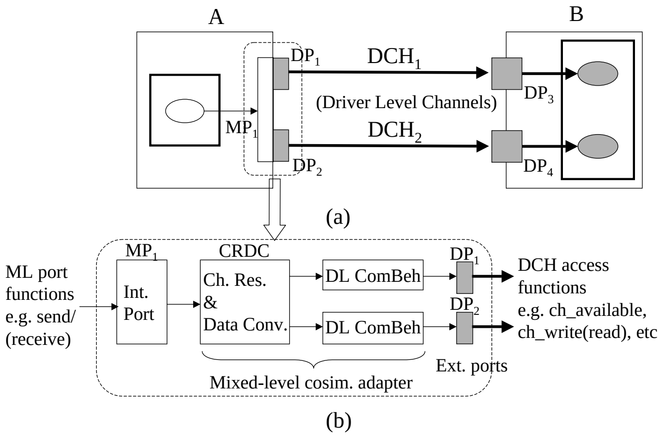


Figure 5: A case of channel refinement.

gether with the related communication behavior (two shaded ovals) while module A and an ML channel (MCH) are still at ML. In the figure, the refined DL ports (two shaded rectangles) are internal ports (in the figure, Int. ports) and an external port (i.e. an ML port, MP_2) is used for the connection with the ML channel.

Figure 4 (b) shows the details of the mixed-level cosimulation adapter between the external port and the two internal ports. Between external and internal ports, as the adapter, there are two blocks: one for **channel resolution** and **data conversion** (Ch. Res. & Data Conv., in short, CRDC) and the other for **DL channel interface** (DL Ch. I/F). Since the original ML port is assumed to be refined to two DL ports, the CRDC block transforms messages (of generic data type) received from the MCH into DL data (i.e. fixed data type). To do that, assuming the ML port is refined to **N** DL ports, first, the message is split into **N** data chunks according to the channel partitioning (**channel resolution**) and each of **N** data chunks is given its specific data type such as `bool`, `int`, `float`, `fixed`, etc (**data conversion**). The DL Ch. I/F block provides the connected DL port with DL channel functions since the internal ports have already fixed communication protocols.

4.2.2 Case of channel refinement first

Figure 5 (a) shows an example that an ML channel is refined to DL channels. In the figure, we assume that the MCH in Figure 4 has been refined to two DL channels. The example can also represent a case that only the ML channel is refined but one (or both) of the module interfaces (connected to the channel) has not been refined. Figure 5 (b) shows the details of the mixed-level cosimulation adapter between the internal port (ML port) and the external ports (DL ports) of module A. In the figure, the internal port, MP_1 (that has not been refined) provides the behavioral part of module A with ML port functions. Two external ports access the two DL channels via DL channel functions (e.g. ch_available, nb_write, b_read, etc, in the case of FIFO channel).

Between internal and external ports, the CRDC block performs channel resolution and data conversion operations as in the case of the module interface refinement shown in Figure 4. In the case of channel refinement, a block called **DL communication behavior** (DL ComBeh) is used to call DL port functions of external ports (i.e. DL ports). The DL ComBeh blocks are equivalent to the communication behavior of the behavioral part when the module interface is refined.

Table 1: Characteristics of communication in the communication refinement of IS-95 system.

	Message Level	Driver Level
Ch. behavior	infinite FIFO	finite FIFO
Ch. functions	send/receive	full/empty/write/read
Port functions	send/receive	b_rd_fifo/b_wr_fifo
Data types	generic	fixed (`short` and `int`)

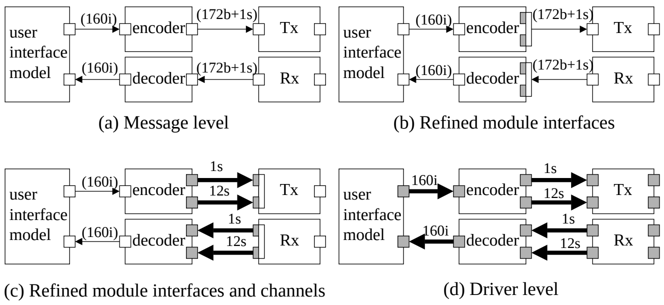


Figure 6: Gradual communication refinement of IS-95 system.

5 A Case Study

For communication specification with two abstraction levels, we used SDL [16] for ML and SystemC [4] (specifically, channels provided by SystemC) for DL. Table 1 shows communication characteristics of two abstraction levels in our case study. Note that the ML channel acts as an infinite FIFO that provides **send** and **receive** functions to transfer messages of generic type. In our case study, as a communication protocol at DL, we use finite FIFO's.

In our case study, we applied the mixed-level cosimulation method to the communication refinement of an IS-95 CDMA cellular phone system exemplified in Figure 1. The system is re-drawn in Figure 6. For simplicity of explanation, the base station model is not shown in the figure but it is also simulated in our experiments. In the figure, each channel is given a number that represents the number of data transferred over the channel at every frame of voice data.[1] Each ML channel carries a single message (denoted with a pair of parentheses in the figure) at each frame. In the figure, on each ML channel, the number in the pair of parentheses represents the number and type (b: `bool`, i: `int`, s: `short`) of data contained in each message. In Figure 6 (c) and (d), the refined DL channels (thick arrows) have fixed data types, `short` or `int`.

We performed communication refinement from all ML communications (Figure 6 (a)) to all DL communications (Figure 6 (d)), i.e. channel partitioning, protocol selection, fixing the protocol parameters, and module interface refinement have been performed.

Figure 6 (b) shows a case of module interface refinement where the interfaces of two modules, encoder and decoder are refined to DL. Figure 6 (c) shows a case of channel refinement where the ML channel between encoder (decoder) and CDMA modem Tx (Rx) is refined to two DL channels. The two cases require mixed-level cosimulation between ML and DL.

Channel-by-channel refinement

In the two cases of Figure 6 (b) and (c), two modules, encoder and decoder have two abstraction levels in their ports since we perform channel-by-channel refinement. For instance, in the case of the encoder, in Figure 6 (c), it has an ML port for communication with the user interface model and two DL ports for communica-

[1] In the IS-95 system, voice is encoded/decoded/transmitted/received on a frame basis. One frame corresponds to 20 ms in reality.

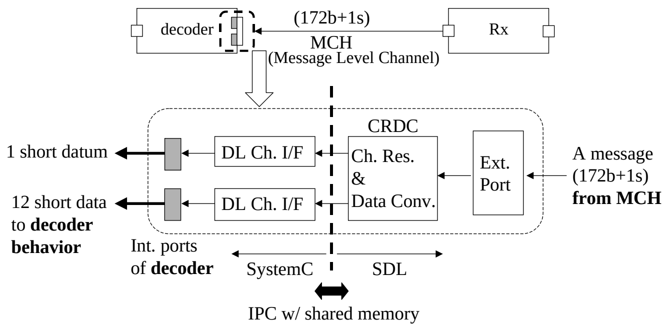
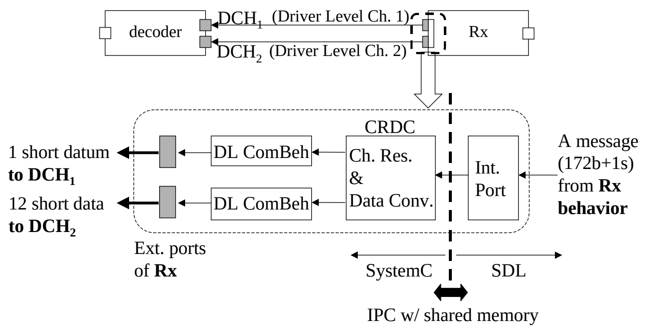


Figure 7: An example of mixed-level cosimulation implementation: module interface refinement case.



decoder
DCH_1 (Driver Level Ch. 1)
DCH_2 (Driver Level Ch. 2)
Rx
CRDC
1 short datum **to DCH_1**
DL ComBeh
Ch. Res. & Data Conv.
Int. Port
A message (172b+1s) from **Rx behavior**
12 short data **to DCH_2**
DL ComBeh
Ext. ports of **Rx**
SystemC
SDL
IPC w/ shared memory

Figure 8: An example of mixed-level cosimulation implementation: channel refinement case.

tion with module Tx. In this case, the encoder module should be simulated in one of the two simulation environments, SDL simulation environment [15] and SystemC simulation environment [4]. In our implementation, we simulate the behavioral parts of the two modules, encoder and decoder in SystemC simulation environment and implement inter-process communication (IPC) with the SDL simulation environment inside of ML port functions. For instance, recalling the example in Figure 3 (c), we simulate the behavioral part of module A at the DL (i.e. SystemC) simulation environment and implement IPC with the ML (i.e. SDL) simulation environment inside of the ML port function, in this example, **MP1.send()**.

Inter-level channel refinement

Figure 7 shows the details of mixed-level cosimulation adapter for the case of module interface refinement of the decoder in Figure 6 (b). As shown in the figure, the ML channel (MCH) from Rx, the external port of decoder module, and the CRDC block are simulated in SDL simulation environment. The internal ports and DCIF blocks are simulated together with the decoder behavior in SystemC simulation environment. The communication between SDL and SystemC environments is executed via IPC using shared memory on Unix. Figure 8 shows the details of mixed-level cosimulation adapter for the case of channel refinement from the example of Figure 7.

Figure 9 shows a snapshot of mixed-level cosimulation of the IS-95 system. In the figure, schematics and codes of SDL and SystemC representations are exemplified. The voice input waveform is sent from the user interface model to the encoder and the voice output waveform is received by the user interface model after the voice input has made a round-trip over encoder, Tx, base station, Rx, and decoder as shown in Figure 1 (a).

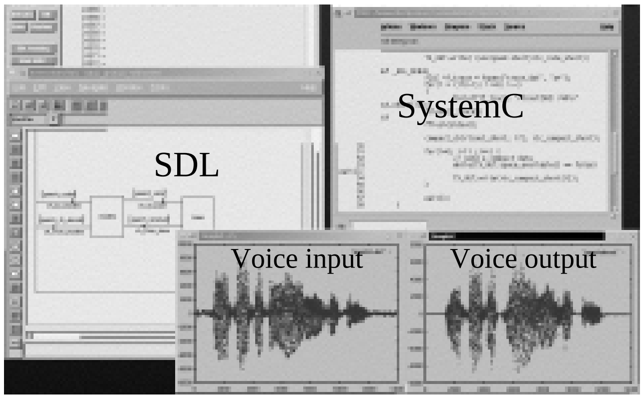


Figure 9: A snapshot of mixed-level cosimulation of the IS-95 system.

6 Conclusion

We proposed a method of mixed-level cosimulation between protocol-neutral and protocol-fixed communication levels. Since the proposed method enables mixed-level cosimulation in both channel-by-channel refinement and inter-level channel refinement, it provides the designer with fine granularity in communication refinement. In our case study, we applied the proposed method to the communication refinement of an IS-95 CDMA cellular phone system.

References

[1] J. A. Rawson et al., “Interface Based Design”, *Proc. Design Automation Conf.*, 1997.

[2] 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.

[3] Coware, Inc., “N2C”, available at http://www.coware.com/cowareN2C.html.

[4] Synopsys, Inc., “SystemC, Version 1.1”, available at http://www.systemc.org/.

[5] J-M. Daveau, G. F. Marchioro, T. Ben-Ismail, and A. A. Jerraya, “Protocol Selection and Interface Generation for HW-SW Codesign”, *IEEE Transactions on VLSI Systems*, vol. 5, no. 1, pp. 136–144, Mar. 1997.

[6] J-Y. Brunel, W.M. Kruijtzer, H.J.H.N. Kenter, F. Petrot, and L. Pasquier, “COSY Communication IP’s”, *Proc. Design Automation Conf.*, pp. 406–409, June 2000.

[7] R. Lysecky, F. Vahid, and T. Givargis, “Techniques for Reducing Read Latency of Core Bus Wrappers”, *Proc. Design Automation and Test in Europe*, pp. 84 – 91, Mar. 2000.

[8] R. Lysecky, F. Vahid, T. Givargis, and R. Patel, “Pre-fetching for Improved Core Interfacing”, *Proc. Int. Symposium on System Synthesis*, pp. 51 – 55, Nov. 1999.

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

[10] J. A. Rawson, “Hardware/Software Co-Simulation”, *Proc. Design Automation Conf.*, pp. 439–440, 1994.

[11] L. Séméria and A. Ghosh, “Methodology for Hardware/Software Co-verification in C/C++”, *Proc. Asia South Pacific Design Automation Conference*, Jan. 2000.

[12] K. Takemura, M. Mizuno, and A. Motohara, “An Approach to System-Level Bus Architecture Validation and its Application to Digital Still Camera Design”, *Workshop on Synthesis and System Integration of Mixed Technology (SASIMI)*, pp. 195–201, Apr. 2000.

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

[14] 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*, pp. 61–66, Oct. 1999.

[15] Verilog, Inc., *ObjectGEODE, ver 1.0*, 1996.

[16] F. Belina, D. Hogrefe, and A. Sarma, *SDL with APPLICATIONS from PROTOCOL SPECIFICATION*, Carl Hanser Verlag and Prentice Hall International (UK) Ltd., 1991.