

# Co-simulation of Wireless Local Area Network Terminals with Protocol Software Implemented in SDL

Petri Kukkala, Marko Hännikäinen and Timo D. Hämäläinen  
Tampere University of Technology  
Institute of Digital and Computer Systems  
P.O. Box 553, FI-33101 Tampere, Finland  
petri.kukkala@tut.fi

## Abstract

*This paper presents the verification of our WLAN terminal (TUTWLAN), with its medium access control protocol and test applications, using cycle-accurate hardware/software co-simulation. The protocol software has been implemented using SDL and automatic C code generation. The hardware implementation of the terminal contains hardware accelerators for time-critical protocol functions. Full system co-simulations were used for both the functional verification and performance evaluation of a single TUTWLAN terminal as well as a network of terminals. With simulations, the performance bottlenecks were identified, and the results enable the implementing of the next generation TUTWLAN terminal as a single-chip.*

## 1. Introduction

TUTWLAN is a proprietary WLAN developed at Tampere University of Technology (TUT) [8]. During years 1997-2001 a network with two generations of prototype terminals were implemented. Both are based on platforms consisting of a separate processor and 1-2 FPGA chips. Work for the new generation of TUTWLAN terminal as a single-chip implementation is being carried out.

A Medium Access Control (MAC) protocol (TUTMAC) has been designed in SDL, which is particularly suitable for systems having a discrete stimuli-response type of behavior. Due to the standardized formalism, SDL enables an automatic code generation for executable protocol software. This significantly enhances the productivity of the protocol development. New functionality can be easily added in SDL, which enables rapid prototyping in co-design.

In the process of implementing the TUTWLAN terminal as a single-chip, traditional ways for functional verification

and performance evaluation are not enough. The first problem is how to probe the desired signals, transactions and behavior due to a mixed hardware/software system, since a physical prototype is not yet available. The next problem is how to have sufficient coverage in the verification particularly due to problems in simulation performance. In this paper, we present the conducted hardware/software co-simulations to address these problems.

## 2. Related work

Co-simulation methods can be classified by the number and types of simulators (homogeneous, heterogeneous), and accuracy (functional, temporal) [1]. Homogeneous co-simulation uses a single language to describe both hardware and software, and heterogeneous co-simulation uses several languages. A dedicated simulator is typically required for each employed language, in which case, a co-simulation kernel is required to interconnect the simulators and manage their interaction [4]. Functional co-simulation disregards the details of timing and communication, and is focused on the functionality, whereas temporal co-simulation performs a more accurate validation including timing information, processor architectures, and bus models.

Concerning the use of SDL, the related work is mainly focused on co-design, and the development of different methods for partitioning the SDL specified system into hardware and software and generating code for them both [3] [11]. Instead, the utilization of co-simulation methods for the virtual prototyping of SDL generated systems is not widely researched. The closest topics are related to the co-simulation of the systems executing a Real-Time Operating System (RTOS) as both kind of systems concern the parallel execution of communicating processes [2] [5].

Our approach for the co-simulation with protocol software implemented in SDL is to use a heterogeneous and temporal method with cycle-accurate hardware models and target compiled protocol software [9]. This approach was

chosen for two reasons. First, the target integration of the SDL generated protocol software consists of the implementation of hardware intensive software routines targeted for a specific hardware platform. This requires accurate verification of hardware/software interfaces. Second, cycle accurate models are required for the reliable performance evaluation of terminals containing both hardware and software.

### 3. TUTWLAN

TUTWLAN is a proprietary WLAN, which solved the problems of scalability, quality of service and security present in standard WLANs [8]. TUTWLAN has a centralized topology, where one base station controls and manage multiple portable terminals. Several configurations have been developed for different purposes and platforms. In this we present one configuration of the network and TUTMAC protocol.

TUTMAC is a dynamic reservation Time Division Multiple Access (TDMA) based protocol designed for TUTWLAN. High processing capacity is required for supporting the real-time data transfer functions, including error coding, encryption, and TDMA scheduling.

Forward Error Correction (FEC) is used to avoid errors in the header of a TUTMAC frame. Cyclic Redundancy Check (CRC) algorithms are used to detect transmission errors; CRC-8 algorithm for headers, and CRC-32 for payload. Retransmissions are used to secure the data transfer if errors exist. The Improved Wired Equivalent Privacy (IWEP) algorithm is used to implement default encryption for the payload while stronger algorithms can be applied separately [6]. The TDMA scheduling implements the accurate frame synchronization between terminals.

The functions are performed for every packet sent and received by a terminal, which emphasizes their impact on performance. Especially the TDMA scheduling yields to demanding momentary real-time processing requirements. High accuracy enables to decrease inter frame space, which consequently increases the efficiency of an access cycle.

### 4. TUTMAC protocol implementation

The implementation of TUTMAC consists of platform independent software and platform dependent environment functions as well as hardware functions. The partitioning of the protocol between hardware and software is based on the classification of functions: all control functionality is implemented with software and time critical functions for dataflow processing with hardware.

The TUTMAC SDL design contains four main blocks, which are presented in Fig. 1. The functionality is described using state machines, and in all, the SDL design contains seven state machines, each having 1–15 states.

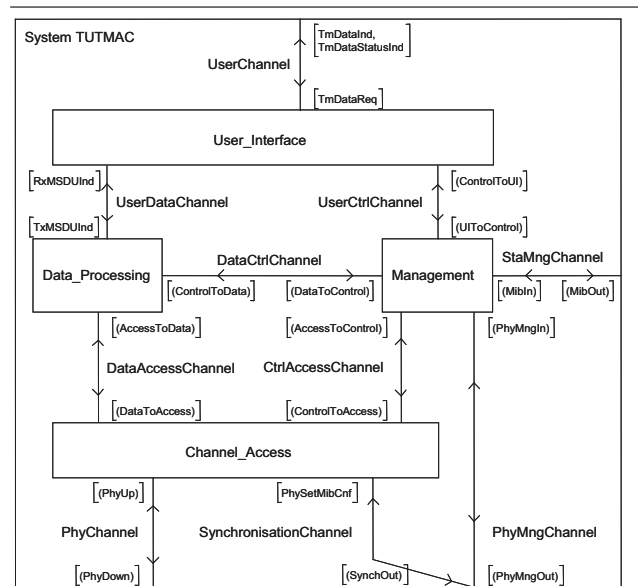


Figure 1. Blocks of the TUTMAC SDL design.

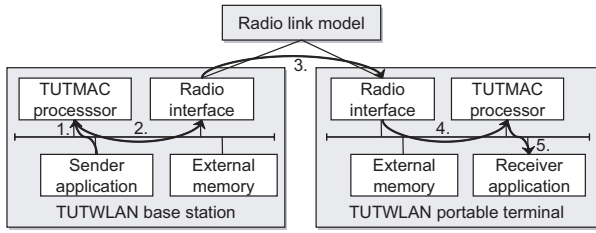
A transmitted user packet is processed by the user interface and data processing that validate the packet and construct a TUTMAC frame. The frames are stored to a frame queue in the data processing. The channel access extracts the frames, and sends them to a radio. Received frames are processed in reversed order to construct user packets.

The executable protocol software is implemented using automatic C code generation from the TUTMAC SDL design [7]. Environment functions are implemented for the target integration of the generated code with the target hardware. Both the generated code and the environment functions are compiled and linked for the target hardware. It should be noted that the environment functions are implemented for a target hardware only once. After that, the functionality in SDL can be modified rapidly.

The protocol software requires 200–300 kB of memory on target hardware. The size of the executable code is 84 kB, and the amount of required static data memory is 25 kB. 100–175 kB of dynamically allocated data memory is required for the queuing of data frames.

The hardware functions are divided into hardware accelerators and radio interface functions. The hardware accelerators are logically included in the data processing block of the TUTMAC SDL design. In the SDL simulation, the accelerators are implemented with SDL, but they are not included in the automatic C code generation. Instead, these blocks are manually implemented with VHDL.

In the hardware implementation, the accelerators treat transferred frames as a stream, and process all data with pre-determined delay. The radio interface functions control the physical interface of an external radio module.



**Figure 2. Data flow of a data transfer.**

## 5. TUTWLAN terminal hardware platform

The TUTWLAN terminal hardware platform consists of four Intellectual Property (IP) blocks: two processor blocks, an external memory block, and a radio interface block. The IP blocks are interconnected by a Heterogeneous IP Block Interconnection (HIBI) [10], and each block has a HIBI wrapper. All platform components are synthesizable and can be used to implement a physical chip.

The processor blocks are self-contained units for software execution. They include an ARM7 processor core, internal memory, real-time clock, and DMA controller. The external memory block has a memory controller and SRAM module. The radio interface contains the TUTMAC hardware accelerators and radio interface functions.

In addition to the hardware platform, the co-simulation of the terminal requires simulation models for an external radio module and physical layer. A separately implemented application is executed in an application processor to generate traffic for the network.

The radio link model emulates the external behavior of the radio and physical layer. The model generates the data, control, and clock signals of a real radio transceiver for the radio interface block. The network co-simulation are carried out with multiple TUTWLAN terminals interconnected by the radio link model that forwards transmitted data frames to the other terminals.

The data flow of a data transfer between the sender and receiver applications are presented in Fig. 2. The sender application starts the transmission, and transfers data to the TUTMAC processor over HIBI (Phase 1). The packets are processed by the protocol software. The constructed data frames are transferred to the radio interface (Phase 2).

The frames are stored in a frame buffer, where they are extracted when the frame synchronization starts the data transmission. The hardware accelerators process the frames, and send them to the radio link model. The frames are transferred to the receiving terminal (Phase 3).

The received frames go through the hardware accelerators and are transferred to the TUTMAC processor (Phase 4). The frames are again processed by the protocol software. The reconstructed user packets are transferred to the application processor (Phase 5).

## 6. TUTWLAN co-simulation

We use Mentor Graphics Seamless Co-Verification Environment (CVE) for the co-simulation of TUTWLAN terminals. The protocol software is executed on ARM Instruction Set Simulator (ISS), while the terminal hardware is simulated on ModelSim. The both simulators are separately controlled using their own simulator interfaces. Seamless CVE interconnects the simulators, and controls the communication between them.

Information about the performed co-simulations is collected by monitoring the execution of software and hardware using custom instrumentation. The models print information about the current state of the model in critical and otherwise interesting execution points. This textual trace contains the time, target processor, and detailed information of each event. A logic waveform from the VHDL simulation is also viewed.

### 6.1. Functional verification

The TUTMAC terminal hardware platform and the target integration of protocol software are functionally verified by co-simulations for two reasons. First, the complex behavior of the interfaces in the IP blocks is efficiently verified with test sequences initiated by software executed on the processor blocks. Second, the hardware intensive environment functions are easy to verify with the detailed view to the behavior of the hardware platform.

Each block in the hardware platform is verified using hardware/software test benches that contain a processor block and the block to be verified. The test cases are focused on a single detailed platform feature at a time, which is necessary to keep the co-simulation time reasonable short.

### 6.2. Performance evaluation

The performance is evaluated by sending packets between a TUTWLAN basestation and a portable terminal interconnected by the radio link model. The packets are sent and received by applications executed on application processors. The data flow of a data transfer between the applications is presented in Fig. 2.

The delays of the data flow can be extracted from the textual trace. Table 1 presents the delays divided into respective components of the platform and protocol. The delays are presented for an average case when a 1000 B user packet is transmitted.

The application delays are resulted from the handshake procedures between the processor blocks. The delays of HIBI transfers are proportional to the amount of transferred data. As seen, the capacity of HIBI does not restrict the performance of the TUTWLAN terminal.

| <i>Component type</i>   | <i>Delay</i> | <i>Proportion</i> |
|-------------------------|--------------|-------------------|
| Frame queuing (SDL)     | 1885 $\mu$ s | 48.7 %            |
| Radio transmission      | 798 $\mu$ s  | 20.6 %            |
| Frame buffering         | 643 $\mu$ s  | 16.6 %            |
| Hardware accelerators   | 215 $\mu$ s  | 5.5 %             |
| Protocol software (SDL) | 207 $\mu$ s  | 5.4 %             |
| Environment functions   | 107 $\mu$ s  | 2.8 %             |
| HIBI transfers          | 12 $\mu$ s   | 0.3 %             |
| Application             | 5 $\mu$ s    | 0.1 %             |
| Total                   | 3872 $\mu$ s | 100.0 %           |

**Table 1. Average delays of data transfers.**

The delays caused by the environment functions and the SDL implemented protocol software, excluding the frame queuing, are due to the passing of SDL signals and the process scheduling in software execution. The delay caused by the protocol software is 5.4 % of the total transfer delay. Thus, the performance of the TUTMAC processor can be considered as sufficient for the execution of the protocol software.

The frame queuing (in SDL) and buffering (in radio interface) delays can be attributed to the protocol behavior with TDMA. The queuing is divided into these two parts because the frame synchronization is implemented twice: in the TUTMAC SDL design for preliminary scheduling, and in the hardware accelerators for more accurate final scheduling. The computation on hardware causes only a minor delay since the largest impact on performance comes from the radio link speed.

The total transfer delay of a packet from the sender to the receiver application, in this case, is 3.87 ms. The data throughput with the average transfer delay and packets of 1000 B is 2.1 Mb/s which is 81 % of the theoretical maximum (2.6 Mb/s). The theoretical maximum was calculated for the utilized 11 Mb/s radio link speed, and one 1500 B data channel.

Co-simulation with accurate models is a slow process. However, relevant information about the network performance, with hundreds of user packets sent, was gathered in 7–8 hours. Comparison between few different implementation alternatives by modifying the TUTMAC SDL model as well as hardware platform was feasible in 3–4 days. Particularly, the communication between the blocks and the different hardware accelerators were evaluated.

## 7. Conclusions

The functional verification with co-simulation facilitated the target integration of the TUTMAC SDL design notably by providing a detailed view into both hardware and software. Further, the results of co-simulation were used to

identify the performance bottlenecks. The effortless modification of the SDL design as well as hardware platform enabled the rapid exploration and evaluation of different implementation alternatives.

Co-design and co-simulation were the only practical way to design the new single-chip version of the TUTWLAN terminal. In addition, a TUTWLAN network with several terminals, each with cycle-accurate behavior, gave new possibilities to virtual prototyping. This significantly extended the normal SDL level protocol design and verification and gave better test coverage.

## References

- [1] A. Amory, F. Moraes, L. Oliveira, N. Calazans, and F. Hessel. A heterogeneous and distributed co-simulation environment. In *Proc. 15th Symposium on Integrated Circuits and Systems Design*, pages 115–120, Sept. 2002.
- [2] M. Bradley and K. Xie. Hardware/software co-verification with RTOS application code, Oct. 2003. Available: <http://www.mentor.com/soc/techpapers/>.
- [3] J.-M. Daveau, G. Marchioro, and A. Jerraya. Hardware/software co-design of an ATM network interface card: a case study. In *Proc. 6th International Workshop on Hardware/Software Codesign*, pages 111–115, Mar. 1998.
- [4] M. El Shobaki. Verification of embedded real-time systems using hardware/software co-simulation. In *Proc. 24th Euro-micro Conference*, pages 46–50, Aug. 1998.
- [5] D. Harris, D. Stokes, and R. Klein. Executing an RTOS on simulated hardware using co-verification. In *Proc. Embedded Systems Conference Fall 2000*, 2000.
- [6] P. Hämäläinen, M. Hännikäinen, T. Hämäläinen, and J. Saarinen. Hardware implementation of the improved WEP and RC4 encryption algorithms for wireless terminals. In *Proc. European Signal Processing Conference*, pages 2289–2292, Sept. 2000.
- [7] M. Hännikäinen, J. Knuutila, T. Hämäläinen, and J. Saarinen. Using SDL for implementing a wireless medium access control protocol. In *Proc. International Symposium on Multimedia Software Engineering*, pages 229–236, Dec. 2000.
- [8] M. Hännikäinen, T. Lavikko, P. Kukkala, and T. Hämäläinen. TUTWLAN - QoS supporting wireless network. *Telecommunication Systems - Modelling, Analysis, Design and Management*, 23(3,4):297–333, 2003.
- [9] P. Kukkala, T. Kangas, M. Hännikäinen, and T. Hämäläinen. SDL generated protocols in Seamless co-verification environment. In *Proc. IEEE Design and Diagnostics of Electronic Circuits and Systems Workshop*, pages 158–165, Apr. 2002.
- [10] E. Salminen, V. Lahtinen, T. Kangas, J. Riihimäki, K. Kuisilinna, and T. D. Hämäläinen. HIBI v.2 communication network for system-on-chip. In *Lecture Notes in Computer Science*, volume 3133, pages 413–422, 2004.
- [11] F. Slomka, M. Dorfel, and R. Munzenberger. Generating mixed hardware/software systems from SDL specifications. In *Proc. 9th International Symposium on Hardware/Software Codesign*, pages 116–121, Apr. 2001.