

Wireless Protocols Design: Challenges and Opportunities

J. L. da Silva Jr., M. Sgroi, F. De Bernardinis, S. F. Li,
A. Sangiovanni-Vincentelli, J. Rabaey

University of California at Berkeley

e-mail: {julio, sgroi, fdb, suetfei, alberto, jan}@noyce.eecs.berkeley.edu

ABSTRACT

Modern wireless communication systems require the deployment of increasingly complex protocols that satisfy tight requirements at low implementation cost, especially in terms of size and power consumption. Most protocol design methodologies currently in use are inadequate, either because they do not rely upon formal techniques and therefore do not guarantee correctness, or because they do not provide sufficient support for performance analysis and design exploration and therefore often lead to sub-optimal implementations. Therefore, we use a refinement-based formal methodology that relies upon the orthogonalization of function and architecture design and emphasizes the use of formal models to ensure correctness and reduce design time. In this paper we present a case study, the Intercom, consisting of a network of mobile terminals supporting voice communication among end users. We use this case study to validate the methodology and identify directions of further research.

Keywords

Case Study, Wireless Protocol Design, Communication Refinement, Function/Architecture Co-design.

1. INTRODUCTION

Wireless systems usually consist of networks of *mobile terminals* and remote *servers* (e.g. coordinating the network operation or handling data storage). The design of a single network unit, terminal or server, cannot be done independently on the rest of the network. First, it is necessary to partition the functionality between network nodes, and design the protocol, i.e. the set of control messages and data packets that are exchanged between the communicating units and the set of rules that define when data is transmitted and received. This requires, among the other things, to select the network topology and derive from the requirements of the whole network the functional and performance constraints that each network unit has to satisfy.

A wireless terminal is a complex system that implements on a single chip a variety of functions: protocols, signal processing, position location, radio modems, RF transceivers, A/D converters.

While the integration of so diverse components presents itself an interesting research topic, in this paper we mainly focus on the problem of designing correct and efficient protocols.

Protocols are usually described using a layered structure, each layer implementing the communication at a different level of abstraction. Different layers handle different data types and formats (from multi-field messages at the top layers to raw bits at the bottom layers) and are usually subject to different timing requirements. ISO has defined a standard (OSI Reference Model) that classifies the typical protocol functions into seven distinct layers. The main advantage of identifying layers from the beginning of the design process is that it allows to decompose the protocol design problem into the design of a number of subproblems (layers) delimited by well-defined interfaces. However, decomposing into a number of smaller problems may imply penalties as it prohibits inter-layer optimizations.

In the wireless communication domain, the protocol design problem presents additional challenges: the limited battery life and the dynamically changing network conditions (e.g. due to physical medium variations or user mobility...) require to define protocols that are energy efficient and dynamically adaptive. Previous work [5] has already shown the importance of considering energy efficiency as a key parameter in the design starting from the top protocol layers. Power consumption, for example, can be significantly reduced by shutdown of the inactive terminal components or by reducing the data to be transmitted at the expense of increased data compression.

Protocol specifications are heterogeneous in the sense that they include both *control* and *data processing* functions. Data processing, e.g. in encryption, error correction, coding algorithms, is typically applied at regular time intervals to streams of incoming data and is often subject to tight timing constraints. Control functions are of two types: *real-time*, if data processing is enabled by the occurrence of external (e.g. user request) or internal (e.g. timers) events, *data-dependent*, if some action is taken as a consequence of a data value (e.g. depending on the CRC result a packet is discarded). These heterogeneous specifications are usually implemented by mapping them into a *heterogeneous architecture* including custom logic, programmable logic and an embedded processor.

Designing protocols is difficult due to the complexity of the task and the tight time-to-market constraints. The protocol design methodologies commonly used in industry are rather informal since they involve a sequence of trial-and-error steps that terminates when the designers reach a certain level of confidence in their design. Not relying upon formal techniques, at the end of the design process there is no guarantee that the implementation satisfies the requirements of the initial specification. Moreover, a

careful performance analysis is not carried out, this resulting in a limited design exploration, based mainly on designer experience. To obviate these drawbacks, we developed a design methodology based on:

- a *formal* model of computation for mixed control and dataflow specifications
- a simulation environment for complex protocols that allows full design space exploration and evaluation of tradeoffs based on *performance analysis*
- a set of fast and accurate *estimation* techniques to evaluate the implementation cost with respect to the key design parameters, e.g. speed, power.
- a set of synthesis and optimization algorithms to derive *correct-by-construction* and *efficient* protocol implementations

The protocol design methodology that relies upon the basic foundations of the POLIS methodology [1] has been modified and extended to address the challenges of this new application domain. To illustrate these aspects, we report here how the methodology has been applied to a real case study, the Intercom, a network of mobile terminals supporting voice communication among end users.

The paper is organized as follows. Section 2 gives an overview of the methodology. Section 3 presents the Intercom case study: first, we describe the functional specification and the performance constraints of the protocol stack and show how they were derived from the overall system requirements; then, we present the results of the design exploration that has allowed us, among the other things, to select an architecture. In Section 4 we discuss the main lessons learned from the case study.

2. DESIGN METHODOLOGY

The basic tenet of our methodology is the orthogonalization of concerns, first and foremost the separation between function and architecture. The architecture/function orthogonalization principle can be applied to all levels of the design hierarchy. The architecture input to a level below can be considered as a functional spec at the lower level of abstraction. Constraints are propagated from one level to the next to make sure that, if the design satisfies the constraints at this level there is no need to verify further up in the hierarchy. We refer to this aspect of the methodology as *successive refinement*.

The design methodology described below (Figure 1) is based on both the POLIS system [1] developed at UC Berkeley and its industrial relative Virtual Component Co-design (VCC) developed by Cadence Design Systems [2].

Functional Specification – The system specification is captured in a pure functional way and validated through functional simulation. Functional specification is a necessary step to guarantee a correct choice between implementation alternatives (e.g., hardware or software). In order to operate effectively, a proper model of computation needs to be selected for the specification. We opted for Co-design Finite State Machines (CFSMs, an extension of FSMs) as underlying model of computation because they are intrinsically good in modeling control paths [1]. CFSMs also can include data computations as part of the transitions, and can therefore model protocol datapaths as well. CFSMs networks are globally asynchronous, locally

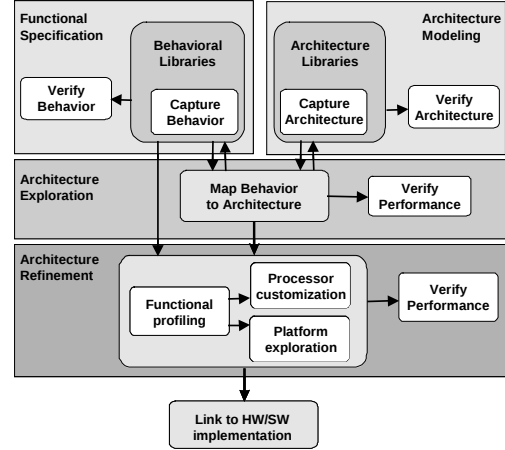


Figure 1 – Design Flow

synchronous; thus they are able to model effectively different HW/SW partitions, and asynchronous communication events.

Architecture Modeling – A set of parameterized alternative architectures that can be used to implement the specification is modeled. Architectures are described in terms of programmable units (CPU's), ASIC's, interconnect networks, and *real time operating systems (schedulers)*. They capture the computational capabilities, the degree of parallelism, the sequentialization of blocks sharing the same resources, and the capacity of the inter-block communication channels.

Architecture Exploration – The functional blocks are mapped onto the candidate architecture. The performance of the resulting designs is then estimated and compared. The mapping consists of associating functional blocks with architectural resources. This mapping affects the behavior of the mapped system by introducing time delays. The effects of these implementation-induced delays can be analyzed through performance simulations.

Architecture Refinement – After the number of possible architectures has been restricted, functional profiling can be used to extract regular and reoccurring operations in protocol processing. On the hardware side, different implementation platforms can be explored, such as configurable, standard cell and custom implementations. On the software side, the effort is mainly focused on the customization of a configurable processor to suit the needs of protocol processing.

Synthesis – After architecture selection and HW/SW partitioning, automatic synthesis of the hardware and software components can take place.

3. CASE STUDY: INTERCOM

The Intercom is a single-cell wireless network supporting full-duplex voice communication among up to twenty mobile users located in a small geographical area (100 m²). The network includes a number of units, called *remote terminals* that operate in one of the following three modes: *idle* (the remote is switched on but has not subscribed to the network and cannot participate to a conference with other remotes), *active* (the remote has subscribed to the network and is enabled to participate in a conference), *communicating* (the remote is participating in a conference). Each remote can request one of the following services: subscription to enter active mode, unsubscription to return to idle mode, query of active users, conference with one or multiple remotes, and

broadcast communication. The system specification includes also performance requirements on the transmission of voice samples, e.g. latency (below 200 ms) and throughput (64 kbps), as well as requirements on low power consumption.

3.1 Protocol Stack

In this section we describe the specification of the Intercom protocol stack and show how it was derived from the system requirements (in a top-down manner) and the wireless channel properties (bottom-up). This process has required, among the other things, to select algorithms (e.g. CRC for error control and TDMA for medium access control) and define a number of key design parameters (e.g. messages and header format, frame length and number of slots).

The first step was to define the network architecture. Since voice is the type of data transmitted over the network, we have selected a centralized network architecture including a special unit, called *base station* that coordinates the network operation. The base station only provides control functionality, while data channels carrying voice are set up as peer-to-peer links between remote terminals. The base station keeps track of the evolving network configuration using an *internal database* that stores information about the currently active users and the IDs (e.g. a slot in TDMA, a code in CDMA...) of the physical channels. Making the implementation choice of providing each Intercom unit with identical capabilities (i.e. each unit can operate as either a base-station or a remote depending on the mode set on power-on) has allowed us to design, at the price of an additional implementation cost¹, a protocol that is robust in case of failures (it enables on-the-fly reconfiguration if the unit operating as base-station fails).

Figure 2 shows the structure of the protocol stack, which is composed of the following layers:

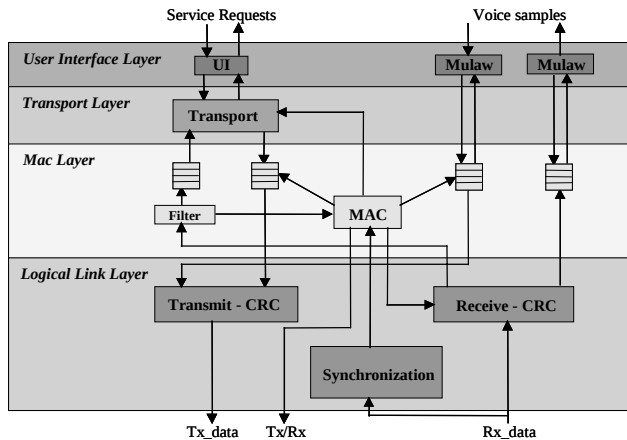


Figure 2 – Intercom Protocol Stack

User Interface Layer. The UI module interfaces the remote terminal with the environment, capturing user service requests and displaying their current status. The UI filters user requests and, to avoid waste of resources, forwards only the relevant ones to the transport layer (for example, the UI discards a StartConference request if the user is not in active operation mode). The Mulaw

¹ In the final implementation we present in the next section, this additional cost consists of two CFSMs.

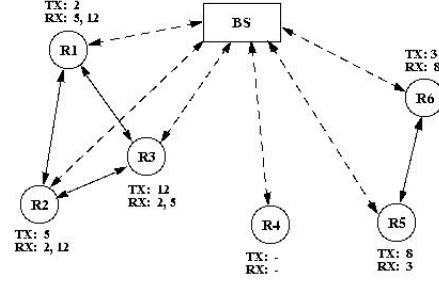


Figure 3 – Intercom Network

module performs logarithmic quantization of a PCM-encoded bit stream, sampled at 64 kbps.

Transport Layer. The Transport Layer defines the format and the retransmission policy for the control messages that are exchanged between the remotes and the base station for network operation management (e.g. user subscription, connection setup). These messages contain information such as remote ID and type of service request and are retransmitted a number of times until an acknowledgment is received.

Mac Layer. For medium access control, we have selected Time Division Multiple Access (TDMA), since it is an appropriate policy for guaranteeing the Quality of Service requirements of voice communication. In a TDMA scheme a physical channel, implementing a virtual channel defined at the application layer, is uniquely identified by a set of two or more time slots (the number of slots depends on the number of users involved in the conference and the bandwidth assigned to each user). Slots are allocated on a per-demand basis. When a remote wants to start a conference, it sends a request to the base station. If the base station detects the availability of a physical channel, it communicates the remotes the slots in which they can transmit and receive. At the remote, this information is stored within an *internal table*, so that, at the beginning of each new slot, the MAC layer can read the table and activate either the transmit or the receive functions. To implement the TDMA scheme, the MAC Layer includes also a set of *queues*, one for each flow of data. Queues are used to shape the input and output flows, e.g. storing the outgoing data during the slots when other remotes are transmitting over the channel. The TDMA slots are allocated as follows: the first two are used for control messages, one uplink (from the remotes to the base station) and one downlink (from the base station to the remotes) while the remaining slots are used for voice communication among users. A frame lasts 62.5ms, while each of the twenty slots lasts 3ms. These values have been derived from the requirements on delay and throughput. An example scenario is shown in Figure 3. Six remotes are active: R1, R2, and R3 are involved in a conference, R5 and R6 are in one-to-one conversation, and R4 is not communicating with anyone. The annotations next to each remote show its transmission and reception slots (R1 transmits in slot 2, R2 in slot 5, R3 in slot 12).

Logical Link Layer. This layer includes the following modules. Transmit/Receive perform a Cyclic Redundancy Check (CRC) function that allows detecting errors in the received packets and eventually discarding them, and a filtering function that distinguishes bit patterns equal to the frame and slot synchronization pilots. Synchronization *detects* frame and slot *synchronization pilots* in the bit stream coming from the wireless

channel. Then, using also a timer and a slot counter, it notifies to the MAC layer the number of the new slot.

Physical Layer. The Physical Layer includes intensive bit-level data processing such as QPSK (de)modulation, timing recovery, phase/frequency correction. The Direct Sequence Spread Spectrum radio operates at a rate of 1.6 Mbps at 2.4 GHz. The design of the physical layer is not in the scope of this paper.

3.2 Design Exploration

This section presents the results of the design exploration followed to select an implementation of the protocol stack. First, we describe the process of architecture evaluation and HW/SW partitioning of the entire protocol stack performed in the Cadence VCC [2] environment. Then, we consider in detail the Transport and the MAC Layers and provide more precise measures of their implementation cost.

The functional specification of the protocol stack has been decomposed in 46 CFSMs and each CFSM has been described either using Whitebox C² or as a State Transition Diagram. Table 1 presents the complexity of each protocol layer.

The target architecture includes an embedded processor,

Layer	CFSMs	C-code (lines)	State-transition Diagram (states)
User interface	1	100	-
Mulaw	2	100	-
Transport	5	300	-
MAC	23	270	42
Transmit	6	120	16
Receive	6	140	2
Synchronization	3	-	17

Table 1 – Complexity of the Intercom Specification

programmable logic, custom logic, a memory sub-system, a communication backplane. The basic architecture has been described using parameterizable modules from the VCC library extended with processor models. We have used conservative performance models so that the real implementation will certainly satisfy the timing requirements. The architecture model consists of the following blocks:

- an embedded processor model parameterized with respect to the clock frequency,
- an RTOS scheduler (Cyclo-Static, Static Priority, Round Robin, etc.) that is parameterized with respect to context switch, suspend, resume and other overheads,
- an ASIC delay model,
- a bus model, that includes both the bus bandwidth (operating frequency and word length) and the operating policy (FIFO, Time Sliced, etc.)

Mappings of the functionality onto the parameterized architecture have been done in the Cadence VCC environment. Figure 4 presents three mapping experiments, done using an ARM model for the processor, a processor frequency of respectively 1, 11 or 200MHz, a preemptive RTOS (tasks at higher levels of the

protocol were assigned lower priorities), an RTOS overhead of 200 cycles, and an ASIC delay of 10 ns for each hardware module. Figure 4 shows for each mapping the processor utilization, including the RTOS overhead. At lower processor frequency, fewer modules can be implemented in software without timing violations (detected as event losses).

The following modules are mapped onto the processor: for the experiment at 1MHz, transport and user interface, at 11MHz we add the Mulaw module, and at 200MHz we add part of the MAC layer. In the mapping at 1MHz only network control functions (e.g. connection setup), with very few context switches and loose time constraints, are implemented as software. When we added a data processing function (Mulaw module) with tighter timing constraints to the software partition with the processor still running at 1MHz, we detected losses of critical events. Only after increasing the frequency to 11MHz, the Mulaw could be executed with no event losses. In the mapping at 200MHz the MAC layer function requires most of the execution time. The low utilization of the processor in this case is due to the MAC operating in burst mode: the processor works mostly during transmit/receive slots, otherwise it waits. In our simulation experiments the processor has one slot allocated for data transmission in a total of 16 slots. We simulated 14 frames, 4 with data transmission and 8 with control transmissions.

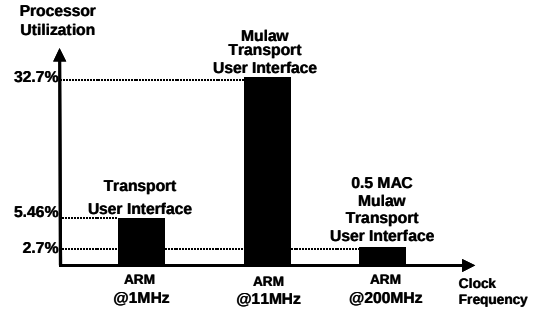


Figure 4 – Architecture Exploration

The design methodology allowed us to perform dozens of different performance simulations in a few days, mapping each of the 46 functional blocks into either hardware or software and also by changing the parameters in our architecture. None of the functional blocks in the Transmit, Synchronization, and Receive layers can run successfully into the ARM processor. However, increasing the clock speed of the processing unit is useless due to RTOS overhead.

3.2.1 Transport Layer

We have conducted an experiment concerning the functional profiling of the transport layer and the processor characterization step needed for architecture definition. The protocol implementation used for this experiment is based on C code generated from an SDL specification using Telelogic Tau [3]. All the data involved in the profiling have been derived in the Tensilica Xtensa [4] processor design environment. The Tensilica processor generation tools were used to configure a sample ARM like processor and then tune the architecture optimally for the protocol application.

The experiments have highlighted the following issues. 1) A significant amount of execution time is spent in memory

² White-box C is ANSI C plus port pragmas.

management routines (32% CPU time on *calloc*, 11% on *memcpy*). 2) Due to the lack of a branch prediction scheme, many instructions flushed by taken branches. A simple branch prediction scheme would improve performance of 12%. 3) Telelogic SDL generated code is excessively modular. 14.7% of all instructions are for procedure calls.

Our results indicated a significant amount of execution time spent in memory management routines, and many instructions flushed by taken branches. The protocol application also has an unusually large number of short function calls. Some of these problems are inherent of the protocol processing, such as excessive looping and branching and the maintenance of a slot set database in memory, others, such as excessive procedure calls and inefficient memory management, are caused by the inefficiency of the code generation tool. Thus, it is crucial to have both: better tools and compilers to generate more efficient code from high-level language descriptions of the protocol and efficient memory management schemes, that can be achieved using the methodology presented in [8]. Adding special instructions to handle branching and memory access should be very helpful.

3.2.2 Mac Layer

Energy efficiency is a key parameter to be considered in HW/SW partitioning, since the variations in power consumption between different implementations may be significant. To quantify this difference we have mapped the control part of the Intercom MAC layer into three different platforms: ASIC, FPGA, ARM processor and made a comparison from the energy perspective. For the ASIC platform power measurements have been done using Epic Tools, for the ARM using a table containing information on the consumption of each instruction. All simulations have been run at 25MHz with 3V supply voltage and using the same technology.

Table 2 quantifies the dependency of the power consumption from the platform: the same MAC Layer function, if implemented as software running on an ARM processor, requires more than 400 times the power consumed by an ASIC implementation.

	ASIC	FPGA	ARM
Power	0.26mW	2.1mW	114mW
Energy	10.2pJ/op	81.4pJ/op	$n \cdot 457pJ/op$

Table 2 –Power Consumption for different implementations

4. CONCLUSIONS

The design of the Intercom protocol has highlighted strengths and weaknesses of the methodology and suggested directions of further research. In particular

1. CFSM [1] is a well-suited model for wireless protocols. It allows representing *both the control and the dataflow components* and therefore permits to fully explore the design space (and make optimizations) across the two domains. One limit of the model is that CFSMs communicate using one-place buffers and non-blocking write communication semantics. This implies that, whenever a new event is emitted over a channel before the previous one is consumed, the latter is overwritten and lost. Modeling lossless communication using CFSMs is possible but usually requires additional overhead in the form of explicit request/acknowledgement protocols. The design experience with this case study has further motivated the definition of a new model, called Abstract CFSMs (ACFSMs) [6].

2. We were able to perform fast and extensive design exploration. To specify, validate, and perform a preliminary architecture exploration the Intercom protocol required us approximately 4 person/months (including time necessary to learn the methodology and get familiar with the tools).

3. Performance analysis was essential to select an architecture where most of the protocol is implemented in hardware to satisfy timing and low-power constraints and only the higher protocol layers are implemented as software tasks.

4. We have separated the design of the physical layer from that of the rest of the protocol stack and defined the interfaces between the two. The design should now be extended to include also the physical layer. This will open new challenges especially in developing new abstractions and, possibly, supporting tools to deal with the increased design complexity.

5. Functional profiling may be used to recognize critical operations in protocol processing and enable the realization of a very efficient implementation of the overall system. Past work has shown that functional profiling is very useful for data processing intensive systems [6]. We are confident that it may also apply to control intensive systems.

6. In the wireless application domain, energy-efficiency is a key design parameter both at the functional and at the architectural level, as we have shown in Table 2. Our group is doing further research to identify fast and reliable power estimation techniques through all the layers and new algorithms to reduce power.

In conclusion, the Intercom case study has shown that a methodology based on the communication refinement paradigm and the orthogonalization of functionality and architecture is effective in designing wireless protocols since it supports full and fast exploration from the early stages of the design process. This paper has described the design the Intercom system from the network-level specification to the final implementation and presented experimental results from the design exploration.

ACKNOWLEDGMENTS

We acknowledge the contributions of Vandana Prabhu, Fred Burghardt, Chunlong Guo, Xuejue Huang, Per Bjureus, Kevin Camera, and Tim Tuan.

REFERENCES

- [1] F. Balarin, et al.. *Hardware-Software Co-Design of Embedded Systems: the POLIS approach*. Kluwer Academic Publisher, 1997.
- [2] Ciertov Virtual Component Codesign (VCC). Cadence Design Systems. <http://www.cadence.com/technology/hwsw/ciertovcc/>
- [3] Telelogic, Inc. <http://www.telelogic.com/>
- [4] Tensilica, Inc. <http://www.tensilica.com/>
- [5] P. Lettieri, M. Srivastava. *Advances in Wireless Terminals*. IEEE Personal Communication. Feb 1999. Vol.6 No.1
- [6] M. Sgroi, L. Lavagno, A. Sangiovanni-Vincentelli. *Formal Models for Embedded System Design*. To appear on the IEEE Design & Test of Computers. Special Issue on System Design, '00.
- [7] S. F. Li, M. Wan and J. Rabaey, Configuration Code Generation and Optimizations for Heterogeneous Reconfigurable DSPs, SIPS99, Taipei, Taiwan.
- [8] S. Wuytack, J. L. da Silva Jr., F. Catthoor, G. de Jong, C. Ykman, *Memory Management for Embedded Network Applications*. IEEE Transactions on Computer-aided Design, May 1999.