

Embedded real-time communication protocol development using SDL for ARM microprocessor

With the rising complexity of communication protocols, designers are provided with highly efficient model-based tools to increase the level of abstraction at which the developers work. Such tool is SDT of Telelogic, that uses the telecommunication standard Specification and Description Language (SDL) and can produce application code automatically from graphical models representing system structure, behaviour and communication. It also allows the designer to simulate and verify the system behaviour based on these graphical models. It is clear that these tools can improve productivity, quality and reuse, thus allowing fast time to market with highly effective and sophisticated devices. An example of such a development is being elaborated in this article.

INTRODUCTION

With the rising complexity of communication protocols, designers are provided with highly efficient model-based tools to increase the level of abstraction at which the developers work. Such tool is SDT of Telelogic, that uses the telecommunication standard Specification and Description Language (SDL) and can produce application code automatically from graphical models representing system structure, behaviour and communication. It also allows the designer to simulate and verify the system behaviour based on these graphical models. It is clear that these tools can improve productivity, quality and reuse, thus allowing fast time to market with highly effective and sophisticated devices.

An example of such a development is being elaborated in this article. It concerns the OMI project ASPIS (ESPRIT 20287), which deals with the design of an ARM based processor intended for multi-mode mobile cellular phones supporting both DECT and GSM/DCS-1800 standards. For simplicity, we shall focus our interest on the DECT software implementation and more particularly on the MAC layer software architecture where the separation between hardware and software usually occurs. Also the integration of the DECT protocol with a Real-Time Operating System (RTOS) will be fully described and detailed.

The next point of focus is the need to achieve development of the software that implements the communication protocol in parallel with the designed hardware. This need is fulfilled by the use of a software model of the system, which behaves as a virtual hardware prototype. This framework allows developers to simulate the communication protocol application under development along with the RTOS and the low-level drivers code in a fully customisable software environment without the need for any hardware support. Before proceeding in detailing all the above-mentioned points, a brief introduction to the ASPIS processor architecture is necessary.

ASPIS PROCESSOR

The ASPIS processor implements the baseband signal processing, the protocols, the Man-Machine Interface (MMI) as well as the overall operation of a complete dual mode DECT/GSM handset. A detailed view of the chip's architecture, which is presented in Figure 1, includes the following functional units:

- The industry-standard ARM7TDMI 32-bit RISK CPU with the THUMB extension for 16-bit compressed code support. The ARM microprocessor handles the protocols, Man-Machine Interface (MMI), various low complexity software functions and overall system control. For debugging purposes an ICEBreaker debug unit is added which is accessible via a standard JTAG port;
- A custom-optimised (Application Specific Instruction set Processor - ASIP) DSP processor which handles the heavy functions required by DECT and GSM such as voice processing;
- RF and audio interfaces units;
- A number of memory mapped control and support peripherals including a buzzer, SIM-card reader, a LCD display driver and a keypad interface;
- A full duplex RS232-compliant UART;
- Interrupt controller, managing 26 internal a 2 external interrupt sources according to a fixed priority scheme;
- A 16-bit wide external memory port, common for the ARM and DSP cores, managed by a dedicated MMU/arbiter block. For time critical routines running on ARM7TDMI and DSP, an internal embedded SRAM is available. The size of the internal SRAM for both the ARM and DSP is respectively of 2Kx32bit and 1Kx16 bit.
- A central address-decoder for accessing the ASPIS memory-mapped units;
- A selectable clock source between two on-chip crystal oscillators, driving the PLL for clock multiplication;
- Power and clock distribution control block under

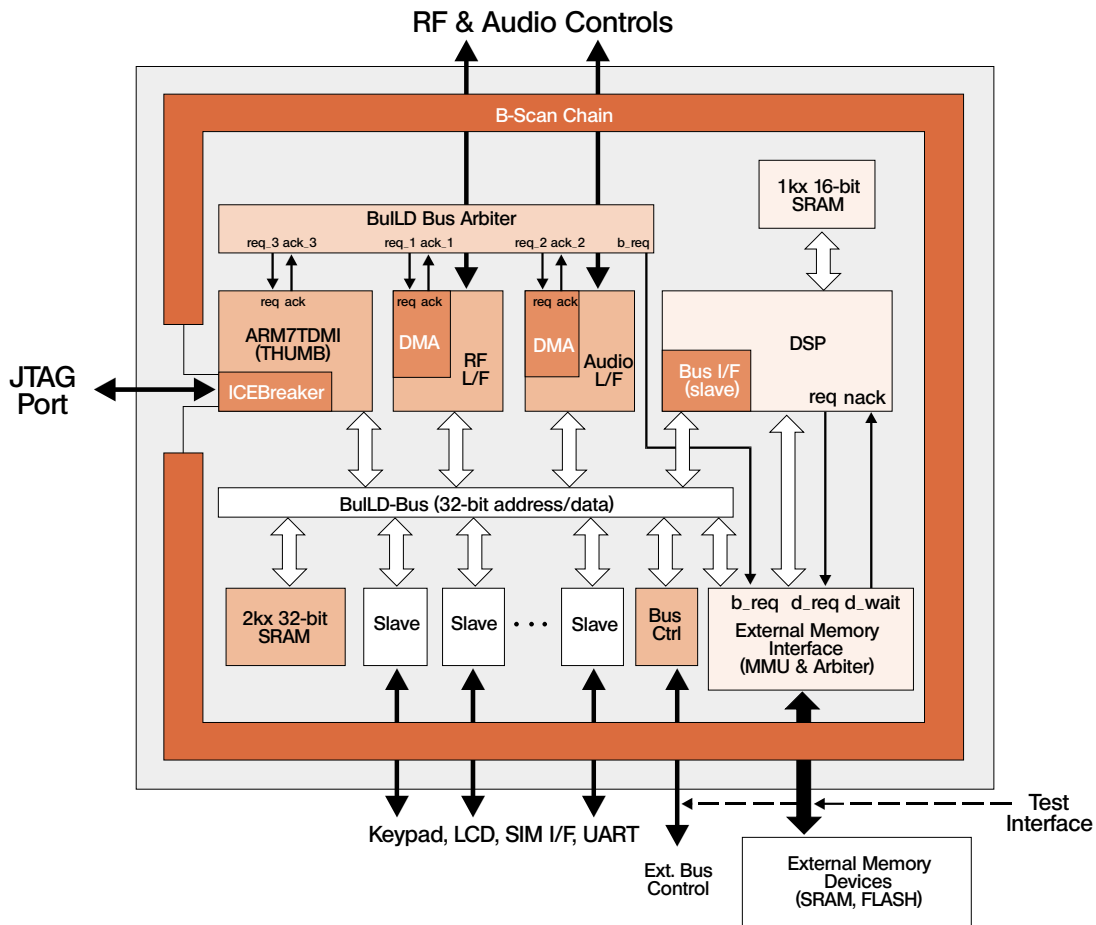


Figure 1. ASPIS processor architecture

software control and watchdog supervision;

- A Boundary Scan Chain, connected to the TAP controller of ICEBreaker, used for system and connectivity testing;
- Clock gating design for low power operation;
- Two crystal oscillators combined with a PLL for frequency multiplication.

An advanced co-simulation methodology has been undertaken during the development of the ASPIS processor in order to ensure that the ASPIS processor will be designed according to the specifications. Two co-simulation environments have been developed: a VHDL co-simulation environment and, an advanced model of the ASPIS processor to be used within the ARMulator debug tool. The first environment is being used for the validation of the ASPIS processor design, while the second, for the cross-development of the DECT/GSM protocols for the target ASPIS platform.

SDL SOFTWARE DEVELOPMENT

The ASPIS software parts include full DECT protocol stack, in compliance with the Generic Access Profile (GAP) which applies for all systems that provide voice services. SDL was used for the design, the validation and implementation of the DECT protocol. But certainly among all the facilities that were provided with the SDT tool, automatic C code generation has proven to

be the most useful. Indeed, it allowed a fast and easy integration of the protocol with the hardware target and the RTOS. However, it should be noted here that in general only Operating Systems (OS) that are supported by the SDL tool can be used. In the particular case of the ASPIS project, Virtuoso OS was chosen which isn't part of the list of OS systems that SDT tool supports. Unfortunately, SDT does not supply a common template file or a general method to achieve integration with a non supported RTOS. Instead, one should choose as a template the RTOS dependent files that are closest in properties to the new RTOS. These files should then be tuned to fit the newly used RTOS. This precise part of the development was carried out within the Software Technologies OMEN (ESPRIT 27874) project and is one of the major successful results obtained.

A slice of the GSM protocol stack has been also developed but for the purposes of this article we shall not detail it any further.

MAC LAYER SOFTWARE ARCHITECTURE

The MAC layer implementation of a DECT handset is composed of two parts: the upper layer which contains the multibearer control (MBC) process and the lower MAC layer which contains three processes (meaning SDL processes); these are: the General process, the

traffic bearer control (TBC) process and the Synchronisation and Monitor bearer (SnMnBC) process. There is also the lower management entity process that communicates with each of the previous mentioned processes.

The MBC process, controls the traffic bearer for a connection-oriented service, in our case the speech connection. It communicates directly with the TBC process ordering it to establish or release a traffic bearer. The TBC process in turn communicates with the ASPIS processor's RF interface by using a set of dedicated RF primitives. Such a primitive is for example the 'SyncBearer.Req' primitive that instructs the RF interface to open a synchronisation slot to get synchronised with the DECT base station. The reply of the RF interface when the synchronisation word is found should be a primitive that we call 'SyncBearer.Cfm'. This primitive instructs the MAC layer that we are synchronised with the base station and that it can proceed with getting extra identity information or proceed with a traffic bearer connection establishment (in other words start a speech connection). There are more than forty basic primitives that can be exchanged between the lower MAC layer and the RF interface all involving specific DECT operations such as the establishment of RSSI measurements, slot quality control and messages exchanged between the peer parts of the DECT protocol layers at the base station and the mobile phone.

The 'General' process handles the RSSI measurements and the paging messages. As for the SnMnBC process, it handles the establishment of synchronisation bearers and monitoring bearers. The lower management entity process contains important DECT functional parameters and manages the overall operation of the MAC processes.

INTEGRATION METHODS WITH THE RTOS

There are two distinct integration methods with an RTOS. These are called light and tight integration. The main difference between these two integrations resides in the mapping of SDL processes into OS system tasks; in light integration all SDL processes are mapped to one OS task while in tight integration each SDL process is mapped into a distinct OS task. Independently of the integration method used, the same automatic generated C code is used. It is only the run-time libraries that differ in each case.

Light integration

Because all the SDL system is mapped to one OS task in light integration, scheduling and signal exchange between SDL processes are not handled by the OS kernel services but by an internal SDT scheduler. As for external messages that are sent from and to the different ASPIS modules such as the radio frequency interface or the keypad interface, these are treated by a special interface known as the "environment functions". This interface is to be written for the used RTOS and contains special functions that converts SDL signals to RTOS signals and vice-versa. Types representing signals, processes as well as a number of functions are available to support the user in writing the environment functions.

Although light integration is easy to achieve and facilitates by great deal portability, it does not take full advantage of the kernel services the RTOS offers. Moreover, no pre-emption is possible, meaning that the next pending process is allowed to execute only when the present process has executed its entire transition. Reasons for which tight integration method was adopted for the ASPIS software implementation.

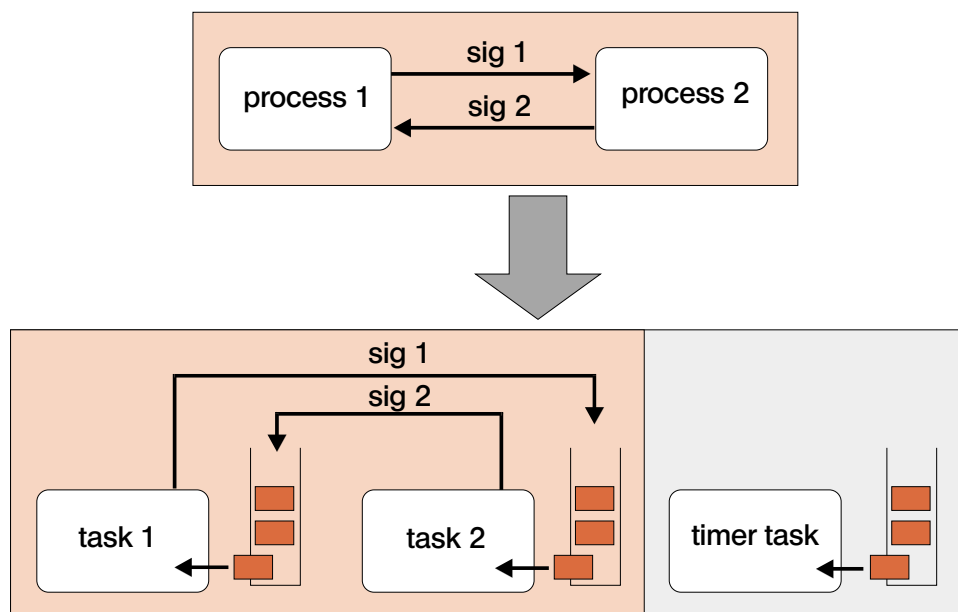


Figure 2. Tight integration model

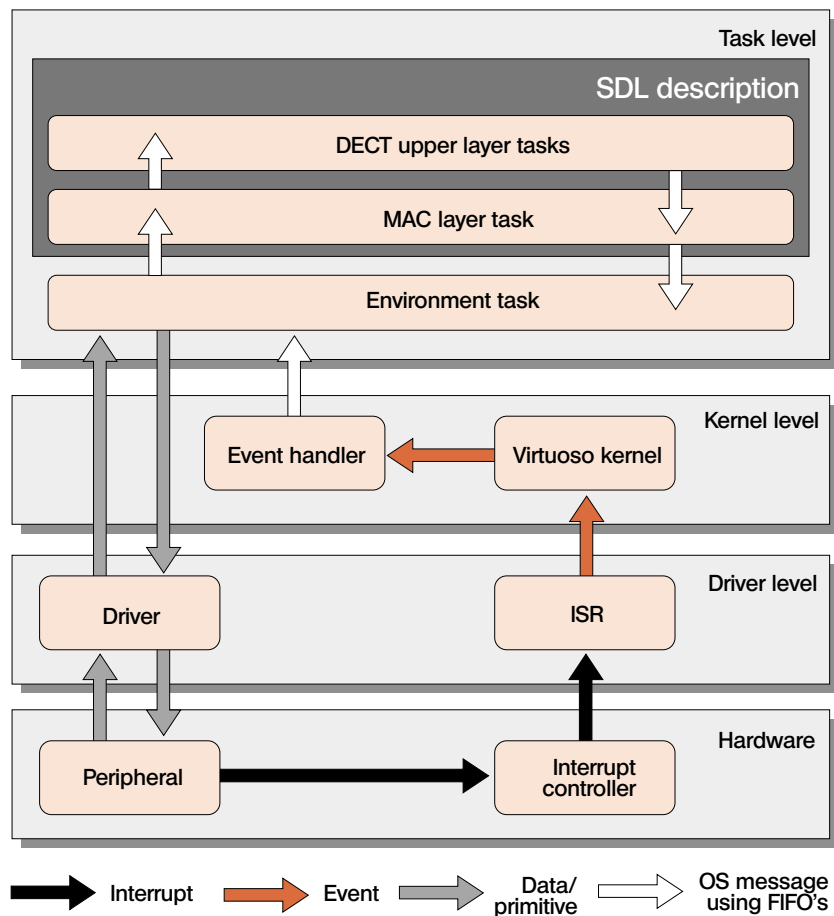


Figure 3. Handling Events with Virtuoso OS

Tight integration

The concept of tight integration is visualised in Figure 2. Each SDL process instance is mapped to one OS task, which is associated with an OS queue. Another possibility is to map the entire process instances to one OS task but this option was not adopted in ASPIS's DECT integration. Another property of tight integration is the use of timer task to handle all SDL timer operations. A number of support functions that are not OS dependent are provided to implement the full semantics of SDL timers. Such functions are used for resetting and setting a timer, check if the timer is in active state...etc.

Scheduling between OS tasks (SDL processes) is managed by the RTOS scheduler by means of OS task priorities or by any other scheduling method that the RTOS provides. This means that pre-emption can normally be used in tight integration.

Tight integration makes use directly of the RTOS kernel services to implement SDL basic concepts. For example, sending a message from one process to another involves OS system queues and calls to OS services for putting/getting the signal to/from a queue. Note that in some cases, because of the limitation of the OS, some SDL operations cannot be implemented. For instance, if the OS does not support dynamic

memory allocation, then a limitation should be imposed in the use of dynamic creation of SDL processes.

Although tight integration is difficult to achieve it make use of a limited number of OS function services. This is even true with the Windows win32 integration. In the case of Virtuoso 3.1, ten OS system function calls are used which are:

- **KS_Enqueue:** this OS is used to put an entry in a FIFO queue.
- **KS_DequeueW:** this OS service is used to read a data element from a FIFO. If the queue is empty, the calling task is put into a waiting list in order of its priority. If the queue is not empty, the oldest entry in the queue is removed and returned to the calling task.
- **KS_DequeueWT:** This primitive is identical to KS_DequeueW at the difference that the calling task is suspended until a timeout expires. This primitive is useful in the timer task operation implementation.
- **KS_Suspend:** This primitive causes the specified task to be placed into a suspended state.
- **KS_SetPrio:** This kernel service is used to change the priority of the specified task. A task switch will

occur if the calling task is no longer the highest priority runnable task. This service was used to avoid a task executing a state transition to be pre-empted by another high priority task.

- **KS_TaskPrio:** this OS service returns the calling task's current priority.
- **KS_Alloc and KS_Dealloc:** these primitives are used respectively to allocate and free a block of memory from a predefined memory map.
- **KS_LowTimer:** This OS service returns the current value in ticks of the kernel system clock.
- **KS_StartG:** this service is used to start a group of tasks.

C Advanced code generation from SDT tool was used in our particular case. The resulting code size and execution time was found to be satisfactory. The ASPIS memory requirements were in accordance with the initial specification deriving from DECT/GSM mobile devices needs.

HANDLING EVENTS IN VIRTUOSO OS

Figure 3 illustrates of the event handling that is used in ASPIS's DECT implementation. At the hardware level, every interrupt that is intercepted by the ARM microcontroller is serviced by the Interrupt Service Routine (ISR), at this level the interrupt is changed into an event signal to the Virtuoso kernel. The event is then handled by the adequate event handler function. Finally, the event is signalled to the SDL process (OS tasks) by outputting an SDL signal (eventually with parameters) to the lower MAC tasks in the case of DECT. Upon reception of the signal the lower MAC will trigger a series of state transitions that depend on the event and the received parameters.

Note the presence of the environment task, which acts as an interface between the SDL description of the DECT protocol and the drivers. It is in this task that SDL signals are created, these are then routed to the correct MAC layer task (SDL process).

MODELLING THE SYSTEM WITH ARMULATOR

The ARM Software Development Toolkit is used in the developing phase for the ARM processor. The toolkit contains all the essential tools to develop and debug applications based on the ARM processor. The brain of the toolkit is the ARM Debugger with its ARMulator extension. The ARMulator is a collection of programs that emulate the instruction sets and architecture of various ARM processors. It provides an environment for the development of ARM-targeted software on the supported workstation and PC host systems.

The ARMulator is an instruction-accurate software model: it models the instruction set without regard to the precise timing characteristics of the processor. Therefore, the ARMulator can be considered as a behavioural model of the ARM core, not a functional one. As a result, it is well suited to software development and benchmarking of ARM-targeted software, though its performance is slower than real hardware and depends on the host system. ARMulator also sup-

ports a full ANSI C library to allow complete C programs to run on the emulated system.

ARMulator is transparently connected to the ARM debugger to provide a hardware-independent ARM software development environment. Communication takes place across the Remote Debug Interface (RDI). You can supply models written in C or C++, which interface to the ARMulator's external interface.

An ARMulator environment consists of five parts:

- Remote Debug Interface: the interface between the ARMulator and its host debugger
- ARM Core Model: the model of the ARM processor itself (e.g. ARM6, ARM7TDMI, StrongARM)
- Memory Model: the model of the memory system outside the ARM, typically just RAM. ARMulator memory models are dynamic, allowing, for example, memory-mapped I/O simulation.
- Coprocessor Models: these model ARM coprocessors directly
- Operating System or Debug Monitor model: a virtual interface between the host and the modelled ARM.

The model of the ASPIS system is based on the Memory Model of the ARMulator. During the simulation of each external memory cycle, the ARMulator calls the class that implements the Memory Model to access the external memory area, simulating this way a memory cycle. The ability to simulate the system arises from the above fact. The developer can modify the Memory Model class and add classes that simulate the behaviour of the macrocells inside the ASPIS. This way, the original flat memory model, that it is initially supplied can be altered to match the memory-mapped space of the targeted system, while the later is still under development.

Therefore, the virtual software model of the ASPIS processor installs itself between the ARMulator and the Memory Model, gaining the control at the beginning of each cycle. The operation of the core of the ASPIS model is simple. It mainly has two capabilities, concerning the memory address of the cycle:

- Address of a memory-mapped register: the model forwards all the attributes of the cycle (address, data, size and operation) to the corresponding part of the model, while informing the others for the beginning of the new cycle. The model polls all the simulated macrocells to decide if a wait or error cycle has to be generated.
- Address other than the memory-mapped registers: the model first informs all the simulated macrocells for the beginning of a new cycle and the passes the control to Memory Model of the ARMulator.

Each time a macrocell gets the control for the core of the model, it checks all its parts to find out if it has to create an interrupt request to the ARM core, or if it has a memory operation to perform. In the first situation an interrupt exception is generated in the following cycle, while in the second a bus access request is made to the core of the model. In the beginning of the following cycle, the core of the model will notice the request for

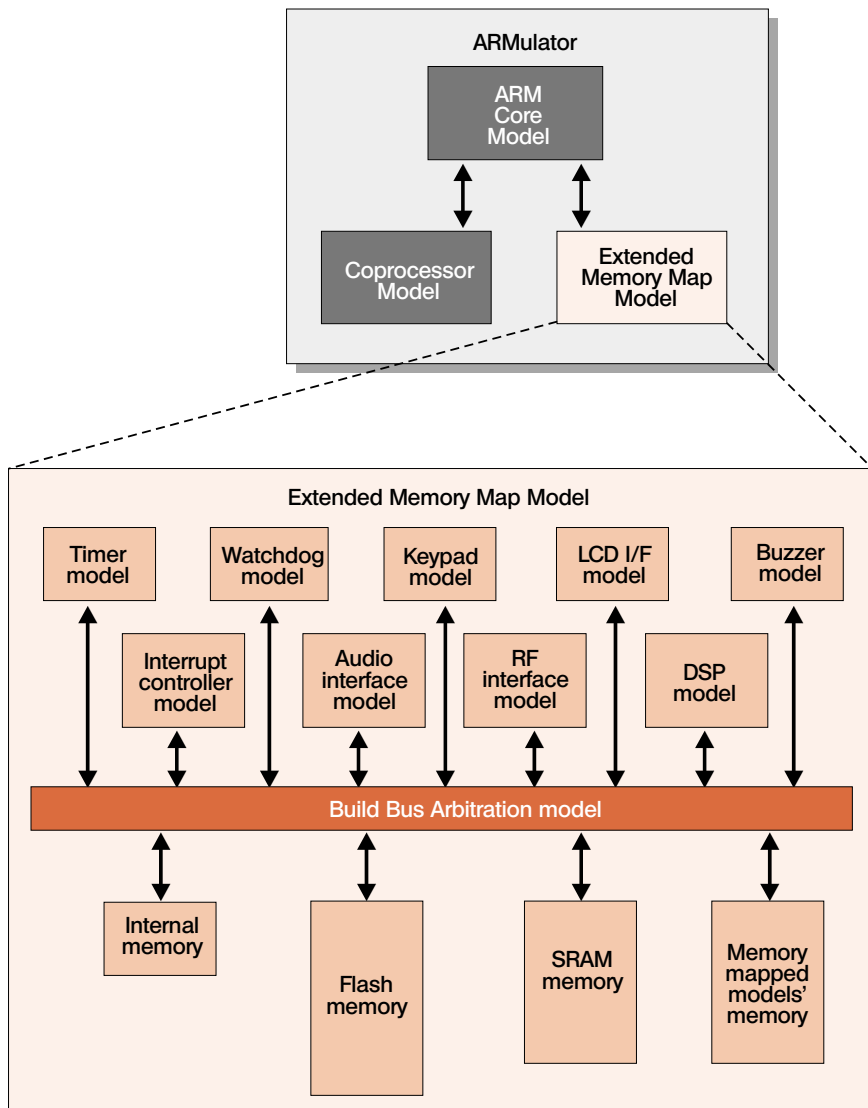


Figure 4. ASPIS model on the ARMulator

the bus and will delay the ARM, giving the model the needed simulation cycles to access the memory.

Each model of a macrocell contains the addresses of the memory-mapped registers of that macrocell and the code to simulate a set of its operation. As the function that implements the model is being called at the beginning of every cycle, the simulation of the whole ASPIS model is a clock-true representation of the real hardware.

The core of the ASPIS model simulates actually the logic of the internal bus controller, which corresponds to the Build Bus Arbiter macrocell. Except this macrocell, models for several other macrocells were developed. To support the operation of the RTOS the model of a timer was needed, while for the development of the software models for the RF I/F, DSP and the Audio I/F were essential. In order to simulate critical interrupt handling code, a model for the Interrupt Controller was developed. All the above software models were made according with the specifications that the hardware

engineers had to follow for building the ASPIS system. The result was the operation of the models to be as close as required to the behaviour of the original macrocells.

The structure of the ASPIS software model is graphically presented in Figure 4, along with its connection with the ARMulator debugging environment.

To ease the implementation and development of the DECT protocol stack an RTOS kernel was more than essential. As it was described earlier, the use of a real time operating system eases the development of the system by separating functions into tasks. Furthermore, if it supports pre-emptive priority driven multitasking it allows the program to utilise the full power of the ARM processor. Virtuoso offers all the above requirements and also it is really small and quick. Characteristics which are critical for an embedded system.

In order to port Virtuoso OS to ASPIS, special drivers for the Interrupt Controller, the Timers and the UART

macrocell had to be developed. This had to be done along with the hardware design phase and was boosted by the ARMulator model that included models for the above macrocells. On the other hand, DECT software development in a simulation environment was achieved with the use of a part of the model that implements the behaviour of the RF I/F macrocell. The used methodology for developing the drivers and the DECT software, using the software models of all the above-mentioned macrocells, proved to be efficient and easy to use.

CONCLUSION

Efficient and effective development of an embedded system requires parallel efforts in both of hardware and software parts in the very early stages. In order to facilitate the software development the use of adequate tools and methodologies are essential. As it has been shown, the use of SDL with automatic C code generation and tight integration methodology combined with the modeling capabilities of the ARM SDT created a framework that is far more practical than the use of a hardware prototype. It also proved to be faster, cheaper, more flexible and closer to the world in which the programmers are accustomed to work.

Our system design experience within the ASPIS project, showed that the whole development phase of the software could be achieved by using a virtual hardware prototype, such as the ARMulator based software model of the ASPIS. The development of low level primitives functions, RTOS integration with the system's architecture, debugging and testing of a communication protocol, such as DECT, were supported by the software model. More importantly, design errors due to misunderstanding of the specifications, or even invalid ones, can be easily detected and corrected during this phase.

The final test for this method of development was the real system. When the actual hardware prototype was ready, the transition of the software part to the hardware was just like a walk in the park. Running the DECT GAP standard procedure successfully was only a minor fraction of the whole development time. No major problems were discovered nor performance failures, only small and easy solved problems due to unmodeled parts of the ASPIS processor and its custom made board. ■

Christos Drosos is a Ph.D. candidate at the Electrical and Computer Engineering Department of University of Patras, Greece. He holds a diploma degree in Electrical and Computer Engineering from the same university in 1998. He is currently working as an embedded systems software engineer at Intracom S.A and has participated in many ESPRIT programs. His research interests includes Hardware/Software Codesign, Object Oriented Techniques, Real-Time Systems and System-On-A-Chip Designs. He can be reached by e-mail at cdro@intracom.gr

Michel Zayadine is a holder of an Electrical Engineering Degree (1992) and a Ph.D. degree (1996) awarded from the Swiss Federal Institute of

Technology of Lausanne. He participated in various projects such as the SWISSMETRO and TeleMan5 of the European program in the field of servomanipulation. He is employed at the Development Programs Department of Intracom S.A., Greece and his responsibilities include technical management and software development. His main research interests are in the field of RTOS, design languages and protocol implementation techniques. Contact him at mzay@intracom.gr

Dimitris Metafas obtained his Diploma degree and Ph.D. degree from Electrical and Computer Engineering Department of the University of Patras, Greece in July 1987 and February 1993, respectively. His expertise is in VLSI Design, VLSI CAD Tools, DSP Architectures, and Embedded Software technologies. From 1987 to 1994 he was a researcher in the VLSI Design Laboratory of the University of Patras where he carried out the development of a Digital Signal Processor for a variety of Real Time Applications, based on elementary arithmetic functions' evaluation algorithms. He had participated in several ESPRIT and RACE programs. From January 1995 on, he has been working with the Development Programs Department of Intracom S.A. His responsibility is the management of research and development projects in the area of embedded SW technologies for mobile communication systems. Contact him at dmeta@intracom.gr

Hans Thielemans is a mechanical engineer graduated at KULeuven Belgium (1979). He worked during 20 years at the mechanical department of the same university, during which he further specialized in robot and machine control and design. His main expertise and interest are in real-time control and operating systems. Back in 1990 he wrote the first version of the Virtuoso system. Recently (1999) he joined Eonic Systems to further work on the development of the Virtuoso kernel. During his stage at the university he was involved in several ESPRIT, TELEMAN and responsible for several ESA projects, mainly in the field of robot control and telemanipulation. At Eonic Systems he is responsible for R&D and European projects. He can be reached at Hans.Thielemans@eonic.com.