High-level architectural co-simulation using Esterel and C

Andre Chatelain, Yves Mathys,
Giovanni Placido
Motorola Inc.
{Andre.Chatelain,Yves.Mathys}@motorola.com
Giovanni.Placido@motorola.com

Alberto La Rosa,
Luciano Lavagno
Politecnico di Torino
{larosa,lavagno}@polito.it

ABSTRACT

This paper introduces an architectural simulation environment, aimed at defining the best SOC architecture for complex system-level applications. The application is modeled using an abstract Timing Modeling Language, that describes the requests (e.g., memory accesses, I/Os, etc.) that the application makes to the architecture. The abstract architecture is modeled at the cycle-accurate level using a mixture of Esterel (a synchronous language) and C. We discuss the results of the application of this tool to a GSM/GPRS application, including a dramatic speed-up of the architectural exploration phase.

1. INTRODUCTION

Today systems on chip (SoC) integrated circuits are very complex systems made of hundreds of millions of transistors. Application software running on such hardware architecture is composed of thousands of functions allocated to processing units and orchestrating the data movement and the chronology of the system. The increasing system complexity, driven by the semiconductor technology roadmap (Moore's law), is exacerbating the problems of validating the complete system and optimizing the system by playing architectural trade-offs.
One approach to solve the system validation problem is to introduce co-design tools [7, 1, 2] where the software and hardware are simulated together at the cycle accurate level.
In practice these techniques have several limitations (e.g., they need a complete hardware and software design), that restrict their impact on the optimization of the system.
One of the main limitation is that the capture of the entire system at the cycle accurate level is a very time consuming task, imposing very long debugging times, since neither the software nor the hardware are stable during the development phase.
Another limitation of the co-simulation approach is the long simulation time, which is exploding with the growing system complexity. Simulation performance of cycle-accurate HDL or ISS models is at most a few thousand instructions (clock cycles) per second on an engineering workstation.

System optimization must occur very early in the design cycle since late changes are quickly getting too expensive. The key requirements for architectural trade-off are:

- quick capture of the key system components, HW and SW;
- fast system simulation;
- flexible simulation platform for easy re-mapping of HW/SW functions;
- performance validation.

The approach that we propose in this paper focuses on the optimization of hardware architectures by early performance analysis through high level simulation of the complete system. We are presenting an architectural simulator, called *ArchAn* for Architecture Analyser, which allows one to define and validate efficient platforms for a class of applications.

ArchAn consists of a mix of cycle accurate models of the hardware architectural components (as described in section 3) and of highly abstract performance models of the software application.
The application is defined as a set of tasks allocated to architectural components (such as processing units, hardware accelerators), and scheduled by events (such as interrupts, DMA requests, RTOS calls). The tasks running on processing units are profiled on an ISS or manually estimated to capture their dynamic behavior.
We take particular care to accurately model the scheduling of the events on the overall platform.

The benefits of this approach are multi folds:

- systems are quickly captured and validated through simulation;
- performance analysis allows one to play architectural trade-offs early in the design cycle;
- the overall tasks synchronization and event management policies (priority level, preemption, tasks partitioning) are validated.

We present results obtained by applying our technique to a baseband IC for mobile phone applications (GSM/EFR/GPRS). The architecture performance is analyzed by simulating multiple TDMA frames. Dynamic observables are selected to monitor the architecture performance and identify bottle necks. Results give clear directions to improve the overall system performance, leading to cost effective solutions.
Moreover, the complete capture of the systems and the architectural analysis took less than two months compared to a previous twelve months effort with a coverification environment.

In section 2 we compare our approach with the state of art of co-design tools; in section 3 the ArchAn simulator is presented; in section 4 and 5 we discuss on the main characteristics of the TML language and of our implementation and finally, in section 6, results are reported for the GSM/GPRS application running on a multiprocessor platform. Some future work proposals are shown in section 7.

2. RELATED WORK

Architectural design starts with an exploration phase where resources (processors, buses, memories, dedicated hardware, I/O peripherals, RTOSs) are selected and system functionality is partitioned to hardware and software elements. System performance simulation then evaluates different implementation options. For real exploration, fast simulation is needed, and the architecture model must abstract behavior, communication and timing models. A separate specification of architecture and behavior will also provide reusable architectural models.

Some codesign tools (CoWare [7], Seamless [1] and Eagle [2]), provide a performance simulation capabilities only from a detailed description of the complete system implementation (hardware and software). This makes architectural exploration very difficult, as the models are too low-level to be re-used in a different architecture. Moreover, the simulation speed is slow, due to the use of an ISS for software timing, and to the cycle-accurate model for hardware timing.
Only high-level architectural modeling, verification and optimization at an early stage of system design can help solving system performance problems before implementation.

This is also the approach used in VCC [9], where architecture and behavior are separately described. Architecture specification is based on a diagram of components to which hardware and software behaviors are mapped; the behavior is described at the functional level and an annotated model of the application is required for system performance evaluation. Architectural components are pieces of C/C++ code that communicate via method calls and an event queue, thus a Discrete Event and not a cycle-based simulator is used (cycle-approximate level).
Performance simulation brings together behavior execution and delay model evaluation on the underlying architecture. Even if an event-driven approach provides a faster simulation, complex functional models with heavy interaction can slow down simulation speed even if simple architectures are considered.

For an early performance analysis of the hardware architecture, detailed functional verification of the application may not be required, since the main goal is to verify if the architecture is well suited to application. Only a delay model, representing requirements imposed by the application on the architecture (e.g., memory and bus accesses), is needed. This model is also faster to re-write and re-verify than a fully functional model of the system.

In our approach we directly describe requirements imposed by the software on the hardware by using the TML language (see Section 4), that is much more abstract than a functional model of the application. Moreover, TML eases software/hardware partitioning, since the actual behavior mapping on hardware and software elements depends on the TML description. Architecture is captured through cycle-accurate models written using a mix of the Esterel [4] and C languages.
In comparison to C/C++, Esterel [5] provides constructs for parallelism, synchronous signals and exception handling, that simplify the description of reactive modules, thus shortening development time of architecture models.
These constructs are also more powerful than those found in hardware-oriented C++ simulation libraries (such as SystemC, CynLib and Ocapi).

The overall hardware architecture is modeled as an Esterel module, thus providing a cycle-accurate simulator based on the highly optimized FSM implementation obtained by the Esterel compiler.
Simulation speed is not directly related to the complexity of system behavior (modeled in TML), and is affected only by the complexity of the considered architecture.

3. THE SIMULATOR

The architectural simulator, ArchAn, allows one to map an application to a target architecture, in order to perform software/hardware trade-offs and validate the overall system on chip. We define the application as a set of SW tasks assigned to processing units, triggered by HW/SW events.
In order to model the application, we defined a high level language Task Modeling Language (TML) which is interpreted by the processing unit (PU) delay models at run time (see the next section for more details). The TML can be generated either by profiling existing application code or by manually estimating new software tasks.
The abstract TML model has thus the following advantages over a detailed software model:

1. customer IP protection;
2. easy capture of application performance;
3. rapid trade-off analysis.

We model the architecture as a set of components such as processing units, interrupt controllers, busses, peripherals. All the architectural modules are described in Esterel and C, and are highly parametrized. A variety of architectural configurations can be built and explored, by taking advantage of the modularity of each component.

The interfaces of the modules are designed in such as way that one can build complex multi-processors, multi-bussing system by instantiating and connecting the different basic components. As components support a variety of parameters, they are then configured to tune their behavior within the architecture.
For example, the PU model can be set up as a DMA sharing a peripheral bus, or a DSP connected to memory and accelerators busses. The parameters are not only used for modifying the module behavior, but also to define its timing. Cascadable busses directly model their timing for read and write accesses.

Event management is modeled by the EMA (Event Management Architecture) component. Each PU is associated with an EMA for scheduling tasks over time. The EMA was designed for flexibility and supports a variety of scheduling schemes. It is worth noting that preemption in the EMA itself is easily modeled using the Esterel constructs (e.g. suspend).

Peripherals are mainly modeled as bus and interrupt traffic generators. The behavior of the peripherals is controlled via a TML instruction (NOTIFY) which is broadcast from a PU to all the components via a "virtual bus". Each NOTIFY instruction contains the symbolic name of the peripheral to which it is directed. Thus by adding or removing such instructions, and replacing

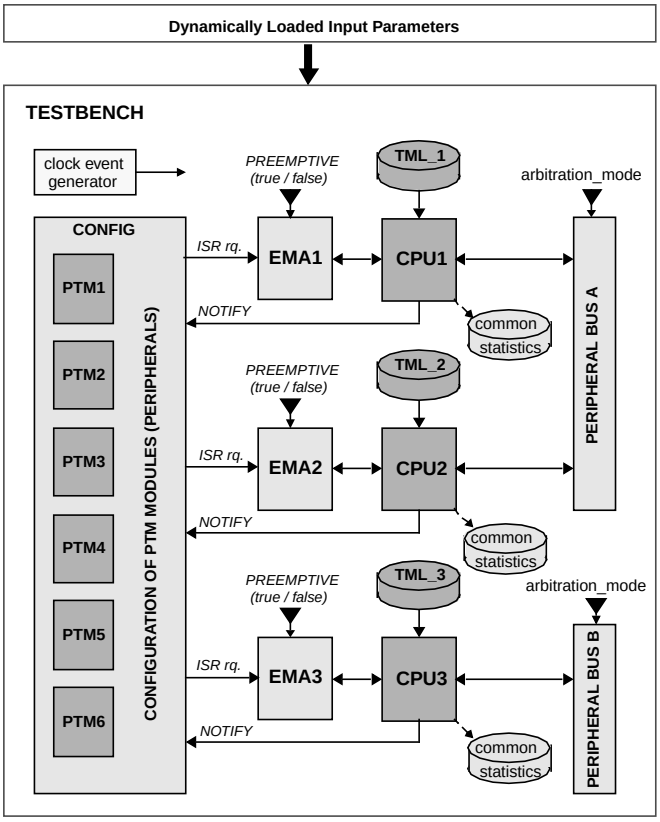

Figure 1: Structure of ArchAn for some architecture.

them with calls to other software tasks in TML, one can modify the architecture (some peripherals are no longer activated) and easily change the hardware/software partition without recompiling the architectural model.

The role of the bus model is to collect traffic statistic. As shown in figure 1, the bus model is not connected directly to peripherals but via signals. The data transfer latencies are computed by summing the delays associated along the paths between sources and destinations (considering busses and bridges).

In summary, the main architectural components are (figure 1):

1. PU module: a generic processing unit which interprets the associated TML file. The TML file contains the execution profiling of the software tasks. Software activities are modeled by the main five instructions: RD, WR, EX, NOTIFY and RQ, which are characterized by different delay values (clock cycles) when executed by the PU. Other parameters model the RTOS timing, such as context switch latency.

2. EMA module: PUs are associated with an EMA (Event Management Architecture) module that handles the different task requests based on priorities. Requests may be generated either by peripherals or by software task. The EMA module hence serves as a model for both the interrupt controller and the OS task scheduler.

3. PTM module: the Peripheral Timing Models generate events to PUs which activate ISR routines modeled as TML on the appropriate PU. These are high level functional models of architectural components, and mainly model the interrupt traffic generated by the I/O peripheral. The PTM behavior can be changed by the NOTIFY TML instruction.

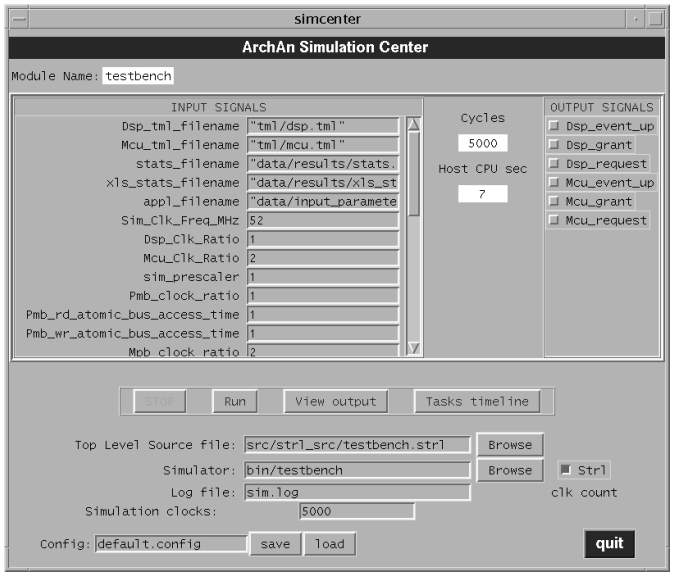

Figure 2: GUI for simulator configuration.

4. BUS module: bus model includes multi-master arbitration schemes and bus cascading features. Currently the simulator supports two arbitration schemes.

   (a) The fixed hardware priority arbitration scheme grants peripheral bus access to the current highest-priority master request. The arbitration priorities (H = high, M = middle, L = low) are defined before the starting of simulation.

   (b) The round robin priority scheme grants peripheral bus access in a circular fashion to master requests.

   A bus cascading mechanism allows us to model complex architectures containing bridges and multiple busses.

The tool provides a graphical interface, shown in figure 2, to assist the user during the simulation setup phase. The user can set several input parameters before running the simulation.
A first class of parameters regards PUs: parameters can be set for defining tasks and ISRs context switch timing. These parameters represent the basic latency due to the saving/restoring of the context registers. A simulation pre-scaler allows the user to simulate with larger granularity, in order to speed up simulation while loosing accuracy.
A second class of parameters allows the user to configure the bus by setting its frequency and the number of clocks cycles required for each read/write access.
Finally, parameters for the arbitration scheme are available. These parameters allow one to choose the arbitration scheme and the number of transfers granted from the bus for a single request of the PU.

All system activities during the simulation are recorded and elaborated to generate graphical and textual reports for:

- task timing statistics including processor load, task execution time, task latency, and

- peripheral bus statistics including bus load, arbitration latency, accesses statistics.

An example is presented on the figure 3, showing the graph of the task time line. You can see scheduled tasks with different priorities in different states (executed, pending, preempted).
The figure shows also the time spent by the PU in idle and the task queue length during the simulation (number of tasks that are running or pending).
In effect, the complete tasks dynamic is displayed to support debugging and tasks scheduling policies (tasks priority, interrupt/RTOS scheduling schemes).

4. TASK MODEL LANGUAGE

The TML (Task Model Language) language was designed for capturing the execution profile of tasks interpreted as a delay model by the PU model. The complete application, tasks and functions, are captured in TML and associated to a PU. The execution profile includes some explicit timing delay and implicit delay due to interruption with the architectural components.

The TML routines can be extracted from existing software by analyzing the worst case execution time on the target PU. If the software has not been written yet or is not accessible, estimation on the execution time can be done by hand or automatically [8, 3].

TML instructions are divided in five classes: READ, WRITE, EXECUTE, NOTIFY and REQUEST.
The delays associated to each READ and WRITE instruction represent the read and write delay when accessing system resources (i.e. peripherals, memories) through the busses. The bus loading statistics are computed by analyzing this class of instructions.
The EXECUTE delay represents the cycle count to execute the task of the function on a target PU. The NOTIFY instruction models the communication between PU and peripherals. It is used for dynamic control of the hardware modules. The delay associated to this instruction is equal to zero, and it has only behavioral effects.
Finally the REQUEST instruction allows to capture task to task communication on a single PU or across multiple PUs.

Besides the architectural set of TML instructions, the language also provides some control flow instructions for conditional execution (IF...THEN...ELSE) and loops (REPEAT).

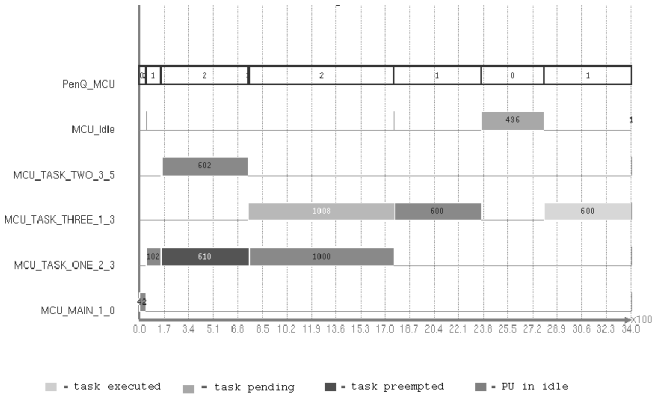

Figure 3: Graphical output of the simulation.

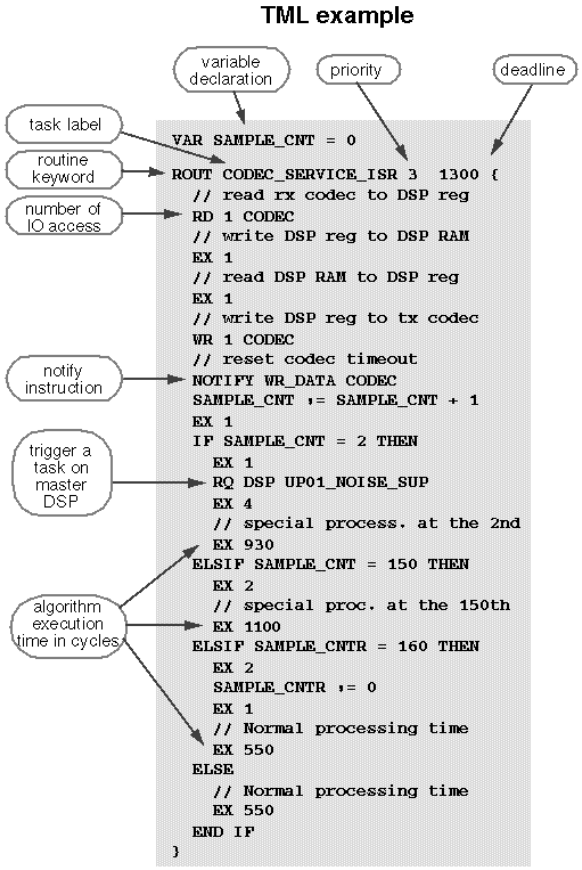

Figure 4: Example of TML code.

The figure 4 shows an example of ISR written in TML.
The routine serves an interrupt request by the CODEC (audio data transfers to the digital processor unit) module. After reading/writing to the CODEC and resetting the time-out value, the execution flow depends on the counter value.
Routines are characterized by PRIORITY and DEADLINE values. The RTOS scheduling algorithm permits preemption between tasks having different priorities. The DEADLINE parameter represents the number of PU clock cycles by which the routine must be completely executed. In case where the tasks misses its deadline, a flag is triggered during simulation which probably indicates that the application failed.

HW/SW partitioning are facilitated as modifications of the TML code do not require re-compilation of the source code and gives a short analysis cycle.

5. IMPLEMENTATION

The main goals of our implementation were

- ease of development of reusable architecture models;
- rapid architecture configuration in order to consider different architecture options;
- a simple language to describe hardware and software behavior and timing;
- fast simulation speed.

The first objective was achieved using the Esterel language: it provides a hierarchical description of the system by using modules that can be independently configured and instantiated multiple times.
Starting from this capability, after defining a clear interface to interconnect architectural modules (see figure 1), we started the development of a library of reusable models (PU, RTOS, BUS, peripherals).
Architecture reconfiguration requires only to writing an Esterel module where architecture module are configured, instantiated and properly interconnected (netlist).

The development of architecture models was simplified by Esterel's capability to interconnect modules using synchronous signals and to call C function within module's body.
As a first example, the bus arbitration scheme is based on checking for the presence of the REQUEST signal from any master on the bus. According to the arbitration scheme used, the arbiter will grant the bus to the winning master by sending the GRANT signal. As communication is synchronous, both check and grant happen in the same simulation cycle.
As a second example, we modeled the fact that tasks activation on an RTOS could be triggered by several sources (interrupts from peripherals or requests from other software tasks). We thus used a C function to insert the task in the specified EMA's scheduler queue. For each simulation cycle, the EMA module uses its scheduling policy (implemented as a C function) and signals to the underlying PU which task has to be executed.

When the simulator starts, each PU parses its associated TML file and creates a tree representation of each defined routine.
The PU module is able to scan the current task instruction tree, and will modify its timing behavior as specified by each architectural TML instruction (EX, RD/WR, NOTIFY, REQUEST).
Context switch is modeled by PU cycles elapsed before (context load) and after (context save) task's TML code is executed.
Bus access is modeled by an amount of simulation cycles elapsed by the granted PU that access the bus.
In order to consider bus cascading, we decompose bus transaction into a sequence of un-cascaded transactions. A user-defined routing table lists all possible communication path allowed in the architecture. Each path is characterized by read and write access delay to be added to the bus cycle duration.

As the TML file is interpreted at run time, it is possible to modify software behavior without recompiling the simulator, thus speeding up the design cycle.

By exploiting Esterel's parallelism and exception handling it is possible to easily specify abstract model of complex hardware. Hardware behavior could be modified by the interaction with PU elements or with other hardware modules. For example, considering a timer peripheral, we may need to start or stop the interrupt generation or modify its frequency.
While interaction with specific hardware is application dependent, we defined a protocol of communication between PU and peripherals that allows one to easily add or remove peripherals from a given bus. When a PU broadcasts a NOTIFY command each peripheral will check if it has to react to the issued command.

As the architecture simulator is a netlist of Esterel modules, the Esterel compiler implements it by flattening all the modules and translating them to a single FSM representation. The compiler will also check that a deterministic behavior is obtained, and will produce an optimized C code implementation of the FSM, as a fast cycle-based simulator.

The output of the simulation consists of traces with task activation, beginning of execution, preemption, resumption and end of execution. Bus transaction are monitored in the same way. All the simulation data are summarized in a statistics file, where PU and bus loads are reported together with minimum/mean/maximum execution time of each task and of bus transactions.
The graphical interface was implemented in Tcl/Tk.

6. RESULTS

The ArchAn simulator has been successfully used for two main projects.

We first completed a project aimed at defining the architecture of a next generation engine management micro-controller. Simulation of a full engine management application [6] helped Motorola to determine the best arbitration scheme used by 3 masters to access a single peripheral bus. For this study, up to seven arbitration schemes were compared. The main metrics were the bus access latencies and the bus loading.

The simulator was then used to assess and validate a baseband IC architecture for mobile phones running GSM/GPRS applications. The complex multi-master, multi-bus architecture was been modeled by instantiating existing components. The Esterel high level models of all peripherals, that generates interrupt requests to the PUs, were included in the main architecture.
Architectural performance was analyzed by simulating several TDMA frames, in order to monitor the architecture performance and identify bottlenecks.

From the statistic file generated by the simulation, we identified several key metrics that allowed us to compare the original architecture with several alternative solutions. The metrics used were:

- Average PU loading: corresponds to the average percentage of computing power during the simulation time;
- Minimum computation power available: for each PU task it corresponds to the minimum percentage of computing power between two activations of the task that is not used by another task at the same priority level. This dynamic measurement gives the amount of margin for each task;
- Maximum hold time: for each PU task it corresponds to the maximum percentage spent in hold state, i.e. context switch, preemption and pending state;
- Maximum number of preemption: corresponds to the maximum number of time each PU task has been preempted when active;
- Total number of preemption: corresponds to the total number of preemption during the simulation time;
- Peripheral bus loading: corresponds to the number of access per second to a peripheral bus performed by each PU.

Among the six metrics used it is important to note that all, except the first one, are dynamic information that can only be extracted using such performance simulation. The second

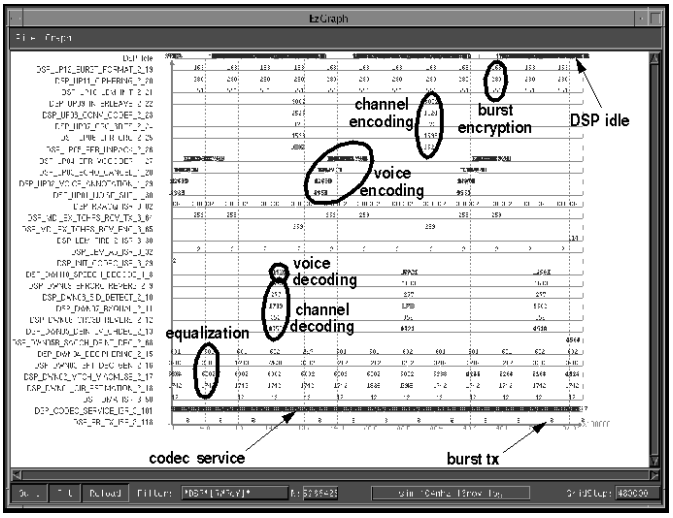

Figure 5: Graphical output simulation of the GSM/GPRS application.

metric showing the execution margins, allowed us to quickly identify architectural bottlenecks. Then, we performed several hardware/software trade-offs, e.g. by removing hardware accelerators and performing the same functions in software. The figure 4 gives an example of a TML code replacing the DMA handling the CODEC.

Several other alternative scenarios were obtained and this allowed us to optimize and leverage the performance/cost ratio of the baseband IC to its GSM/GPRS application.

Figure 5 shows the task time line results of the GSM/GPRS application. Twelve TDMA frames were simulated, the black circles put in evidence the uplink and downlink tasks execution.

7. FUTURE WORK

In order to explore complex memory architectures including caches, bridges and shared memories, we will consider bus transactions refined with memory addresses.
The model of the RTOS is currently limited to the scheduler. Other RTOS functions may be considered: tasks polling, IPC and semaphores will better model RTOS activity on the architecture and will simplify the models of complex software behavior.
Finally, architecture reconfiguration will be eased by a graphical tool, where the user interconnects library components and configures their parameters.

8. CONCLUSIONS

We have discussed a simulation environment that allows one to quickly play architectural trade-offs, based on an abstract but realistic model of the application. The architecture is modeled as a synchronous cycle-accurate high-level model (bit vectors are not needed at this level of abstraction). The application is modeled as a set of tasks interacting with each other and with the Processing Units and peripherals. By using this environment we were able to successfully model not less than five different variants of the same archictecture, by changing the hardware/software partition and using different DMA parameters, in less than two months. Modeling of the same architecture using a traditional co-verification environment took over one year, and did not allow any substantial exploration.

9. REFERENCES

[1] *Mentor Graphics Seamless CVE Home Page.* http://www.mentorg.com/seamless/.

[2] *Synopsys' Eagle Home Page.* http://www.synopsys.com.tw/products/hwsw/eagle_ds.html.

[3] D. Sciuto M. Vincenzi A. Balboni, W. Fornaciari. The use of a virtual instruction set for the software synthesis of hw/sw embedded systems. In *9th International Symposium on System Synthesis.*

[4] G. Berry, P. Couronné, and G. Gonthier. The synchronous approach to reactive and real-time systems. *IEEE Proceedings*, 79, Sept. 1991.

[5] G. Berry and G. Gonthier. The esterel synchronous programming language: design, semantics, implementation. *Science of Computer Programming*, 19, Sept. 1992.

[6] Y. Mathys and A. Chatelain. Using hw/sw modeling to optimize embedded control systems. *FDL*, 1999.

[7] K.V. Rompaey, D. Verkest, I. Bolsens, and H.D. Man. CoWare - A design environment for heterogeneous hardware/software systems. In *Proc. European Design Automation Conf.*, Sep. 1996.

[8] K. Suzuki and A. Sangiovanni-Vincentelli. Efficient software performance estimation methods for hardware/software codesign. In *Proc. Design Automation Conf.*, pages 605–610, Jun. 1996.

[9] Cadence Design System. *Cadence ships Cierto VCC environment for HW/SW co-design and reports customer success.* Press Release, Jan. 10,2000.