A Methodology for Architecture Exploration and Performance Analysis Using System Level Design Languages and Rapid Architecture Profiling

Alena Simalatsar and Roberto Passerone
Dipartimento di Ingegneria e Scienze dell'Informazione
University of Trento
email: {simalats, roby}@disi.unitn.it

Douglas Densmore
Department of Electrical Engineering and Computer Sciences
University of California, Berkeley
email: densmore@eecs.berkeley.edu

***Abstract*—**
The implementation of service-rich, highly interconnected applications and the increasing demand for performance, requires the development of highly optimized and flexible computing platforms. However, the tight real-time requirements of such systems, together with constraints on cost and physical size of the devices, results in increased design complexity and system heterogeneity. This creates a large design space. In this paper, we propose a structured approach based on system level specification languages that supports the rapid exploration and performance evaluation of computing platforms, including their middleware components, through simulation of abstract models. Accuracy is achieved through an off-line rapid architecture profiling procedure. We focus on a process network model, which is more suitable to the description of concurrent functions and data-dominated applications than a traditional sequential programming model. We describe the structure of our simulation framework, and use it to evaluate the performance of the lower layers of the UMTS protocol when mapped on Software Defined Radio oriented architectures.

I. Introduction

The construction of distributed communication infrastructures, and the use of highly connected embedded systems, makes it possible today to realize new and innovative applications and services. These systems are often context-aware and leverage the mobility afforded by wireless connectivity [1]. The implementation of such applications, and the increasing demand for high performance, requires the development of highly optimized computing platforms. However, the tight real-time requirements of such systems, together with constraints on cost and physical size of the devices, results in increased design complexity and system heterogeneity. This presents an overwhelmingly large design space for developers. Tools based on Register Transfer Level (RTL) simulation and evaluation boards are too detailed for an effective exploration of system design alternatives, and are typically biased toward specific implementation styles. In addition, the lack of abstraction makes the design, as well as the validation process, difficult if not impossible.

To overcome these problems, new methodologies have been developed for fast architectural exploration and optimization based on a stepwise refinement of the design specification. One example is the platform-based design (PBD) methodology [2], where platforms at higher levels abstract the details of lower level platforms, and can be used for fast performance estimation. This process is essential for quickly converging toward a platform that is not only optimized for the desired functionality, but can also support its future extensions. Tools for the automatic mapping and synthesis of optimized platforms are faced however with extreme complexity, due to the size of the solution space, and are typically confined to specific domains of applications. The alternative is manual architecture selection, coupled with fast performance simulation that computes metrics with quick turnaround time. Early attempts by the industry to introduce such technology [3] have not been successful in the market due to a variety of reasons, including the lack of appropriate performance models and the use of proprietary languages.

In this paper, we approach the problem by presenting an infrastructure based on system level specification languages and rapid architecture profiling using both existing and newly developed tools which support the PBD paradigm. Within PBD, we focus on the design abstraction that corresponds to the deployment of an application on a computing platform that may include general purpose processors and other programmable or reconfigurable components (i.e., FPGAs). Our contribution is a structured approach to the construction of functional as well as architectural models. These not only include performance metrics such as execution timing, but are also able to quickly and flexibly evaluate different scheduling policies and mapping strategies. This is achieved by separating the functional models from the architectural models, and by connecting them through control signals to regulate the overall execution. The architectural models of processors include the specification of middleware components, such as the scheduler, which may have substantial impact on the performance and the correctness of the implementation. Our solution is evaluated both qualitatively and quantitatively on the data link and physical layers of the UMTS protocol mapped onto an architecture oriented towards the implementation of Software Defined Radios. We show that we can explore interesting mappings quickly (in a matter of minutes) as well as measure

978-1-4244-1995-1/08/$25.00 ©2008 IEEE.

metrics such as throughput, latency, and utilization.

This paper is organized as follows: Related work is discussed in Section I-A. Section II gives an overview of our methodology. Section III discusses in detail the functional and architectural models used in our case study. Finally, results and conclusions are discussed in Sections IV and V.

A. Related Work

The literature on design space exploration is vast. We detail only the work that is close to our defined area of application.

1) Academic Approaches: Kempf et al. have presented similar ideas in the analysis of multi-processor SoC platforms [4]. In this work, the authors present a methodology based on SystemC that makes use of a Virtual Processing Unit to schedule a set of tasks, which are then simulated using a discrete event model. Our approach is similar, but is more focused on selecting an appropriate architecture, including other heterogeneous components such as FPGAs, and extends the methodology to a timed process network model, rather than discrete event. The way we handle preemption is also different, and based on the Result Oriented Modeling (ROM) technique proposed by Schirner and Dömer [5]. In ROM, efficiency of simulation is achieved by optimistically executing a transaction to completion and by deferring checks for preemption to its end. Then, if necessary, appropriate corrective actions are taken to adjust the timing of concurrent transactions. We apply this technique to processes running on processing elements instead of bus transactions. In addition, we explicitly distinguish between the processes and the scheduling policy, which are connected through special ports, to support an easy mapping of processes onto computing elements.

Optimistic execution requires that certain properties be satisfied by the model. For this, we employ a mixed dataflow and reactive model where the activation of the dataflow actors is controlled by a scheduler. This is inspired by the model used in FunState [6]. In its basic form, the FunState model includes functions that communicate over queues and arrays of registers. Their activation is controlled by a finite state machine (FSM), which determines the progress of the execution, so that tokens are consumed and produced at a time consistent with the execution latency of the functions. The main purpose of FunState is heterogeneous modeling and schedulability analysis, which is supported by symbolic formal verification techniques. However, its design as an internal representation makes FunState harder to use for architectural and design space exploration. In particular, it does not distinguish between function and architecture. We are mainly interested in the performance evaluation of possibly different configurations of the system. Thus, in our methodology, we use schedulers to represent an operating system, while the functions are assigned to different CPUs to model the mapping.

Some work has been already done in the area of abstract RTOS modeling in SystemC. In [7] Mahadevan et al. present ARTS framework aimed to maintain preemptive scheduling capabilities in SystemC. Le Moigne et al. particularly, describe an implementation of a generic RTOS model, supporting basic performance analysis for different scheduling policies [8]. They present two approaches to scheduling tasks on a single processor. The first uses a dedicated SystemC thread in order to simulate the behavior of an RTOS. The second avoids using the thread and embeds the RTOS procedures in the tasks. The second approach has advantages in terms of simulation performance, but makes it hard to explore different hardware architectures. Our RTOS model is therefore conceptually similar to the first approach.

Several frameworks have been developed for architectural exploration and performance evaluation. For instance, MILAN [9], which is built on top of GME [10], supports the integration of different simulators at various levels of granularity, and integrates the design space exploration tool DESERT [11]. At its core, DESERT allows the designer to express the flexibility in a platform by specifying structural constraints in OCL. An efficient symbolic technique is used to explore only those architectures that satisfy the constraints, thus pruning a large part of the design space. Performance evaluation is then carried out by integrating lower level simulators. Our architecture exploration paradigm differs substantially from that employed in DESERT. We employ an abstract description of an operating system, instead of structural constraints, and thus are able to relate the execution of a function with its implementation on an architecture without resorting to low level simulators, which are only used during a characterization phase. However, the combined use of structural and scheduling constraints for fast generation and exploration of architectures could be a promising avenue of future research. Similarly, the Metropolis framework [12] defines a general infrastructure and a rich metamodel for the design of heterogeneous systems that is capable of representing functions, generic performance metrics, and complex mapping using synchronizations constraints. Within Metropolis, Meyerowitz et al. focus on high level modeling of embedded micro-architectures retargetable for different instruction sets [13]. Their approach to architecture design space exploration is similar to ours and recent presentations suggest the use of this technique for Software Defined Radio [14]. We use their proposed annotation mechanism [15] as part of our flow for profile information on the ARM9 processor. However, we take a more pragmatic approach, and rely on an existing language (SystemC) on top of which we build a simple structure to distinguish between functional and architectural elements. While this limits expressiveness, our simple strategy overcomes certain efficiency and usability problems that are associated with the definition of quantities and scheduling policies in the Metropolis metamodel. Our schedulers can be seen as special cases of the Metropolis quantity managers.

Functional languages have also been used to support design space exploration. In ForSyDe [16], the system is initially specified as a deterministic network of synchronous processes, a model that facilitates the functional description by abstracting away detailed timing. The specification is then refined into an implementation by applying a series of network *transformations*, that may or may not preserve the semantics of the network. These transformations can, for example, partition the system into sub-domains that run at different speeds, corresponding to different implementations, thus providing feedback on the performance of the refined model. The transformation-based refinement in ForSyDe has clear advantages in terms of the ability to prove correctness and maintain consistency with the original specification. However, the distinction between functionality and architecture is lost, and a change of mapping may require substantial restructuring of the system. Our approach to mapping, instead, makes this task simpler, since only the mapping of functions to computing elements must be changed.

2) Industrial Approaches: An industrial tool for creating platform descriptions with mapping capabilities is VaST Systems Technology's Comet/Meteor [17]. Comet focuses on creating high performance processor and architecture models at the system level. It uses virtual processors, buses, and peripheral devices to create candidate architectures for design space exploration. These are called virtual system prototypes (VSP). VSP models are provided by VaST in the form of libraries or can be entered by the user in C/C++/SystemC. Meteor is an embedded software development environment. It also interacts with VSPs for cycle accurate simulation and parameter driven configuration. Meteor is the environment to develop software for the VSPs created by Comet. This process follows much more closely a typical design process for a microprocessor including optimizing code development than our approach. Code is developed for a specific VSP environment as opposed to capturing the pure functionality of an application.

Mirabilis Design's Visual Sim [18] product family supports a more formal approach by providing a wide variety of models of computation including discrete event, dynamic dataflow, synchronous dataflow, boolean dataflow, continuous time, and finite state machines. The design process in Visual Sim begins by constructing a model of the system using the parameterizable library provided. This model can be augmented as well with C, C++, Java, SystemC, Verilog, or VHDL blocks. The library blocks operate semantically using a wide variety of models of computation. The design is then partitioned into software, middleware, or hardware. Finally, the design is optimized by running simulations and adjusting parameters of the library elements. The underlying simulation kernel is Ptolemy [19]. This tool focuses very much on design space exploration through the manipulation of the library block parameters and unlike our approach begins with a unified design and refines it into its HW and SW components through a manual ad-hoc refinement process.

Cofluent's Systems Studio [20] also provides transaction level SystemC models which perform design space exploration. The functional description is a set of communicating processes executing concurrently. The platform model is a set of communicating processes and shared memories linked by shared communication nodes. The platform model has performance attributes associated with it as well. This approach is very similar our approach but is more focused on keeping within a particular model of computation to describe the functionality. In addition the HW components are generic and not indicative of any existing architectural services.

II. Methodology

Our particular implementation of the PBD methodology follows the steps presented in Figure 1.

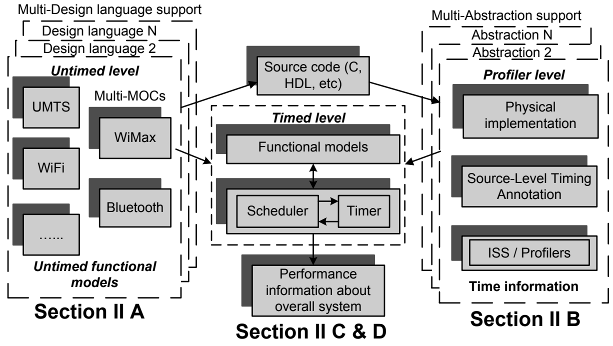

Fig. 1. Design Exploration Methodology

A. Functional Modeling

We first build a system level model of the functionality, with no notion of time (untimed level) [21], used to verify correctness and to study concurrency issues. This can be done in the model of computation which most suites the application. We then develop the functional model in a system level design language. SystemC [22], [23] was chosen for our case study over traditional sequential programming because it is a component model which natively supports concurrency, a computation paradigm that is more appropriate for today's reactive embedded systems. Our functional specification follows the process network model [24], where processes are concurrent and communicate over FIFO channels. A strength of our approach is the flexibility to specify models of computation and design languages.

B. Profiling

In order to create a SystemC performance model for a particular architecture we need to know the performance of each functional block executed on this architecture. To do so, we extract the code (C in this case) from our functional model (SystemC) and profile the execution on a set of processor emulators. We use two primary profiling flows for our work.

The first is used to get information for general purpose processors (in this case ARM9 and ARM7). This process involves the use of embedded profilers or instruction set simulators, in particular, Keil ARM Development Tool [25], as well as GNU compiler tools, virtual hardware, and a custom designed code annotator. Initially the code is cross compiled for the particular architecture target we are interested in. This executable is then fed to a virtual hardware simulator (in this case, simplescalar [26]). These results along with the original source code and corresponding binaries are fed to a code annotator which produces annotated code detailing the actual running time of individual segments of the original code for the given architecture target. This flow is shown in the top half of Figure 2.

The second flow involves libraries of elements which can be implemented on a platform FPGA (such as Xilinx's Virtex series). From the library, a set of systems are automatically generated by creating various legal permutations of the library components. One permutation may consist of a single soft processor core connected to a bus while another may have multiple processors and memory elements. Each of these individual systems is then run through the synthesis process provided by the FPGA. At this time, code which should be run on these systems is partitioned and bound to processing elements (such as the Microblaze soft processor). At the end of synthesis, the required execution cycles can be obtained for the application along with information about the cycle time from the physical design tools. Together, this can be used to calculate an overall execution time. This process is described in much more detail in [27].

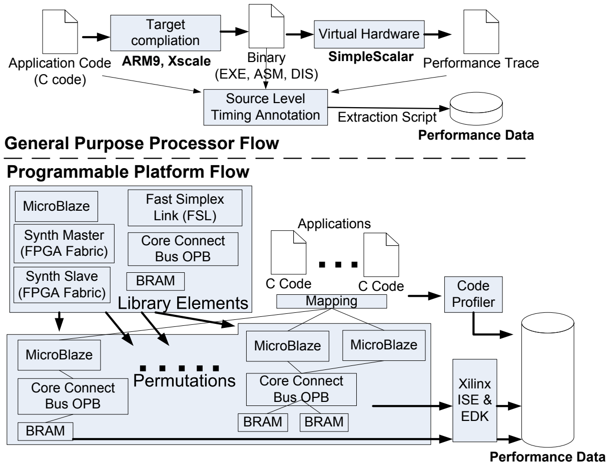

Fig. 2. General Purpose and Programmable Processor Profiling Flows

C. Architecture Modeling

For the model of the architecture, we exploit SystemC's ability to support different models of computation, by working at the timed level. Each architectural element is modeled as a resource manager. These are responsible for granting access to the resource according to a desired scheduling policy and for correctly accounting for timing (and, possibly, other performance metrics) by correctly sequencing the operations. For computational resources such as CPUs, the resource manager takes the form of a scheduler that implements a certain scheduling policy. The annotated profiling data is used to construct a performance model at the timed transaction level, which we use to determine the performance of the overall system, and to compute the utilization of the processors.

Our scheme has several advantages over the use of the profiler alone, which by itself can provide the system performance. First, processor emulators are slow and their performance depends on the underlying architecture. In contrast, SystemC is relatively fast and independent of the microcontroller architecture (see Section III-B). Secondly, SystemC is more flexible and makes it easier to partition the functionality onto different processor cores, and to combine their performance. This is essential as platforms evolve to include more processing elements.

A simple example of a system with two process networks that may have different priorities is shown in Figure 3. Each process network contains four processes interconnected by FIFOs. All processes in this example are mapped onto the same CPU. At the simulator level, mapping takes place by connecting the process to the scheduler via dedicated bidirectional communication channels, which are used to exchange control information. The scheduler is also connected to a timer similar to that proposed by Yoo et al. [28].

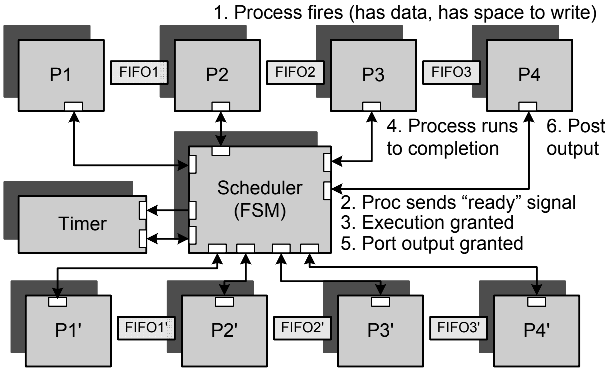

Fig. 3. Scheduling Mechanism

The scheduler is modeled as a finite state machine which controls the execution of the system. The activation of each process is controlled by the typical firing conditions of process networks, i.e., the availability of data at the input queues, and the availability of space at the output queues. These conditions are notified to the processes every time data is written to or read from the attached FIFOs. When a firing condition is satisfied, the process triggers the scheduler by sending a Ready_to_Run signal through the dedicated bidirectional channel and then waits for permission to start computation, which will be granted by the scheduler when the processor is available and when no higher priority process is ready to run. The process is run to completion, and stops before the results are written to the output FIFO. The computation is done in logically zero time. Instead, the scheduler will again trigger the process to post its outputs at the correct time, which will not only account for the process execution latency, but also for the time spent in running higher priority processes that had become active and preempted its execution. This way, following the ROM methodology, a process is never physically suspended as a result of preemption, thus reducing the overhead due to context switches. Instead, the scheduler verifies if any preemption has occurred, and, if so, updates the completion time by delaying it by the appropriate amount.

D. Preemptive scheduling

Several policies can be implemented by the scheduler. An example of a fixed-priority preemptive scheduler is shown in

Algorithm 1. The scheduler manages a priority list whose items are process descriptors which include the time, T_{init}, at which the process initiates its computation (negative if not scheduled yet) and a variable, τ, indicating the time left for the process to finish its computation.

Algorithm 1 Fixed Priority Preemptive Scheduler

```
 1: if ( new process new_P ) then
 2:     if ( current_P.priority $\leq$ new_P.priority ) then
 3:         current_P.$\tau$ $-=$ current time - current_P.$T_{init}$;
 4:     end if
 5:     Add_item( new_P );
 6: else if (timeout current_P) then
 7:     notify current_P post data;
 8:     list.pop( );
 9: end if
10: current_P = list.top( );
11: if ( current_P.$T_{init}$ $\leq 0$ ) then
12:     trigger current_P execution (notify event);
13: end if
14: current_P.$T_{init}$ = current time;
15: reset_timer( current_P, current_P.$\tau$ );
```

The procedure may be triggered either by a new process entering the enabled state or by a timeout from the timer that signals that a process has terminated execution and needs to post its data (lines 1 and 6). In the case of a new process, the algorithm first compares its priority with the one of the currently running process. If the new process has higher priority, then we update (decrease) the time to completion τ of the current process with the time it has been executed, which is the difference between the current time and the time it was last given the resource (line 3). In all cases, the new process is added to the list of processes (line 5). If a timeout occurs, it signals that the current process has reached the end of its computation. It is therefore removed from the list, and granted permission to post its data to the output (line 7).

The CPU is then given to the process at the top of the list (line 10). If the process starts execution for the first time, then its body is actually invoked (line 12). Time will not advance during its execution, since all timing is accounted for by the scheduler. To do so, the process descriptor is updated to record the starting time (line 14), and the timer is reset with the remaining time to completion for the process.

Other scheduling policies, such as round robin or EDF, can be implemented as well. By using a standard API for the scheduler process, these can be exchanged quickly to evaluate their impact on the mapped functionality and on the overall performance. A higher level resource manager using the technique described in [5] can then be used to mediate the data transfer between processor cores over a bus or other communication channels.

III. Case Study

We test our methodology on architectures for Software Defined Radios (SDR). SDR is a radio technology in which both modulation and demodulation are performed in software or using a programmable device [29]. The major advantages are flexibility and ease of adaptation, since the radio function can easily be changed to new standards. Programmability also promises economy of scale for manufacturers, who can rely on common platforms reused across different domains of applications. The requirements in terms of performance, latency, and cost make the design of these architectures difficult. We explored alternative implementations of part of the UMTS protocol [30] on programmable architectures.

A. Functional Model

In this paper we focus on the User Equipment Domain of the UMTS protocol [30], which is of great interest to mobile devices and is subject to stringent implementation constraints. The protocol stack of UMTS for the User Equipment Domain has been standardized by the 3rd Generation Partnership Project (3GPP) up to the Network layer, including the Physical (PHY) and Data Link (DLL) layers. Our model includes the implementation of the DLL layer and the functionality of the PHY layer. The DLL layer contains the Radio Link Control (RLC) and the Medium Access Control (MAC) sublayers and performs general packet forming. The RLC communicates with the MAC through different *logical* channels to distinguish between user data, signaling and control data. Depending on the required quality of service, the MAC layer maps the logical channels into a set of *transport* channels, which are then passed onto the Physical Layer. Finally, the Physical Layer handles lower level coding and modulation in order to reduce the bit error rate of the transmitted data.

The architecture of the protocol stack is very complex due to the high number of different logical and transport channels. In this work we focus on a subset of the functionality, described next, together with the architecture chosen for performance estimation.

1) Protocol Stack: The functional model of the UMTS protocol, shown in Figure 4, is composed of six DLL layer modules (actors of a dataflow process network) and of fourteen PHY layer modules for the transmitter, and of five DLL modules and twelve PHY modules for the receiver. Our functional model includes only a subset of the UMTS protocol stack and corresponds to the bidirectional Dedicated Channel that, in our case, is limited to point-to-point uplink user data transmission. Due to the complexity of the protocol stack we have made some initial assumptions in order to restrict our implementation and simplify the functional model. The RLC is divided into three separate entities for Transparent (Tr), Unacknowledged (UM) and Acknowledged (AM) transmission modes. We have limited our analysis to the Unacknowledged mode since this is a superset of the Transparent mode, and can be used to a certain degree to estimate the performance of the Acknowledged mode. Thus, the results of performance analysis of the UM allow us to make approximate estimations of the performance of the complete RLC layer.

Likewise, the MAC sublayer is divided into different entities that handle the mapping between the logical and the transport channels. Of these, we model the MAC-d entity which is the only one involved with the baseline (not enhanced) Dedicated

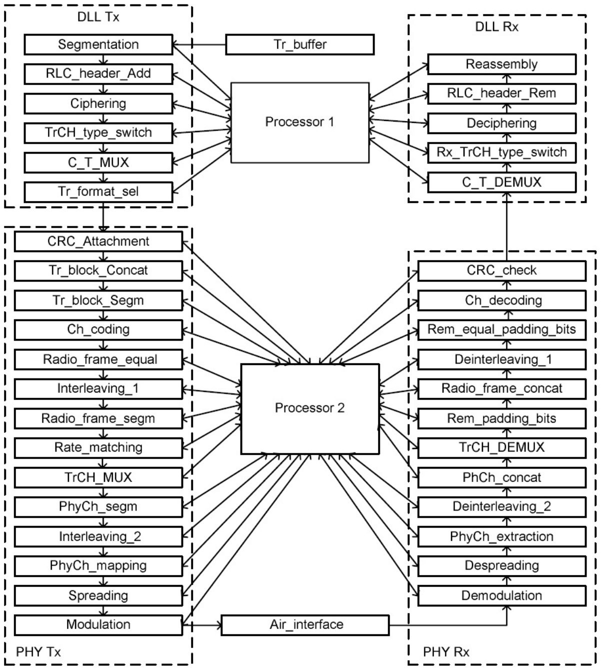

Fig. 4. Functional Model Mapped to Two Processors

Channel. The other blocks used for Dedicated Channel handling were introduced in more recent versions of the standard and are required for high speed and quality of service support.

The transmitter part of the PHY functional model is implemented upon the uplink model presented in [31] and is extended with two more blocks (Spreading and Modulation) described in [32] in order to complete the chain of digital operations. The receiver model is implemented as the chain of backward, to those of the transmitter, operations applied to the data coming from the Air_interface module. For the moment, the Ch_coding and Ch_decoding blocks include implementation of only Convolution coding and Viterby decoding algorithms respectively. This way we were able to study only the lowest data rate (12.2kbps). Other data rates require the implementation of the Turbo coding and decoding algorithms, which we are going to complete in the future.

Every module is attached to two FIFO queues, one for data input and the other for data output, connected to the next module. Currently, the same FIFO queues are used for inter-processor communication representation. Each block signals to the next the availability of a packet to be processed, by depositing it into the queue with blocking read and blocking write. The transmission buffer is a random data generator used to perform the simulation. The Reassembly module displays the final data as received. The Air_interface module is organized as a channel that adds some random distortion and time delays to the transmitted data.

As described in Section II-C, the modules are connected to each processor of the architecture via bidirectional channels. Each function uses only the channel dedicated to the processor used for its execution. This way, remapping a function to another processor can be achieved by simply switching to another dedicated channel. In Figure 4 we have shown a mapping example, which represents the execution of different protocol layers on separate processors.

B. Architectural Model

To design an optimal architecture we need to decide what elements should be available on the platform to achieve the best trade-off between the metrics of interest. In our design flow, these elements can include general-purpose processors, Digital Signal Processors (DSP), Field Programming Gate Arrays (FPGAs), or their mix. This step also includes identifying the kind of processors to be used (and their performance), as well as their number and general interconnection topology.

For this case study, we focused on the ARM7 (A7), ARM9 (A9), and Microblaze processors (μB). These are suited for embedded applications and work well with the profiling flow described earlier. In our case for profiling, our library elements consisted of the Microblaze processor core (6.00a) enabled with an FPU on Xilinx's Virtex II-Pro 2VP30. This was part of the ML310 development board. In addition it was connected to the On-Chip Peripheral Bus (OPB), enabled caching in 32MB of DDR SRAM, and used its iLMB and dLMB (instruction and data local memory buses) to access 112KB of BRAM. Its core frequency was 100MHz. The ARM7 TDMI-S is a small size, low power 32-bit RISC microcontroller with 128KB on-chip Flash ROM and 16KB of RAM. The ARM9 TDMI is a higher performance 32-bit processor. It has 16KB caches for both instructions and data. We run this both at 250 and 400 Mhz.

Procedure	**DLL**	**PHY**	Configuration	
		GPP Flow		
Create Static Exe	<2s	<3s	Envir.	Fedora 5
Create Debug Exe	<1s	<1s	Proc.	Xeon 3GHz
Create DisAsm File	<2s	<1s	Mem.	3.5GB
Simplescalar	<8s	8-80m[1]	s=sec	1.Dependent on
Annotation	<2m	1h-16h[1]	m=min	loops to amortize
SystemC Exe	<1m		h=hour	cache misses
		FPGA Flow, Xilinx 9.1i EDK		
Generate BStream	35m	Same	Envir.	Ubunutu 7.1
Update BStream	<3s	Same	Proc.	P4 2GHz
Download BStream	<1m	Same	Mem.	2GB

TABLE I
ARCHITECTURE PROFILING PROCESS AND COST

Table I details the time spent in the two profiling flows to get information for the architecture models. As is shown, profiling is not always fast. However, it should be noted that profiling various models is fully independent so that the time for profiling is only dictated by the most computationally complex model (not the set of models). Every model needs to be profiled only once for each architectural model. The profiling information is used after to simulate a combination of models in SystemC, which is performed (for 1000 packets) in less than a minute. It does not take a lot of time to configure (map) another functional model to another architecture configuration. We simply need to comment one line and uncomment another for each function that needs to be remapped. Again all this is

Mapping #	**1**	**2**	**3**	**4**	**5**
DLL **PHY**	**A9(4)** **A9(4)**	**A9(2)** **A9(2)**	**A9(4)** **μB**	**A9(2)** **μB**	**A7** **A9(4)**
Mapping #	**6**	**7**	**8**	**9**	**10**
DLL **PHY**	**A7** **A9(2)**	**A7** **μB**	**μB** **A9(4)**	**μB** **A9(2)**	**μB** **μB**

TABLE II
MAPPING CONFIGURATIONS

not very time consuming and allows a designer to explore a large design space.

Figure 5 provides a comparison of the performance (execution times in ns) for the DLL functions for the ARM9 processor at 250Mhz, the ARM7 processor, and the Microblaze processor. This is provided as a sample to show that while general trends in execution time can be seen, they are unpredictable and require a true profiling flow as opposed to crude estimates.

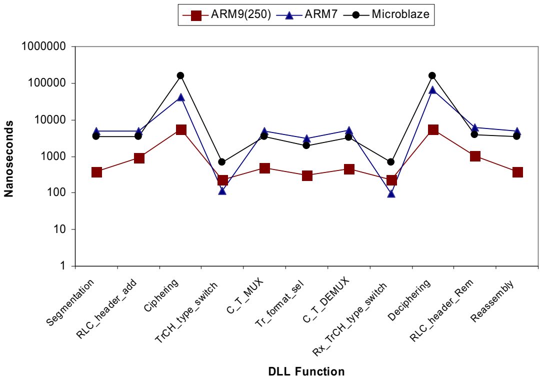

Fig. 5. Sample Execution Times Obtained Through Profiling

The architecture configurations (mappings) we explored are shown in Table II. Mapping of architecture models was done at the DLL/PHY level. Combinations of the Microblaze, ARM7, and ARM9 at 400 (A94) and 250Mhz (A92) were used. If one desired, mapping could be done within the sub-functional blocks of both the PHY and DLL models as well. This would greatly expand the size of the design space.

IV. EXPERIMENTAL RESULTS

This section presents the performance analysis for the Data Link (DLL) and the Physical (PHY) layers mapped to the architecture configurations presented in Table II. Three metrics were used to characterize the performance in our design space exploration. The first metric is the processing element load (utilization). The second and third are latency and throughput of the communication system, respectively. To calculate these parameters, one must know the time that each function takes to be executed on the particular processing element. We obtain the execution time of each task mapped onto different processing elements using the profiling methods presented in Section II and illustrated partially in Figure 5. These methods represent the lower level of abstraction (the profiler level), which we use to extract the relevant performance data to be used at the simulation's higher level.

Figure 6 illustrates the percentage load (utilization) of four analyzed processors with respect to the functionality mapped onto them under a fixed transfer rate used for speech transmission (12.2 kbps).

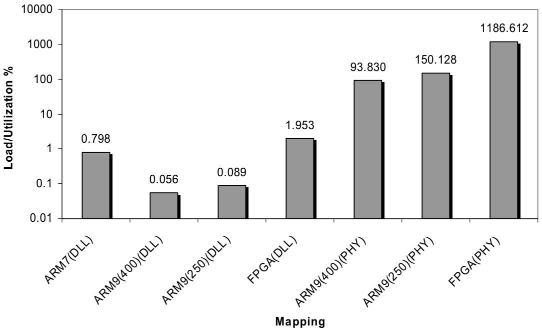

Fig. 6. Mapping Effect on System Utilization

This investigation examines 7 configurations. Two of them have the ARM9 mapped to the DLL (mappings 1-4). One uses the ARM7 for DLL (mappings 5-7). One uses the Microblaze for the DLL (mappings 8-10). The remaining three examine mapping the ARM9 and Microblaze to the PHY (the ARM7 was not mapped to the PHY). The results of the analysis show considerable variability across the processors. This analysis gives us a measure of the residual computing power available to the rest of the protocol, to potential other protocols running concurrently, and, at DLL layer, to higher level applications. This is essential information for the correct architecture choice and partitioning of the system. When the PHY functionality is mapped to the FPGA (mappings 3, 4, 7, 10) or to the ARM9 at 250Mhz (mappings 2, 6, 9) the load exceeds 100%, and is therefore invalid. Some boards for SDR development presently in the market, for example [33], use FPGAs to perform only modulation/demodulation computation (roughly a tenth part of the PHY). Our results show that this is reasonable and that full PHY functionality is not well suited to the current crop of soft processor FPGA cores due to their low frequencies and relatively simple, general purpose pipelines.

The architecture mappings we have studied are composed of two processors. One of them is used to run the functionality of the PHY layer exclusively. Because our case study includes the implementation of only one protocol stack, we consider that the right mapping combination is achieved when the PHY processor is loaded at almost 100%. The other processor is not only dedicated to the DLL layer, but also to the other higher protocol layers and applications, thus, it should not be loaded by the functionality of the DLL completely.

A second class of results are presented in Figure 7. They are devoted to the analysis of the latency and throughput of the analyzed mappings. From this graph we can see that the throughput adequate for the speech data transfer (12.2kbps) is supported only by mappings that have ARM9 (400Mhz) used to run the functionality of PHY (mappings 1, 5, 8). In these cases, the ARM9 dedicated to the PHY is loaded at almost 100%. The architectures with the ARM9 (250Mhz)

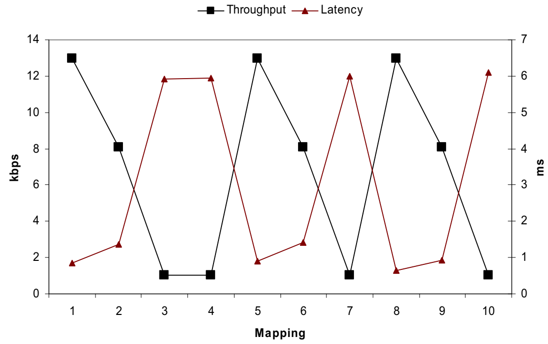

Fig. 7. Mapping Effect on System Performance

used to run the same functionality are slightly overloaded and do not give appropriate throughput. In this situation we can either change (increase) the clock frequency or change the mapping by transferring part of the functionality of the PHY onto another available processor. The load on the processor by DLL is negligible in comparison to PHY. That is why the throughput of the overall system is close to that which can be achieved by using only one processor and, therefore, is very low. Equal distribution of functionality between the processors may increase this value significantly, while the latency will not change much.

V. Conclusion

We have presented a methodology for the design space exploration of processor-based computing platforms. The methodology is based on the separation between the functional model, described in SystemC as a process network, and an architecture and performance model, described as a resource manager or scheduler. Preemption is accounted for by adjusting the timing of availability of output data, without actually suspending processes, which would adversely impact simulation performance. The performance data is obtained through low level profiling which can be done independently of the simulation process. We have presented an example of preemptive scheduler, and tested the methodology on a subset of the UMTS protocol to analyze the processor load and other parameters under different configurations. The analysis shows considerable variability in performance. This justifies the use of the PBD approach to design these kinds of systems, which must be fine tuned to support different standards without incurring a costly design space exploration process.

Our future work is focused towards facilitating design deployment by automatically generating the required connections between processes and schedulers, according to a given mapping, and extending the library of available schedulers.

Acknowledgment

This work was supported in part by ArsLogica, SpA.

References

[1] C. Andersson, *GPRS and 3G Wireless Applications.* John Wiley & Sons, April 2001.

[2] A. L. Sangiovanni-Vincentelli, "Defining platform-based design," *EEdesign*, February 2002.

[3] W. LaRue, S. Solden, and B. Bhattacharya, "Functional and performance modeling of concurrency in VCC," in *Concurrency and Hardware Design, Advances in Petri Nets*, ser. LNCS 2549, 2002, pp. 191–227.

[4] T. Kempf et al, "A modular simulation framework for spatial and tempral task mapping onto mutli-processor SoC platforms," in *DATE05*, Nice, France, April 16–20 2005.

[5] G. Schirner and R. Dömer, "Result-oriented modeling-a novel technique for fast and accurate TLM," *IEEE Trans. Computer-Aided Design*, vol. 26, no. 9, September 2007.

[6] K. Strehl, L. Thiele, M. Gries, D. Ziegenbein, R. Ernst, and J. Teich, "FunState – an internal design representation for codesign," *IEEE Trans. VLSI Syst.*, vol. 9, no. 4, August 2001.

[7] S. Mahadevan, K. Virk, and J. Madsen, "Arts: A systemc-based framework for multiprocessor systems-on-chip modelling," *Design Automation for Embedded Systems*, vol. 11, no. 4, pp. 285–311, 2007.

[8] R. L. Moigne, O. Pasquire, and J.-P. Calvez, "A generic RTOS model for real-time systems simulation with systemc," in *DATE06*, Paris, France, Feb. 16–20, 2004.

[9] A. Bakshi et al., "MILAN: A model based integrated simulation framework for design of embedded systems," in *LCTES 2001*, Snowbird, UT, June 2001.

[10] G. Karsai, J. Sztipanovits, A. Ledeczi, and T. Bapty, "Model-integrated development of embedded software," *Proceedings of the IEEE*, vol. 91, no. 1, January 2003.

[11] S. Neema, J. Sztipanovits, and G. Karsai, "Constraint-based design-space exploration and model synthesis," in *EMSOFT03*, Philadelphia, PA, October 13–15 2003.

[12] F. Balarin et al., "Metropolis: An integrated electronic system design environment," *Computer Magazine*, pp. 45–52, April 2003.

[13] T. C. Meyerowitz and A. L. Sangiovanni-Vincentelli, "High level CPU micro-architecture models using Kahn process networks," in *SRC TechCON*, Portland, Oregon, October 24–25 2005.

[14] T. Meyerowitz, R. Chen, A. L. Sangiovanni-Vincentelli, and J. Harnish, "Modeling a heterogeneous multiprocessor for software defined radio," in *CHESS review meeting*, Berkeley, CA, February 2006.

[15] T. Meyerowitz, A. Sangiovanni-Vincentelli, M. Sauermann, and D. Langen, "Source level timing annotation and simulation for a heterogeneous multiprocessor," in *DATE08*, Munich, Germany, March 10–14 2008.

[16] I. Sander and A. Jantsch, "System modeling and transformational design refinement in ForSyDe," *IEEE Trans. Computer-Aided Design*, vol. 23, no. 1, pp. 17–32, January 2004.

[17] VaST Systems, *Comet/Meteor*, http://www.vastsystems.com.

[18] Mirabilis Design, *Visual Sim*, http://www.mirabilisdesign.com.

[19] J. Buck and S. Ha and E. A. Lee and D. G. Messerschmitt, "Ptolemy: A framework for simulating and prototyping heterogeneous systems," *Readings in Hardware/Software Co-Design*, pp. 527–543, 2002.

[20] CoFluent Design, *CoFluent Studio*, http://www.cofluentdesign.com.

[21] A. Donlin, "Transaction level modeling: Flows and use models," in *CODES+ISSS '04*. New York, NY, USA: ACM Press, 2004, pp. 75–80.

[22] "SystemC," http://www.systemc.org.

[23] T. Grötker, S. Liao, G. Martin, and S. Swan, *System Design with SystemC.* Norwell, MA: Kluwer Academic Publishers, 2002.

[24] G. Kahn, "The semantics of a simple language for parallel programming," in *Proceedings of the IFIP Congress 74*, 1974, pp. 471–475.

[25] "Keil," http://www.keil.com.

[26] D. Burger and T. M. Austin, "The simplescalar tool set, version 2.0," *SIGARCH Comput. Archit. News*, vol. 25, no. 3, pp. 13–25, 1997.

[27] D. Densmore, A. Donlin, and A. L. Sangiovanni-Vincentelli, "FPGA architecture characterization for system level performance analysis," in *DATE06*, Munich, Germany, March 6–10, 2006.

[28] S.Yoo and A.A.Jerraya, "Hardware/software cosimulation from interface perspective," in *Computers and Digital Techniques, IEE Proceedings*, vol. 152, no. 3, May 2005, pp. 369–379.

[29] J. Mitola, "The software radio architecture," *IEEE Communication Magazine*, vol. 33, no. 5, pp. 26–38, May 1995.

[30] 3^{rd} Generation Partnership Project, "General universal mobile telecommunications system (UMTS) architecture," 3GPP, Technical Specification TS 23.101, December 2004.

[31] ——, "Multiplexing and channel coding (FDD)," 3GPP, Technical Specification TS 23.212, May 2007.

[32] ——, "Spreading and modulation (FDD)," 3GPP, Technical Specification TS 23.213, May 2007.

[33] "SFF SDR development platform," Lyrtech," Technical Specifications, February 2007.