UMTS MPSoC Design Evaluation Using a System Level Design Framework

Douglas Densmore[1], Alena Simalatsar[2], Abhijit Davare[3],
Roberto Passerone[2], Alberto Sangiovanni-Vincentelli[1]

[1] University of California, Berkeley
[2] University of Trento
[3] Intel Corporation

ABSTRACT

Rapid design space exploration with accurate models is necessary to improve designer productivity at the electronic system level. We describe how to use a new event-based design framework, Metro II, to carry out simulation and design space exploration of multi-core architectures. We illustrate the design methodology on a UMTS data link layer design case study with both a timed and untimed functional model as well as a complete set of MPSoC architectural services. We compare different architectures (including RTOSes) explored with Metro II and quantify the associated simulation overhead.

1. INTRODUCTION

In the race for higher performance computing, multi-processor platforms offer flexibility and a wide range of alternative design solutions that are able to optimally trade-off the design metrics of interest. This is especially true for embedded applications, often faced with hard to satisfy real-time and energy requirements which are best addressed by a distributed implementation. This trend is also apparent in the design of modern microprocessors, where the use of multi-threaded cores is favored over faster clocks to speed up the software execution. The design of multi-core architectures is, however, made complex by a large design space, the difficulty of integrating heterogeneous components, and time-to-market pressures. We have argued that only with a coherent and general design methodology can we address all these challenges [14]. The methodology should be applicable during all phases of the design process from specification to implementation, it should support the design chain across divisional and company boundaries, it should favor re-use at all levels of abstraction and should be based on rigorous semantics foundations. Platform-based design (PBD) was developed with these goals in mind.

A methodology can be applied even in the absence of supporting tools and flows. However, there is no question that the full leverage of the principles can be achieved only with appropriate design software. In this paper we discuss the use of a new event-based design framework, Metro II [4], for the simulation and design of multi-processor platforms and present a non-trivial UMTS case study to show the results that can be obtained from architecture exploration. In particular, we show that this approach may be used to carry out quick design space exploration with accurate, low-overhead simulation.

This paper is organized as follows: Section 2 discusses related work. Section 3 describes the Metro II design framework. Section 4 describes the architectural model design. Section 5 examines the UMTS case study and results obtained. Finally, Section 6 concludes the paper.

2. RELATED WORK

The Metropolis [3] design framework was the precursor of Metro II and it was developed for supporting the PBD methodology. This framework was the first system to leverage the concept of a semantic metamodel (abstract semantics) to manage the integration of heterogeneous components, to allow declarative and operational design entry, and to manage architectures and functionality in a unified way. The concept of separation of concerns, mapping of functionality to architectural components as a way of refining a design from specification to implementation, and communication as a first class citizen were all instrumental to build the framework. These concepts originated from work over years of research and development. The roots of Metropolis can be found in Polis [2] that was the first framework to be based on the separation of functionality and architecture. Polis was based on a single model of computation, Co-design Finite State Machines (CFSM), that captured locally synchronous, globally asynchronous designs typical of the automotive design space. Its first extension was a commercial tool developed by Cadence called VCC [9] based on the same modeling paradigm where the architectural modeling, called *architectural services*, and the simulation environment were taken to the next stage. In parallel, several research projects addressed similar issues.

Design space exploration via separation of concerns is also promoted by Spade [7], which is a Kahn process network (KPN) based workbench. It employs a Y-chart based approach (Dan Gajski was the first to represent the design process as a spiral moving from function to architecture) to design with functionality and architecture separated. In this case they are termed *workload* and *resources* respectively. It employs trace-driven simulation where time can be

978-3-9810801-5-5/DATE09 © 2009 EDAA

accounted for and performance data collected. DESERT [12] is a design space exploration tool which 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. In ForSyDe [13], 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. A few commercial offerings besides the original VCC work appeared that address the issue of design exploration using mapping of functional components to architectural ones (for example Coware ConvergenSC and Co-fluent Studio).

Some work has been done in the area of multi-processor SoC platforms analysis. Kempf et al. [6] have proposed a SystemC-based methodology that makes use of a Virtual Processing Unit to schedule a set of tasks, which are then simulated using a discrete event model. In [8] Mahadevan et al. have presented a generic SystemC-based simulation framework named ARTS. It allows the analysis of the system model with respect to different mappings of application tasks (represented as a direct acyclic graph) onto computing components and an operating system chosen for task scheduling. The ARTS framework requires precharacterization of the tasks mapped onto processing elements and employs the event-driven model as a basis for mapping technique. Le Moigne et al. describe a SystemC implementation of a generic RTOS model for real-time systems, supporting basic performance analysis for different scheduling policies [11]. 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.

3. DESIGN FRAMEWORK OVERVIEW

Metro II [4] is inspired by Metropolis since it is also based on abstract semantics and implements a PBD design methodology. However, it takes it a step further allowing designers to import designs that are developed using tools foreign to Metro II. Concurrency, synchronization, timing, and causality are all represented explicitly in Metro II. Further, the execution semantics is "purified" with respect to those of Metropolis by cleanly separating the functional execution of the components, their quantity (e.g., time and power) annotations, and the execution of the component assembly that satisfies a set of implicit and explicit constraints.

A *component* is the basic building block in Metro II. Components communicate through *ports* with compatible *interfaces*. Two *events* are associated with each method call on a port: a begin event and an end event. Imperative code within the component may control how events are executed, but separate declarative constraints over events can be used to influence execution as well. A key feature of Metro II is the ability to specify the functional and architectural models separately. The two are then *mapped* together to produce a system model with performance metrics. Mapping is realized by adding declarative constraints between events from the functional model and events from the architectural model.

Metro II has a three-phase execution semantics. Each process in Metro II has two states: *running* or *suspended*. Processes execute concurrently until an event is *proposed* on a required (output) port or until they are blocked on a provided (input) port. Once all processes are suspended, the simulation switches to the second phase of execution. In this phase, events are *annotated* by annotators, which represent the metrics of interest within the model. In this way, events and the methods they correspond to can be associated with cost. In the third and final phase, events are enabled according to schedulers and constraint solvers. These enabled events then become *inactive* again while simultaneously allowing their associated processes to resume to the *running* state. A collection of three completed phases is referred to as a *round*. Figure 1 illustrates the process states and the three phases in the execution semantics.

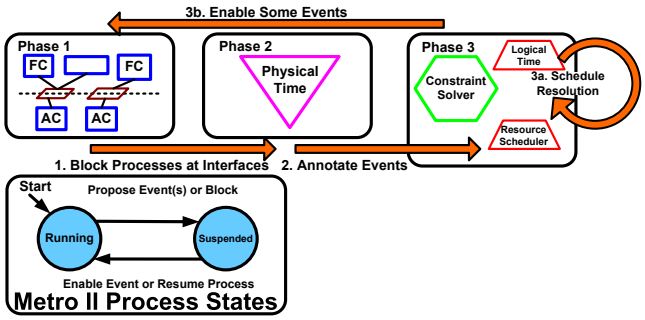

Figure 1: Metro II Three Phase Execution Semantics

The functional model covered in more detail in Section 5 is described as a process network or actor-oriented model, where concurrently executing processes communicate with each other through point-to-point channels. Metro II allows this communication to be specified either declaratively or imperatively. In the imperative model, the writer and reader query the FIFO via method calls - exposing more events to the framework and leading to more phase changes. In the declarative model, fewer phases changes take place, but the burden is instead placed on the constraint solver in phase 3. These tradeoffs will be examined more in Section 5.4.

4. MPSOC MODELING

Metro II supports the development of separate architecture service models to complement the functional modeling effort. In this work we create models with multiple processing elements which provide "costs" when mapped to a functional design. This section details the development of architectures used in the UMTS case study covered in Section 5.

4.1 General Architecture

Architectures in Metro II are *services* which map to the functional models and provide *costs*. They minimally need to:

1) Contain components which expose events to the functional model for mapping. These components (called *architectural tasks*) each have their own thread of execution. These threads will generate events associated with ports. These ports have events associated with their interface functions and correspond to functional model events. 2) Register the events associated with architectural tasks to the necessary annotators and schedulers. This association will provide the costs of the architectural services and ultimately the cost of the overall simulation.

The architecture models to be discussed in this work are composed of the following three portions: 1) Tasks - active components which serve as the mapping target for each component in the functional model. 2) Operating System - explicit, imperative mecha-

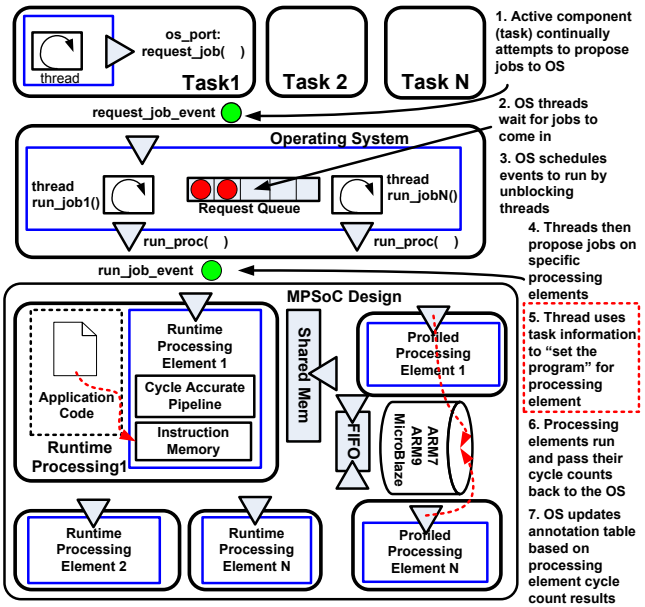

Figure 2: MPSoC Architecture Service Topology

nism for scheduling and assigning tasks to processing elements. 3) Processing Elements - workhorses of the architectural model. Used to model the core cost of a service. Figure 2 illustrates these three aspects.

4.2 Architectural Tasks

Tasks are lightweight, active components in the architecture model. The thread for each task constantly proposes events for its provided services. Mapping creates a rendezvous constraint between the event generated by the task thread and the functional event. Therefore, there is a 1:1 mapping between these tasks and functional components. Due to the rendezvous constraint, the task remains blocked until the corresponding event from the functional model is proposed. Step 1 in Figure 2 illustrates the task's role in architecture model execution.

4.3 Operating System

The operating system assigns tasks to processing elements (in a many-to-one relationship). It also carries out phase 1 scheduling - reducing the work to be done in phase 3. This is done by pruning the events proposed in the first phase. An investigation of scheduling policies is in Section 5.4. An OS is an active component with N threads (where N is the number of processing elements it controls). It maintains a queue of requested jobs which processing elements query to determine their execution status. The queue contains proposed events, processing element assignments, proposal order, and assigned priorities. Scheduling controls how events are added to and removed from this queue. Access to this queue is coordinated such that there are a limited number of outstanding requests for a given processing element. Steps 2-5 in Figure 2 illustrate the OS's role in the architecture model execution.

The OS is also used to access the annotation tables for events. The annotation tables are used by the annotator in phase 2. These tables relate event costs to architectural services. The OS updates the appropriate entry in the table after a request is completed and the true cost known. The OS may also add cost related to overhead (e.g. context switching). Tables are updated dynamically at runtime and do not require to be statically created with the netlist. Tasks themselves need know nothing about this process and only need to indicate which service they require.

4.4 Processing Elements

The third piece of the architecture platform are the actual processing elements. Steps 6-7 in Figure 2 illustrate two different types of processing elements that may be used and the interface to inform them which processing routine they should compute a cost for.

4.4.1 Runtime Processing

The first architectural modeling style is *runtime processing*. In this style, the processing elements are cycle accurate, microarchitectural models which execute code dynamically.

The OS provides information gathered from the task as to which operation is requested. The instruction memory of the processing element is loaded with pre-compiled code for the operation. The second step is then to execute that code at runtime which returns the cycle requirements for that code. The operating system will use that information to update the annotation table.

While this style may result in a slower simulation time as compared to the following approach, it simply requires that the code for the function be available. It requires no other modeling work by the user and is as accurate as the level of detail in the microarchitectural model.

4.4.2 Profiled Processing

The second style is *profiled processing* where precomputed performance metrics are stored for lookup. Again the OS will indicate which services are requested. The processing element can perform trivial table lookups or more complex calculations based on the current state of the processing element. Characterization methods for this approach have been shown in [10], [5], and [15]. An advantage of this approach are fast lookups as compared to the runtime processing approach. Drawbacks are that the modeling is often more limited in its usage, granular interaction between services is not fully captured, and characterization must be carried out prior to simulation. This precharacterization however only needs to be done once per computation routine.

5. UMTS CASE STUDY

It is possible to explore different types of applications (e.g. stream based or control applications) with MetroII. Due to the space limits we present only a UMTS Data Link layer case study, where we enumerate 48 different points in the explored design space.

5.1 Functional Model

We focus on the User Equipment Domain of the UMTS protocol [1], which is of 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 Unacknowledged mode of the DLL layer, which is composed of the functionality of the Radio Link Control (RLC) and Medium Access Control (MAC) sublayers. For the purposes of this case study, our model was largely separated into the RLC and MAC functionality as well as both receiver and transmitter portions. Simulation consists of processing 100 packets, each packet being 70 bytes. The functional model is represented as dataflow with blocking read and blocking write FIFOs and is shown in Figure 3.

The time annotation of the functional model is carried out by means of a scheduler. The scheduler is modeled as a finite state

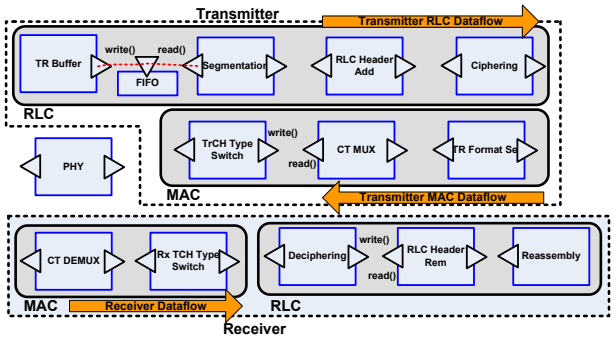

Figure 3: UMTS Metro II Functional Model

machine which controls the execution of the system. Each process is activated by the typical firing conditions of process networks, i.e., the availability of data at the input FIFO, and the availability of space at the output FIFO. When a firing condition is satisfied, the process triggers the scheduler by sending a "Ready_to_Run" signal through the dedicated bi-directional scheduling channel and then waits for permission to start computation, which will be granted by the scheduler when the resources are available and when no higher priority process is ready to run. In logically zero time the process runs to completion, and stops before the results are written to the output FIFO. 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. In this manner, 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.

5.2 Architecture Model

The architecture model assigns one task for each of 11 UMTS components (TR Buffer and PHY were not mapped as they represent the environment). The OS employs three different scheduling policies: round robin (RR), priority (PR) based on processing requirements, and first-come, first-serve (FCFS). Processing elements communicate through point to point FIFO links or through shared memory, whose arbitration effects influence processor utilization.

The runtime processing elements were fed C code reflecting the kernels of each UMTS component. The runtime processing element used was a cycle accurate datapath model of the Leon 3 Sparc processor. The pre-profiled processors use the same code but carry out offline characterization as detailed in [10] and [5]. The processors profiled were the ARM7, ARM9, and Xilinx's MicroBlaze.

5.3 Mapped System

Table 1 describes the 48 mappings investigated. These vary from 1 PE to 11 PEs. Partitions are broken down by Rx, Tx, RLC, and MAC functionality. Each is categorized into one of 9 separate *classes* based on the number of processing elements and the mix of pre-profiled and runtime processing elements. Mappings are further categorized as purely runtime processing (R), purely profiled processing (P) elements, or a mix (M).

5.4 Results

Results relating to design effort, processing time, framework simulation time, and event processing are analyzed. Five different

#	Type	Partition	#	Type	Partition
1	1: R	11 S	25	7: M	S (4), 9 (5), 7 (6), μ (7)
2	2: P	11 μ	26	7: M	S (4), 9 (5), μ (6), 7 (7)
3	2: P	11 7	27	7: M	μ (4), S (5), 7 (6), 9 (7)
4	2: P	11 9	28	7: M	μ (4), S (5), 9 (6), 7 (7)
5	3: R	4 S (1)	29	7: M	μ (4), 7 (5), S (6), 9 (7)
6	4: P	4 μ (1)	30	7: M	μ (4), 7 (5), 9 (6), S (7)
7	4: P	4 7 (1)	31	7: M	μ (4), 9 (5), 7 (6), S (7)
8	4: P	4 9 (1)	32	7: M	μ (4), 9 (5), S (6), 7 (7)
9	5: M	2 S (2), 2 μ (3)	33	7: M	7 (4), S (5), μ (6), 9 (7)
10	5: M	2 μ (2), 2 S (3)	34	7: M	7 (4), S (5), 9 (6), μ (7)
11	5: M	2 S (2), 2 7 (3)	35	7: M	7 (4), μ (5), S (6), 9 (7)
12	5: M	2 7 (2), 2 S (3)	36	7: M	7 (4), μ (5), 9 (6), S (7)
13	5: M	2 S (2), 2 9 (3)	37	7: M	7 (4), 9 (5), μ (6), S (7)
14	5: M	2 9 (2), 2 S (3)	38	7: M	7 (4), 9 (5), S (6), μ (7)
15	6: P	2 μ (2), 2 7 (3)	39	7: M	9 (4), S (5), μ (6), 7 (7)
16	6: P	2 7 (2), 2 μ (3)	40	7: M	9 (4), S (5), 7 (6), μ (7)
17	6: P	2 μ (2), 2 9 (3)	41	7: M	9 (4), μ (5), S (6), 7 (7)
18	6: P	2 9 (2), 2 μ (3)	42	7: M	9 (4), μ (5), 7 (6), S (7)
19	6: P	2 7 (2), 2 9 (3)	43	7: M	9 (4), 7 (5), μ (6), S (7)
20	6: P	2 9 (2), 2 7 (3)	44	7: M	9 (4), 7 (5), S (6), μ (7)
21	7: M	S (4), μ (5), 7 (6), 9 (7)	45	8: R	1 S
22	7: M	S (4), μ (5), A9 (6), 7 (7)	46	9: P	1 μ
23	7: M	S (4), 7 (5), μ (6), 9 (7)	47	9: P	1 7
24	7: M	S (4), 7 (5), 9 (6), μ (7)	48	9: P	1 9
(1)(Rx MAC, Tx MAC, Rx RLC, Tx RLC), (2)(Rx MAC, Rx RLC) (3)(Tx MAC, Tx RLC), (4)(Rx MAC), (5)(Rx RLC), (6)(Tx MAC) (7)(Tx RLC) (S = Sparc, μ = Microblaze, 7 = ARM7, 9 = ARM9)					

Table 1: Mapping Scenarios for UMTS Case Study

models were used: a timed SystemC [15] model, a timed Metro II model, an untimed Metro II UMTS model, a SystemC architectural model, and a Metro II architectural model. In specific configurations, Metro II constraints were used as opposed to explicit synchronization. The selection of constraints, functional model configuration, architectural model parameters, and mapping assignment is all achieved through small changes to the top level netlist. All results are gathered on a 1.8 GHz Pentium M laptop running Windows XP with 1GB of RAM.

5.4.1 Processing Time and Utilization

Figure 4 shows the UMTS estimated execution times (cycles) along with the average processing element utilization. Utilization is calculated as the percentage of simulation rounds in which architectural "proposed" events in the OS are "enabled". Low utilization indicates that a processing element cannot proceed and hence is unable to fulfill functional model requests. The x-axis (mapping #) is ordered by increasing execution times. The data is collected for each of the three scheduling algorithms.

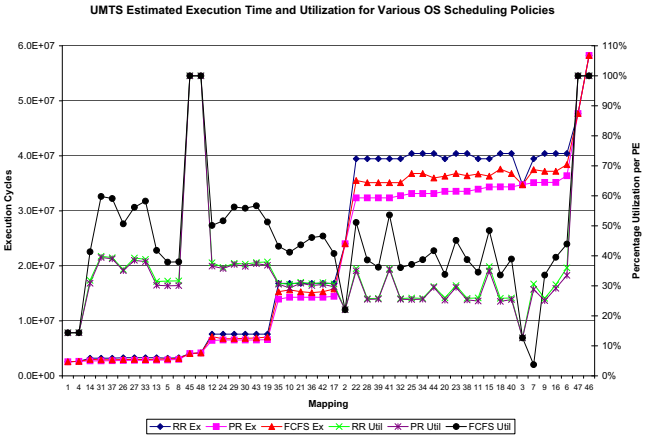

Figure 4: UMTS Estimated Execution Time vs. Utilization

For round robin scheduling, the lowest and highest execution times are obtained with mapping #1 (11 Sparcs) and mapping #46 (1 μBlaze) respectively. Mapping #1 is 2167% faster than mapping #46. This shows a large range in potential performances across

mappings. It is interesting to note that there are 23 different mappings which offer better performance than the 11 μBlaze or 11 ARM7 cores (mappings 2 and 3). This illustrates that inter-processor communication is a bottleneck for many designs and despite having more concurrency, those designs cannot keep pace with smaller, more heavily loaded mappings. Among all 4 processor systems, mapping #14 (two ARM9s used for Rx, two Sparcs used for Tx) and #31(Rx MAC on μBlaze, Rx RLC on ARM9, Tx MAC on ARM7, and Tx RLC on the Sparc) have the lowest execution times. Many of the execution times are similar and the graph shows that there are essentially 4 performance groupings.

The lowest utilization values for round robin occur in the 11 processor setups (average of 15%). The highest is 100% for all single processor setups. The max utilization before 100% is 39%. This gap points to ineffciency in the round-robin scheduler. It may be a goal of the other scheduling algorithms to close this gap. Also notice that for similar execution times, utilization can vary as much as 28% (e.g., in mappings #41 and #32).

The priority based scheduling keeps the same relative ordering amongst the execution times but reduces them on average by 13%. The highest is an 18% reduction (e.g., in #22) and the smallest reduction is 9% (e.g., in #8). The utilization numbers are actually reduced as well by an average of 2%. The largest reduction was 7% (e.g., in #6) and the smallest was 1% (e.g., in #31). As expected there was no change to the utilization or execution times for mappings involving either 11 processing elements (fully concurrent) or those with 1 element (no scheduling options). The utilization drop results from high priority, data dependent jobs running before low priority, data independent jobs.

FCFS scheduling also does not change the relative ordering of execution times but is not as successful at reducing them. The average reduction is only 7%. The maximum reduction is 11% (e.g., in mapping #24) and the minimum reduction is 4% (e.g., in #5). However, utilization is increased by 27%. The max increase was 45% (e.g., in #31) and the minimum improvement was 20% (e.g., in #5). FCFS increases utilization due to the fact that many jobs with lower priority often request processing in the same round as high priority jobs. While technically they are both "first", priority would negate this fact. FCFS's round robin tie breaking scheme helps smaller jobs in this case.

The analysis of execution and utilization for UMTS shows that high utilization is difficult to obtain due to the data dependencies in the application. Also, some of the partitions explored do not balance computation well amongst the different processing elements in the architecture. Many of the more coarse mappings only make this problem worse. A solution is to further refine the functional model to extract more concurrency. From an execution time standpoint, scheduling can improve the overall execution time but not enough to make a large majority of these mappings desirable for an actual implementation.

An accuracy comparison was performed with mappings #2, #6, and #46. These actual designs were created on the Xilinx ML310 development board where mappings #2 and #46 had only a 3.1% and 2% increase respectively in execution time. Mapping #46 inaccuracy is due to start up code and IO operations not captured by the model. #2 suffers from a slightly oversimplified point-to-point communication scheme in the model as compared to the Fast Simplex Links (dedicated point-to-point data streaming interfaces) used by the MicroBlazes. For mapping #6 (when scheduling affects the outcome) the increase was 16.2% (RR), 18% (PR), and 15%(FCFS), which implies the requirement of more refined OS model to capture the actual scheduling overhead. This comparison shows that Metro II simulation can closely (within 5%) reflect actual implementations and in the cases where the difference is greater, a tradeoff between modeling detail, simulation performance, and accuracy can be quickly analyzed.

5.4.2 Design Effort

The untimed Metro II UMTS functional model contains 11 processes while the architectural model may contain up to 26 processes. This is a large design spread across 85 files and 8,300 lines of code. The changing of a mapping is trivial however, requiring only changing a few macros and recompiling 2 files (2.3% of total; $<$20 sec). All 48 mappings can be done in less than 16 minutes.

The conversion of the SystemC timed functional model to an untimed Metro II functional model removes 1081 lines of code (related to scheduling and timing - both of which are in the architecture model). Metro II mapping removes much of the overhead associated with SystemC model synchronization.

Metro II constraints for the read/write semantics of a FIFO only require 60 lines of code which is 1.4% of the total code cost. The average difference of the entire conversion to Metro II was only 1% per file. More than half of these lines (58%) have to do with registering the constraints with the solvers.

The conversion of a SystemC runtime processing model (the Sparc processing element) to Metro II only requires 92 additional lines. This was a small 3.4% increase (2773 lines to 2681). This includes adding support for loading new code at runtime, returning the cost of operation to the netlist, and exposing events for mapping. This result is encouraging for importing code.

5.4.3 Framework Simulation Time

Figure 5 illustrates the percentage of the actual simulation runtime spent in each of Metro II's simulation phases for the 9 classes of mappings. The SystemC entry indicates the time spent in the SystemC simulation infrastructure upon which Metro II is built.

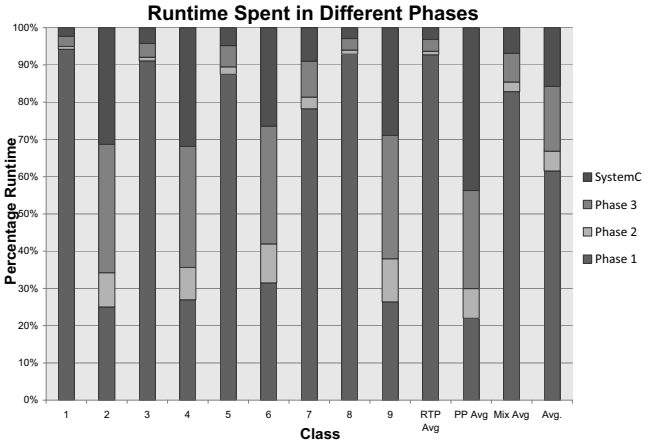

Figure 5: Metro II Phase Runtime Analysis

On average, 61% of the time is spent in Phase 1 (lowest section on the bar graph), 5% in Phase 2 (second section), and 17% in Phase 3 (third section). For models with only runtime processing elements (R) the averages are 93%, 0.9%, and 3% respectively. This indicates that in runtime processing, the Metro II activities of annotation and scheduling are negligible in the runtime picture. For pure profiled (P) mappings they are 21%, 7% and 26%. In this case one can see that Metro II now accounts for a greater percentage of runtime (phase 1 alone is representative of other simulation environments). For mixed classes the numbers are 82%, 2.6% and 7.6%. Again the runtime processing elements dominate. It should

be noted that while Ps have higher averages, the average runtime to process 7,000 bytes of data was 54 seconds. Phase 1 runtime (and SystemC overhead) are the main contributors to overall runtime.

If we consider the SystemC timed functional model, Metro II timed functional model, and the Metro II untimed functional model mapped to an architecture, the Metro II timed functional model had an average increase of 7.4% in runtime for the 9 classes while the mapped version had a 54.8% reduction. This reduction is due to the fact that Metro II phases 2 and 3 have significantly less overhead than the timer-and-scheduler based system required by the SystemC timed functional model.

Class	Event/Ph.	Comp. %	Comm. %	Coord. %	Avg Wait
1	0.091	0.083	0.083	0.833	3839.240
2	0.091	0.083	0.083	0.833	3839.240
3	0.169	0.125	0.042	0.833	6276.190
4	0.169	0.125	0.042	0.833	6276.190
5	0.131	0.170	0.114	0.716	5117.003
6	0.169	0.170	0.114	0.716	6276.190
7	0.150	0.101	0.088	0.811	5691.130
8	0.176	0.319	0.043	0.638	6718.550
9	0.176	0.319	0.043	0.638	6718.550
Avg	0.147	0.166	0.072	0.761	5639.143

Table 2: Metro II Phase Event Analysis

5.4.4 Framework Event Analysis

Table 2 shows the average number of event state changes per phase and the average number of phases an event waits.

On average, only 0.14 events are annotated or scheduled per round. Because of the architectural model integration with the UMTS functional model there are a limited number of synchronization points (which satisfy a rendezvous constraint and hence an event state change). As shown in Figure 5, Phases 2 and 3 do not account for a large portion of the runtime so while the event state change activity is low, it does not translate to increased runtime. Runtime is not increased directly by changing an event's state but rather by the total number of events in phases 2 and 3.

Events in classes 1 and 2 on average wait 42% less than the worst case. These classes are precisely those which provide maximum concurrency (11 processing elements). The worst is in classes 8 and 9 (single processing elements). As one would expect, when the scheduling overhead is lower and more processing elements are available, events wait much less for resource availability.

Finally it should be noted that runtime processing vs. pre-profiled processing does not impact event proposal or event state change. Comparing classes 1 with 2 or 3 with 4 confirms this. This contrasts heavily with the runtime of the simulation (in which PE type is a key factor). The runtime processing in the microarchitectural model is treated as a black box by Metro II such that the internal events are unseen and do not trigger phase changes. This indicates that SystemC components can be imported quite easily into Metro II without affecting the three-phase execution semantics.

The 3rd, 4th, and 5th columns of Table 2 categorize events in phase 1. Computational events request processing element services directly. Communication events transfer data between FIFOs and coordination events maintain correct simulation semantics and operation. The table indicates that events in the system are heavily related to coordination. Classes 8 and 9 have the lowest percentage of coordination events (64%) since these are 1 PE systems.

6. CONCLUSIONS

We illustrated how an event-based design framework, Metro II, may be used to carry out architectural modeling and design space exploration. Experimental results show that Metro II is capable of capturing functional modeling, architectural modeling, and mapping for a UMTS case study with limited overhead as compared with a baseline SystemC model. We showed that the design effort involved in carrying out 48 separate mappings with a variety of architectural models is minimal. Within the framework, we detail the runtime spent in the three different Metro II execution phases and provide an idea of how events move throughout the system.

Future work involves identifying and removing events not relevant for annotation or scheduling from Metro II's second and third phases, support for a wider variety of declarative constraints, and the analysis of other case studies such as h.264.

7. REFERENCES

[1] 3^{rd} Generation Partnership Project. General universal mobile telecommunications system (UMTS) architecture. Technical Specification TS 23.101, 3GPP, December 2004.

[2] F. Balarin, M. Chiodo, P. Giusto, H. Hsieh, A. Jurecska, L. Lavagno, C. Passerone, A. Sangiovanni-Vincentelli, E. Sentovich, K. Suzuki, and B. Tabbara. *Hardware-software co-design of embedded systems: the POLIS approach*. Kluwer Academic Publishers, Norwell, MA, USA, 1997.

[3] F. Balarin, Y. Watanabe, H. Hsieh, L. Lavagno, C. Passerone, and A. Sangiovanni-Vincentelli. Metropolis: An integrated electronic system design environment. *Computer*, 36(4):45–52, 2003.

[4] A. Davare, D. Densmore, T. Meyerowitz, A. Pinto, A. Sangiovanni-Vincentelli, G. Yang, H. Zeng, and Q. Zhu. A next-generation design framework for platform-based design. In *Conference on Using Hardware Design and Verification Languages (DVCon)*, February 2007.

[5] 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.

[6] 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.

[7] P. Lieverse, P. van der Wolf, and E. Deprettere. A trace transformation technique for communication refinement. In *CODES '01: Proceedings of the Ninth International Symposium on Hardware/Software Codesign*, pages 134–139, 2001.

[8] S. Mahadevan, K. Virk, and J. Madsen. Arts: A systemc-based framework for multiprocessor systems-on-chip modelling. *Design Automation for Embedded Systems*, 11(4):285–311, 2007.

[9] G. Martin and B. Salefski. Methodology and technology for design of communications and multimedia products via system-level IP integration. In *Proceedings of the conference on Design, automation and test in Europe*. IEEE Computer Society, 1998.

[10] 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.

[11] 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.

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

[13] I. Sander and A. Jantsch. System modeling and transformational design refinement in ForSyDe. *IEEE Trans. Computer-Aided Design*, 23(1):17–32, January 2004.

[14] A. Sangiovanni-Vincentelli. Quo vadis, SLD? reasoning about the trends and challenges of system level design. *Proceedings of the IEEE*, 95(3):467–506, March 2007.

[15] A. Simalatsar, D. Densmore, and R. Passerone. A methodology for architecture exploration and performance analysis using system level design languages and rapid architecture profiling. In *Third International IEEE Symposium on Industrial Embedded Systems (SIES)*, La Grande Motte, France, June 11–13, 2008.