Framed Complexity Analysis in SystemC for Multi-level Design Space Exploration

Armin Wellig and Julien Zory
Advanced System Technology – STMicroelectronics, Geneva, Switzerland
{armin.wellig, julien.zory}@st.com

Abstract

To guarantee efficient system-on-a-chip (SoC) solutions, design space exploration at various abstraction levels is needed. In this paper, we describe a simulation framework, which is particularly suited for multi-level exploration. Identifying the capturing of design metrics and their incorporation into an exploration framework as the main tasks to be resolved, we will propose a flexible monitoring tool implemented in SystemC to tackle the first item. We present a unified approach to capture and record behavioral, storage and communication characteristics at several abstraction levels of a typical SoC design flow. Figures such as increase in simulation time are included to characterize the developed tool. To address the design decision-making criteria, relevant design metrics are defined for each abstraction level. We will distinguish among four distinct levels modeling algorithm and architecture transitions. Finally, iteration control aspects of Turbo decoders used in wireless systems serve as a case study.

1. Introduction

There are several challenges for today's technologies enabling system-on-chip (SoC) development. Key design objectives to guarantee successful SoC solutions are rapid testing, integration and verification of HW/SW components; efficient IP reuse at various abstraction levels; and design space exploration (DSE) of various algorithms, architecture templates and implementation technologies. One way to tackle these challenges is the introduction of a unique modeling language, which allows a seamless flow from algorithm development over architecture investigation down to synthesis. A promising language is referred to as SystemC, which is a new open source library in C++ developed by a consortium of silicon vendors, CAD tool providers and universities [1]. The designer can add refinements to his behavioral, communication and storage modules starting at a high level of abstraction (e.g., capturing functional data flow characteristics of a specific algorithm) down to the cycle-accurate register-transaction-level (RTL) models. This homogeneous simulation environment eases testing of inter-level modules and simplifies HW/SW co-verification. It also allows easy integration of new and existing IP modules implemented at various abstraction levels [2]. Another major advantage is an early integration of Firmware into the functional validation flow. Due to tight time-to-market constraint and the need to support multiple services in future wireless standards, early integration and verification of Firmware will become an important competitive asset for the design of efficient solutions. In this paper, our main focus is to introduce a simulation framework, which is particularly suited for multi-level DSE in a SystemC design flow context already well adjusted to solve many other SoC challenges [3].

The multi-level (and multi-dimensional) design space offers a large range of possible trade-offs and decisions made at each refinement level are correlated with previous choices. An integrated and systematic DSE across the algorithm, architecture and circuit level is then essential to guarantee efficient implementations. In our case study, we show how statistical information of behavior, storage and communication modules can guide design choices at a specific abstraction level. The extracted statistics serve as input to many existing methodologies ranging from purely quantitative [4-8] to essentially analytical [9] approaches. It is not our goal to introduce new DSE methods, but to incorporate existing ones into a multi-level framework. Identifying the two main tasks – capturing of statistical information (i.e., design metrics) at a specific abstraction level and its incorporation into an exploration framework – the paper is structured as follows. In Section 2, we start by briefly reviewing trends in system modeling and related DSE techniques. In Section 3, we define the various abstraction levels necessary to introduce the multi-level DSE framework. In Section 4, we describe a systematic multi-level DSE methodology. In Section 5, the concepts, tool objectives and implementation characteristics of framed design metrics monitoring are discussed. Finally, these concepts are applied to specific aspects of a typical wireless algorithm known as Turbo decoding, in Section 6.

2. Related Work

The ability to perform algorithm and architecture level design space investigations in the early phases of the design process can have a dramatic impact (orders of magnitude!) on the area, speed, and power consumption of a system's implementation [10,11]. This suggests a multi-level DSE approach. However, the key concept in most of the research and commercial efforts in the field of DSE and verification is *co-simulation*. Current co-simulation frameworks (e.g.,

[12-14]) typically combine two (rather low-level) simulators, one for simulating the programmable components (e.g., ISS) running the software and one for the dedicated hardware (e.g., VHDL). Major drawbacks of this type of co-simulation are its inflexibility, lack of abstraction level granularity and efficient multi-level exploration capabilities.

In [15], a multi-level DSE framework was presented, which decouples the application (described as Kahn Process Networks [16]) and architecture models targeted towards multimedia applications. For wireless application, usually a data flow modeling approach is chosen. In [17], an integrated and systematic design approach is described, which allows the easy insertion and reuse of IP in SpecC (an alternative to SystemC). On top of defining a seamless SystemC design flow similar to [17], we present a multi-level DSE framework, which alleviates the inflexibility gap and supports easy interaction among different refinement levels. We introduce a unified approach to capture behavioral, memory and communication design metrics based on a unique principle of capturing and recording events. Hence, for example, the designer can explore memory and bus bandwidth requirements concurrently using the same tool at various abstraction levels.

Only a few references have been stated, with the goal to put our contributions into perspective. There are many other work items and contributions on very specific aspects of modeling (e.g., PTOLEMY [18]), tools (e.g., ATOMIUM [19]) and design space exploration (e.g., HW/SW partitioning heuristics [14]), which will not be discussed in this paper.

3. Overview of Abstraction Levels

Thanks to the homogeneity of the SystemC simulation environment and associated description language, moving from an abstraction level to another one, the models pass only a refinement process motivated by decision criteria during the exploration phase. In this paper, we split the design flow of interest into two algorithm and two architecture levels as follows:

- *Level Alg1*: Untimed functional data flow models implementing loss-less inter-block communication; The system parameters characterizing each module are extracted and represented as a static list. The main design goals in the exploration framework are functional breakdown and algorithm transformations such as bit width optimizations.
- *Level Alg2*: Untimed functional data flow models implementing loss-less inter-block communication, where the system parameters can be controlled dynamically at run-time. The main design goals in the exploration framework are control and data stream splitting and further algorithm transformations such as "algorithmic shut-down" power savings.
- *Level Arch1*: Untimed functional models using transaction level implementation of the inter-block communication; Definition of the parameter interfaces enables Firmware integration and testing. The main design goals in the exploration framework are to capture bus and storage statistics to guide the system architecture topology and to "prepare" the behavioral units to enable concurrency analysis in the next refinement level.
- *Level Arch2*: Relative timing information is added to the transactions and behavioral units. In essence, delays are introduced to reflect the estimated timing characteristics of the various modules. The main design goals in the exploration framework are HW/SW resource specifications, mapping of the application on those resources and their scheduling.

The various levels are illustrated in Figure 1a and 1b.

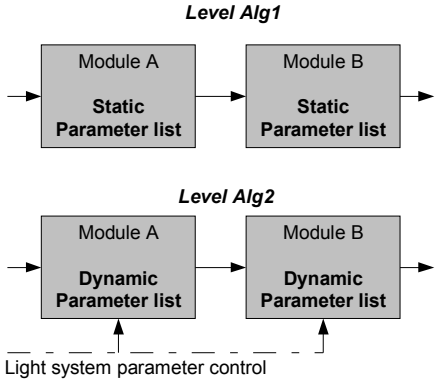

Figure 1a: Algorithmic abstraction levels

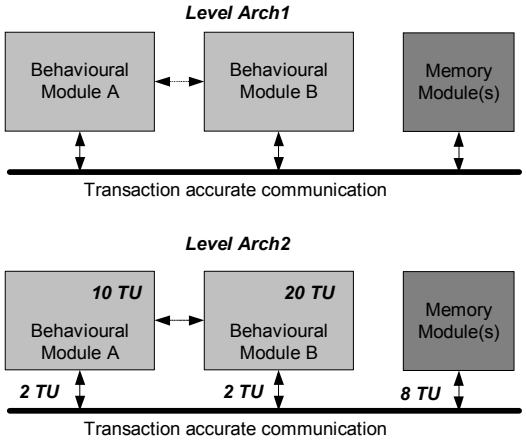

Figure 1b: Architectural abstraction levels, where TU denotes a relative Time Unit

Note, that the number of level refinements are application specific and mainly depends on the maturity of already performed DSE at a given abstraction level.

4. Design Space Exploration Framework

At every abstraction level, *statistics* can be extracted to guide design decisions subject to some *design objectives* and *constraints*. For instance, wireless applications are often specified by latency and throughput constraints; the designer intends to propose a system architecture with good communication performance, low power consumption and minimal die area as design objectives, which are typically in conflict with each other. This leads to a *multi-objective*

optimization problem, where the designer is really trying to find good compromises (or trade-offs) rather than a single solution as in single-objective optimization problems. Multi-objective optimization is an active field of research [20,21], and will not be elaborated further in this paper.

At each abstraction level, the designer may have to solve a multi-*objective* optimization problem. Moreover, in a multi-*level* design investigation framework a design objective may become a constraint in the next level. For example, the communication performance (e.g., BER) mainly dictates the wireless algorithm development at high abstraction levels, whereas it will become a constraint during the architecture investigation phase. Moreover, some of the objectives (and constraints) may not be directly accessible at every abstraction level. For instance, it is fairly difficult to incorporate precise power consumption and die area figures at a high level of abstraction. Nevertheless, case studies indicate that high-level decisions regarding selection and optimization of algorithms and architectures can improve (low level) design metrics by an order of magnitude, while gate and circuit level optimization typically offers a factor of two or less improvements [10,11]. It is therefore pertinent to incorporate low-level objectives very early in the decision phase by accessing and interpreting relevant model statistics. This can be achieved by introducing a weighting scheme to put the processing blocks (i.e., behavior, communication and memory) in relative perspective.

Next, relevant design metrics to be monitored in a multi-level DSE framework are introduced. We also show how the metrics undergo a similar refinement as the model themselves when passing from one abstraction level to another one:

- *(W)MOPS*: It specifies the number of million of operations possibly per time unit. A weighting scheme is introduced to put the mathematical operations in relative perspective (e.g., in terms of power dissipation characteristics or die area) and to handle more accurately information such as data types or word lengths. Intuitively, the higher the computation, the more power consumption and longer delays one expects. Thus, the introduction of weights can account for low-level design objectives early on in the design phase.

The "refinement process" of design metrics can be well illustrated for *(W)MOPS*. Starting at *Level Alg1* the designer may just need a rough computational complexity estimate to compare and select good algorithm candidates. Then, at *Level Alg2*, the designer provides a "golden" functional reference model with fixed word lengths and data types to start architecture exploration work. A weighting scheme can be introduced to put the mathematical operations in relative perspective and hence, to elaborate different algorithm transformations (e.g., >> operator is weighted differently then * operator). Multiple architecture templates are analyzed at *Level Arch1*., where relative delays (or weights) are introduced in the simulation chain to address its relative processing cost. Finally, for a specific template architecture, the designer can incorporate more detailed information such as Instruction Set Architectures (e.g., [22]) to model software components (e.g., DSP) at *Level Arch2*. Thus, the computational complexity is expressed in MIPS (millions of instruction per unit of time) rather than in (W)MOPS.

- *Behavioral concurrency statistics*: It illustrates the operational distribution statistics, which is a measure of the number of operations concurrently executed by each behavioral module during a certain time interval (e.g., one radio frame or one clock cycle). It allows identifying workload bottlenecks and design hot spots. Furthermore, the statistics indicate the regularity of the functional units over time, which impacts scheduling efficiency, conditional code structures in software (which in turn can increase significantly the (code) memory requirements) and hardware overhead of irregular cell structures.
- *Memory size*: Data stored in arrays or memory modules, respectively, is an indicator for overall memory requirements. Depending on the abstraction level, more detailed information such as maximum transport block sizes or peak memory requirements over time can also be captured to guide memory topology decisions.
- *Memory access statistics*: The memory access statistics – Where, When and What (WWW) – characterize storage properties such as temporal and spatial locality. This information will guide the functional memory refinements (e.g., temporal and static memory, buffers etc.) towards more efficient architectural design decisions.
- *Bus load concurrency*: The maximum height of the traffic distribution graph indicates worst case bus bandwidth requirements and should guide heterogeneous multi-bus topology decisions.
- *Bus access statistics*: The access statistics can guide the arbitration algorithm, thereby impacting the bus bandwidth requirements and architectural choices.

Note, that this is far from being an exhaustive list of possible design metrics. Our approach is scalable with respect to incorporating new metrics in the overall framework, which will impact the quality of the DSE efforts at each abstraction level. We want to stress that it is essential to provide an integrated and systematic DSE based on a set of design metrics across the algorithm, architecture and circuit level, since decisions made at each refinement level are correlated with previous design choices. Interaction (and feedback) among the different abstraction levels is simplified in a homogenous simulation environment such as SystemC, which removes the need for tedious model translations.

5. Framed Design Metric Monitoring

The main motivation behind design metric monitoring is the *automation* of complexity analysis at *every* abstraction level (and design space exploration) early on in the development phase. Our goal was to design a tool, which is tightly coupled a SystemC design flow by taking advantage of the object-oriented nature of the underlying C++ description language.

This section is structured as follows. First the implementation guidelines are outlined, followed by an overview of the framing mechanism and the supported frame protocols. Then, communication bandwidth issues and the underlying monitoring principle of capturing and recording events are discussed. Finally, figures such as reduction in simulation speed are included to characterize our prototype.

5.1. Tool Guidelines

Some of the important tool guidelines are summarized next:

- Dynamic (and statistical) analysis capabilities
- Compatibility with the SystemC simulation kernel semantics [23]
- Seamless use of the tool by the system designer: One should perform complexity analysis with minimal syntax changes to the SystemC modules to extract the design metrics. The *same* tool can be used to monitor data processing, storage and bus statistics at every abstraction level.
- Fairly transparent to the SystemC evolution: Since SystemC is still evolving, we decided to develop a tool that is loosely coupled to SystemC so that both of them can evolve independently but cooperate afterwards.
- Limited cost in simulation time (processing overhead as well as additional control and data communication bandwidth)
- Well suited for distributed simulations: Due to increased complexity of multi-level system investigations and simulation slow downs for low abstraction levels (more detailed models), one needs to support a distributed simulation approach to ensure reasonable simulation speeds.

5.2. Overview of the Framing Mechanism

A framing mechanism is proposed to support statistical analysis. Frames are controlled dynamically at run-time and define the laps of "time" during which design metrics are recorded for a given set of system parameters. The frames can be of different granularity and type. Possible options are time-driven (e.g., 1 cycle), data-driven (e.g., 10 bits), functional-driven (e.g., loop iterations) or system-driven (e.g., variable data rate links). For example, the behavioral concurrency characteristics of specific functional units are analyzed for various data rate links to study more efficient architectures with respect to average versus peak requirements. Each time the data rate changes at run-time, a new frame is initialized. The recorded statistics can be used to suggest further functional breakdowns. For instance, some of those refined blocks are then only activated for peak data rates and would not contribute to the power consumption during lower data rate scenarios.

To understand the frame control implementation as illustrated in Figure 2, one needs to understand the basics of communication and synchronization in SystemC [1]. In SystemC, the communication features can be identified as channels, interfaces and events. A channel is an object, which serves as a container for communication and synchronization. Channels implement one or more interfaces. An interface specifies a set of access methods to be implemented within a channel. An event is a flexible synchronization primitive that is used to construct application dependant synchronization. All three features had to be implemented as a monitor channel (MC) to allow communication and control between the monitor agent (MA), a SystemC module responsible for database (DB) management and module synchronization, and the data flow modules (DFM) under monitoring. The supported frame protocol features are discussed next.

5.3. Frame Protocol

The monitor channel supports the following protocol functionality:

1) Each DFM to be monitored sends a registration request (unique name, type $\in$ {master, slave}) prior to any signal processing to the MA, which initializes the DB accordingly. Upon reception of an acknowledgment (ACK), the DFM resumes the processing.

2) A DFM can register itself as master or slave. Master DFMs control the setup of new frames by sending a request to the MA. Currently, the tool supports two different modes. In Mode 1, only one DFM (registered as master) is associated with a MA to support "module independent" monitoring. Similar steps as in 1) are performed for each new frame setup request. In Mode 2, several DFMs are assigned to the same MA to support "module dependant" monitoring. Thus, the processing requirements among different modules sharing the same framing scheme can be compared. For example, the system control overhead during each loop iteration of a specific algorithm can be analyzed. In this case, the master DFM, implementing the algorithm consisting of loop iteration statements, requests a new frame at every iteration. The MA's tasks are then to setup new DB bins and to synchronize the master-slave DFMs as described next.

5.4. Communication Bandwidth

To minimize additional communication bandwidth among the MA and registered DFMs, local DBs are allocated to each of the DFMs. Each time an event occurs, the corresponding event counter of the local DB is updated. It is less costly to send one message every frame than to send an update over the MC for each new event (as shown in Diagram 2). We can get the active DFM directly from the SystemC kernel. The SystemC class *sc_simcontext* contains this information and has methods to access it (e.g., *sc_get_curr_simcontext(·)*). This enables us to pinpoint the local DB of the currently processed DFM. The monitoring DB at the MA is only updated once a new frame setup request is sent or at the end of the simulation.

Further characteristics of this implementation strategy are the decoupling of the tool and the SystemC development. Both tools are loosely coupled through the customized monitor channel, which is the only part that should be adjusted to support new SystemC Releases. Moreover, this

approach supports distributed simulations, since the number of independent MAs is not limited. Furthermore, there are no issues with inter-module communications such as global DB updates.

5.4. Implementation Principle

Identifying the three functional entities to be monitored as *behavior* (set of program statements computing something), *channel* (communication links) and *variables* (data stored in memory), the basic monitoring concept is common and consists of detecting and recording events. The object-oriented C++ description language supports features such as *operator overloading* and *namespace* definitions, which fit well the implementation requirements of these tasks. The pseudo-code of the overloaded addition *operator*+(·) for the SystemC specific integer data type (*sc_int*) is illustrated below, where in this example the overloading is implemented as an external function.

```
namespace wmops {
[ …]
  template<int V, int W>
  sc_int& operator + ( const sc_int<V>& v, const sc_int<W>& w )
  {
      // Record the data in the corresponding local Data base
      countOp( ADD, SC_INT, size );

      // Perform the actual operation
      v.sc_dt::operator+(w);
  }
[ …]
}
```

Note, that the use of a *namespace* limits the syntax modifications required to achieve monitoring to a minimum (e.g., syntax changes in the functional units are highlighted below).

```
#include wmops.h
[ …]
wmops_sc::sc_int a,b,c;
a = b + c;    // calling the monitoring operator+( )
```

The same implementation principles of capturing and recording events are applied to the communication and storage modules. For example, a loss-less communication channel at high level of abstraction can be implemented using customized FIFOs as defined in SystemC (i.e., *sc_fifo*). To monitor the communication, a hierarchical FIFO channel inheriting from the *sc_fifo* class can be implemented to record traffic events. At the transaction level, the channel models and corresponding statistics (e.g., control requests, block sizes in bytes, etc.) are further refined. Nevertheless, the recording is still based on the same principle – *capture and record traffic events*. The same concepts are applicable to storage monitoring. Due to the limited space, we will not discuss implementation details any further.

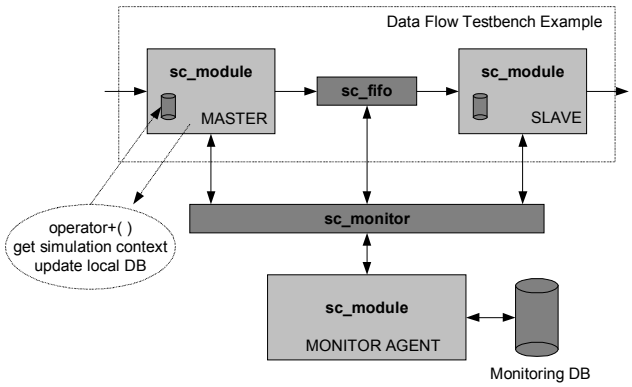

Figure 2: Framing control in SystemC

5.5. Prototype Characteristics

Our prototype was analyzed to estimate the increase in simulation time. The speed reduction factors (SRF) for capturing the behavioral events and the underlying communication overhead are illustrated. Diagram 1 shows the SRFs of the SystemC specific integer data type (*sc_int*) with respect to the basic integral type (*int*). The arbitrary list of overloaded operators and functions under test are {+, -, *log(·), abs(·), exp(·),* >, < }. The SRF was simulated for three cases – (1) monitoring turned off, (2) turned on with and (3) without context lookup. In the latter case, a global DB is updated with events from all modules in the test-bench. Since the SystemC kernel is not accessed, the granularity to pinpoint the event-generating module is lost.

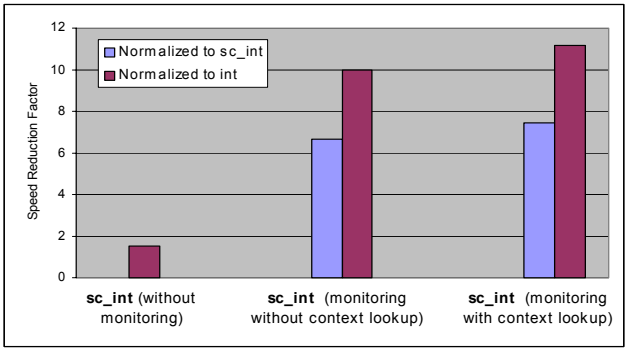

Diagram 1: Relative simulation speed statistics of the SystemC specific integer type (*sc_int*)

In the above diagram, the communication overhead between the MA and the DFMs is not included. The following figure illustrates the reduction in simulation speed with respect to the relative communication frequency (i.e., communication events are normalized by the number of monitored operations per module).

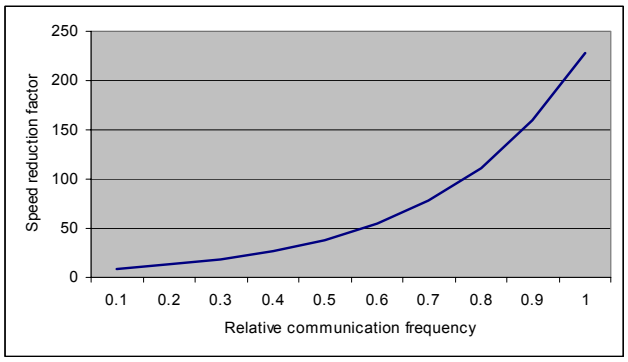

Diagram 2: Communication bandwidth overhead

We observe that the simulation time increases exponentially relative to the ratio of communication (i.e., DB updates sent to the MA) and actual signal processing operations. Nevertheless, in typical applications, we have observed that the number of operations has to be significantly larger than the communication frequency to yield meaningful design statistics.

6. Application Example: Turbo Decoding

The purpose of forward error correction (FEC) is to improve the capacity of a channel by adding some carefully designed redundant information to the data being transmitted through the channel [24]. Turbo codes are a class of FEC codes that offer energy efficiencies close to the limits predicted by information theory. Berrou *et al.* introduced an efficient iterative decoding algorithm known as Turbo decoding [25], which outperforms all previous known coding schemes. The principal modules of an iterative Turbo decoder are the component decoders (also known as maximum-a-priori (MAP) decoders), the generators for interleaved (or deinterleaved) addresses for each iteration, and some memories for fulfilling various buffering requirements. These functional units are illustrated in Figure 6. The abstraction levels of a Turbo decoding architecture have been investigated in [26, 27].

Due to the limited scope, we will only focus on a small piece of the design puzzle. To illustrate the tool's statistical analysis capabilities to support design space exploration, we investigate an iterative decoding problem known as *Iteration Control*. Continuing the iterative decoding process beyond a certain point delivers no additional gain in communication performance (i.e., bit-error-rate (BER)). Either the code block has been correctly decoded, or further iterations cannot correct the remaining errors. Hence, iterating further is wasted. A comprehensive list of iteration control algorithms can be found in [26]. A low-power design goal is to minimize the processing overhead and memory accesses, while satisfying BER constraints. We will limit the design space to two simple stopping criteria introduced in [28]. The analysis is based on functional models at *Level Alg1*.

The first criterion is referred to as the *sign-change-ratio* (SCR) criterion and can be summarized as follows. Let *S(n)* denote the number of sign changes in the extrinsic information $\Lambda_{\text{int}}^{2e}$ of the second functional MAP decoding unit from iteration *(n-1)* to iteration *n* and *N* is the code block size. Simulation shows that if $S(n) \leq (0.005 \sim 0.03)N$, iterative decoding can be stopped with minor communication performance degradation [27]. The second criterion is referred to as the *hard-decision-aided* (HDA) criterion. The hard decisions of the information sequence at the end of each iteration provide information on the convergence of the iterative decoding process. Simulation shows that when $sign\left(\Lambda_{LLR}^{(n)}(u_k)\right) = sign\left(\Lambda_{LLR}^{(n-1)}(u_k)\right)$ for all $k \in$ *code block of size N*, the communication performance degradation is small enough for terminating the iterative process [28].

Computational and memory (size and access) requirements per information bit are illustrated for the Turbo codes specified in [29]. The stopping threshold for the SCR criterion is set to $0.01 \cdot N$ and the simulations are performed over an additive-white Gaussian noise (AWGN) channel. Note, that the *Log-MAP* algorithm is considered [26].

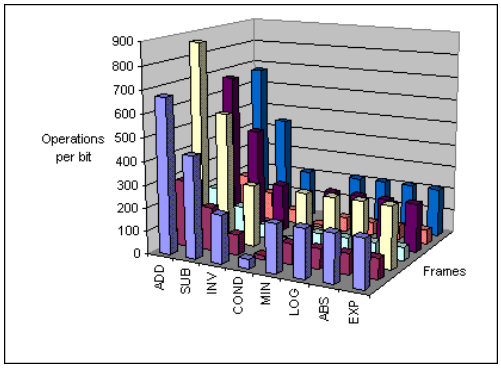

Figure 3: Computational Complexity per information bit of the MAP 2 decoding block in time (snap shot of SCR criterion at SNR = -1.0 dB for every iteration)

The computational distribution statistics of the second MAP unit illustrates the variable processing requirements to decode one code block for a specific signal-to-noise ratio (SNR). Note, that for generic interleaving tables, the two functional MAP units are bound to process data sequentially. Therefore, the computational complexity figure of the first MAP unit was omitted (and is simply a duplicate of the second MAP without the iteration control processing). The average number to decode one information bit at a specific SNR is shown next.

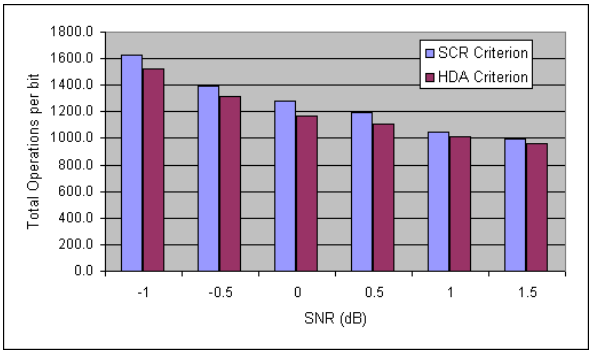

Figure 4: Average number of total operations to decode one information bit with respect to a specific SNR

We observe that for various channel scenarios (SNRs), the resource utilization is different. The designer has to compromise among communication performance (i.e., flexible number of iterations), low power design (i.e., minimum number of iterations) and constant throughput and latency constraints.

Another trade-off between a fixed and a variable iteration implementation are memory size and access statistics. To implement the iteration control the memory overhead consists of storing *N* soft bit (word-length dependant) values in the SCR criterion and *N* hard bit (word-length independent) values for HDA criterion, respectively. However, this overhead needs to be weighted by the power dissipated during the memory accesses. The array access statistics of one MAP unit is illustrated below for a fixed and variable number of iterations.

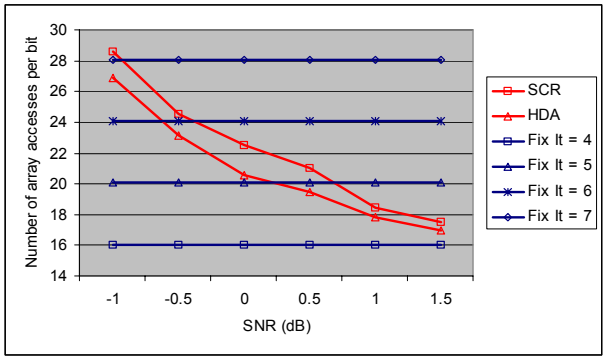

Figure 5: Array accesses per information bit for fixed versus variable number of iterations

We observe that setting the number of iterations to 7 to account for the worst case (here, SNR = -1 dB) implies 50% access overhead with respect to the best case (here, SNR = 1.5 dB). Thus, a low-power design suggests to include Iteration control.

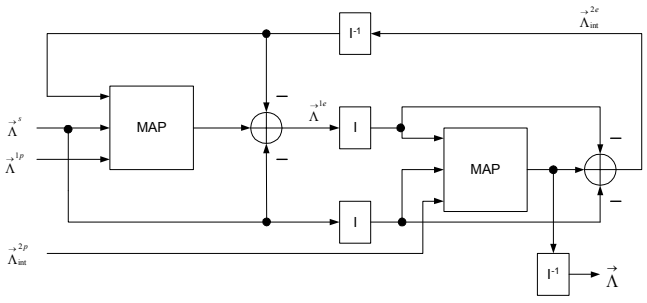

Figure 6: Turbo decoding overview

$\vec{\Lambda}^{S}$ denotes the systematic information in the log-domain, $\vec{\Lambda}^{1p}$ denotes the parity information of the first component encoder and $\vec{\Lambda}^{2p}_{int}$ denotes the interleaved parity information of the second component encoder in the log-domain, *I* and I^{1} specify the interleaver and deinterleaver, respectively.

7. Conclusion

Good techniques and tools are needed to efficiently explore the large design space of emerging systems at various abstraction levels. For instance, there is a growing need for supporting multiple applications and standards in wireless systems, which demands the development of smart solutions in increasingly competitive markets. In this paper, we presented a multi-level design space exploration framework based on a homogenous simulation environment, which allows easy interaction among the refinement levels, simple integration of existing IP and efficient verification of HW/SW components. Identifying the capturing of design metrics and their incorporation into an exploration framework as the main tasks to be resolved, a flexible monitoring tool implemented in SystemC was presented. A unified approach based on the concept of detecting and recording events was introduced to monitor behavioral, memory as well as communication statistics at several levels of abstraction. This ensures easy usage and quick tool evolution triggered by additional (or refined) design metrics to be captured in an integrated framework. Despite of some simulation speed degradation, we believe that framed Complexity analysis is a key asset for successful design space exploration of complex systems with a strong dynamic behavior like next generation mobile terminals. Finally, the utility of framed monitoring to support design decisions has been demonstrated on Iteration Control aspects of Turbo decoding used in wireless systems.

REFERENCES

[1] "Functional Specification for SystemC 2.0", Update for SystemC 2.0.1, Version 2.0-Q, April 2002, www.systemC.org.

[2] A. Haverninen *et al.*, "SystemC™ based SoC Com-munication Modeling for the OCP™ Protocol", White paper, www.systemC.org, Oct. 2001.

[3] J. Sanguinetti and D. Pursley, "High-Level Modeling and Hardware Implementation with General-Purpose Languages and High-level Synthesis", White Paper, Forte Design Systems, http://www.forteds.com

[4] M. Palesi and T. Givargis, "Multi-Objective Design Space Exploration Using Genetic Algorithms", In Int. Workshop on HW/SW Codesign, CODES 2002.

[5] T. Givargis, F. Vahid and J. Henkel, "System-level Exploration for Pareto-optimal Configurations in Parameterized Systems-on-a-chip", In Int. Conference on Computer Aided Design, Nov. 2001.

[6] P. Lieverse, "A Methodology for Architecture Exploration of Heterogeneous Signal Processing Systems", Kluwer Academic Publishers, Journal of VLSI Signal Processing, 29, p. 197-206, 2001.

[7] H.P. Peixoto and M.R. Jacome, "Algorithm and Architecture-level Design Space Exploration Using Hierarchical Data Flows ", ASAP, Zurich, 1997.

[8] B. Kienhuis *et al.*, "An Approach for Quantitative Analysis of Application-Specific Dataflow Architectures", ASAP, Zurich, July 1997.

[9] L. Thiele *et al.*, "Design Space Exploration of Network Processor Architectures", Symposium on High Performance Computer Architecture (HPCA8), USA, February 2002.

[10] J. Rabaey and M. Pedram, "Low Power Design Methodologies", Kluwer Academic Publishers, 1996.

[11] M. Pedram, "Power Minimization in IC Design: Principals and Applications", In ACM Transactions on Design Automation of Electronic Systems, ACM Press, 1996.

[12] C. Passerone *et. al,* "Fast HW/SW co-simulation for virtual prototyping and trade-off analysis", Proc. of the design Automation Conference, 1997.

[13] W. Sung and S.Ha, "Efficient and Flexible Cosimulation Environment for DSP Application", IEICE Trans. On Fundamentals of Electronics, vol. E81-A, no. 12, pp. 2605-2611, Dec. 1998.

[14] G. De Micheli and R.K. Gupta, "Hardware/Software Co-Design", Invited Paper, Proceedings of IEEE, March 1997.

[15] B. Halderen *et al.*, "Sesame: Simulation of Embedded System Architectures for Multi-level Exploration", Proc. of the conference of the Advanced School for Computing and Imaging (ASCI), pp. 99-106, May. 2001.

[16] S.L. Coumeri and D.E. Thomas, "A simulation environment for hardware-software codesign", Proc. of the Int. Conference on Computer Design, Oct. 1995.

[17] D.D. Gajski *et al.*, "The SpecC Methodology", Technical Report ICS-99-56, Dec. 1999.

[18] Edward A. Lee, "Overview of the Ptolemy Project", Technical Memorandum UCB/ERL M01/11, 2001, http://ptolemy.eecs.berkeley.edu

[19] F. Catthoor et al., "Custom Memory Management Methodolgoy", Kluwer Academic Publishers, 1998.

[20] C. Coello, "A comprehensive survey of evolutionary-based multi-objective optimisation techniques", Knowledge and Information Systems, vol. 1, no.3, pp. 269-308, 1999.

[21] D.E. Goldberg, "Genetic Algorithms in Search, Optimization and Machine Learning", Addison-Wesley, 1989.

[22] S. Pees *et al.,* "LISA - Machine Description Language for Cycle-Accurate Models of Programmable DSP Architectures", Proceedings of the DAC, 1999.

[23] W. Mueller *et al.* , "The Simulation Semantics of SystemC", Design, Automation, and Test Conference (DATE), March 2001.

[24] J.G. Proakis, "Digital Communications", 3rd Edition, McGraw-Hill, 1995.

[25] C. Berrou, A. Glavieux and P. Thitimajshima, "Near Shannon limit error correction coding and decoding: Turbo codes", IEEE Int. Conf. on Communications, pp. 1740-1745, May 1993.

[26] A. Worm, "Implementation Issues of Turbo-Decoders", PhD thesis, University of Kaiserslautern, July 2001.

[27] M.J. Thul et al., "A Scalable System Architecture for High-throughput Turbo-Decoders", IEEE Workshop on Signal Processing Systems (SIPS), October 2002

[28] R.Y. Shao, S.Lin and M. Fossorier, "Two Simple Stopping Criteria for Turbo Decoding", IEEE Transactions on Communications, Vol. 47, No. 8, August 1999.

[29] 3GPP TS 25.212, Release 4, www.3gpp.org