# Simulation Based Tuning of System Specification

Yaseen Zaidi
Institute of Computer Technology
Vienna University of Technology
A-1040 Vienna, Austria
Email: zaidi@ict.tuwien.ac.at

Christoph Grimm
Institute of Computer Technology
Vienna University of Technology
A-1040 Vienna, Austria
Email: grimm@ict.tuwien.ac.at

Jan Haase
Institute of Computer Technology
Vienna University of Technology
A-1040 Vienna, Austria
Email: haase@ict.tuwien.ac.at

*Abstract*—**We report an approach targeted to aid design exploration, early decisioning in model refinement, optimization and trade offs. The approach consists of SystemC AMS coupling with a descriptive functional simulator. System engineering tools are typically used in design and analysis of system prototypes captured at very high level. Naturally, in high level analysis accuracy and detail of results is compromised in lieu simulation speed and design effort. In the presented approach, the much needed abstraction and simulation speed is retained during simulation of platform architecture while near implementation models (RTL, SPICE) may be also be cosimulated with the architecture.**

## I. Introduction

Until now Electronic System Level (ESL) methodologies dominated digital domain of design. However, the launching of Analog and Mixed Signal (AMS) Extensions of SystemC by the Open SystemC™ Initiative marks the beginning of analog ESL. The automotive and signal processing prevalent applications spurred several years of academic and institutional research in high level AMS modeling, which eventually gave rise to justification of AMS abstraction [1]. The abstraction concepts of time [2], structure, frequency, behavior [2], communication [2], computation [2] and data [2] have been diligently used in designing the constructs and semantics of SystemC AMS language and simulator's kernel. The system designer, thus, can readily capture diverse AMS behavior with mere tens of lines of code by using Electrical Linear Networks (ELN), Linear Signal Flow (LSF) and Timed Data Flow (TDF) Models of Computation (MoC).

The semantics of SystemC AMS provide capability to explore design space, investigate possible analog-digital partitioning and co-designing. Fast simulation speed allows the designer to design experiments, reorganize architectures, develop virtual prototypes, perform what-if scenarios, tune specification or use the system simulation model to drive later verification. Note that the notions such as cycle accuracy by a clock, boolean equation solver, adaptive stepping, nonlinear solver, analog operations (slew rate, crossing, wave smoothening), time derivatives and integrals have been deliberately left out. However, such constructs are absolutely necessary for accurate modeling for a designer to proceed further down toward refinement from architecture level. Since extending SystemC AMS with the solver concepts of hardware design AMS languages [3] would violate the philosophy of analog abstraction, the modeling capability, therefore, is complemented as needed, with the modeling paradigms established in standard Electronic Design Automation (EDA) flows.

To bridge the gap, we integrate Cadence mixed signal simulator directly in SystemC AMS simulator. SystemC AMS supports top-down design flow and refinement methodology via its Timed Data Flow (TDF) model of computation. The simulator uses constant time stepping (coarse solver) for overall system level simulation, but for blocks needing deeper behavioral characterization, our work utilizes precise mixed signal solvers and adaptive time stepping of Cadence tools. The main advantage of the approach is that cosimulation is employed at selective blocks of the system architecture. Therefore, the overall system architecture remains abstract in time and behavior, however, the block where cosimulation is employed local behavior is precise in computation, timing, numerical accuracy and stability.

In Section II we present integration detail of Cadence IUS simulator with OSCI SystemC AMS. In Section III we apply the integration setup to the executable description of a complex mixed signal transceiver, in which an abstract model block (parity checker) is replaced and simulated with RTL model of Data Encryption Standard (DES). In Section III-C and III-D we evaluate performance of the integration package and discuss limitation of data access between two simulations. Section IV is about the final remarks.

## II. SystemC AMS and Cadence Integration

A C based integration package has been designed to coordinate the system and circuit level (Cadence) simulations. The interaction with Cadence tools is possible due to open nature of the OSCI simulator. The package uses standard facilities of operating system and procedural language interface hosted in Cadence simulator. The package can be easily modified for use with any simulator that has procedural language interface.

### A. Application Specific Customization of Simulator

The OSCI SystemC AMS is an agile, open source and open architecture simulator. The simulator can be used at the front end in analog ESL designs. The simulator is canonic by design in order to maintain speed and simplicity, but it offers enrichment features. Illustrated in Fig. 1, SystemC AMS is a layered architecture specifically designed

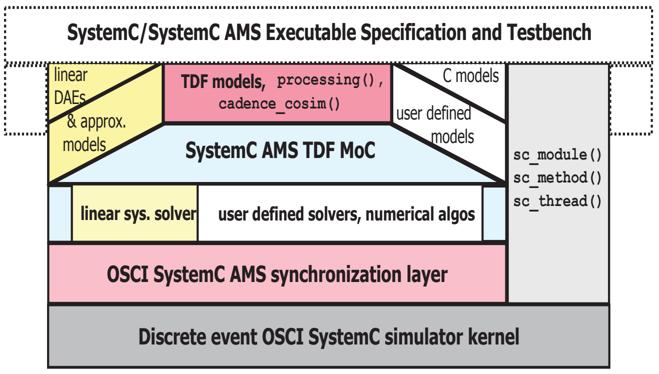
SystemC/SystemC AMS Executable Specification and Testbench
linear DAEs & approx. models
TDF models, processing(), cadence_cosim()
C models
user defined models
SystemC AMS TDF MoC
sc_module()
sc_method()
sc_thread()
linear sys. solver
user defined solvers, numerical algos
OSCI SystemC AMS synchronization layer
Discrete event OSCI SystemC simulator kernel

Fig. 1. Open architecture nature of OSCI simulators

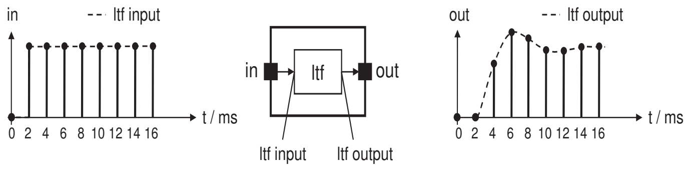

Fig. 2. Continuous time representation of a sampled discrete time step signal using an LTF function. The LTF processes the discrete time signal at input port as continuous time. The continuous time signal (filtered) is again sampled to discrete time at the output port

for the user to enhance the simulator with specialty features through C/C++ add-ons: programming interface e.g. Verilog Procedural Language Interface (VPI) and VHDL Procedural Interface (VHPI), dedicated solvers e.g. DASSL for non-linear differential algebraic equations and supplementary libraries or packages e.g. Monte Carlo, to name a few.

## B. Anatomy of the Timed Data Flow Model of Computation

The TDF model of computation models continuous time behavior through time abstraction. The continuous amplitude behavior is approximated as highly sampled signals with discrete time and discrete amplitudes. The TDF processes a discrete time sample as a continuous time one, by holding the sample until the next discrete time. A Laplace Transform Function (LTF) in TDF is solved by using discrete sample values held constant between sampling time, mimicking a continuous time signal. The output is continuous in both time and value, however, the TDF again samples the continuous time signal and writes a discretized signal in discrete time to the output using the output port attributes. This is demonstrated by the example of Fig. 2.

The TDF model of computation can be used to act on discrete time samples by a user defined C++ function. To gain simulation speed, the TDF uses the notion of samples that are equally time separated and their count is fixed at elaboration by a static scheduler. Static scheduling [4], was agreed upon by the architects of the SystemC AMS for observing fast speed and averting the computational overhead of dynamic scheduling. The step size used in TDF model of computation is held constant throughout the simulation, which also accelerates simulation as opposed to adaptive stepping. The TDF sets rate attribute to maintain data flow at the ports and delay attribute for providing latency. Additionally, TDF encapsulates a synchronization layer to synchronize all SystemC AMS model of computations with discrete event SystemC.

An elaborated TDF cluster is a graph of TDF blocks or nodes. The input and output port rates are used to calculate a firing schedule of all nodes. The schedule maintains the dataflow rates in and out of the nodes. For example, if node $B$ of Fig. 3 is the block that needs to simulated and refined as a circuit level model, the TDF `processing()` method will not be activated unless node $A$ produces three samples, for which the node $B$ will consume two samples. In this case, the balance equation of branch $AB$ for $R$ repetition in each firing and a minimum, positive $n$ is $3 \times \; R_A = 2 \times \; R_B$, yielding a schedule $\begin{bmatrix} R_A & R_B \end{bmatrix}$ = $\begin{bmatrix} 2n & 3n \end{bmatrix}$ where $n \in Z^+$, while $BABAB$, $BBABA$, $AABBB$ and $BBBAA$ all are possible schedules. The port rates must be reasonable in the TDF cluster to calculate a valid schedule between the nodes.

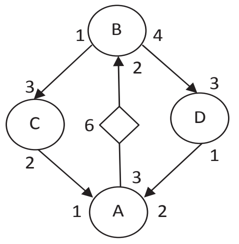

Fig. 3. A static schedule determines node firing vector before simulation in a TDF cluster of nodes

In our work, we encapsulate a circuit level cosimulation interface within the TDF `processing()` method. The advantage of wrapping cosimulation call inside TDF is that, firstly the TDF executes the cosimulation call as a user defined method of own computation. Secondly, the synchronization of the two simulations is autonomous because the TDF has no information of a foreign simulator, and the external simulation is synchronized with both SystemC and SystemC AMS kernels every time the TDF block (that issues the cosimulation call) is synchronized locally.

## C. Design Refinement

SystemC AMS refinement methodology is top down which starts with executable specification of the system. The intended system behavior can be captured using the TDF model of computation which supports timed flow of signals. The next step concerns replacing the abstract models of system with more accurate models (behavioral or structural). For analog case the replacement process is stepwise, however, for digital behavior (algorithmic nature) the replacement can be immediate, which is the approach followed in application example of Section III. The simulator speed is assisted by static scheduling, constant sampling, time abstraction and data flow, which permit multiple simulation runs for performance evaluation and feasibility of architectures. Fig. 4 illustrates the application of the

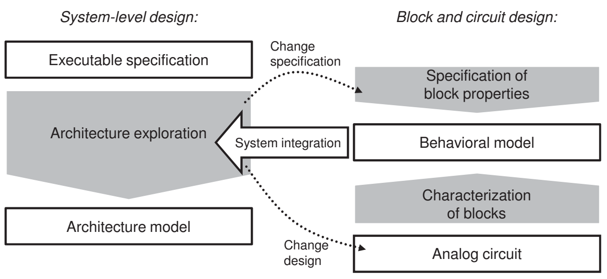

Fig. 4. Specification refinement from system concept to platform architecture

SystemC AMS extensions in system refinement. The user can lower the level of abstraction for the intended domain of implementation such as space state, LSF, LTF and ELN. For instance, a lowpass filter may be described as pole-zero ($H_s = \frac{H_o}{1+\tau s}$), by cut-off frequency ($H_s = \frac{H_o}{1+\frac{1}{2\pi f_c}}$), space state equations ($y(t) = x(t) - \frac{1}{2\pi f_c}\frac{\mathrm{d}}{\mathrm{d}t}y(t)$) or using circuit components ($\frac{1}{1+sRC}$).

### D. Bridging Coarse and Fine Grain Simulations

The TDF `processing()` method wraps a cosimulation method `cadence_cosim()`. The cosimulation method can be instantiated in clockless discrete event blocks described in `sc_module()` or `sc_method()` that connect TDF blocks. In the listing below, a TDF node invokes an interface for fine grain cosimulation passing it data read via the input port. The node will remain in processing state in the cluster of TDF nodes until `cosim_data` is returned. Note that in the absence of cosimulation interface (`cadence_cosim()`), the user would describe block behavior with own code.

```
SCA_TDF_MODULE(TxRx) {
sca_tdf::sca_in<double>  in;
sca_tdf::sca_out<double> out;
void set_attribute()
void initialize()
processing() {       // SystemC AMS user defined method
in.read(input);      // read input port
/* invoke cosimulation and pass input data */
char *cosim_data = cadence_cosim(carrier_freq,
rate, bit_resol)
out.write(output);  // write cosimulated data to output
}
};
```

The `cadence_cosim()` method formats input port data and communicates it to the Cadence cosimulator via TCP sockets. The method is flexible to take additional arguments from the SystemC AMS description that might be necessary in cosimulation. Typically, the port arguments are `set_rate`, `set_delay()` and `set_attribute()` which aid in model computation. The arguments are communicated to the cosimulator via socket connection. The cosimulator uses simulator's procedural language interface (VPI or VHPI) to write the simulation data originated at the SystemC AMS model to the circuit level model. The testbench configures the circuit level model according to the port arguments received from SystemC AMS model. This configuration provides semantics for the circuit level model. For example, the SystemC AMS `set_attribute()` may be interpreted as `generic` for VHDL model. Similarly, `set_rate` may be interpreted as number of data at clock edges, and `set_delay()` is abstracted as clock latency. The designer must judge whether the arguments are needed to be mapped in the circuit model. In many cases, such as in the case study of Section III, the need of providing equivalence for system attributes in the circuit level model is unnecessary.

The simulation data returned by `cadence_cosim()` in the return argument is stored as a data stream. The argument `cosim_data` is the cosimulation data saved by the VHPI/VPI interface and received in the socket connection. The `processing()` method converts the stream into the SystemC AMS data type needed for the output port. The TDF blocks only reads input port data from SystemC AMS model, the computation is carried out at the cosimulator and the computed resulted is written at the TDF output in SystemC AMS. The cosimulator applies its fine solvers on abstract input data generated at SystemC AMS. Therefore, the computation whether digital or analog is precise both in accuracy and time. Moreover, the analog non-idealities and non-linearities would rise that typically cannot be observed in SystemC AMS uni-simulation.

### E. Loose Coupling

The time marking of the samples at SystemC AMS has no use at the cosimulator and therefore this information is not communicated to it. Because top level SystemC AMS description is abstract, the connectivity of modules with each other is supported by a few signals. When an abstract model is replaced (by cosimulation) by a circuit level model, the mapping of ports between two models is not one to one. Thus, in cosimulation the signal flow path between the two simulations is of interest. However, the cosimulated model (refined) requires many more signals. We accommodate such non-interfacing signals of the cosimulated model in it's local testbench. The dataflow samples originated at SystemC AMS are applied to the cosimulated model with the local clock of the testbench and timed accordingly with the model's other signals.

### F. C Level Access of Cosimulation

The access of cosimulation objects is made through procedural interface of the cosimulator. The interface consists Verilog VPI or VHDL VHPI task functions that register simulation begin/end callbacks, start/stop time callbacks, navigate design hierarchy, obtain handles on various design objects, monitor objects that participate in cosimulation and save value change events on those objects. The relationships and rules for these tasks are illustrated in Alg. 1. In brief, whenever an event occurs on a simulation object contained in the monitored list, the data and time of that event and object is recorded. If the object is the signal flow object between the two simulations, at the end of cosimulation, the saved stream shall be the cosimulation data used by SystemC AMS simulator.

```
Input: Set of model instances to be cosimulated, start
       and stop times, socket ID
Output: Values of the interested signals or ports
while not stop time do
    foreach Model $i$ do
        register begin simulation callback
        register end simulation callback
        register add ValueChange callback at start
        register remove ValueChange callback at stop
        navigate Model $i$ hierarchy to all SubModels
        foreach SubModel $j$ do
            navigate design hierarchy
            get port and signal objects
            assign ASCII handle to each object
            initialize objects
            add desired object $k$ to monitor list
            add ValueChange callback on $k$
            foreach $k \in SubModel\ j \in Model\ i$ do
                if ValueChange event occurs then
                    read (value,time)
                    write value to socket ID
                end
            end
        end
    end
end
remove ValueChange callback
```

**Algorithm 1:** Typical tasks in C level access of HDL simulation

## G. Lock Node Synchronization

The TDF interfaces with the synchronization layer shown in light pink color in Fig. 1. The layer also synchronizes SystemC methods (`sc_module()`, `sc_method()`, `sc_thread()` to SystemC AMS TDF. By the nature of static scheduling of synchronous data flow graphs, once the cosimulation has begun no dynamic activity is expected at the node's input. When a TDF block has not produced the required number of samples, the next block in the data flow will not execute in a firing which is a single non-interruptible step. The unavailability of samples at the block output means temporarily imbalance of equation system and that the TDF solver has not finished computation to produce the required number of samples. Thus, the block is in momentarily suspension for data flow activity. As soon as the computation is complete, the computed result in form of samples is written to the output. This is the basic synchronization idea used in connecting SystemC AMS with Cadence IUS, which is in effect data synchronization.

For cosimulation of TDF node B inside a TDF cluster of Fig. 5, with input port rate $r_{B1}$ is defined by a composite mathematical expression $f$ of sub-expressions $f_A$, $f_B$ and $f_C$:

$$f \to (f_C\,(f_B\,(f_A)))$$

The node B resides between TDF nodes A with output rate $r_A$ and node C with input rate $r_C$. A possible static schedule exists:

$$A \to B \to C$$

The number of tokens represented by $r_B$ must be available before the simulation wrapper is activated.

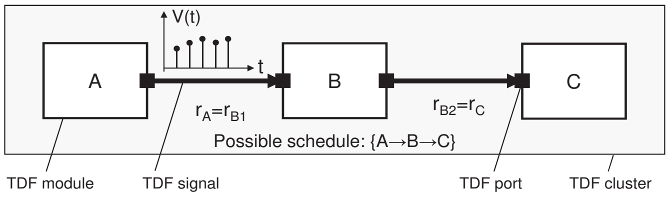

Fig. 5. Number of tokens needed in cosimulation must be available before cosimulation of node B begins

In cosimulation, the TDF member function `processing()` has no computation method (solver), rather it instantiates the cosimulation wrapper `cadence_cosim()`. Hence, the TDF processing is subjected to the time the cosimulator takes at the remote machine. Until this time, the TDF node is inactive in communication to all forward nodes, while other TDF nodes might be processing that have enough samples at their inputs. Therefore, SystemC AMS simulator is not in a complete deadlock state because of the cosimulator. If the cosimulator takes longer, the remaining TDF nodes may run out of their samples. This will produce a cyclic dependency in the TDF cluster, which will be cleared as soon as the cosimulation data (samples) arrive in the node that had triggered cosimulation. Recall that cosimulation data is the return argument of `cadence_cosim()` which is written to the TDF block output ports by `processing()`.

## III. Application Example

A very high level specification of a Amplitude Shift Keying (ASK) modem is captured in SystemC AMS described by the behavior blocks of Fig. 6. The cryptographic block (right most grayed blocks in Fig. 6) is simply modeled as XOR parity logic (8-bits plus parity). The transmitter and receiver are connected through AWGN noisy medium shown in the left most gray block.

In cosimulation, the encode and decode code shown in listing below is replaced by two `cadence_cosim()` methods, one for transmit and the other for receive path. The methods cosimulate 64-bit Data Encryption Standard RTL (537 slices, 232 Flip Flops and 1026 Look Up Tables on Xilinx 3s50pq208-5 device). SystemC AMS `set_attribute` sets the consumer and producer token rates for each block. The transceiver data passes through the noisy channel and is encrypted/decrypted using the cryptographic DES and DES3 soft cores.

```
..
/* decode */
sc_bv<8> val = Rx.read().range(7,0);      // discard parity
out.write(val ^ "10101010");}
/* encode */
parval.range(7,0) = in.read() ^ "10101010";
parval8=parval.range(7,0).xor_reduce();  // add parity
Tx.write(parval);
..
```

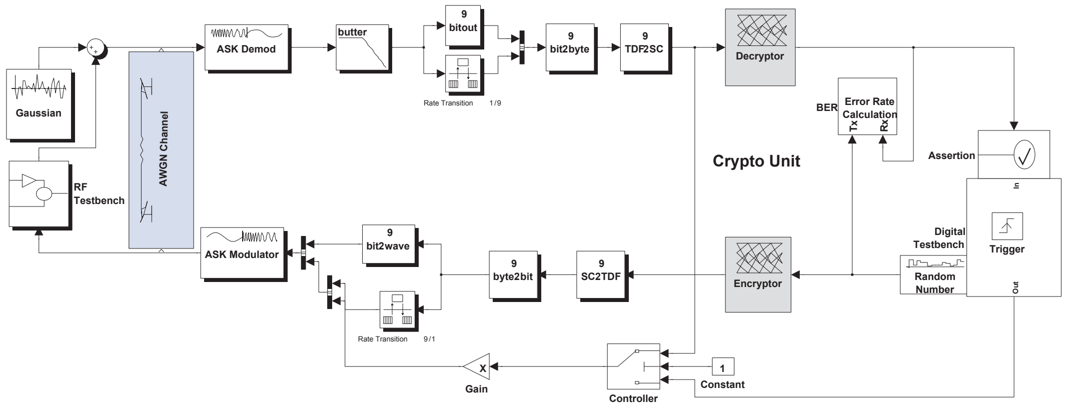

Fig. 6. ASK transceiver modeled in SystemC AMS, crypto unit is cosimulated as DES in RTL with Cadence IUS

## A. Measurements

Fig. 7 shows ASK signal before (blue trace) and after (green trace) the channel effects. There is 80 dB loss in the three peaks shown in the frequency spectrum while the is no deterioration in the phase response. The signal is demodulated in the receive path and the encrypted bits are recovered by a lowpass filter whose response is shown in Fig. 8. The bit stream is illustrated in both analog and digital representations in Fig. 8.

Fig. 9 is the eye mask of the detected bits immediately after lowpass filter. The 64-bit data encrypted signal is interposed with Inter Symbol Interference (ISI). We note a keep out area marked by at least 14 µs horizontal and 44 µs vertical eye openings. Moreover, the signal contains RMS jitter (standard deviation of the jitter calculated from the horizontal histogram) of 31 µs and a peak to peak jitter (difference between the extreme data points) of 94 µs. Jitter measurement in absence of cosimulation (abstract 8-bit parity checker) is shown in Fig. 10. In this case, signal quality is higher marked by wider eye opening (height = 109 µs and width = 31 µs). The strength of jitter is additionally illustrated as histograms at the occurrence in Fig. 9 and 10.

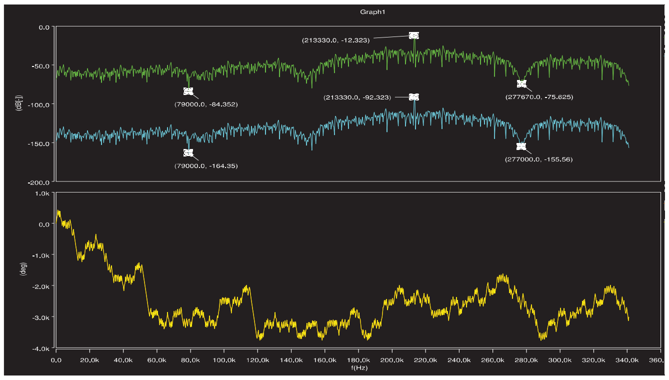

Fig. 7. Noisy channel effect on input signal, spectrum and phase response

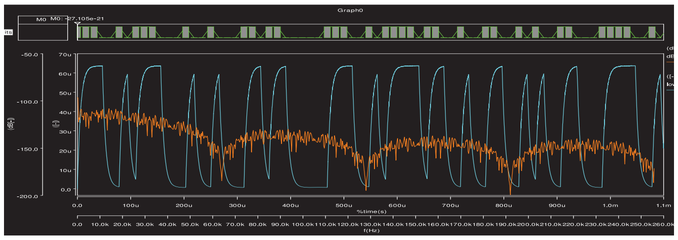

Fig. 8. Lowpass Filter detected bits - analog and digital representation

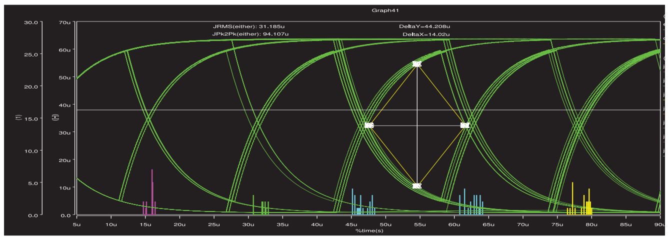

Fig. 9. Jitter in detected bits that were encoded as 64-bit DES cryptography

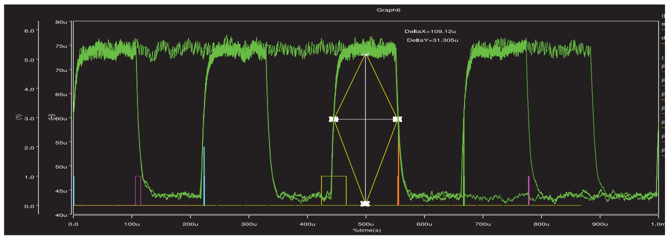

Fig. 10. Jitter in detected bits encoded as 9-bit XOR parity

## B. Specification Tuning

Fig. 11 shows the available degree of freedom to the designer through abstraction (high simulation speed) and refinement (cosimulation). The designer can put together a system prototype by abstractly coding a modulation scheme (height edge of the cube in Fig. 11), selecting a candidate of cryptographic model from a cosimulation library (width)

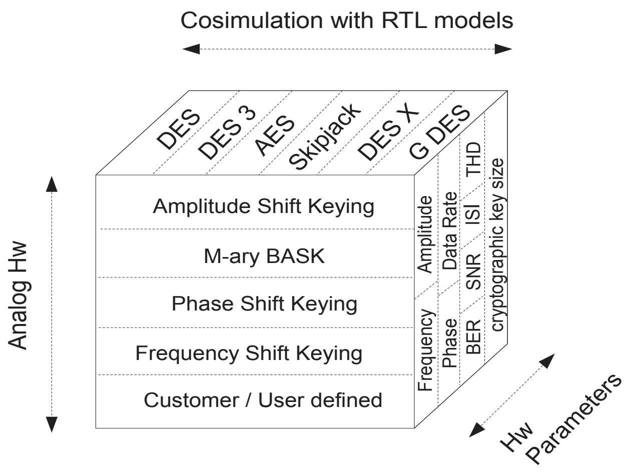

Fig. 11. Design exploration and specification refinement using cosimulation

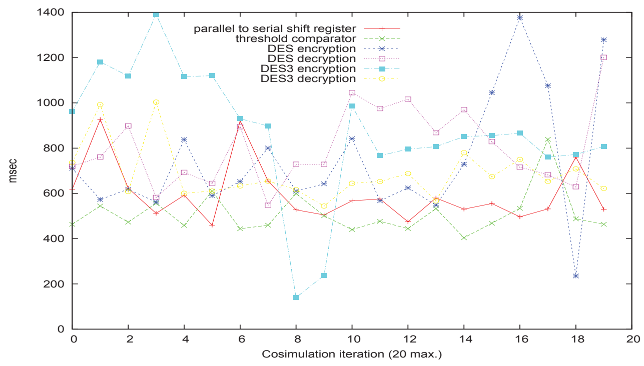

Fig. 12. Execution times of cosimulation interface for different models

and analyzing performance of virtual prototype on parameters of interest (cube depth). For ASK switching scheme, the detection of wave bits can be improved by experimenting with carrier frequency and threshold voltage. A second option is reduce the channel bandwidth by encoding the amplitude of cryptographic symbols with different levels (e.g. M-ary Binary ASK with 6-levels for 64-bits). For phase based modulations e.g. Phase Shift Keying (PSK), the receiver may contain a phase locked loop for more accurate detection. For frequency modulation, the parameter of interest for measuring signal distortion with time (non-linear phase shift and frequency relationship) would be group or envelope delay. Similarly, the designer can fine-tune bit error rate, signal to noise ratio and transmission rate. SystemC AMS provides abstract analog modeling capabilities for designing such experiments.

In automotive keyless entry application, interference and reflections are primarily affect secure ASK transmission. Data security is directly affected by the size of the cipher key. The bigger sizes, however, inherently result in higher distortion and ISI. Cadence cosimulation suite coupled by SystemC AMS can provide further assistance in architecture exploration such as by simulating RTL cores of Advanced Encryption Standard and Skipjack with the system level executable specification.

### *C. Performance Evaluation*

Fig. 12 indicates computation times of the cosimulation (Intel Xeon 3.39 GHz running Debian Linux) interface for several models of varying complexity. The computation includes complete automation including compiling, elaboration and simulation of RTL models. The latency is inherent of distributed computing and typically ranges between 400 and 1000 ms for both DES and DES3. These times can be reduced by using shared memory as opposed to sockets and also by simulation of pre-compiled and pre-elaborated models.

### *D. Framework Limitation*

Currently, C level access of analog simulation data is not possible either by any tool or IEEE standard. However, to this end, work is underway standardization committees. If the model to be cosimulated contains SPICE description, then the model must be encapsulated inside a Verilog-AMS wrapper. The VHPI/VPI application cannot access continuous time objects: SPICE nets, VHDL-AMS `quantity` or Verilog-AMS `electrical` objects. Hence, HDL-AMS nets that are probed by the C interface must be real valued discrete time objects as registered by the simulation kernel. This is not a limitation in principle as low level continuous time signals typically do not directly come in the signal flow in the system level blocks. The designer however would be interested in their indirect contribution to the overall analog behavior and on the signal flow to the other blocks.

## IV. Conclusion

A scalable simulation based approach of refining system architecture level blocks by replacing them with circuit level description is presented. The simulation setup takes advantage of high level modeling capabilities of SystemC AMS simulator which include open architecture, TDF MoC, static scheduling and constant time stepping whereas finer behavior is simulated with Cadence functional simulator. The synchronization between the simulators is facilitated by SystemC AMS synchronization layer. Synthesizeable DES and DES3 models are directly cosimulated with the executable specification of an ASK transceiver. The effect of noise is analyzed as a test case for adjusting system specific parameters. The simulation setup can assist in selecting the cryptographic algorithm. The simulation speed although in hundreds of ms allows the designer to add near implementable models in the architecture being explored and evaluate system performance therefrom.

## References

[1] A. Vachoux, C. Grimm and K. Einwich, “SystemC-AMS Requirements, Design Objectives and Rationale,” In Design Automation and Test in Europe (DATE), pp. 388–393, 2003.
[2] A. Jantsch, *Modeling Embedded Systems and SoCs: Concurrency and Time in Models of Computation,* Morgan Kaufmann Pub., 2004.
[3] Al-Junaid, H., Kazmierski, T. and Wang, L. (2006) SystemC-A Modeling of an Automotive Seating Vibration Isolation System. In: Forum on Specification and Design Languages (FDL 2006), September 19-22, 2006.
[4] E. A. Lee and D. G. Messerschmitt “Static Scheduling of Synchronous Data Flow Programs for Digital Signal Processing,” *IEEE Transactions on Computers*, vol. 36, pp. 24–35, 1987.