System-level Synthesis from Transaction-level Models: Algorithms and Tools

Rainer Dömer
Center for Embedded Computer Systems
Irvine, CA 92697–2625
Email: doemer@cecs.uci.edu

Daniel D. Gajski
Center for Embedded Computer Systems
Irvine, CA 92697–2625
Email: gajski@cecs.uci.edu

***Abstract*— With design complexities increasing daily, the multi-core community is entertaining the idea of increasing the level of abstraction to transaction-level modeling (TLM) and design. However, the proper definition, style or semantics of TLM is not clear. Nor is it clear how to synthesize or verify TLMs. In this paper, we will introduce several TLM models and define their semantics. This formalism will allow us to define design decisions and corresponding model transformations that can be used to transform one model into another. These transformations and refinements are the enabler for automatic synthesis and verification on TLM. We will also discuss the algorithms and flow for model transformation according to the OSI network layers and show how to build tools with inputs and outputs at transaction level. We will conclude with preliminary tools and results that promise a productivity gain of several orders of magnitude.**

I. Introduction

The complexity of embedded designs has reached a level beyond what human system designers can produce with traditional approaches and EDA tools. Our approach summarized in this paper incorporates more than 15 years of research in system synthesis to provide a solution that will reduce both time and effort needed in the system design process. Given a system specification of the application described graphically in form of hierarchically composed C code together with a platform target architecture description, our approach allows to automatically generate transaction-level models (TLM) [1] for simulation, analysis and verification, as well as a pin- and cycle-accurate model (P/CAM) for implementation.

II. TLM Abstraction Levels

The standard product design starts with an application code for which designers envision a multi-core platform architecture. This application code is then partitioned and mapped to components in the platform, thus leading to a system specification. Each component in the architecture must further be refined to a pin- and cycle-accurate level for synthesis with standard EDA tools. Similarily, the application code must be refined to allow communication through the network on the platform.

In order to automate this refinement, we need to define proper abstraction levels, design decisions at each level, and necessary refinement steps for each system model, in order to generate a new model corresponding to those design decisions.

In general, three models are necessary,

(a) the system specification, written by application designers
(b) TLM, to validate the system specification on the selected platform, and
(c) P/CAM, generated by system designers for input to standard EDA tools.

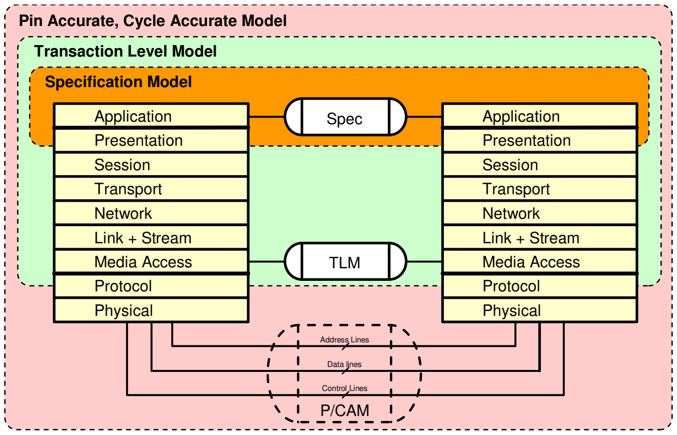


Fig. 1: Model abstraction and communication layers.

Figure 1 illustrates the abstraction levels of the models in the design flow with respect to OSI layers [2]. The input specification model is a untimed, hierarchical, functional description of the system, using abstract communication channels. The timed intermediate TLM is partitioned into the system's processing elements, communicating over fast and timing-accurate TLM channels. The final implementation model is pin- and cycle-accurate and feeds directly into standard design tools at lower levels.

III. Design Flow

The design flow to the corresponding abstraction levels allows an application designer to capture the system specification at a higher abstraction level. The specification is then validated and evaluated to determine its necessary specifics and required properties using a transaction-level model (TLM).

The application engineer can then change the platform components and connections or the application code until satisfactory results are obtained. Once the platform and the code satisfy the given requirements, the system designer generates pin- and cycle-accurate code.

The TLM and P/CAM models can be generated automatically using a decision-based refinement methodology. Such a methodology associates with each design decision or design change a corresponding model refinement or change, resulting in a model transformation that produces a new model that reflects the selected design decisions.

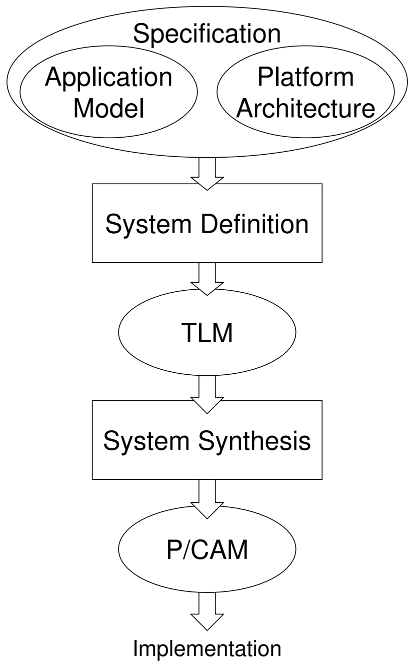


Fig. 2: Embedded system design flow.

Figure 2 shows an overview about the overall design flow and the system design environment supporting it. Such a design flow starts with the capture of the *application model*, a purely behavioral description of the system functionality. Independently, the system *platform architecture* is defined as a system netlist of major system components, including processors, dedicated hardware accelerators, memories and IPs, interconnected by system busses, bridges, and transducers. Together, the application model and the platform architecture form the system specification as input to the design environment.

The system specification can be seen as a combination of the application model and platform architecture, integrated with additional information taken from the system component database. From the system specification, model generation tools automatically generate transaction-level models (TLMs) towards validation and exploration, while system synthesis tools generate a pin- and cycle-accurate model (P/CAM) that serves as input to standard EDA tools for the system implementation.

A. Application model

The input application model is a purely functional, executable specification of the intended design. It consists of a hierarchy of sequential or concurrent functional blocks that communicate by use of abstract channels reflecting various types of message-passing communication semantics. In other words, the model is a hierarchical composition of blocks defined as ANSI C code.

To enable true design space exploration, the application model does not contain any implementation details. In particular, the model is architecture-less, that is, it is void of any structural information.

To allow functional validation, the application model also contains stimulus and monitor behaviors that build a testbench for the design model.

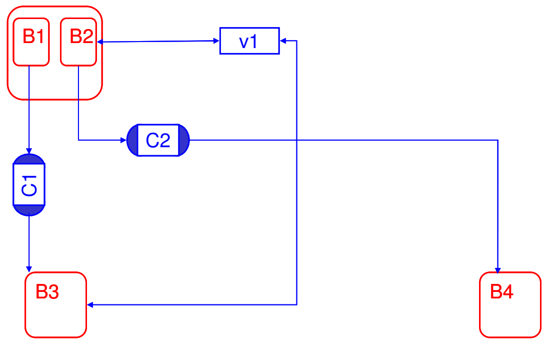


Fig. 3: Application model example.

Figure 3 shows an example of a simple application model. Four concurrent functional blocks `B1` through `B4` communicate via shared variables (`v1`) and abstract channels `C1` and `C2`.

B. Platform architecture

As outlined above, the platform architecture is the second input to our design flow. The platform model describes a system netlist of the major components, such as software processors, dedicated hardware blocks, memories and intellectual property (IP) components. Following a general block diagram paradigm, the system components are interconnected by system busses which in turn can be connected by bus bridges and transducers.

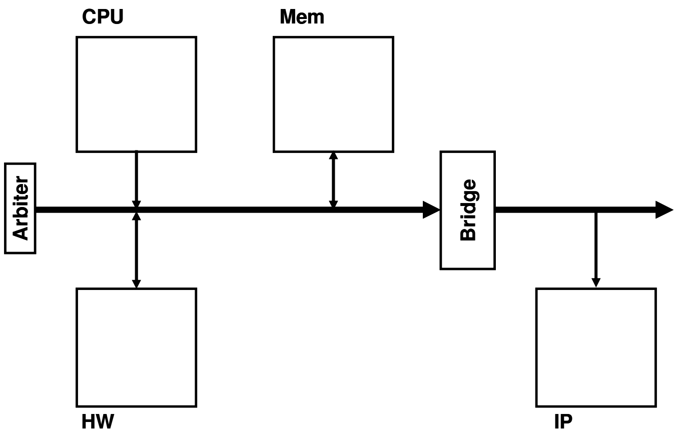


Fig. 4: Platform architecture example.

Figure 4 shows a platform architecture suitable for the example shown in Figure 3. This simple example system consists of a general-purpose processor `CPU`, a hardware accellerator `HW`, a shared memory `Mem`, and a third-party block `IP`. The four components are connected by the main processor bus and a bridge to the IP bus.

C. System specification

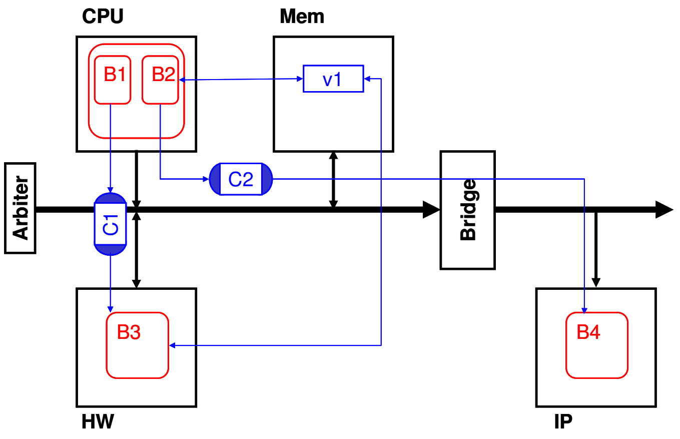


Fig. 5: System specification example.

Figure 5 shows the system specification of the example design as a model that combines the functional aspects of the application model with the structural information of the platform architecture. Note that the two aspects, behavior and structure, are fully complementary (i.e. non-overlapping). This is highlighted in Figure 5 which simply is an overlay of Figure 3 and Figure 4.

D. TLM generation

From the system specification model, our envisioned design environment can then automatically generate a corresponding transaction-level model (TLM).

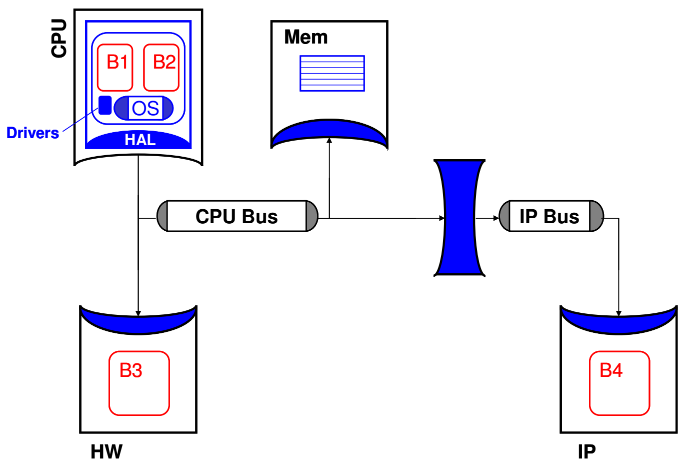


Fig. 6: Generated transaction-level model (TLM).

Figure 6 shows the generated TLM for the simple example defined in Figure 5. In the model, transaction-level communication layers have been inserted to reflect the transactions on the system busses between the components. The busses themselves are represented by TLM channels `CPU Bus` and `IP Bus`. In the software component `CPU`, additional layers of hierarchy have been inserted to accurately reflect the hardware abstraction layer (HAL) of the processor. Also, the functional blocks `B1` and `B2` are now modeled as tasks, being scheduled by an abstract operating system `OS` channel and communicating via integrated `Drivers`. The inserted components stem from template models in the system database which are customized according to the actual design decisions applied by the system designer.

E. Pin- and cycle-accurate model generation

System synthesis tools allow to automatically generate a pin- and cycle-accurate model (P/CAM) that reflects the intended implementation of the system accurately down to the interconnecting pins and wires.

Figure 7 shows the generated P/CAM for the TLM shown in Figure 6. The lower-level communication layers, that were abstracted away in the TLM channels, are now properly modeled as an inner layer that samples and drives the explicit bus wires according to the selected communication protocol and timing.

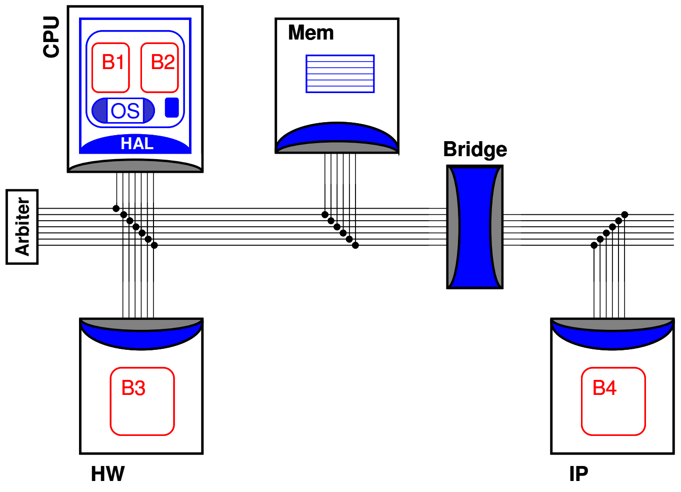


Fig. 7: Generated pin- and cycle-accurate model (P/CAM).

IV. System Environment

The above described models and design flow can be united in a system environment for automatic generation of TLMs. Such an environment includes extensive simulation and analysis engines for detailed feedback about design model behavior and quality metrics. Apart from capturing the system specification and later design decisions, it's graphical user interface (GUI) supports a wide variety of visualizations for simulation and analysis results. This allows the system designer to focus her/his efforts on the critical aspects in the system design flow and exploration, thus arriving at an optimal design implementation in a short amount of time.

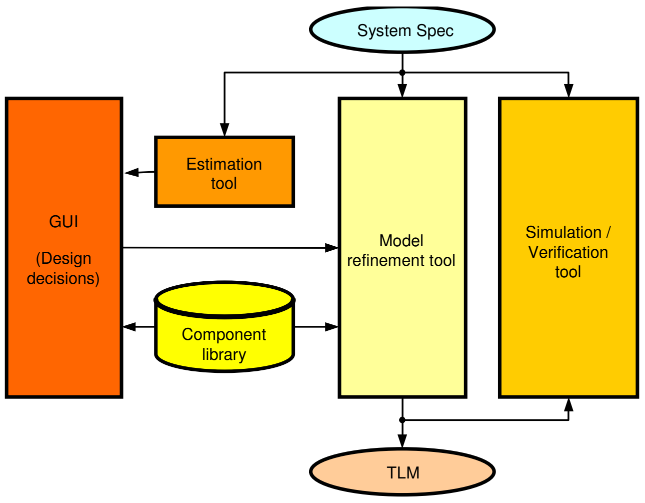


Fig. 8: System design environment.

Figure 8 shows the main components of the environment, all driven and visualized by an easy-to-use GUI. The heart of the environment is the model refinement engine that combines the application model and platform architecture to a system specification model and allows it to be further refined down to a transaction-level or pin-accurate model. The refinement engine is supported by a component library with models and property annotations for processor, hardware, and IP components.

Validation of both input and output models is performed by integrated simulation and verification tools. An estimation tool is also included, allowing early and rapid feedback about the quality metrics of the design at hand.

Such an environment offers the following features:

- Graphical entry of platform target architecture as a netlist of components and busses.
- Graphical entry of system specification as application code consisting of communicating processes.
- Automatic generation of platform transaction-level models (TLMs) for simulation, analysis and verification.
- Extensive platform simulation and analysis through fast and accurate transaction-level simulation.
- Evaluation and exploration of platform quality and behavior through large set of profiling and analysis tools.

A. *ESE Frontend*

In the rest of the paper, we demonstrate and describe the design flow using an example of a MP3 decoder using a prototype tool called Embedded System Environment (ESE) Frontend. ESE Frontend is a tool that simplifies and automates the generation of Transaction Level Models (TLM). Automatic model generation allows designers to move from idea to an executable model in less than one hour. Furthermore, it enables extensive exploration and validation of the computation and communication design space.

The system architecture is defined as a netlist of major system components, including processors, dedicated hardware accelerators, and other processing elements. Independently, the system application is specified hierarchically and concurrently by behavior and channel blocks containing ANSI C code. Together, these two inputs form the system specification that is captured and serves as input to the ESE Frontend.

In addition to the main model refinement engine, ESE Frontend features platform validation with fast simulation and profiling and analysis tools, and platform exploration of architecture alternatives and parameter variations. Both, software and hardware design flows are integrated to allow for an application development in true co-design manner.

As the output, ESE Frontend automatically generates a model at the transaction level that allows to co-simulate the system platform fast and accurately for early and rapid feedback of the design characteristics.

ESE Frontend offers the following advantages over current embedded design flows:

- Freedom from system-level design languages: Graphical entry of block diagrams and hierarchical C code.
- Easier design space exploration: Automatic TLM generation from application code and design decisions.
- Fast verification cycles: TLMs allow fast yet accurate simulation.
- Early validation of design constraints: Profiling and analysis tools provide feedback for evaluation.

V. Design Example

We will now use an MP3 decoder application [3] as example to demonstrate system design using ESE Frontend.

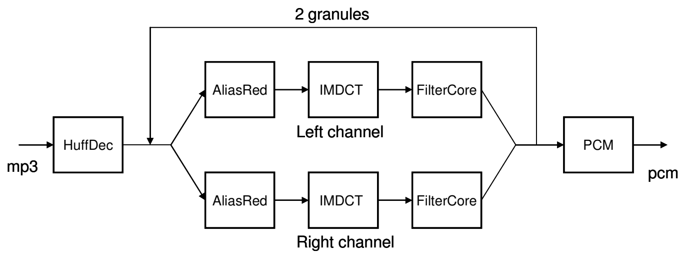
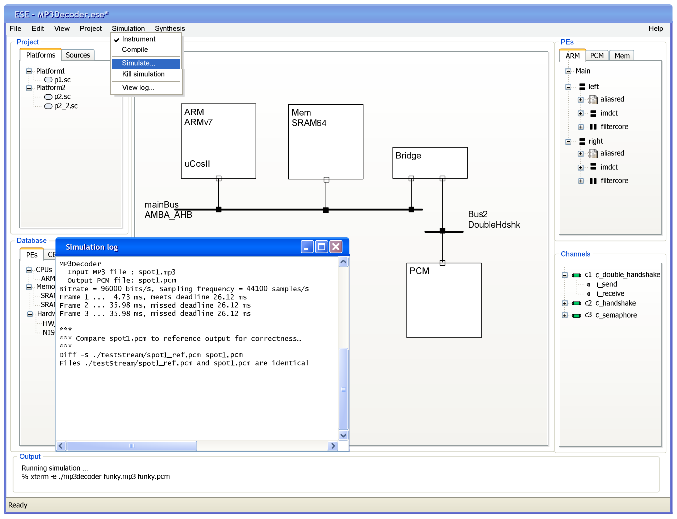


Fig. 9: MP3 decoder example, functional block diagram.

Based on a reference C code, we have captured the functionality of the MP3 decoder in ESE Frontend. Our application model reflects the major functional blocks in the decoder pipeline, as shown in Figure 9. In addition, the application model contains smaller control blocks that handle the input and output of the byte streams, as well as a testbench wrapped around the design such that the functionaly can be validated through simulation.



```
MP3Decoder
  Input MP3 file : spot1.mp3
  Output PCM file: spot1.pcm
Bitrate = 96000 bits/s, Sampling frequency = 44100 samples/s
Frame 1 ...  4.73 ms, meets deadline 26.12 ms
Frame 2 ... 35.98 ms, missed deadline 26.12 ms
Frame 3 ... 35.98 ms, missed deadline 26.12 ms

***
*** Compare spot1.pcm to reference output for correctness…
***
Diff -s ./testStream/spot1_ref.pcm spot1.pcm
Files ./testStream/spot1_ref.pcm and spot1.pcm are identical
```

```
Running simulation ...
% xterm -e ./mp3decoder funky.mp3 funky.pcm
```

Fig. 10: Screenshot of ESE Frontend.

Next, we have captured an initial platform architecture that maps the entire MP3 decoder functionality on an embedded ARM7TDMI processor. Only the PCM output is performed by a dedicated hardware unit that emits the decoded PCM sound samples according to the timing specified in the MP3 stream. The ARM processor and the PCM output unit both have their own local bus, connected together by a bridge unit. Figure 10 shows this platform architecture in a screenshot of ESE Frontend.

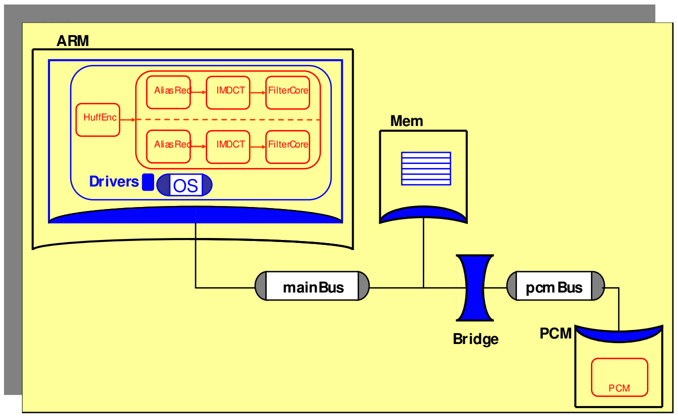


Fig. 11: MP3 decoder example, generated TLM 1.

To evaluate this architecture of the MP3 decoder, we used ESE Frontend to generate a TLM (Figure 11) and simulated the model. The simulation results showed that the ARM processor alone cannot meet the required frame speed of 26.12ms.

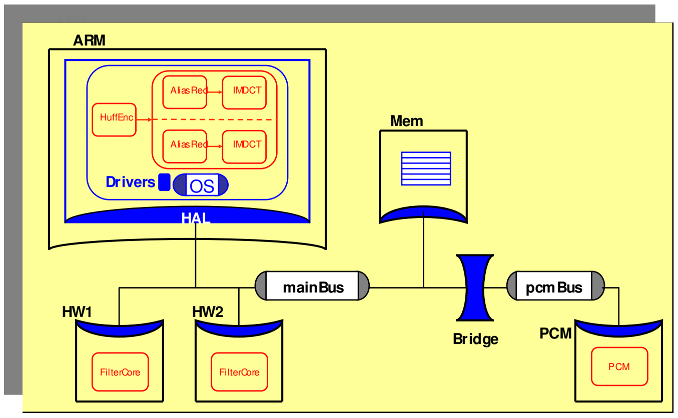


Fig. 12: MP3 decoder example, generated TLM 2.

To speed up the design, we extended our platform architecture by introducing two additional hardware accelerators dedicated to the FilterCore blocks for the left and right audio channel, respectively. The improved model (Figure 12) showed a significant speed improvement, but the frame deadline could still not be met due to high bus contention on the AMBA main bus.

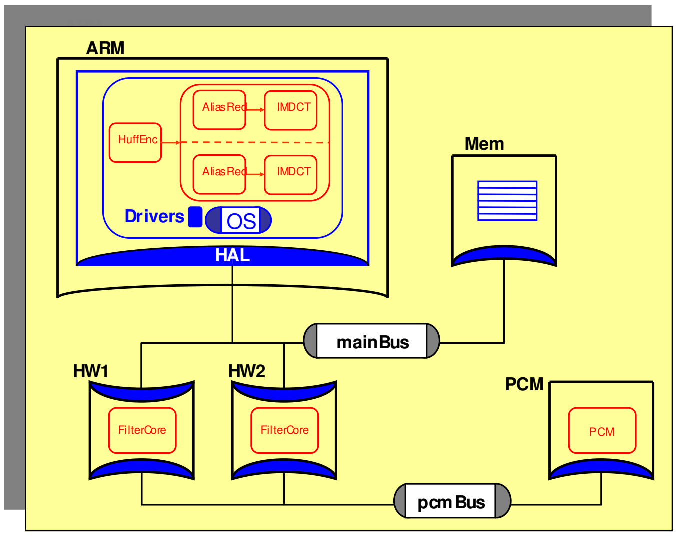


Fig. 13: MP3 decoder example, generated TLM 3.

Again, we adjusted the target platform. We connected the FilterCore units directly to the PCM output unit, eliminating the need for the bus bridge, as shown in Figure 13. This design successfully met the frame delay.

Figure 14 shows some simulation and estimation results obtained for each of the TLM alternatives. The graphs clearly show that only the third design alternative meets the frame delay deadline.

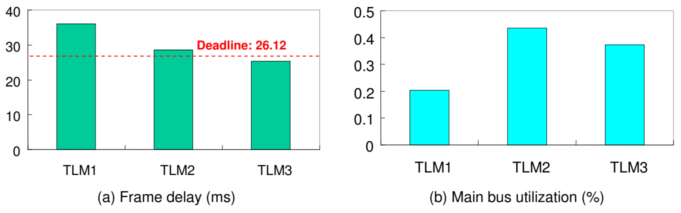


Fig. 14: Estimation results for the generated TLM alternatives.

We would like to emphasize that the entire design exploration for this example can be performed in less than one hour of time. This is possible due to an intuitive GUI that allows easy capturing and modifying of design models, and in particular due to the automatic model generator that creates TLMs within seconds for the selected platform architecture.

In summary, the MP3 design study clearly shows that ESE Frontend enables rapid design space exploration.

VI. Conclusion

In summary, ESE Frontend offers a true system-level design flow with the following benefits:

- Design decisions and models can be easily exchanged in electronic form, providing simplified globally-distributed design.
- Designs can be easily modified and prototyped, providing better market penetration through customization.
- Models and design decisions can be reused, providing easier change and version management.
- Models are automatically generated, providing shorter time to market.
- No need for manual model development, providing 1000x productivity gains.

Acknowledgment

The authors would like to thank the members of the System-on-Chip Environment group in the Center for Embedded Computer Systems at UC Irvine who contributed to this work, especially Andreas Gerstlauer, Junyu Peng, Dongwan Shin, Samar Abdi, and Roger Ang.

References

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

[2] ISO, *Reference Model of Open System Interconnection (OSI)*, 2nd ed., Internation Organization for Standardization (ISO), 1994, iSO/IEC 7498 Standard.

[3] P. Chandraiah and R. Dömer, “Specification and design of an MP3 audio decoder,” Center for Embedded Computer Systems, University of California, Irvine, Tech. Rep. CECS-TR-05-04, May 2005.

[4] D. D. Gajski, J. Zhu, R. Dömer, A. Gerstlauer, and S. Zhao, *SpecC: Specification Language and Design Methodology*. Kluwer Academic Publishers, 2000.

[5] A. Gerstlauer, R. Dömer, J. Peng, and D. D. Gajski, *System Design: A Practical Guide with SpecC*. Kluwer Academic Publishers, 2001.

[6] “SpecC Technology Open Consortium,” http://www.specc.org.

[7] “Open SystemC Initiative,” http://www.systemc.org.