Embedded System Environment

A Framework for TLM-based Design and Prototyping

Samar Abdi[1], Yonghyun Hwang[2], Lochi Yu[3], Hansu Cho[4], Ines Viskic[5], Daniel D. Gajski[6]

Abstract— **This paper presents Embedded System Environment (ESE), which is a comprehensive set of tools for supporting a model-based design methodology for multi-processor embedded systems. It consists of two parts: ESE Front-End and ESE Back-End. ESE Front-End provides automatic generation of SystemC transaction level models (TLMs) from graphical capture of system platform and application C/C++ code. ESE generated TLMs can be used either as virtual platforms for SW development or for fast and early timing estimation of system performance. ESE Back-End provides automatic synthesis from TLM to Cycle Accurate Model (CAM) consisting of RTL interfaces, system SW and prototype ready FPGA project files. ESE generated RTL can be synthesized using standard logic synthesis tools and system SW can be compiled along with application code for target processors. ESE automatically creates Xilinx EDK projects for board prototyping. Our experimental results demonstrate that ESE can drastically reduce embedded system design and prototyping time, while maintaining design quality similar to manual design.**

Keywords-Transaction level model, system synthesis, virtual prototyping, multi-processor embedded systems, MPSoC.

I. INTRODUCTION

Multiprocessor platforms are increasingly being used in embedded system design to deal with growing complexity and performance demands of modern applications. The design goal is to choose the optimal platform for a given application and the optimal mapping of the application to that platform. Such design decisions require early and accurate estimation of performance for a given design choice. Conventional *Cycle Accurate Models* (CAMs) are supported by a wide variety of design automation tools; they provide accurate metric estimation, but are too slow for early decision making.

SystemC [5] *Transaction Level Models* (TLMs) [6] are emerging as the next higher level of modeling abstraction above CAMs. Although TLMs have been extensively used in the industry for early software validation, they have not yet been widely adopted for design. There are primarily two reasons for this trend: TLM semantics for early and accurate design metric estimation are missing; and tools for automatic synthesis of CAMs from TLMs have not been available.

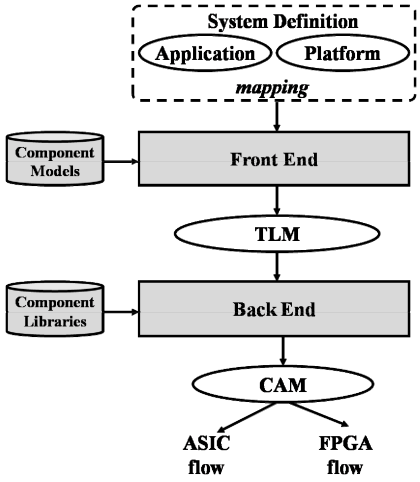

Fig. 1: ESE Design Flow.

We present here the Embedded System Environment (ESE), which is a comprehensive set of model generation and system synthesis tools to support a TLM-based design methodology. ESE consists of two parts, the front-end and the back-end, as shown in Figure 1. The input to front-end is the system definition consisting of an application model mapped to a given platform. It automatically generates a TLM of the system for fast and early design evaluation. The back-end reads this TLM and synthesizes the required software and hardware to produce the cycle-accurate model (CAM). The CAM is the hand-off point to standard FPGA and ASIC design automation tools. Therefore, ESE enables a structured and automated design flow from an abstract specification to an implementation, based on well-defined design decisions.

The rest of the paper is organized as follows. The following section presents the current state of the art academic and commercial tools for system level design. Section III describes the design specification which is used as input to ESE. Section IV delves into the details of the front-end tools for TLM automation and performance estimation. Section V discusses the back-end synthesis tools for embedded software and RTL interfaces. Experimental results pertaining to the front-end and

1. Samar Abdi is with Concordia University, Canada
2. Yonghyun Hwang is with Qualcomm, USA
3. Lochi Yu is with Universidad de Costa Rica, Costa Rica
4. Hansu Cho is with Samsung, Korea
5. Ines Viskic is with Microsoft, USA
6. Daniel D. Gajski is with the University of California, Irvine, USA

978-1-4244-7074-7/10/$26.00 ©2010 IEEE

DOI 10.1109/RSP_2010.SS1

back-end tools are presented in Sections IV and V, respectively. Finally, we wrap up with conclusions on the benefits and advantages of ESE and future outlook.

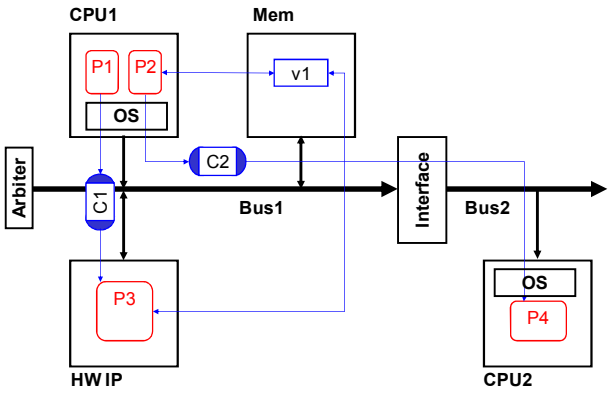

Fig. 2. ESE specification as a mapping from application to platform.

II. Related Work

Several academic and commercial tools have been developed around system level design methodologies in the past decade [1]. ESE has been developed at the University of California, Irvine, as a successor to the System-on-Chip Environment (SCE) [4], which enables a specify-explore-refine design methodology for system implementation from applications written in SpecC [2]. Metropolis [20] is a well known modeling and simulation environment from UC Berkeley, based on the *Platform-Based Design* (PBD) paradigm, which calls for the separation of function and architecture during design. In PDB, the system design problem is reduced to the mapping of a desired function onto the given target platform to create a specific design instance. ESE follows the PBD principle for design entry but also provides model generation, estimation and synthesis, besides simulation.

A class of system level design tools has been developed specifically for streaming multimedia applications. SystemCoDesigner [22] supports system design for streaming applications written in a dynamic dataflow-oriented model of computation. The model can be transformed into C code for software implementation or synthesized using Forte's Cynthesizer tool [25] for hardware implementation. Daedalus [23] supports applications specified in the *Kahn Process Network* (KPN) model of computation and provides tools for software-hardware design, similar to SystemCoDesigner. PeaCE (Ptolemy extension as a Codesign Environment) [21] is yet another hardware/software co-design framework that supports applications modeled in extended synchronous data flow using Ptolemy [19].

Several commercial tools for system level design have recently been introduced by the design automation industry. CoFluent Studio [15] is a graphical frontend for SystemC; it captures application functionality, system architecture, and their mapping. SpaceStudio [17] provides a SystemC-based system level integrated development environment built on top of Eclipse to create process-based SystemC application models and TLMs out of pre-defined library blocks or by importing and wrapping existing C, C++, or SystemC code. CoWare provides a frontend for SystemC TLM capture, modeling, and simulation [16]. Carbon's SoC Designer [14] integrates cycle-accurate hardware, ISS, and bus models for fast cycle-accurate system simulation. VaST [13] and Virtutech [24] provide virtual prototyping tools that use binary translation for software simulation, but do not guarantee timing accuracy.

In contrast to the above tools, ESE supports a concurrent C-based imperative model for application specification, which can support both control flow and data flow oriented applications. It supports automation of both TLM and CAM and raises the level of design abstraction to **enable application developers to design systems** without the need to make early distinction between software and hardware.

III. Design Specification

The input to the TLM-based design process is the system specification consisting of the application mapped to a multi-core platform as shown in Figure 2. The application, platform, and mapping entry in ESE are simplified by an intuitive *Graphical User Interface* (GUI) [3].

The application is captured as a set of concurrent communicating processes. The processes at the leaf level are symbolic representations of functions that may be specified using common programming languages such as C and C++. Even legacy code (usually available in C) can be inserted in leaf-level processes. Processes communicate to other processes and memories using well-defined ports. The ports provide communication functions that are implemented in abstract platform independent channels. The process ports are bound to channel instances in order to define the application level communication between processes. Communication channels enable a clear separation computation from communication. These channels provide a rich set of user-level communication mechanisms, such as handshake, FIFO, and asynchronous read/write.

The hardware platform is composed in the GUI from a set of processing elements (PEs), busses, and interface components called *transducers*. Figure 2 shows a typical embedded multi-processor platform consisting of multiple CPUs (*CPU1* and *CPU2*), a hardware accelerator (*HW*), and memory (*Mem*). The communication architecture of the platform consists of two busses (*Bus1* and *Bus2*) connected by an interface component. During platform definition, components and connections are instantiated from a given library. In general, these components can be embedded processors, memories, custom hardware units, or third party IPs. For communication, the designer may use system level interconnects such as shared busses (with centralized or distributed arbitration), bridges, serial links, or network on chip. The software platform is defined by configuring the software parameters of the processing elements. These configurations include the *Operating System* (OS) definition, task scheduling policy, and memory management. The platform architecture can be completely or partially defined and more components and connections can be added at a later stage.

A mapping from application to platform may also be

defined graphically in ESE. The C/C++ processes are mapped to PEs. Channels are mapped to busses or routes in the hardware platform. Figure 2 shows the mapping as an overlay of the application model on the platform.

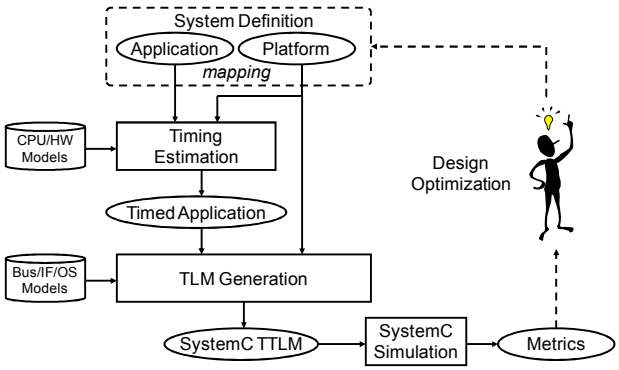

Fig. 3. TLM automation in ESE Front-End.

IV. ESE FRONT-END

The goal of ESE front-end is to enable early design space exploration by automatically generating fast and accurate TLMs from the system specification. The TLM generation process is shown in Figure 3. The basic idea is to automatically generate a high speed TLM that can be simulated to obtain metrics about the design; these metrics may be performance, power, reliability, security and so on. If the metrics are not satisfactory, the designer may change the application model, the platform, or the mapping decisions.

A practical design space exploration flow requires the capability to generate TLMs quickly. Therefore, manually coding TLMs is not an option. TLMs must also provide reliable metrics. The metric estimation supported by ESE-generated TLMs is timing. Timing is annotated inside the TLM such that TLM simulation can predict timing for any input. ESE uses a retarget-able technique to automatically annotate cycle approximate timing to the TLM.

Data models of the PEs, busses, and OS are used for timing annotation. The PE data model includes the data path and the memory hierarchy information of the PE. Therefore, it includes the number and type of architectural components and the size and configuration of caches. The bus model defines the bus transaction delays for various bus modes such as word, burst, or pipelined transfer. The OS data model defines policies and timing delays for performing dynamic scheduling of processes, as well as the delays for inter-process communication.

The TLM generation occurs in two steps as shown in Figure 2. The first step is the timed application generation, in which the application process code is instrumented with delay annotations and instantiated inside PE models to create the timed application model. In the second step, executable models of the busses, transducers, and OSes are instantiated and linked with the timed application model. The OS model is used to capture resource contention and dynamic scheduling of processes mapped to the same PE. The abstract channel communication between the processes is transformed into a sequence of bus transactions, based on the mapping of channels to busses and routes. The final result of the above steps is the *Timed TLM* (TTLM) in SystemC. The SystemC TLLM can be compiled and simulated natively on the host machine to obtain timing metrics. These metrics can then be used for design optimization.

A. Timed Application Generation

The timing annotation of application code can be done at various levels of accuracy by considering different features of the PE, such as operation scheduling policy, cache size, and replacement policy. The PE data model is a set of these parameter values. While generating the timed TLM, each basic block in the application is analyzed to compute the estimated number of cycles needed to execute it on the given PE. The number and combination of parameters used to model the PE, determine the accuracy of the estimation. Therefore, several timed TLMs are possible depending on the detail of PE modeling. The more detailed the PE model, the longer is the delay computation time. A tradeoff is needed to determine the optimal abstraction of PE modeling. In our work, we have considered operation scheduling policy, datapath structure, cache configuration and branch delay as the most important parameters for PE modeling.

The computation timing can be estimated and annotated automatically into application tasks during TLM generation [8]. A retarget-able and reusable PE data model is created by specifying the operation scheduling policy, data path structure, cache hit-rate, and branch delay. The application code in each task is converted into a target-independent control-data flow graph (CDFG) representation using the LLVM compiler infrastructure [12]. Then the PE data model is used to schedule the execution of each basic block of task code on the PE to which the task has been mapped. This early scheduling provides an estimated operation execution delay for each basic block in the process. A stochastic memory delay model and the PE cache model are used to determine the delays of fetching the operations and loading/storing the operands in memory. Similarly, a stochastic branch prediction model is used to compute the delay resulting from branch mis-prediction. The basic blocks are then annotated with the sum of operation, memory, and branch delays to produce a timed process model. The timed processes are instantiated inside the SystemC modules of the PE to which they have been mapped, resulting in the timed application model.

B. TTLM Generation

There are three steps in TTLM generation. First, a model of the platform must be generated based on the specification provided in the GUI. Second, the application level channels described in Section III must be refined for implementation on the platform model. Finally, the timed application model, along with the refined channels, must be linked with the platform model to produce the final SystemC TTLM. For generating the platform TLM, we define a SystemC template for each platform object [7]. Busses are modeled as *sc_channels* that provide synchronization and memory access methods; transducers are modeled as *sc_modules* and include

logic for protocol conversion and data buffering; OSes are modeled as *sc_modules* and provide a subset of standard POSIX methods [11]. The platform is modeled as a top level *sc_module* that connects the various platform objects.

We define a *Universal Bus Channel* (UBC) template for abstracting the system bus as a single unit of communication. UBC provides the basic communication services of synchronization, arbitration and data transfer that are part of a transaction. In the generated TLM, we do not distinguish between different bus protocols. UBC provides 5 bus communication functions namely: *Send/Recv* for synchronized communication, *Read/Write* for memory access and *MemoryService* for memory control. Synchronization is required for two processes to exchange data reliably. Synchronization between two processes takes place by one process setting a flag in the UBC and the other process checking and resetting the flag. We will refer to the process setting the flag as the *slave* and the process resetting the flag as *master*. Since a bus is a shared resource, multiple transactions attempted at the same time must be ordered sequentially. This arbitration is modeled in the UBC using the SystemC *sc_mutex* class. After arbitration, the master process sets the *address* of the transaction in the UBC to notify the slave of the beginning of the transaction.

If the communicating processes are mapped to PEs that do not share the bus, we must define transducer components in the platform to act as bus interfaces. The transducer facilitates multi-hop transactions, where one process sends data to another process over multiple busses. The basic functionality of the transducer is to simply receive data from the sender process, store it locally and send it to the receiver process once the latter becomes ready. The transducer model, therefore, consists of a *sc_thread* that waits for transactions from the sender over the UBC, a local buffer (modeled using *sc_fifo*) to store the data, and another thread to forward the data in the buffer to the recipient process over the UBC.

The OS model is divided into two separate models: OS behavior model and OS overhead model [11]. The OS behavior model captures fundamental OS services, such as task management, event handling for synchronization, scheduling, time modeling, and interrupt handling. In addition to these services, it supports abstract channels and kernel system calls to provide communication among tasks. The OS overhead model has processor and OS-specific pre-characterized overhead information to provide cycle approximate estimation.

The application level channels are refined based on the mapping of the application channels to routes in the platform. During refinement, the signature of the application channel methods is not modified. Instead, the implementation of the channels is transformed by adding code for data packeting and routing. The application level channels model transactions of typed data of potentially unlimited size. However, once these channels are mapped, the data must be broken down into fixed size packets based on the bandwidth of the route. The route determines the specific UBCs used by the sender process and the receiver process to transfer the data. At the end of TTLM generation, we compile the UBCs, transducer modules, memory modules, OS models, refined channels and the timed application processes to create the executable TLM binary.

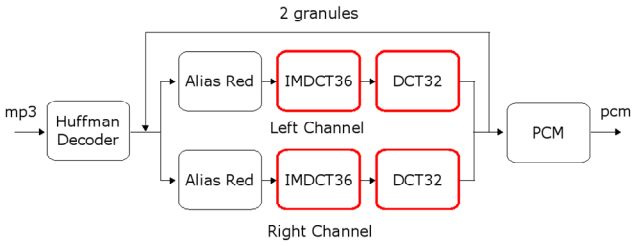

Fig. 4. MP3 decoder application.

C. *Front-End Results*

We demonstrate the benefits of ESE-generated TLMs using the MP3 decoder application. The MP3 decoder reference model consisted of 12K lines of C code, with the DCT32 and IMDCT36 function being the most compute intensive as highlighted in Figure 4. We developed four heterogeneous platform architectures for MP3 decoder; these will be referred to as SW+0, SW+1, SW+2 and SW+4 for simplicity. SW+0 is a pure SW implementation on the Microblaze processor from Xilinx. SW+1 implements the left channel DCT in custom HW and the rest on Microblaze. SW+2 implements both left channel DCT and IMDCT in separate custom hardware components and the rest on Microblaze. Finally, SW+4 implements both left and right DCT and IMDCT in custom HW, with the rest on Microblaze.

Design	Timed TLM Generation	Timed TLM Simulation	ISM Sim.	CA Sim.
SW+0	31 s	1 s	3.6 h	16 h
SW+1	50 s	22 s	N/A	18 h
SW+2	47 s	25 s		18 h
SW+4	71 s	36 s		18 h
Average	**~ 1min**	**~ 20 s**	**3.6 h**	**~ 18 h**

Table 1. Generation and simulation times for MP3 TLMs.

SystemC TLMs for the MP3 designs were automatically generated using ESE. The TLM generation and simulation results for the MP3 decoder are shown in Table 1. As we can see, for all the design choices, the executable TTLMs were generated in a few seconds. The generation time includes the time to compile the TTLM on the host machine. The TTLM simulation time for a given sample of MP3 data is also in the order of few seconds.

In contrast, the instruction set model (ISM) simulation time of the SW+0 design is 3.6 hours. The ISM and RTL models for other design choices were not developed, but their simulation is expected to take even longer as indicated by the cycle-accurate model (CAM) simulation results. The full system RTL simulation using CAMs took between 16 to 18 hours. Therefore, for any reasonable design space exploration, it is much more productive to use automatically generated TLMs as opposed to ISMs or CAMs.

ISM Error

Cache size	SW+0	
	Board	ISM Error
0K/0K	27.2M	39.48%
2K/2K	8.9M	18.38%
8K/4K	5.8M	3.55%
16K/16K	4.4M	-16.32%
32K/16K	4.3M	-16.60%
Average	N/A	18.86%

TLM Estimation Error

Cache Size	SW+0	SW+1	SW+2	SW+4
0K/0K	6.27%	9.00%	18.18%	18.61%
2K/2K	6.68%	-7.16%	-15.79%	-9.35%
8K/4K	4.74%	9.13%	-1.66%	-0.18%
16K/16K	-13.83%	4.66%	2.63%	3.65%
32K/16K	-13.89%	-8.29%	1.57%	2.29%
Average	9.08%	7.65%	7.97%	6.82%

Table 2. Estimation accuracy of MP3 TLMs.

To measure the accuracy of the ESE-generated TLMs, we built FPGA board prototypes of all the designs. These board designs were used as reference points for measuring the accuracy of the simulation models. As shown in Table 2, different cache configurations (i-cache/d-cache for the Microblaze processor were used. The TLMs estimation error was almost half of the error in ISMs on average and less than 19% in the worst case.

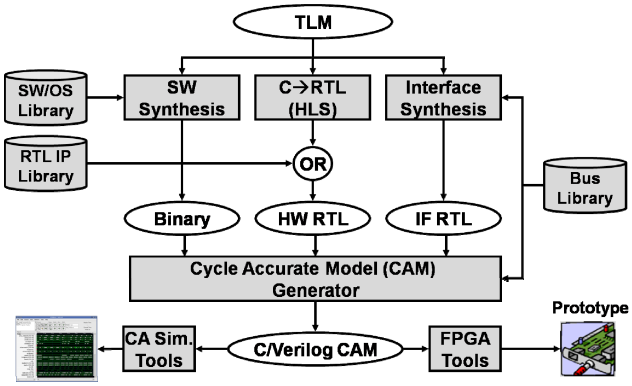

Fig. 5. ESE Back-End prototyping flow.

V. ESE BACK-END

After the design optimization steps are completed, a satisfactory design is obtained at the system level. However, this design is still in the form of a TLM, which is not yet ready for prototyping with standard EDA tools. The TLM must be transformed into the aforementioned CAM for hand-off to ASIC and FPGA implementation tools. The synthesis of the CAM from TLM is supported by three modules within the ESE back-end, as shown in Figure 5.

The software synthesis module produces the processor specific C/C++ code for software implementation. The application code is imported as is from the TLM. If an OS is present, the OS model is replaced with the actual OS kernel library. The key part of SW synthesis is the generation of communication layers from the abstract channels in the TLM. The generated software codes are cross-compiled for the target embedded processors.

For hardware implementation, the RTL code for the specific hardware PE must be generated. If a RTL model of the PE is already available, the SystemC model of the PE in the TLM is simply replaced with this RTL model. If a RTL implementation is not available, ESE generates a SystemC model ready for high-level synthesis (HLS) using Forte Cynthesizer.

The final step in CAM generation is the generation of RTL communication structure of the system. The bus protocol library is used to instantiate the bus controllers for all the busses in the system. The RTL description of all the transducers is also generated automatically based on the mapping of channels to routes in the platform. Interrupt controllers are also instantiated and configured, if needed. The CAM produced by the ESE back-end consists of binary code for all the software PEs in the system and RTL Verilog code for all the hardware PEs, busses, and transducers. The CAM may be simulated using standard Verilog simulators or synthesized using logic synthesis tools. ESE also exports the CAM as a project to Embedded Development Kit (EDK) from Xilinx [18] for FPGA prototyping.

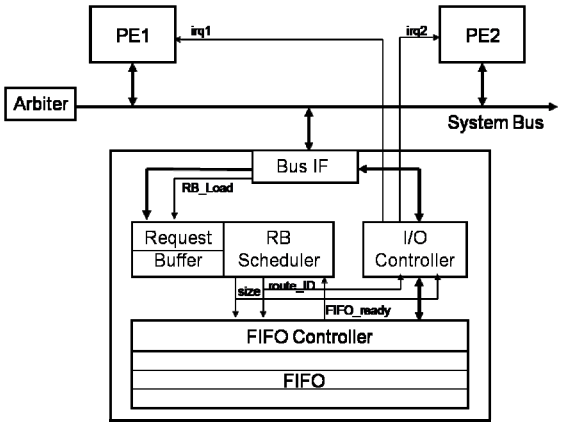

Fig. 6. Transducer RTL generated by ESE Back-End.

A. *Transducer RTL Generation*

The transducer consists of a First-In-First-Out (FIFO) buffer along with dedicated controllers for each bus connection as shown in Figure 6. *BusIF* is a controller which implements the cycle accurate bus protocol, and is instantiated from the bus library. It interfaces the system bus to the internal logic of the transducer, as implemented using the *Request Buffer* and *I/O Controller*. The *Request Buffer* is a register file indexed by the routing paths that include this transducer. The communicating PEs write send/receive *request,* containing the message size, to a dedicated register for the chosen route. Once the transducer is ready to process the PE request, the I/O controller sends an interrupt signal to the PE. The I/O controller then reads/writes the message using the *BusIF* before writing/reading it to the FIFO. Similar to the request buffer, the FIFO is also partitioned by route and stores transaction data before forwarding it to the next PE/transducer in the route. Automatic generation of the transducer RTL logic is straightforward based on the above template and semantics [10].

B. *Generation of Communication Software*

During communication software generation, the abstract UBC calls of the TLM are transformed into equivalent C code, which is specific to the PE, the platform, and the channel mapping. A set of communication functions for routing, packeting, synchronization and transfer are generated. These functions are specific to port interfaces of the application

processes. An example in Figure 7 shows the typical code generated for a *Send* method in interface *i*.

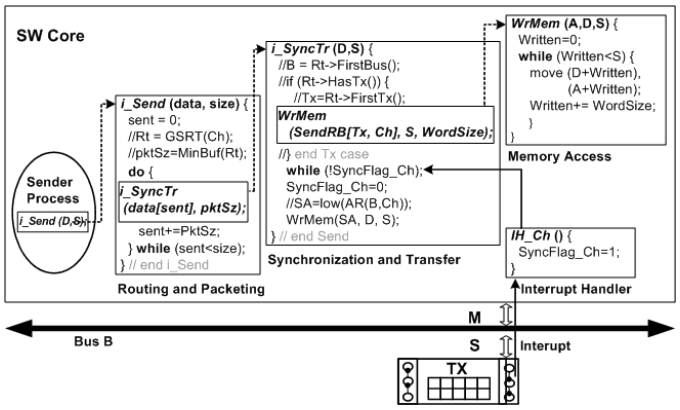

Fig. 7. Communication SW stack generated by ESE Back-End.

In order to automate communication code generation, we define a set of communication specific system parameters [9]. We define a *Global Static Routing Table* (GSRT) that stores the mapping of each application level channel to a platform route. For each channel *Ch*, routed through a transducer *Tx*, we define *Buffer-Size(Tx, Ch)* to be the buffer partition in bytes for *Ch* on *Tx*. We also define the transducer addresses for send and receive request buffers per channel as *SendRB(Tx, Ch)* and *RecvRB(Tx, Ch)*, respectively. The packet size for *Ch* is, therefore, the minimum of *Buffer-Size(Tx, Ch),* for all *Tx* in the route of *Ch*.

ESE supports synchronization using interrupts and polling. A synchronization flag table is maintained for each PE. Each channel *Ch* gets a unique entry *SyncFlag_Ch* in this table. For interrupt based synchronization, we also define a binding from the interrupt source to the flag and the handler instance. For polling, the flag is bound to an address in the slave PE. Finally, for the data transfer implementation, we define the bus word size and the low to high address range for each channel *Ch* on bus *B* as *AR(B, Ch)*. For each software PE we also define *WordSize* to be the number of bytes per word. The code generated for the interface communication method is a do-while loop, with a temporary variable to keep track of already sent/received data. A lower level method *i_SyncTr* is called by the routing/packeting layer to synchronize with the corresponding process and send or receive each packet.

The routing of channel *Ch* determines the synchronization code generated inside the *i_SyncTr* method. Given the route object *Rt*, obtained from the GSRT, we determine the first bus *B* in *Rt*. We also determine if *Rt* contains any transducers. If so, we assign *Tx* to be the first transducer in *Rt*. The first step of packet synchronization is to make a transducer request for the transaction. This is done by generating code to write the packet size (in bytes) into the request buffer at the address given by the parameter SendRB(Tx, Ch) or RecvRB(Tx, Ch), depending on the transaction type. Once the request is written, the transducer initiates lower level synchronization via interrupt or polling, just like any other slave PE.

Lower level synchronization is implemented by generating code for using flag *SyncFlag_Ch* in the *i SyncTr* method. In case of interrupt synchronization, the flag is set by the associated interrupt handler. If the flag is not available, the process is suspended until the next interrupt. In case of polling, the flag is periodically read from the corresponding slave PE. Finally, data transfer is performed by generating a call to the PE-specific *WrMem* or *RdMem* functions. These functions write or read data of given bytes using bus transactions of size *WordSize*. The starting address of the transfer is obtained from the address range *AR(B,Ch)*.

Design	Code Size (LOC)		Comm. Delay (ms)	
	Manual	ESE	Manual	ESE
SW + 1	162	168	35.45	35.18
SW + 2	192	208	70.89	70.04
SW + 4	252	288	140.36	139.40

Table 3. Comparison of manually developed vs. ESE-generated SW.

C. *Back-End Results*

Table 3 shows the quality of the communication SW generated by ESE as compared to the manually designed SW quality. The MP3 designs described in Section IV-C are reused. The total code size for implementing the abstract communication channels in ESE was only marginally larger (<4%) than the manually written SW. On the other hand, ESE generated SW executed marginally faster (6% - 9%) than manual SW. Therefore, we can conclude that ESE generated code is comparable in quality to manual code.

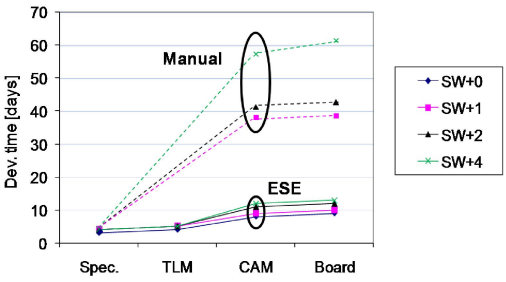

Fig. 8. Design productivity gains using ESE Back-End.

However, automatic CAM generation by ESE Back-End leads to significant productivity gains in system prototyping. Figure 8 illustrates the productivity gain in development time as a result of using ESE. For MP3 platforms with HW components, the CAM development time was in the order of several weeks. As a result, board prototypes for these designs took between 40 to 60 days. ESE drastically cuts prototype development time by automatically generating TLMs and CAMs. With ESE, the final board prototypes for MP3 designs were available in less than a week after the specification model was finalized. Consequently, ESE results in significant savings in design cost and shorter development cycles.

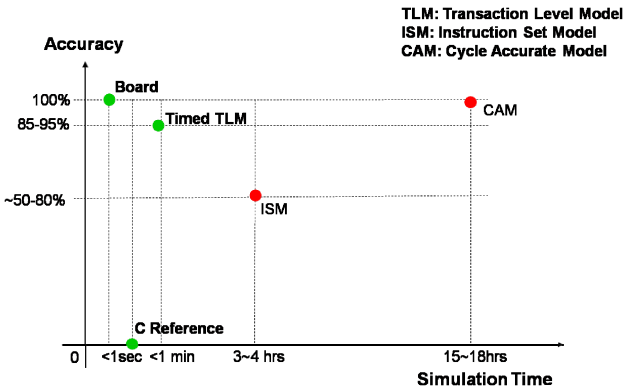

Fig. 9. Comparison of TLMs with conventional models.

Figure 9 illustrates the savings in validation time resulting from a TLM-based design methodology supported by ESE. Each CAM simulation of the MP3 designs took 15 to 18 hours, which makes them impractical for design space exploration. Although at-speed board testing is faster than TLM simulation, bugs found in board testing are typically difficult to trace back to the specification. ESE-generated TLMs execute at speeds close to reference C simulation with high accuracy, thereby making them ideal for early system validation.

VI. CONCLUSIONS

We presented Embedded System Environment (ESE), which is a toolset for early TLM-based design and rapid system prototyping from TLMs. ESE-generated TLMs were shown to be both fast and accurate. Therefore, ESE reduces validation time from an order of several hours or even days to a few seconds. As a result, designers can use ESE to make platform and application optimizations at a higher level, automatically generate TLMs, and verify the optimizations in a few seconds. ESE also automatically generates CAMs, which can be exported to FPGA design tools, leading to board prototyping in a matter of hours as opposed to weeks or months of manual prototyping. In the future, we plan to support more types of platforms, and more MoCs for application specification.

ACKNOWLEDGMENT

The work described in this paper was done while the authors were affiliated with the Center for Embedded Computer Systems (CECS) at University of California, Irvine. We thank the following: Quoc-Viet Dang for developing the ESE GUI, Gunar Schirner for help with OS modeling and ESE release, Pramod Chandraiah for providing the MP3 application model, Andreas Gerstlauer for the survey of system design tools, and CECS for supporting the ESE project.

REFERENCES

[1] Daniel D. Gajski, Samar Abdi, Andreas Gerstlauer, and Gunar Schirner. *Embedded System Design: Modeling, Synthesis and Verification*. Springer, August 2009.

[2] Daniel D. Gajski, Jianwen Zhu, Rainer Doemer, Andreas Gerstlauer, and Shuqing Zhao. *SpecC: Specification Language and Methodology*. Kluwer Academic Publishers, March 2000.

[3] ESE: Embedded Systems Environment, UC Irvine, available online at http://www.cecs.uci.edu/~ese.

[4] Rainer Doemer, Andreas Gerstlauer, Junyu Peng, Dongwan Shin, Lukai Cai, Haobo Yu, Samar Abdi, and Daniel D. Gajski. System-on-Chip Environment: A SpecC-based Framework for Heterogeneous MPSoC Design. EURASIP Journal on Embedded Systems (JES), 2008(647953):13, 2008.

[5] Open SystemC Initiative (OSCI). http://www.systemc.org/.

[6] Lukai Cai and Daniel D. Gajski. Transaction level modeling: An overview. CODES+ISSS 2003.

[7] Lucky Lo Chi Yu Lo and Samar Abdi. Automatic SystemC TLM generation for custom communication platforms. In International Conference on Computer Design, pp 41-46, 2007.

[8] Y. Hwang, S. Abdi, and D. Gajski. Cycle-Approximate Retargettable Performance Estimation at the Transaction Level. In DATE, March 2008.

[9] Abdi, S., Schirner, G., Viskic, I., Cho, H., Hwang, Y., Yu, L., and Gajski, D. Hardware-dependent software synthesis for many-core embedded systems. ASPDAC 2009

[10] H. Cho, S. Abdi, and D. Gajski. Interface synthesis for heterogeneous multi-core systems from transaction level models. LCTES 2007

[11] Y. Hwang, G. Schirner, S. Abdi. Automatic Generation of Cycle-Approximate TLMs with Timed RTOS Model Support. International Embedded Systems Symposium,2009

[12] LLVM(Low Level Virtual Machine) Compiler Infrastructure Project. .http://www.llvm.org.

[13] VaST:.http://www.vastsystems.com/

[14] Carbon SoC Designer. http://www.carbondesignsystems.com/.

[15] CoFluent Design. CoFluent Studio. http://www.cofluentdesign.com/.

[16] CoWare. http://www.coware.com/.

[17] Space Codesign Systems. http://www.spacecodesign.com/.

[18] Xilinx. *Embedded System Tools Reference Manual*. 2005.

[19] Joseph Buck, Soonhoi Ha, Edward A. Lee, and David G. Messerschmitt. Ptolemy: A framework for simulating and prototyping heterogeneous systems. International Journal of Computer Simulation, Special Issue on Simulation Software Development, 4:155–182, April 1994.

[20] Felice Balarin, Harry Hsieh, Luciano Lavagno, Claudio Passerone, Alessandro Pinto, Alberto Sangiovanni-Vincentelli, Yosinori Watanabe, and Guang Yang. Metropolis: A design environment for heterogeneous systems. In Wayne Wolf and Ahmed Jerraya, editors, Multiprocessor Systems-on-Chips. Morgan Kaufmann, 2004.

[21] Soonhoi Ha, Sungchan Kim, Choonseung Lee, Youngmin Yi, Seongnam Kwon, and Young-Pyo Joo. PeaCE: A hardware-software codesign environment of multimedia embedded systems. ACM Transactions on Design Automation of Electronic Systems (TODAES), 12(3):1–25, 2007.

[22] Joachim Keinert, Martin Struebeuhr, Thomas Schlichter, Joachim Falk, Jens Gladigau, Christian Haubelt, Juergen Teich, and Mike Meredith. SystemCoDesigner - an automatic ESL synthesis approach by design space exploration and behavioral synthesis for streaming applications. ACM Transactions on Design Automation of Electronic Systems (TODAES), 14(1):1–23, 2009.

[23] Hristo Nikolov, Mark Thompson, Todor Stefanov, Andy D. Pimentel, Simon Polstra, Ranjan Bose, Claudiu Zissulescu, and Ed F. Deprettere. Daedalus: Toward composable multimedia MP-SoC design. In Proc. of the ACM/IEEE Int. Design Automation Conference (DAC '08), pages 574–579, June 2008.

[24] Virtutech. Virtutech Simics. http://www.virtutech.com/.

[25] Forte Design Systems. Cynthesizer. http://www.forteds.com