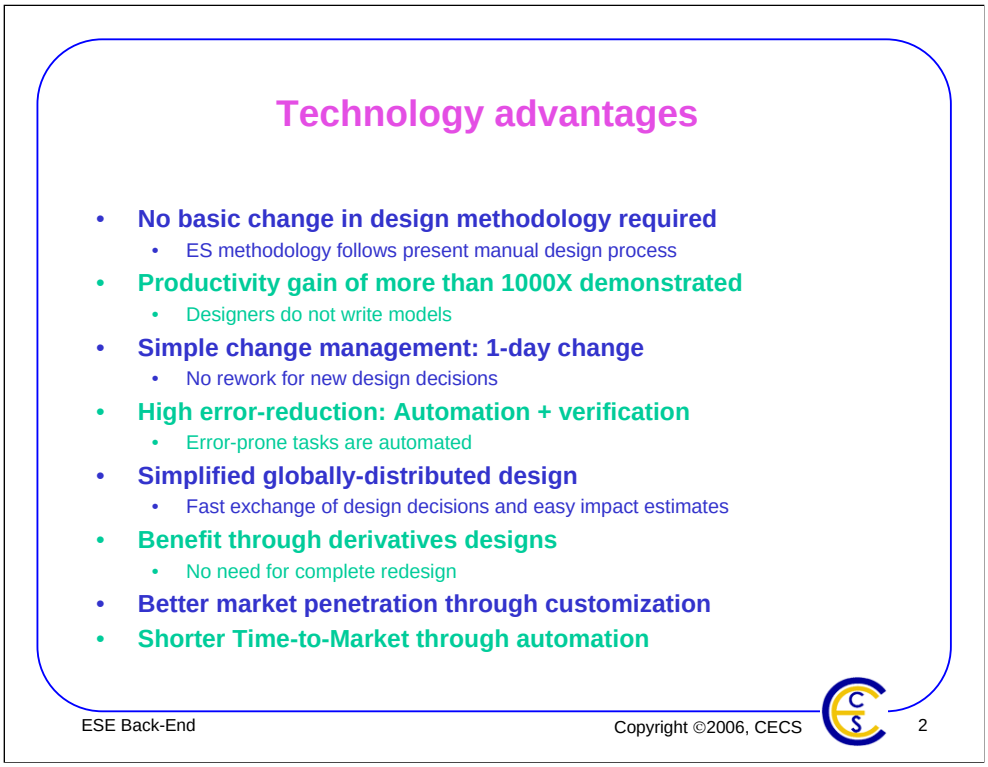
ESE Back End 2.0

D. Gajski, S. Abdi
(with contributions from H. Cho, D. Shin, A. Gerstlauer)
Center for Embedded Computer Systems
University of California, Irvine

http://www.cecs.uci.edu

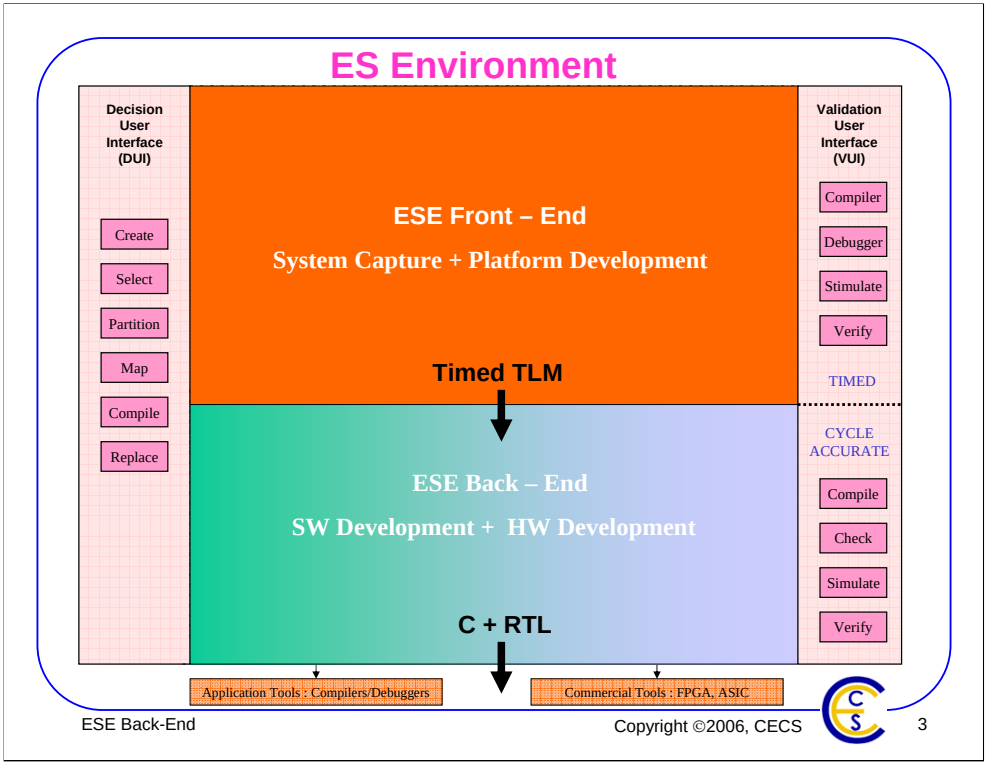


Technology Advantages

This new ESE methodology does not require any changes in the present corporate methodology and offers three orders of magnitude of productivity gain because of automatic model generation, synthesis and verification.. It reduces bugs since the mundane tasks of generating models and verifying them is automatic.

It also allows simple change management of few hours for small changes and few days for large changes. Since all the models and changes are made automatically it is easy to ship those models around the world for design, checking and upgrades.

However, the main advantage lies in the fact that every system or product can be easily upgraded with only few days of work. This type of customization allows better market penetration and shorter time-to-market.



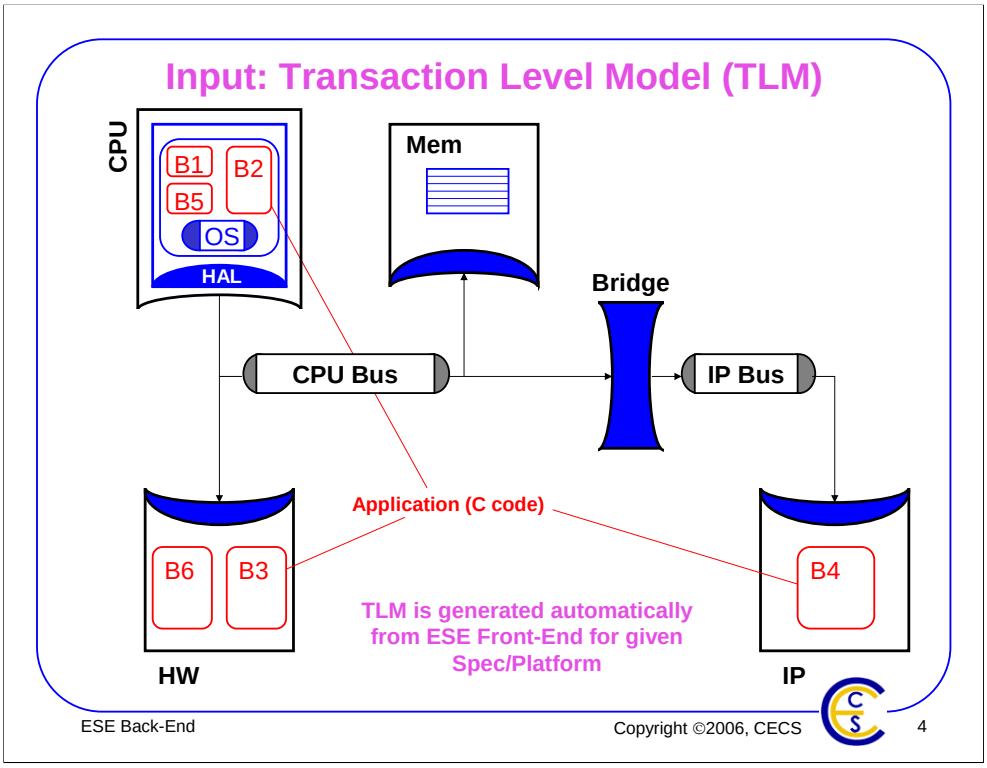
ES Environment (ESE)

The ESE consists of a **front-end** and a **back-end** supported by two interfaces.

The front-end consists of **System Capture**, which is a graphical user interface for capturing the definition of the platform architecture and product application code. **Platform Development** tool generates timed Transaction-Level Models (TLMs) of the platform architecture executing the product application defined by the capture tool. These timed TLMs provide reliable performance metrics and are used for early exploration of design choices.

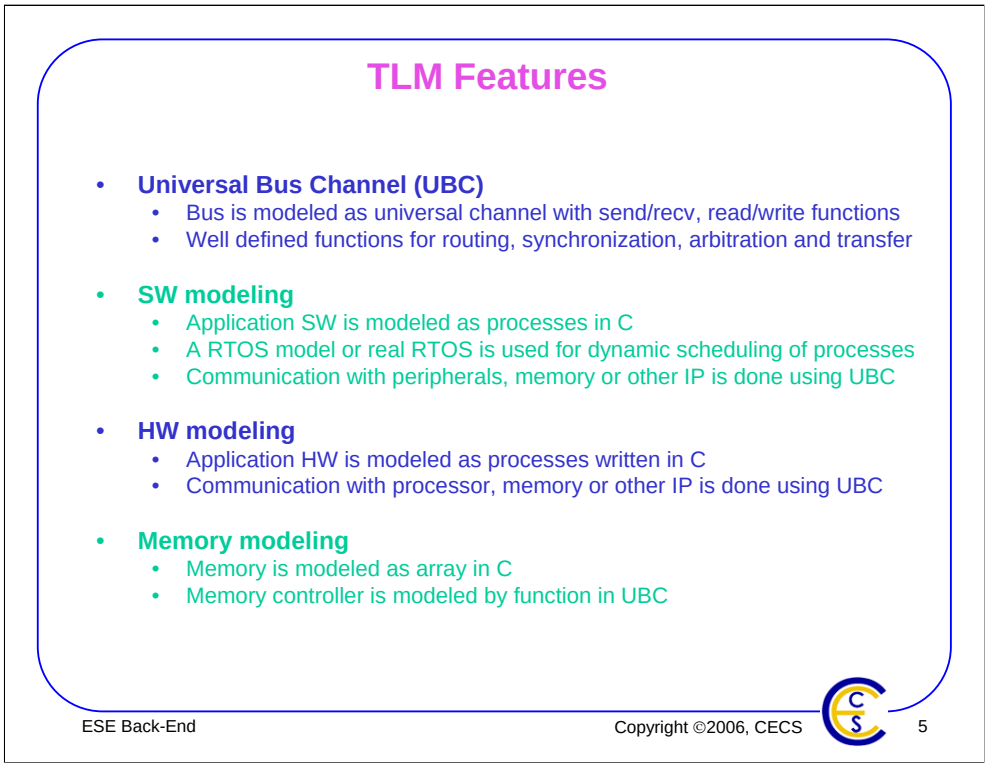
In the back-end, the **HW Development** component is used to generate cycle-accurate or RTL description of the HW components which can be further refined by commercially available tools for ASIC or FPGA manufacturing. **SW Development** generate firmware necessary to run communication and application SW on the platform.

Validation User Interface is used to debug and validate developed SW and HW. **Decision User Interface** is used by the designer, to estimate the quality metrics and make decisions such as component selection, task scheduling, mapping of SW functions to HW components and others.



Transaction Level Model (TLM)

The ESE front-end automatically generates the transaction level model of the system for debugging and validation of SW and reference C code for HW. This model is also used for application SW development as well as for design of custom HW and interfaces. This way SW and HW can be developed concurrently. It simulates very fast so that productivity of developers increases by an order of magnitude (from days to hours).



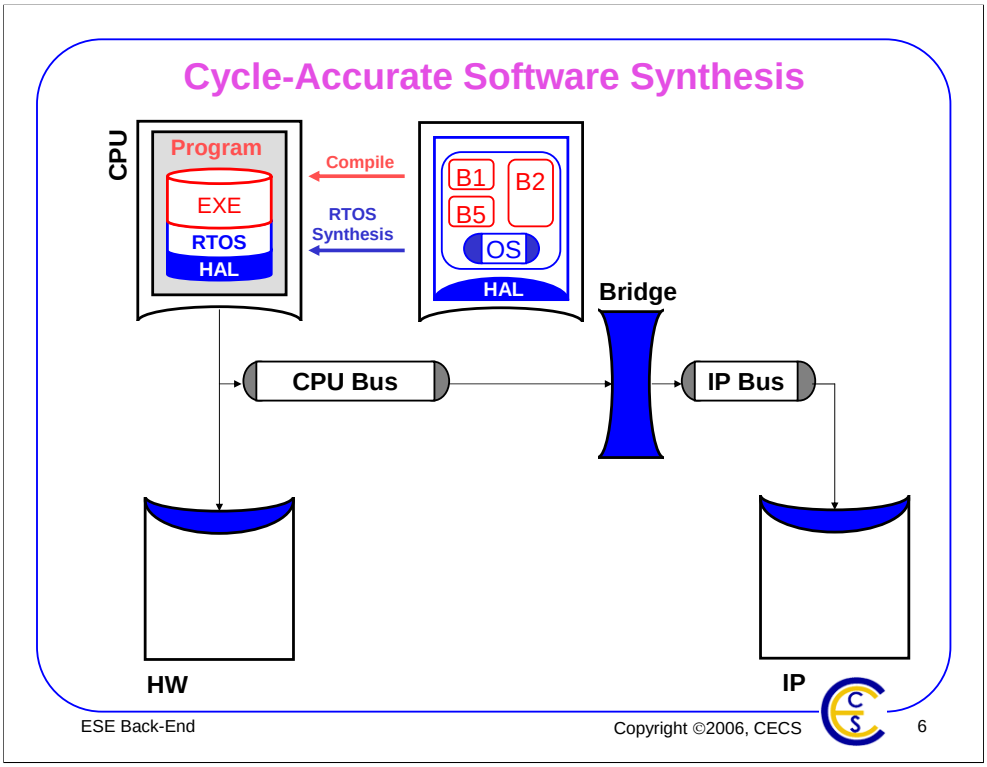
TLM Features

TLM models the bus using a universal bus channel (UBC). The UBC provides different types of functions like Send, Receive, Memory Read/Write and Memory service. Application processes executing on different components of the platform use UBC functions to communicate amongst themselves. UBC functions implement routing of messages, process synchronization, bus arbitration and data transfer in an abstract, application independent manner.

SW is modeled as a set of C processes. If there are more than 1 processes, then a dynamic scheduler is required to schedule them on the host CPU. In TLM, a model of the RTOS may be used for this purpose. This RTOS model emulates the actual RTOS scheduler, but runs at a much higher speed. Communication between SW application processes and other processes or memory mapped to peripherals is enabled by the UBC interface to the SW processor.

HW modeling is also done using C processes and is similar to SW modeling, with the exception that there is no RTOS model for the HW. Each HW process is synthesized with a different controller, so all scheduling is static.

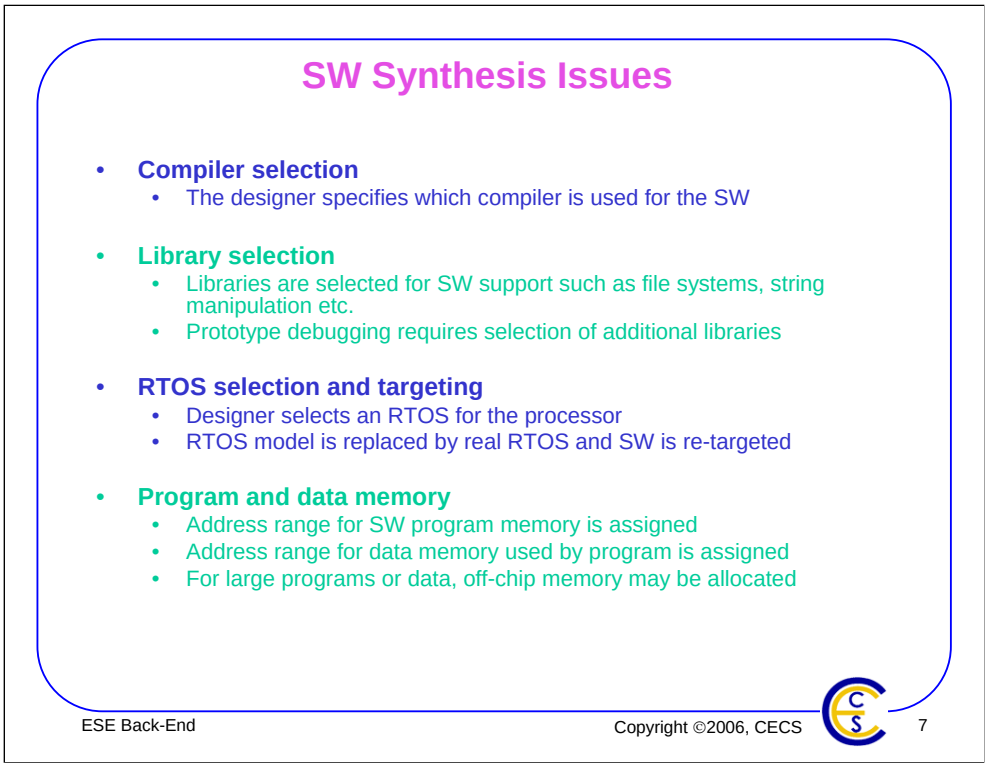
Memory is modeled as an array in C. A special memory access function in UBC is used to control access to this array (and hence models the memory controller).



Cycle-Accurate Software Synthesis

There are primarily two steps in SW synthesis. The first part is the insertion of a real RTOS to replace the OS model of the TLM. The second part is the transformation of UBC function calls with RTOS and platform specific C code. The transformed application code is compiled and linked with the RTOS and hardware abstraction layer (HAL) libraries to create the final binary that executes on the CPU.

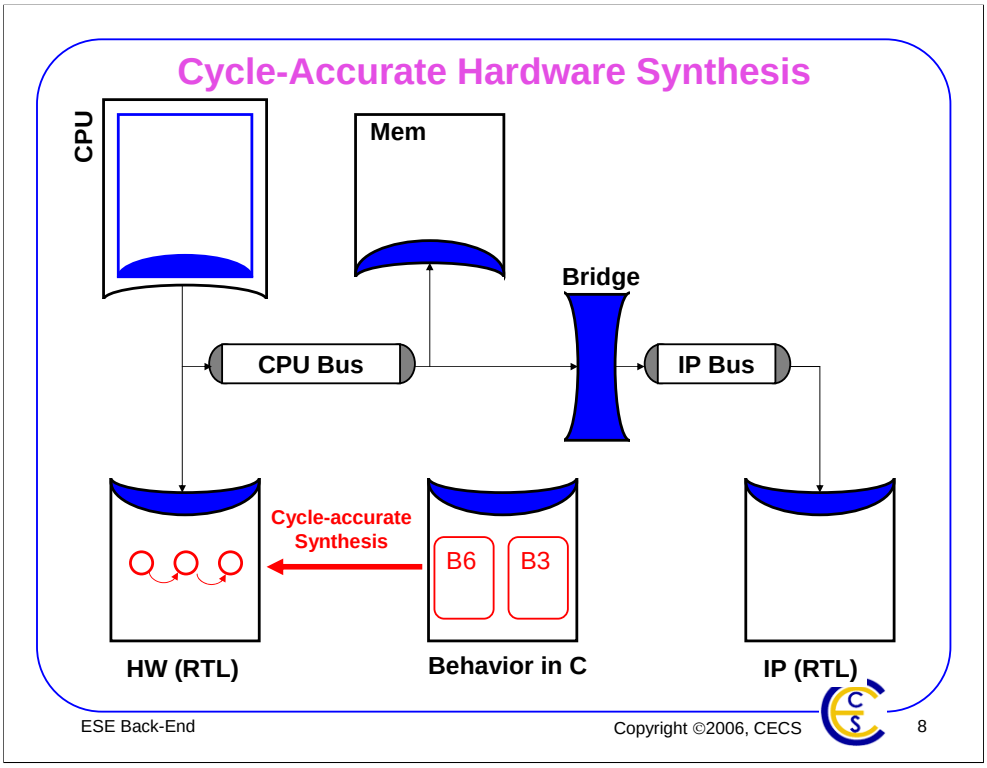
For SW performance estimation purpose, the application SW can also be developed using commercially available tools, such as Instruction Set Simulators (ISS) that are compiled and inserted into the TLM. ESE will upgrade the TLM by inserting the Instruction Set Simulator and compiled binaries. Similarly, users can develop custom operating system and insert it into the model. Obviously, cycle-accurate HW and SW models run much slower and should be inserted selectively. Instead timed TLMs generated by ESE front-end may be used for fast and accurate performance estimation.



SW Synthesis Issues

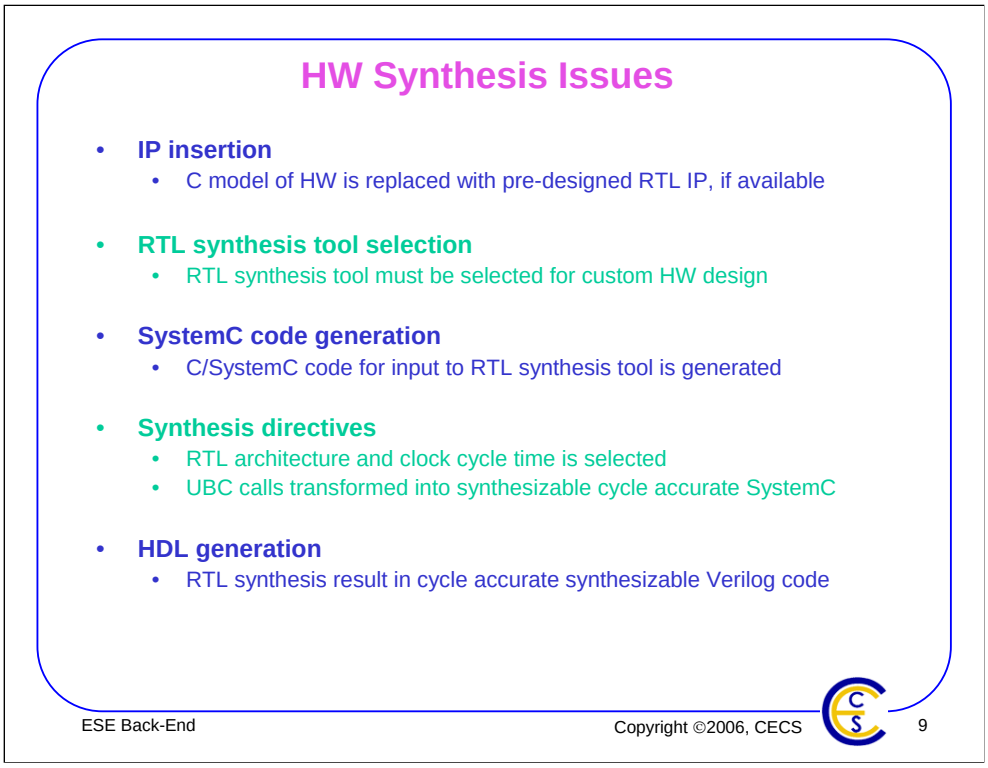
If the intended target is a FPGA board, the SW must be configured so that it can be input to FPGA design tools, such as Xilinx Embedded Development Kit or Altera SOPC Builder. A compiler is selected for the SW. Often times, SW application will require additional libraries such as file system libraries or string manipulation. All such libraries must be selected and added to the SW description. Similarly, if the prototype is to be debugged, most FPGA providers include debug libraries and JTAG interfaces for the processors that come with the board. These libraries may also be selected if debugging is required. The purpose of library selection is to estimate the lower bound of required program memory and allows appropriate address range selection.

The model of the RTOS is replaced with the actual RTOS. Therefore, all function calls in the application must be replaced by appropriate function calls to the actual RTOS library. Hence, the SW is retargeted for RTOS. Based on all this input, a default address range of the SW program and data is created. This address range is passed to the compiler and FPGA design tool for creating the final address map for each bus in the system. If on chip memory is not sufficient, off-chip memories may be selected for both program and data.



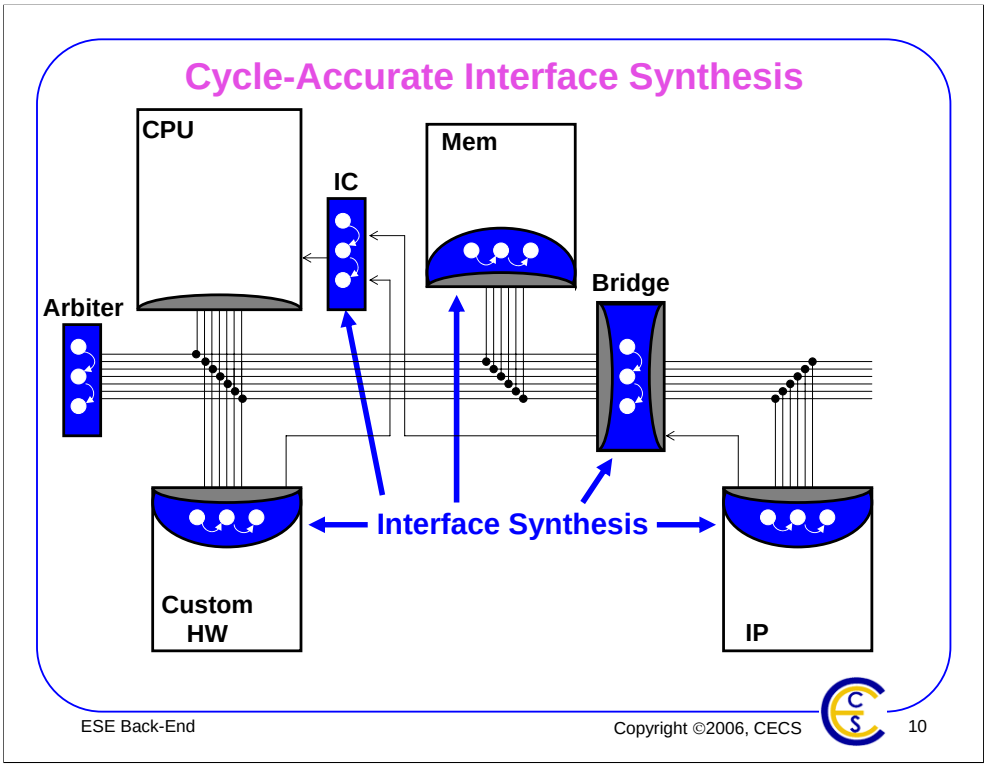
Cycle-Accurate Hardware Synthesis

The HW components can be designed using standard C to RTL synthesis tools such as those provided by Forte. However, most commercial and academic RTL synthesis tools have constraints on interface and SystemC coding style. ESE automatically generates SystemC code for synthesis with Forte. This SystemC code includes the user C processes and models custom HW bus interfaces using cycle accurate SystemC synthesizable by Forte. Using automatic bridge generation feature of ESE, it is possible to generate RTL interface between system buses and custom HW bus supported by RTL synthesis tool. Therefore, ESE provides a seamless integration of commercial C to RTL synthesis tools in the back-end.



HW Synthesis Issues

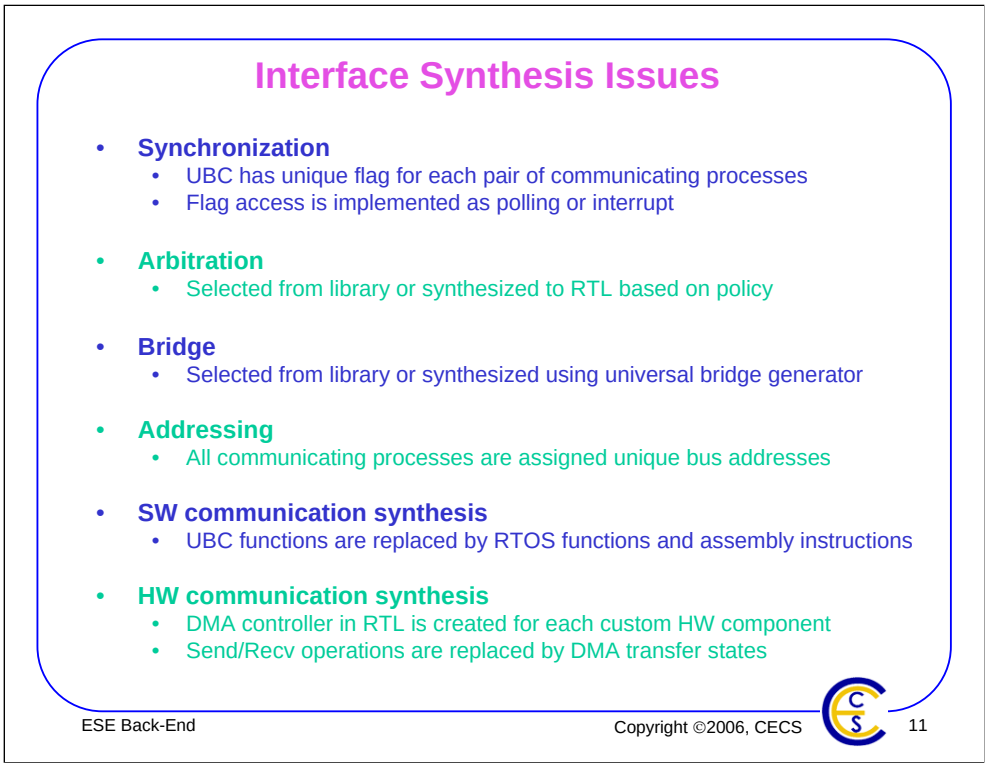
If a HW component may be replaced by an available IP, then no RTL synthesis is required for that component. This case is applicable if the C model of the HW is simply an abstract representation of an existing IP which the designer intends to use in the system. If no such IP is available, commercial C to RTL tools may be used to generate the custom HW. ESE generates the required SystemC code for the supported RTL synthesis tools such as Forte. The user can also select synthesis directives such as RTL architecture for NISC and target clock speed. The send/recv function calls to UBC from C processes are automatically transformed into equivalent cycle accurate SystemC code for RTL synthesis. The result of RTL synthesis is Verilog code that can be input to logic synthesis tools such as Design Compiler, XST or Synplicity.



Cycle-Accurate Interface Synthesis

In the final step ESE automatically synthesizes the interfaces for communication between components. This way users can test the communication between newly developed cycle-accurate HW and SW. This is the final step in the development of the system on a chip or a prototype in a FPGA.

Once the SW and HW components are debugged and tested on the CA level, users can switch back to TLM for development of application SW. Similarly, upgrading the system could also start on the transaction level.



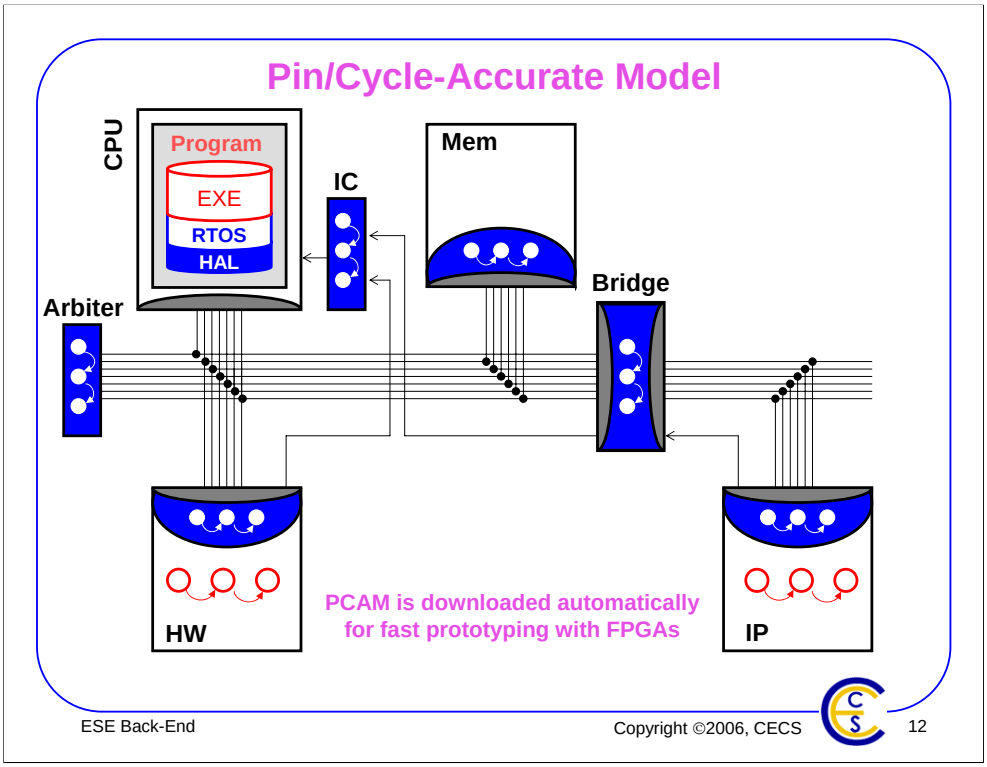
Interface Synthesis Issues

Interface synthesis consists of implementing the abstract communication methods of the UBC. The abstracted flag-based synchronization in UBC is implemented as an interrupt-based, polling-based or other user defined scheme. If the CPU does not have capacity for required interrupts, an interrupt controller is used. The designer may synthesize an arbiter or select one from the library.

Bridges are automatically generated in ESE for connecting cores with incompatible interface protocols. The RTL implementation of the Bridge can be obtained from the Bridge Generator tool in ESE by providing a set of parameters, such as the bus protocols, size of internal buffer etc.

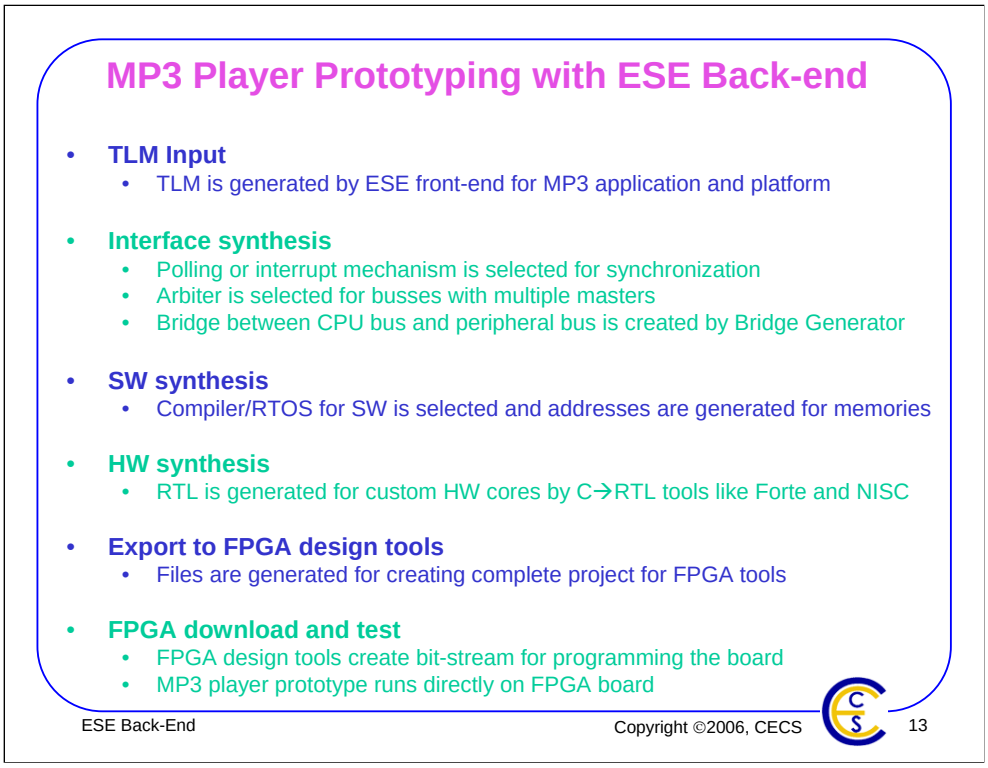
A unique address is given to every process in the model for sending or receiving data. In the SW application code, the UBC calls for send/recv are synthesized into RTOS function calls for interrupt handling and C code for data transfer.

For HW interface synthesis a DMA controller is added to acts as an interface between the HW and the system bus. A send or receive call in HW application is translated into DMA setup, start and done states.



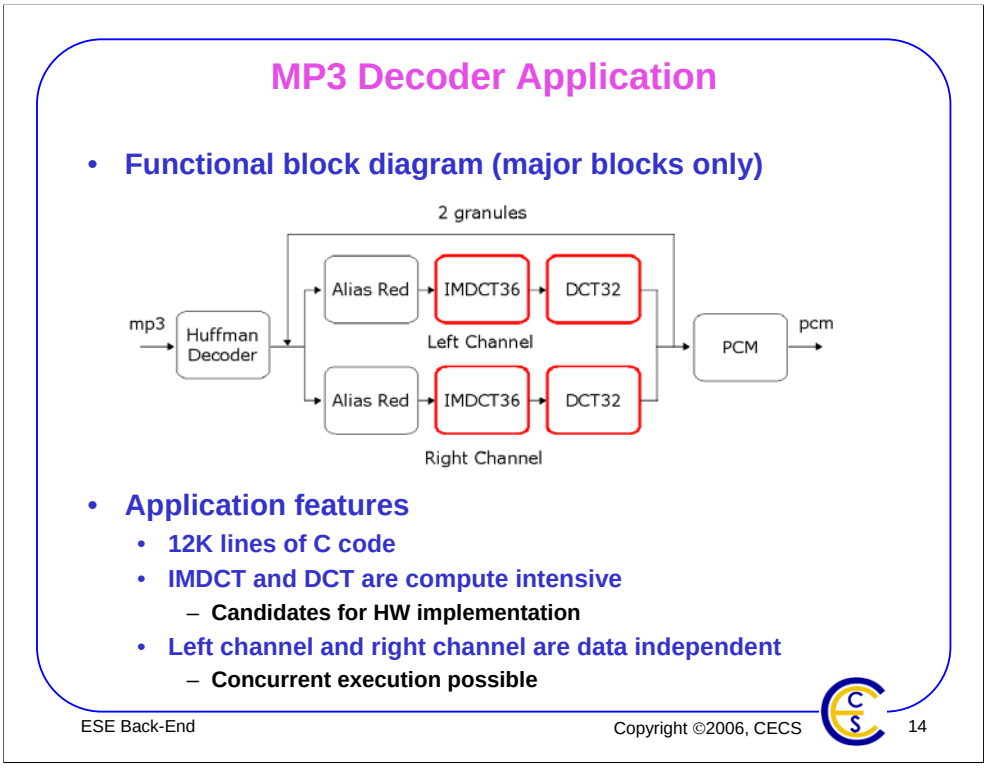
Pin/Cycle-Accurate Model

The ESE back-end generates the pin accurate model of the system for FPGA download. This model includes all the SW code, including application, libraries and RTOS. I also contains all the HW components as synthesizable Verilog. Depending on the prototype target, ESE generates files that will be needed by the respective FPGA design tools. These files and the model is exported to the FPGA design environment where the designer compiles the SW and performs logic synthesis for the HW components. Finally, a bit-stream is generated that directly programs the FPGA with the prototype. This programmed FPGA typically has a hyper-terminal user interface that can be used to debug the prototype.



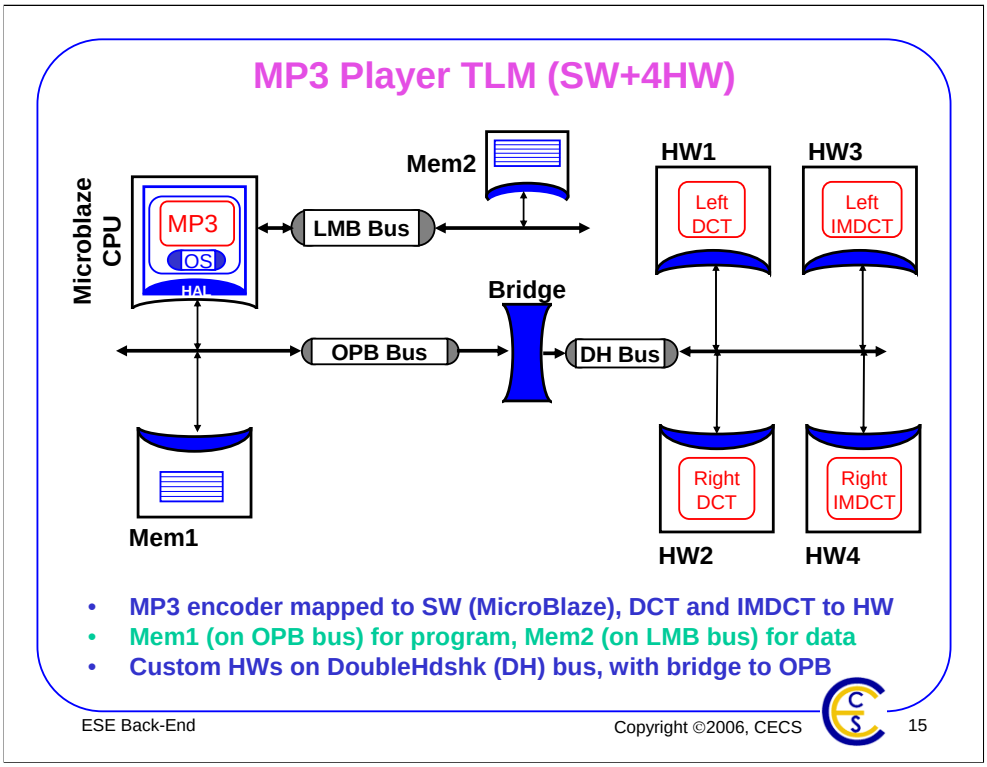
MP3 Player Prototyping with ESE Back-end

To demonstrate the ESE support for large size application and heterogeneous multi-processor platforms, we selected four MP3 decoder designs. The TLMs for these platforms and application mapping were generated automatically using the ESE front-end. The above steps were followed to generate the PCAM using the ESE back-end. The PCAM was downloaded to a Xilinx prototyping board and tested.



MP3 Decoder Application

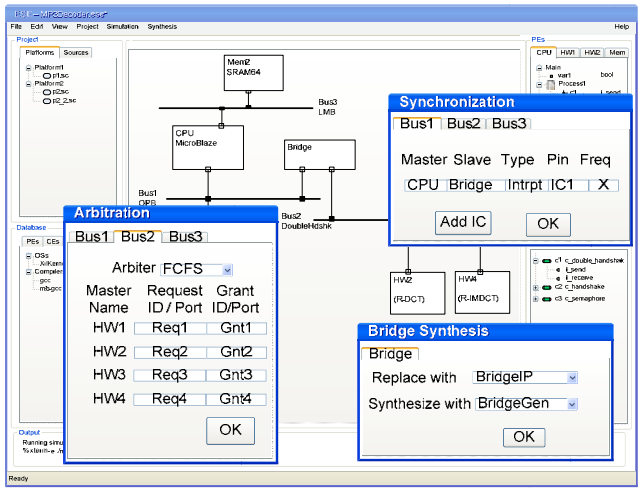
To demonstrate the usefulness of ESE, an MP3 decoder application was chosen. The block diagram above shows the IMDCT and DCT transforms that are applied during the stereo decoding on the left and right channels of the MP3 input. These function blocks are the most time consuming part of the decoding and are hence ideal for implementation using custom HW for faster decoding. The C model is also used to create test benches with golden PCM output files. These test benches are used later to verify the ESE generated PCAMs in both the simulation model and the board prototype.



MP3 Player TLM (SW+4HW)

The MP3 application is primarily implemented in SW on MicroBlaze processor. The above TLM figure shows the most complex MP3 design done with ESE. Some compute intensive parts of the application have been mapped to HW. These include the left and right DCTs and the left and right IMDCTs. The CPU generates decoded data that passes through the DCT and IMDCT and finally sent to speaker input. Communication is therefore from CPU to DCT/IMDCT HWs and back. Since the filters and IMDCT are to be designed as custom HW with a proprietary DoubleHandShk (DH Bus) interface, a bridge is inserted between CPU (OPB Bus) and the DH Bus. Mem1 holds the program running on CPU while Mem2 holds the data for the CPU. For optimization purposes, we use a direct connection between CPU and Mem2 using LMB bus supported on the Microblaze core.

Interface Synthesis



- **Interrupt signals and connections are selected**
- **Arbiter is selected and request / grant pins are connected**
- **RTL code for Bridge is generated using BridgeGen**

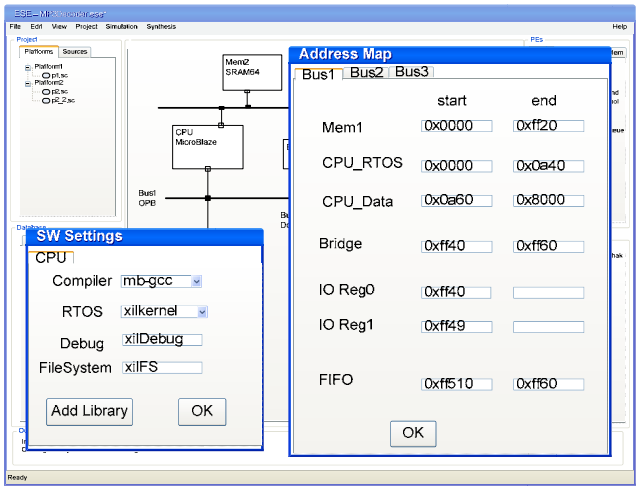
Interface Synthesis

A bridge between Microblaze OPB bus and DH bus is generated automatically with BridgeGen.

Communication between the CPU and the HW components must pass through the bridge. Since this communication is synchronized, we need a method to implement synchronization between CPU and Bridge. Since Microblaze has an available interrupt input, we use it for synchronization with Bridge. The Bridge acts as a slave device and generates the interrupt that is input to Microblaze.

On the DH bus, HW components must communicate with the bridge. Therefore, they are assigned to be masters on the bus since the bridge is a slave on DH bus. Thus, an arbiter is selected to perform first come first serve (FCFS) arbitration between HW1, HW2, HW3 and HW4 on bus 2.

SW synthesis

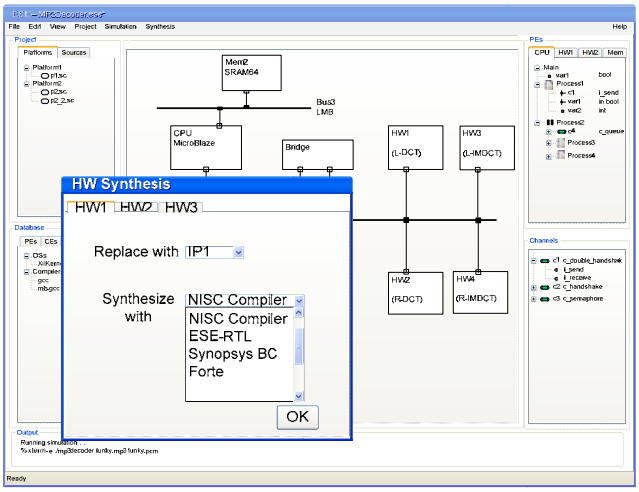


- **Compiler, RTOS and libraries are selected for SW**
- **Default addresses for all addressable memory/bus is generated by ESE**

SW synthesis

SW synthesis includes selection of compiler, RTOS and libraries for the Microblaze processor. The compiler used is mb-gcc, the RTOS is xilkernel, provided by Xilinx, and file system and debug libraries (also from xilinx) are selected. The default address map for program memory on Mem1 and data memory on Mem2 is generated by ESE. Also, addressable registers inside the bridge are assigned addresses on Bus1. Similar addressing is done for HW components and Bridge on Bus2.

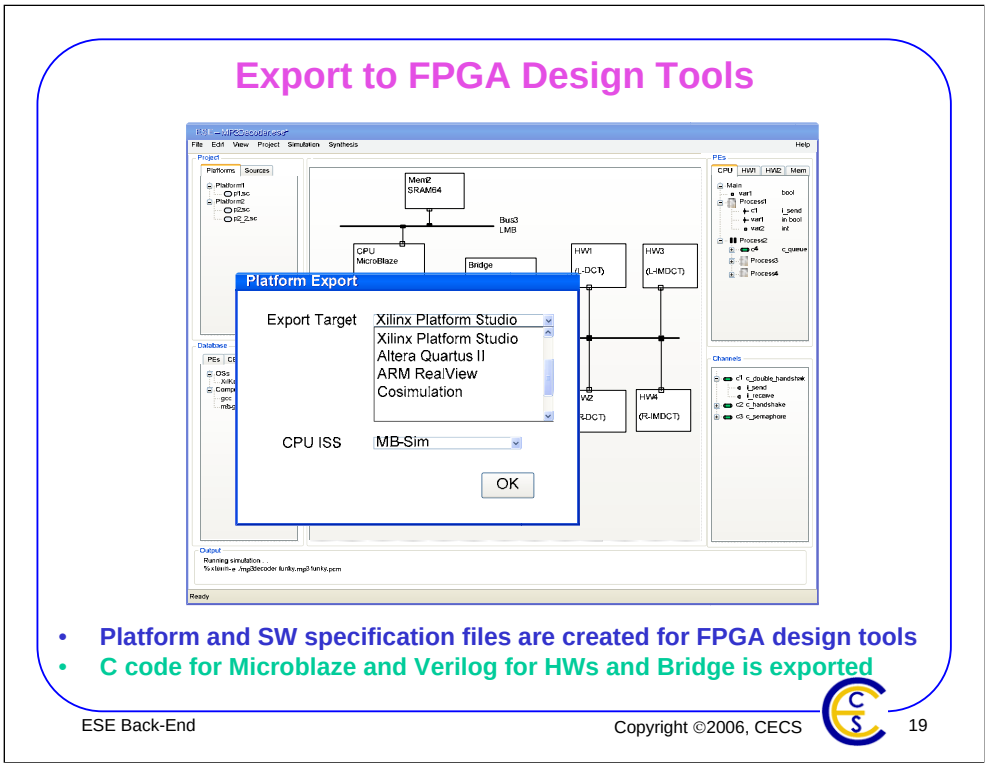
HW synthesis



- **RTL code for HW components is generated using C→RTL tools**

HW Synthesis

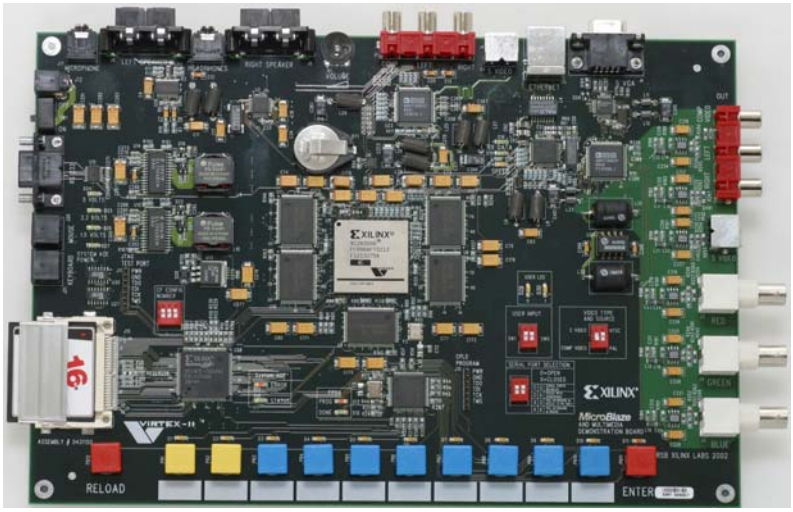
HW components for the MP3 design are synthesized using NISC compiler. ESE integrates other commercial tools such as Forte as well. SystemC code and constraints are automatically generated for each HW component automatically by ESE. It is also possible to import pre-designed IPs into ESE designs. Therefore, ESE can be used for incorporating legacy designs or for upgrades to next generation product.



Export to FPGA Design Tools

The model is now ready to be exported to FPGA design tools. Depending on the design tools, ESE will generate all the files needed to create a project for FPGA prototyping. Once the design tools are launched, the designer only needs to compile all the SW for each CPU in the platform and synthesize the ESE generated Verilog for all the HW components. In this case, the target board is Xilinx and the platform model is exported to Xilinx Platform Studio. Thus the design is handed off for FPGA designers and prototype testers.

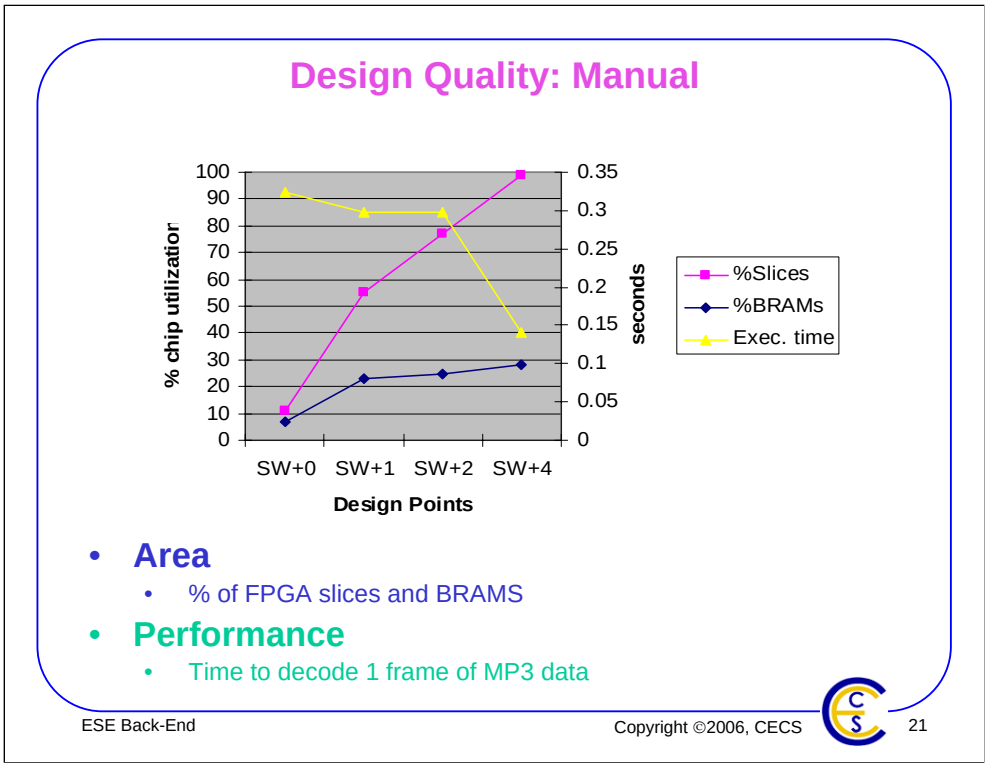
Benefit: FPGA Prototype in 1 Week



- Bit stream from FPGA design environment is downloaded to board
- Implemented prototype is tested with MP3 music files

FPGA Prototype in 1 Week

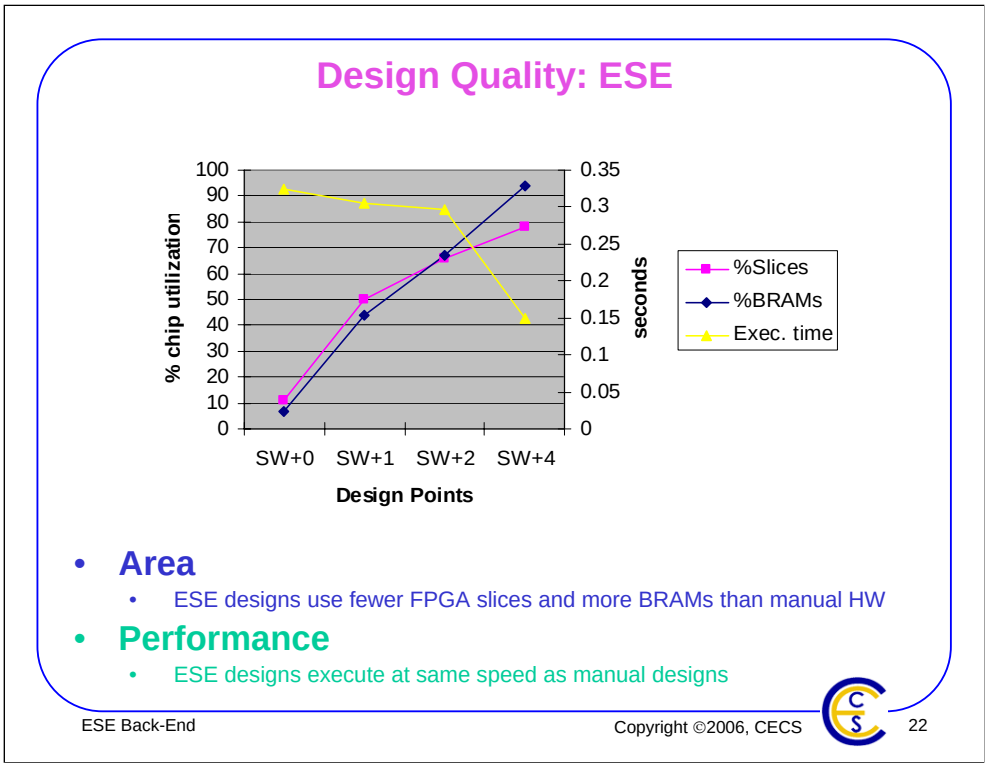
The Xilinx Platform studio creates the bit stream needed for programming the board, which is downloaded using a PC cable connection. The MP3 player prototype is now ready to be tested on board. Using ESE, system level design decisions can be implemented quickly and synthesizable models for SW and HW generated automatically. Therefore, months of development effort in system prototyping can be reduced to less than a week with ESE.



MP3 Manual Design Quality

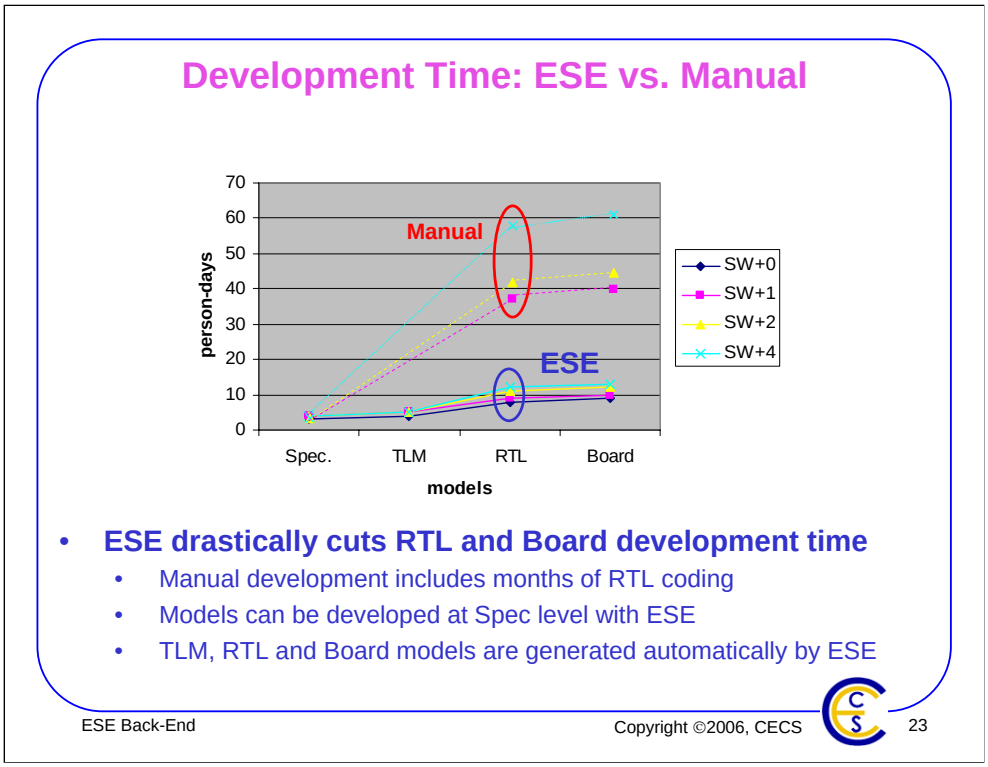
In order to establish benchmark for ESE designs, 4 manual designs for MP3 decoder were developed. SW+0 refers to a purely software implementation where all reference C code for MP3 decoder mapped to Microblaze processor. SW+1 is the design with one DCT implemented in custom HW with remaining MP3 code on Microblaze. SW+2 implements both left and right channel DCTs in HW. Finally SW+4 implements both left and right channel DCTs and IMDCTs in custom HW. The above chart shows the number of FPGA slices and BRAMs needed to implement the 4 designs. It also plots the time to decode 1 frame of MP3 data by each design.

It can be seen that as more HW components are added, the area increases while decoding time comes down. By adding HW DCTs we get SW+1 and SW+2, which increase the design cost, but give only marginal improvements in performance. Once the IMDCTs are also moved to HW, we get SW+4 which has the highest area, using almost all the available slices on the FPGA. However, it gives the best performance of all the designs, with more than twice the decoding speed as the pure SW implementation.



MP3 ESE Design Quality

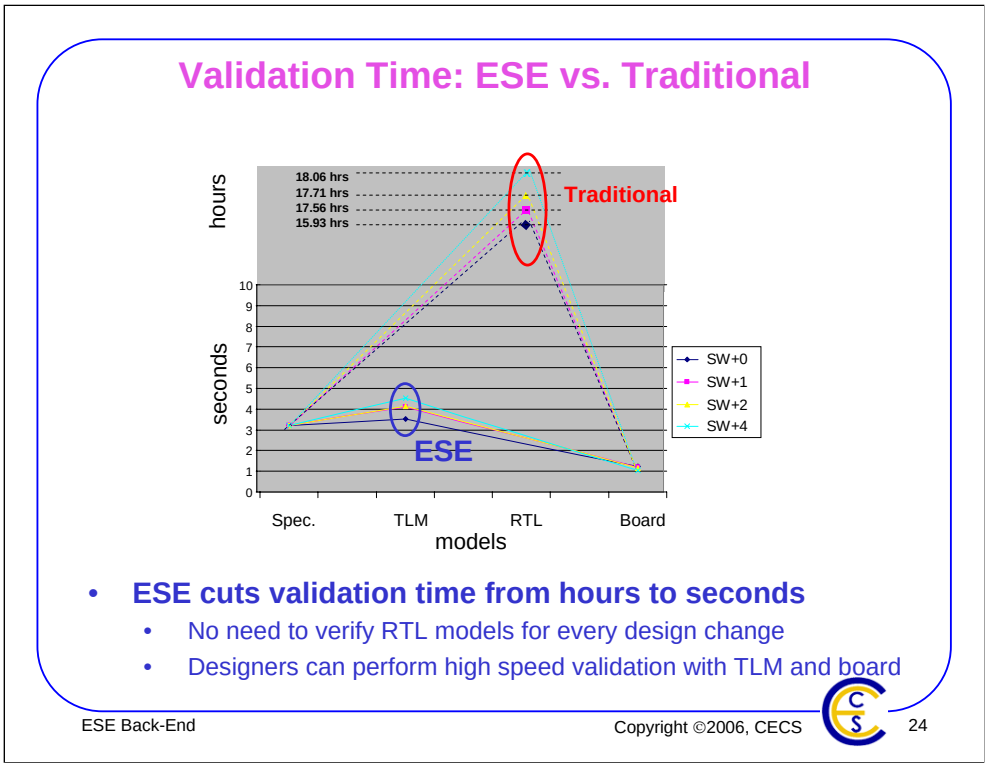
The same partitioning and platform as manual MP3 designs was used to implement MP3 decoder with ESE. It can be seen that ESE designs use more BRAMs but fewer FPGA slices than manual design. This is because the ESE generated HW components used more memory intensive controllers than the manual designs. The execution speed of ESE designs were comparable to manual design. Therefore, ESE can automatically generate prototypes for multi-core platforms with the same quality as manual design.



Development Time: ESE vs. Manual

Traditional design practice starts with RTL and embedded SW coding for selected platforms. The reference C specification model is used for developing test bench to verify the cycle accurate models. For MP3 platforms with HW components, the RTL development time was in the order of months. As a result, board prototypes for these designs took between 40 to 60 days.

ESE drastically cuts prototype development time by automatically generating TLM and RTL models. 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.

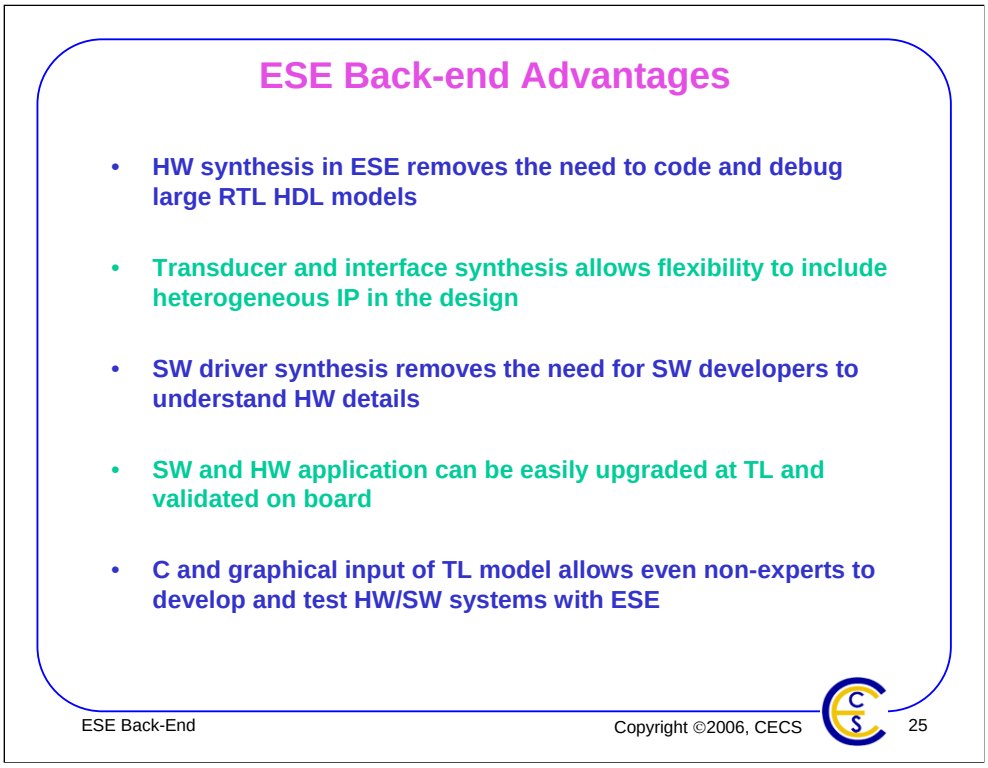


Validation Time: ESE vs. Traditional

As a consequence of traditional cycle accurate modeling, designers must make design optimizations and changes on RTL and low level SW code. Each change needs to be verified using time consuming cycle accurate simulations. Each cycle accurate simulation cycle took 15 to 18 hours for MP3 designs. This is a significant component of design time. Although at speed on-board verification is faster than even reference C simulation, bugs found in on-board testing are difficult to trace back to RTL.

ESE removes the burden of cycle accurate simulations by moving the design abstraction to TLM. ESE generated TLMs execute at the same speed as reference C simulation. Design changes are made at the transaction level and can hence be verified and debugged using the automatically generated high speed TLMs. TLMs are easier to debug and maintain because their code size is at least an order of magnitude less than RTL.

Automatic RTL generation is also less likely to introduce bugs in the design compared to manual RTL optimizations. This has been true in the past when the modeling abstraction moved from gate level to RTL with the use of logic synthesis tools. Therefore, ESE reduces verification time from an order of several hours or even days to a few seconds. As a results, 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 Back-end Advantages

There are numerous advantages of using ESE. The product specification and implementation is easily captured with proprietary GUI. All models are generated automatically after proper design decisions are made by the users. This saves enormous amount of time in learning modeling languages and writing and interfacing appropriate models. SW, HW and interface synthesis allow design optimization at higher abstraction level as well as easy IP import.

Product upgrades are simplified because ESE allows convenient reuse of legacy application code and design decisions.

ESE allows parallel development of SW, HW and application code and their integration. Prototype ready cycle accurate models are generated automatically from TLMs leading to significant savings in model development and verification time.

Acknowledgments

- **We would like to acknowledge the previous R&D teams who contributed many concepts and methods used in ESE 2.0**
  - SpecCharts/SpecSyn ('92): F. Vahid, S. Narayan, J. Gong, S. Bakshi
  - SpecC/SCE ('00) team: R. Doemer, J. Zhu, A. Gerstlauer, J. Peng, D. Shin, L. Cai, H. Yu
- **We also want to thank P. Chandraiah for MP3 reference code**