Good afternoon, The topic of my presentation is hardware/software co-verification.
The contents of this presentation including 4 parts. In part one, I will discuss the motivation of co-verification, and give you some concepts on co-verification models, as well as the benefits we can get from it. Next, I will talk about the co-verification platform from mentor graphics, which is called Seamless CVE. After we have basic understanding of co-verification, we will go further to study how we can apply this technology into our soc design. And the last part will be the conclusion. Ok, let’s start.
Nowadays, most of the digital chip designs are of the form of embedded system or system-on-a-chip. Since the integration of various logic circuits and software code, the system becomes more and more complex. In the hardware domain, a system usually consists of a processor or multiple processors, memory blocks, IP blocks, and a bus for all the components to communicate with others. The software part is as important as the hw one, it needs to communicate with processor and IP blocks, it also needs to drive I/Os. The most difficult and important of an embedded system design is the integration or verification of hw and sw.
Traditionally, the design cycle is divided into three phase. The first phase is high-level system design, including system requirement analysis, partitioning the system into sub-systems and defining the hw/sw interface. Then hardware team and software team separately start to design and implement their modules according to the specification established from phase one. Each of the teams are doing their own job without any discussion with the other. After everything is successfully implemented and tested, the modules are ready for the physical integration. Actually phase three can be treated as two steps. First, the hw team needs to create a prototype for their hardware design, while at the same time, the sw team has nothing to do but waiting. Finally, the software design can be loaded into the hw prototype to test the functionality, performance and reliability.
As we have all know, the time-to-market is one of the critical consideration for a company. To shorten this period can help the company to occupy the market early and make huge profit, not to mention the human and other resources saved. Besides, it is not uncommon for verification teams to spend as much as 50 to 70 percent of time and resources on the functional verification. There is high motivation to finding problems earlier in the design cycle because the later a problem is found, the more expensive to fix it. Obviously, An advance verification can help to reduce the cost and make it easier for both hw and sw teams to debug. So that's why we need hw/sw co-verification.
This comparison slice can help us understand more on. From this chart, we can see that the integration must be done after the prototyping, which means a huge waste of our precious time. (click..) A newer design cycle involving the co-verification make it possible for us to start the verification ahead of hardware implementation. The whole verification/integration process overlaps with both designing and prototyping. And this methodology not only save time for developing but also simplify the process of hw/sw integration.
Here we can see a simple model of HW/SW co-verification. The basic concept behind co-verification is to merge the respective debug environments used by hardware and software teams into a single framework. The logic simulator is made to communicate with the software debugger/simulator by a simulation interface, enabling users to get simultaneous control and visibility into the internals of the processor as well as the hardware logic that surrounds it. The interface controlling communications between software and logic simulators, called co-verification kernel, plays the most important role in co-verification, ensuring the accuracy and consistency of data transfer.
Co-verification Vs. Simulation

- RTL Simulation
  - Pure RTL models
  - Slow: <100Hz
  - Unable to address all debug requirement
- Co-Verification
  - Orders of Magnitude Faster
  - Increased Comprehension
  - Support for Abstract Models

Though pure logic simulation with RTL models of the processor and memories, can be used to simulate a design with a processor component, it however is painfully slow and not sufficient. The overall simulation speed is generally below 100hz, while the optimized models from co-verification tools can speed it up to orders of magnitude faster.

-- increased comprehension: Co-verification tools allows a symbolic debugger to be attached that allows interactive and graphical debugging, including ability to step through source code (C and assembly), set break point, observe register and memory contents, etc. Thus allowing for greater control and comprehension over pure simulation.

-- Abstract: For very high data throughput simulation, co-verification tools support models from higher-level language, like C or C++, thus allowing designers visibility into system performance and architectural tradeoff issues at a very early stage in the design cycle
Next I will talk about a industry leading hw/sw co-verification tool by mentor graphics – Seamless Co-Verification Environment (CVE).
Mentor Graphics Seamless CVE provides a high-performance virtual platform for interactive HW/SW debugging. The picture shows the basic architecture of the co-verification environment. Basically, the CVE can be divided into 3 parts, software simulator/debugger, seamless co-verification kernel and logic simulator. Software simulator execute machine code produced by compiling target software for the specified processor. Besides, seamless provide an alternative scheme for designer to write embedded software in high-level language and to compile it for execution on the workstation. So it gives the user opportunity to test software without generating machine code for the target processor, and it runs much faster than the software simulator. The Debugger interface allow user to control the software functions, to exam the registers and memory, and to analyze data transition in an easy way. Logic simulator performs the hardware portion of co-verification, such as the hardware design, bus-interface models and memory models. The hw interface can show you the wave of all signals with your design. Merging these two separate domains into one co-verification environment is the task of the co-verification kernel, which is the core of the seamless application. It controlled the communications between hw and sw, maintains a consistent view of memory contents between the hw and embedded sw simulation environments through coherent memory server, and coordinates the co-verification of hw and sw.

---ISS: By running the ISM on a software simulator, software coverification - sometimes referred to as Instruction Set Simulation (ISS) - can be performed much more quickly than logic co-verification because it does not have to calculate all the signal transitions that occur in the gates and registers within the processor.

-- Processor’s pin behavior model and optimizable memory model is simulated in this part in the form of HDL description.
What we wanted is to accelerate the co-verification process with seamless application. There are two factors that help CVE to do that. The first one is separating processor’s functions from its interface, and the other one is selective suppression of bus cycles. These accelerating factors are provided by the three main components within seamless co-verification environment.

Processor Support Packages port The first factor is realized by Seamless Processor Support Packages (PSP) is the component came with seamless CVE to replace the processor modules in your hardware design. Seamless have PSP for over 100 microprocessor and DSPs, including ARM, Ceva, IBM, MIPS and Motorola. Yes, you have to choose the right PSP for your target processors or to replace your processor modules, in order to perform co-verification with seamless CVE. Actually, A PSP is consist of two parts, Instruction Set Model executed on the software simulator and Bus-interface Model ran on the logic simulator. It seems complicated, but that is why seamless is hundreds times faster than pure logic simulation. The flexible memory system gives CVE the ability to suppress bus cycles on purpose. Optimizable Memory Models are able to be instantiated in the hardware design to simulate any kind of memory modules that would be used on the actual product. The word optimizable means they can be optimized for the purpose of hiding clock cycles for parts of the logic simulation. All the optimizable memory blocks are managed by the coherent memory server existed in the co-verification kernel. It takes the responsibility to maintain an identical (coherent) view of memory for both hw and sw simulators. Let’s talk about them one by one. The Coherent Timers are used to synchronize the hw and sw in some degree when the design is running in time-optimized mode.
First, the processor support packages. The slice shows how the psp is separated into two components and how the two components are placed in the co-verification environment. ISM is a high-level software model to execute as a separate, much faster processor on a software simulator to perform all kinds of processor operations, such as fetching instructions, and reading/writing registers. Bus-interface model described in HDL language is simulated on logic simulator for the processor’s input/output pin behavior. That means let the software, which runs fast, do the data processing job but the logic simulator, which runs slow, do the pin transitions between processors and IPs. The arrangement makes the simulation runs faster. These two components are co-operated with the help of seamless kernel to act together as one signal processor.
An ISM is a software application that contains all a processor has inside to perform all the functional behavior of processor’s instruction set. It is not a full model of the respective processor, but an abstract model of the data processing during instruction execution. It can fetch, decode, and execute instructions, read and write memory and I/O data, and simulate the processor’s registers and other internal data-handling functions. It sends out data-access requests and instruction timing to logic simulator and gets back responses and exception requests through seamless kernel. Using a software simulator interface, you can control the execution of the software and view or change the memory contents on optimizeable memory models.
The bus-interface model is a hardware model existed on hardware domain, defining the interactions of input/output pins with the surrounding hardware. During co-verification, bus interface model only performs the pin behavior instead of modeling the internal logic of a processor, which was done by the ISM. Since it only executes the necessary logical operations to carry out the required pin transitions within the pin interface when receiving requests for bus cycle from software simulator, the co-verification can run really fast than complete functional model. Usually, this model can execute these types of bus cycles, xxxxxxx....

Data Access Cycle - Data access requests take place when the processor reads data from or writes data to memory and I/O ports. In co-verification, these requests originate from the ISM or Host Code Execution program as they model the software execution. When the bus-interface model receives a data-access request from the co-verification kernel, it executes the requested bus cycle by asserting signals on the appropriate input and output pins.

Exceptions and bus Control Cycles – The surrounding hardware design can also initiate events, such as faults, interrupts, bus-arbitration requests and grants, and coprocessor operations. The model simulates the bus interaction that the processor performs in response to such an event. In the case of interrupts, the software simulator receives the data that allows its interrupt-service routines to service the interrupt.
Now we will talk about how the ISM+BIM work together to finish the software execution. First, the designer can use any cross-language tools, such as assemblers, compilers and linkers to generate machine code for target processor. When the sw is ready for test, introduce Seamless application into design flow and load the sw modules into the ISM address space. Based on the access address, the co-verification kernel determines whether to pass the requests to the bus-interface model or to service the requests through local memory without causing any bus activities for logic simulator. The ISM reports the clock cycles for a given instruction to the co-verification kernel for clock synchronization with hardware part, while the bus-interface module responses with processed data or any exceptions, such as interrupts and resets.
The availability of optimizable memory models makes it possible for seamless to suppress logic bus activities by allowing some memory access to be serviced by software instead of hardware simulator. And this scheme accelerates the process of co-verification. The chart illustrates the relationship among seamless application, hw and sw components and the optimizable memory models. The optimizable memory models are maintained by the Coherent Memory Server, which allows independent access to these models form both sw and hw. This is important, because it lets the hw and sw portions to maintain an identical (coherent) view of memory. It suppresses logic activities and allows the ISM to access memory in full software speed. There are four categories of optimizable memory models available to Seamless applications.

Generic Optimizable Memory Modules – Seamless provides a set of generic memory model templates: a dynamic RAM, a static RAM, a dual port RAM, a FIFO and a register element. You can choose this modules when your design requires only simple memory blocks.

Denali Memory Models – This is the best option when your memories are external to your ASIC/SoC design, since with this method you can generate models for almost any commercial memory device using PureView application.

HDL Interface – This can be used to modify the existing HDL memory modules to work with the Seamless Coherent Memory Server. It is good if your memory models are internally developed or provided by vendors. By
1. You can find the generic memory templates described in VHDL or verilog in these 2 directories.

2. After obtaining a SOMA file for your designed memory model, you can load and configure the memory parameters and then output the appropriate HDL wrapper required to instantiate the model within the HDL design by using MenMaker or PureView applications. In the program, select vhdl/verilog and hardware simulator. After that, include the model in your design.

3. First, compile an additional vhdl file, which defines the CVE functions used for memory interface. Then add an extra use statement to use package of CVE_DIRECT. NEXT, replace the definition of the memory storage array by cve_RegisterMemory(), replace the statements where a data value is either read from or written to a memory location with either cve_ReadMemory() or cve_WriteMemory() calls.
Co-verification can verify the integration of sw and hw to a high degree of accuracy. But detail co-verification could be much too slow, since the logic simulator has to execute all the bus cycles needed for all memory operations and instruction fetches. Repeating these kinds of simulation in detail means nothing if the software has already verified the basic memory access. So we can optimize the co-verification by trading simulation detail for speed to obtain the right mix of accuracy and performance. Based on the above statement, the memory modules which are frequently accessed by processor should be optimized. The left side of the chart shows the traditional simulation, while the right one shows the optimized co-verification. With the help of coherent memory server, the ISM/ISS can access the optimizable memory modules directly without any operation from the logic simulator. In this case, some parts of the bus cycles for memory access can be hided, enabling higher simulation performance. However, the bus cycles or activities suppressing must be harmless to the simulation of hardware.
Seamless provides 3 categories of optimization, data access optimization, instruction fetch optimization and time optimization.

1. Normally for an un-optimized co-verification, each memory access by process cause bus-interface model to generate a read or write bus cycle in the logic simulator. Once the hardware is proved to be functioning correctly, we do not need these cycles to be generated any more. So by active data access optimization, Seamless disables these bus activities for logic simulator and allows the ISM to directly access the optimizable memory through coherent memory server. Though without these activities, the logic simulator continues to advance with software simulator to maintain the cycle accuracy.

2. Same as the data access optimization, instruction fetch optimization eliminates the bus activities in logic simulator for all instruction fetches that take place in optimizable memory. It is a convenient way to hide a class of bus cycles that rarely affect co-verification accuracy.

3. Time optimization eliminates all the logic simulation for parts of your program where hardware timing is not important. Unlike the other method, time optimization eliminates hardware clock cycles in addition to bus activities. That makes the software run on the full speed without advancing the logic simulation until a specified number of software cycle is occurs or a preset condition is met, such as exceptions or access to un-optimizable memory. So the hw and sw will not be always synchronized.
Now, let look at the bus activities in the logic simulator for an un-optimized co-verification. The left upper part shows a segment of machine code, the corresponding bus cycles that the logic simulator should perform, including instruction fetches, memory operations and I/O operations. From the waveform, we can tell that the hardware simulator keeps repeating the same bus cycles for some of the operations, like fetches, memory read/write. And these bus activities are be able to be disable for speed up purpose. Please notice the memory arrangement, only the address range from 5,000 to 8
When the data access optimization and the instruction fetch optimization are applied, the bus activities of all the instruction fetches and parts of the memory access are eliminated. Only those of the I/O operation and last memory writing to address 80,000. As we mentioned before, these two optimization methods operate only on the OPTIMIZABLE memory address range. In this case, the address of 80,000 is located in the un-optimizable range, so that the memory operation can not be eliminated. Compared the waveforms from both simulations for the same code, we can tell that the optimized one is 3 times faster than the non-optimized one.
This slice indicates that with a generic design, using different optimization methods, the simulation performance can be increased from 10 times with un-optimized seamless way, up to 10,000 times with time optimization, compared to the full logic simulation.
Seamless Coherent Timers

- Maintain an accurate count of clock cycles when running in time-optimized mode.
- Be programmed to make logic simulator to service timer events appropriately.

When co-verification running in time-optimized mode, the software simulation is decoupled from much slower logic simulation. So the sw and hw simulations are no longer being synchronized. This cause a problem when the software simulation depends on the timing of some devices that are modeled in the logic simulation, such as a real-time operating system depending on an interrupt from a timer. The coherent timers interface allow the existing timer to maintain an accurate count of clock cycles when running in the time-optimized mode. Besides, the coherent timers facility can also be programmed to force the co-verification session to exit the time-optimization mode temporary after a preset timing threshold is approached, so that the logic simulator can be synchronized with software execution. Then the timer model in logic simulator can detect the threshold count and service the timer event appropriately. After that, the co-verification session comes back to time-optimized mode.
Now I will simply talk about how to start to use seamless CVE.
These is the basic steps you need to do for performing hw/sw co-verification with seamless CVE
This is the chart of the demonstration design from the tutorial provided by mentor graphics. And this is also hw8 for our class. However, since there are some problem with the hw tutorial, we still cannot finish that hw with the tutorial. So I will talk about the steps for that homework.
Demo Tutorial

- Setup Environment
- Compile Software
- Invoke Seamless
- Invoke Logic Simulator - ModelSim VHDL
- Configure the Processor - DLX
  - Setup SW simulator/debugger – MDB
  - Map memory instances
  - Define memory access detail
- Start Co-verification Session
  - Enable Data Access Optimization
  - Enable Time Optimization
  - Enable Instruction Fetch Optimization

Invoke Logic Simulator - allows Seamless memory models and processor models within the design to register themselves with Seamless. ---- Bus-interface Model

Setup SW simulator/debugger – ISM + MDB debugger

Map memory instances – memory model templates
It is time for conclusions
Conclusions

- Early Integration
  - Shorten time-to-market
  - Lower cost for verification and correction
- Easy Debugging
  - Control SW execution
  - View/Change memory contents
- Enhancing Performance
  - ISM + BIM
  - CMS + Optimizable Memory
References

- Mentor Graphics, 
  *Seamless Reference Manuals*
- Milan Saini, 
  *Co-Verification Enhances Time to Market Advantage of Platform FPGAs*
- Mike Andrews, 
  *Seamless-izing Memories for Seamless Hardware/Software*
- J. M. Ng, 
  *Enhancing HW/SW Co-Verification for Embedded System with Seamless*
- CIC brady@cic.org.tw, 
  *HW/SW Co-Verification using Mentor Graphics Seamless CVE: A Case Study with AUK system*

References: 3 english and 2 chinese documents.
Any questions??
Thanks