

---

# EEL 5722C

## Field-Programmable Gate Array Design

### Lecture 19: Hardware-Software Co-Simulation\*

Prof. Mingjie Lin



# What is HW-SW co-simulation?

---

- A basic definition:

*Manipulating simulated hardware with software*

- **Goal of co-simulation:** To verify as much of the product functionality, hardware and software, as possible before fabricating the ASIC. (Assume that the ASIC contain some significant amount of software)
- In this lecture: Learn the methods and motivations of hardware-software co-simulation.

# Overview

---

- **In the Past:**  
co-simulation was adopted late in the process, after hardware is deemed to be working and stable. Software developers were often left to develop code for months with severely limited ability to test.
  - Painful integration process, design flaw and could re-spin the silicon
- **Now:**
  - Behavioral model simulation has matured, and simulation tools in general have improved to allow better simulation throughout the development cycle
  - Co-design tools provide project architects with a simulation environment at a very high level of abstraction.

# overview

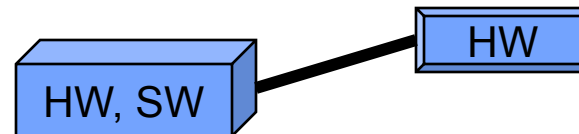
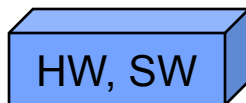
---

- Possible to convert the English description of functionality into a formal specification language, and allow designers to work out the functional split between the hardware and the software in a simulation environment.
- Debug implementation can proceed with event and cycle driven simulators. With codesign tool, can simulate the product before actual implementation. Testing can be real cheap.
- Component integration is done at co-simulation environment with confidence before actual fabrication.

# Simulation components

---

1. Hardware design: Memory, CPU or many ASICs each with one or more CPUs
2. Simulation platform:
  - PC or workstation. Everything exists as process.
  - Hybrid platforms with co-processors: off-load part of the load to co-processor, peripheral and test benches remain in software.




# 3. Emulation

---

- Special simulation environment with hardware
  - runs whole design
  - expensive
  - 10% of real time
  - FPGA arrays may be the hardware
  - allow designers of large products to find a class of problem that cannot be found in simulation
  - can attach to real devices

# 4. Algorithms

---

- Event driven simulation:(gate level simulation)
  - Most accurate as every active signal is calculated for every device during the clock cycle as it propagates
  - Each signal is simulated for its value and its time of occurrence
  - Excellent for timing analysis and verify race conditions
  - computation intensive and hence very slow
-  Cycle-based simulation:
  - Calculate the state of the signals at clock edge(0 or 1)
  - suitable for complex design that needs large number of tests
  - 10 times faster than event driven simulation, 20% area efficient

# Algorithm contd.

---

- Data-Flow Simulator
  - Signals are represented as stream of values without notion of time. Functional blocks are linked by signals. Blocks are executed when signals present at the input.
  - Scheduler in the simulator determines the order of block executions.
  - High level abstraction simulation used in the early stages of verification, typically to check the correctness of the algorithms.



# Simulation's Requirement on Hardware

---

- Most simulators can handle behavioral models except the emulators, that require synthesizable codes.
- Some simulators may not handle HDLs
- Cycle-based simulators can handle asynchronous designs at severe performance penalty
- Choose simulator (s) in the beginning of design stage. You may use multiple simulators concurrently in a project.
- Example: You may use emulator for a problem that requires timing until few hundred microseconds of failure point, then export the state of the system into a simulator.

# Simulator's requirement on Software

---

- Simulation environment has effects on application software
  - Programmers certainly need alternate version of application that do not have user interface code or any references to chips that is not part of the simulation environment.
  - Reduce size of functionality and tables for speed.
  - Example: consider simulating a 500 MHz processor that initializes a 8KB table, that might take few minutes. Such trivial tasks together consume time but do not add to the quality of the simulators.

# Simulation Methods

---

- co-design is a way to simulate at a very high level of abstraction, prior to the actual implementation. These simulations follow the theme of trading details for run-time speed.
- By creating a functional model which can be tested, system designers can make sure the requirements are clear.
- making a single model of both hardware and software functionality, the design boundary between the two is effectively removed.
- Having a running model also allows engineers to test different hardware/software functionality splits for performance and get some rough timing estimates for various ideas.
- Running a functional model also allows engineers to find fundamental bugs in the design before implementing them. Can reuse for performance updates.

# Co-simulation methods

---

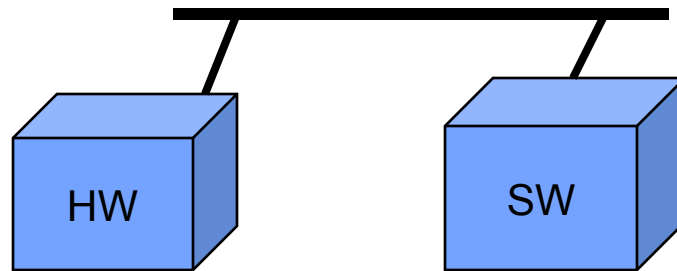
- POLIS: from UC Berkeley
  - Cadence's Cierto VCC is based on ideas from POLIS.
- Synopsys's COSSAP and Eaglei tools: promise a way to check the implementation against the original algorithmic specification for function equivalence.
- But the standard method of co-simulation is to run software directly on simulated hardware. It is implied that the CPU is part of the ASIC. Thus CPU is simulated at the same level as other hardware
  - Good if your purpose is to design the CPU. However, if you use a core from the vendor, you are wasting valuable simulation resources.

# Co-simulation methods (contd)

## Heterogeneous co-simulation

---

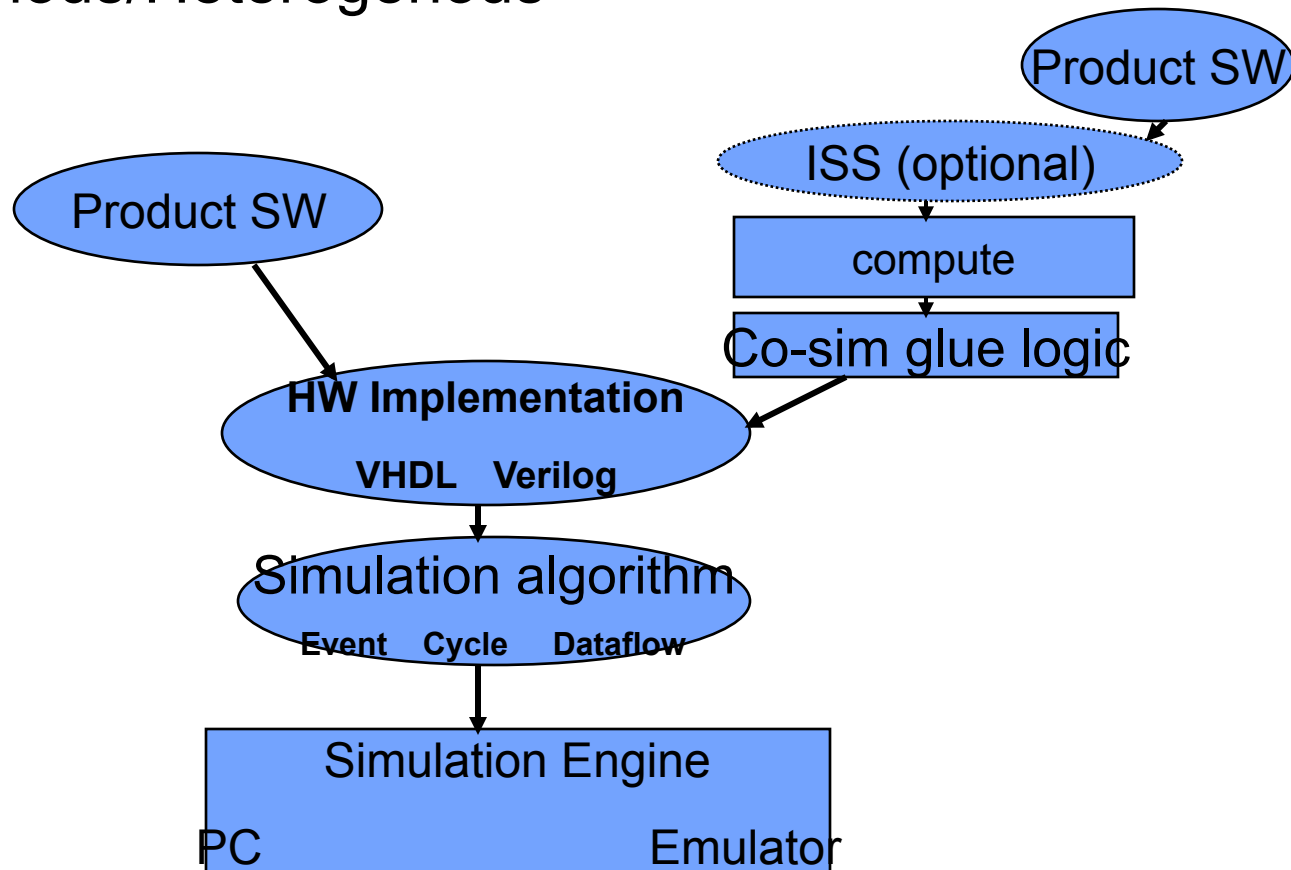
- Network different type of simulators together to attain better speed.



- Claims to be actual co-simulation strategy as it affords better ability to match the task with the tool, simulates at the level of details.
  - For Example: let hw run in many simulators, sw on native PC/ workstation or in instruction-set-simulator (ISS). Simulation tool interfaces all these.

# Heterogeneous co-simulation

Homogenous/Heterogenous



# Heterogeneous cosimulation

---

- How about performance?
  - Complex enough to describe any situation
  - Proponents: since software not running hardware simulation speed, in actual the performance will be more.
    - How fast the software running when not doing hardware related task?
  - If target CPU is not PC, you may use cross compiler
  - When software runs directly on PC/WS, runs at the speed of WS
  - When software can not run directly as processes on WS, you need instruction set simulator ( ISS interprets assembly language at instruction level as long as CPU details are not an issue)
    - ISS usually runs at 20% of the speed of actual or native processes.

# Hardware density of Heterogeneous simulation

---

- How much time software accesses hardware?
- Hardware density depends on applications and with in an application.
- In loosely coupled CPU system, the block responsible for hardware initializations has 30% instructions to access the hardware.
- In tightly coupled system, every memory reference could go through simulated hardware.
- In general hardware density is important for simulation speed.
- The base hardware and tools that communicate between the heterogenous environment can attribute to the speed too.
- If simulation is distributive (most often it happens these days), the network bandwidth, reliability and speed matters too



# Cosimulation strategies

---

- What you simulate is what you get. Simulation is important for bug free test of the product. The product schedule forces suitable strategies.
- Due to decrease in feature size and increase in die size, more functionality are pushed to hardware ( could never happened in the past). Creates challenges for testing due to increased functionality.
  - Formal design methods, code reviews and code reuse have help. Emulation engine is also of help but expensive.
- For typical strategies, we need to know the thoroughness to test and system's environment. If it involves with health and safety, then detail testing strategy is sought.

# Strategy

---

- Multi-pronged functional test strategy to build levels of assurance
    - Basic initial tests prove functionality and complex tests are built upon working.
    - Any single test method has some coverage hole. Event driven tests are closest to the real hardware but its slowness is coverage hole!
    - Make balance between required test coverage and what might be avoided
  - A simulation strategy might call for the functional specification to be written as a functional model (co-design).
    - Hardware designer could use event driven tests for hardware blocks
    - Software designer could do basic debug using ISS or cross compiler and with fake hardware calls. For detailed functional blocks, software could interface. After, completion of blocks, these can be dropped into the functional model for regression tests.
-

# Strategy

---

- *Simulation speed*: Degrades when real components replace the functional blocks. The simulation speed depends on simulation engine, the simulation algorithm, the number of gates in the design, and whether the design is primarily synchronous or asynchronous
- Low cost cycle based simulation is a good compromise. Since it can not test physical characteristic of a design, event driven simulator may be used in conjunction.
- Cycle based simulators and emulators may have long compilation. Hence, not suitable for initial tests that needs many changes.
- Event driven and cycle based simulators have fairly equal debugging environments, all signals are available at all times. Emulators on the other hand, require the list of signals to be traced to be declared at compilation time

# Strategy

---

- If the next problem can be found in a few microseconds of simulated time, then slower simulators with faster compilation times are appropriate.
- If the current batch of problems all take a couple hundred milliseconds, or even seconds of simulated time, then the startup overhead of cycle based simulation or even an emulator is worth the gain in run time speed.
- How about the portability of test benches?
- Test after fabrication?
  - Fast simulators are useful. Track down the hardware fault is difficult. May patch the problem so as to make the problem reappear easily unless regression tests.

# Strategy

---

- To determining which parts of the system software to run and how much software debug can be done without the hardware.
- Software engineer need to go through the code and disable functionality which is too costly for simulation, or if the sequence is important, find ways to reduce its execution time.
- The degree of fidelity between the simulated environment and the real world is both a requirement of simulation and a constantly shifting target throughout the simulation effort

# Summary

---

- Issues and trade off discussed
- The use of HDLs by programmers and logic designers is good sign of convergence.
- Cosimulation is crucial for tightly coupled hardware and software ASICs.
- Every project is different with varying objectives. Hence choose the strategy as required.

# Final issues

---

- Come by my office hours (right after class)
- Any questions or concerns?