# Explain the fundamental key concepts, motivations…

* HW/SW co-design is designing the HW and SW together to solve a given problem.
  + (system level design, Key concepts: concurrency, integrated (HW/SW meet goal together), Modelling (simulation)).
  + Platform based is using existing HW and making the system fit (very common and works well if the platform is suitable)
  + Show timeline with platform (home developed ASIC) vs. codesign.
  + HW/SW co-design:
    - Implementation can start sooner as we do not have to wait for the HW
    - Testing can be done before silicon is done allowing “cheap” re-spin.
    - Mapping of processes to HW and SW can be postponed until proper data is available to make a decision on.
    - HW/SW co-synthesis and co-simulation allow simulation of entire system.
    - Diciplines:
      * Specification (use-case + non-functional, preferably auto-generatable to SystemC)
      * System modeling and abstraction (TLM, SystemC)
      * Partitioning and Design Space Exploration
        + Mapping and metrics.
      * Performance estimation and analysis
      * Validation and verification
        + Validation: Are we doing the right thing?
        + Verification: Are we doing the thing right?
  + Problems:
    - Anything which do not match an existing platform.
    - Anything which has performance requirements that warrant HW-mapping.
    - Buzz-word appeal (the boss heard about it) – like SOAP.

# Explain embedded processing cores like soft, firm and hard-cores

* Before processing cores let’s talk about cores in general:
  + Soft
    - HDL, no optimization
  + Firm
    - HDL with optimization for given target.
  + Hard
    - ASIC (may be embedded in FPGA), fully optimized but fixed.
* Processing cores
  + A processing core (e.g. ARM or PowerPC) may be fully soft core, firm (Altera NIOS II) or hardcore (Directly implemented within the FPGA silicon, Altera Excalibur)).
* HW/SW design flow.
  + SoPC may be used for prototyping before an ASIC or as final design
  + Design flow is individual, but a favored is HW/SW codesign (see 1).
  + Tools may be supplied by the FPGA producer (Altera design sweet) or 3-party or even open source or a combination.
    - IP components (RTL HDL).
    - Hardcore processor IS and ISS (simulator).
    - IP Components SystemC library
    - Compiler for Hard, soft or firm core processor.
    - RTL/HDL “compiler”
    - HDL editor (custom instruction)
    - BSP/HAL/OS for processor core.
  + FPGA allows building a prototype earlier and makes changing the HW much easier than a traditional ASIC development.
  + FPGA may be used directly to boost performance in combination with the processing core.
  + FPGA allows for only including the functionality required (smaller physical foot-print and gate count).
  + Fast HW prototyping with existing Softcore IPs.
  + If the higher performance is not needed FPGA is an expensive and power consuming and development-slow solution compared to an existing microcontroller. (FPGA HDL is much slower to write than SW code).
  + ASIC is superior to FPGA, if an existing and matching implementation exist.

# Present a methodology based on UML for a HW/SW co-design…

* Exercise 2
  + Signal-noise cancellation and video MUX and MPEG4 encoding.
* Methodology = language to express elements and relations and a process explaining the which part to use, when and how
  + SystML has more HW than UML, but is still lacking in time modeling (MARTE)
  + Three levels: informal (diagrams only), structural (skeleton generation), executable (SysML -> SystemC -> C+HDL -> HW)
  + With a SysML -> SystemC it can be simulated and verified - TLM.
  + UML profile = UML + stereotypes and description of same. Possibly extra diagrams (SysML)
  + Requirement: Requirement diagram, use cases
  + Structure: BDD (Block Definition Diagram), IBD
  + Behavior: State chart, Activity, sequence
  + Constraints: Parametric
* System level design approach UML-> top down.
  + Y-chart (Behavior -> Structure -> Physical) – (System -> Process -> Logix -> Gate -> Transistor)
  + Draw Y-chart and explain relation to UML and SystemC – possibly RTL and process mapping

# Explain the concept of modeling and different models of computation

* Modeling is about representing a system or subsystem in a way that allows better understanding, simulation, verification (possibly formal) and tracing (possibly).
  + MoC conceptual, abstract description of system behavior.
  + Well-defined formal definition and semantics.
  + Decomposition
* Process-based models
  + Processes + IPC (Kahn Network (unbounded queue), Dataflow (unbounded fifo, DSP app.), SDF (Synchronous DataFlow) as DataFlow with fixed producer/consumer count – statically schedulable.
* State-based models
  + FSM, + data (FSMD (with data)), + Hierachy (SFSMD (Superstate)),+ concurrency (HCFSM (Hiragical Concurrent)) – Formal analysis (FSM), state space explosion.
* Process State Machines (combination of above) – similar to HCFSM + processes and communication channels)
* Design:
  + Processing element and Communication Element mapping. ISS (Instruction Set Simulation for processor simulation), RTL for HW. SystemC stop at BasicBlock level (CAM).
  + OSI model for communication. TLM at different layers of accuracy (until the physical layer (bus protocol, arbitration and wiring) – it is CAM)
* TLM – Transaction Level modeling, system level in Y-chart. Timed TLM is an intermediate version.
* BCAM – Bus Cycle Accurate Model – model accurate to a given layer of communication – should be cycle accurate, but may be less in the beginning (Network/Protocol TLM => BCAM can be fluffy)
* CCAM – Computation Cycle Accurate Model. Add cycle time to the computations (SystemC wait)
* CAM = CCAM + BCAM – simulation here is fully accurate but slow.
* The price of accuracy -> simulation time.
* Mapping SystemC block -> module, process -> thread, interface -> ports/channels (or interfaces for module-local code)

# Give an overview of SystemC and the modeling library…

* In the old days you make a prototype and said make this with these components – it must do the same. SystemC is the same without a physical prototype.
* SystemC is event driven with a simulation kernel handling the scheduling. No guarantee of ordering, but must be reproducible (deterministic).
* Language (not method). Modeling with execution. Test cases. Modelling HW on development PC (before HW exist) – SW may be transferred “directly”. Executable specification.
* SW first the HW (not really SystemC, but may be done in the same framework). Use SW implementation model as check for HW implementation – HDL code is much more complex than SW.
* Model timing in communication and computation to simulate system.
* SystemC -> CHDL (IP Core) to avoid writing the VHDL in hand.
* When SystemC library for parst IP cores grow it becomes far faster, also some IP cores are shipped with SystemC version.
* SystemC is based on C++ (despite the name) and contains:
  + Time, Concurrency, Modules, Processes, Interfaces, Ports, Channels, Events, Event-driven sim. Kernel
  + Data types
* SystemC at the TLM level uses primitive channels (built-in) and is often untimed.
* SystemC at the RTL level include some Bus abstractions to a given degree of accuracy.
  + Modules is blocks
  + Process (PE) is Thread or Method (no wait, static sensitivity)
  + Ports
  + Channels (signal, fifo, buffer)
  + Clocks
  + Events
  + Data Types
* Tools:
  + CoFluent transforms UML+SysML+MARTE to SystemC and runs the code for simulation.
  + ModelSim. SystemC is used in conjunction with ModelSim to achieve a complete do-simulation with both HW (VHDL) and SW (SystemC). To gain a simulation that takes both HW and SW into account. The combination is a good idea if one e.g. has some VHDL code already. Co-verification.
  + Eclipse SystemC plug-in – gives a full simulation environment for unit-test like test cases. Used to validate the system model. The validated SystemC code may be used as a requirement specification to a HW designer.

# Explain in detail the SystemC modeling elements like

* Modules
  + Components (blocks in SysML). Abstracts a piece HW and/or SW – possibly an entire system at the top level.
  + Modules contain all the below + data types and/or functions
* Threads
  + One of two ways to have an active process within a module. The Thread has dynamic (and/or static) sensitivity, meaning that it can wait on different events and for different durations of time.
  + Sensitivity << event/clock (this will be used by wait()).
* Methods
  + The other way to have an active process within a module. Here wait is illegal and only static sensitivity is used when method returns (unlike thread this do not stop the process).
  + Methods cannot be terminated.
  + next\_trigger is a special case for method to allow it to “wait” for an event
* Ports
  + A port is an interface to a channel. The channel has a port in each end.
  + sc\_port, sc\_fifo\_in, sc\_fifo\_out, sc\_in, sc\_out
* Channels
  + A channel is a communication element (CE).
  + sc\_mutex, sc\_sempahore, sc\_fifo, sc\_buffer, sc\_signal (mapped to sc\_in and sc\_out), sc\_signal\_resolved
  + Architectual level use the sc\_fifo versy much.
  + At the lower level (CAM) – HW level, the sc\_signal is the channel of choice
  + Signal remains unchanged until next delta-cycle, like HW (flip-flop)
* Event
  + The thread and methods may be sensitive to an occurance (notify) on an event.
  + Events may be used for notification between modules. Events are instantaneous and if no one is listening it is lost, hence the notion of notify(SC\_ZERO\_TIME) to postpone to the end of evaluation cycle (simulator kernel detail)
  + Clock is simply a special form of event that is triggered at a specific interval.
  + It is not possible to determine which event triggered a wait to return (or the next\_trigger to execute).

# Explain the application to platform mapping concept …

* Part of System Synthesis
* When the System level structure is completed we have a bunch of Processes connected by Communication channels (possibly as a Program State Machine). The mapping of these to HW (to PEs and CEs) allocates the system.
* Often the platform is know, perhaps with an FPGA, processor, DSP, … all of a given size and speed, and the Processes and Communication Channels should be mapped to this.
* Alternatively the optimal platform for a given set of processes and communication channels may be determined (or attempted determined).
* Using TLM + Networked TLM or lower to estimate the Processes and Communication channels.
* Automatic mapping (only applicable for data flow, not control logic – in the book)
* Load balancing
  + Determine the load of each Process and how they communicate with each-other
  + Determine the capability of each processor in the platform
  + Map the most clock demanding process (maximum clock cycles required) to the process with most processing power remaining.
  + Map the next most demanding to the process with most processing power remaining which does not prevent the communication scheme of the process.
  + Continue until done.
  + This is a very simplistic approach as HW implementation and SW implementation cannot be compared in this way – clock cycles are not just clock cycles.
  + Also it ignores communication.
* Longest processing time
  + Assigns a quality to a given communication path.
  + Instead of just looking at how much processing power the PE has remaining, we calculate the cost of assigning the given process to a given PE and choose the one with the lowest cost.
  + This requires more calculations than load balancing – polynomial time.
  + Re-running with a different starting point not included.
  + Still compare FPGA clock cycles to Processer clock cycles.
  + More accurate as it includes communication.
* Exercise 4
* I prefer a more manual approach where we look at the type of functionality a given Process implements.

# What is HW/SW Co-design partitioning and design space exploration

* Part of System Synthesis.
* Partitioning (also known as allocation and binding). Mapping is (7), as simplified partitioning.
  + Partitioning do not compare apples and oranges (FPGA clock cycles and CPU clock cycles)
* Other factors than just required clock cycles and communication – power consumption, money, physical size, development time, disposal (HazMat), in-house resources (developer skill, IPs, …), …
* FPGA partitioning guidelines
  + High speed (100Hz ->
  + Simple
  + Parallel execution suitable
  + Small memory requirements.
  + Small risk of change
  + Ex. 4
* Design space exploration.
  + Consider all possible mappings based on performance, cost, size, power, …
  + The more criteria the more dimensions.
  + As an example use Performance vs. cost. Here we can choose the cheapest that is fast enough.
  + Pareto optimal points are where a given mapping (cost,performance) is better in either cost, performance or both than any other point (better and equal in all, better in at least one).

# What is the difference between validation and verification?

* Validation: Are we doing the right thing?
* Verification: Are we doing the thing right?
* There are two forms of verification; formal and simulation (test)
* Formal verification with restrictions are possible (only binary FSM, …)
  + Has promise in equivalence checking for regression test – model checking, FSM checking.
* Simulation (Draw specification, + simulator + DUT + stimulus (input) and monitor (test cases).
  + DUT (Design under test)
  + Direct input, indirect input, output
  + Simulating at different levels of abstraction (simulation time vs. accuracy)
  + Non-exhaustive, scalable,
  + Selecting the input
    - Random, pseudo-random.
    - If assertions in code possible (do we have code) it is better.
  + If only some of the design is known (e.g. old style method where SW is not begun until HW is done) may give a faulty result. Co-design may be the answer-
  + The lower the level of abstraction the slower the simulation TLM vs. RTL vs. CAM.
  + UppAll
  + SysML parametrics and SystemC may be used to auto-generate an input/output set.
* Using SystemC is easy, as it is simply C++, and we have access to the values indicating the time spent on a given PE and in communication. We can simply write a test case which injects some input into the system and then verifies the output (values, time)