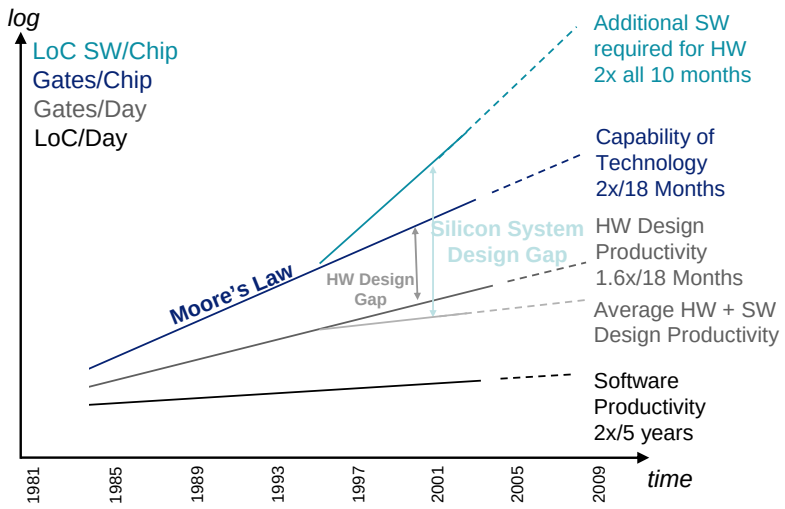
EE382V: Embedded System Design and Modeling

Lecture 10 – System-Level Design Automation

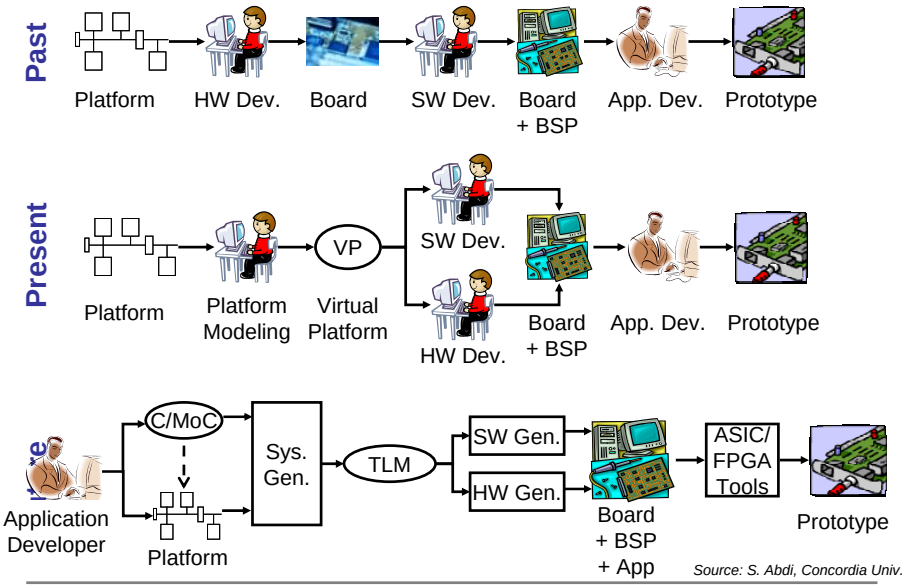
Andreas Gerstlauer
Electrical and Computer Engineering
University of Texas at Austin
gerstl@ece.utexas.edu

Productivity Gaps



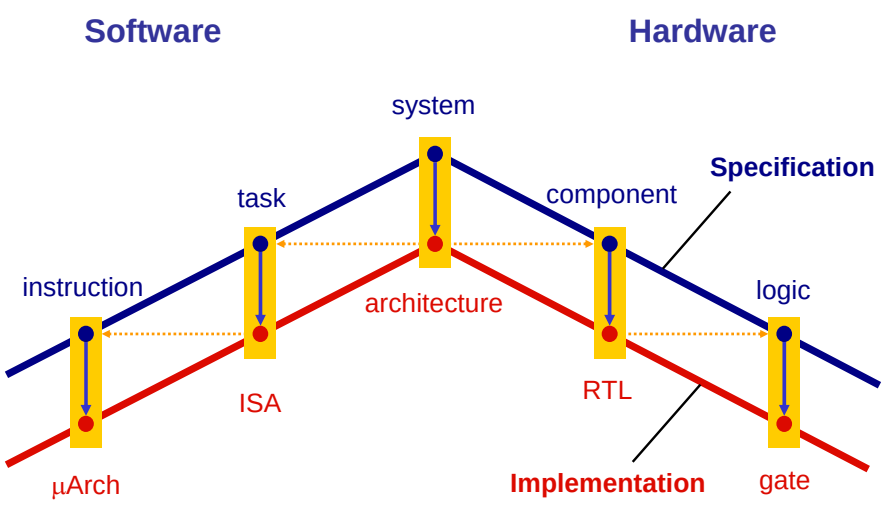
Source: W. Ecker, W. Müller, R. Dömer, *Hardware-dependent Software - Principles and Practice*, Springer 2009.

System Design Process



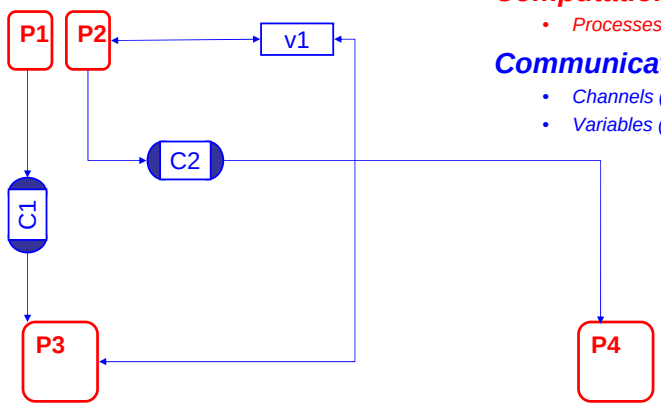
Source: S. Abdi, Concordia Univ.

Double Roof Model



Source: A. Gerstlauer, C. Haubelt, A. Pimentel, et al., "Electronic System-Level Synthesis Methodologies," TCAD, 2009.

Input: Application Behavior Specification

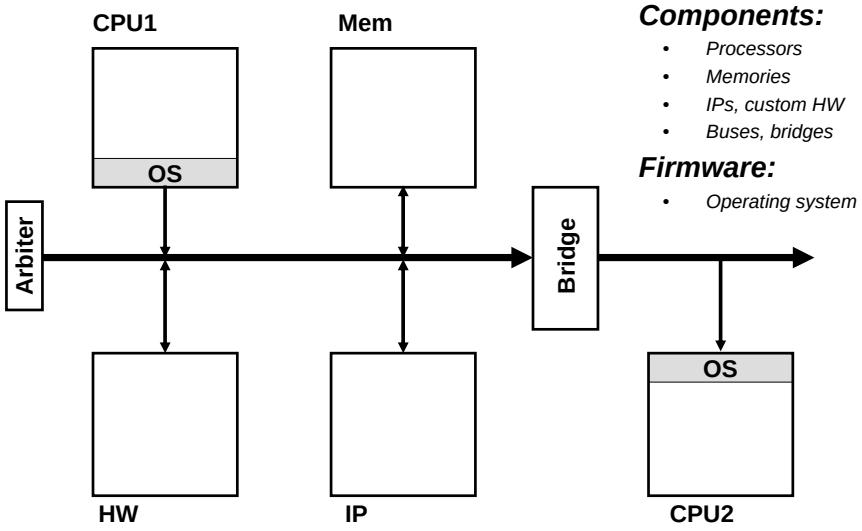


Computation
- *Processes (in C)*

Communication
- *Channels (in C)*
- *Variables (in C)*

EE382V: Embedded Sys Dsgn and Modeling, Lecture 10 © 2010 A. Gerstlauer 5

Input: Platform Architecture Template

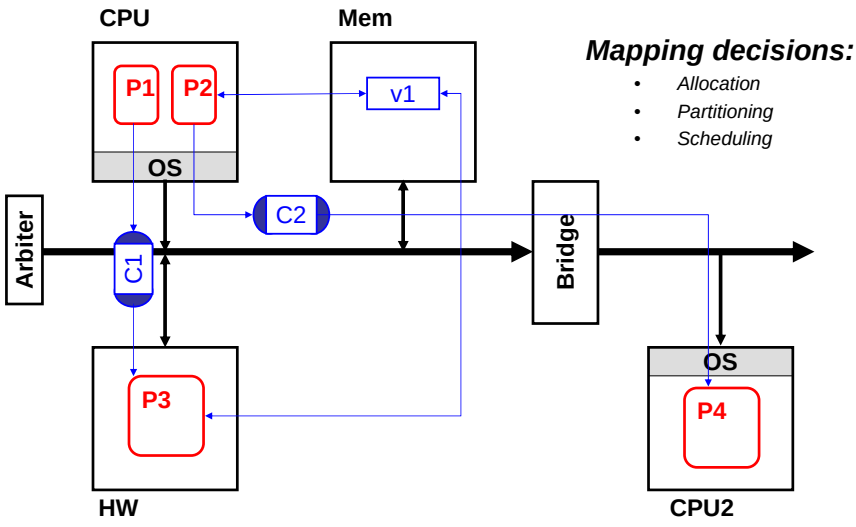


Components:
- *Processors*
- *Memories*
- *IPs, custom HW*
- *Buses, bridges*

Firmware:
- *Operating system*

EE382V: Embedded Sys Dsgn and Modeling, Lecture 10 © 2010 A. Gerstlauer 6

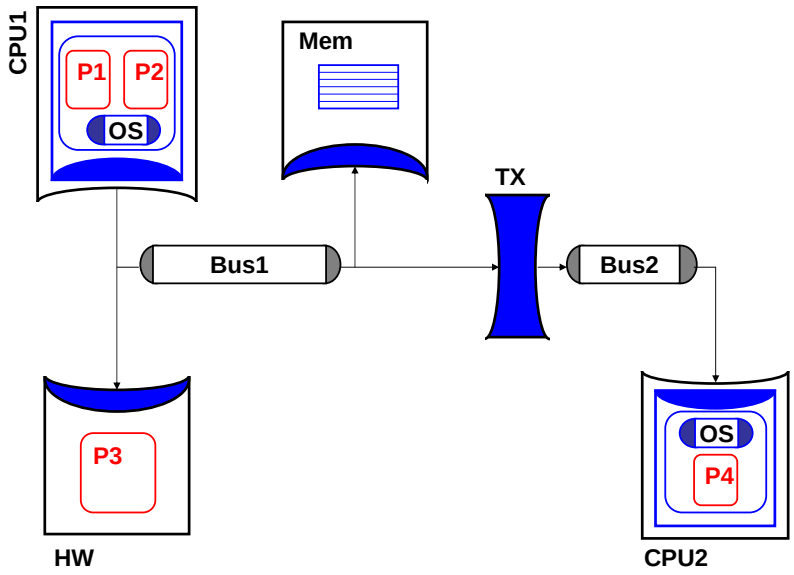
System Definition



System Definition = Application + Platform + Mapping

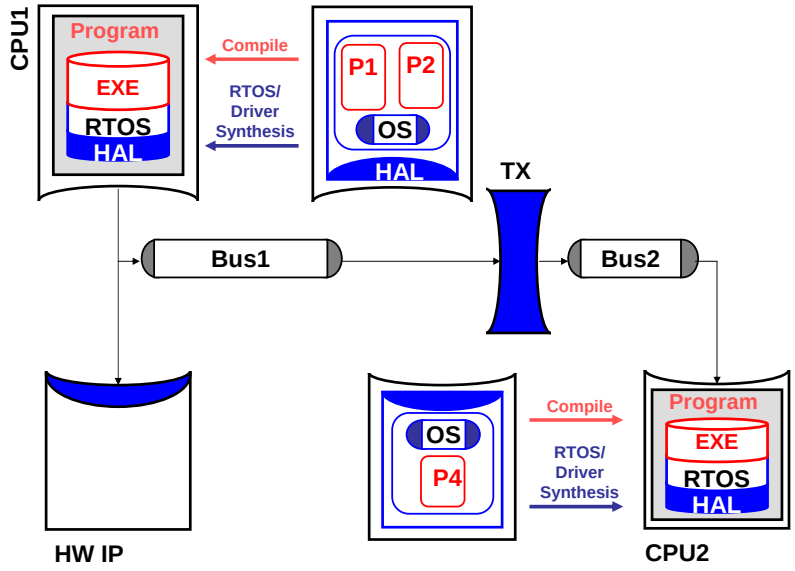
EE382V: Embedded Sys Dsgn and Modeling, Lecture 10 © 2010 A. Gerstlauer 7

Output: Refined TLM



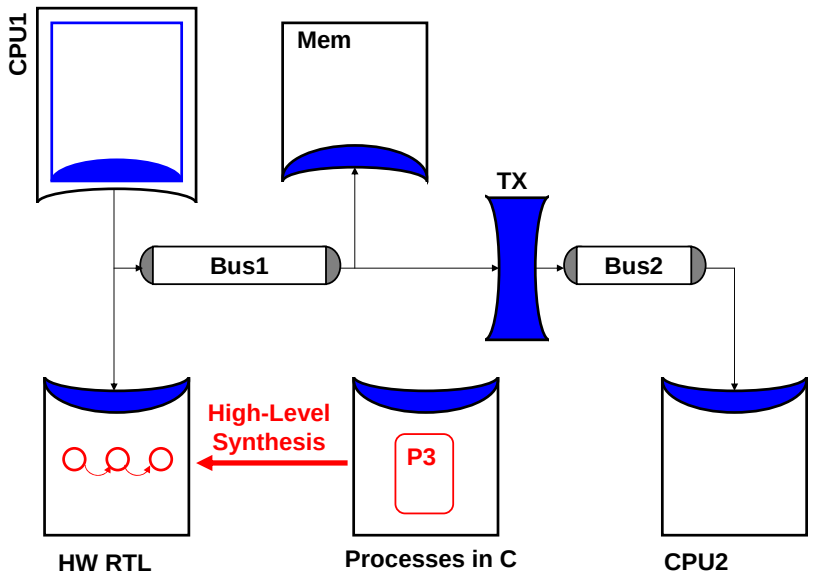
EE382V: Embedded Sys Dsgn and Modeling, Lecture 10 © 2010 A. Gerstlauer 8

Software Synthesis



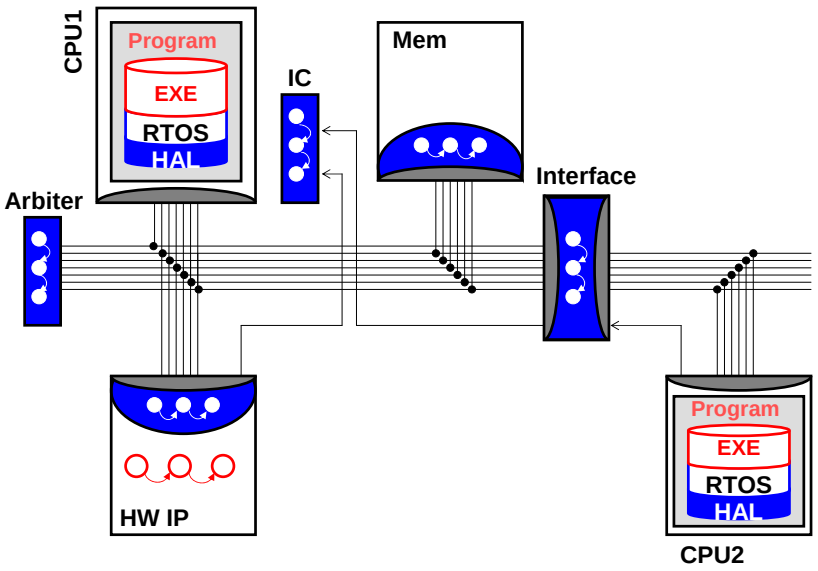
EE382V: Embedded Sys Dsgn and Modeling, Lecture 10 © 2010 A. Gerstlauer 9

Hardware Synthesis



EE382V: Embedded Sys Dsgn and Modeling, Lecture 10 © 2010 A. Gerstlauer 10

Cycle-Accurate Model



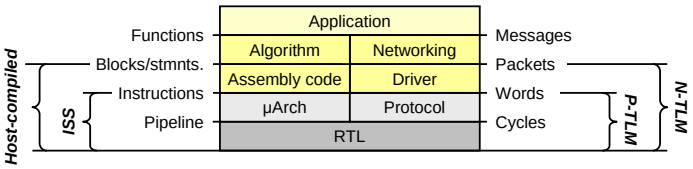
Lecture 10: Outline

- **Modeling**
  - Simulation models
  - Analytical models
- **Synthesis**
  - Optimization and decision-making
  - Design space exploration
- **Verification**
  - Simulation-based methods
  - Formal and hybrid methods

Models and Languages

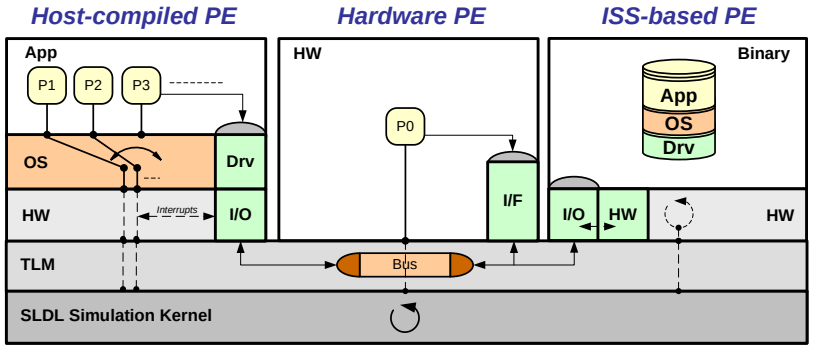
- **Applications: Models of Computation (MoCs)**
  - Process-based models: data-flow & parallelism
    - Threads, Kahn Process Networks (KPN), process calculi: CSP, CCS
    - Dataflow: synchronous (SDF), boolean, cyclo-static, … [Simulink, LabView]
  - State-based models: control-flow & reactivity
    - Finite State Machines (FSM), FSM with data (FSMD)
    - Hierarchical, concurrent FSMs (HCFSM) [StateCharts]
  - Heterogeneous models [UML, Ptolemy]
    - Program state machines (PSM) [SpecC]
- **Architectures: System-Level Design Languages (SLDLs)**
  - Discrete event execution semantics
    - C-based [SpecC, SystemC]
    - Separation of computation and communication
  - Hybrid discrete/continuous cyber-physical modeling
    - Analog/mixed-signal extensions [SystemC-AMS, SpecC-AMS]
    - Differential equation solvers [Matlab]

Modeling Layers



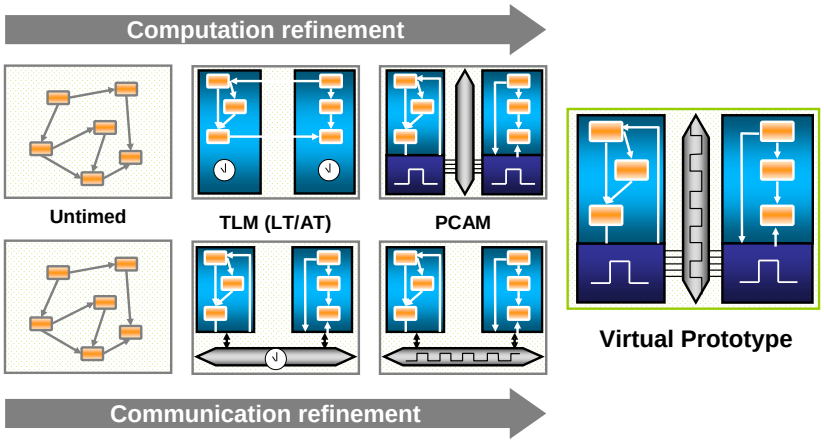
- **Computation**
  - Host-compiled modeling
    - Abstract execution (at source level) above instructions
  - ➢ Functionality and timing
    - ➢ Native execution of functionality
    - ➢ Back-annotation of timing & power estimates
    - ➢ Models of execution environment (OS & HW)
- **Communication**
  - Transaction-level modeling (TLM)
    - Abstract transactions (function calls) above pins and wires
    - Below complete messages
  - ➢ Functionality and timing
    - ➢ Varying levels of granularity
    - ➢ Speed vs. accuracy

Virtual Platform Prototyping



- OS model
  - Multi-tasking scheduler
  - Interrupt handling
  - Drivers
- Application model
  - Functionality
  - Timing
- I/O
  - Bus interface
- Processor hardware model
  - Instruction-set (ISA) functionality
  - Micro-architecture timing

Virtual Platform Prototyping

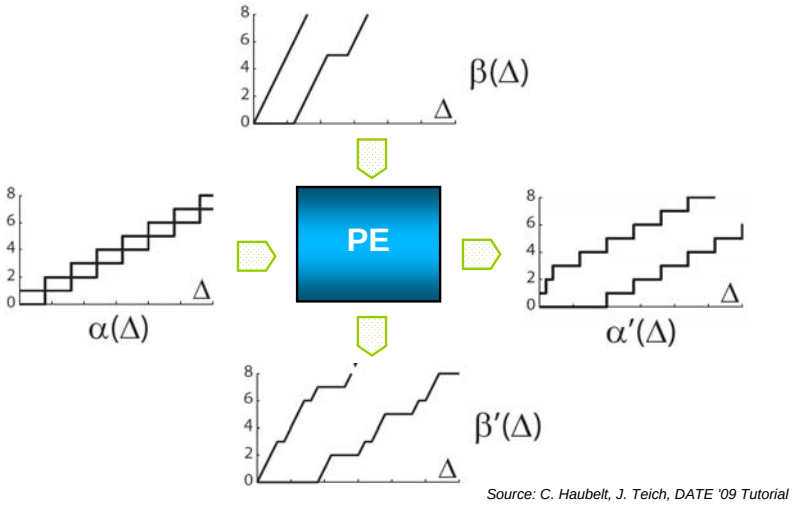


Static Analysis

- **Measurement/simulation does not provide guarantees**
  - Exhaustive simulations not feasible
  - ➢ Compute upper/worst-case bounds (or avg-/best-case)
- **Worst-case execution time (WCET) analysis**
  - Micro-architecture analysis
    - – Computer bounds for each basic execution block
    - – Symbolically simulate statements on processor model (pipeline)
    - ➢ Conservative assumptions for dynamic effects (caches, predictors)
  - Path analysis
    - – Enumerate possible paths and take maximum of block sequence
    - ➢ Possible paths often highly dynamic (loop bounds, false paths)
  - ➢ Basis for back-annotation or static system analysis
    - ➢ Combine static code analysis with dynamic system simulation
    - ➢ Static or dynamic model of inter-process cross-dependencies

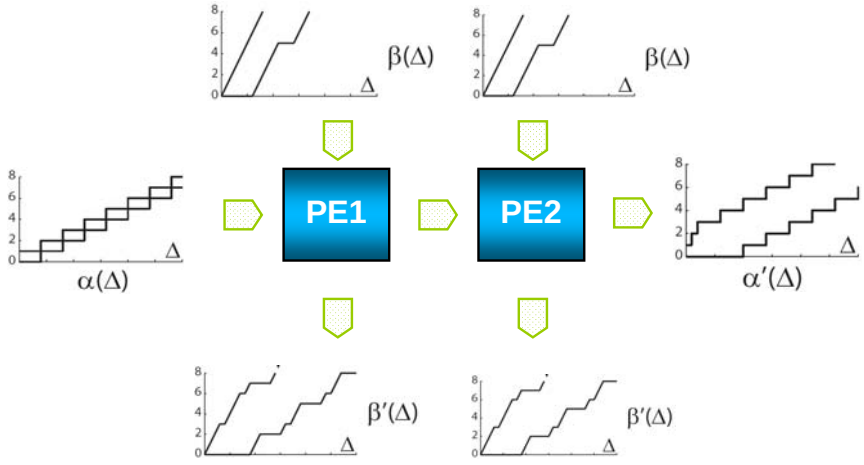
Analytical System Evaluation

- **Modular Performance Analysis (MPA)**
  - Network calculus, real-time calculus (RTC)



Source: C. Haubelt, J. Teich, DATE '09 Tutorial

MPSoC Analysis with MPA

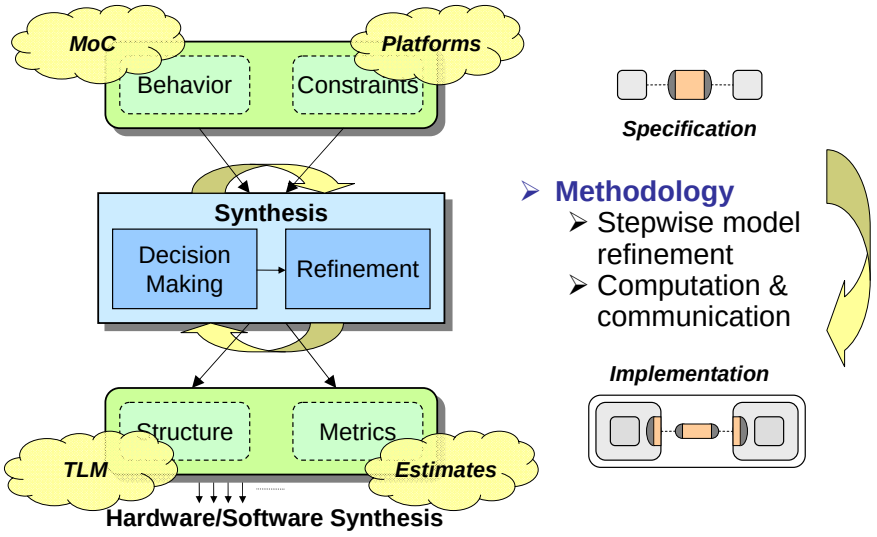


Source: C. Haubelt, J. Teich, DATE '09 Tutorial

Lecture 10: Outline

- ✓ Modeling
  - ✓ Simulation models
  - ✓ Analytical models
- **Synthesis**
  - Optimization and decision-making
  - Design space exploration
- Verification
  - Simulation-based methods
  - Formal and hybrid methods

X-Chart

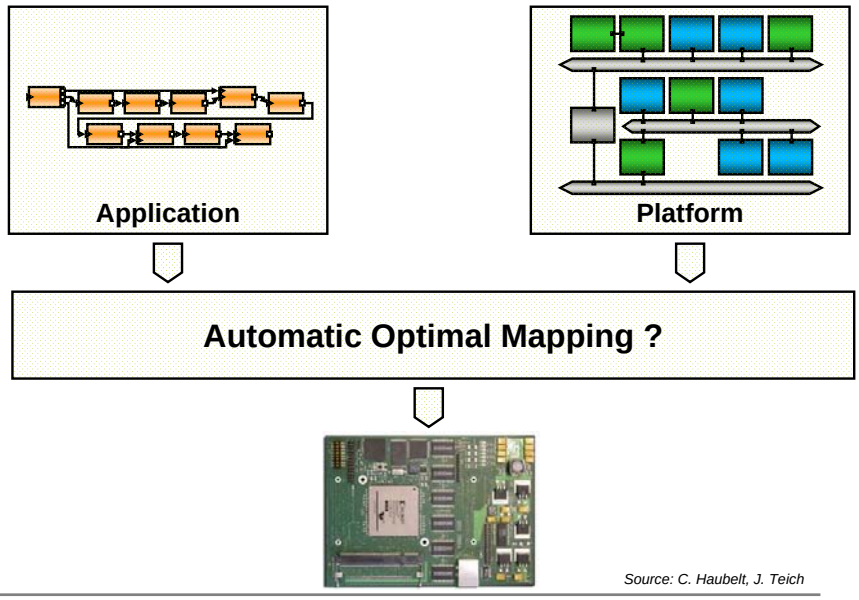


Source: A. Gerstlauer, C. Haubelt, A. Pimentel, et al., "Electronic System-Level Synthesis Methodologies," TCAD, 2009.

System Synthesis

- **Map specification onto architecture**
  - Functionality + constraints ⇨ structure + metrics
- **Synthesis tasks**
  - Allocation
    - Select resources from a platform/architecture template (database)
  - Binding
    - Map processes onto allocated computational resources
    - Map variables onto allocated storage units
    - Route channels over busses, gateways and address spaces
  - Scheduling
    - Determine order of processes bound to the same resource
    - Determine order of transaction routed over the same (arbitration)
  - ➢ Partitioning = (allocation +) binding
  - ➢ Mapping = (allocation +) binding + scheduling

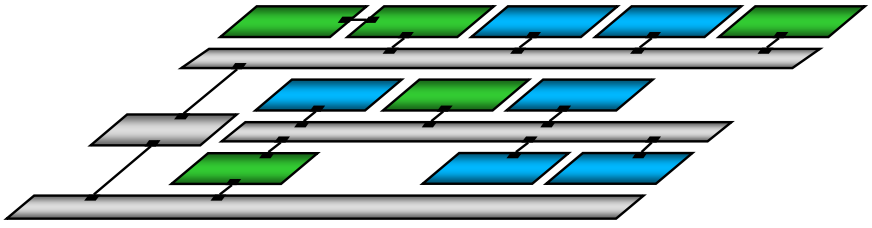
System Synthesis



Source: C. Haubelt, J. Teich

Resource Allocation

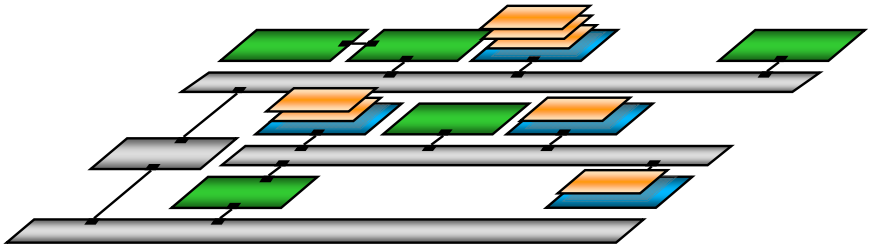
- **Resource allocation, i.e., select resources from a platform for implementing the application**



Source: C. Haubelt, J. Teich

Process Binding

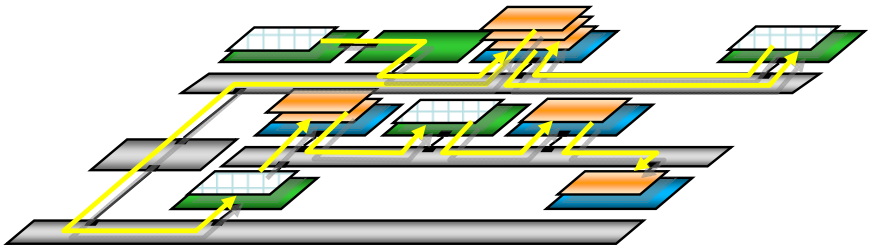
- **Process mapping, i.e., bind processes onto allocated computational resources**



Source: C. Haubelt, J. Teich

Channel Routing

- **Channel mapping, i.e., assign channels to paths over busses and address spaces**



Source: C. Haubelt, J. Teich

Optimization Approaches

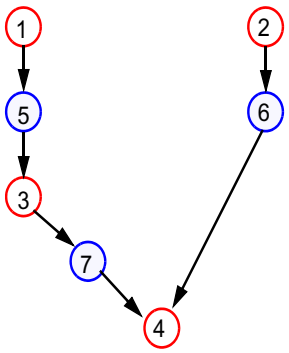
- **Exact methods**
  - Enumeration, exhaustive search
  - (Integer) Linear Programs
- **Heuristics**
  - Constructive
    - Random mapping, hierarchical clustering
  - Iterative
    - Random search, simulated annealing, min-cut (Kernighan-Lin)
  - Set-based ("intelligent" randomized search)
    - Evolutionary Algorithms (EA)
    - Particle Swarm Optimization (PSO)
    - Ant Colony Optimization (ACO)

Source: C. Haubelt, J. Teich

Example (1)

- **Basic model with a data flow graph and static scheduling**

Data flow graph $G_P(V_P, E_P)$



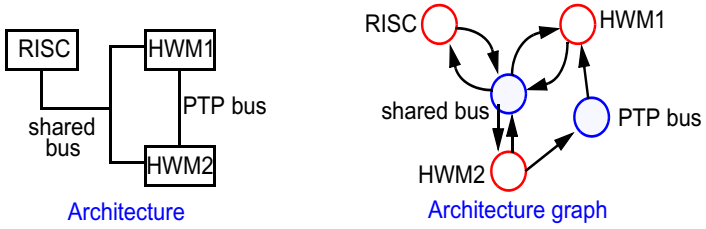
Interpretation:

- V_P consists of functional nodes V_P^f (task, procedure) and communication nodes V_P^c .
- E_P represent data dependencies

Source: L. Thiele

Example (2)

Architecture graph $G_A(V_A,E_A)$:

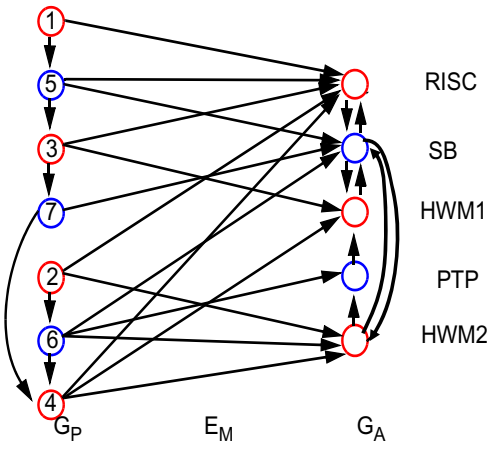


- V_A consists of functional resources V_A^f (RISC, ASIC) and bus resources V_A^c. These components are potentially allocatable.
- E_A model directed communication.

Source: L. Thiele

Example (3)

Definition: A specification graph is a graph $G_S=(V_S,E_S)$ consisting of a data flow graph G_P, an architecture graph G_A, and edges E_M. In particular, $V_S=V_P \cup V_A$, $E_S=E_P \cup E_A \cup E_M$



Source: L. Thiele

Example (4)

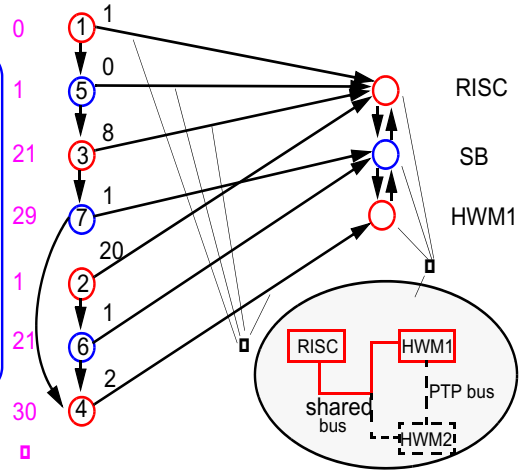
Three main tasks of synthesis:

- Allocation ▯ is a subset of V_A.
- Binding ▯ is a subset of E_M, i.e., a mapping of functional nodes of V_P onto resource nodes of V_A.
- Schedule ▯ is a function that assigns a number (start time) to each functional node.

Source: L. Thiele

Example (5)

Definition: Given a specification graph G_S an implementation is a triple (▯,▯,▯), where ▯ is a feasible allocation, ▯ is a feasible binding, and ▯ is a schedule.



Source: L. Thiele

Cost Functions

- **Measure quality of a design point**
  - may include C … system cost in [$]
    L … latency in [sec]
    P… power consumption in [W]
  - requires *estimation* to find C, L, P
- **Example: linear cost function with penalty**

$$f(C, L, P) = k_1 \cdot h_C(C, C_{max}) + k_2 \cdot h_L(L, L_{max}) + k_3 \cdot h_P(P, P_{max})$$

  - h_C , h_L , h_P … denote how strong C, L, P violate the design constraints $C_{max}, L_{max}, P_{max}$
  - k_1 , k_2 , k_3 … weighting and normalization

Source: L. Thiele

Integer Linear Programming (ILP)

- **ILP formulation of multi-processor SDF mapping**
  - Binding and scheduling decision variables
    - $A_{i,j} \in \{0,1\}$: Actor i mapped to processor j
    - $S_i(t), E_i(t)$: Number of started/ended executions of actor i till time t
  - Constraints
    - Unique actor mapping: $\sum_j A_{i,j} = 1$
    - Actor execution time: $S_i(t) = \sum_j A_{i,j} E_i(t + d_{i,j})$
    - Token balance equations: $c_{i1,i2} S_{i2}(t) \le p_{i1,i2} E_{i1}(t)$
    - Sequential (non-overlapping) execution: $\sum_j A_{i,j} (S_i(t) - E_i(t)) \le 1$
  - Cost function
    - Minimize linear combination of latency, throughput & cost

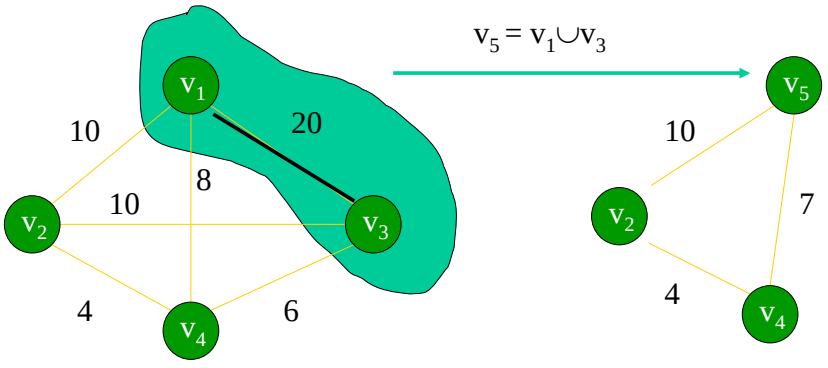
➢ **Exact & optimal, but NP-hard (exponential complexity)**

Constructive Methods

- **Random mapping**
  - Each object is assigned to a block randomly
- **Hierarchical clustering**
  - Stepwise grouping of objects
  - Closeness function determines how desirable it is to group two objects
- ➢ **Constructive methods**
  - Often used to generate a starting partition for iterative methods
  - Show the difficulty of finding proper closeness functions

Source: L. Thiele

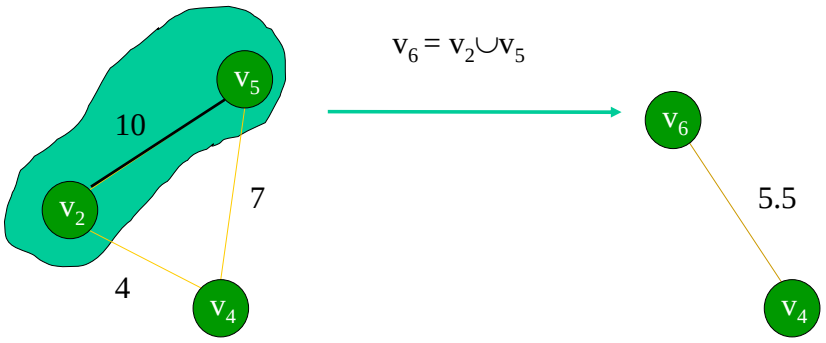
Hierarchical Clustering - Example (1)



Closeness function: arithmetic mean of weights

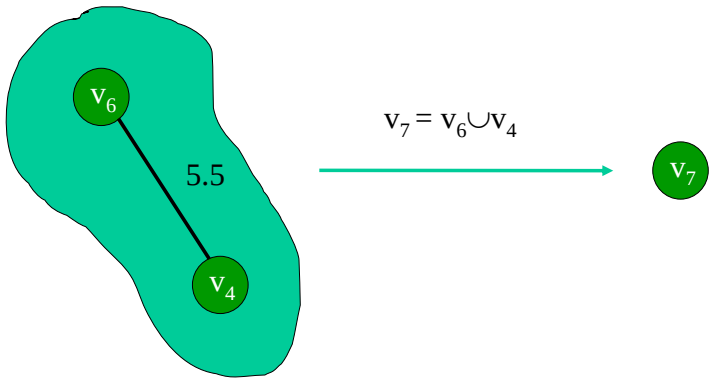
Source: L. Thiele

Hierarchical Clustering - Example (2)



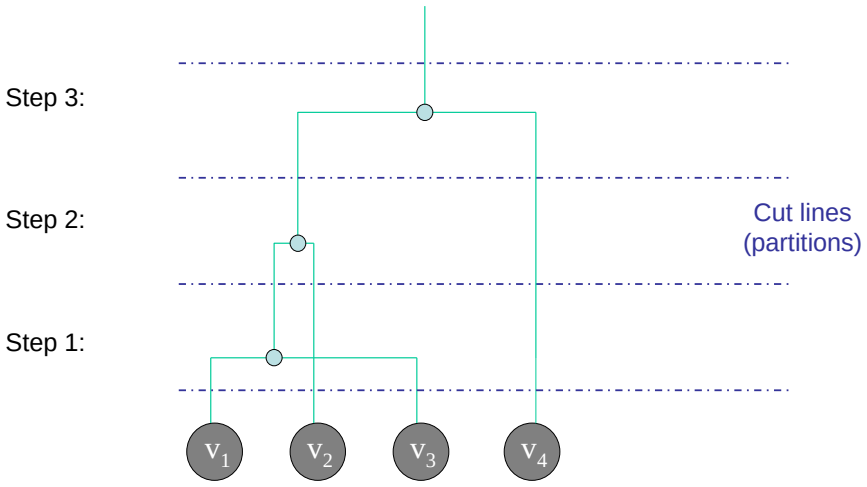
Source: L. Thiele

Hierarchical Clustering - Example (3)



Source: L. Thiele

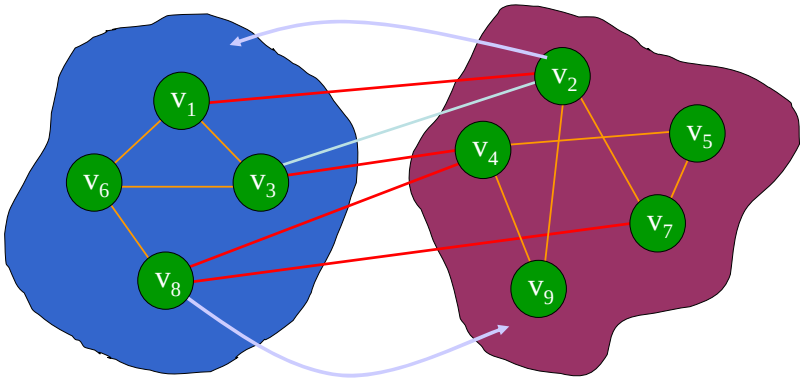
Hierarchical Clustering - Example (4)



Source: L. Thiele

Iterative Methods - Kernighan-Lin (1)

- **Simple greedy heuristic**
  - Until there is no improvement in cost: re-group a pair of objects which leads to the largest gain in cost



Example: Cost = number of edges crossing the partitions
Before re-group: 5 ; after re-group: 4 ; gain = 1

Source: L. Thiele

Iterative Methods - Kernighan-Lin (2)

- **Problem**
  - Simple greedy heuristic can get stuck in a local minimum.
- **Improved algorithm (Kernighan-Lin)**
  - As long as a better partition is found
    - From all possible pairs of objects, virtually re-group the "best" (lowest cost of the resulting partition); then from the remaining not yet touched objects virtually re-group the "best" pair, etc., until all objects have been re-grouped.
    - From these *n/2* partitions take the one with smallest cost and actually perform the corresponding re-group operations.

Source: L. Thiele

Iterative Methods - Simulated Annealing

- **From Physics**
  - Metal and gas take on a minimal-energy state during cooling down (under certain constraints)
    - At each temperature, the system reaches a thermodynamic equilibrium
    - Temperature is decreased (sufficiently) slowly
  - Probability that a particle "jumps" to a higher-energy state:

$$P(e_i, e_{i+1}, T) = e^{\frac{e_i - e_{i+1}}{k_B T}}$$

- **Application to combinatorial optimization**
  - Energy = cost of a solution (cost function)
  - Iteratively decrease temperature
    - In each temperature step, perform random moves until equilibrium
    - Sometimes (with a certain probability) increases in cost are accepted.

Source: L. Thiele

Iterative Methods - Simulated Annealing

- **Cooling Down**
  - temp_start = 1.0
  - temp = α • temp (typical: $0.8 \leq \alpha \leq 0.99$)
  - Terminate when temp < temp_min or there is no more improvement
- **Equilibrium**
  - After defined number of iterations or when there is no more improvement
- ➢ **Complexity**
  - From exponential to constant, depending on the implementation of the cooling down/equilibrium functions
  - The longer the runtime, the better the quality of results
  - Typical: construct functions to get polynomial runtimes

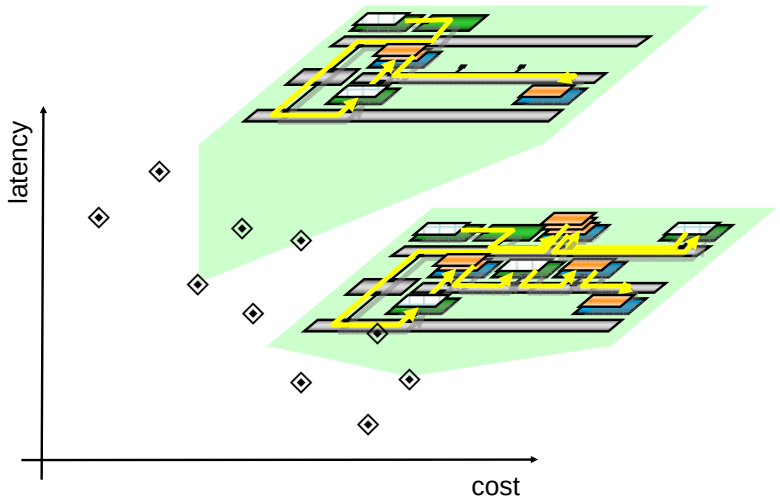
Source: L. Thiele

Design Space Exploration

- **Multi-objective optimization**
  - In general, several solutions (implementations) exist with different properties, e.g., area and power consumption, throughput, etc.
  - Implementations are often optimized with respect to many (conflicting) objectives
  - Finding best implementations is task of Multi-Objective Optimization
- ➢ **Exact, constructive & iterative methods are prohibitive**
  - ➢ Large design space, multiple objectives, dynamic behavior
- ➢ **Set-based approaches (EA, ACO, PSO)**
  - ➢ Randomized, problem independent (black box)
  - ➢ Often inspired by processes in nature (evolution, ant colonies, diffusion)

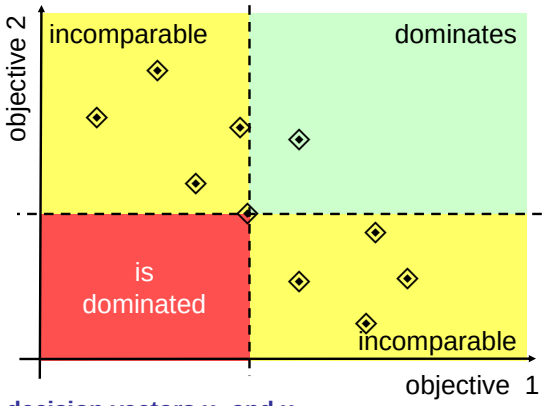
Source: C. Haubelt, J. Teich

Objective Space



Source: C. Haubelt, J. Teich

Pareto Dominance

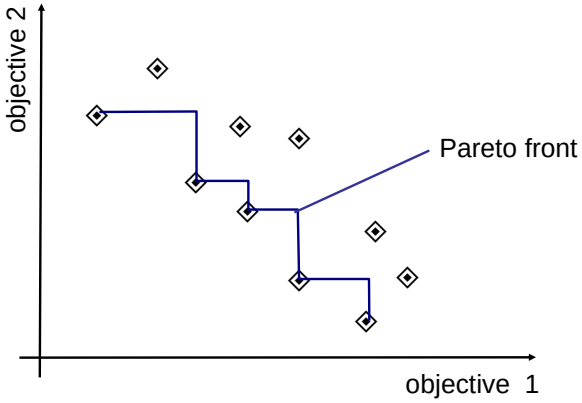


- **Given: two decision vectors x_1 and x_2**
  - $x_1 \succ\succ x_2$ (strongly dominates) if $\forall i: f_i(x_1) < f_i(x_2)$
  - $x_1 \succ x_2$ (dominates) if $\forall i: f_i(x_1) \leq f_i(x_2) \wedge \exists j: f_j(x_1) < f_j(x_2)$
  - $x_1 \sim x_2$ (indifferent) if $\forall i: f_i(x_1) = f_i(x_2)$
  - $x_1 \| x_2$ (incomparable) if $\exists i,j: f_i(x_1) < f_i(x_2) \wedge f_j(x_2) < f_j(x_1)$

Source: C. Haubelt, J. Teich

Pareto Optimality

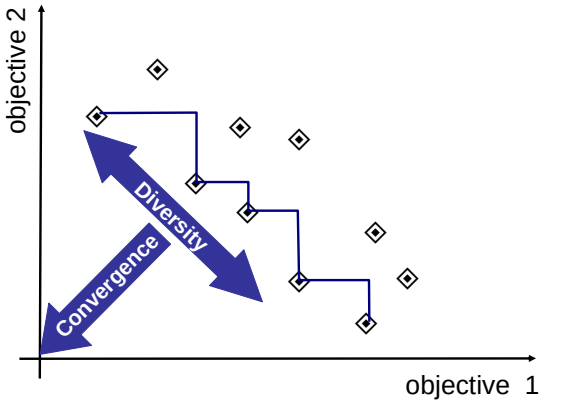
- **Set of all solutions *X***
- **A decision vector *x* ∈ *X* is said to be *Pareto-optimal* if ∄*y* ∈ *X*: *y* ≻ *x***



Source: C. Haubelt, J. Teich

Optimization Goals

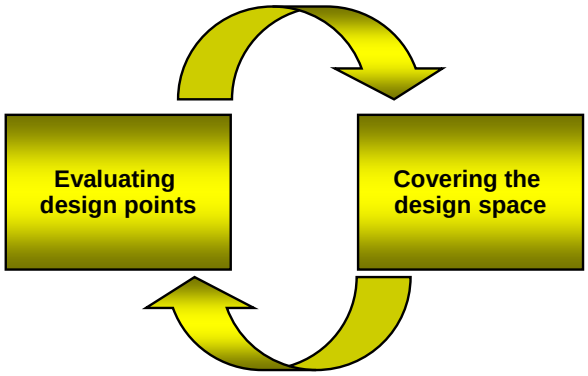
- **Find Pareto-optimal solutions**
- **Or a good approximation (convergence, diversity)**
- **With a minimal number of iterations**



Source: C. Haubelt, J. Teich

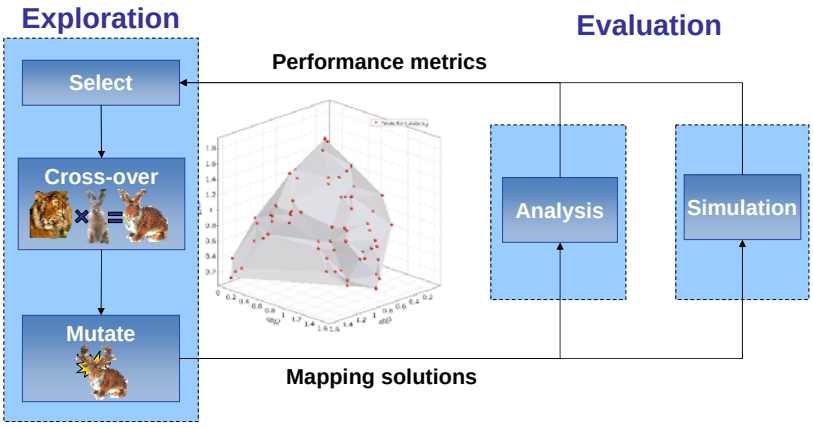
Design Space Exploration

- **Design Space Exploration is an iterative process:**
  - How can a single design point be evaluated?
  - How can the design space be covered during the exploration process



Source: C. Haubelt, J. Teich, Univ. of Erlangen-Nuremberg

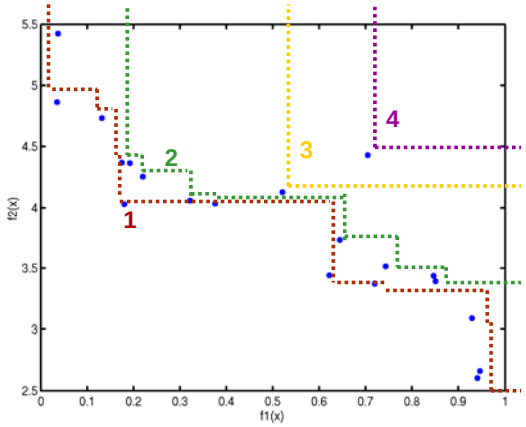
Multi-Objective Evolutionary Algorithm



- **Randomized, directed/intelligent search**
  - Inspired by evolutionary processes in nature

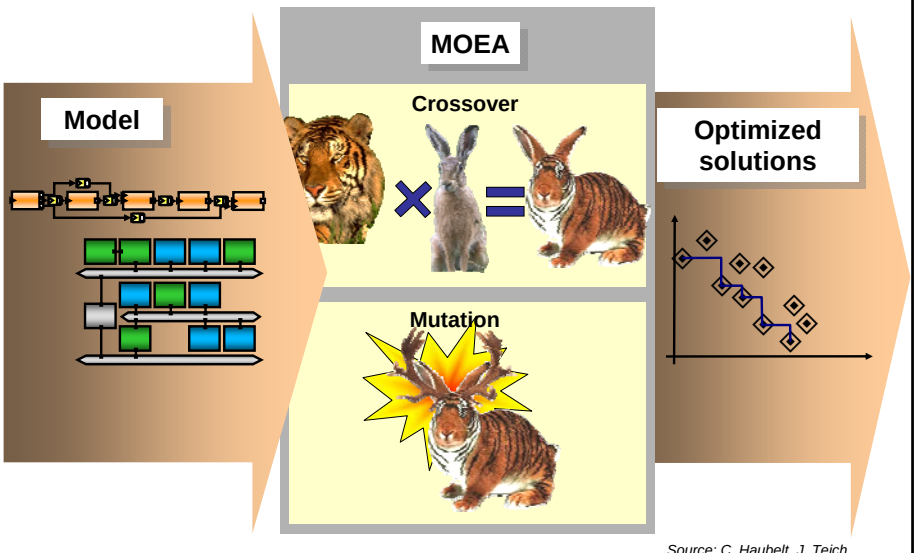
Fitness Selection

- **Pareto ranking**



Source: C. Haubelt, J. Teich

Recombination



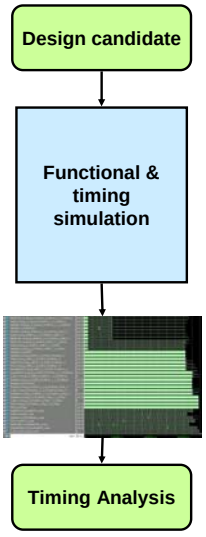
Source: C. Haubelt, J. Teich

Evaluation Approaches

- **Dynamic simulation**
  - Profiling, instruction-set simulation (ISS)
  - ➢ *Long simulation times, corner cases*
  - ➢ *Target vs. host machine-dependent characteristics*
  - ➢ *Limited metrics (performance, operations)*
- **Static analysis**
  - Worst-case execution time (WCET), memory footprint, etc.
  - System cost functions, schedulability & real-time analysis
  - ➢ *Inaccurate bounds, manual interference (false paths)*
- **Combinations**
  - Host-compiled, back-annotated simulation
  - Trace-driven simulation
  - ➢ *Tradeoff between accuracy and speed*

Virtual Platform Simulation

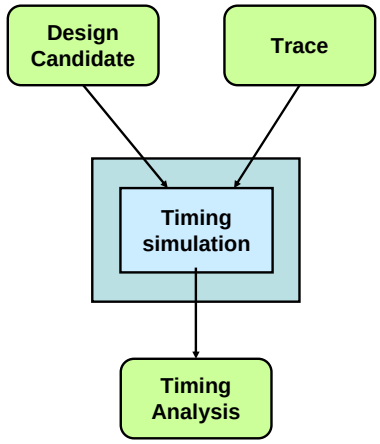
- **Static timing back-annotation**
  - Source level
  - Instructions, basic blocks or functions
  - ➢ Estimation of basic metrics
- **Dynamic system simulation**
  - System description language
  - Simulation host
  - Functionality & timing
  - Generate trace
- **Timing analysis**
  - Latency, throughput, response time, etc.



Source: C. Haubelt, J. Teich

Trace-Driven Simulation

- **Drive simulation via pre-existing, static traces**
  - Traces for system block behavior
- **Examples**
  - Trace-driven simulation
  - Arrival curve extraction from traces
  - Trace generation from arrival curves



Source: C. Haubelt, J. Teich, DATE '09 Tutorial

Lecture 10: Outline

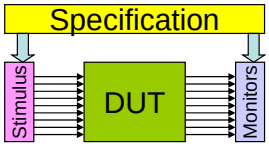
- ✓ **Modeling**
  - ✓ Simulation models
  - ✓ Analytical models
- ✓ **Synthesis**
  - ✓ Optimization and decision-making
  - ✓ Design space exploration
- **Verification**
  - Simulation-based methods
  - Formal and hybrid methods

Design Verification Methods

- **Simulation based methods**
  - Specify input test vector, output test vector pair
  - Run simulation and compare output against expected output
- **Formal Methods**
  - Check equivalence of design models or parts of models
  - Check specified properties on models
- **Semi-formal Methods**
  - Specify inputs and outputs as symbolic expressions
  - Check simulation output against expected expression

Simulation

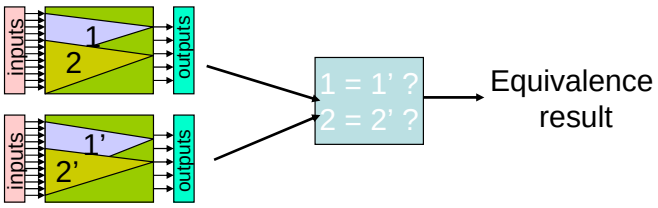
- **Create test vectors and simulate model**
  - Simulation, debugging and visualization tools [Synopsys VCS, Mentor ModelSim, Cadence NC-Sim]



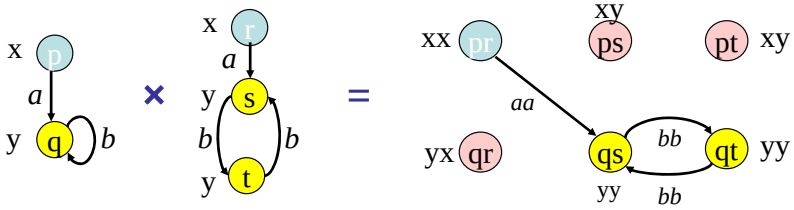
- **Inputs**
  - Specification
    - Used to create interesting stimuli and monitors
  - Model of DUT
    - Typically written in HDL or C or both
- **Output**
  - Failed test vectors
    - Pointed out in different design representations by debugging tools

Equivalence Checking

- **LEC uses boolean algebra to check for logic equivalence**

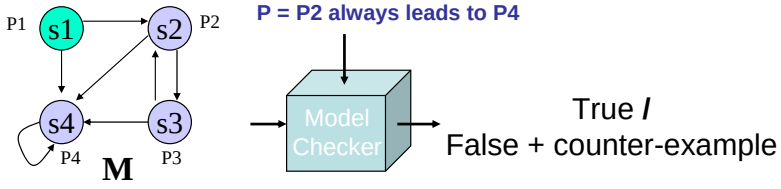


- **SEC uses FSMs to check for sequential equivalence**



Model Checking

- **Model *M* satisfies property *P*? [Clarke, Emerson '81]**
- **Inputs**
  - State transition system representation of *M*
  - Temporal property *P* as formula of state properties
- **Output**
  - True (property holds)
  - False + counter-example (property does not hold)



Lecture 10: Summary

- **Modeling**
  - Simulation vs. analysis
  - Basis for evaluation and exploration
    - Synthesis and verification
- **Synthesis**
  - Single-objective constructive or iterative solutions
  - Multi-objective design space exploration
- **Verification**
  - Formal or semi-formal methods