

ECE382M.20: System-on-Chip (SoC) Design

Lecture 5 – Task Scheduling

Sources:

Prof. Margarida Jacome, UT Austin

Prof. Peter Marwedel, TU Dortmund

Andreas Gerstlauer

Electrical and Computer Engineering

The University of Texas at Austin

gerstl@ece.utexas.edu



The University of Texas at Austin

Chandra Department of Electrical
and Computer Engineering

Cockrell School of Engineering

Lecture 5: Outline

- **Scheduling**
 - Uni-processor scheduling
 - Task graph scheduling
- **Special considerations**
 - Scheduling overheads
 - Interrupts, caches
- **System-level design**
 - MPSoC synthesis

Multiplexing HW/SW Modules

SUBROUTINES
Hierarchical
Sequential, static

COROUTINES
Symmetric
Sequential, static

PROCESSES
Symmetric
Concurrent, dynamic

Modularity
Complexity

ECE382M.20: SoC Design, Lecture 5

© Margarida Jacome, UT Austin

3

Scheduling

- A schedule is a mapping of a set of tasks to start times such that none overlap
 - Optimize throughput, latency (schedule length/makespan)
 - Dependency, real-time (deadline) constraints

$G=(V,E)$

τ

D_t

t

- Static scheduling
 - Fixed start times or fixed order (semi-static)
- Dynamic scheduling
 - Under control of an operating/runtime system

ECE382M.20: SoC Design, Lecture 5

© Peter Marwedel, Dortmund Univ.

4

Uni-Processor Scheduling

➤ Trivial without deadlines

- Makespan is constant, throughput = $1 / \text{makespan}$

• Aperiodic, independent task set

- Earliest Due Date (EDD) to minimize max. lateness

• Periodic, independent task set

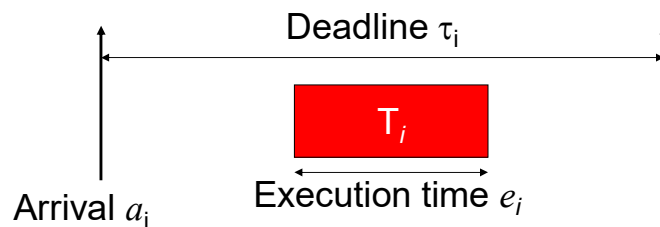
- Maximize CPU utilization while meeting deadlines
- Rate Monotonic Scheduling (RMS) optimal w/ fixed priority
- Earliest Deadline First (EDF) optimal w/ dynamic priority

• Dependent task graph

- Latest Deadline First (LDF) to minimize max. lateness
- Modified EDF* for dynamic schedule

Aperiodic, Independent Task Model

- e_i is execution time of task i
- Arrival time a_i of task i
- Deadline τ_i of task i (→ real-time!)



➤ Waiting time

- Difference between start time s_i and arrival time a_i

➤ Response time

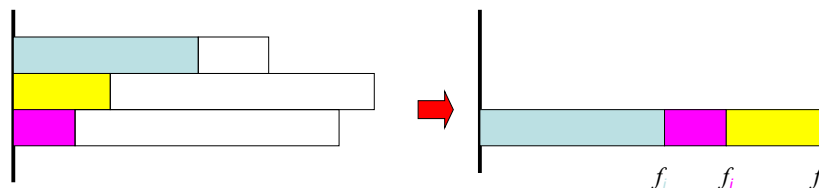
- Difference between finish time f_i and arrival time a_i

➤ Lateness/slack

- Difference between response time r_i and deadline τ_i

Simultaneous Arrival Times

- **Preemption is useless**
 - All tasks arrive at the same time
- **Total schedule makespan/length is fixed to $\sum e_i$**
 - Optimize average waiting time: Shortest Job First (SJF)
 - Optimize lateness under deadlines: real-time scheduling
- **Earliest Due Date (EDD)**
 - Execute task with earliest due date (deadline) first
 - Sort tasks by their (absolute) deadlines



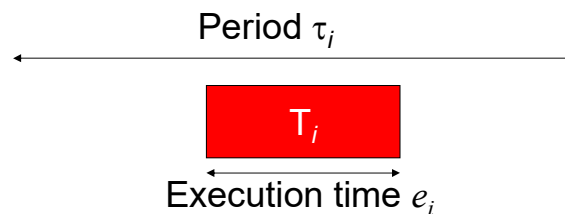
- EDD is optimal to minimize the maximum lateness

Different Arrival Times

- **Requires preemption for optimality**
 - Avg. waiting time: Shortest Remaining Time First (SRTF)
 - Real-time: Earliest Deadline First (EDF)
 - Alternative: Least Laxity (LL) / Least Slack Time First (LSF)
- **EDF and LL/LSF are optimal**
 - Minimizing maximum lateness
 - Differences in response times
- **If preemption is not allowed**
 - All arrival times need to be known to achieve optimality
 - Problem becomes NP-hard
 - EDF remains best online (unknown arrival times) algorithm

Periodic Task Model

- e_i is execution time of task i
- Tasks arrive at regular, periodic times $a_i = \{\tau_i, 2\tau_i, 3\tau_i, \dots\}$
- Deadline τ_i is period of task i (always real-time)



- Waiting time w_i
- Response time r_i
- Lateness/slack $l_i = r_i - \tau_i$

Scheduling Metrics

- How do we evaluate a periodic scheduling policy
 - Ability to satisfy all deadlines ($\rightarrow$ real-time)
 - CPU utilization
 - Percentage of time devoted to useful work
 - Scheduling overhead
 - Time required to make scheduling decision
 - Schedulability
 - Find a schedule iff one exists that satisfies all deadlines
 - Worst-case CPU utilization that guarantees schedulability for any task set
- Constraints
 - Set of tasks T with period τ_i each
 - Response time $r_i = \text{finish time } f_i - \text{arrival time } a_i$
 - Deadline d_i ; $r_i < d_i$, in periodic case often $d_i = \tau_i$

Periodic Task Scheduling

- **Scheduling Policies**

- RMS – Rate Monotonic Scheduling
 - Task Priority = Rate = $1/\text{Period}$
 - RMS is the optimal preemptive *fixed-priority* scheduling policy
- EDF – Earliest Deadline First
 - Task Priority = Current Absolute Deadline
 - EDF is the optimal preemptive *dynamic-priority* scheduling policy

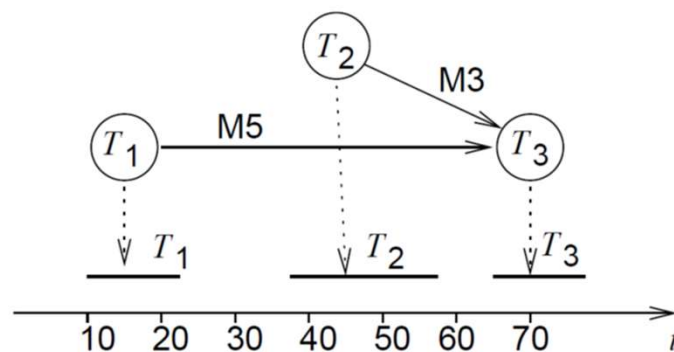
- **Scheduling assumptions**

- Single processor
- All tasks are periodic
- Zero context-switch time
- Worst-case task execution times are known
- No data dependencies among tasks

➤ **RMS and EDF have both been extended to relax these**

Dependent Tasks

- **Task graph**

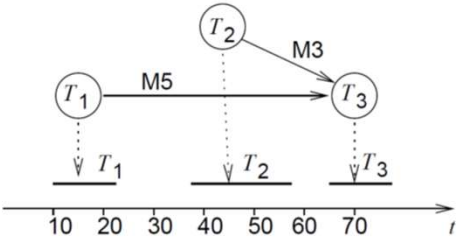


➤ **Periodic or aperiodic**

- All tasks must have same period

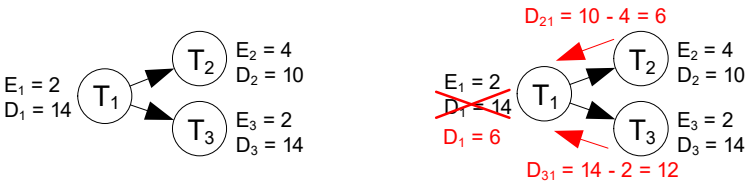
Simultaneous Arrival Times

- **Latest Deadline First (LDF)**
 - Process task graph from sinks to sources
 - Among tasks without successors, insert the ones with the latest deadline into the schedule
 - At runtime, process in generated static reverse order
- **Optimal for uni-processor**
 - Non-preemptive
 - If no local deadlines, just topological sort



Different Arrival Times

- **Modified Earliest Deadline First (EDF*)**
 - Process graph from sinks to sources
 - Propagate deadlines adjusted for execution times
 - Under global time basis (adjusted for arrival times)
 - At each node, deadline = $\min(\text{original}, \text{propagated})$
 - Run from source to sinks in deadline order



- **Optimal for uni-processor**
 - Preemptive

Scheduling Considerations

- **Special extensions**
 - HW accelerators (special task arrival/dependency models)
 - Task execution times (worst-case + dynamic scheduling)
 - Context switch overheads (fold into task execution times)
 - Interrupt overhead (treat as high-priority tasks)
- **What if your set of processes is unschedulable?**
 - Change deadlines in requirements
 - Reduce execution times of processes
 - Get a faster CPU
 - Change the HW/SW partitioning!

Scheduling Overhead

- **Context switch time**
 - Non-zero context switch time can push limits of a tight schedule
 - Hard to calculate effects
 - Depends on order of context switches
 - In practice, OS context switch overhead is small
- **May want to test**
 - Context switch time assumptions on real platform
 - Scheduling policy

What about interrupts?

- **Interrupt overhead**

- Interrupts take time away from processes
- Other event processing may be masked during interrupt service routine (ISR)
- Perform minimum work possible in the interrupt handler



- **Device processing structure**

- ISR performs minimal I/O
 - Get register values, put register values
- Interrupt service process/thread performs most of device function
- Model as high-priority sporadic tasks
 - Frequent aperiodic tasks
 - Turn into periodic tasks via sporadic task server process

Caches & Shared Resources

- **Shared resources can cause additional problems**

- Even if individual processes are well-behaved, processes may interfere with each other
- Worst-case execution time with *bad cache behavior* is usually much worse than execution time with good cache behavior

- **Perform schedulability analysis without caches**

- Take any online performance gains as “free lunch”

- **Timing isolation**

- E.g. cache partitioning

Lecture 5: Outline

✓ Scheduling

- Uni-processor scheduling
- Task graph scheduling

✓ Special considerations

- ✓ Scheduling overhead
- ✓ Interrupts, caches

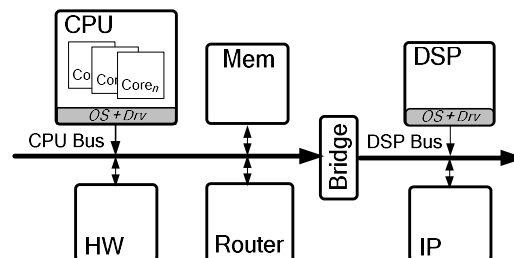
• System-level design

- MPSoC synthesis

Multi-Processor Systems-on-Chip (MPSoCs)

• Multi-processor

- Heterogeneous
- Asymmetric multi-processing (AMP)
- Distributed memory & operating system



• Multi-core

- Heterogeneous or homogeneous or identical
- Symmetric multi-processing (SMP)
- Shared memory & operating system
- Multi-core processors in a multi-processor system

• Many-processor/many-core

- > 10 processors/cores ...

MPSoC Mapping

- **Partitioning & scheduling strongly interact**
 - Restricted homogeneous multi-core scheduling
 - Heterogeneous case is NP-complete in general
 - Exploit parallelism, throughput vs. latency (pipelining)
- **Multi-processor SoC (MPSoC) mapping heuristics**
 - For simple cases, partition first and schedule separately
 - In general, solve partitioning & scheduling jointly
- **Dedicated constructive or iterative heuristics**
 - Static mapping (partitioning & scheduling)
- **Multi-objective MPSoC mapping meta-heuristics**
 - Multiple objectives, dynamic platform effects

ECE382M.20: SoC Design, Lecture 5

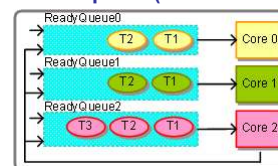
© Margarida Jacome, UT Austin

21

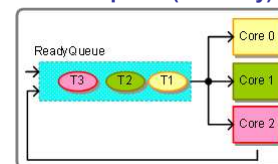
Multi-Core Scheduling

- **Partitioned scheduling**
 - Partition tasks to cores
 - Apply uni-processor scheduling on each core
 - Optional load-balancing
- **Global scheduling**
 - Fixed priorities [G-RMS]
 - Fixed job priority [G-EDF]
 - Dynamic [P-Fair]

Partitioned queue (+ load balancing)



Global queue (+ affinity)



- **Can not account for heterogeneity & dependencies**

ECE382M.20: SoC Design, Lecture 5

© 2026 A. Gerstlauer

22

Multi-Processor Scheduling

- **General job-shop scheduling**
 - Minimize makespan of independent tasks on m processors
 - Classical multi-processor scheduling: atomic jobs, no migration
- **General task graph (DAG) scheduling**
 - Minimize makespan for dependent tasks on m resources
 - Minimize resources under makespan constraint
 - Pipelined variants for periodic task graph invocations
- **Heuristics**
 - Exact: e.g. ILPs
 - Constructive: e.g. list scheduling
 - Iterative: e.g. simulated annealing

Constructive Methods – List Scheduling

- **Greedy heuristic**
 - Process graph in topology order (source to sink)
 - Process ready nodes in order of priority (criticality)
 - List scheduling variants only differ in priority function
 - Priority needs to encode all estimated costs statically
- **Widely used scheduling heuristic**
 - Operation scheduling in compilation & high-level synthesis
 - HLS scheduling (see later lectures)
 - Iterative modulo scheduling for software pipelining
 - Job-shop/multi-processor scheduling
 - Graham's algorithm (optimal online algorithm for ≤ 3 processors)
 - Heterogeneous earliest-finish time first (HEFT)
 - Natural fit for minimizing makespan/latency
 - $O(n)$ complexity

Constructive Methods – List Scheduling

```

l = 0;
i = 0...n: pi ← Idle;
Ready ← Initial tasks (no dependencies);
while (!empty(Ready)) {
    forall pi: status(pi) == Idle {
        t = first(Ready, pi); // by priority
        pi ← (t, l, l + exec_time(t));
    }

    l = min(l + 1, finish_time(pi));

    forall pi: finish_time(pi) == l {
        Ready ← successors(current(pi));
        pi ← Idle;
    }
}

```

ECE382M.20: SoC Design, Lecture 5

© 2026 A. Gerstlauer

25

Iterative Methods

- **Basic principle**
 - Start with some initial configuration (e.g. random)
 - Repeatedly search *neighborhood* (similar configuration)
 - Select *neighbor* as candidate (make a *move*)
 - Evaluate *fitness* (cost function) of candidate
 - Accept candidate under some rule, select another neighbor
 - Stop if quality is sufficient, no improvement, or end time
- **Ingredients**
 - Way to create an initial configuration
 - Function to find a *neighbor* as next candidate (make *move*)
 - *Cost* function
 - Can be analytical (static costs) or simulation (capture dynamic effects)
 - *Acceptance* rule, stop criterion
 - No other insight into problem needed

Source: L. Thiele

ECE382M.20: SoC Design, Lecture 5

© 2026 A. Gerstlauer

26

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)
 - Can use simulation or any other evaluation/estimation model
- Iteratively decrease temperature
 - In each temperature step, perform random moves until equilibrium
 - Increases in cost are accepted with certain probability (depending on cost difference and “temperature”)

Source: L. Thiele

ECE382M.20: SoC Design, Lecture 5

© 2026 A. Gerstlauer

27

Iterative Methods - Simulated Annealing

```
temp = temp_start;
cost = c(P);
while (Frozen() == FALSE) {
    while (Equilibrium() == FALSE) {
        P' = RandomMove(P);
        cost' = c(P');
        deltacost = cost' - cost;
        if (Accept(deltacost, temp) > random[0,1]) {
            P = P';
            cost = cost';
        }
    }
    temp = DecreaseTemp(temp);
}
```

$$\text{Accept}(\text{deltacost}, \text{temp}) = e^{-\frac{\text{deltacost}}{k \cdot \text{temp}}}$$

Source: L. Thiele

ECE382M.20: SoC Design, Lecture 5

© 2026 A. Gerstlauer

28

Iterative Methods - Simulated Annealing

- **Random moves: `RandomMove (P)`**
 - Choose a random solution in the neighborhood of P
- **Cooling Down: `DecreaseTemp ()`, `Frozen ()`**
 - Initialize: `temp_start = 1.0`
 - `DecreaseTemp`: `temp =  $\alpha$  • temp` (typical: $0.8 \leq \alpha \leq 0.99$)
 - Terminate (frozen): `temp < temp_min` or no improvement
- **Equilibrium: `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

Source: L. Thiele

ECE382M.20: SoC Design, Lecture 5

© 2026 A. Gerstlauer

29

Multi-Objective Optimization (MOO)

- **Many (conflicting) optimizations goals**
 - Several optimal solutions exist
 - Different tradeoffs among properties
- **Exact & constructive methods (list scheduling)**
 - Prohibitive, large design spaces, static cost models
- **Iterative single-objective methods (simulated annealing)**
 - Only return a single solution
- **Set-based iterative approaches**
 - Randomized, problem independent (black box)
 - Often inspired by processes in nature (evolution, ant colonies, particle diffusion)

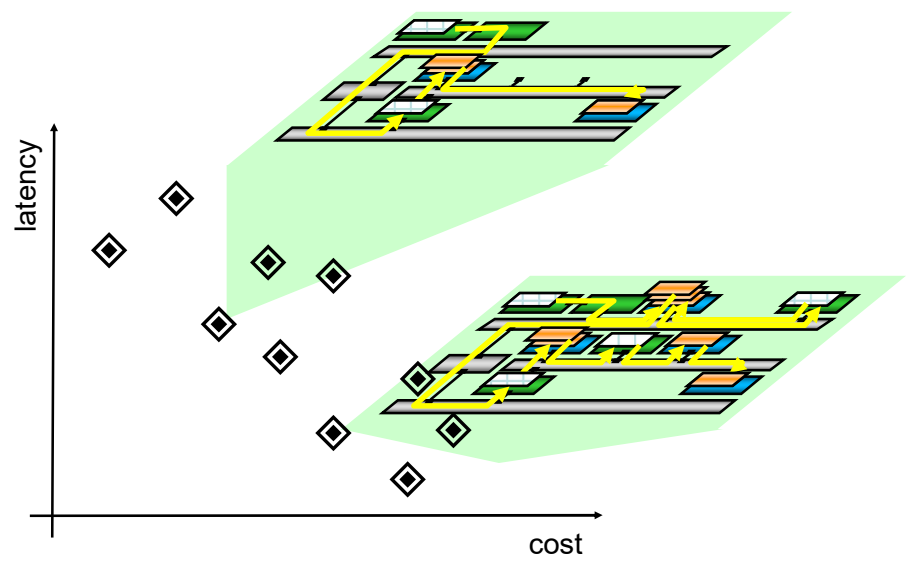
Source: C. Haubelt, J. Teich

ECE382M.20: SoC Design, Lecture 5

© 2026 A. Gerstlauer

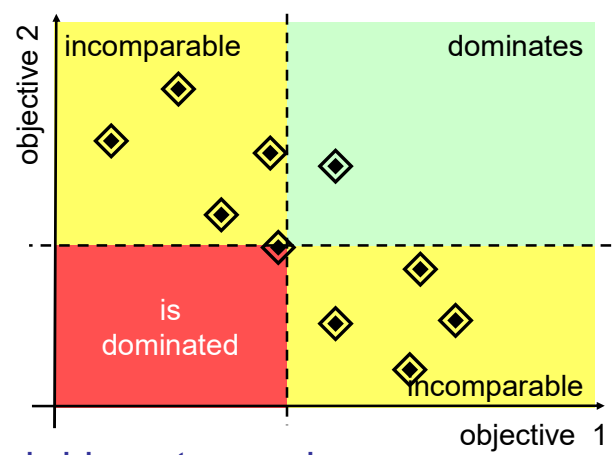
30

Objective Space



Source: C. Haubelt, J. Teich

Pareto Dominance

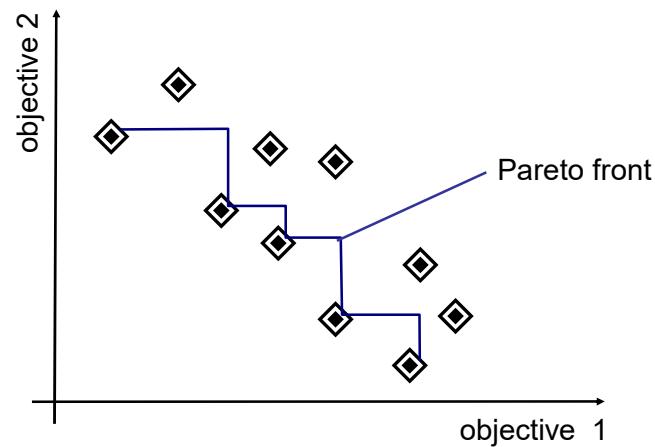


- Given: two decision vectors x_1 and x_2
 - $x_1 \gg x_2$ (strongly dominates) if $\forall i: f_i(x_1) < f_i(x_2)$
 - $x_1 \succ x_2$ (dominates) if $\forall i: f_i(x_1) \leq f_i(x_2) \wedge \exists j: f_j(x_1) < f_j(x_2)$
 - $x_1 \sim x_2$ (indifferent) if $\forall i: f_i(x_1) = f_i(x_2)$
 - $x_1 \parallel x_2$ (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 $x \in X$ is said to be *Pareto-optimal* if $\nexists y \in X: y \succ x$



Source: C. Haubelt, J. Teich

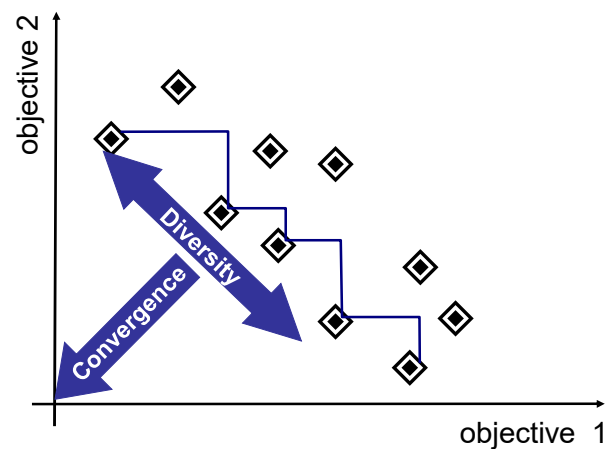
ECE382M.20: SoC Design, Lecture 5

© 2026 A. Gerstlauer

33

Optimization Goals

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



Source: C. Haubelt, J. Teich

ECE382M.20: SoC Design, Lecture 5

© 2026 A. Gerstlauer

34

Evolutionary Algorithms (EAs)

- **Multi-objective evolutionary algorithms (MOEAs)**
 - Capable to explore the search space very fast, i.e., they can find some good solutions after a few iterations (generations)
 - Explore high dimensional search spaces
 - Can solve variety of problems (discrete, continuous, ...)
 - Work on a population of individuals in parallel
 - Black box optimization (generic evaluation model)
- **Fitness evaluation**
 - Simulation, analysis or hybrid
 - Tradeoff between accuracy and speed
 - Hierarchical optimization
 - Combination with second-level optimization

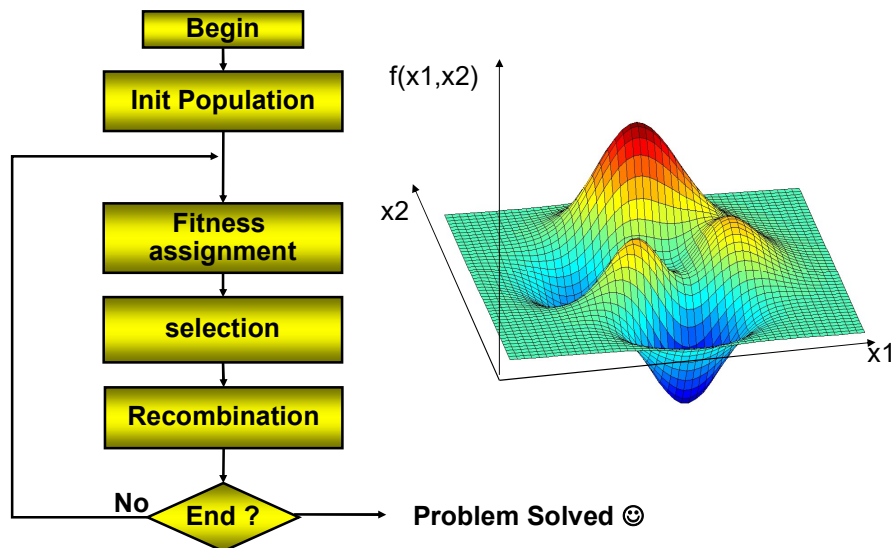
Source: C. Haubelt, J. Teich

ECE382M.20: SoC Design, Lecture 5

© 2026 A. Gerstlauer

35

Multi-Objective Evolutionary Algorithm



Source: C. Haubelt, J. Teich

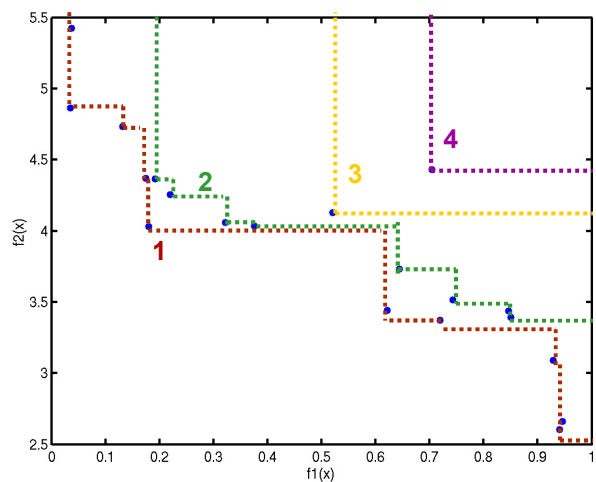
ECE382M.20: SoC Design, Lecture 5

© 2026 A. Gerstlauer

36

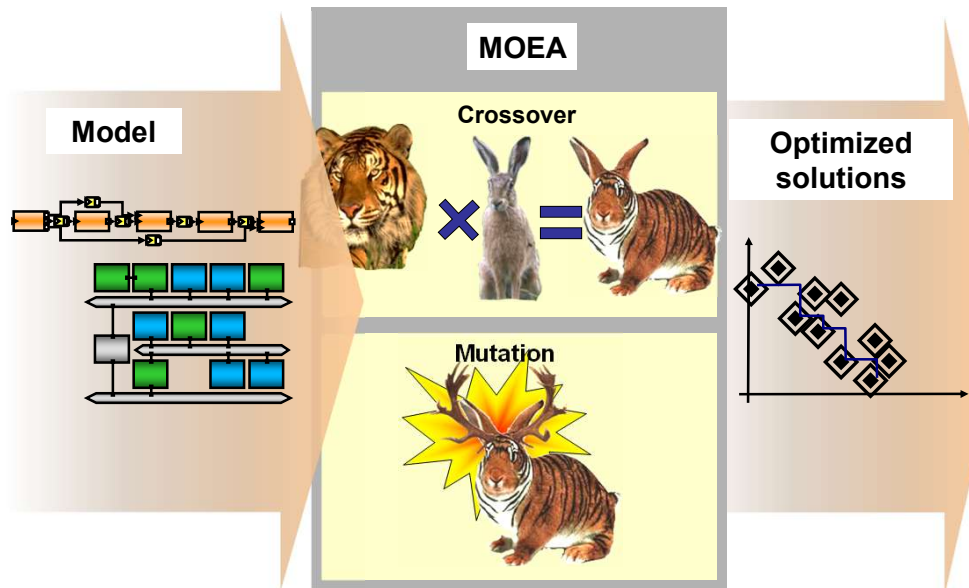
Fitness Selection

- Pareto ranking



Source: C. Haubelt, J. Teich

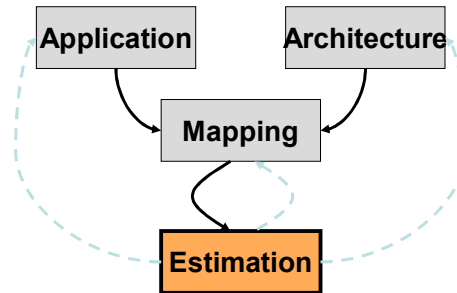
Recombination



Source: C. Haubelt, J. Teich

MPSoC Synthesis Summary

- **Design space exploration (DSE)**
 - Parallel application models & multi-processor architectures
 - Multi-objective, Pareto optimality
 - Traditional HW/SW co-design approaches not sufficient
- **Iterative exploration**
 - Determine mapping
 - Partitioning & scheduling
 - Evaluate solutions
 - Virtual platforms
- **DSE approaches**
 - Evolutionary algorithms
 - Reinforcement learning



➤ ECE382N.23: Embedded System Design & Modeling

Source: Lothar Thiele, ETH Zürich