1. Introduction HW/SW Codesign

Is about: specification & modelling of mixed HW/SW-Solutions at high abstraction levels, Optimized partitioning, scheduling & estimation with holistic HW/SW-component consideration to improve design quality (cost reduction, time-to market) and optimized performance (low latency, high system throughput)

Motivation: increasing complexity & function diversity/performance, lower cost & shorter development cycles

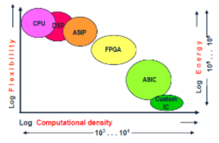
Embedded System: application specific processing system embedded in bigger technical context, consists of cooperating optimized HW/SW components

Requirements for HW/SW Systems:
- **RAS** (Reliability, Availability, Serviceability): when R(t)=exp(-λt)

$$R(t) = MTTF(system) = \sum MTTF(subsystems) = \sum \frac{1}{failiure_{rate}}$$
$$\mathrm{A(t) = MTTF/(MTTF + MTTR)}$$
$$S(t) = MTTR\ (Mean\ time\ to\ repair)$$

- **Efficiency:** Cost, energy, execution time, area
- **Real-time capability:** system reacts to external stimuli from environment in defined time; *Hard real-time condition:* Non-compliance may lead to system failure
- **Flexibility** (freely programmable CPU resources) Risk minimization, Time-to-market, Post-shipment upgrades

Computational density: Compute operations per area and time CD=ops/Lmin²; Computing Power CP=CD*N with N area in squares Lmin²



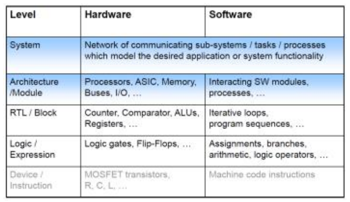
Functional diversity: number of operations which can be changed instantaneously of compute entity

Moore's Law: doubling of chip capacity every 2-3 years, how to deal with design gap?

Design Productivity Improvements by raised levels of abstraction: Polygons mask layout ➔Transistor circuity➔Logic gates (standard cells)➔RTL (Register Transfer Block, ALUs, Registers…) design➔HW-description languages and behavioural synthesis

Platform based SOC Design: Conquer design complexity by reuse maximization: Shorter development cycles & higher chances for (first time) fault-free Design. Standard on-Chip busses/interfaces, CPU's, SW-development environments

Abstraction Levels:



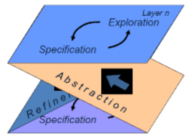
Level	Hardware	Software
System	Network of communicating sub-systems / tasks / processes which model the desired application or system functionality	
Architecture /Module	Processors, ASIC, Memory, Buses, I/O, …	Interacting SW modules, processes, …
RTL / Block	Counter, Comparator, ALUs, Registers, …	Iterative loops, program sequences, …
Logic / Expression	Logic gates, Flip-Flops, …	Assignments, branches, arithmetic, logic operators, …
Device / Instruction	MOSFET transistors, R, C, L, …	Machine code instructions

2. Design Methodology

System design: process to implement a desired function with a given set of physical components;

Appropriate design process: Improves quality of the product, Reduces cost and development time (time-to-market)

Design Flow: has proven practical value, identifies design faults during early phases of design (at high abstraction level), Avoid time consuming and costly iterations across multiple abstraction levels; Top-Down-Design



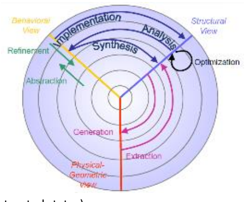
- **Specification:** Description of system behavior with formal, executable models
- **Exploration**: Comparison of alternative realizations with respect to cost, performance, robustness, …
- **Refinement**: Synthesis of a structural system representation out of the functional specification

- **Design space exploration:** roots on efficient estimation and simulation techniques which allow design characteristic evaluation prior to costly realization / implementation

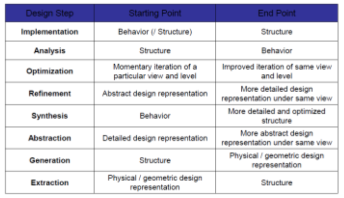
Design at High Layers of Abstraction: Higher efficiency in design representation (few lines of HDL code represent multiple 1000 logic gates) and Oversees a much bigger implementation space (Avoids local optima)

Design Verification by Simulation: Simulation can't achieve exhaustive coverage of input combinations: 32-bit ALU has 2^{32} x 2^{32} = 2^{64} input combinations, but is meaningful to reasonable subset of input combinations; Typical input patterns obtain confidence in design but cannot prove correctness nor completeness

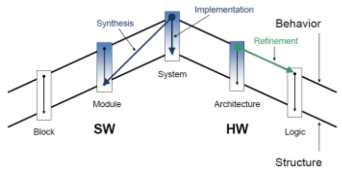
Simulation Acceleration: -Divide and Conquer: Parallel simulation of system blocks, - Mixed-Level Simulation: Simulate components at different level of detail, Reduction of simulated real-time: 1s real time can be eternity (seed config with saturated states);



Design Views:



Design Step	Starting Point	End Point
Implementation	Behavior (/ Structure)	Structure
Analysis	Structure	Behavior
Optimization	Momentary iteration of a particular view and level	Improved iteration of same view and level
Refinement	Abstract design representation	More detailed design representation under same view
Synthesis	Behavior	More detailed and optimized structure
Abstraction	Detailed design representation	More abstract design representation under same view
Generation	Structure	Physical / geometric design representation
Extraction	Physical / geometric design representation	Structure



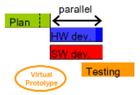
3. Specification & Modeling

Specification: defines supported functionality of system -> model is useful

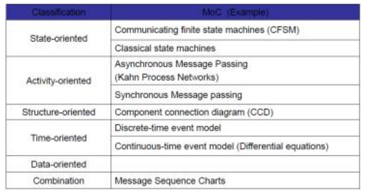
Models: describe how a system functions; Characteristics: Formal (Complete/partial) description of a system, without unnecessary detail (abstraction), Understandable and simple to modify

Architectures: describe how the system is implemented

Virtual Prototypes: allow for the HW and SW components of a system to be developed in parallel (instead of sequential) by an ISA compatible HW-model



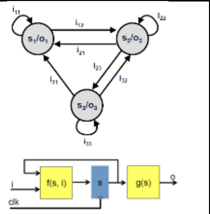
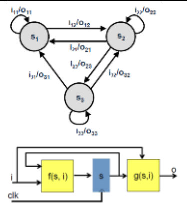
Model Classification:



Classification	MoC (Example)
State-oriented	Communicating finite state machines (CFSM)
	Classical state machines
Activity-oriented	Asynchronous Message Passing (Kahn Process Networks)
	Synchronous Message passing
Structure-oriented	Component connection diagram (CCD)
Time-oriented	Discrete-time event model
	Continuous-time event model (Differential equations)
Data-oriented	
Combination	Message Sequence Charts

Graph Models: -

State oriented: states (vertices) connected by state transitions (edges), triggered by external events; best suited for describing control units (real-time controllers, timing-latency important)

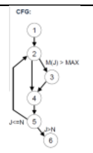
Moore State Machine:	Mealy State Machine:
	
+: No combinatorial path (limits logic depth),useideticaldesignstyle	**+:** Fewer states, clear layout; Most general FSM
$\mathrm{T_{clk} > \Sigma T_{logic} + T_{setup} + T_{pd}}$	**-:** Long combinatorial paths when multiple FSMs are concatenated; output depend on current state and input Avoid whenever possible!
$\mathbf{t_{setup} + t_{pd} + N * t_{gate} < 1/f}$	
-: Large number of states	

Control Flow Graph (CFG)

a directed, possibly cyclic graph; Vertices represent code without jumps; Edges represent jumps in the control flow

Transitions in a CFG are triggered solely by the completion of the preceding block

Only a single branch is taken to transition from one block to the next (unique!)



- **Activity oriented:** describe a system as a set of actions which resolve dependencies. best suited for transformational systems (digital signal processing; data passed through a transfer function at a fixed rate.)

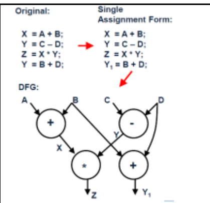
Data Flow Graph (DFG)

describe the data dependencies between a number of operations

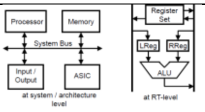
a directed, acyclic graph; Vertices =operations; Edges = data flow; , multiple-edges being traversed possible (unique)

DFG's calculations are triggered by availability of data

cannot portray branches in code, but can depict parallelization



- **Structure Oriented Model:** describe a system as a set of physical components and their interconnects; used to depict the physical configuration of a system.



- **Data Oriented Model:** describe a system as a hierarchy of data structures, best suited for describing systems in which the structural representation of data is more important than the system's functionality (e.g. databases)

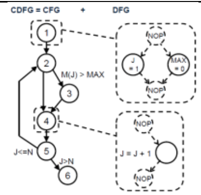
- **Combined Models:** merges benefits of simpler models, allows complete description of a complex system. best for systems that span a large design domain, e.g. real-time systems or ASICs.

Control Data Flow Graph:

Simultaneous description of the control-structure (e.g. branches) and data dependencies

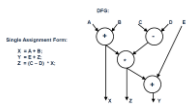
CFG: State machine representing the sequential control flow; The operations contained within a block (vertex) are expanded in form of a DFG

DFG: NOP operations provide a uniform entry and exit point for each block



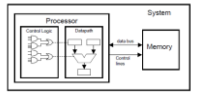
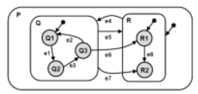
Model Characteristics:

Concurrency: often simpler to split system into concurrent sub-systems: e.g. 2 FSMs with 1 state is simpler than 1 FSM with 2 states.

Data oriented concurrency	Control oriented concurrency
No specifc order, single assignment rule: every variable appaers only once on the left hand 	Explicit control instructions (fork-join concurrent behaviour) determine order of operations

State Transitions: transitions depend on conditions/states; system with N-states can have up to N² transitions => control centric behavior

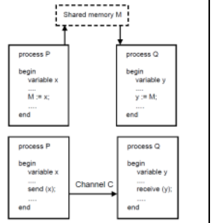
Hierarchy: real systems are too complex to be viewed in entirety ➔ hierarchy splits system into smaller subsystems so developers can focus on their sub-system (allows reuse, not in depth understanding needed)

Structural hierarchy	Behavioural/functional hierarchy
Every component is made up of a sub-structure to lower level of abstraction 	Divides functions into sequential or concurrent sub-functions 

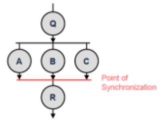
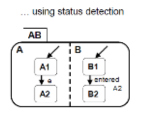
Program Structures/Constructs: many functions can be described best by sequential algorithms including branches, iterations, subroutines..

Completion/Abschluss: process ability to indicate it has stopped: All calculations are made or all variables got assigned their new value

Communication: Connect HW/SW subsystems

Shared-Memory	Message-passing
Sending process writes global variable into shared resource; all receiving processes can now read var; sync must be done separate 	**1.** Data between processes is exchanged through communication channels (uni-bidirectional, point-to-point, shared bus) **2.** channel can be blocking on non-blocking transfer -blocking-trans: sending process waits until receiving process hast accepted data -non-blocking-trans: sending process writes data in queue and continues processing. Receiver can read it at its leisure => standard today, additional memory for queue needed.

Synchronization: concurrent processes are never fully independent from each other. Sync to exchange data; Connect HW/SW subsystems

Control oriented sync	Data oriented sync
Control structure of functions determine sync 	Sync by useing inter-process communication (shared memory, message passing) 

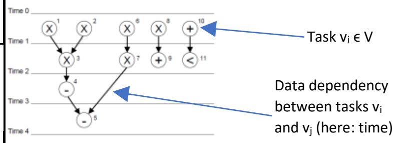
Exeption Handling: Events like a reset or interrupt can abrupt terminate a process. If such event/exception occurs control is passed to a pre-defined exception handling routine.

Non-Determinism: allows specification of multiple options due to unclear best suiting operations for app. ➔ put off final decision for later in design process

4. System Synthesis & HW/SW Partitioning

Design synthesis: *Allocation:* Selection and provisioning of processing resources; *Mapping:* Assignment of functions to resources; *Scheduling:* Determination of execution sequences and start times for tasks/processes under consideration of data dependencies in the task graph

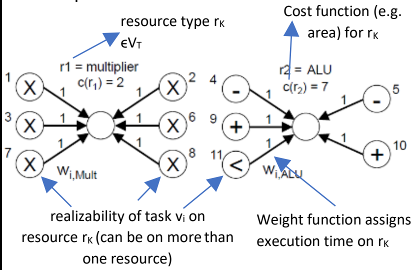
Task Graph: (DFG) Vertex = tasks/processes; edges= data dependencies



Schedule: assigns each task v_i a start time $t_j=\tau(v_i)$, so that

$$\tau(vj)_{start} \geq \tau(vi)_{start} + di_{texe}$$
$$\text{Latency: } \mathrm{L}\ \max\{\tau(vi) + di\} - \min\{\tau(vi)\}$$

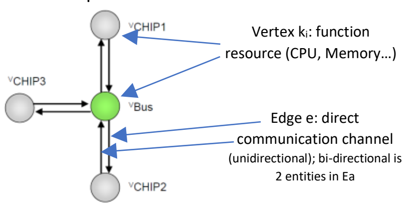
Resource Graph:


resource type r_K ϵV_T
Cost function (e.g. area) for r_K
r1 = multiplier
c(r1) = 2
r2 = ALU
c(r2) = 7
W i,Mult
W i,ALU
realizability of task vᵢ on resource r_K (can be on more than one resource)
Weight function assigns execution time on r_K

Allocation: function $\alpha(r_k)$ assigns each resource a number of available resource instances

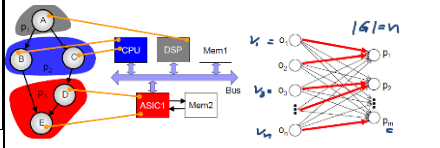
Mapping: $r_k = \beta(v_i)$ indicates a resource type, $\gamma(v_i)$ indicates the instance of the resource type r_k , which executes the task v_i

Architecture Graph:

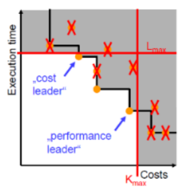

vCHIP1
vCHIP3
vBus
vCHIP2
Vertex kᵢ: function resource (CPU, Memory…)
Edge e: direct communication channel (unidirectional); bi-directional is 2 entities in Ea

Partition: assigns each vertex v_i of task graph to exactly one vertex q_i of architecture graph; Objective is to identify partition with the lowest cost for a given target function.

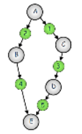
Target function: $F(P) = \mathrm{k_1 * area(P) + k_2 * latncy(P) + k_3 * power = min.}$



Pareto-Analysis and Design Space Reduction: Every combination of architecture/mapping corresponds to a design point in the multi dimensional space of possible target functions; Elimination of suboptimal design points via Pareto-Analysis (design point that cannot be improved in any target function without being deteriorated in at least one other target function



Communications Vertices: Assignment of costs c(rk) and estimated communication latencies between tasks



Classification of partitioning methods: Constructive vs. transformational/iterative

Classification of partitioning algorithms: structural vs. functional

Criteria for partitioning: Abstraction level, Task granularity, Metrics and Estimation, Target function

Target/Cost-functions:

$$cost\_f(P) = \mathrm{k_1 * area(P) + k_2 * latncy(P) + k_3 * power}$$
$$cost\_f(P) = \mathrm{k_1 * h(area, \overline{area}) + k_2 * h(ltncy, \overline{ltncy}) + k_3 * h(pwr, \overline{pwr})}$$

h(): Zero cost function indicates how close metric is to target value (0 if x<<u>x</u>)

Closeness-functions: Measure indicating a force to group two objects during partitioning process; increased by number of connections/data/memory rages…

$$Closenes(pi,pj) = \frac{k1 * inputs_{i,j} + wires_{i,j}}{MaxConn(P)^{k2}} * \frac{sizemax}{Min(size_i, siz_j)^{k3}}$$

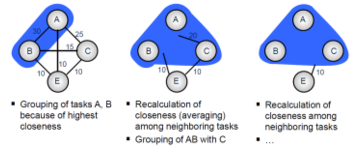
First term prefers objects with common data; Second term fosters largest possible groups while avoiding that all objects

Partitioning Methods: Complexity of partitioning problem O (m^n) with m: architecture components and n: task objects (e.g. n = 20, m = 4 = 1012 possible partitions) ➔ Cannot be dealt with „exhaustive search" ➔ use heuristic methods instead of exact ones (like ILP – integer linear programming);

Constructive algorithms: Sequential adding of objects to existing groups based on closeness functions; Usually serve as start partitions for later usage of iterative methods; Difficult to identify or define a meaningful closeness function

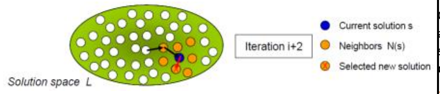
- Random grouping: tasks are randomly mapped to resources in sequential fashion; complexity O(n)

- Hierarchical Clustering: (Functional) object / task is assigned to a group; Subsequent recalculation of closeness functions; Iteration of above steps till termination condition is fulfilled; Termination criteria: Number of remaining clusters/groups or getting below a certain closeness boundary (e.g. ≥ 15); Characteristics O(n^2); applicable to sets with large number of objects; cannot overcome local minima - Multistage Clustering: Alternative method with different closeness functions per partitioning iteration

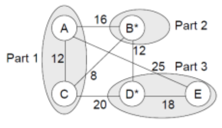

A
B
C
E
30
25
15
10
20
Grouping of tasks A, B because of highest closeness
Recalculation of closeness (averaging) among neighboring tasks
Grouping of AB with C
Recalculation of closeness among neighboring tasks
…

Transformational algorithms/Iterative methods: (Iteratively) modifys already existing partitions with the expectation to find an even better solution; Typically uses target functions as optimization objective; Computation complexity of iterative methods grows linearly with number of partitioning alternatives investigated

- Local Search: Start at: Initial solution; Iteration: Selection of solution(s) in neighbourhood of current solution due to cost function ➔ Acceptance of best neighbour as new solution for next iteration; can escape local minima



- Group Migration – Min Cut: Move objects to different groups and determine the resulting deltas in target function; Object with biggest reduction / smallest increase (prevents local minimums) in target function is moved to new group (calc internal & external costs!); Every object can be moved only once (prevents loops); When all objects have been moved, select partition with best target fcn



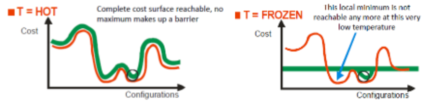
	Min-Cut		
Iter	Part 1	Part 2	Part 3
i	A, C	B	D, E
i+1	C	B	A, D, E
i+2	-	B	A,C,D,E

i+1: delta cost = -13 for A to Part3
i+2: delta cost = -32 for C to Part 3

- Ratio Cut method: Prevent clustering of all objects into a single group by:

$$\text{Ratio} = \frac{\text{cut}(P)}{\text{size}(p1) \times \text{size}(p2)}$$

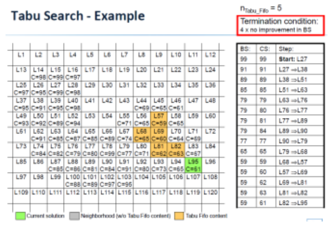
- Simulated Annealing: Simulated degradation of temperature T such that a thermal equilibrium is attained for each T; Also worse solutions out of neighbourhood may be taken, i.e. deteriorations are accepted if exp(-delta_f/T) > config(x); As temperature is reduced stepwise the exponent e approaches to infinity ➔ probability to accept degradings is getting smaller with lower temp; SA is an exact (optimal) method when temperature degradation happens arbitrarily slowly; O(e^x-x^n);



- Greedy Partitioning: Starting from a pure SW partition objects are moved into HW partition until performance requirements are met P_{init} = { p_{sw}, p_{hw} } = { O, Ø }, minimize HW portion for reasons: area, development effort

- Gupta Partitioning: Starting from a pure HW partition, objects are moved to SW partition as long as performance requirements are still met and target function is improved P_{init} = { p_{sw}, p_{hw} } = { O, Ø }, minimize SW portion while considering performance condition and target function optimization

- Tabu Search: Heuristic search method; fast and nearly optimal solving of optimization problems; Starting: initial solution; Iteration: picks the best neighbour or the one with least degradation of result; Loops are prevented by considering only solutions which haven't been considered before (storing of last n solutions in TabuFifo); Escapes from local minima; Accepting a new solution implies; removal of oldest solution from TabuFifo (if TabuFifo is full); Length of TabuFifo influences effectiveness of method TabuFifo too small: Loops may occur; too large: Possibly no new neighbours are found which weren't considered yet


Tabu Search - Example

n_{Tabu_Fifo} = 5

Termination condition: 4 x no improvement in BS

L1	L2	L3	L4	L5	L6	L7	L8	L9	L10	L11	L12
L13	L14 C=98	L15 C=99	L16 C=97	L17	L18	L19	L20	L21	L22	L23	L24
L25 C=97	L26 C=95	L27 C=99	L28 C=98	L29	L30	L31	L32	L33	L34	L35	L36
L37 C=95	L38 C=91	L39 C=95	L40 C=98	L41	L42	L43	L44 C=69	L45 C=65	L46 C=61	L47	L48
L49 C=93	L50 C=92	L51 C=89	L52 C=94	L53	L54	L55 C=71	L56 C=65	L57 C=59	L58 C=65	L59	L60
L61	L62 C=91	L63 C=85	L64 C=87	L65 C=85	L66 C=89	L67 C=74	L68 C=65	L69 C=60	L70 C=64	L71 C=69	L72
L73	L74 C=84	L75 C=82	L76 C=79	L77 C=80	L78 C=99	L79 C=77	L80 C=71	L81 C=62	L82 C=63	L83 C=67	L84
L85	L86	L87 C=85	L88 C=86	L89 C=81	L90 C=84	L91 C=91	L92 C=80	L93 C=73	L94 C=65	L95 C=61	L96
L97	L98	L99	L100 C=88	L101 C=89	L102 C=97	L103 C=96	L104	L105	L106	L107	L108
L109	L110	L111	L112	L113	L114	L115	L116	L117	L118	L119	L120

BS:	CS:	Step:
99	99	Start: L27
91	91	L27 → L38
89	89	L38 → L51
85	85	L51 → L63
79	79	L63 → L76
79	80	L76 → L77
79	81	L77 → L89
79	84	L89 → L90
77	77	L90 → L79
65	65	L79 → L68
59	59	L68 → L57
59	60	L57 → L69
59	62	L69 → L81
59	63	L81 → L82
59	61	L82 → L95

🟩 Current solution ▨ Neighborhood (w/o Tabu Fifo content) 🟨 Tabu Fifo content

5. Scheduling

Aim: Determines the execution sequence and start times of tasks between different and onto the same resource under consideration of data dependencies in the task graph

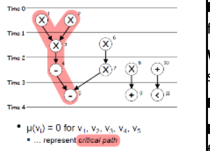
Classification:

Preemptive scheduling: Possibility to interrupt execution of a task during run time (to benefit other task) and resume execution on same/different resource; Only meaningful when processing time considerably larger than dispatch/switch latency

Static scheduling: Determines the execution sequence and start times of tasks at design or compilation time, Requires well-defined environment, mostly in data flow problems, + lower scheduling complexity at run time

Scheduling without resource constraints: (Theoretically) relevant to determine the lower bound for (processing) latency

- As Soon As Possible (ASAP): Every task is executed as early as possible; Characteristics: Local, constructive algorithm; Typically results in suboptimal solutions; O(x^n); **no constraints**



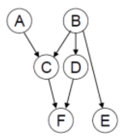
- As Late As Possible (ALAP): define a latency limit L^L; mobility μ of task gives start time window: $\mu(v_i) = \tau(v_i)^L - \tau(v_i)^S$; if $\mu(v_i)$=0, the vi is part of critical path

Timing Constraints: Absolute: Deadlines: Latest possible start and termination times of tasks; Release time: Earliest possible start time of tasks; Relative: time relationships between tasks (intersected min/max nr of time steps between)

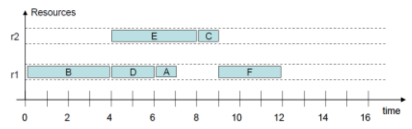
Scheduling with resource constraints: Considers availability of limited resources during scheduling; Optimization problems: Determine minimum latency under a given allocation α; Minimize cost (area) for given latency bound LL; Scheduling with constraints are NP-hard; Heuristic methods required

- ASAP/ALAP with Conditional Task Shift: Starting point is an ASAP-/ALAP schedule; Check if schedule obeys resource constraint: e.g. α(mult) = 2; α(ALU) = 2; In case of resource constraint violation, tasks with positive mobility are shifted to later (ASAP)/ earlier (ALAP) time slot

- List Scheduling: Enhancement of ASAP considering global criteria (Nr. of succeeding vertices, Weight of the path (longest path), Mobility of vertices) to determine execution sequence of tasks; In each step select vertices with maximum priority to start. (but check dependency's in task graph first!)

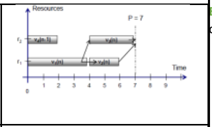
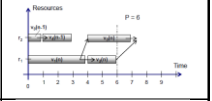
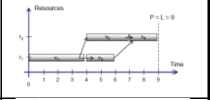
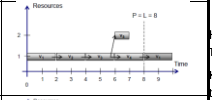
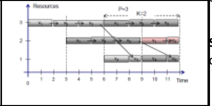


Task	d	mapping	priority
A	1	r1	4
B	4	r1	8
C	1	r2	6
D	2	r1	6
E	4	r2	3
F	3	r1	3



Periodic scheduling: Scheduling of iterative tasks with execution interval (period) P for planning loops and Pipelining (Concurrent scheduling of sub-tasks from different iterations); τ(vi, n) = τ(vi) + n P; n: iteration index

Concurrent Scheduling of iterations: simultaneous processing of tasks belonging to different iterations ➔ otherwise sequential

Not-overlapping Schedules: Tasks scheduled in the base interval [0,…,P] do not expand over the boundaries t = 0 und t = P. Relevant for architectures with synchronization points at interval boundaries. (here also concurrent)	
Overlapping Schedules: Tasks may expand beyond interval boundaries, however, repeat with period P. (here also concurrent)	
Sequential Scheduling of iterations: All tasks belonging to iteration n have to be completely finished before tasks of the subsequent iteration may be started.	
Fully-static Scheduling: All iterations of a task are bound to the same resource (instance).	
Cyclo-static with periodicity K: K subsequent iterations of a task may be bound to different resources. The resource of the iteration (K + n) has to be the same as the resource of the iteration n.	

Dynamic scheduling: Determines the execution sequence and start times of tasks during run time, mostly applied to control flow problems; information that is known at runtime only can be taken into account

Dispatch latency L_D: max time between stop of v_i and start of v_j on same resource

Resource load U: Given: G(V, E) with a single resource type of allocation 1 and a schedule of latency L: $U = \frac{\sum di}{L} * 100\%$

Processing time t_{ex}: $t_{ex}(v_i) = \tau_e(v_i) - \tau_b(v_i)$ with $t_b(v_i)$: v_i uses resource for the first time; $\tau e(v_i)$: v_i is completely processed (finishing time)

Wait time t_W: $t_W(v_i) = \tau_e(v_i) – t_r(v_i) – d_i$ with t_r: earliest possible start time (release time)

Flow time t_F: $t_F(v_i) = \tau_e(v_i) – t_r(v_i)$

Lateness t_L: $t_L(v_i) = \tau_e(v_i) – t_d(v_i)$ with td: deadline (latest possible finishing time)

Tardiness t_T: $t_T(v_i) = \max\{\tau_e(v_i) – t_d(v_i), 0\}$

Optimization Criteria: Multi-user systems: Minimization of the mean wait time $W = \frac{1}{V} * \sum t_W(vi)$ and flow time: $F = \frac{1}{V} * \sum t_F(vi)$; Minimization of the max response time (time between process start and output of first valid results); Real-time systems: In addition to mean wait times and flow times, misses of deadlines are of special interest: max Lateness = max(t(vi)); Number of tasks that miss their Deadline sum(u(vi))

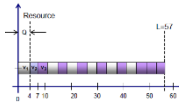
Strategies:

First come first served (FCFS): simple to realize (like FIFO); Suited for uniform tasks: similar processing times, identical priorities, no real-time requirements but fluctuation of tWait:

Shortest job first (SJF): Minimization of mean wait time or flow time, requires sorting of tasks, not preemptive

Shortest remaining time next (SRTN): Pre-emptive version of SJF; dynamic priority assignment; At any time t the task with the min remaining processing time is selected from all schedulable tasks; in real time systems only estimation

Round robin (RR): Circular queue with fix time interval Q, after which the context is switched at the latest; Tasks are processed in turn; Advantage: avoids „starvation" of tasks; Drawback: Frequently long wait times



6. Design Estimation Techniques

Design parameter estimation allows to bound relevant system aspects prior to system implementation to support design decisions and system optimization.

Estimation Metrics:

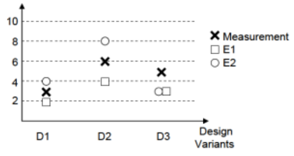
Quality and Costs: HW (test, manufacture), SW (memory, development), Performance (throughput, clock cycles), Communitcation (transfer rate), Power, Time (Design, Time-to-market)

Estimation accuracy: $Acc = |E(D) - M(D)|$; E(D) is the estimated and M(D) the measured values for a design D. Relative error: $RE = \frac{|E(D)-M(D)|}{M(D)}$

Estimation fidelity: Fidelity F is defined as percentile of correctly predicted comparisons between multiple implementations:

$$F = \frac{2}{n(n-1)} \sum_{i=1}^{n} \sum_{j=i+1}^{n} \mu_{ij} 100\,\% \qquad \mu_{ij} = \begin{cases} 1 & E(D_i) > E(D_j) \wedge M(D_i) > M(D_j), \vee \\ & E(D_i) < E(D_j) \wedge M(D_i) < M(D_j), \vee \\ & E(D_i) = E(D_j) \wedge M(D_i) = M(D_j) \\ 0 & \text{else} \end{cases}$$

Accuracy: E1: 1/3+2/6+2/5=32/30; E2: 1/3+2/6+2/5=32/30
Fidelity: E1: 1/3(1+1+1)=3/3; E2: 1/3(1+0+1)=2/3
1P each for correct Accuracy/Fidelity values
E1 is better than E2



HW-Cost Metrics: Manufacturing (area, SoC – CPU, Mem), Module (Pin Count), Test (time on test device), Development (Team size, Complexity, lion share!)

HW-Performance Metrics: Compute performance, Communication band with, throughput, Processing Time/Latency, Clock Rate

$$Texe = Ninstr * T * CPI$$

SW-Cost Metrics: HW (components: CPU, RAM), Development (Teamsize, dominant!); Memory Demand (Program and Data Memory)

SW-Performance Metrics: MIPS (million instructions per second, equal to MFLOPS or MACS); Memory (different access latencies: Cache, SRAM, DRAM)

Communication Metrics: Max. Bit Rate (channel specific upper bound data transfer rate); Average Bit Rate; directly impacts processing performance

Other Metrics:

Power dissipation: insignificant for λ>0,1µm, in future dominant because of leakage currents

$$P = P_{stat} + P_{short} + P_{cap}; \qquad P_{cap} = \alpha f_{clk} C_{load} {V_{dd}}^2$$

Design-for-Test: BIST (build in self test); LSSD (Level sensitive scan design)

Development time: can be significantly reduced by usage of standard, programmable components

Time to market: The earlier a product is available on the market, the bigger are its business volume and profit margins; „6 months delay in product entry may result in 33% less profit over a period of 5 years"

Maximum Operator Latency: functional unit of type rk with latency delay(rk)

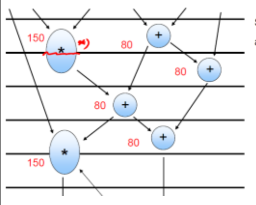
$$T = \max_k \left(delay(r_k)\right)$$

Slack: (Positive) slack denotes that fraction of the clock period (zeitdauer) which is not utilized (nicht ausgelastet) by a functional unit vk

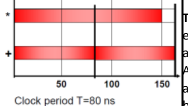
$$slack(T, r_k) = \left(\lceil delay(r_k)/T \rceil \right) \cdot T - delay(r_k);$$

occur(r_k): Number of operators of type r_k

$$avg\_slack(T) = \frac{\sum_{k=1}^{|E_R|} \left(occur(r_k) \times slack(T, r_k)\right)}{\sum_{k=1}^{|E_R|} occur(r_k)} \qquad utilization(T) = 1 - \frac{avg\_slack(T)}{T};$$



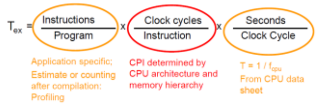
Slack(80, +) = 0; slack(160, *) = 10ns
avg_slack(80) = 20 ns / 6 = 3.3 ns



Clock period T=80 ns
Latency L=5

Pipelining: with P equal stages, put in registers to rise clock

Software Estimation by Generic Model: Advantage: One compiler sufficient for multiple CPUs – CPU technology data contain details such as CPI, register set, ISA, etc.; Easy retargeting to different CPU by means of new technology data set; No need that compiler exists already at design time of CPU; Disadvantage: Lower accuracy – Technology data is estimated


Tex =
Instructions / Program
x
Clock cycles / Instruction
x
Seconds / Clock Cycle
Application specific; Estimate or counting after compilation: Profiling
CPI determined by CPU architecture and memory hierarchy
T = 1 / fcpu From CPU data sheet

Instruction Count Estimation:

DFG

$$IC(P, r_k) = \sum_{v_i \in V} IC(v_i, r_k)$$

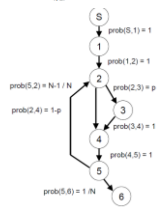
- Sum of instructions of all tasks running on a CPU of type r_k

- CFG
  - Consist of sequential tasks, conditional branches and loops
  - Estimation via execution frequency of specific tasks

$$IC(P, r_k) = \sum_{v_i \in V} IC(v_i, r_k) \times freq(v_i)$$

- freq(S) = 1
- freq(1) = 1 x freq(S)
- freq(2) = 1 x freq(1) + (N-1) x freq(5) / N
- freq(3) = p x freq(2)
- freq(4) = 1 x freq(3) + (1-p) x freq(2)
- freq(5) = 1 x freq(4)
- freq(6) = 1 x freq(5) / N

- freq(1) = 1
- freq(2) = N
- freq(3) = p x N
- freq(4) = N
- freq(5) = N
- freq(6) = 1



Memory Space: Program memory (Aggregate instructions of all tasks times operand size of the respective CPU)

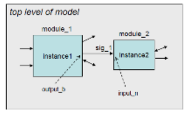
$$IC(P, r_k) \times instr\quad size(r_k)$$

7. VHDL/SystemC Praktikum

SystemC based on C++ with Extensions: SystemC class library to implement (Concurrency, Communication, Time management) and Simulation kernel

Structure of a SystemC Module: *Header File* (module.h): contains: Module declaration (Ports, Sockets; Member variables); Signals; Sub-modules *Implementation File* (module.cpp): Implementation/definition: Member functions; Processes; (Constructor)

Connecting Modules with Signals:



Member Functions and Processes: can read / write signals, ports, member variables; call interface functions of sockets via: signal_or_port.read();

Processes:

- Enable modelling concurrency (Communicate via signals or events, Processes cannot be called directly by other processes / member functions ➔ triggered by sensitivity (event, signal));

- Are special member functions (No return values and no parameters)

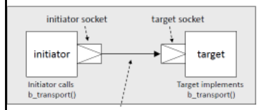
- Have to be registered with the simulation kernel in the module constructor

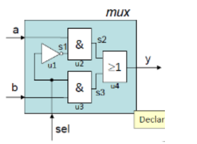
- 2 types of processes: SC_METHOD (On activation, process is infinitely fast executed from beginning to end) and SC_THREAD (On activation, commands are executed infinitely fast until next wait statement, on next activation until subsequent wait)

- Evaluate-update scheme to simulate concurrency: Phase1 (Process Execution, PE): all processes with change on a sensitive signal / event are executed (sequence undefined); Phase2 (Signal Assignment, SA): assignment of modified signals; Repeat PE and SA phases until system is stable, then increase simulation time; The sequence PE/SA is called "delta cycle" (no simulation time is consumed)

Transaction: call of a function of an interface

Transaction Level Modeling (TLM): Targets: Reduce modelling effort; Allow for easier model adaptability; Increase simulation speed; Enable efficient architecture exploration; Use models: SW development on virtual prototype, Architecture exploration, HW verification, Modeling styles: loosely timed, approximately timed, Support a range of different abstractions, Interaction via blocking or non-blocking transactions, Different number of transaction phases, Definition of a generic payload (extensible), Important standard for IP Intellectual Property) exchange, Mainly targeted at memory mapped bus


initiator socket
target socket
initiator
target
Initiator calls b_transport()
Target implements b_transport()



VHDL:

Entity: defines a „Black Box" with information: Model Name and Ports (inputs / outputs); No information concerning function and its implementation -> Architecture; more than one architecture design per Entity possible but an architecture belongs to exactly one Entity (allocate by Configuration)

Functionality Approaches: Modelling behaviour (through processes and concurrent signal assignments); Modelling the structure (through instantiation of given components and their interconnection)

Inherent parallelism of HW: All statements in the statements section of the architecture are CONCURRENT ! The sequence of the statements is irrelevant! (Inherent parallelism of HW)

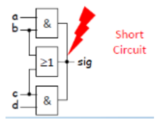
Processes: Complex functionalities cannot be modeled using only concurrent signal assignments -> Process: Interface between concurrent and sequential modeling; Acts like one concurrent statement, however, process statements are executed sequentially (if-else structure)

Concurrency – Delta Cycles: Evaluate-Update scheme: 1. PA (Process Activation phase): all activated processes are executed, sequence undefined; 2. SA (Signal Assignment phase): signals get newly assigned values; If SA activates further processes, repetition of PA-SA Sequence until stable state is reached

Sequential Statements: If-Else, Case-When

Modeling Synchronous Circuits: Assigned signals will become registers in HW; Apply event-Attribute only to clock!

Typical Modeling Errors: More than one assignments to the same signal in concurrent signal assignment/process; Missing signal assignment when modeling combinatorial logic ➔ Undesired Latch for Signal s2



IF a='1' THEN; s1 <= b OR c; s2 <= b AND c; ELSE s1 <= b XOR c;

The executable model is machine readable, can be simulated, and is intended toadvance the understanding of the system's behavior. In addition, the model