

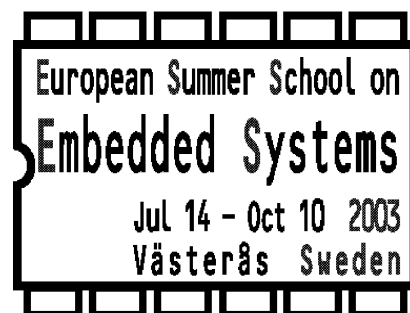
MÄLARDALENS HÖGSKOLA

ESSES 2003

European Summer School on Embedded Systems

Lecture Notes Part X

Embedded Systems: Introduction and Overview



Editors: Ylva Boivie, Hans Hansson, Jane Kim, Sang Lyul Min

Strängnäs, August 20-22, 2003

ISSN 1404-3041

ISRN MDH-MRTC-106/2003-1-SE

MRTC

MÄLARDALEN REAL-TIME
RESEARCH CENTRE

www.mrtc.mdh.se



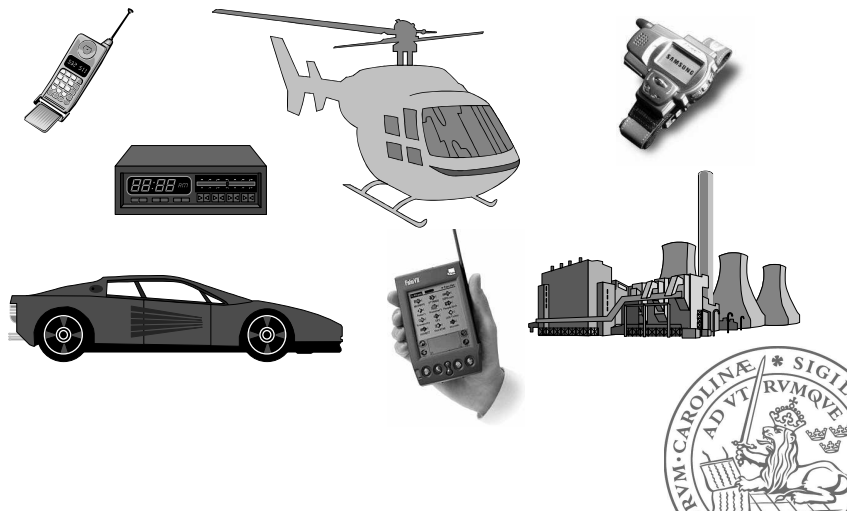
LUND INSTITUTE
OF TECHNOLOGY
Lund University

Design of Embedded Systems

Kris Kuchcinski
Dept. of Computer Science
Lund University
Sweden

<http://www.cs.lth/~kris>

Examples of Embedded Systems





LUND INSTITUTE
OF TECHNOLOGY
Lund University

Constraint Programming Approach

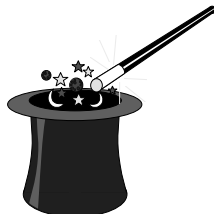
Kris Kuchcinski
Dept. of Computer Science
Lund University
Sweden

<http://www.cs.lth/~kris>

Quotations

"Constraint programming represents one of the closest approaches computer science has yet made to the Holy Grail of programming: the user states the problem, the computer solves it."

Eugene C. Freuder
CONSTRAINTS, April 1997



Introduction and Motivation

Synthesis of the following code
(inner loop of differential equation integrator)

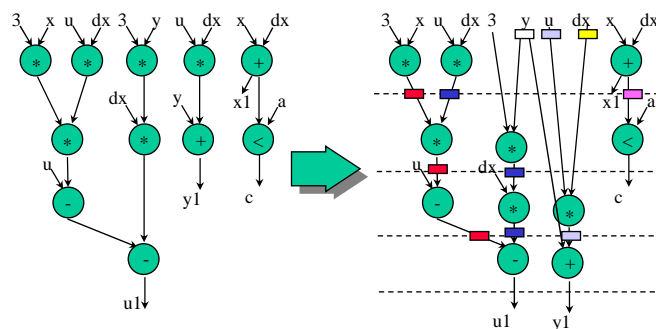
```

while c do
  begin
    x1 := x + dx;
    u1 := u - (3*x*u*dx);
    y1 := y + u*dx;
    c := x < a;
    x := x1; y := y1; u := u1;
  end;
  
```



3

Introduction and Motivation



data-flow graph

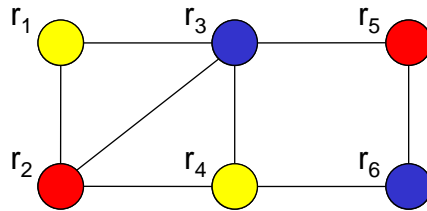
scheduled data-flow graph

register allocation



4

Register Allocation as Graph Coloring



Constraints:

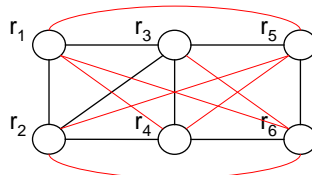
$[r_1, r_2, r_3, r_4, r_5, r_6] :: 0..2,$
 $r_1 \neq r_2, r_1 \neq r_3, r_2 \neq r_3,$
 $r_2 \neq r_4, r_3 \neq r_4, r_4 \neq r_6,$
 $r_5 \neq r_6.$



Register Allocation as Clique Finding

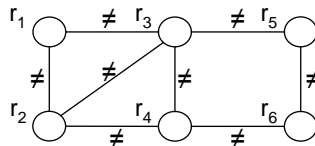
- for all r_i, r_j which are not connected by an edge:
 $r_i \neq 1 \vee r_j \neq 1$
- The maximal clique can be found by maximizing the following cost function:

$$\text{cost} = \sum_i r_i$$



Constraint Consistency

- All constraints are stored in the *constraint store*
- *Consistency* methods are applied to find inconsistent values and prune variables' domains
- Different types of consistency methods:
 - Node consistency
 - Arc consistency
 - Path consistency
 - ...



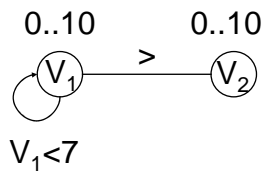
Consistency Properties

- Node consistency
 - A network is node consistent if in each node domain each value is consistent with unary constraint (e.g., $X > 7$)
- Arc consistency
 - A network is arc consistent if for each arc connecting variables V_i and V_j for each value in the domain of V_i there exist a value in the domain of V_j consistent with binary constraint (e.g., $X > Y$)

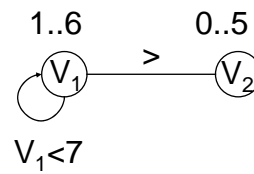


Node and Arc Consistency

- Example



Not node consistent
Not arc consistent

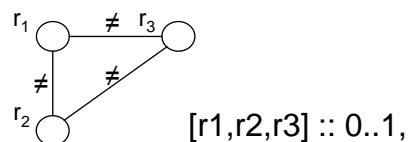


node consistent
arc consistent



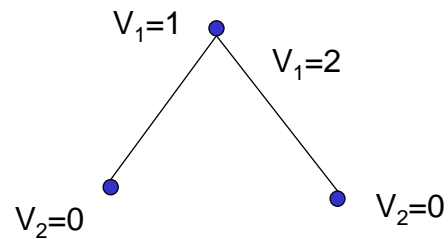
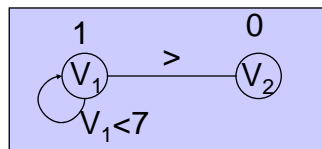
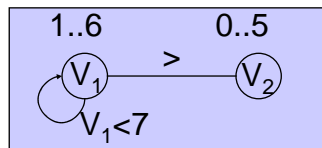
Need for search

- Node, arc and path consistency are in general not complete (complete for some problems with particular structures)
- Complete algorithm: N -consistency for N variable problems \rightarrow exponential complexity
- Example:



Search

- Solver is not complete and search for a solution is needed



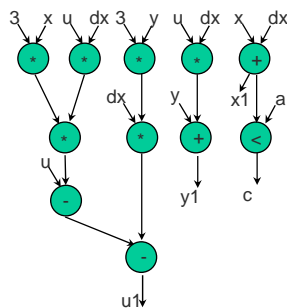
Constraint properties

- may specify partial information — need not uniquely specify the values of its variables,
- non-directional — typically one can infer a constraint on each present variable,
- declarative — specify relationship, not a procedure to enforce this relationship,
- additive — order of imposing constraints does not matter,
- rarely independent — typically they share variables



More realistic example Scheduling

Scheduling of the data-flow graph



Constraints:

- for all op_i and op_j such that op_i before op_j
 $T_i + D_i \leq T_j$
- for all op_i and op_j that can use the same resource
 $T_i + D_i \leq T_j \vee T_j + D_j \leq T_i \vee R_i \neq R_j$



Problems

- Constraint propagation for
 $T_i + D_i \leq T_j \vee T_j + D_j \leq T_i \vee R_i \neq R_j$ is weak
- Not all solvers support disjunctive constraints.
 - Other solution (reified constraints):

$$T_i + D_i \leq T_j \Leftrightarrow B1,$$

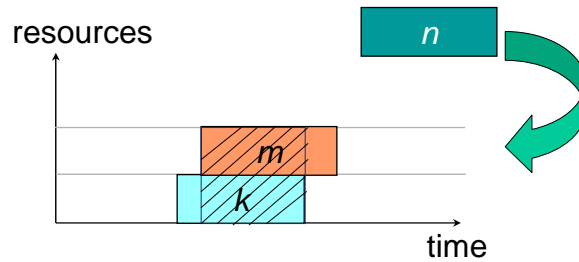
$$T_j + D_j \leq T_i \Leftrightarrow B2,$$

$$R_i \neq R_j \Leftrightarrow B3,$$

$$B1 + B2 + B3 \geq 1.$$



Propagation problems



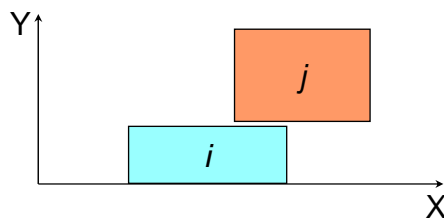
$$T_k + D_k \leq T_n \vee T_n + D_n \leq T_k \vee R_k \neq R_n$$

$$T_m + D_m \leq T_n \vee T_m + D_m \leq T_n \vee R_m \neq R_n$$



Global constraints

- Non-overlapping rectangles



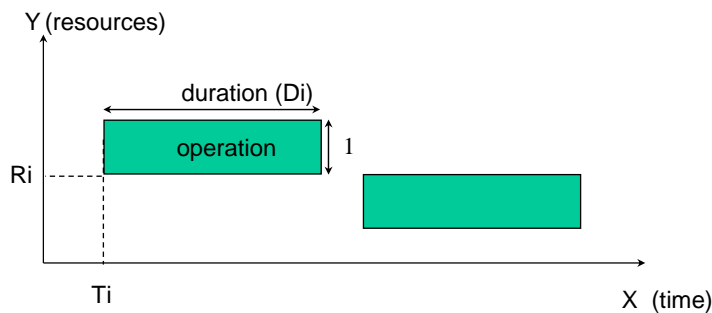
$$\text{diff2}([[X_i, Y_i, DX_i, DY_i], [X_j, Y_j, DX_j, DY_j]])$$

- All knowledge in one "place" – makes it possible to define good consistency methods (OR, mathematics, geometry, etc.)
- Specific algorithms for consistency – more efficient



Global Constraints – Scheduling

- diff2 constraint



diff2([[T1, R1, D1, 1], [T2, R2, D2, 1]], ...)



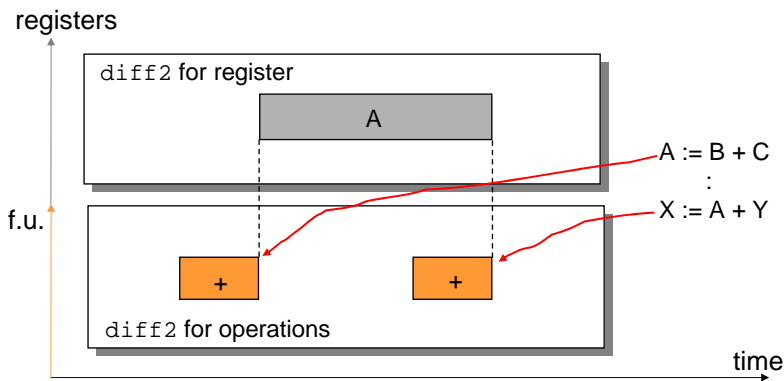
17

Scheduling Example Constraints

$T1 + 2 \leq T6$, $T2 + 2 \leq T6$,
 $T3 + 2 \leq T7$, $T4 + 2 \leq T8$,
 $T5 + 1 \leq T9$, $T6 + 2 \leq T10$,
 $T7 + 2 \leq T11$, $T10 + 1 \leq T11$,
 diff2([[T1,R1,2,1], [T2,R2,2,1], [T3,R3,2,1],
 [T4,R4,2,1], [T6,R6,2,1], [T7,R7,2,1],
 [T5,R5,1,1], [T8,R8,1,1], [T9,R9,1,1],
 [T10,R10,1,1], [T11,R11,1,1]]).



Registers



- can be done together with or after functional units allocation/binding and scheduling,



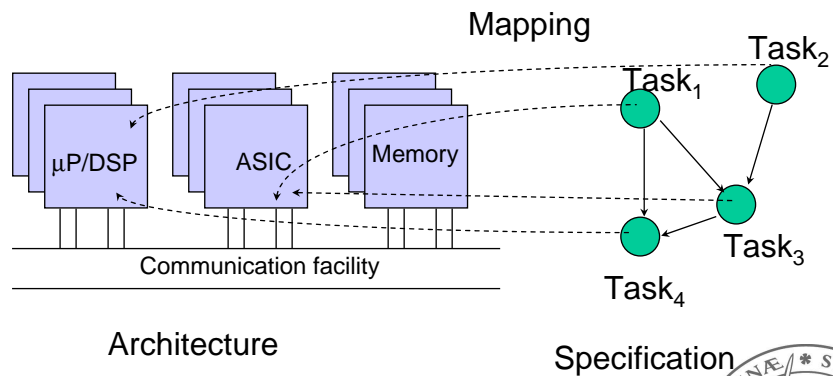
19

Other Synthesis Problems Defined with Constraints

- High-level synthesis:
 - Chaining,
 - Conditional execution,
 - Pipelined components,
 - Algorithmic pipelining,
 - Switching activity reduction (power consumption)
 - ...
- System design
 - different aspects of design space exploration
 - scheduling
 - component assignment
 - memory allocation/data assignment
 - power/energy consumption
 - ...



Design Space Exploration



21



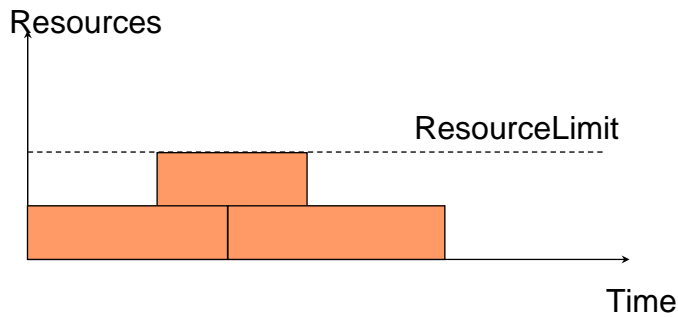
Additional Constraints element

- Element constraints
 - $\text{element}(N, [X_1, X_2, \dots, X_n], \text{Value})$
 - propagation from N to Value
 $N=i \rightarrow \text{Value} = X_i$
 - propagation from Value to N
 $\text{Value} = x \rightarrow N=i \text{ and } X_i = x \dots$
- Examples- $\text{element}(N, [2, 3, 4, 4], V)$
 - $N :: 1..2, V :: \{2, 3\}$
 - $V = 4, N :: 3..4$
- Used to define discrete cost functions and different relations



Additional constraints cumulative

Cumulative constraint

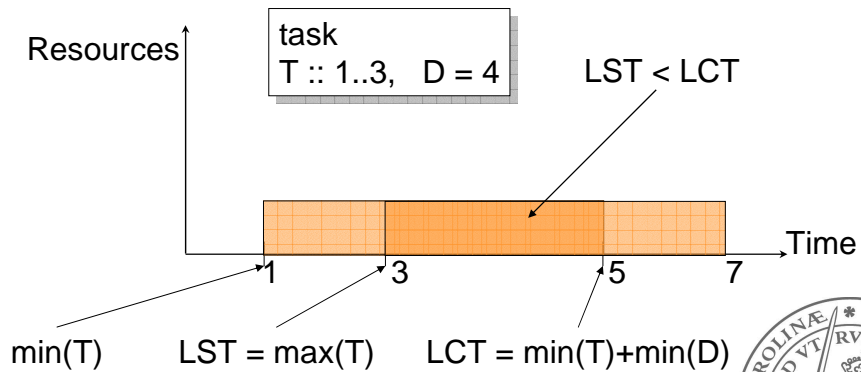


`cumulative([Tk, Tn, Tm], [Dk, Dn, Dm], [1, 1, 1], ResourceLimit)`

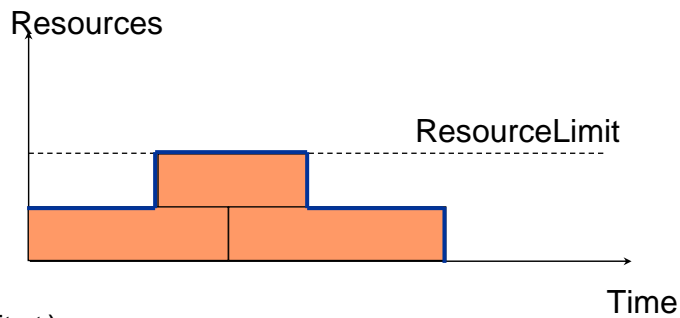


Cumulative propagations

- Execution interval which will always be occupied by a task.



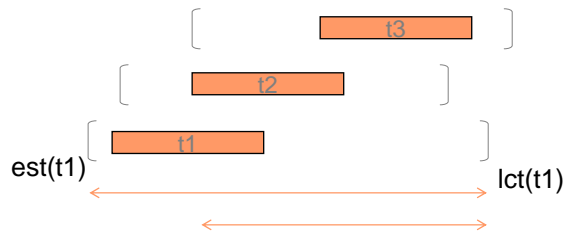
Cumulative propagations — profile based



for each $[t_i, t_j)$
 for each task $_n$ whose exec. interval overlaps with $[t_i, t_j)$
 if $(ResourceLimit - resource_usage < task_n(resources))$
 T_n in $\{ complement(t_i - min(D_n) + 1 .. t_j - 1) \}$
 check $D_n \dots Res_n \dots$



Cumulative propagations — edge finding



t3 cannot be between t1 and t2 iff
 $lct(t1) - est(t1) < D1 + D2 + D3$

t3 cannot be before t1 and t2 iff
 $lct(t1) - est(t3) < D1 + D2 + D3$

t3 must be last !!!

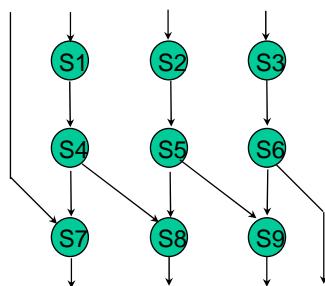


Edge Finding Algorithm

- Martin-Shmoys algorithm with $O(n^2)$ complexity.
- Up phase
 - for each unique lct we create a set $S = \{t \mid LCT(t) \leq lct\}$ and make checking whether a task can be the first or before
- Down phase
 - similar but using est and checking whether a task can be the last one or after all tasks.



System Synthesis Example



- original MILP formulation- 47 timing variables, 225 binary (bus 153) and 1081 constraints (bus 416)
- commercial linear programming package used to solve the problem (XLP, developed by XMP Software, Inc.)

Processor	Cost	Execution time								
		S1	S2	S3	S4	S5	S6	S7	S8	S9
P1	4	2	2	1	1	1	1	3	-	1
P2	5	3	1	1	3	1	2	1	2	1
P3	2	1	1	2	-	3	1	4	1	4



28

Modeling of cost and execution time

- Execution time
 $\text{element}(P1, [2, 3, 1], D1)$
 ...
 $\text{element}(P9, [1, 1, 4], D4)$
- Cost
 $(P1=1 \vee P2=1 \vee \dots \vee P9=1) \Leftrightarrow C1,$
 ...
 $(P1=6 \vee P2=6 \vee \dots \vee P9=6) \Leftrightarrow C6,$
 $\text{Cost} = 4 \cdot C1 + 4 \cdot C2 + 5 \cdot C3 + 5 \cdot C4 + 2 \cdot C5 + 2 \cdot C6$



System synthesis results

Design	Cost	Performance (time units)	Performance optimization		Cost optimization		
			MILP (s)	CLP (s)	B&B Nodes	CLP (s)	B&B Nodes
Bus	10	6	6438.00	0.43	84	0.55	92
	6	7	5371.80	0.53	114	0.68	144
	5	15	3691.20	0.43	68	0.70	103
point-to-point links	15	5	3732.00	0.43	20	1.67	125
	12	6	26710.20	1.42	98	2.18	169
	8	7	32320.20	1.00	58	2.59	198
	7	8	4510.80	1.64	75	2.02	112
	5	15	38501.20	1.50	32	1.48	77



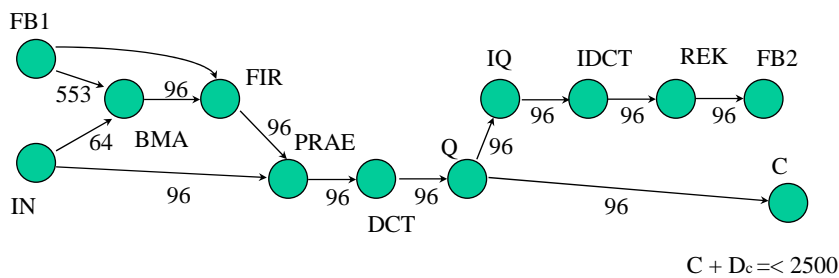
System synthesis results with local memory

Design	Cost	Performance (time units)	Performance optimization			Cost optimization	
			MILP (s)	CLP (s)	B&B Nodes	CLP (s)	B&B Nodes
Bus	28	6	6592.20	0.71	76	2.58	252
	23	7	5371.80	1.07	193	1.94	266
	22	8	123252.60	0.95	124	14.85	856
	21	10	316860.60	114.92	4 534	119.55	8 799
	18	11	236724.00	88.23	7 015	2.37	477
	17	12	138004.20	0.93	268	10.39	3 076
	14	15	3581.40	0.54	22	9.89	3 076
point-to-point links	38	5	-	0.56	24	2.08	107
	30	6	-	0.99	59	3.75	155
	25	7	-	1.60	79	5.58	314
	23	8	-	1.82	57	3.21	184
	22	10	-	4.50	84	59.25	855
	19	11	-	27.34	794	101.03	2 851
	18	12	-	97.72	2 686	8.66	1 047
14	15	-	1.18	14	4.95	328	



31

An Example



$$C + D_c \leq 2500$$

Video Coding Algorithm H.261



32

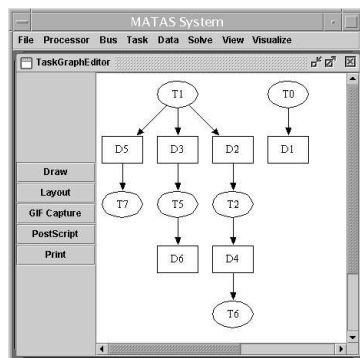
Task Mappings to Processors

Task	Uni- versal	BMA array	PAR1	DCT array	FIR array	BMA pipe	FIR seq	FIR pipe	DCT seq	DCT pipe
IN	-	-	-	-	-	-	-	-	-	-
FB1	-	-	-	-	-	-	-	-	-	-
BMA	7234	484	-	-	-	3617	-	-	-	-
FIR	7234	-	-	-	510	-	3461	1170	-	-
PRAE	1280	-	128	-	-	-	-	-	-	-
DCT	12312	-	-	132	-	-	-	-	6156	474
Q	-	-	-	-	-	-	-	-	-	-
IQ	-	-	-	-	-	-	-	-	-	-
IDCT	12312	-	-	132	-	-	-	-	6156	474
REK	1536	-	256	-	-	-	-	-	-	-
C	132	-	-	-	-	-	-	-	-	-
FB2	-	-	-	-	-	-	-	-	-	-

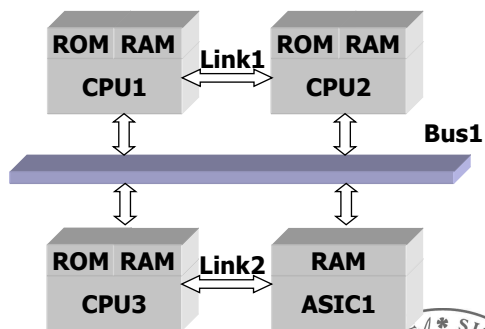


33

Scheduling with Memory Constraints



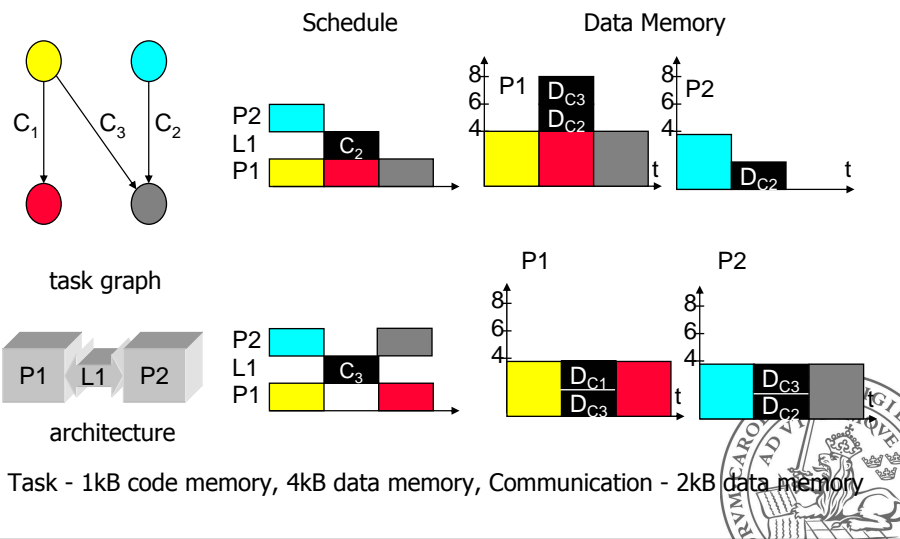
annotated task graph



target architecture



Memory importance



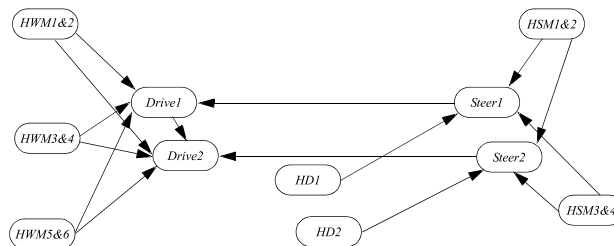
Experimental results H.261 example

EXP. #	ALGORITHM	PIPELINE DEGREE	DEADLINE	AVERAGE EXECUTION TIME	Σ DATA MEMORY	Δ TIME	Δ DATA MEM.
1	both	1	2871	2871	2683	-	-
2	greedy	4	6743	1686	3812	0	0
3	memory	4	6781	1696	3259	1%	-16%



Scheduling of Mars Path Finder under Power Consumption Constraints

The Mars rover operates on a very limited power supply. The power is provided by solar panels. The power obtained from solar panels was measured at different temperatures, and the results were as follows: 14.9W at -40°C, 12.0W at -60°C, and 9.0W at -80°C. There is also a battery power source, which provides a maximum of 10.0W, and it is not a replenishable energy source, so the battery power should be used as little as possible. The Mars rover has 6 driving and 4 steering motors, which need to be warmed up before their respective driving and steering can be performed.



Scheduling of Mars Path Finder under Power Consumption Constraints

Operation	Duration	
Heating steering motors (HSM1&2, HSM3&4)	5s	At least 5s and at most 50s before steering starts
Heating wheel motors (HWM1&2, HWM3&4, HWM5&6)	5s	At least 5s and at most 50s before driving starts
Hazard detection (HD1 & HD2)	10s	At least 10s before steering starts
Steering (Steer1, Steer2)	5s	At least 5s before driving
Driving (Drive1, Drive2)	10s	At least 10s before next hazard detection starts



Scheduling of Mars Path Finder under Power Consumption Constraints

Task	Duration	Power -40°C	Power -60°C	Power -80°C
Heat two motors	5s	7.6W	9.5W	11.3W
Drive	10s	7.5W	10.9W	13.8W
Steer	5s	4.3W	6.2W	8.1W
Hazard Detection	10s	5.1W	6.1W	7.3W
CPU	Constant	2.5W	3.1W	3.7W

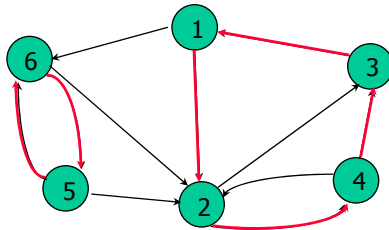


Modeling

- Precedence constraints:
 - $t_{hd1} + d_{hd1} \leq t_{steer1},$
 - ...
 - $t_{steer1} + d_{steer1} \leq t_{drive1},$
 - $t_{hwm12} \leq t_{steer1} + 50,$
- Power consumption constraints:
 - $cumulative([t_{hd1}, \dots, t_{sm12}],$
 - $[p_{hd1}, \dots, p_{sm12}],$
 - $[d_{hd1}, \dots, d_{sm12}], Power)$
- Optimize "Power"



Cycle (circuit) constraint



cycle(2, [[2,6], [3,4], [1], [2,3], [2,6], [2,5]])



[[2],[4], [1], [3], [6], [5]]



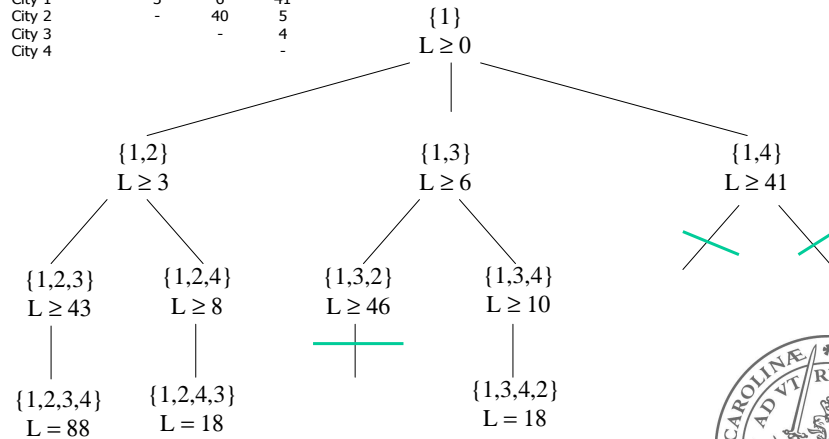
Search

- Standard search uses depth-first-search with backtracking.
- Optimization uses branch-and-bound or similar methods.



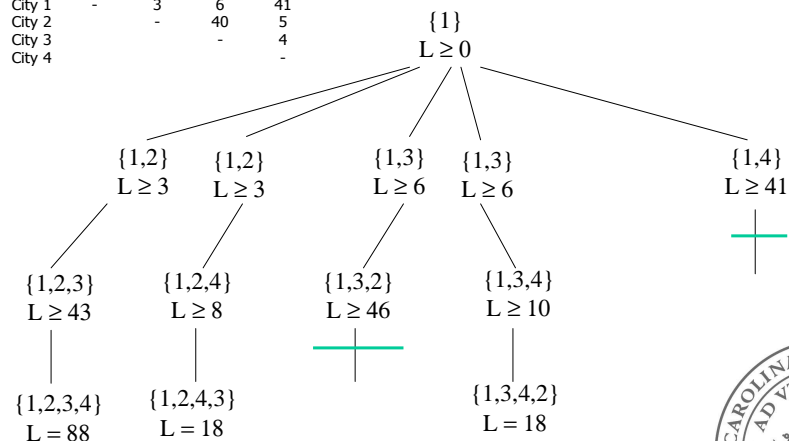
Typical branch and bound search (TSP problem)

	City 1	City 2	City 3	City 4
City 1	-	3	6	41
City 2	-	-	40	5
City 3	-	-	-	4
City 4	-	-	-	-



Search with restart (CLP typical)

	City 1	City 2	City 3	City 4
City 1	-	3	6	41
City 2	-	-	40	5
City 3	-	-	-	4
City 4	-	-	-	-



Search (cont'd)

[City1::2..4, City2::{1,3..4}, City3::{1..2,4}, City4::1..3]

- How to select order of variable assignment?
 - dynamic vs. static
 - criteria
- How to select values to be assigned from variable's domain?
 - a single value
 - sub-domain
 - ...



Variable Selection

- Static and dynamic
 - input order (static)
 - first-fail principle (smallest size of the domain)
 - smallest value in the domain
 - largest value in the domain
 - largest difference between the smallest and second smallest value in its domain
 - smallest max value in the domain
 - ...



Value Selection

- Single value
 - minimum in the domain and then upwards
 - maximum in the domain and then downwards
 - middle and then towards smallest and largest
 - random
 - ...
- Domain split
 - split into two sub-domains
 - split into N
 - ...

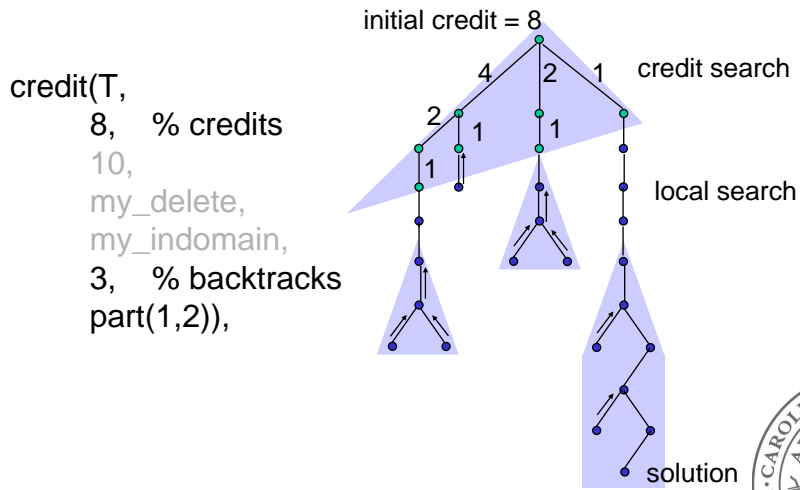


Search improvements

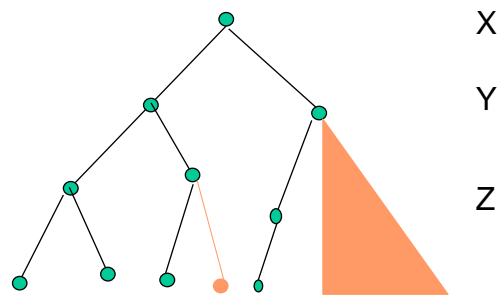
- Partial enumeration algorithms (instead of labeling)
 - Credit Search,
 - Limited Discrepancy Search (LDS).
- Assignment of subintervals instead of values to domain variables — possibly examines a bigger part of a solution space.
- Problem-dependent specific heuristics.
- Neighbourhood search...



Credit search

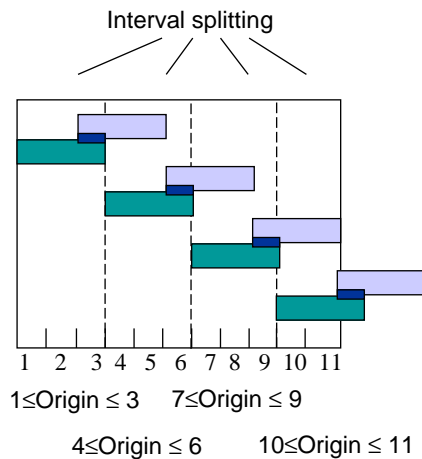


Limited discrepancy search



min_max(lds ([X,Y,Z], 1, input_order, indomain), Cost)

Interval splitting



For each task:

```
Origin :: min..max
duration

Rest    :: 0..duration-1,
Quotient :: 0..max,
Quotient*duration+Rest #= Origin.
```

Enumeration procedure:

```
labeling(Origins, Quotients) :-
    labeling(Quotients, first_fail, indomain),
    labeling(Origins, first_fail, indomain).
```

Summary and conclusions

- **Advantages:**
 - focus on a specification of the problem, not on a solution method.
 - unified framework for different algorithms to be used to solve a problem (by encapsulating them as constraints).
 - easy definition of problems with many heterogeneous constraints.
 - easy extension of a problem by adding new constraints.
 - collaboration of solvers and distributed computing

Summary and conclusions

- **Limitations:**
 - NP-hard problems.
 - often non-predictable behavior of a solver.
 - difficult to define and add new constraints:
 - into existing systems — interface problems.
 - new propagation algorithms need to be developed.
 - difficult to match constraints with actual problems.



CP finite domain systems

- SICStus Prolog
- CHIP from COSYTEC
- IF/Prolog
- ILOG
- Mozart/Oz
- Gnu Prolog
- JaCoP – Java based our own solver



Selected CP Web resources

- Constraints archive
<http://www.cs.unh.edu/ccc/archive>
- Guide to constraints programming
<http://kti.ms.mff.cuni.cz/~bartak/constraints>
- Sicstus manual
http://www.sics.se/isl/sicstus/sicstus_toc.html
- Gnu Prolog
<http://www.gnu.org/software/prolog/prolog.html>
- Mozart/Oz
<http://www.mozart-oz.org/>



Other resources

- Book
 - K. Mariott and P. J. Stuckey *Programming with Constraints: An Introduction*, The MIT Press, 1998.
- Conferences
 - Principles and Practice of Constraint Programming (CP)
 - The Practical Application of Constraint Technologies and Logic Programming (PACLP)
- Journal
 - Constraints (Kluwer Academic Publishers)



Selected Papers

- Kuchcinski, K., *Embedded System Synthesis by Timing Constraints Solving*, Proc. 10th International Symposium on System Synthesis, Antwerp, Belgium, September 17-19, 1997.
- Gruian, F. and Kuchcinski, K., *Operation Binding and Scheduling for Low Power Using Constraint Logic Programming*, Proc. 24th Euromicro Conference, Workshop on Digital System Design, Västerås, Sweden, August 25-27, 1998.
- Kuchcinski, K. and Wolinski, Ch., *Global Approach to Assignment and Scheduling of Complex Behaviours based on HCDG and Constraint Programming*, Journal of Systems Architecture, 2003, Elsevier Science.



Selected Papers (cont'd)

- Kuchcinski, K., *Constraints Driven Design Space Exploration for Distributed Embedded Systems*, Journal of Systems Architecture, vol. 47, no. 3-4, pp. 241-261, 2001, Elsevier Science.
- Szymanek, R. and Kuchcinski, K., *A Constructive Algorithm for Memory-Aware Task Assignment and Scheduling*, Proc. 9th International Symposium on Hardware/Software Codesign, Copenhagen, Denmark, Apr. 2001.
- Szymanek, R. and Kuchcinski, K., *Partial Assignment Technique for Task Graph Scheduling*, 40th DAC, Anaheim, USA, June 2003.
- Kuchcinski, K. *Constraint-driven scheduling and resource assignment*, ACM Trans. on Design Automation of Electronic Systems, vol. 8, no. 3, pp. 355-383, 2003.





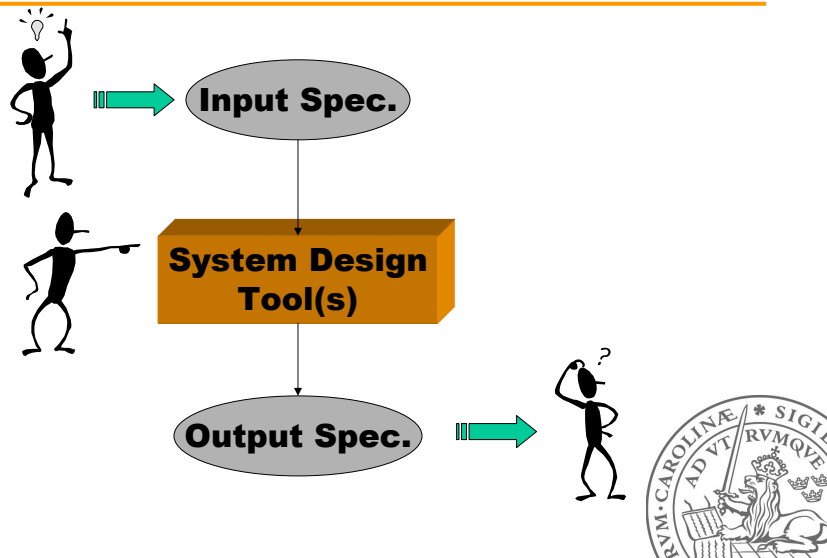
LUND INSTITUTE
OF TECHNOLOGY
Lund University

Methodologies for Embedded System design

Kris Kuchcinski
Dept. of Computer Science
Lund University
Sweden

<http://www.cs.lth/~kris>

General Methodology



Input to System Design

- Executable specification (functional requirements):
 - usually provided as interacting processes/tasks,
 - very often multi-language specifications,
 - can be simulated and verified,
 - can be used to perform analysis, e.g, estimation.
- Specification languages: C, C++, VHDL, Verilog, SystemC, Esterel, SDL, etc.
- Set of (non-functional) design requirements (cost, speed, I/O rate, power consumption, etc.).

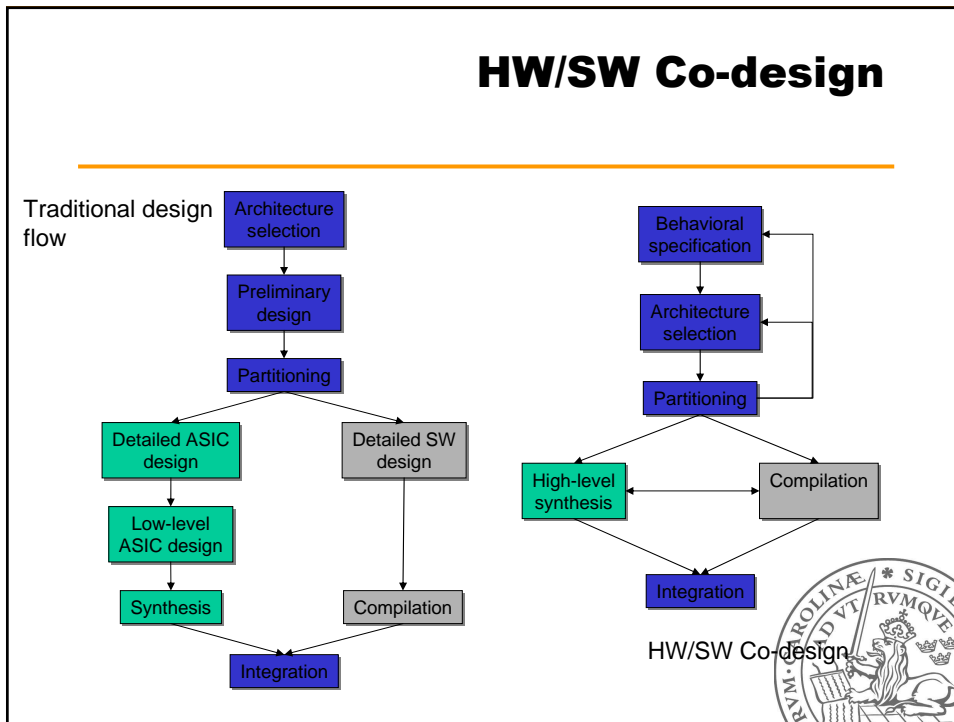


Output from System Design

- A set of system modules assigned to system components (CPU's, DSP's, ASIC's, etc.).
- Communication modules.
- Each module can be further synthesized to hardware using *high-level synthesis* or *compiled to software*.



HW/SW Co-design



Basic Characteristics of the Methodology

- *Behavioral specification* is given for the complete heterogeneous system, regardless of how different parts will be later implemented.
- *Analysis techniques* are provided; specially different estimation techniques.
- *Synthesis tools* are used to automatically explore a design space.
 - high-level synthesis, RTL synthesis,
 - compilers, cross-compilers,
 - interface generators,
 - etc.

Estimation of Design Parameters

- Estimation of parameters such as size, cost, power consumption.
- Does not need to be very precise but has to be “consistent” — follows real design parameters.
- Usually 15%-20% inaccurate.
- Trade-off between accuracy and estimation time.

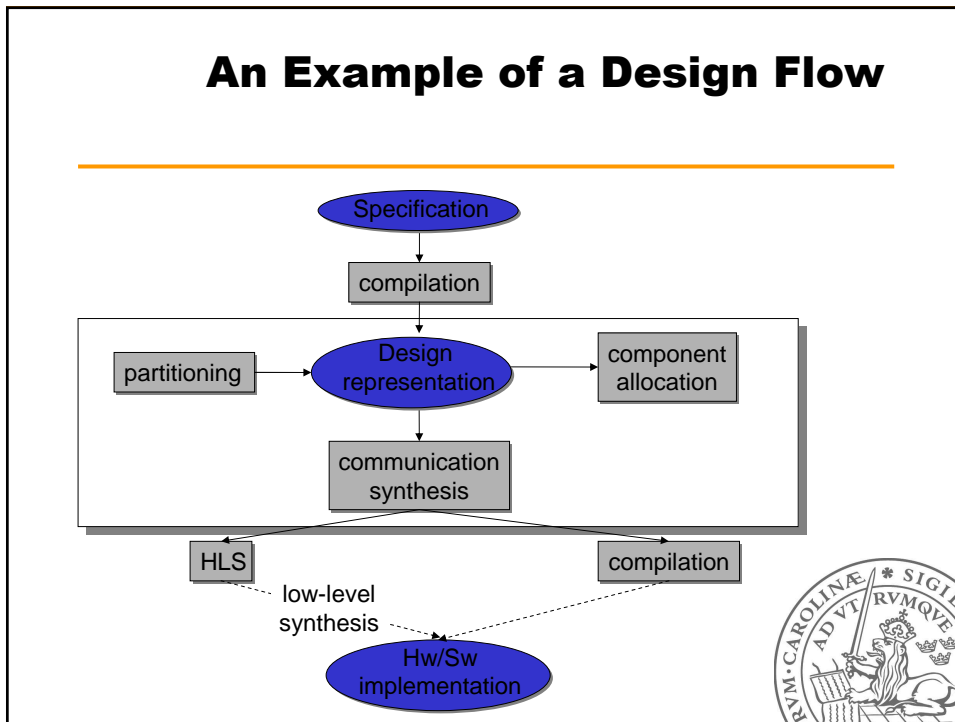


Improvements of the Design Process

- High-level specification is made before architecture selection and implementation *decisions can be made more accurate* (better exploration of architectures).
- A uniform description of HW and SW makes it possible to *move parts of the systems between HW and SW*.
- HW and SW development is moved closer and the *integration cost is reduced*.
- An *early evaluation of system characteristics* is possible.



An Example of a Design Flow



Specification Example

“Extended” VHDL

```

port(IP1,IP2:in INTEGER; OP1,OP2:out INTEGER);
...
signal S1,S2,S3,S4,S5,S6:INTEGER;
P1 : process
...
  receive(IP1);
...
  send(S1,...);
...
  send(S3,...);
...
  receive(S6);
...
end process P1;

P2 : process
...
  receive(IP2);
...
  receive(S1);
...
  send(OP2,...);
...
end process P2

P3 : process
...
  receive(S4);
...
  send(S2,...);
...
end process P3;

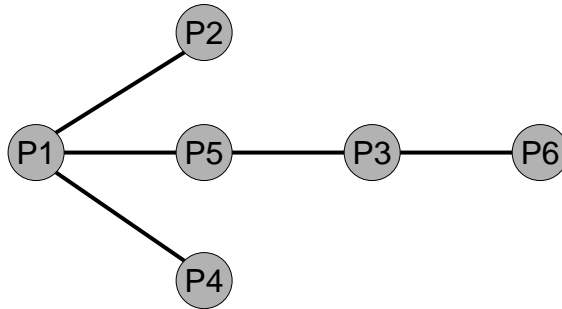
P4 : process
...
  receive(S3);
...
  send(S5,...,S6,...);
...
end process P4;

P5 : process
...
  receive(S1,S5);
...
  send(S4,...);
...
end process P5;

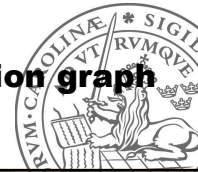
P6 : process
...
  receive(S2);
...
  send(OP1,...);
...
end process P6;

```

Representation Example



Process communication graph



Allocation of System Components

- Decides about the kind and number of components for implementation of the system
 - processing elements: μ procesosrs, micro-controllers, DSP's, ASIP's, ASIC's, FPGA's, etc.
 - storage elements: memories, register files, registers, etc.
 - communication devices: buses, point-to-point links, networks, etc.
 - specialized I/O devices: A/D, D/A, frame grabbers, etc.

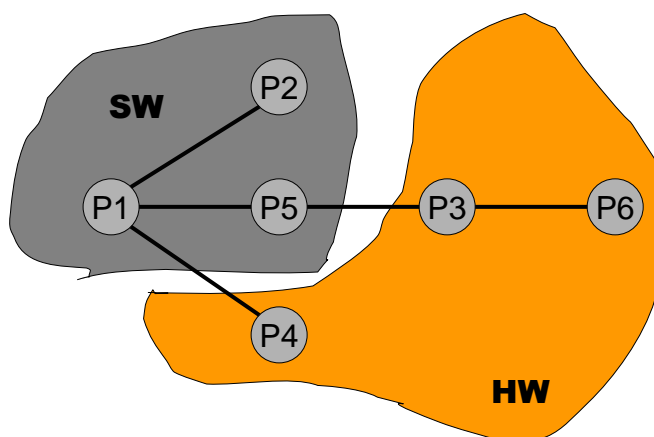


Partitioning

- *Functional* partitioning vs. *structural* partitioning.
- Abstraction level.
- Partitioning granularity (*fine* or *course*):
 - modules,
 - processes and procedures,
 - instructions.
- Partitioning objective:
 - performance,
 - minimal communication,
 - low power,
 - combination of several criteria.



Partitioning Example

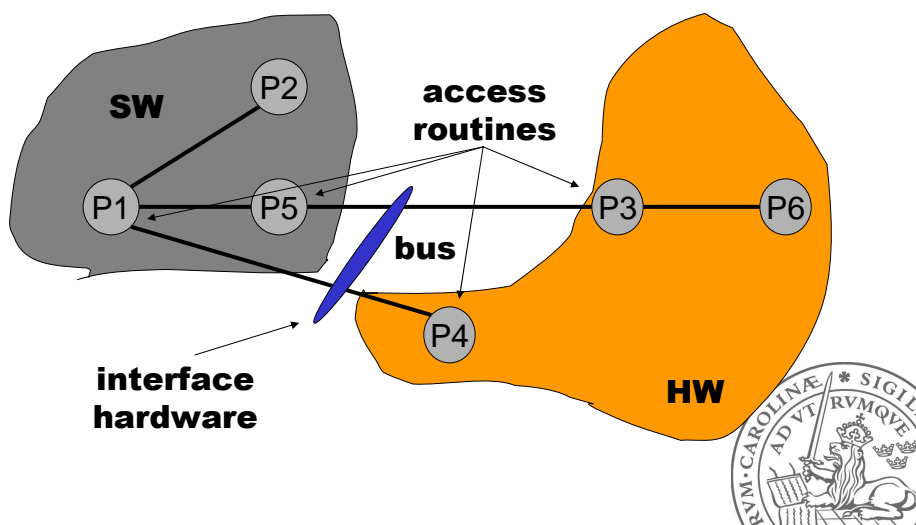


Communication Synthesis

- Creation of abstract communication channels by communication clustering.
- Communication refinement
 - selection of communication lines width,
 - protocol selection,
 - etc.
- Interface generation:
 - device drivers,
 - communication hardware,
 - etc.



Communication Synthesis Example



Design Decisions

- Different types of design decisions
 - selection of components, partitioning, assignments, scheduling, etc.
 - decisions regarding runtime system done off-line or are postponed to runtime (e.g., static vs. runtime scheduling)
- Design decisions are mutually dependent
- Huge design space



Design Automation

- Uses internal representations which are usually based on graphs.
- Graph algorithms (shortest path, Hamiltonian circuit, topological sort, depth-first-search, breadth-first-search, SAT, etc.).
- Optimization methods — (M)ILP, CLP, heuristics, etc.
- *Tractable* and *intractable* problems.
- *Decidable* and *undecidable* problems.
- *Decision problems* and *combinatorial optimization problems*.



Design Automation Consequences

- Most of the problems which need to be solved in design automation are NP-complete or NP-hard.
- Usually only small problems can be solved exactly.
- Need for algorithms which do not guarantee optimal solutions but “good enough” solutions
 - *approximation algorithms* — guarantee a solution with a cost that is within some margin of the optimum,
 - *heuristics* — algorithms that are constructed based on “rules-of-thumb”; nothing can be said in advance about the quality of the result.



Examples of Embedded Systems (cont'd)

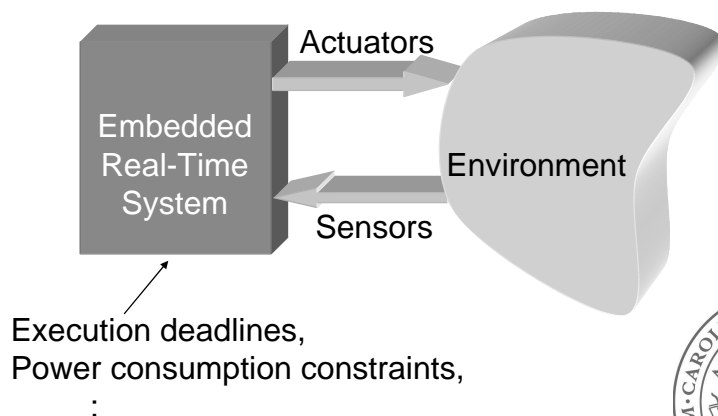
Anti-lock brakes	Modems
Auto-focus cameras	MPEG decoders
Automatic teller machines	Network cards
Automatic toll systems	Network switches/routers
Automatic transmission	On-board navigation
Avionic systems	Pagers
Battery chargers	Photocopiers
Camcorders	Point-of-sale systems
Cell phones	Portable video games
Cell-phone base stations	Printers
Cordless phones	Satellite phones
Cruise control	Scanners
Curbside check-in systems	Smart ovens/dishwashers
Digital cameras	Speech recognizers
Disk drives	Stereo systems
Electronic card readers	Teleconferencing systems
Electronic instruments	Televisions
Electronic toys/games	Temperature controllers
Factory control	Theft tracking systems
Fax machines	TV set-top boxes
Fingerprint identifiers	VCR's, DVD players
Home security systems	Video game consoles
Life-support systems	Video phones
Medical testing systems	Washers and dryers

Source: Embedded Systems Design: A Unified Hardware/Software Introduction, (c) 2000 Vahid/Givargis



Embedded Systems

"A device that includes a programmable computer but is not itself a general-purpose computer."



Embedded Systems (cont'd)

- Computing systems embedded within electronic devices
- Hard to define. Nearly any computing system other than a desktop computer
- Billions of units produced yearly, versus millions of desktop units
- Perhaps 50 per household and per automobile

Source: Embedded Systems Design: A Unified Hardware/Software Introduction, (c) 2000 Vahid/Givargis



Embedded Systems (cont'd)

- Non User-Programmable.
- Based on programmable components (e.g., Micro-controllers, DSP's...) but often contain application specific hardware (IC's, ASIC's).
- Reactive Real-Time Systems:
 - React to external environment,
 - Maintain permanent interaction,
 - Ideally never terminate,
 - Are subject to external timing constraints (real-time).

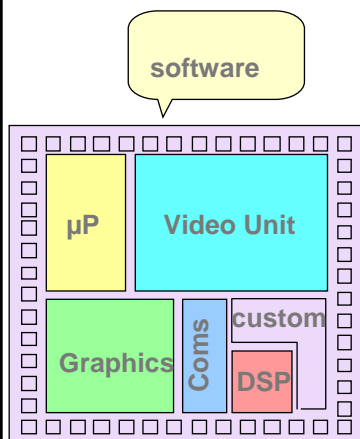


Characteristics of Embedded Systems

- Sophisticated functionality.
- Real-time operation.
- Low manufacturing cost.
- Low power.
- Designed to tight deadlines by small teams.
- “Resource conscious” vs. “Unlimited resources” programming



SoC Embedded System

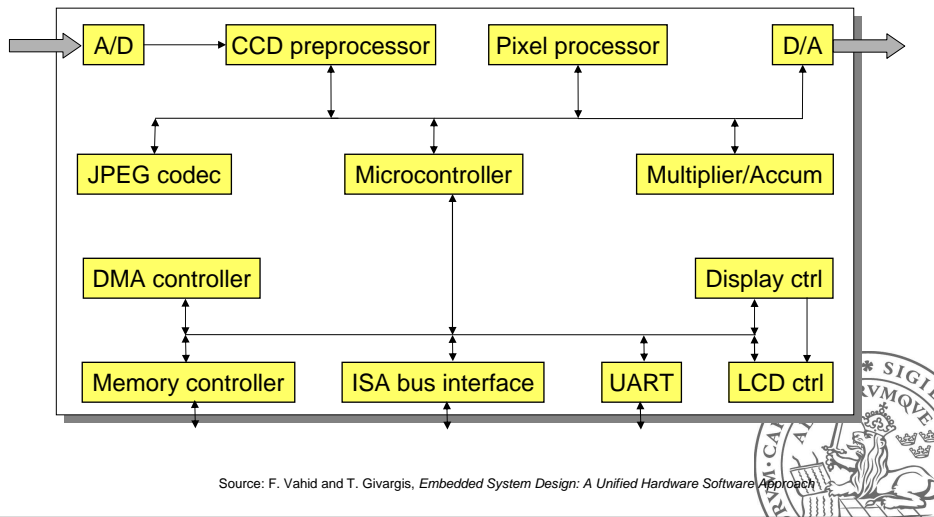


- Assembly of “prefabricated components” often purchased from external vendors (“IP”)
 - “black box” hierarchy
- Design & Verification at the System level
 - rather than the logic level
 - Interface and communication
- Great Importance of Software

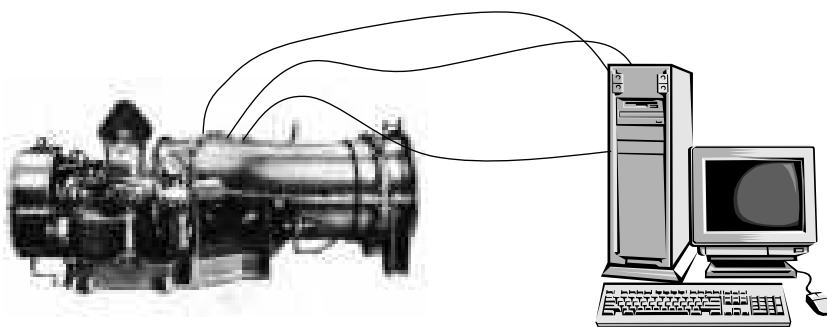
Source: Alberto Sangiovanni-Vincentelli, 35th DAC



A Digital Camera Example



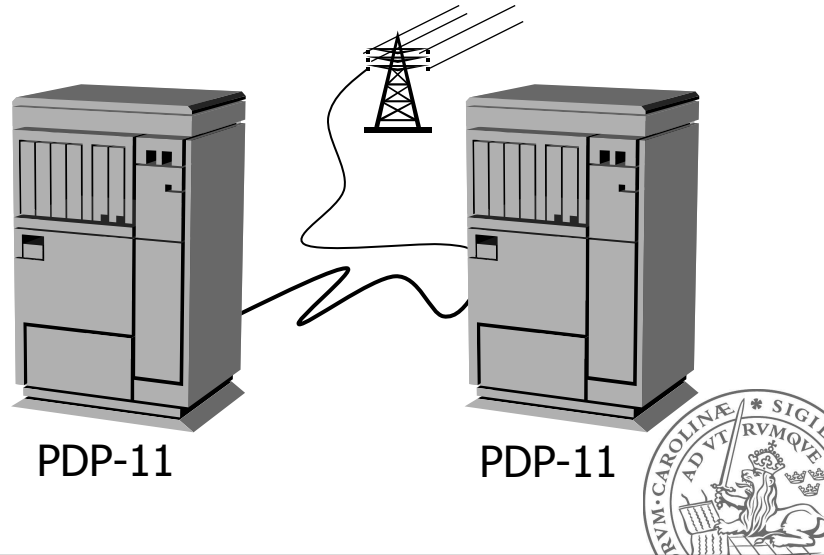
Real-time gas turbine testing system



MI-2 helicopter engine

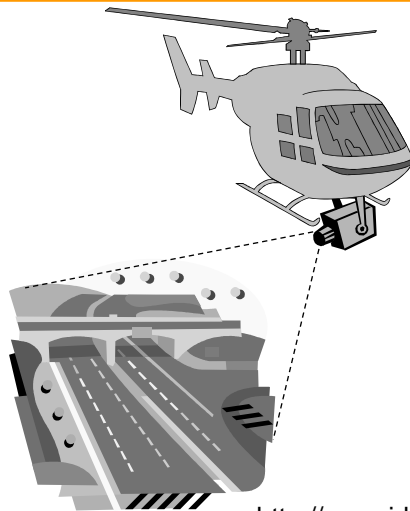
"Minicomputer"
8kB RAM
cassette tape

TELEX-I and TELEX-II systems



WITAS

WITAS project



- Autonomous system.
- Real-time system.
- Image processing.
- Mission planning.
- Incorporation of GIS systems.
- Interface with ground operator.
- ...

<http://www.ida.liu.se/ext/witas>



Typical Hardware Components of DSP System

Component class	Implements	Compiler	Specification
DSP processor	Low data-rate DSP Slow control loops Appl. Spec. alg.	(Retargetable) code generator High level synth.	Assembly C DFL
Microcontroller	User interface Slow control loops	C compiler	C
Hardware accelerator	High data-rate DSP	High level synth. RT level synth.	C, DFL VHDL
Communication blocks and memory	Internal & external communication Storage & buffering	Memory mgmt. (A)synchronous interface synth.	Data-sheets STG
Others	Usually FSM's - clock generators - DMA blocks	RT level synth. Asynchronous synth.	VHDL

Source: H. de Man, et. al. "Co-design of DSP Systems"
Hardware/Software Co-design, Kluwer 1995.

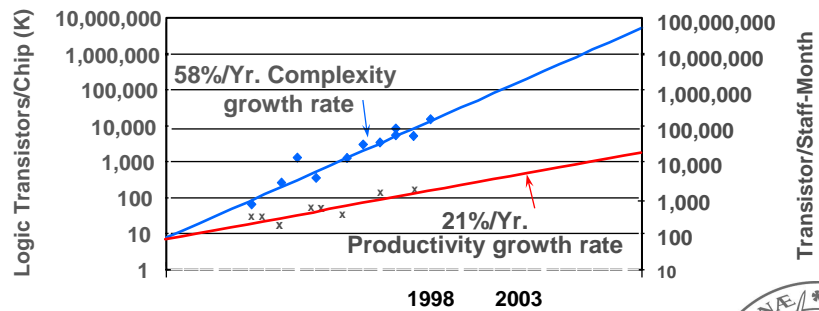


Importance of Embedded System Design Methodologies

- Hardware complexity.
- Heterogeneous systems containing hardware (both digital and analog) and software.
- Heterogeneous components (CPU's, DSP's, ASIC's, buses, point-to-point links, etc.).
- Heterogeneous requirements — performance, cost, power consumption, etc.
- System-on-chip.
- Shorter design cycles required by time-to-market constraints.
- ...



Design Complexity and Designer Productivity Gap



Source: Bryan Preas, Xerox PARC, 35th DAC



Software vs. Hardware Design short summary

- Software
 - flexibility,
 - reconfigurability, easy update, etc.,
 - complex functionality,
 - cost,
 - ...
- Hardware
 - speed,
 - power consumption,
 - cost in large volumes,
 - ...



Design of Embedded Systems

- Need to be done using high-level specification, programming and hardware description languages — not assembly languages and gate/transistor level design.
- Requires efficient design space exploration and synthesis/compilation tools.
- Different design requirements has to be taken into account, e.g., cost, performance, testability, quality of service, power consumption.
- Multi-language design framework.



Importance of High-Level Design Methods

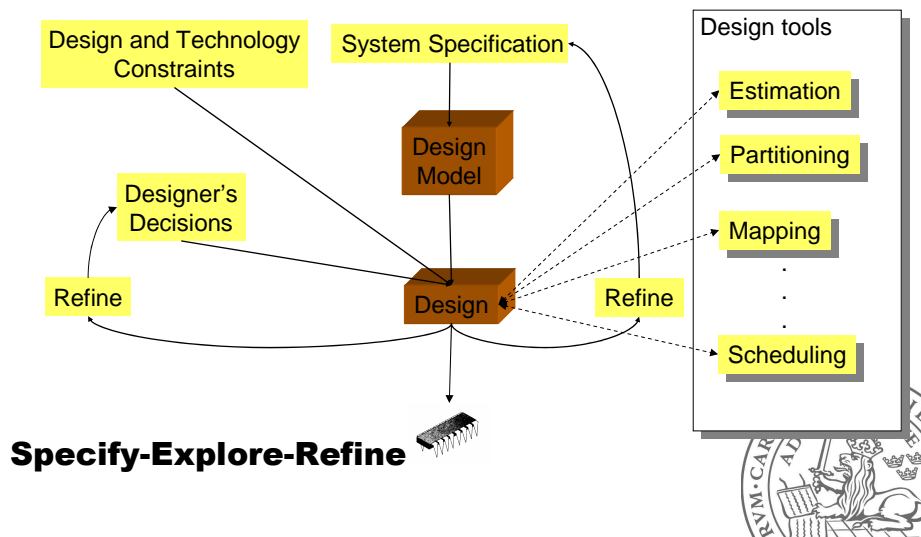
System Verification Processing Speeds

System Implementation	Processing time (s/frame)
Behavioral model	1 200 (20 min/frame)
RTL model	144 000 (1.6 days/frame)
Gate model	228 000 (2.6 days/frame)
Gate model on hardware accelerator	1 200
Rapid Prototype	0.5
Target Hardware	0.05

Source: Paul Clemente, Ron Crevier, Peter Runstadler "RTL and Behavioral Synthesis A Case Study", VHDL Times, vol. 5, no. 1.



General Design Flow



Specification and Programming

- Specification languages, such as UML, SDL.
- Programming languages, such as C, C++, Java, Esterel, assembly languages.
- Hardware description languages, such as VHDL, Verilog, SystemC.
- **Example:** combining SystemC and C++ gives unified simulation environment for hardware and software.



Hardware Description Languages

- Cover several levels of design abstraction as well as behavioral and structural description domain.
- Contain typical features of programming languages, such as data types and program statements.
- Special features:
 - time concept,
 - structure description,
 - parallelism.
- VHDL (IEEE standard), Verilog, SystemC.



Design Representations (Computational Model)

- Used to represent/model digital systems under design.
- Generated by a compiler from system specification or coded directly in the model.
- Represent the semantics, structure and timing of the system.
- Usually based on some kind of annotated graph representation.
- Used internally by design automation systems or by the modeler/designer.

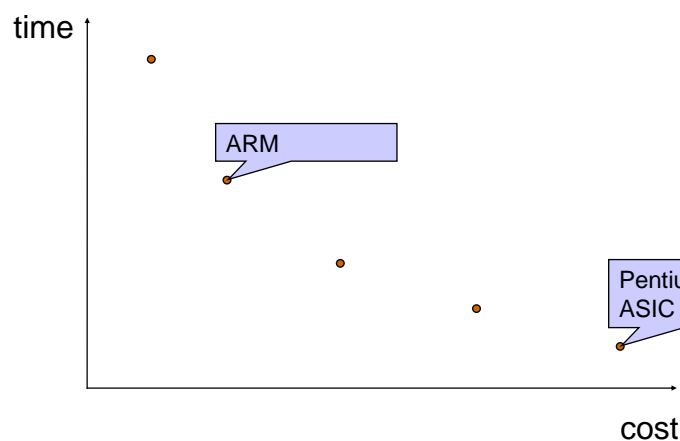


Design — Synthesis

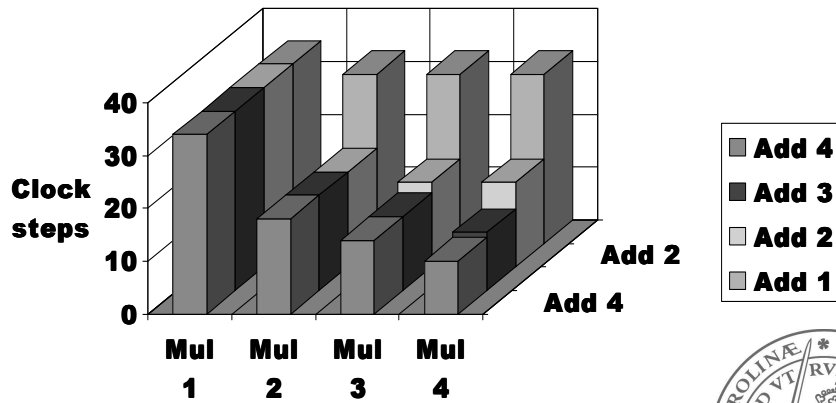
- *Software translation* into target code for a processor (real-time operating system might be used).
- *Hardware synthesis* — translation of a behavioral representation of a design into a structural one.
- *Communication synthesis* — generates hardware and software which interconnects system components.



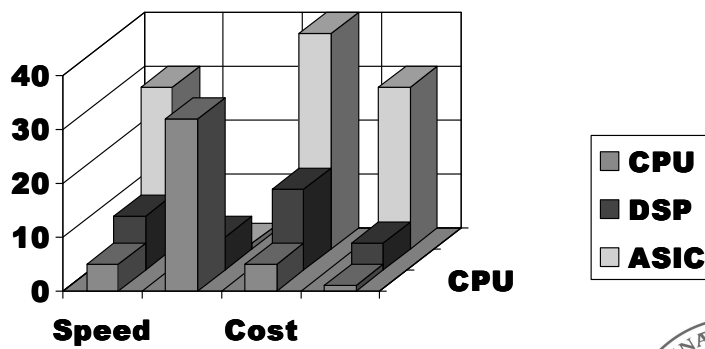
Pareto points



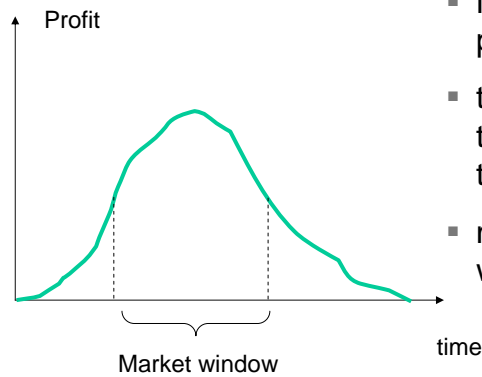
Discrete Cosine Transform Partial Design Space



Design Space Exploration



Time-to-market constraint



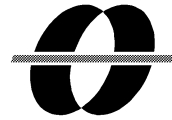
- Need time for new product development,
- the biggest profit is in the market window time,
- missing the market window can be costly.



Summary

- Embedded systems are important class of electronic systems which can be found everywhere,
- Combine hardware and software solutions,
- Cover several engineering and research areas:
 - microelectronics,
 - real-time systems,
 - software development,
 - etc.
- Need careful design which optimizes different design parameters.





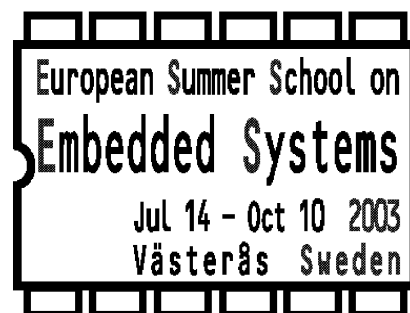
MÄLARDALENS HÖGSKOLA

ESSES 2003

European Summer School on Embedded Systems

Lecture Notes Part XII

Embedded Systems: Introduction and Overview



Editors: Ylva Boivie, Hans Hansson, Jane Kim, Sang Lyul Min

Strängnäs, August 20-22, 2003

ISSN 1404-3041

ISRN MDH-MRTC-106/2003-1-SE

MRTC

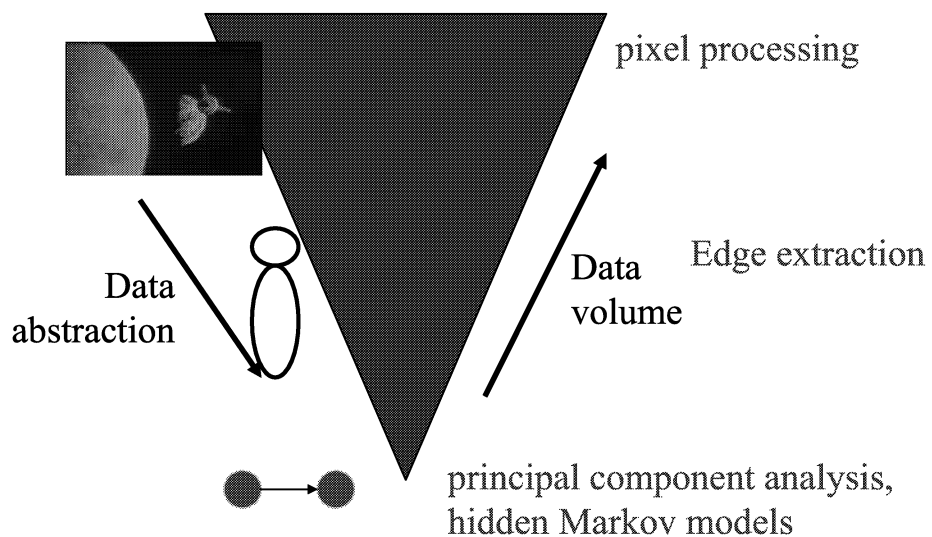
MÄLARDALEN REAL-TIME
RESEARCH CENTRE

www.mrtc.mdh.se

Embedded Computing Examples

Wayne Wolf
Dept. of Electrical Engr.
Princeton University

The multimedia processing funnel



The Parapet Project

⌘ Goal: design SoC networks for real-time distributed vision.

- ☒ The best way to get a good design example is to create our own.
- ☒ Video is a high-performance, low-power, cost-sensitive application.
- ☒ Vision is an important problem.

Parapet goals

⌘ Algorithms (gesture recognition).

- ☒ How do we adapt algorithms to the needs of real-time embedded video.

⌘ Distributed systems.

- ☒ Communicating cameras.

⌘ Embedded software.

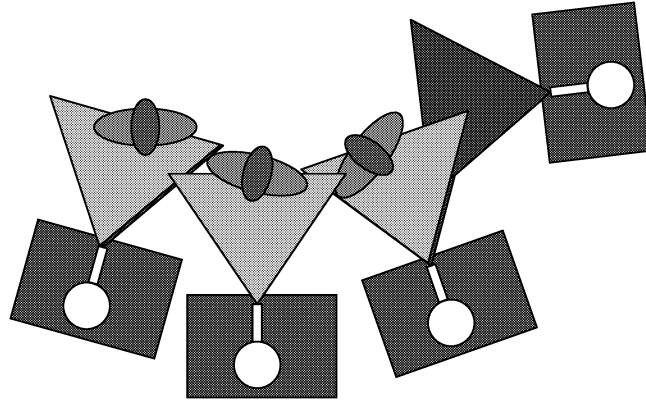
- ☒ Middleware, code optimization.

⌘ SoC architecture.

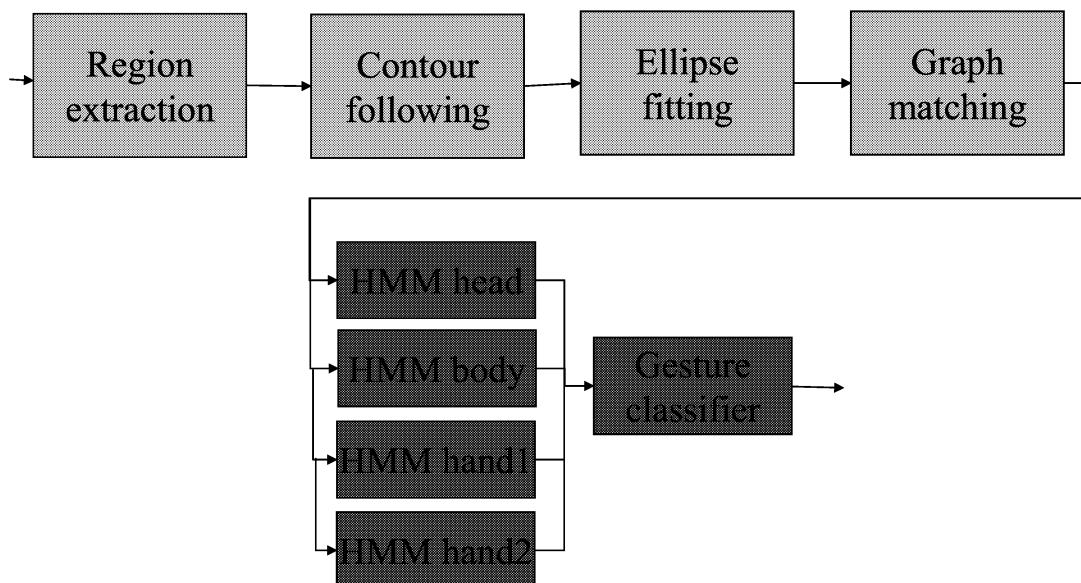
- ☒ Heterogeneous multiprocessors.

Smart cameras for smart rooms

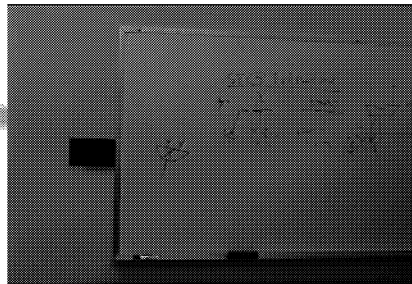
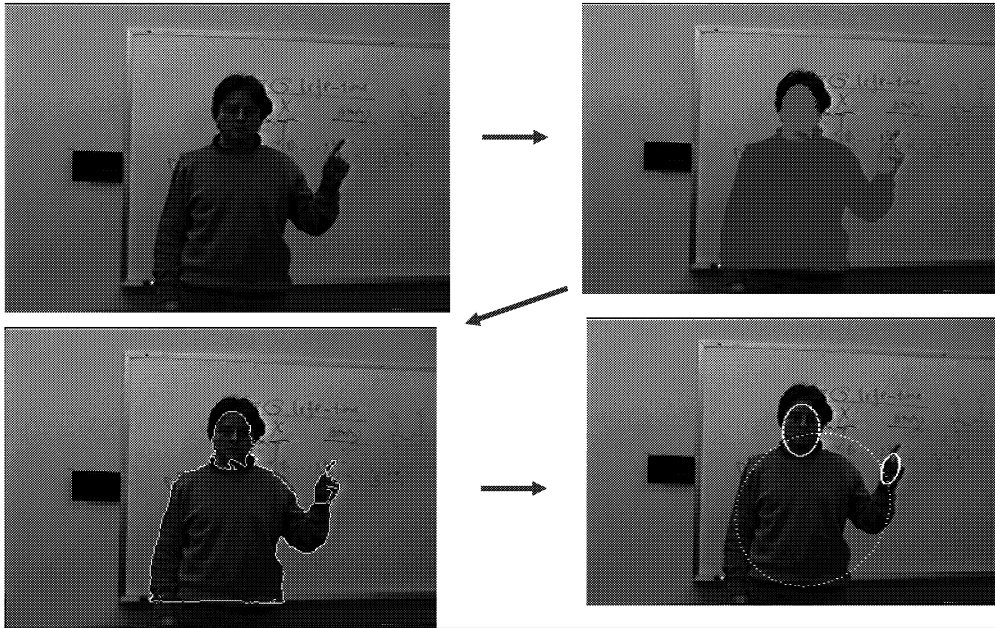
⌘ Coordinated cameras track subject:



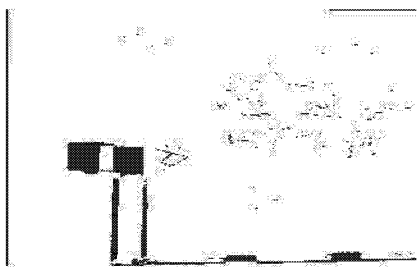
Ozer et al: human activity recognition algorithm



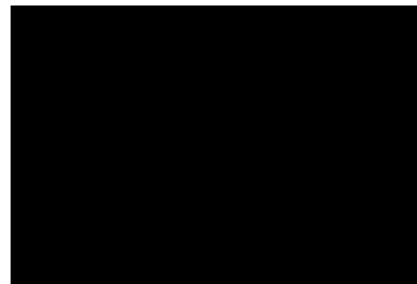
Real-time analysis



Original



Region finding



Ellipse fitting

Tuning the smart camera software

⌘ Initial C/Trimedia was direct translation from Matlab.

⌘ Goals:

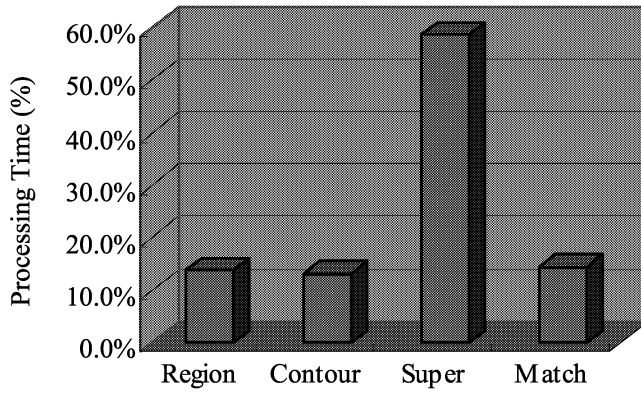
- ⊞ Increase frame rate.
- ⊞ Reduce latency.
- ⊞ Identify bottlenecks for next-generation architecture.

Real-time vs. just fast

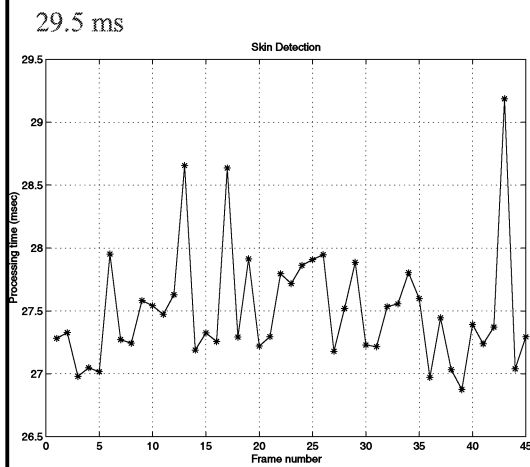
⌘ Real-time computing adheres to constraints:

- ⊞ Must perform at a given rate.
- ⊞ To satisfy the rate, must minimize variations in processing time.

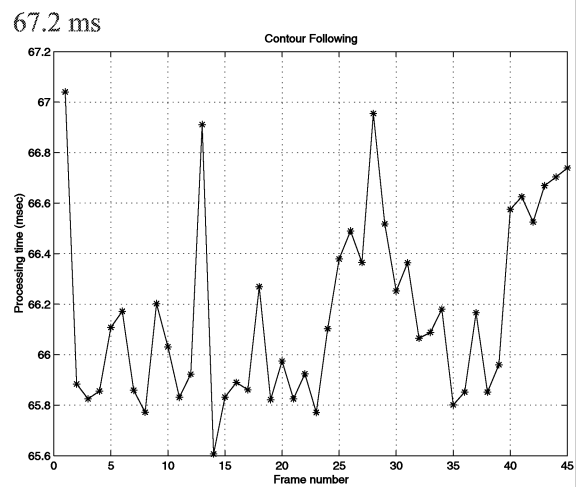
Stage times before optimization



Smart camera CPU times



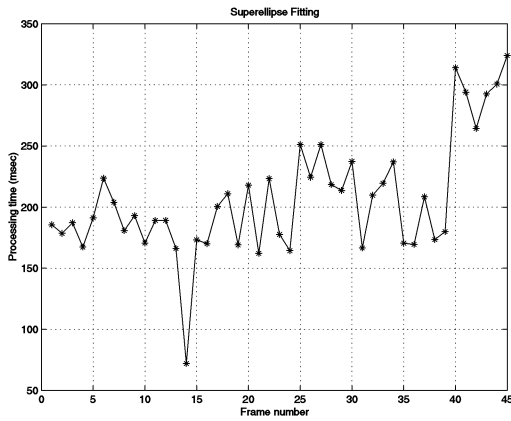
Skin detection



Contour detection

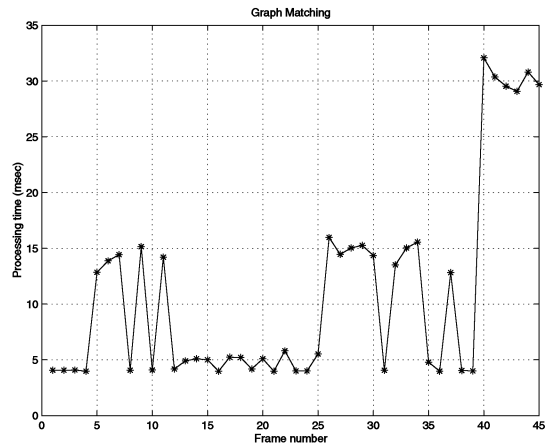
Smart camera CPU times, cont'd.

250 ms



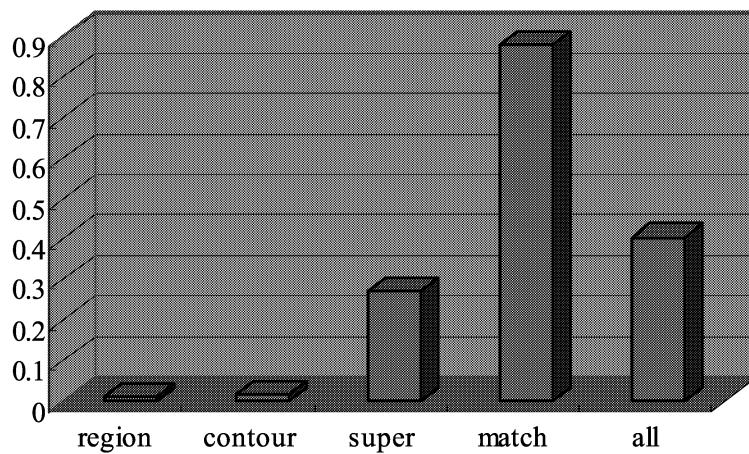
Superellipse fitting

35 ms



Graph matching

Normalized standard deviation of stage times



Optimizations

- ⌘ Change the algorithm.
- ⌘ Change the program structure.
- ⌘ Change the instructions.

Algorithmic changes

- ⌘ Superellipses were expensive to fit and overkill.
 - ☐ Replaced with ellipse fitting.
- ⌘ Improved adjacency algorithm.

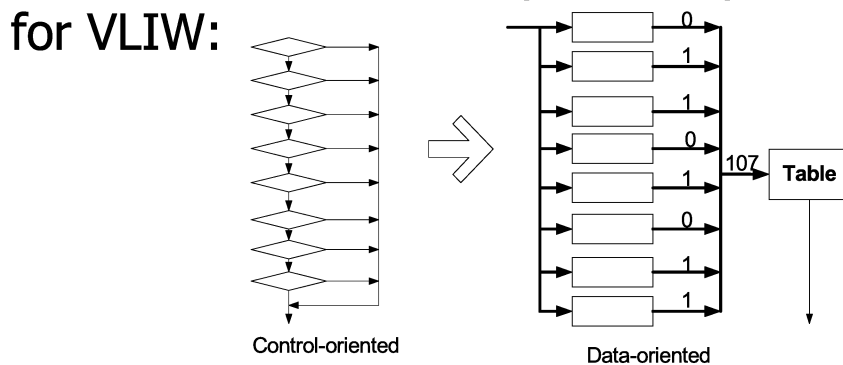
Region finding

- ⌘ Operates on 3 x 3 window.
- ⌘ Roughly linear in frame size.
- ⌘ Sequential algorithm---window moves one pixel per step.



Program changes

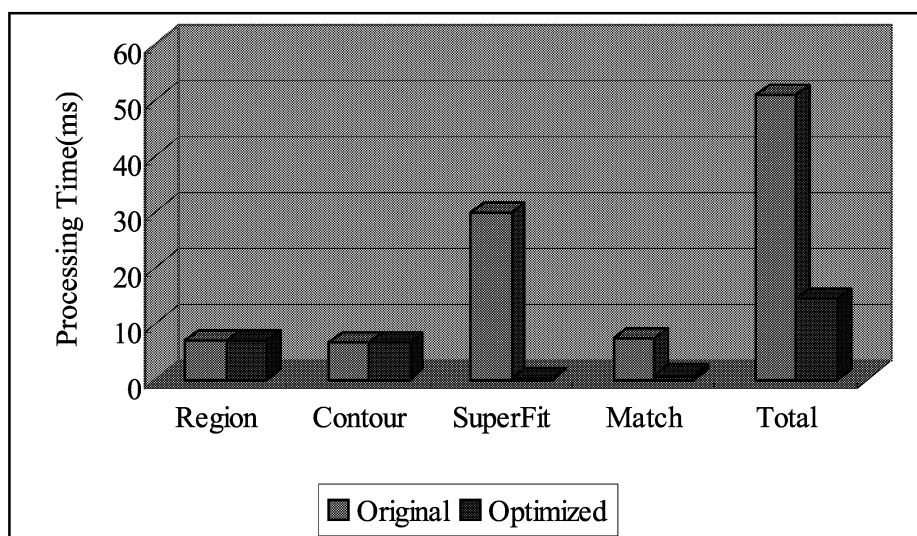
- ⌘ Contour fitting is very control intensive:
 - ☒ Compares local configurations of bits.
- ⌘ Transformed into data-parallel operations for VLIW:



Instruction changes

- ⌘ Trimedia provides library of intrinsic functions that map onto Trimedia instruction sequences.
- ⌘ Goal: eliminate branches.
 - ☒ Special instructions.
 - ☒ Loop unrolling.

Before and after stage times



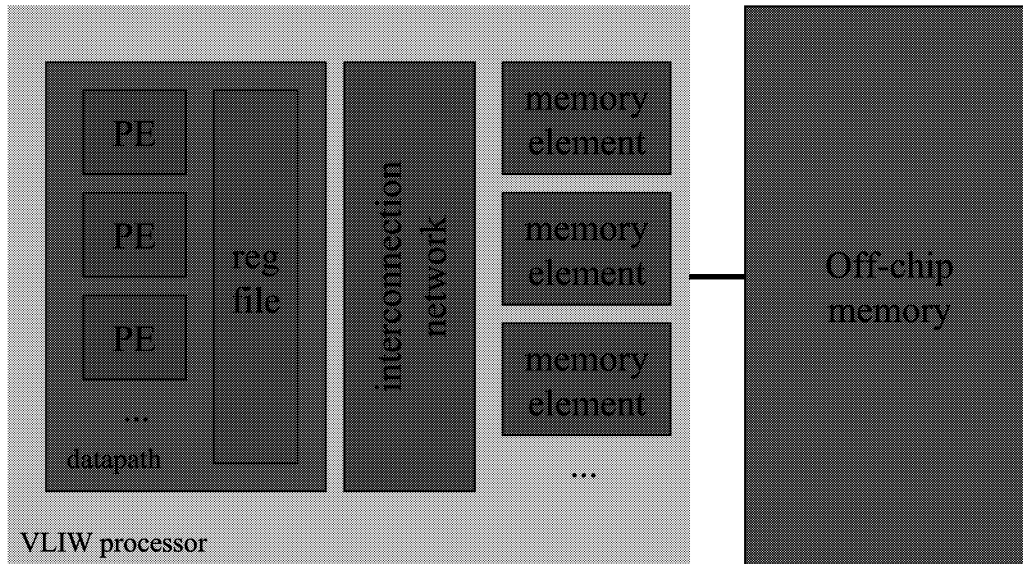
Results

- ⌘ Before: 5 frames/sec.
- ⌘ After: 31 frames/sec w/o HMM, 25 frames/sec with HMM.
- ⌘ Latency approx. 100 ms.
- ⌘ Smaller variation in frame processing time.
- ⌘ Intel port runs well.

Architectural experiments

- ⌘ Fritts/Wolf:
 - ☐ characterize applications;
 - ☐ compare architectural styles (VLIW, superscalar);
 - ☐ evaluate architectural parameters (clock rate, pipelining, etc.).

VLIW processor model



Workload characteristics experiments

- ⌘ Goal: compare media workload characteristics to general-purpose load.
- ⌘ Used MediaBench benchmarks.
- ⌘ Compiled on Impact compiler, measured with with Impact simulator.

Basic characteristics

- ⌘ Comparison of operation frequencies with SPEC
 - ⊞ (ALU, mem, branch, shift, FP, mult) => (4, 2, 1, 1, 1, 1)
 - ⊞ Lower frequency of memory and floating-point operations
 - ⊞ More arithmetic operations
 - ⊞ Larger variation in memory usage
- ⌘ Basic block statistics
 - ⊞ Average of 5.5 operations per basic block
 - ⊞ Need global scheduling techniques to extract ILP

Basic characteristics, cont'd

- ⌘ Static branch prediction
 - ⊞ Average of 89.5% static branch prediction on training input
 - ⊞ Average of 85.9% static branch prediction on evaluation input
- ⌘ Data types and sizes
 - ⊞ Nearly 70% of all instructions require only 8 or 16 bit data types

Multimedia looping characteristics

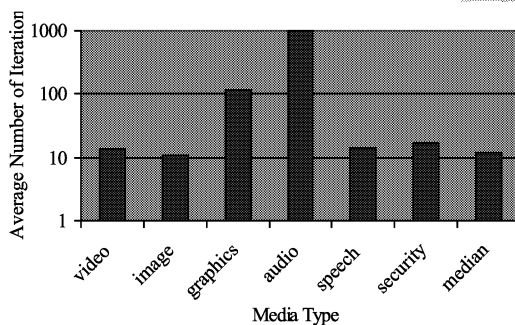
⌘ Highly loop centric

- ⊠ 95% of CPU time in two innermost loop levels
- ⊠ Significant processing regularity
- ⊠ About 10 iterations per loop on average

⌘ Complex loop control

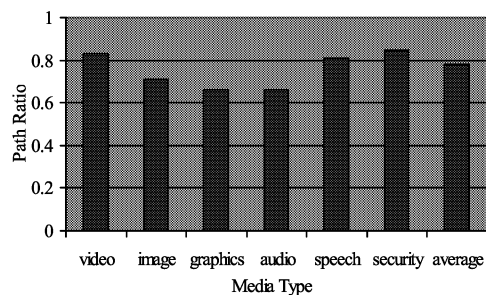
- ⊠ = average # of instructions executed per loop invocation/total # of loop instructions
- ⊠ Average path ratio of 78%--high complexity

Average iterations per loop and path ratio



- average number of loop iterations

- average path ratio



Instruction level parallelism

⌘ Instruction level parallelism

⊞ base model: single issue using classical optimizations only

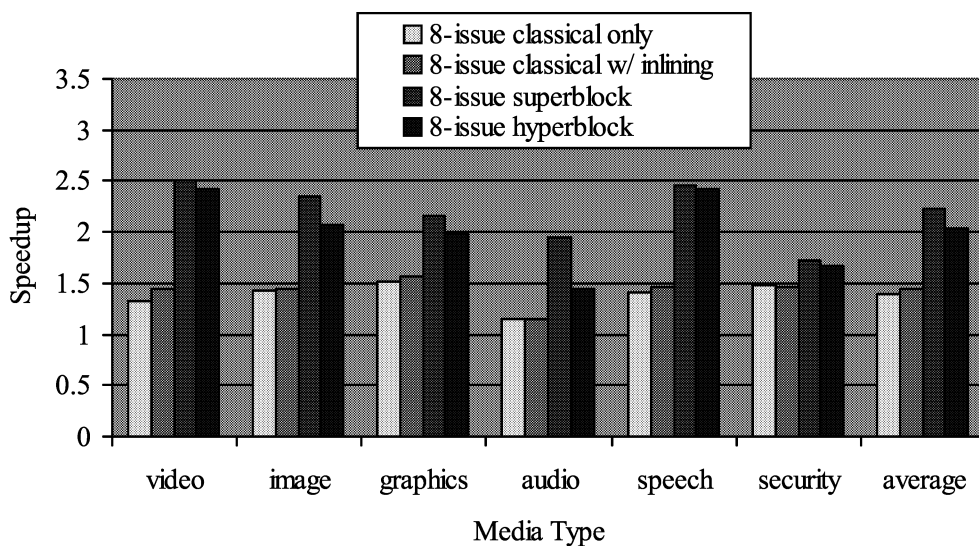
⊞ parallel model: 8-issue

⌘ Explores only parallel scheduling performance

⊞ assumes an ideal processor model

⊞ no performance penalties from branches, cache misses, etc.

ILP results



Multiprocessor architectures for video

⌘ Interested in high-speed video processing.

⊞ 150 frames/sec.

⌘ Want reasonably low-power operation for pervasive applications.

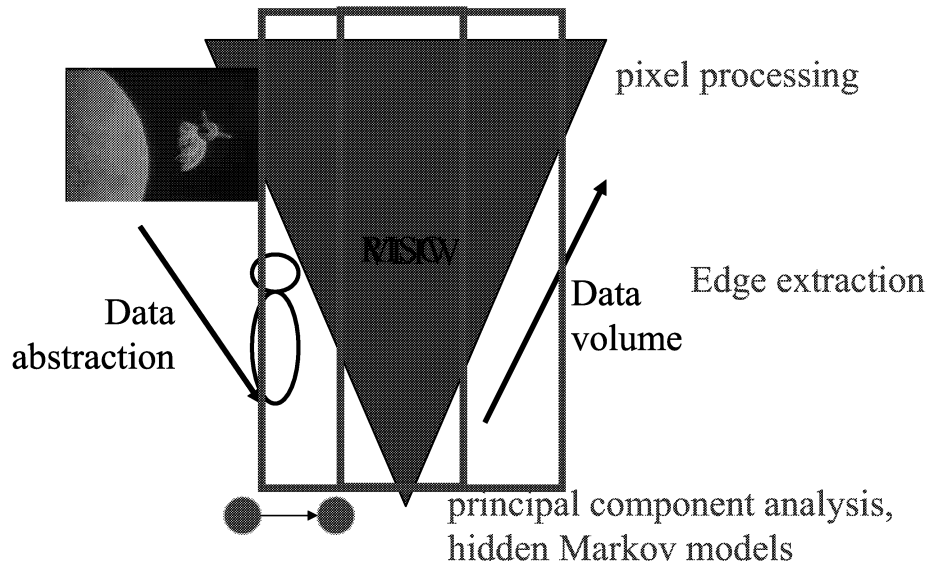
High-speed smart cameras

⌘ High frame rates provide better motion capture.

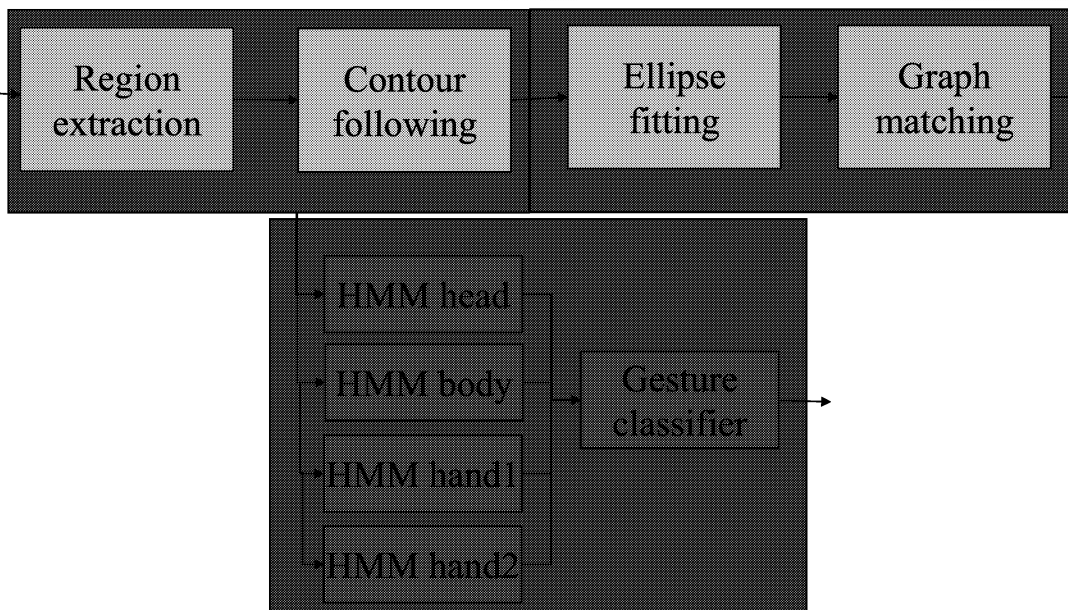
⌘ Frame rate of 150 frames/sec is considered desirable.

⌘ Stanford CMOS camera can digitize at 10,000 frames/sec.

Why heterogeneous architectures make sense



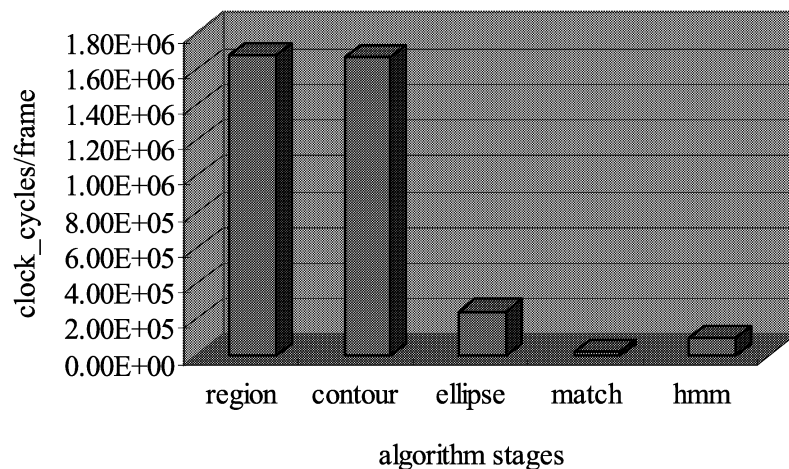
Algorithm flow



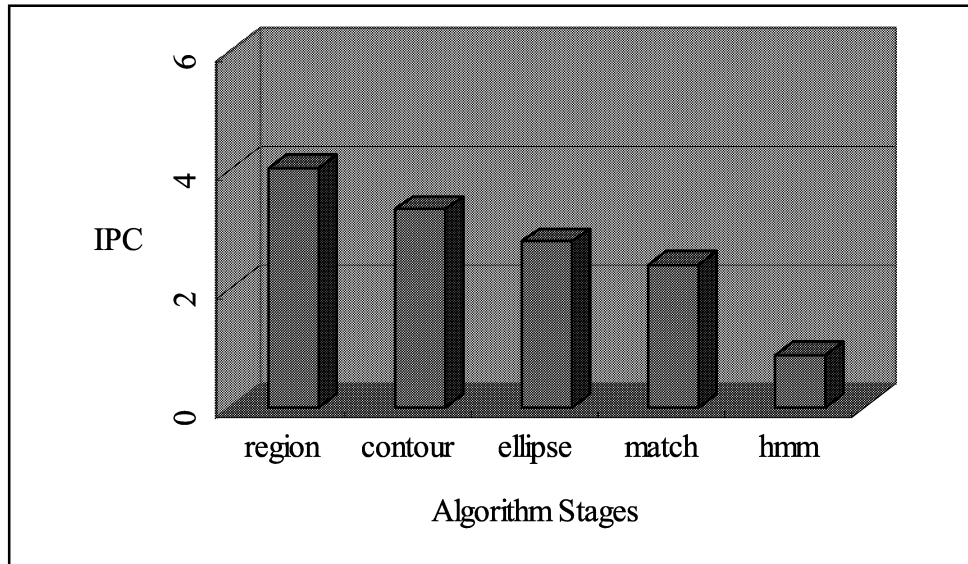
Memory structure

- ⌘ Feed-forward data communication.
- ⌘ Not much global memory required.
- ⌘ Allocating memory depends on data volumes, access patterns, flexibility.

Average processing time by stage



Average IPC by stage

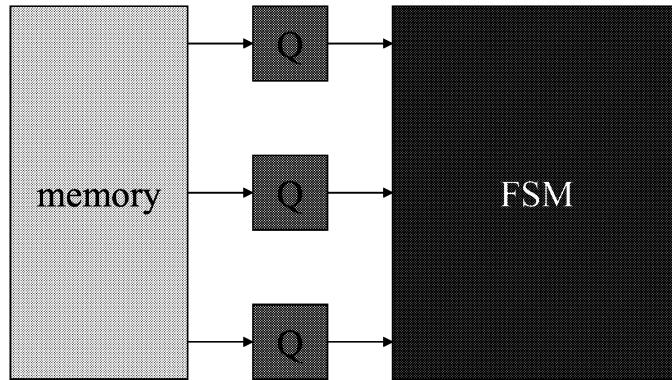


Tiehan's VLIW implementation

- ⌘ Unroll loop to perform multiple comparisons in parallel.
- ⌘ Pack results into bit vector to address results table.
- ⌘ Register file, cache provide for reuse of pixel values.

Contour crawler machine

⌘ Hardware implementation of VLIW code:



Crawler and memory

⌘ Crawler performance depends on memory system.

⌘ Access patterns vary in 2 dimensions:

1	2	3
8	X	4
7	6	5

Memory system design

- ⌘ Want to minimize number of partitions to reduce row/column overhead.
- ⌘ Only memory organization that allows for all parallel accesses is one-word partition.
- ⌘ Assume we fetch one row or column at a time---3 fetches/cycle.

Single contour crawler

- ⌘ Assuming row/column access pattern, crawler is faster than VLIW by a relatively small constant.

Multiple crawlers

- ⌘ Assuming we can patch together contours, we can start multiple crawlers.



- ⌘ Multiple crawler performance is limited by memory.
 - ⊠ Multiple crawlers' memory accesses can conflict.

Full-frame SIMD

- ⌘ Can build a large SIMD array with one processor per pixel.
- ⌘ Area*delay:
 - ⊠ Speed is roughly constant.
 - ⊠ PE is probably about the same size as the crawler.
 - ⊠ Not clear it is worth the silicon.

Heterogeneous system

⌘ Region:

- ⊞ Stream processor with current algorithm.
- ⊞ Stream processor + RISC for others.

⌘ Contour:

- ⊞ Crawler.

⌘ Ellipse:

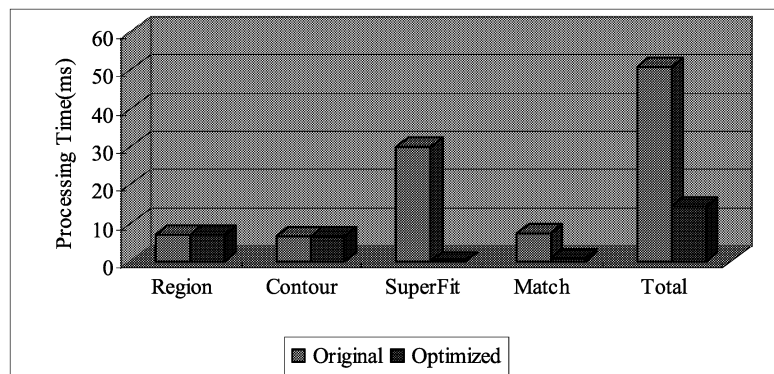
- ⊞ Superscalar/RISC.

⌘ Graph:

- ⊞ RISC.

Stage pipelining

- ⌘ Significant amount of time in non-streaming stages.



Heterogeneous vs. VLIW

⌘ VLIW:

- ⊞ Off-the-shelf IP.
- ⊞ Easy to program.
- ⊞ 10 mm² in 0.13 micron.

⌘ Heterogeneous:

- ⊞ Requires more design of blocks, memory.
- ⊞ Pipelineable for 2.3X speed-up.

Heterogeneous multiprocessor size

stage	PE	area (mm ²)
background	MIPS32 4Km	0.9
contour	custom	0.001
ellipse, graph	MIPS64 5Kf	5
total frame processor		5.901
classification	MIPS64 5Kf	5
number of frame processors		3
grand total		22.703

Research problems in embedded computing

Wayne Wolf
Dept. of Electrical Engr.
Princeton University

Networks-on-chips

- ⌘ Link technology.
- ⌘ Irregular network architectures.
- ⌘ Network synthesis.
- ⌘ QoS.

Platform architectures

- ⌘ Choose a problem, design a platform.
- ⌘ Figure out how to measure the reusability of a platform.

Memory systems

- ⌘ Controllable caches.
- ⌘ Distributed memory architectures.
- ⌘ Software methods for exploiting memory.

Low power design

- ⌘ System-level power strategies.
- ⌘ Effects of leakage.

Processor architecture

- ⌘ Choosing specialized instruction sets.
- ⌘ Multiple data widths.
- ⌘ Adaptations for networks-on-chips.

Software

- ⌘ Exploiting specialized instruction sets.
- ⌘ Exploiting configurable caches.

Networks of SoCs

- ⌘ Integrating network functions into platforms.
- ⌘ On-the-fly adaptation to network conditions.

Kris Kuchcinski

Lund University



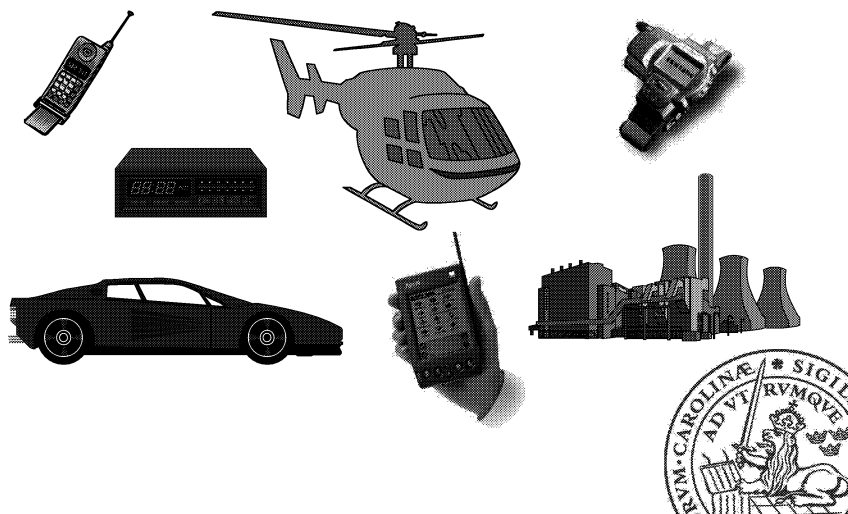
LUND INSTITUTE
OF TECHNOLOGY
Lund University

Design of Embedded Systems

Kris Kuchcinski
Dept. of Computer Science
Lund University
Sweden

<http://www.cs.lth/~kris>

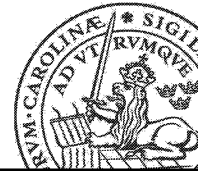
Examples of Embedded Systems



Examples of Embedded Systems (cont'd)

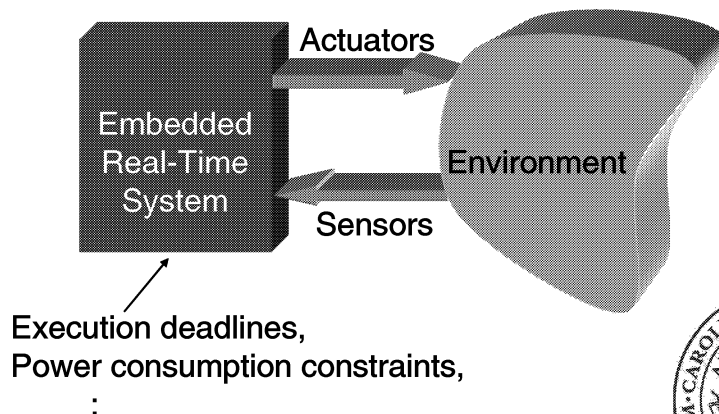
Anti-lock brakes	Modems
Auto-focus cameras	MPEG decoders
Automatic teller machines	Network cards
Automatic toll systems	Network switches/routers
Automatic transmission	On-board navigation
Avionic systems	Pagers
Battery chargers	Photocopiers
Camcorders	Point-of-sale systems
Cell phones	Portable video games
Cell-phone base stations	Printers
Cordless phones	Satellite phones
Cruise control	Scanners
Curbside check-in systems	Smart ovens/dishwashers
Digital cameras	Speech recognizers
Disk drives	Stereo systems
Electronic card readers	Teleconferencing systems
Electronic instruments	Televisions
Electronic toys/games	Temperature controllers
Factory control	Theft tracking systems
Fax machines	TV set-top boxes
Fingerprint identifiers	VCR's, DVD players
Home security systems	Video game consoles
Life-support systems	Video phones
Medical testing systems	Washers and dryers

Source: *Embedded Systems Design: A Unified Hardware/Software Introduction*, (c) 2000 Vahid/Givargis



Embedded Systems

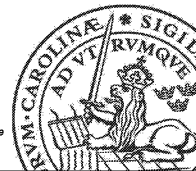
"A device that includes a programmable computer but is not itself a general-purpose computer."



Embedded Systems (cont'd)

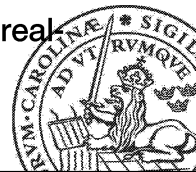
- Computing systems embedded within electronic devices
- Hard to define. Nearly any computing system other than a desktop computer
- Billions of units produced yearly, versus millions of desktop units
- Perhaps 50 per household and per automobile

Source: Embedded Systems Design: A Unified Hardware/Software Introduction, (c) 2000 Vahid/Givargis



Embedded Systems (cont'd)

- Non User-Programmable.
- Based on programmable components (e.g., Micro-controllers, DSP's...) but often contain application specific hardware (IC's, ASIC's).
- Reactive Real-Time Systems:
 - React to external environment,
 - Maintain permanent interaction,
 - Ideally never terminate,
 - Are subject to external timing constraints (real time).

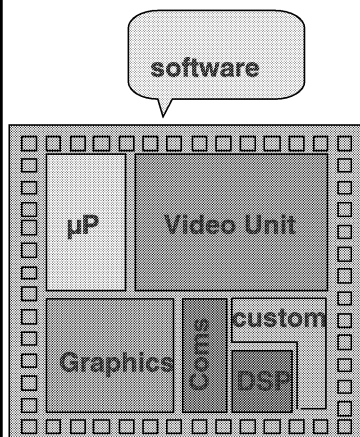


Characteristics of Embedded Systems

- Sophisticated functionality.
- Real-time operation.
- Low manufacturing cost.
- Low power.
- Designed to tight deadlines by small teams.
- “Resource conscious” vs. “Unlimited resources” programming



SoC Embedded System

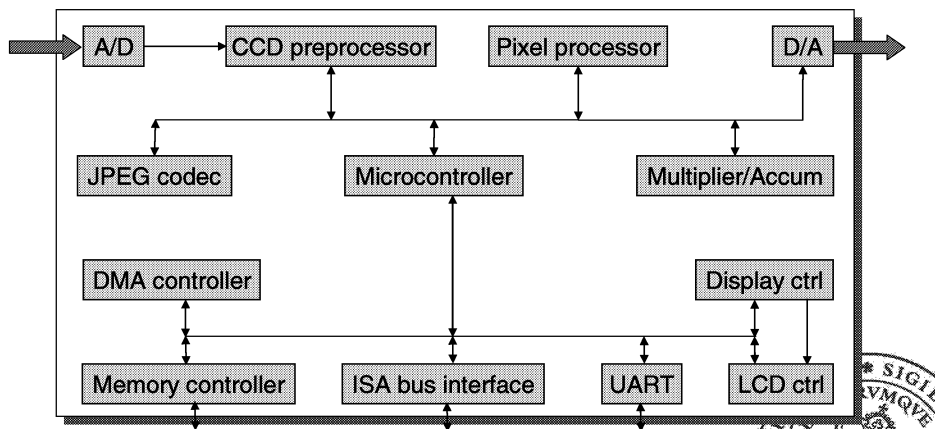


- Assembly of “prefabricated components” often purchased from external vendors (“IP”)
 - “black box” hierarchy
- Design & Verification at the System level
 - rather than the logic level
 - Interface and communication
- Great Importance of Software

Source: Alberto Sangiovanni-Vincentelli, 35th DAC

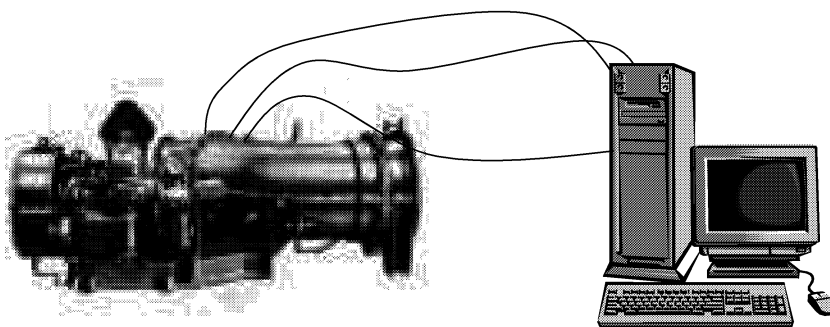


A Digital Camera Example



Source: F. Vahid and T. Givargis, *Embedded System Design: A Unified Hardware Software Approach*

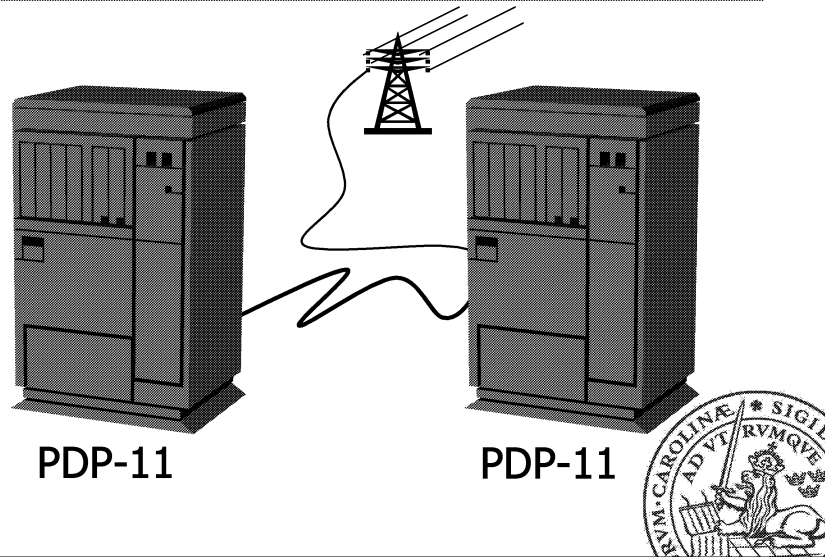
Real-time gas turbine testing system



MI-2 helicopter engine

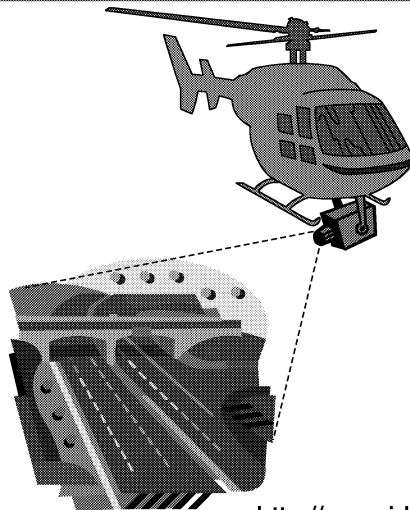
"Minicomputer"
8kB RAM
cassette tape

TELEX-I and TELEX-II systems



WITAS

WITAS project



- Autonomous system.
- Real-time system.
- Image processing.
- Mission planning.
- Incorporation of GIS systems.
- Interface with ground operator.
- ...

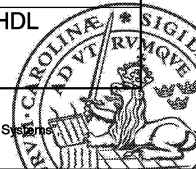
<http://www.ida.liu.se/ext/witas>



Typical Hardware Components of DSP System

Component class	Implements	Compiler	Specification
DSP processor	Low data-rate DSP Slow control loops Appl. Spec. alg.	(Retargetable) code generator High level synth.	Assembly C DFL
Microcontroller	User interface Slow control loops	C compiler	C
Hardware accelerator	High data-rate DSP	High level synth. RT level synth.	C, DFL VHDL
Communication blocks and memory	Internal & external communication Storage & buffering	Memory mgmt. (A)synchronous interface synth.	Data-sheets STG
Others	Usually FSM's - clock generators - DMA blocks	RT level synth. Asynchronous synth.	VHDL

Source: H. de Man, et. al. "Co-design of DSP Systems"
Hardware/Software Co-design, Kluwer 1995.

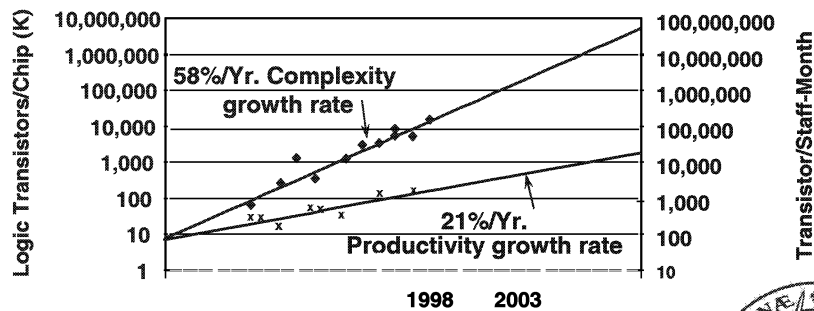


Importance of Embedded System Design Methodologies

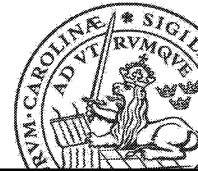
- Hardware complexity.
- Heterogeneous systems containing hardware (both digital and analog) and software.
- Heterogeneous components (CPU's, DSP's, ASIC's, buses, point-to-point links, etc.).
- Heterogeneous requirements — performance, cost, power consumption, etc.
- System-on-chip.
- Shorter design cycles required by time-to-market constraints.
- ...



Design Complexity and Designer Productivity Gap



Source: Bryan Preas, Xerox PARC, 35th DAC



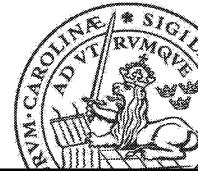
Software vs. Hardware Design short summary

- Software
 - flexibility,
 - reconfigurability, easy update, etc.,
 - complex functionality,
 - cost,
 - ...
- Hardware
 - speed,
 - power consumption,
 - cost in large volumes,
 - ...



Design of Embedded Systems

- Need to be done using high-level specification, programming and hardware description languages — not assembly languages and gate/transistor level design.
- Requires efficient design space exploration and synthesis/compilation tools.
- Different design requirements has to be taken into account, e.g., cost, performance, testability, quality of service, power consumption.
- Multi-language design framework.



Importance of High-Level Design Methods

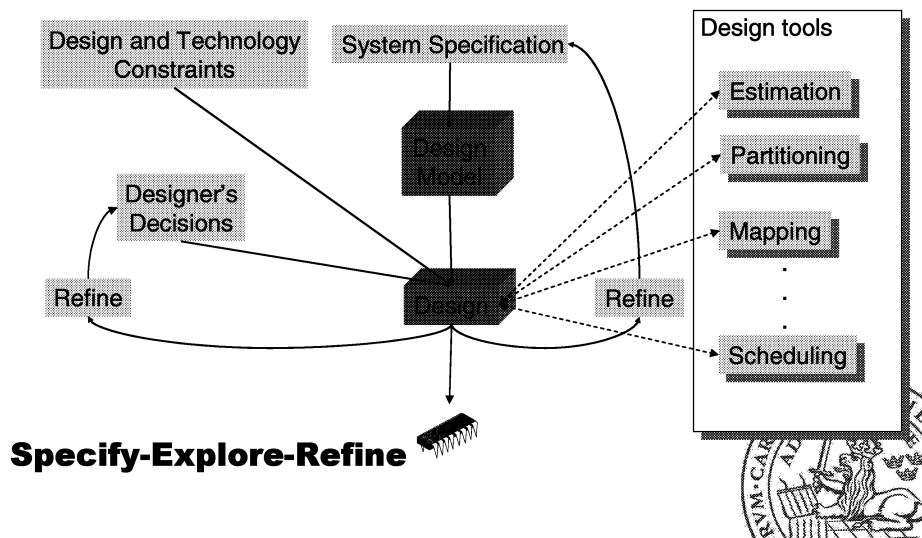
System Verification Processing Speeds

System Implementation	Processing time (s/frame)
Behavioral model	1 200 (20 min/frame)
RTL model	144 000 (1.6 days/frame)
Gate model	228 000 (2.6 days/frame)
Gate model on hardware accelerator	1 200
Rapid Prototype	0.5
Target Hardware	0.05

Source: Paul Clemente, Ron Crevier, Peter Runstadler "RTL and Behavioral Synthesis A Case Study", VHDL Times, vol. 5, no. 1.



General Design Flow



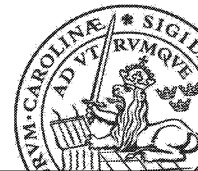
Specification and Programming

- Specification languages, such as UML, SDL.
- Programming languages, such as C, C++, Java, Esterel, assembly languages.
- Hardware description languages, such as VHDL, Verilog, SystemC.
- **Example:** combining SystemC and C++ gives unified simulation environment for hardware and software.



Hardware Description Languages

- Cover several levels of design abstraction as well as behavioral and structural description domain.
- Contain typical features of programming languages, such as data types and program statements.
- Special features:
 - time concept,
 - structure description,
 - parallelism.
- VHDL (IEEE standard), Verilog, SystemC.



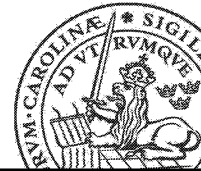
Design Representations (Computational Model)

- Used to represent/model digital systems under design.
- Generated by a compiler from system specification or coded directly in the model.
- Represent the semantics, structure and timing of the system.
- Usually based on some kind of annotated graph representation.
- Used internally by design automation systems or by the modeler/designer.

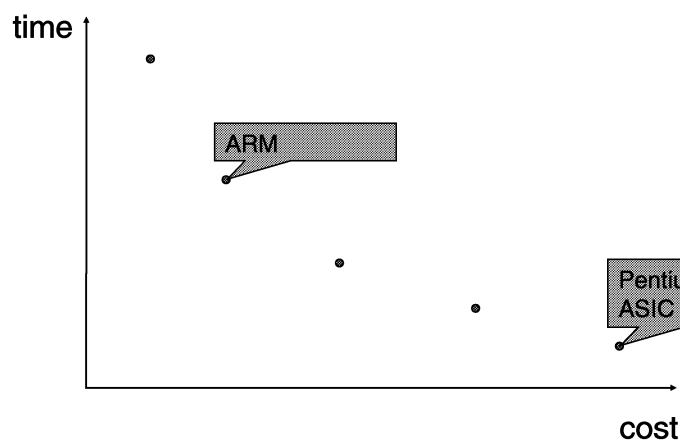


Design — Synthesis

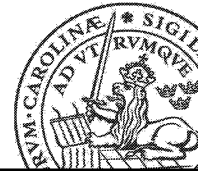
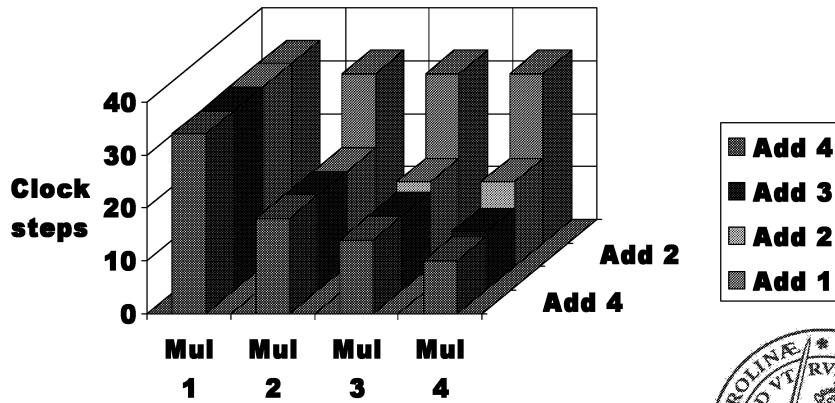
- *Software translation* into target code for a processor (real-time operating system might be used).
- *Hardware synthesis* — translation of a behavioral representation of a design into a structural one.
- *Communication synthesis* — generates hardware and software which interconnects system components.



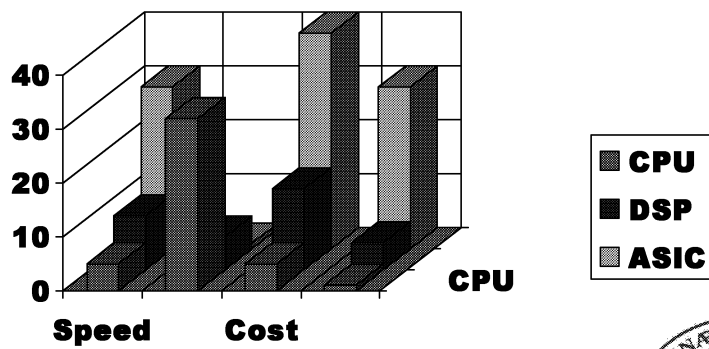
Pareto points



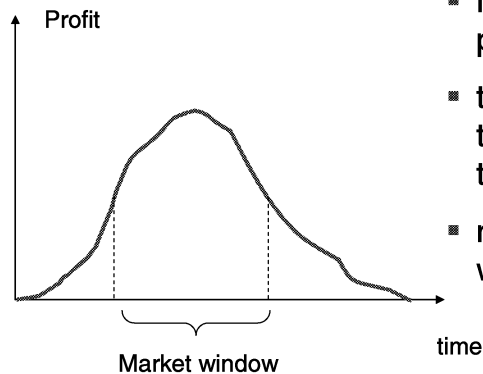
Discrete Cosine Transform Partial Design Space



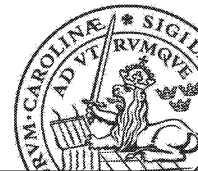
Design Space Exploration



Time-to-market constraint



- Need time for new product development,
- the biggest profit is in the market window time,
- missing the market window can be costly.



Summary

- Embedded systems are important class of electronic systems which can be found everywhere,
- Combine hardware and software solutions,
- Cover several engineering and research areas:
 - microelectronics,
 - real-time systems,
 - software development,
 - etc.
- Need careful design which optimizes different design parameters.





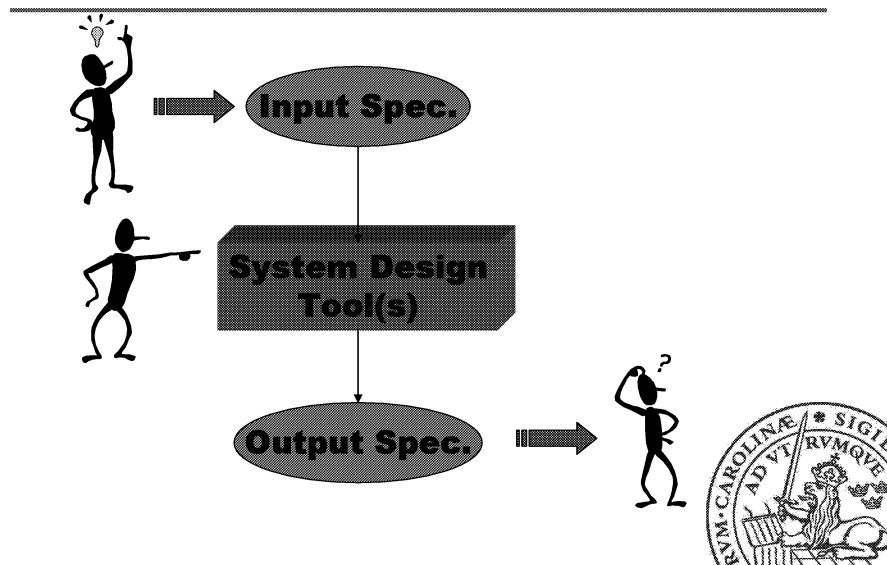
LUND INSTITUTE
OF TECHNOLOGY
Lund University

Methodologies for Embedded System design

Kris Kuchcinski
Dept. of Computer Science
Lund University
Sweden

<http://www.cs.lth/~kris>

General Methodology



Input to System Design

- Executable specification (functional requirements):
 - usually provided as interacting processes/tasks,
 - very often multi-language specifications,
 - can be simulated and verified,
 - can be used to perform analysis, e.g, estimation.
- Specification languages: C, C++, VHDL, Verilog, SystemC, Esterel, SDL, etc.
- Set of (non-functional) design requirements (cost, speed, I/O rate, power consumption, etc.).

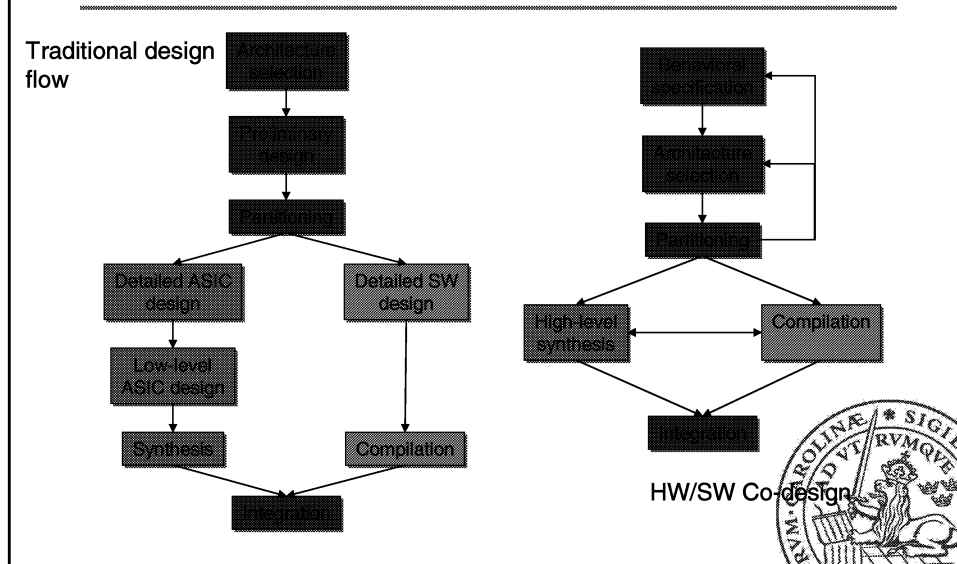


Output from System Design

- A set of system modules assigned to system components (CPU's, DSP's, ASIC's, etc.).
- Communication modules.
- Each module can be further synthesized to hardware using *high-level synthesis* or *compiled to software*.



HW/SW Co-design



Basic Characteristics of the Methodology

- *Behavioral specification* is given for the complete heterogeneous system, regardless of how different parts will be later implemented.
- *Analysis techniques* are provided; specially different estimation techniques.
- *Synthesis tools* are used to automatically explore a design space.
 - high-level synthesis, RTL synthesis,
 - compilers, cross-compilers,
 - interface generators,
 - etc.



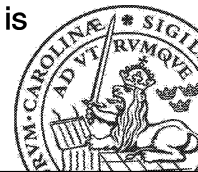
Estimation of Design Parameters

- Estimation of parameters such as size, cost, power consumption.
- Does not need to be very precise but has to be “consistent” — follows real design parameters.
- Usually 15%-20% inaccurate.
- Trade-off between accuracy and estimation time.

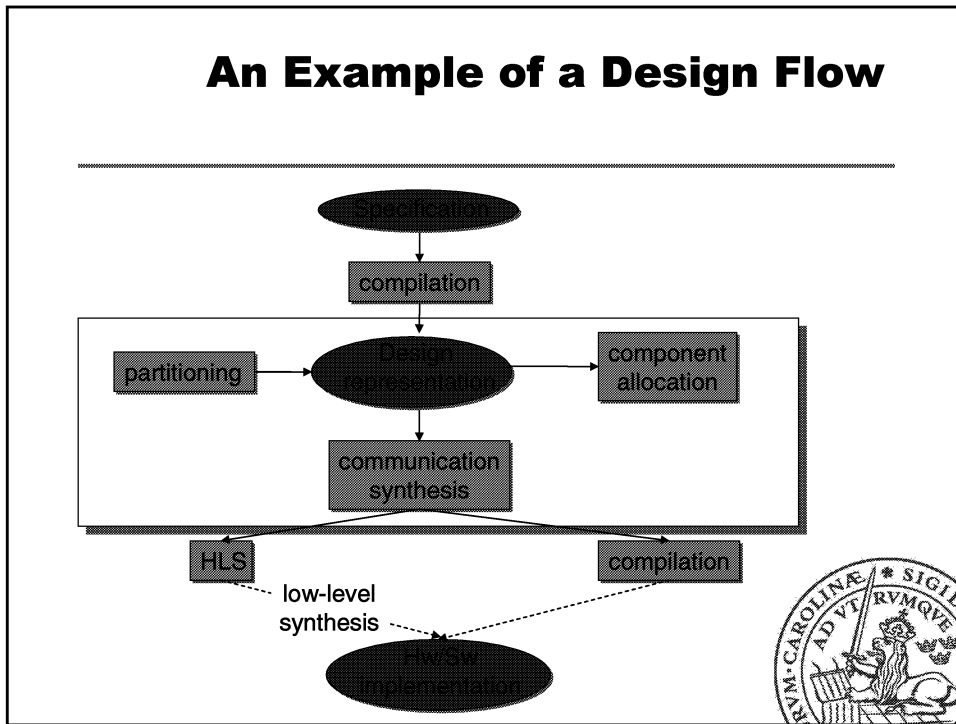


Improvements of the Design Process

- High-level specification is made before architecture selection and implementation *decisions can be made more accurate* (better exploration of architectures).
- A uniform description of HW and SW makes it possible to *move parts of the systems between HW and SW*.
- HW and SW development is moved closer and the *integration cost is reduced*.
- An *early evaluation of system characteristics* is possible.



An Example of a Design Flow



Specification Example

“Extended” VHDL

```

port(IP1,IP2:in INTEGER; OP1,OP2:out INTEGER);
...
signal S1,S2,S3,S4,S5,S6:INTEGER;
P1 : process
...
  receive(IP1);
...
  send(S1,...);
...
  send(S3,...);
...
  receive(S6);
...
end process P1;

P2 : process
...
  receive(IP2);
...
  receive(S1);
...
  send(OP2,...);
...
end process P2;

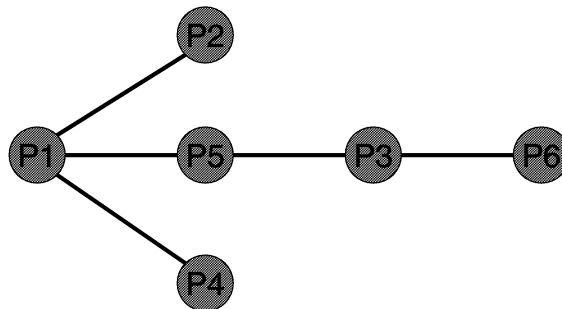
P3 : process
...
  receive(S4);
...
  send(S2,...);
...
end process P3;

P4 : process
...
  receive(S3);
...
  send(S5,...,S6,...);
...
end process P4;

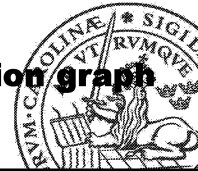
P5 : process
...
  receive(S1,S5);
...
  send(S4,...);
...
end process P5;

P6 : process
...
  receive(S2);
...
  send(OP1,...);
...
end process P6;
  
```

Representation Example



Process communication graph



Allocation of System Components

- Decides about the kind and number of components for implementation of the system
 - processing elements: μ procesosrs, micro-controllers, DSP's, ASIP's, ASIC's, FPGA's, etc.
 - storage elements: memories, register files, registers, etc.
 - communication devices: buses, point-to-point links, networks, etc.
 - specialized I/O devices: A/D, D/A, frame grabbers, etc.

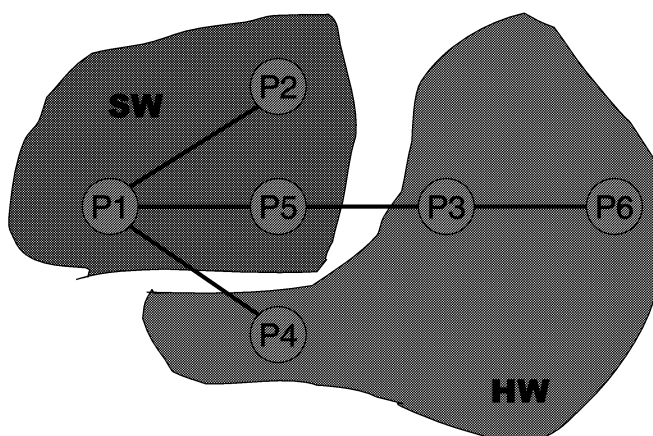


Partitioning

- *Functional* partitioning vs. *structural* partitioning.
- Abstraction level.
- Partitioning granularity (*fine* or *course*):
 - modules,
 - processes and procedures,
 - instructions.
- Partitioning objective:
 - performance,
 - minimal communication,
 - low power,
 - combination of several criteria.



Partitioning Example

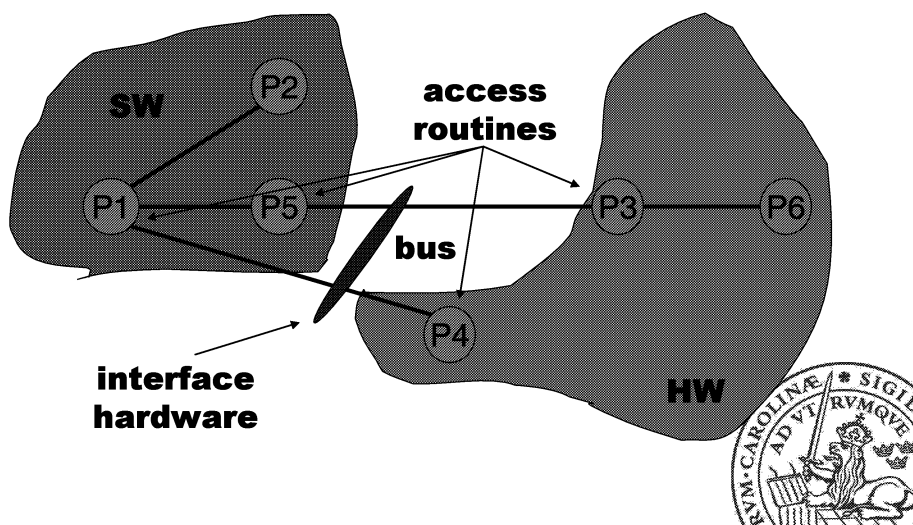


Communication Synthesis

- Creation of abstract communication channels by communication clustering.
- Communication refinement
 - selection of communication lines width,
 - protocol selection,
 - etc.
- Interface generation:
 - device drivers,
 - communication hardware,
 - etc.

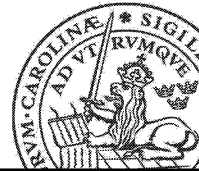


Communication Synthesis Example



Design Decisions

- Different types of design decisions
 - selection of components, partitioning, assignments, scheduling, etc.
 - decisions regarding runtime system done off-line or are postponed to runtime (e.g., static vs. runtime scheduling)
- Design decisions are mutually dependent
- Huge design space



Design Automation

- Uses internal representations which are usually based on graphs.
- Graph algorithms (shortest path, Hamiltonian circuit, topological sort, depth-first-search, breadth-first-search, SAT, etc.).
- Optimization methods — (M)ILP, CLP, heuristics, etc.
- *Tractable* and *intractable* problems.
- *Decidable* and *undecidable* problems.
- *Decision problems* and *combinatorial optimization problems*.



Design Automation Consequences

- Most of the problems which need to be solved in design automation are NP-complete or NP-hard.
- Usually only small problems can be solved exactly.
- Need for algorithms which do not guarantee optimal solutions but “good enough” solutions
 - *approximation algorithms* — guarantee a solution with a cost that is within some margin of the optimum,
 - *heuristics* — algorithms that are constructed based on “rules-of-thumb”; nothing can be said in advance about the quality of the result.





LUND INSTITUTE
OF TECHNOLOGY
Lund University

Constraint Programming Approach

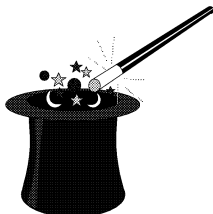
Kris Kuchcinski
Dept. of Computer Science
Lund University
Sweden

<http://www.cs.lth/~kris>

Quotations

"Constraint programming represents one of the closest approaches computer science has yet made to the Holy Grail of programming: the user states the problem, the computer solves it."

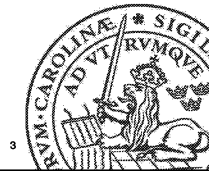
Eugene C. Freuder
CONSTRAINTS, April 1997



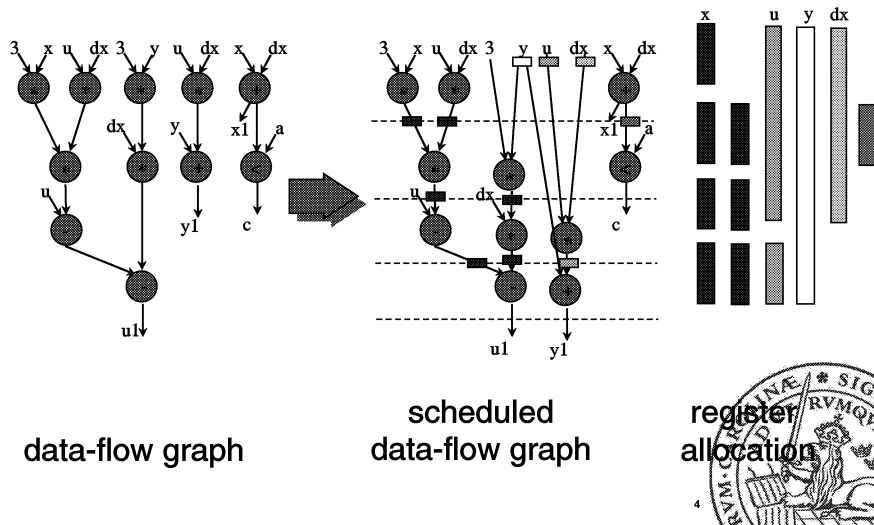
Introduction and Motivation

Synthesis of the following code
(inner loop of differential equation integrator)

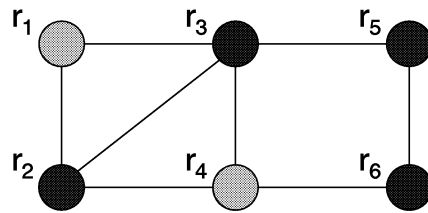
```
while c do
  begin
    x1 := x + dx;
    u1 := u - (3*x*u*dx);
    y1 := y + u*dx;
    c := x < a;
    x := x1; y := y1; u := u1;
  end;
```



Introduction and Motivation

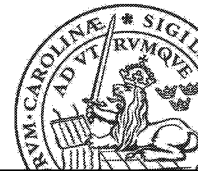


Register Allocation as Graph Coloring



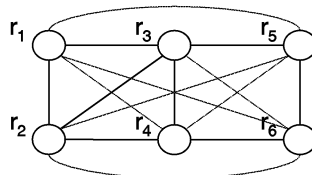
Constraints:

$[r_1, r_2, r_3, r_4, r_5, r_6] :: 0..2,$
 $r_1 \neq r_2, r_1 \neq r_3, r_2 \neq r_3,$
 $r_2 \neq r_4, r_3 \neq r_4, r_4 \neq r_6,$
 $r_5 \neq r_6.$



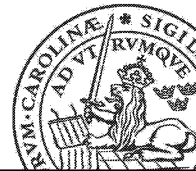
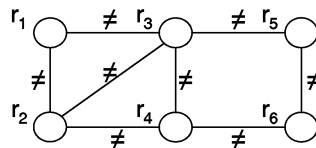
Register Allocation as Clique Finding

- for all r_i, r_j which are not connected by an edge:
 $r_i \neq 1 \vee r_j \neq 1$
- The maximal clique can be found by maximizing the following cost function:
 $cost = \sum_i r_i$



Constraint Consistency

- All constraints are stored in the *constraint store*
- *Consistency* methods are applied to find inconsistent values and prune variables' domains
- Different types of consistency methods:
 - Node consistency
 - Arc consistency
 - Path consistency
 - ...



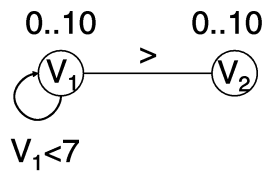
Consistency Properties

- Node consistency
 - A network is node consistent if in each node domain each value is consistent with unary constraint (e.g., $X > 7$)
- Arc consistency
 - A network is arc consistent if for each arc connecting variables V_i and V_j for each value in the domain of V_i there exist a value in the domain of V_j consistent with binary constraint (e.g., $X > Y$)

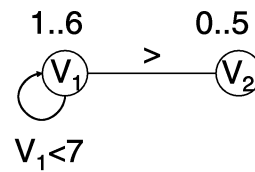


Node and Arc Consistency

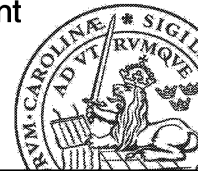
- Example



Not node consistent
Not arc consistent

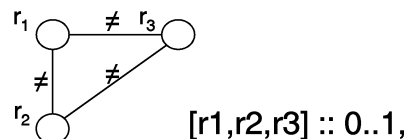


node consistent
arc consistent



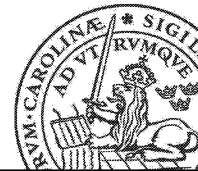
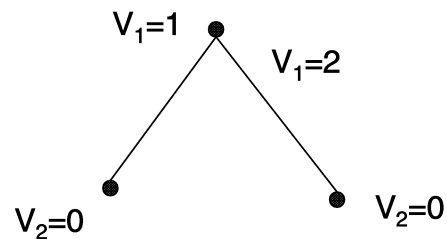
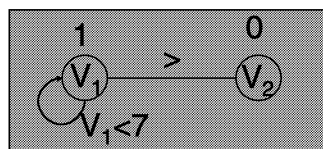
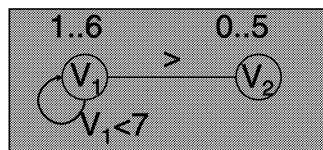
Need for search

- Node, arc and path consistency are in general not complete (complete for some problems with particular structures)
- Complete algorithm: N -consistency for N variable problems \rightarrow exponential complexity
- Example:



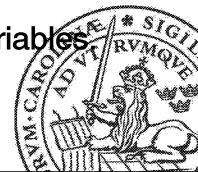
Search

- Solver is not complete and search for a solution is needed



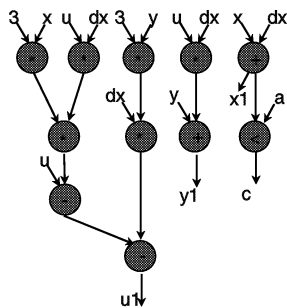
Constraint properties

- may specify partial information — need not uniquely specify the values of its variables,
- non-directional — typically one can infer a constraint on each present variable,
- declarative — specify relationship, not a procedure to enforce this relationship,
- additive — order of imposing constraints does not matter,
- rarely independent — typically they share variables



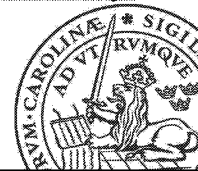
More realistic example Scheduling

Scheduling of the data-flow graph



Constraints:

- for all op_i and op_j such that op_i before op_j
 $T_i + D_i \leq T_j$
- for all op_i and op_j that can use the same resource
 $T_i + D_i \leq T_j \vee T_j + D_j \leq T_i \vee R_i \neq R_j$



Problems

- Constraint propagation for
 $T_i + D_i \leq T_j \vee T_j + D_j \leq T_i \vee R_i \neq R_j$ is weak
- Not all solvers support disjunctive constraints.
 - Other solution (reified constraints):

$$T_i + D_i \leq T_j \Leftrightarrow B1,$$

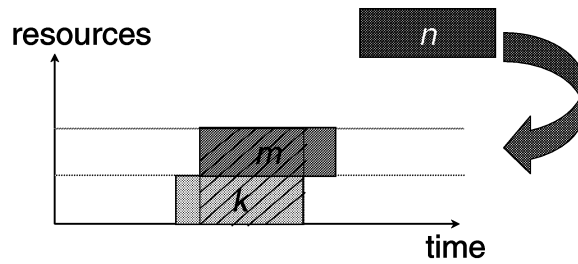
$$T_j + D_j \leq T_i \Leftrightarrow B2,$$

$$R_i \neq R_j \Leftrightarrow B3,$$

$$B1 + B2 + B3 \geq 1.$$

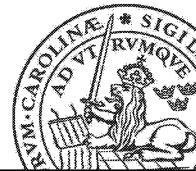


Propagation problems



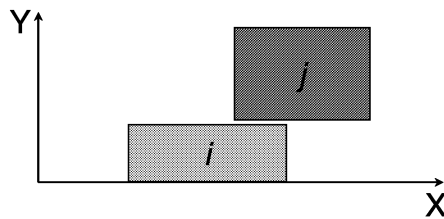
$$T_k + D_k \leq T_n \vee T_n + D_n \leq T_k \vee R_k \neq R_n$$

$$T_m + D_m \leq T_n \vee T_m + D_m \leq T_n \vee R_m \neq R_n$$



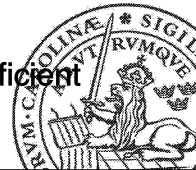
Global constraints

- Non-overlapping rectangles



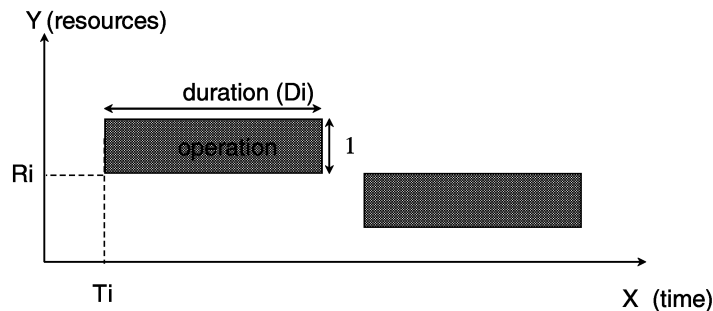
$$\text{diff2}([[X_i, Y_i, DX_i, DY_i], [X_j, Y_j, DX_j, DY_j]])$$

- All knowledge in one "place" – makes it possible to define good consistency methods (OR, mathematics, geometry, etc.)
- Specific algorithms for consistency – more efficient

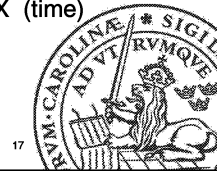


Global Constraints – Scheduling

- diff2 constraint



diff2([[T1, R1, D1, 1], [T2, R2, D2, 1]], ...)

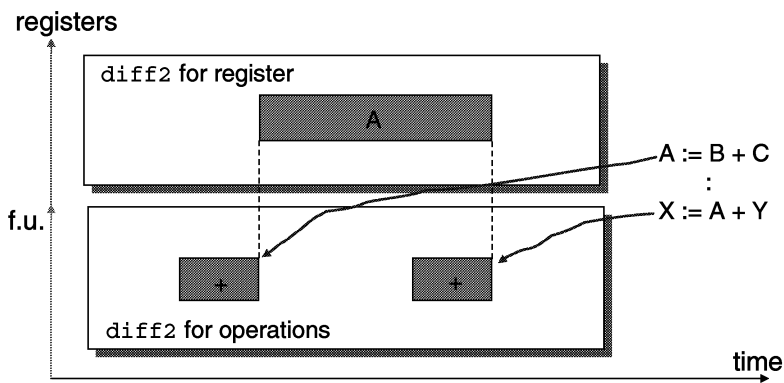


Scheduling Example Constraints

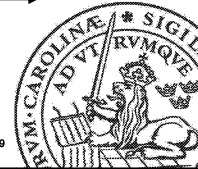
$T1 + 2 \leq T6$, $T2 + 2 \leq T6$,
 $T3 + 2 \leq T7$, $T4 + 2 \leq T8$,
 $T5 + 1 \leq T9$, $T6 + 2 \leq T10$,
 $T7 + 2 \leq T11$, $T10 + 1 \leq T11$,
 diff2([[T1,R1,2,1], [T2,R2,2,1], [T3,R3,2,1],
 [T4,R4,2,1], [T6,R6,2,1], [T7,R7,2,1],
 [T5,R5,1,1], [T8,R8,1,1], [T9,R9,1,1],
 [T10,R10,1,1], [T11,R11,1,1]]).



Registers



- can be done together with or after functional units allocation/binding and scheduling,

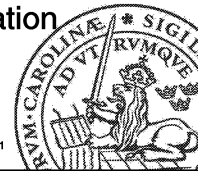
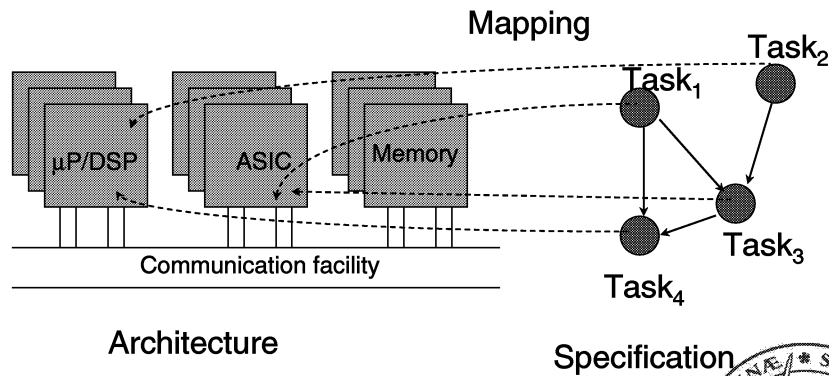


Other Synthesis Problems Defined with Constraints

- High-level synthesis:
 - Chaining,
 - Conditional execution,
 - Pipelined components,
 - Algorithmic pipelining,
 - Switching activity reduction (power consumption)
 - ...
- System design
 - different aspects of design space exploration
 - scheduling
 - component assignment
 - memory allocation/data assignment
 - power/energy consumption
 - ...



Design Space Exploration



21

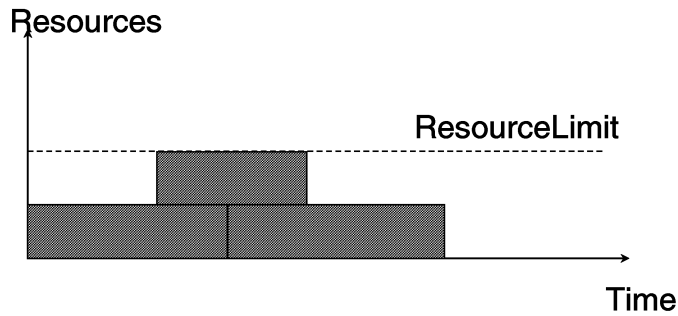
Additional Constraints element

- Element constraints
 - $\text{element}(N, [X_1, X_2, \dots, X_n], \text{Value})$
 - propagation from N to Value
 $N=i \rightarrow \text{Value} = X_i$
 - propagation from Value to N
 $\text{Value} = x \rightarrow N=i \text{ and } X_i = x \dots$
- Examples- $\text{element}(N, [2, 3, 4, 4], V)$
 - $N :: 1..2, V :: \{2, 3\}$
 - $V = 4, N :: 3..4$
- Used to define discrete cost functions and different relations



Additional constraints cumulative

Cumulative constraint

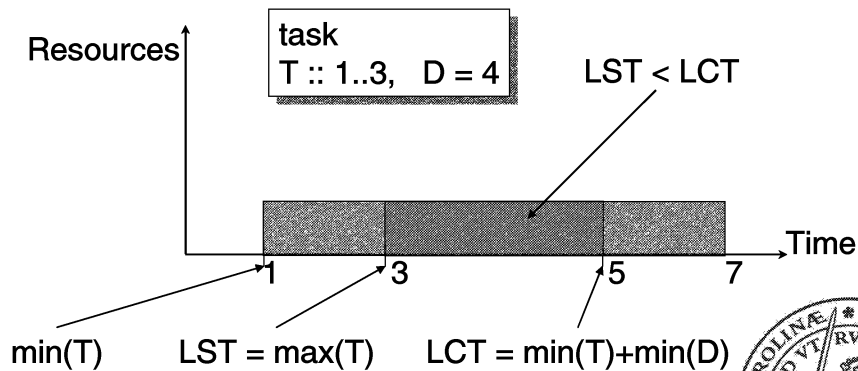


`cumulative([Tk, Tn, Tm], [Dk, Dn, Dm], [1, 1, 1], ResourceLimit)`

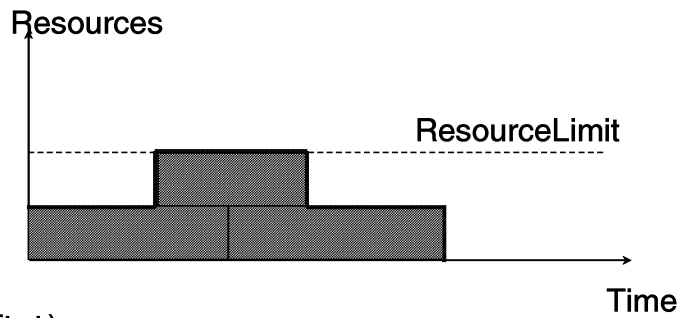


Cumulative propagations

- Execution interval which will always be occupied by a task.



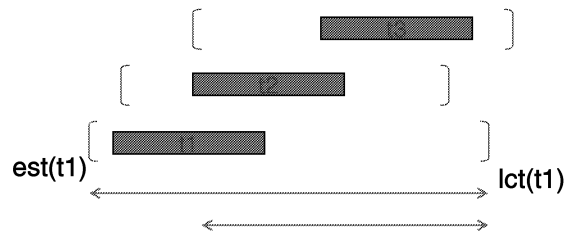
Cumulative propagations — profile based



for each $[t_i, t_j)$
 for each task_n whose exec. interval overlaps with $[t_i, t_j)$
 if $(ResourceLimit - resource_usage < task_n(resources))$
 T_n in $\{ complement(t_i - \min(D_n) + 1 .. t_j - 1) \}$
 check $D_n \dots Res_n \dots$



Cumulative propagations — edge finding



t3 cannot be between t1 and t2 iff
 $lct(t1) - est(t1) < D1 + D2 + D3$

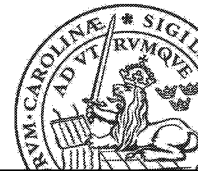
t3 cannot be before t1 and t2 iff
 $lct(t1) - est(t3) < D1 + D2 + D3$

t3 must be last !!!

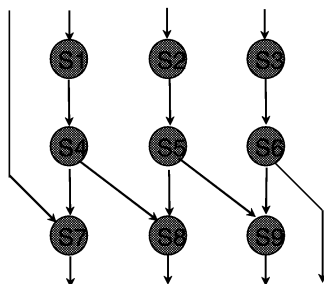


Edge Finding Algorithm

- Martin-Shmoys algorithm with $O(n^2)$ complexity.
- Up phase
 - for each unique lct we create a set
 $S = \{t \mid LCT(t) \leq lct\}$ and make checking whether a task can be the first or before
- Down phase
 - similar but using est and checking whether a task can be the last one or after all tasks.



System Synthesis Example



▪ original MILP formulation- 47 timing variables, 225 binary (bus 153) and 1081 constraints (bus 416)

▪ commercial linear programming package used to solve the problem (XLP, developed by XMP Software, Inc.)

Processor	Cost	Execution time								
		S1	S2	S3	S4	S5	S6	S7	S8	S9
P1	4	2	2	1	1	1	1	3	-	1
P2	5	3	1	1	3	1	2	1	2	1
P3	2	1	1	2	-	3	1	4	1	4



Modeling of cost and execution time

- Execution time
 - element(P1, [2, 3, 1], D1)
 - ...
 - element(P9, [1, 1, 4], D4)
- Cost
 - (P1=1 ∨ P2=1 ∨ ... ∨ P9=1) ⇔ C1,
 - ...
 - (P1=6 ∨ P2=6 ∨ ... ∨ P9=6) ⇔ C6,
 - Cost = 4*C1 + 4*C2 + 5*C3 + 5*C4 + 2*C5 + 2*C6



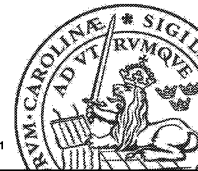
System synthesis results

Design	Cost	Performance (time units)	Performance optimization		Cost optimization		
			MILP (s)	CLP (s)	B&B Nodes	CLP (s)	B&B Nodes
Bus	10	6	6438.00	0.43	84	0.55	92
	6	7	5371.80	0.53	114	0.68	144
	5	15	3691.20	0.43	68	0.70	103
point-to-point links	15	5	3732.00	0.43	20	1.67	125
	12	6	26710.20	1.42	98	2.18	169
	8	7	32320.20	1.00	58	2.59	198
	7	8	4510.80	1.64	75	2.02	112
	5	15	38501.20	1.50	32	1.48	77



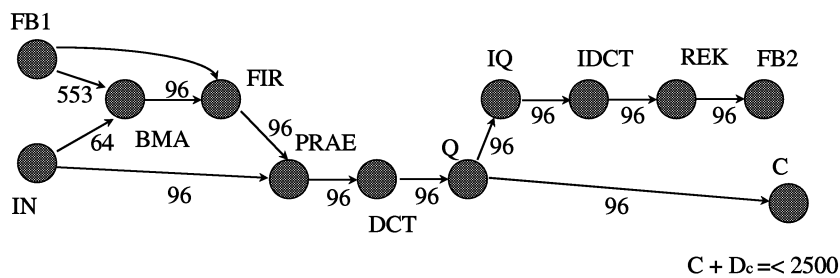
System synthesis results with local memory

Design	Cost	Performance (time units)	Performance optimization			Cost optimization	
			MILP (s)	CLP (s)	B&B Nodes	CLP (s)	B&B Nodes
Bus	28	6	6592.20	0.71	76	2.58	252
	23	7	5371.80	1.07	193	1.94	266
	22	8	123252.60	0.95	124	14.85	856
	21	10	316860.60	114.92	4 534	119.55	8 799
	18	11	236724.00	88.23	7 015	2.37	477
	17	12	138004.20	0.93	268	10.39	3 076
	14	15	3581.40	0.54	22	9.89	3 076
point-to-point links	38	5	-	0.56	24	2.08	107
	30	6	-	0.99	59	3.75	155
	25	7	-	1.60	79	5.58	314
	23	8	-	1.82	57	3.21	184
	22	10	-	4.50	84	59.25	855
	19	11	-	27.34	794	101.03	2 851
	18	12	-	97.72	2 686	8.66	1 047
	14	15	-	1.18	14	4.95	328

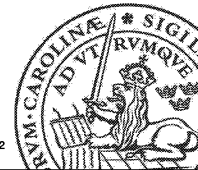


31

An Example



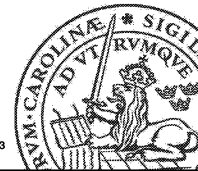
Video Coding Algorithm H.261



32

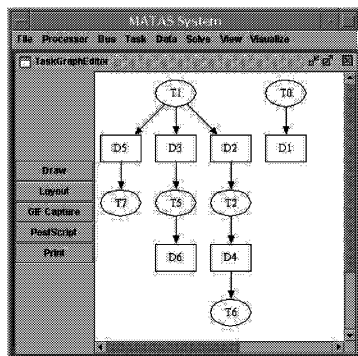
Task Mappings to Processors

Task	Uni- versal	BMA array	PAR1	DCT array	FIR array	BMA pipe	FIR seq	FIR pipe	DCT seq	DCT pipe
IN	-	-	-	-	-	-	-	-	-	-
FB1	-	-	-	-	-	-	-	-	-	-
BMA	7234	484	-	-	-	3617	-	-	-	-
FIR	7234	-	-	-	510	-	3461	1170	-	-
PRAE	1280	-	128	-	-	-	-	-	-	-
DCT	12312	-	-	132	-	-	-	-	6156	474
Q	-	-	-	-	-	-	-	-	-	-
IQ	-	-	-	-	-	-	-	-	-	-
IDCT	12312	-	-	132	-	-	-	-	6156	474
REK	1536	-	256	-	-	-	-	-	-	-
C	132	-	-	-	-	-	-	-	-	-
FB2	-	-	-	-	-	-	-	-	-	-

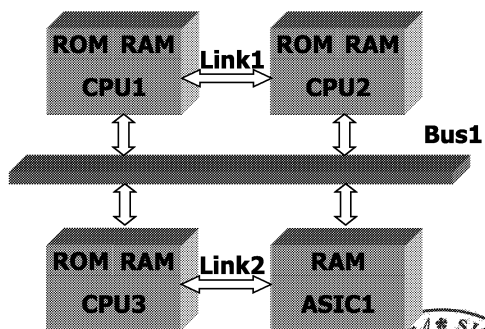


38

Scheduling with Memory Constraints



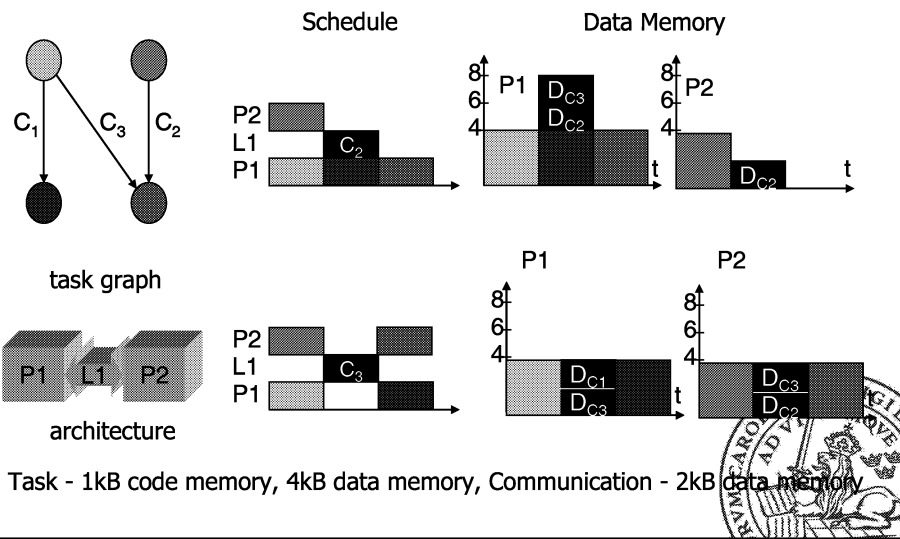
annotated task graph



target architecture



Memory importance



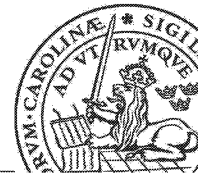
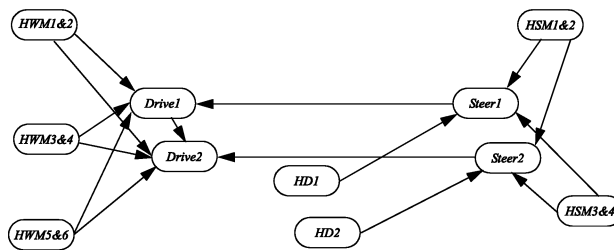
Experimental results H.261 example

EXP. #	ALGORITHM	PIPELINE DEGREE	DEADLINE	AVERAGE EXECUTION TIME	Σ DATA MEMORY	Δ TIME	Δ DATA MEM.
1	both	1	2871	2871	2683	-	-
2	greedy	4	6743	1686	3812	0	0
3	memory	4	6781	1696	3259	1%	-16%



Scheduling of Mars Path Finder under Power Consumption Constraints

The mars rover operates on very limited power supply. The power is given by solar panels. The power obtained from solar panels was measured at different temperatures and the results were the following: 14.9W at -40°C, 12.0W at -60°C and 9.0W at -80°C. There is a battery power source too, which gives maximal 10.0W and it is not replenishable energy so the battery power should be used as little as possible. The mars rover has 6 driving and 4 steering motors, which need to be warmed up before respective driving and steering can be performed.



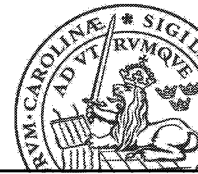
Scheduling of Mars Path Finder under Power Consumption Constraints

Operation	Duration	
Heating steering motors (HSM1&2, HSM3&4)	5s	At least 5s and at most 50s before steering starts
Heating wheel motors (HWM1&2, HWM3&4, HWM5&6)	5s	At least 5s and at most 50s before driving starts
Hazard detection (HD1 & HD2)	10s	At least 10s before steering starts
Steering (Steer1, Steer2)	5s	At least 5s before driving
Driving (Drive1, Drive2)	10s	At least 10s before next hazard detection starts



Scheduling of Mars Path Finder under Power Consumption Constraints

Task	Duration	Power -40°C	Power -60°C	Power -80°C
Heat two motors	5s	7.6W	9.5W	11.3W
Drive	10s	7.5W	10.9W	13.8W
Steer	5s	4.3W	6.2W	8.1W
Hazard Detection	10s	5.1W	6.1W	7.3W
CPU	Constant	2.5W	3.1W	3.7W

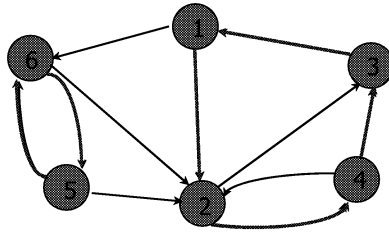


Modeling

- Precedence constraints:
 - $t_{hd1} + d_{hd1} \leq t_{steer1},$
 - ...
 - $t_{steer1} + d_{steer1} \leq t_{drive1},$
 - $t_{hwm12} \leq t_{steer1} + 50,$
- Power consumption constraints:
 - $cumulative([t_{hd1}, \dots, t_{sm12}],$
 - $[p_{hd1}, \dots, p_{sm12}],$
 - $[d_{hd1}, \dots, d_{sm12}], Power)$
- Optimize "Power"



Cycle (circuit) constraint



cycle(2, [[2,6], [3,4], [1], [2,3], [2,6], [2,5]])



[[2],[4], [1], [3], [6], [5]]



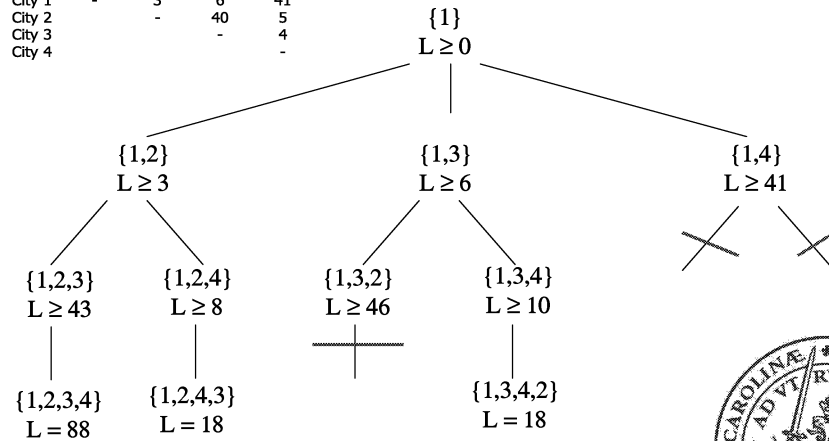
Search

- Standard search uses depth-first-search with backtracking.
- Optimization uses branch-and-bound or similar methods.



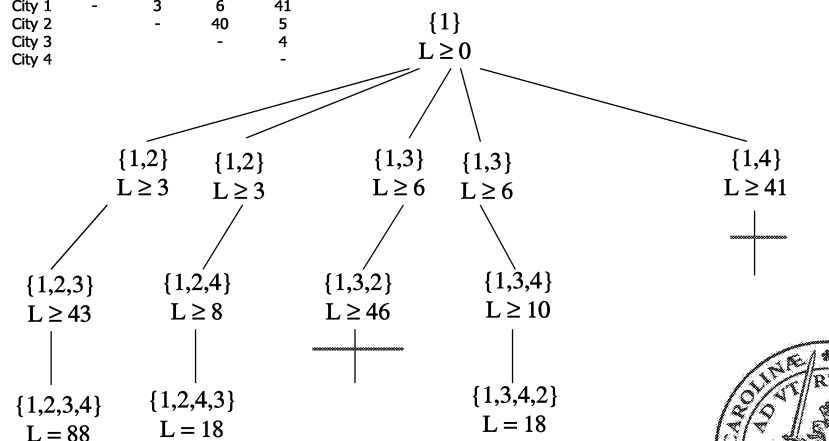
Typical branch and bound search (TSP problem)

	City 1	City 2	City 3	City 4
City 1	-	3	6	41
City 2	-	-	40	5
City 3	-	-	-	4
City 4	-	-	-	-



Search with restart (CLP typical)

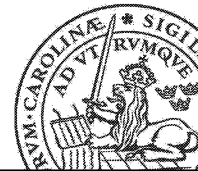
	City 1	City 2	City 3	City 4
City 1	-	3	6	41
City 2	-	-	40	5
City 3	-	-	-	4
City 4	-	-	-	-



Search (cont'd)

[City1::2..4, City2::{1,3..4}, City3::{1..2,4}, City4::1..3]

- How to select order of variable assignment?
 - dynamic vs. static
 - criteria
- How to select values to be assigned from variable's domain?
 - a single value
 - sub-domain
 - ...



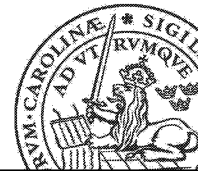
Variable Selection

- Static and dynamic
 - input order (static)
 - first-fail principle (smallest size of the domain)
 - smallest value in the domain
 - largest value in the domain
 - largest difference between the smallest and second smallest value in its domain
 - smallest max value in the domain
 - ...



Value Selection

- Single value
 - minimum in the domain and then upwards
 - maximum in the domain and then downwards
 - middle and then towards smallest and largest
 - random
 - ...
- Domain split
 - split into two sub-domains
 - split into N
 - ...

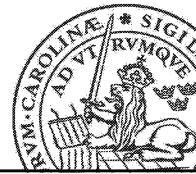
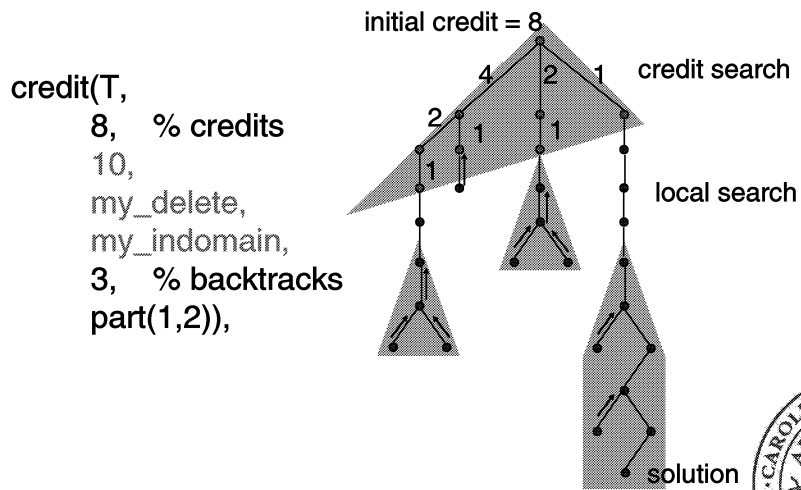


Search improvements

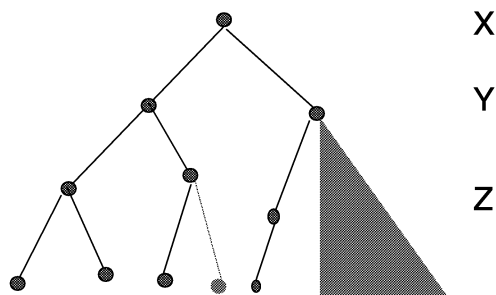
- Partial enumeration algorithms (instead of labeling)
 - Credit Search,
 - Limited Discrepancy Search (LDS).
- Assignment of subintervals instead of values to domain variables — possibly examines a bigger part of a solution space.
- Problem-dependent specific heuristics.
- Neighbourhood search...



Credit search



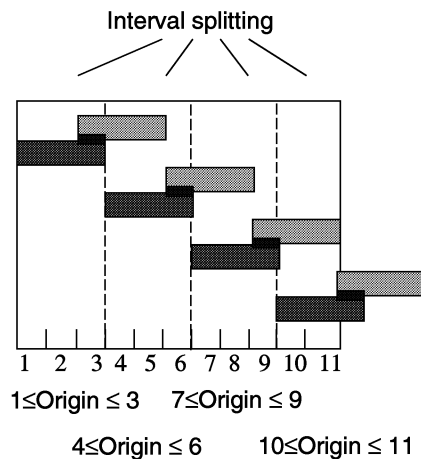
Limited discrepancy search



min_max(lds ([X,Y,Z], 1, input_order, indomain, Cgsf



Interval splitting



For each task:

```
Origin :: min..max
duration

Rest    :: 0..duration-1,
Quotient :: 0..max,
Quotient*duration+Rest #= Origin.
```

Enumeration procedure:

```
labeling(Origins, Quotients) :-
    labeling(Quotients, first_fail, indomain),
    labeling(Origins, first_fail, indomain).
```

Summary and conclusions

- **Advantages:**
 - focus on a specification of the problem, not on a solution method.
 - unified framework for different algorithms to be used to solve a problem (by encapsulating them as constraints).
 - easy definition of problems with many heterogeneous constraints.
 - easy extension of a problem by adding new constraints.
 - collaboration of solvers and distributed computing

Summary and conclusions

- **Limitations:**
 - NP-hard problems.
 - often non-predictable behavior of a solver.
 - difficult to define and add new constraints:
 - into existing systems — interface problems.
 - new propagation algorithms need to be developed.
 - difficult to match constraints with actual problems.



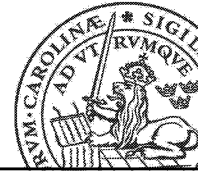
CP finite domain systems

- SICStus Prolog
- CHIP from COSYTEC
- IF/Prolog
- ILOG
- Mozart/Oz
- Gnu Prolog
- JaCoP – Java based our own solver



Selected CP Web resources

- Constraints archive
<http://www.cs.unh.edu/ccs/archive>
- Guide to constraints programming
<http://kti.ms.mff.cuni.cz/~bartak/constraints>
- Sicstus manual
http://www.sics.se/isl/sicstus/sicstus_toc.html
- Gnu Prolog
<http://www.gnu.org/software/prolog/prolog.html>
- Mozart/Oz
<http://www.mozart-oz.org/>



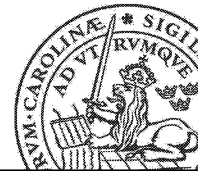
Other resources

- Book
 - K. Mariott and P. J. Stuckey *Programming with Constraints: An Introduction*, The MIT Press, 1998.
- Conferences
 - Principles and Practice of Constraint Programming (CP)
 - The Practical Application of Constraint Technologies and Logic Programming (PACLP)
- Journal
 - Constraints (Kluwer Academic Publishers)



Selected Papers

- * Kuchcinski, K., *Embedded System Synthesis by Timing Constraints Solving*, Proc. 10th International Symposium on System Synthesis, Antwerp, Belgium, September 17-19, 1997.
- * Gruian, F. and Kuchcinski, K., *Operation Binding and Scheduling for Low Power Using Constraint Logic Programming*, Proc. 24th Euromicro Conference, Workshop on Digital System Design, Västerås, Sweden, August 25-27, 1998.
- * Kuchcinski, K. and Wolinski, Ch., *Global Approach to Assignment and Scheduling of Complex Behaviours based on HCDG and Constraint Programming*, Journal of Systems Architecture, 2003, Elsevier Science.



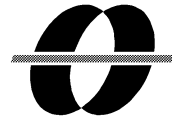
Selected Papers (cont'd)

- * Kuchcinski, K., *Constraints Driven Design Space Exploration for Distributed Embedded Systems*, Journal of Systems Architecture, vol. 47, no. 3-4, pp. 241-261, 2001, Elsevier Science.
- * Szymanek, R. and Kuchcinski, K., *A Constructive Algorithm for Memory-Aware Task Assignment and Scheduling*, Proc. 9th International Symposium on Hardware/Software Codesign, Copenhagen, Denmark, Apr. 2001.
- * Szymanek, R. and Kuchcinski, K., *Partial Assignment Technique for Task Graph Scheduling*, 40th DAC, Anaheim, USA, June 2003.
- * Kuchcinski, K. *Constraint-driven scheduling and resource assignment*, ACM Trans. on Design Automation of Electronic Systems, vol. 8, no. 3, pp. 355-383, 2003.



Sponsored by:





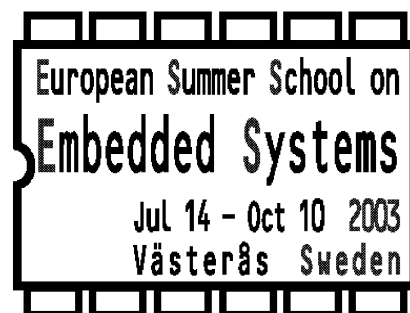
MÄLARDALENS HÖGSKOLA

ESSES 2003

European Summer School on Embedded Systems

Lecture Notes Part XI

Embedded Systems: Introduction and Overview



Editors: Ylva Boivie, Hans Hansson, Jane Kim, Sang Lyul Min

Strängnäs, August 20-22, 2003

ISSN 1404-3041

ISRN MDH-MRTC-106/2003-1-SE

MRTC

MÄLARDALEN REAL-TIME
RESEARCH CENTRE

www.mrtc.mdh.se

Real-Time Communication

Kang G. Shin

The University of Michigan

Real-Time Communication

Kang G. Shin

Real-Time Computing Laboratory
Department of Electrical Engineering and
Computer Science
The University of Michigan
Ann Arbor, MI 48109, USA
kgshin@eecs.umich.edu

Why and What Real-Time Communication?

- Between sensors & control panel and processors
Between processors
- Non RT protocols: throughput
RT protocols: (absolute/prob.) delay guarantees
- Message delay = formatting and/or packetization +
queueing + xmission + depacketization
- How to derive message deadlines?
- Message drop preference in case of congestion.
- RT traffic sources:
 - Constant rate
 - Variable rate
- Traffic characteristics may change inside the network

Network Topologies

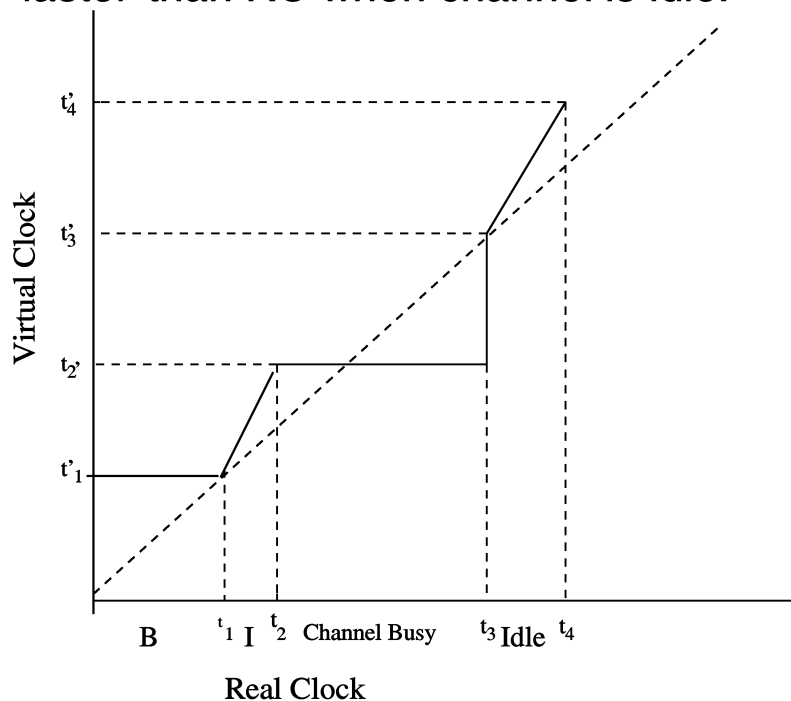
- Diameter, node degree, fault-tolerance
- Types:
 - Shared:**
 - Point-to-point:**
- Switching
 - Packet SW:** routing, flow control, & scheduling
 - Circuit SW:**
 - Cut-thru SW:** Wormhole, virtual cut-thru
- Interconnections: system and node levels, I/O connectivity

Virtual-time CSMA

- 2 synchronized clocks, “real” and “virtual,” for each station.

RC: to record message arrival times

VC: to count up to a virtual time to start xmission, $VSX(M)$. *freezes* when channel is busy and *runs faster* than RC when channel is idle.



- When $VC \geq VSX(M)$, packet M becomes eligible for xmission. If collision, modify VSX value.
- Question: How to initialize when channel becomes free, and how to modify $VSX(M)$ when \exists a collision involving M?

More on VTCSMA

τ :	Signal propagation time
A_M :	Arrival time of message M
T_M :	Time required to xmit M
D_M :	Deadline of M
L_M :	Latest time M to be sent to meet D_M
$\Lambda_M(t)$:	$D_M - T_M - \tau - t$

$$VSX(M) = \begin{cases} A_M & \text{for VTCSMA-A} \\ T_M & \text{for VTCSMA-T} \\ L_M & \text{for VTCSMA-L} \\ D_M & \text{for VTCSMA-D} \end{cases}$$

In case of collision, M is retransmitted immediately with prob p or not; and $VSX(M)$ is modified to be a random # in

$$I = \begin{cases} (\text{current VC}, L_M) & \text{for VTCSMA-A} \\ (0, T_M) & \text{for VTCSMA-T} \\ (\text{current RC}, L_M) & \text{for VTCSMA-L} \\ (\text{current RC}, D_M) & \text{for VTCSMA-D} \end{cases}$$

More on VTCSMA

- When channel changes from busy to idle

$$VC = \begin{cases} \text{no change} & \text{for VTCSMA-A} \\ 0 & \text{for VTCSMA-T} \\ RC & \text{for VTCSMA-L and -D} \end{cases}$$

- Effects of clock skew

Node	M	Actual RC at M's arrival	RC at node	D_M	L_M
1	1	8	actual RC-1	32	16
2	2	9	actual RC+1	36	20

VC runs twice faster than RC.

N_1 and N_2 xmit M_1 and M_2 at their VC values 16 and 20, corresponding to their RC values 8 and 10 which are identical.

- Lower loss rate than CSMA but wastes net bandwidth by idling the net.

Window Protocol

- Use events on the bus to synchronize node actions
- Each node maintains a time “window”
- When L_M of a packet falls in this window and the channel is idle, the packet is *eligible* for xmission.
- Each node maintains a stack of window history, (upper bound of window, ID of collided message or 0).
- Shrink or enlarge window based on the events on the bus.
- What does a collision even when $w = 1$ mean and what to do?
- What’s a slot?

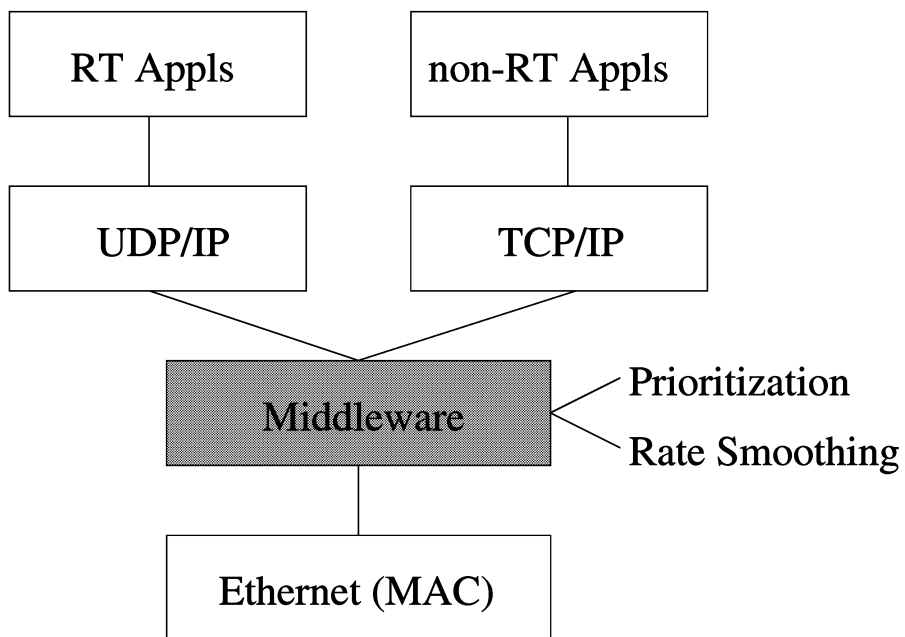
Ethernet-based RT Communication

- Advantages
 - low price & simple management
 - mature technology
- Difficult to provide RT guarantees — CSMA/CD
 - packet collision
 - multiple collisions due to bursty non-RT packets

Ethernet MAC protocol

- CSMA/CD protocol
 - No exclusive medium control
 - Listen before transmit
 - Stop transmission upon sensing collision
- Binary Exponential Backoff (BEB): backoff range increases with collisions.

A New Solution



Fixed-Rate Traffic Smoothing



Bursty stream

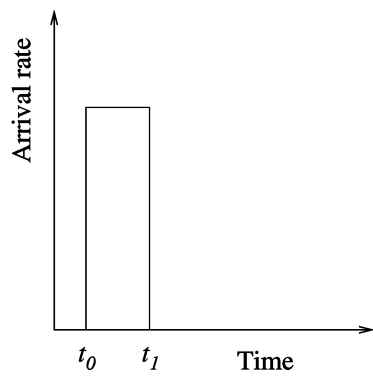


Smoothed stream

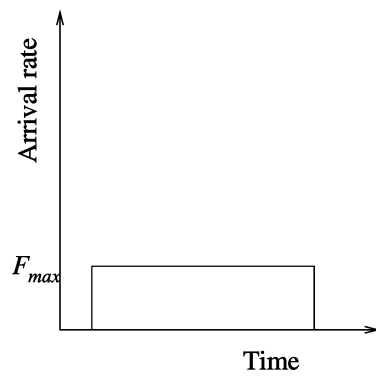
Disadvantages of Fixed-Rate Smoothing

- Poor scalability due to network-wide input limit distributed to stations
- Low network utilization by non-RT traffic
 - constant station input limit
- Solution – adaptive-rate traffic smoothing

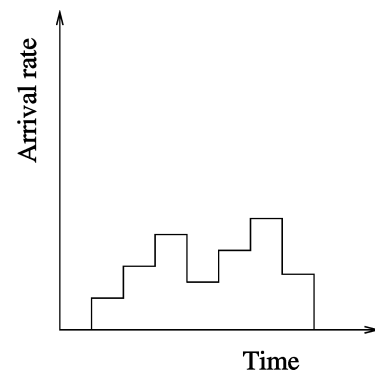
Fixed-Rate vs. Adaptive-Rate Smoothing



(a)



(b)



(c)

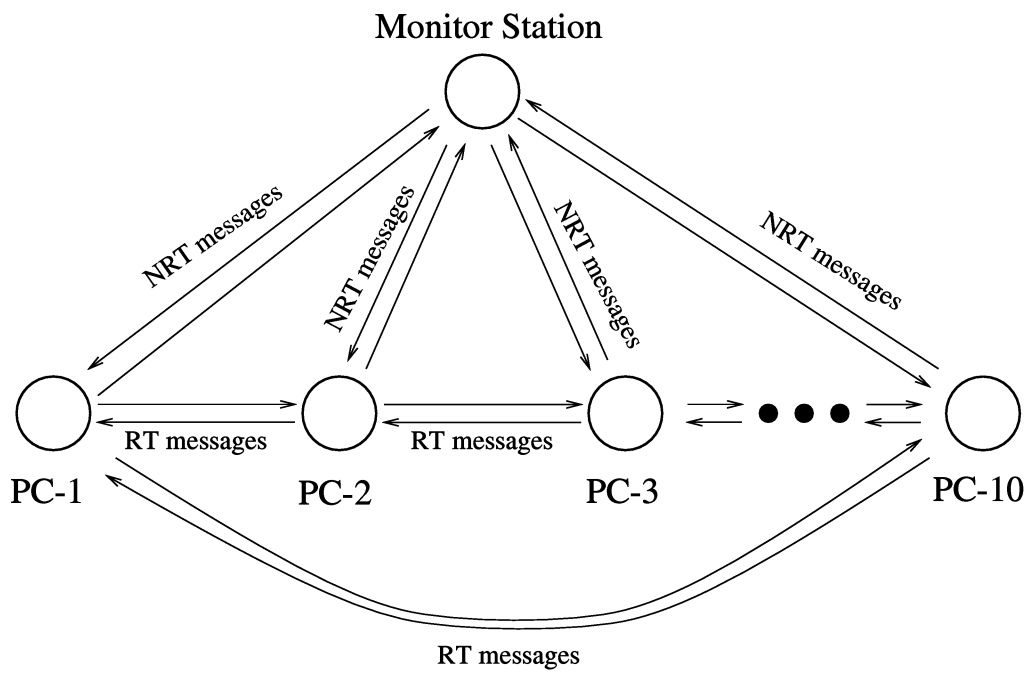
Harmonic-Increase and Multiplicative-Decrease Adaptation

- Parameter: minimum packet inter-arrival time or refreshing period (RP)
- Uses NIC-collected collision statistics
- Adaptation
 - No collision: RP decreased by Δ periodically
 - Upon collision: RP doubled

Experimental Evaluation of Adaptive-Rate Smoothing

- 11 PCs
- 10BASE-T Ethernet LAN
- Collision domain diameter = 10 m

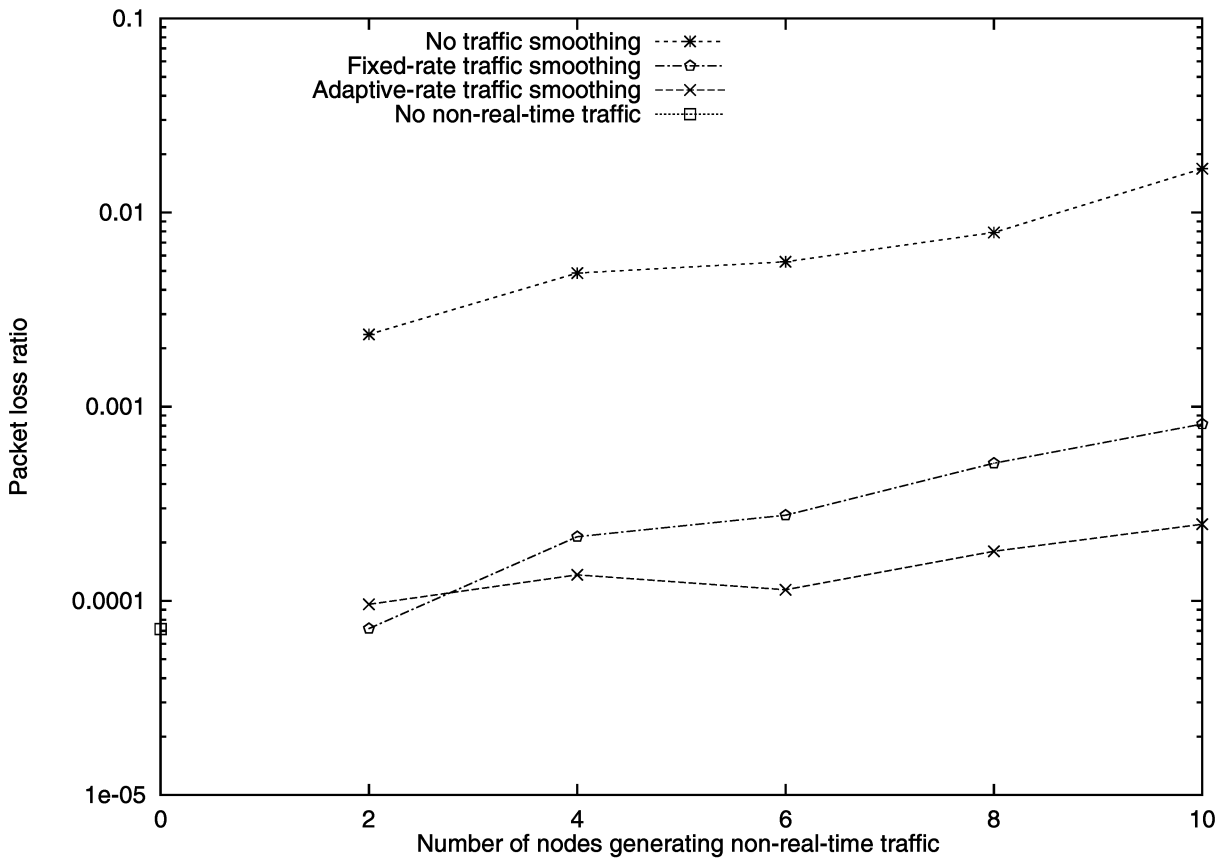
Topology



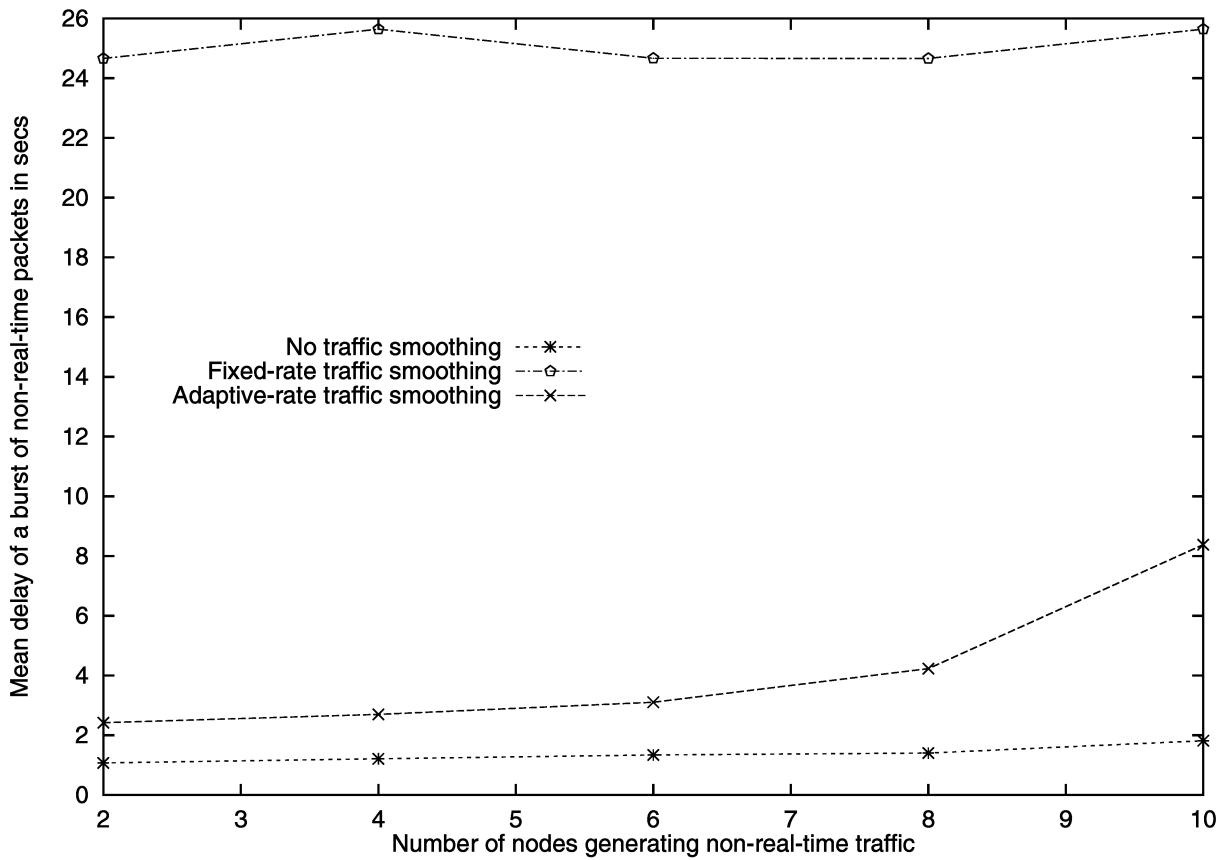
Traffic Generation Statistics

- RT traffic
 - a 100 byte long message every 0.3 sec from PC-n
 - total RT traffic generation rate – 53.3 kbps
 - roundtrip deadline – 129.6 msec
- non-RT traffic
 - non-greedy mode
 - * a burst (1 MB) every 25 sec from an activated node
 - * 320 kbps from a single node
 - greedy mode
 - * zero burst inter-arrival time
 - * variable throughput

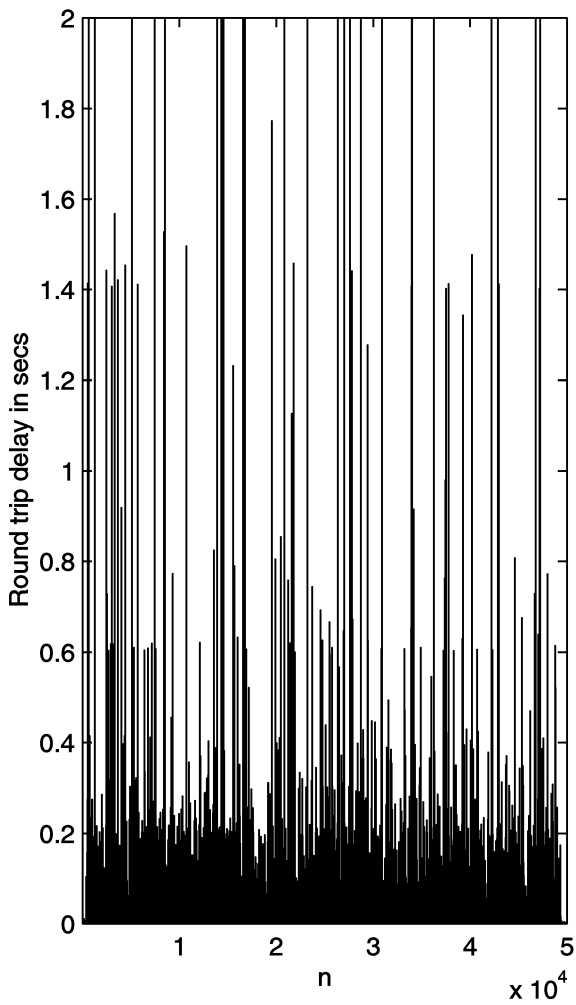
RT Message-Loss Ratio (non-greedy mode)



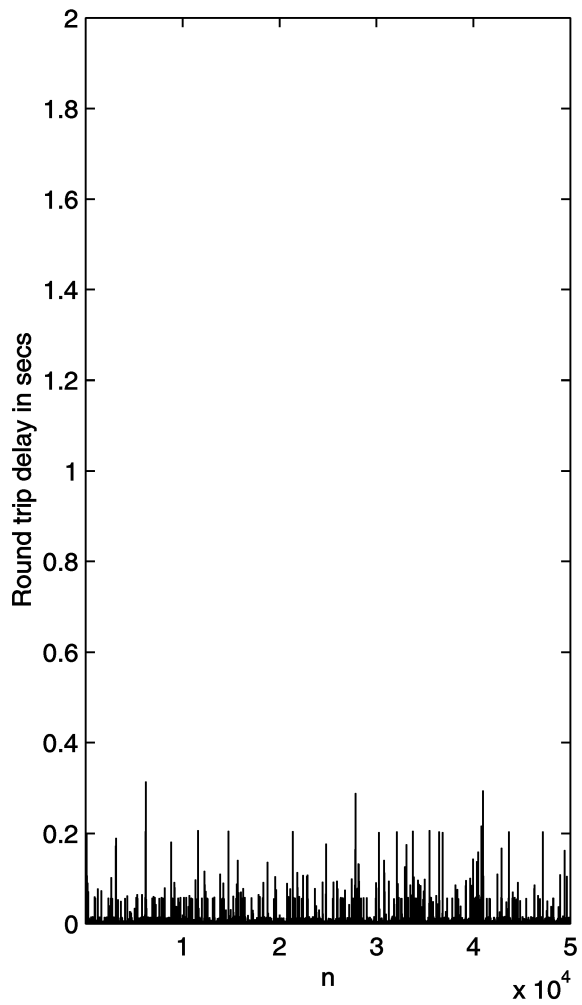
Avg Xmit Time of non-RT Bursts (non-greedy mode)



RT Traffic Roundtrip Delays

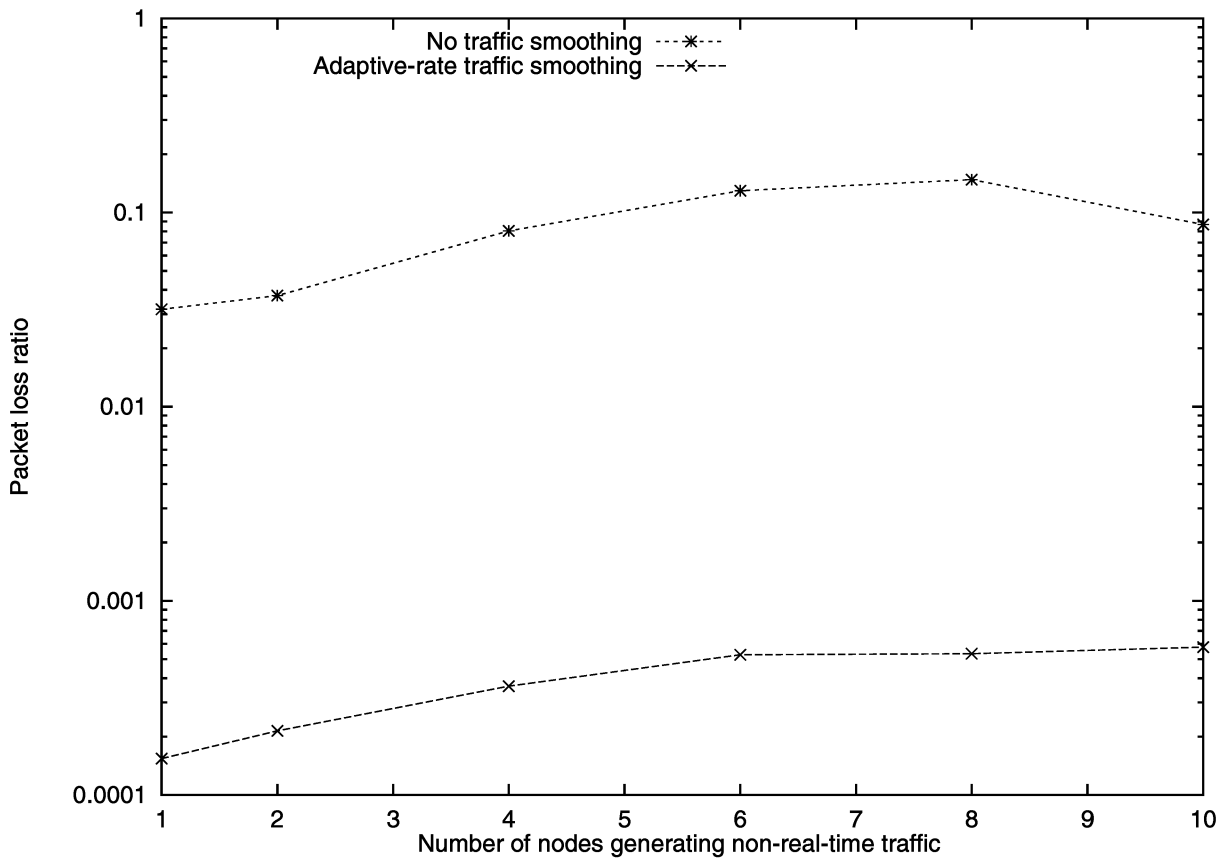


No smoothing

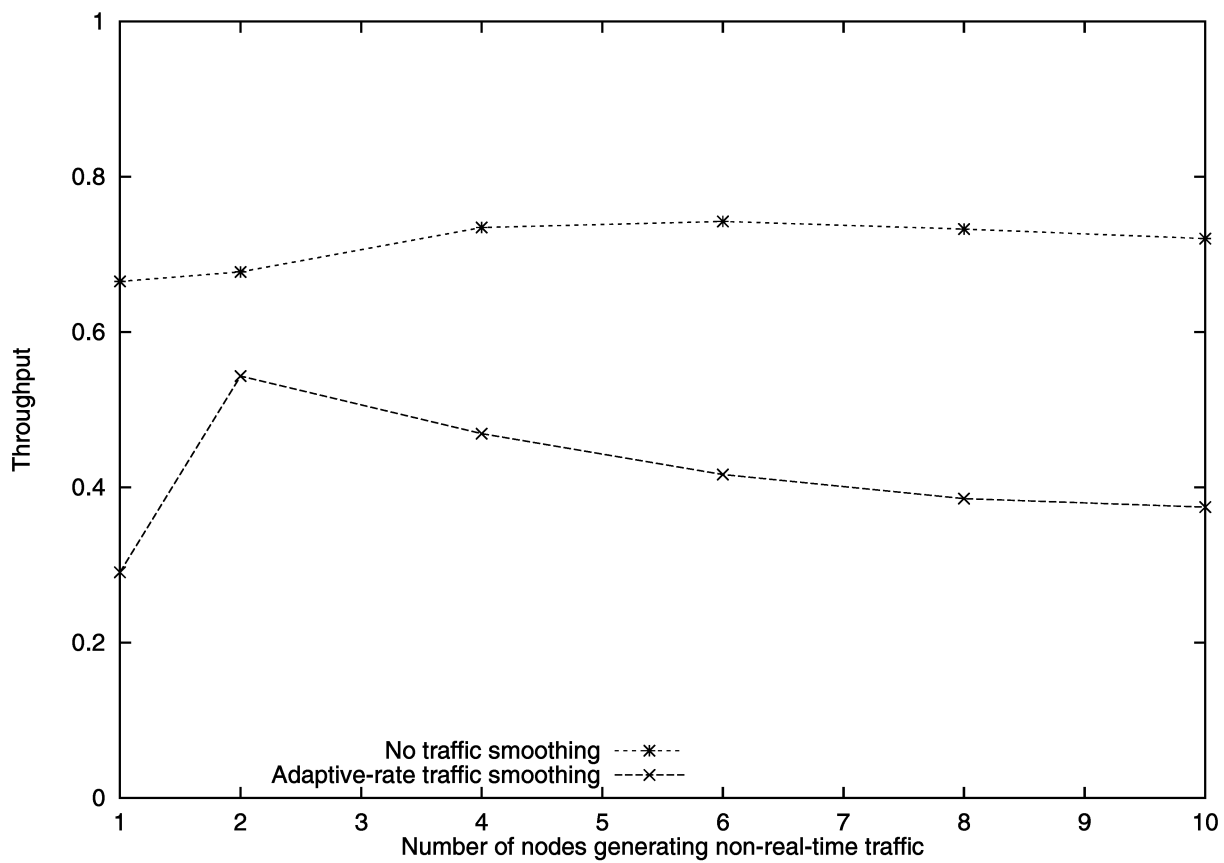


adaptive smoothing

RT Message-Loss Ratio (greedy mode)



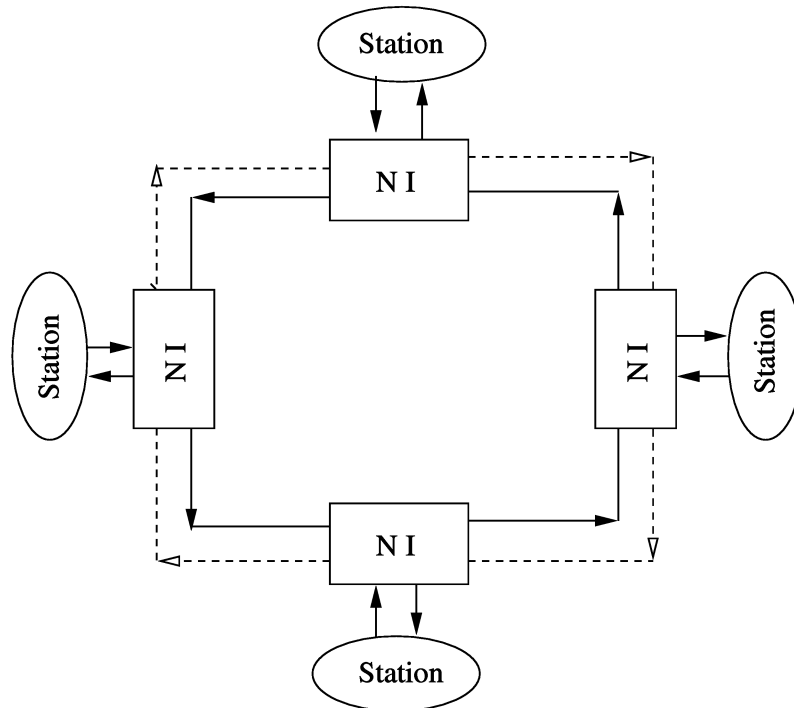
Throughput of non-RT Traffic



Summary on Real-Time Ethernet

- Ethernet augmented with middleware for soft RT guarantees
 - compatible with TCP/IP, UDP/IP and Ethernet standard
 - implemented on Linux and Windows NT
- Prioritization and traffic smoothing
- Fixed-rate traffic smoothing
- Adaptive-rate traffic smoothing

Token-Passing Protocols



Overheads: med. propagation time, token xmission time and capture delay, and NI latency.

Timed-Token Protocol (TTP): 2 types of traffic

- Synchronous/RT: xmit up to h units of time (UOTs) every T UOTs.
- Asynchronous/non-RT: uses any bandwidth left unused by synchronous traffic.

Target Token Rotation Time (TTRT)

- Token cycle time $\leq 2 \times \text{TTRT}$
How do you determine TTRT?
- Time available to xmit packets per TTRT
 $t_p = \text{TTRT} - \Theta$
- Node i gets SBA, $B \times \frac{h_i}{t_p}$.
- If cycle time $>$ ($<$) TTRT token is *late* (*early*). Xmit only synch. traffic up to h_i (AND a certain amount of asynch. traffic).
- *Questions*
 - How to determine h_i ?
 - Why token early or late?
 - Is avg token cycle time still $\leq \text{TTRT}$?

Supporting Periodic Communication

- Node i needs to xmit c_i bits of RT traffic every P_i UOTs

- $TTRT \leq P_i/2$

- Synch. bandwidth is greater than $t_p B c_i / \lfloor \frac{P_i}{TTRT} - 1 \rfloor$

- How to deal with token loss?

Claim-token (contains TTRT it requests)

Beacon packet if every node doesn't

reinitialize its own claim-token or a

normal packet within $TTRT(i)$ after xmitting claim-token.

Station immediately downstream from a break is the only node that will keep xmitting.

IEEE 802.5 Token Ring Protocol

SD	AC	ED
----	----	----

Token

SD	AC	ED	DA	SA	Message	ECC	ED	FS
----	----	----	----	----	---------	-----	----	----

SD: Starting Delimiter
 AC: Access Control
 ED: Ending Delimiter
 DA: Destination Address

SA: Source Address
 FS: Frame Status
 00 dest unavailable
 10 frame uncopiable
 11 frame copied

- FS field checked by the sender
- Sender removes data frame it sent
- Priority arbitration via AC field
 - 3 bits for current and reserved priorities
 - When data frame or a token goes by, a node checks the reserved priority beginitemizets: do nothing if higher else write its priority into AC
 - Upon completion of current xmission, the sender issues a token with priority in AC
 - Node that increased priority is responsible for restoring it to prior priority value
- Schedulability Analysis of Token Ring
 T_1, T_2, \dots, T_n are schedulable iff $\forall i \exists t \leq d_i$ such that $\sum_{j+1}^i e_j \lceil \frac{t}{P_j} \rceil + \text{system overhead} + b_i \leq t$.

Polled Bus Protocol

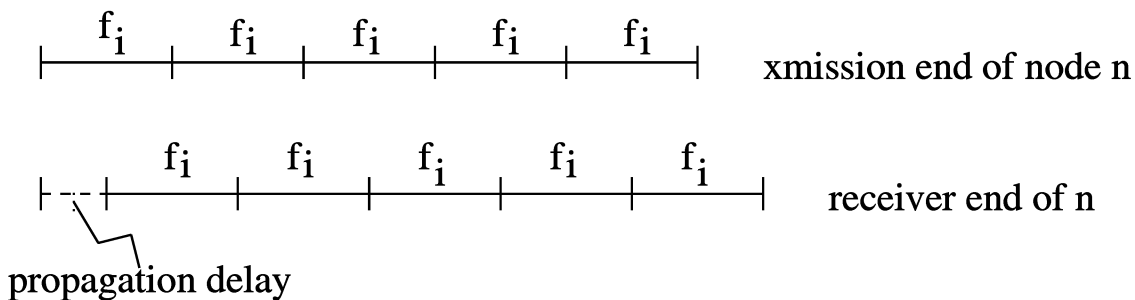
- Synchronous contention resolution
- Stations contend for the channel during a contention period
- The station with the highest priority message gets access to the medium
- Example: Controller Area Network (CAN)

Bus Acquisition Algorithm

- Calculate a unique ID of m beginitemizets.
- If bus not busy then
 - Write its ID to the bus, one beginitemizet at a time, starting with MSB
 - Wait for a fi nite time and sample the bus
 - If the value read by the processor is different from the value it wrote into the bus, it drops out
 - After m rounds, the processor with the highest ID has sole control of the bus.
- The bus is assumed to wired-OR all impinging signals and all stations are synchronized.
- Key: how to design station IDs?

Stop-and-Go or Framing Protocol

- Frame = a time interval
Each frame type represents a diff time interval and is associated with a *traffic class*.
- Upon arrival of a type- f_i pkt at an intermediate node n , it's held by n at least till the beginning of next instance of f_i .



- All nodes eligible for xmission are served in nonpreemptive priority order, with shorter-frame pkts having priority over longer-frame pkts
- If net load \leq a certain limit, a type- m pkt will be xmitted within f_m time units of benditemizeng eligible for xmission, i.e., bounded delay at each hop

Hierarchical Round-Robin Scheduling Protocol

- Guarantees each traffic class i to transmit m_i pkts every T_i UOTs.
- Traffic is classified into n classes, where each class i is associated with (n_i, b_i, Φ_i)
 - n_i : max # of class- i pkts transmittable during any given frame of which source j is allocated a certain max # α_j .
 - If α_j pkts are transmitted or no class- i pkts left for transmission, class- $(i+1)$ pkts if any are transmitted, etc., for a max of b_i pkts during that class- i frame
 - Φ_i = frame associated with class i
 $\Phi_1 < \Phi_2 < \dots < \Phi_n$.

Point-to-Point Networks

- Attractive because of fault-tolerance capability
- Allow multiple conversations to go on simultaneously on different links
- Access to the links can be controlled easily
- Drawback: Higher latency
- Desirable: efficient broadcast algorithms

Guaranteed Delivery

- Message generation characteristics
 - Source, Destination
 - Maximum message length: S_{max} (bytes)
 - Minimum inter-arrival time: R_{max} (msgs/sec)
 - Maximum burst size: B_{max} (msgs)
 - Desired bound on message latency: D
- In any time interval of length t , the number of messages generated may not exceed $B_{max} + t \cdot R_{max}$.
- A pair of *uni-directional* real-time channels should be established between source and destination *before* messages can be transmitted on them.

Delivery Time Guarantee

- The *logical generation time*, $\ell(m)$, for a message m is defined as

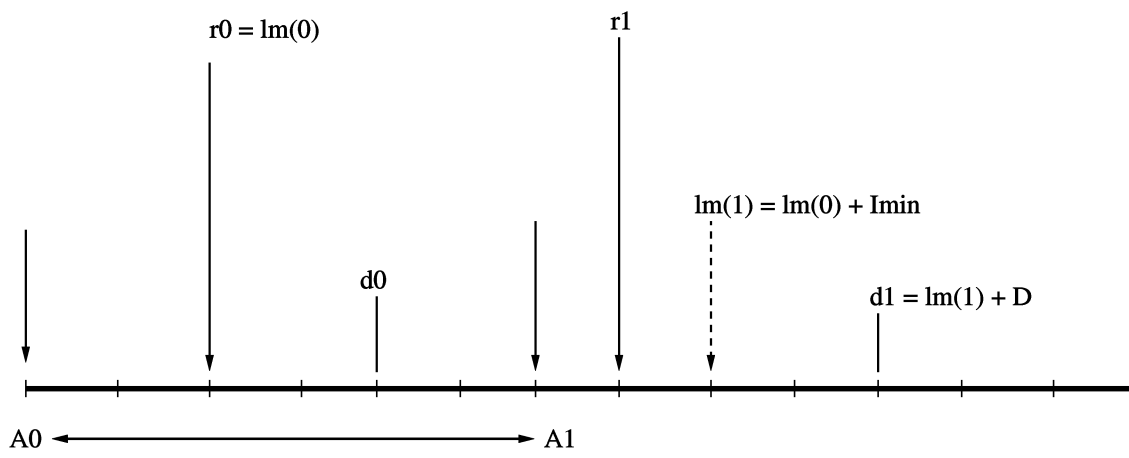
$$\ell(m_0) = t_0$$

$$\ell(m_i) = \max\{\ell(m_{i-1}) + I_{min}, t_i\}$$

where t_i denotes the actual generation time of message m_i , and I_{min} is the reciprocal of R_{max} .

- If D is the end-to-end delay for the channel, the system *guarantees* that any message m_i will be delivered to the destination node by time $\ell(m_i) + D$.

Illustration



Channel Establishment

- Select a source to destination route for the channel
- Compute a feasible worst-case delay at each link (if possible) based on the characteristics of all channels in the system
- Check whether the total delay is acceptable and redistribute the delays
- Compute the buffer requirement at each link
Note: This computation is dependent upon the link message scheduling algorithm used during transmission.

Delay Computation

- should maintain the feasibility of existing channels
- should obtain the minimum feasible delay
- should be linked to the run-time message scheduling policy
- distinguish between the feasibility testing and the run-time message scheduling policy
- use an optimality result proved by Dertouzos ('74)

Assignment Procedure

- Arrange the channels in ascending order of their associated delay d_i .
- Assign the highest priority to the new channel M_{k+1} . Assign priorities to the other channels based on their delay.
- Compute the new (worst-case) response times r'_i for the existing channels based on this priority assignment.
- Find the smallest position q such that $r'_i \leq d_i$ for all channels with priority less than q .
- Assign priority $q + 1$ to the new channel and compute the response time r'_{k+1} .

Response Time

Consider a set of channels

$\{M_i = (C_i, d_i, p_i), i = 1, \dots, m\}$ which share a common link ℓ .

$$S_i = \{d_i\} \cup \{kp_j \mid j = 1, \dots, i-1; k = 1, \dots, \lfloor (d_i/p_j) \rfloor\}$$

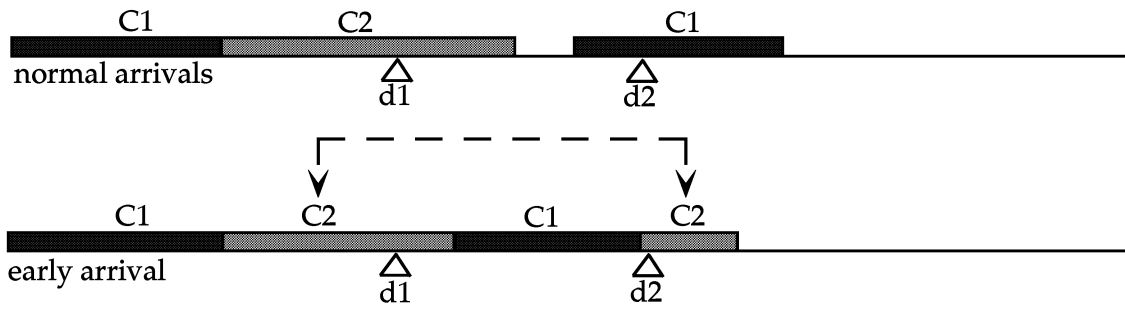
$$W_i(t) = \sum_{j=1}^{i-1} C_j \cdot \lceil t/p_j \rceil + C_i$$

The worst-case response time for messages belonging to M_i is the smallest value of t such that $W_i(t) = t$.

Run-time Scheduling

- Problem with fixed-priority scheduling: arrivals are not strictly periodic
 - arrival time at a node depends on the actual delay at the previous node
 - model allows burst arrivals
- High priority arrivals can disrupt the scheduling of lower priority messages

Illustration



Run-time Scheduling

- Deadline Scheduling can be used to overcome this problem
- Based on the logical arrival time of the message at a node

The logical arrival time for m_i at node b , $\ell_{c,b}(m_i)$, is defined as

$$\ell_{c,b}(m_i) = \ell_{c,a}(m_i) + d_{c,a}$$

where $d_{c,a}$ is the worst-case delay for messages on channel M_c at node a .

- Is the feasibility testing still valid?

Run-time Scheduling

Uses a multi-class Earliest Due Date (EDD) algorithm

Queue 1 Packets belonging to real-time channels with $\ell_c(m_i) \leq \text{current_time}$, arranged in the order of increasing deadlines.

Queue 2 Other packets arranged in the order of increasing deadlines.

Queue 3 Packets belonging to real-time channels with $\ell_c(m_i) > \text{current_time}$, arranged in the order of increasing logical arrival time.

Buffer Management

- Buffer space is reserved for channels at the source, destination, and at intermediate nodes.
 - depends on B_{max} , R_{max} , and the link delays
- Flow-control enforced: time-based
 - packets in Queue 3 are considered for transmission only when their logical arrival time \leq current time + *horizon*

Buffer Requirement

- min space = $S_{max} \cdot \lceil (d_{prev} + d_{node}) / I_{min} \rceil$
- buf space = $S_{max} \cdot \lceil (d_{prev} + d_{node} + H) / I_{min} \rceil$
- max space
= $S_{max} \cdot \lceil d_{cumul} / I_{min} \rceil + S_{max} \cdot B_{max}$

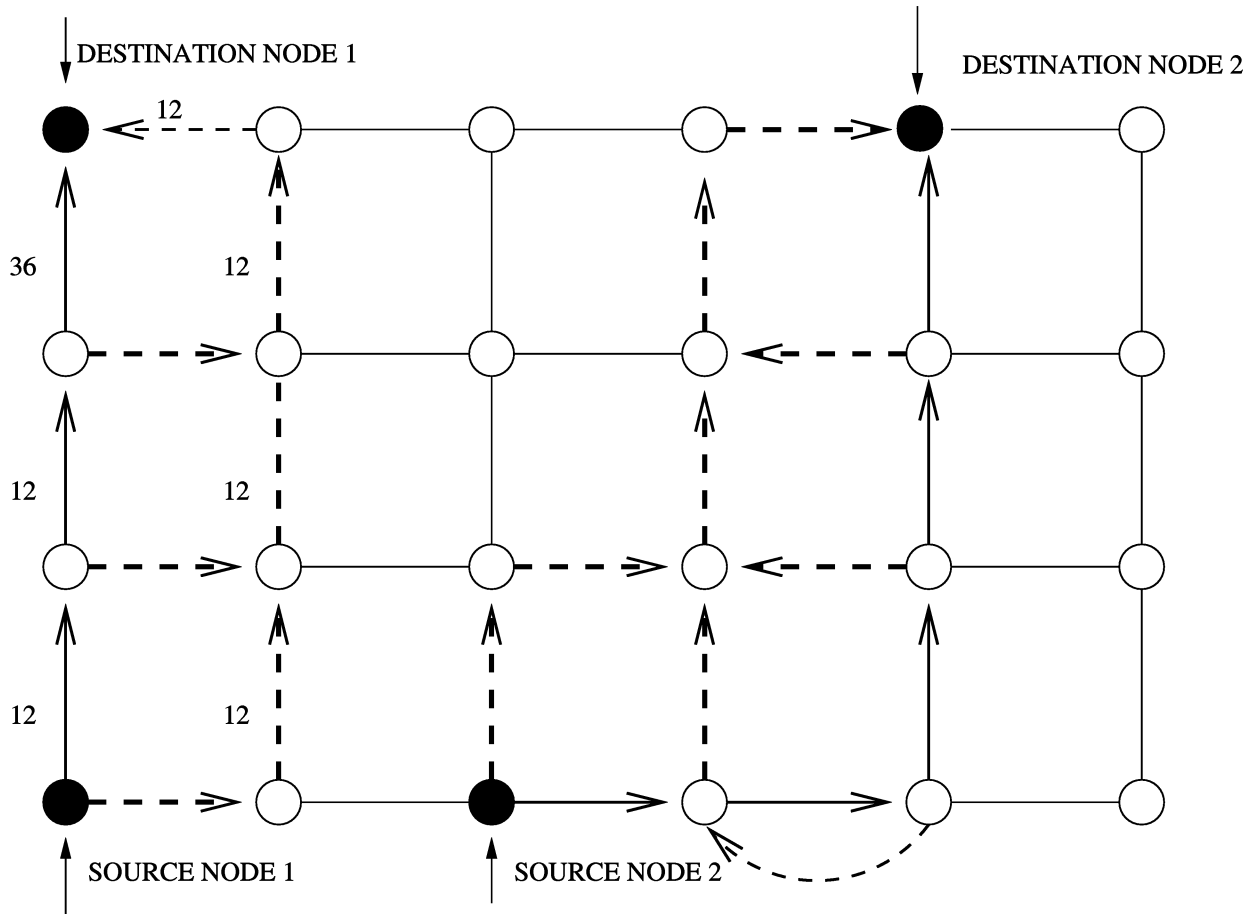
Fault-Tolerant Real-Time Channel

- Static routing makes real-time channels unable to tolerate component failures
- Dynamic routing would make it difficult to guarantee delivery-delay bounds
- Possible solutions:
 - Partially-dynamic routing or local detours
 - Multiplexed backup channels
 - Reactive approach, i.e., do nothing until breaks.

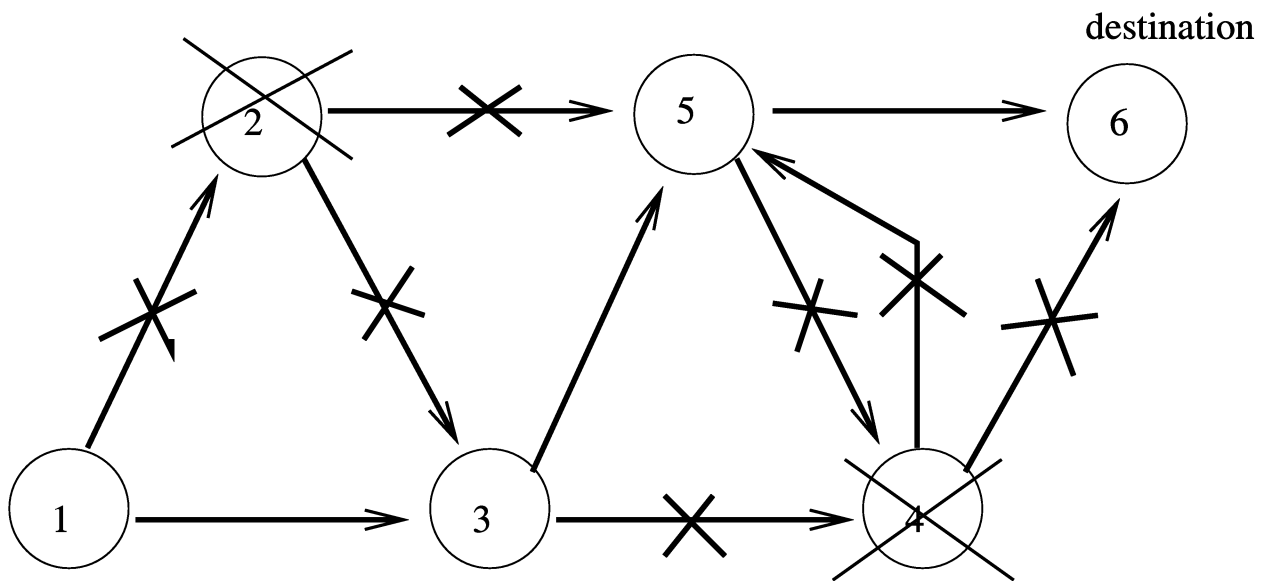
Partially-Dynamic Routing

- Set up a primary real-time channel
- Enhance the channel with some extra links and nodes
- Use the primary under normal circumstances, and use the extra links/nodes when the primary breaks down.

Single Failure Immune RTC



Isolated Failure Immune RTC



Backup Channels

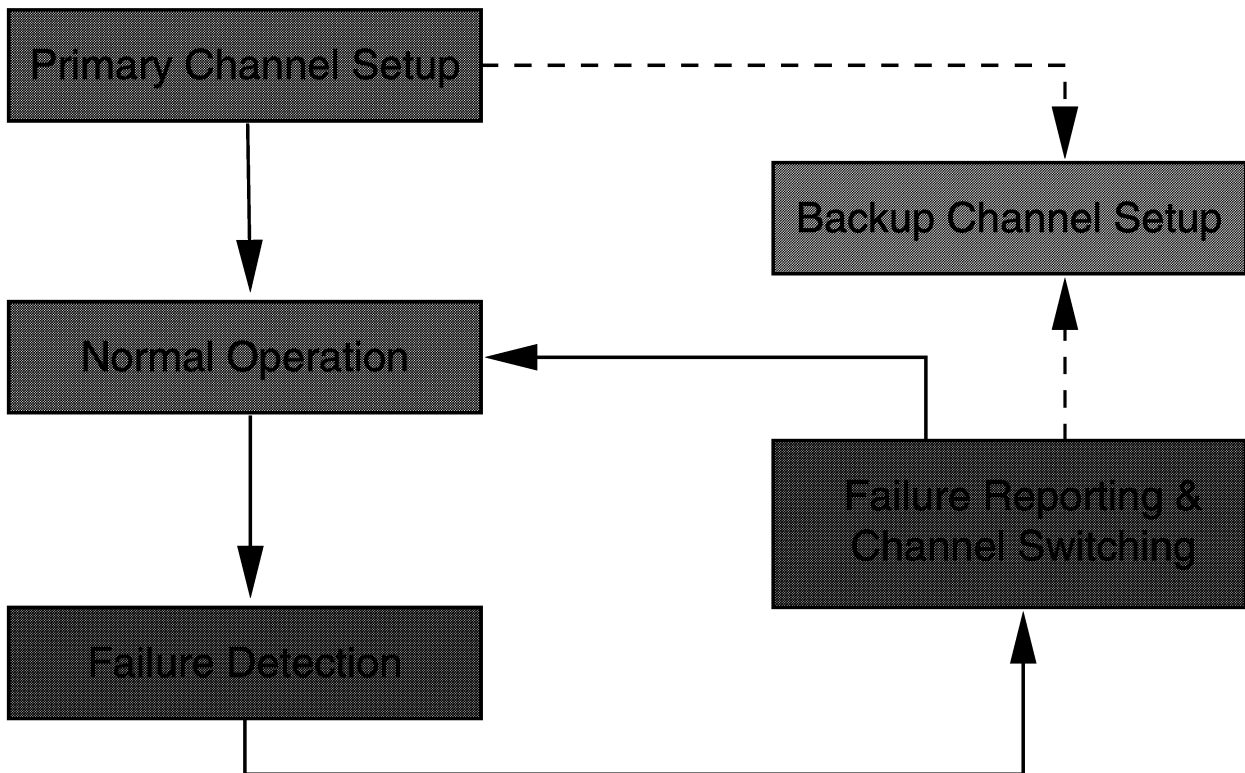
Approach:

- A dependable connection = a *primary* channel + *backup* channels
- Reservation of *spare resources* in advance
- Advance recovery-route selection (off-line end-to-end rerouting)

Issues:

- *Per-connection* dependability-QoS control
- Spare resource allocation
- Channel failure detection
- Time-bounded failure recovery
- Resource reconfiguration

Overview of Self-Healing Recovery



Summary

- E2E real-time communication is achieved via
 - connection establishment
 - run-time scheduling
 - buffer management
- Extensions for fault-tolerance
 - Local detours: SFI and IFI
 - Backup channels and their multiplexing
 - Reactive approach

References can be found from
<http://kabru.eecs.umich.edu>

Wayne Wolf

Princeton University

Challenges in Embedded Computing

Wayne Wolf
Dept. of Electrical Engr.
Princeton University

The push from Moore's Law

⌘ Moore's Law continues---1.4 billion transistor chips @ 10 GHz in 2012.

microprocessor transistors/chip					
<i>1999</i>	<i>2001</i>	<i>2003</i>	<i>2006</i>	<i>2009</i>	<i>2012</i>
21M	40M	76M	200M	520M	1.40B

⌘ What do we do with all those transistors?

Filling up chips

- ⌘ Effective use of VLSI manufacturing requires:
 - ⊞ large volumes;
 - ⊞ standard services, but with product differentiation;
 - ⊞ fast design time.
- ⌘ Fast turnaround means:
 - ⊞ lots of memory;
 - ⊞ embedded processors.

The application push

- ⌘ Internet is a platform for distributed computing.
 - ⊞ But must come up with interesting applications.
- ⌘ Non-PC appliances will become increasingly important.
 - ⊞ But they must be cheap.

Potential applications

⌘ Wireless

- ☒ already here
- ☒ 3G pushes more complex applications

⌘ Multimedia

- ☒ digital cameras with compression popular
- ☒ more sophisticated applications are possible

One way to use a billion transistors

⌘ Multimedia-rich Internet information appliances:

- ☒ Multimedia allows people to use the Internet for person-person communication.
- ☒ Complex functionality requires large software content.
- ☒ Multimedia requires memory.
- ☒ High-volume market requires single chip.

Characteristics of embedded systems

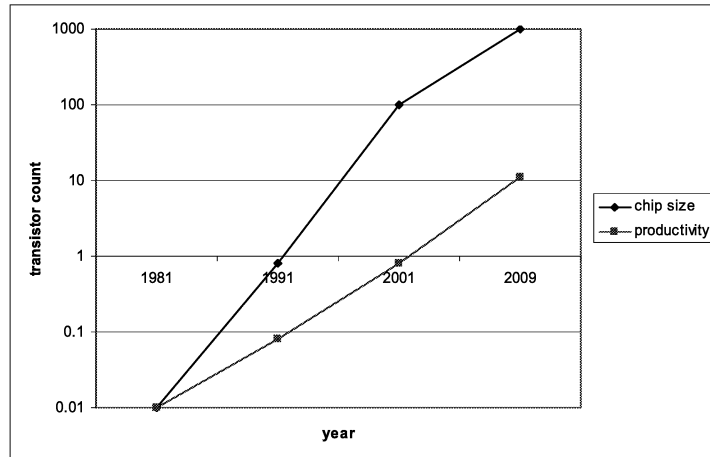
- ⌘ Very high performance.
 - ☒ Vision + compression + speech + networking all on the same platform.
- ⌘ Multiple task, heterogeneous.
- ⌘ Real-time.
- ⌘ Often low power.
- ⌘ Highly reliable.
 - ☒ I reboot my piano every 4 months, my PC every day.

System design challenges

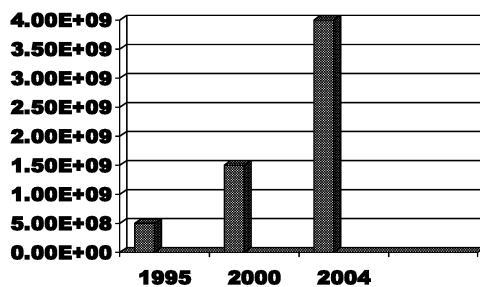
- ⌘ Single-chip multiprocessor architectures.
- ⌘ Embedded real-time software.
- ⌘ Embedded memory.
- ⌘ IP-based design.
- ⌘ Design verification.
- ⌘ Testing.

Design productivity gap

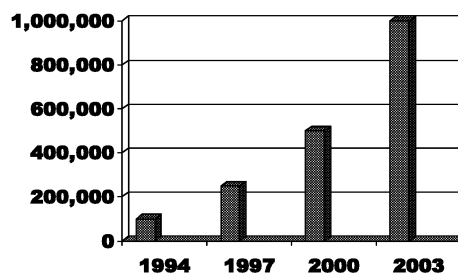
✂ From ITRS 99:



Design costs



Fab line cost



Mask costs

What do we do?

- ⌘ Move some design tasks to software.
 - ☒ Need abstract models of hardware for performance, power.
- ⌘ Create platform architectures.
 - ☒ Tweak the platform to create products.
 - ☒ Design reliability, low power, etc. into the platform for use by software.

Architectures and tools

- ⌘ Tools:
 - ☒ More domain-specific at higher levels of abstraction.
 - ☒ Some horizontal abstractions (RTOS, compiler, etc.)
- ⌘ Architectures:
 - ☒ There will be more than one platform.
 - ☒ Domain knowledge is embodied in the architecture.

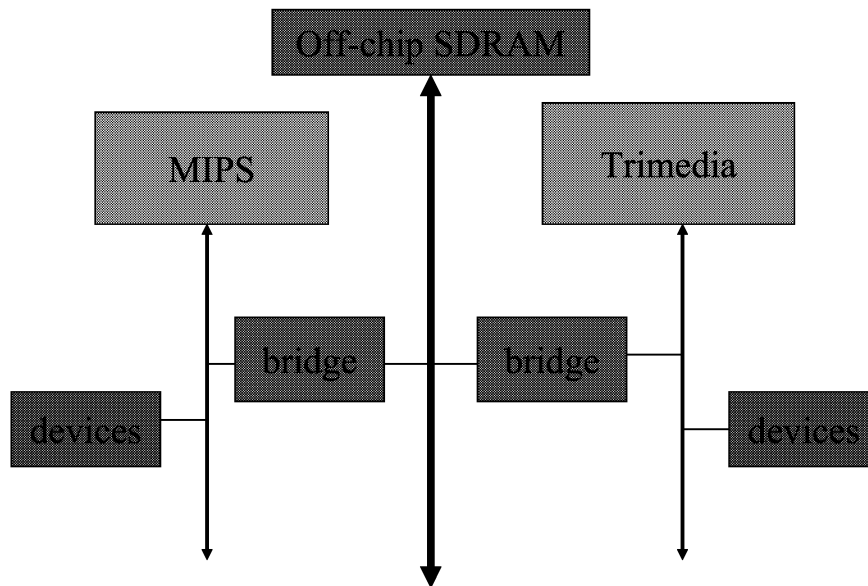
Why multiple platforms?

- ⌘ People still care about cost.
- ⌘ People care about power consumption.
- ⌘ Sufficiently general solutions don't fit on one chip.

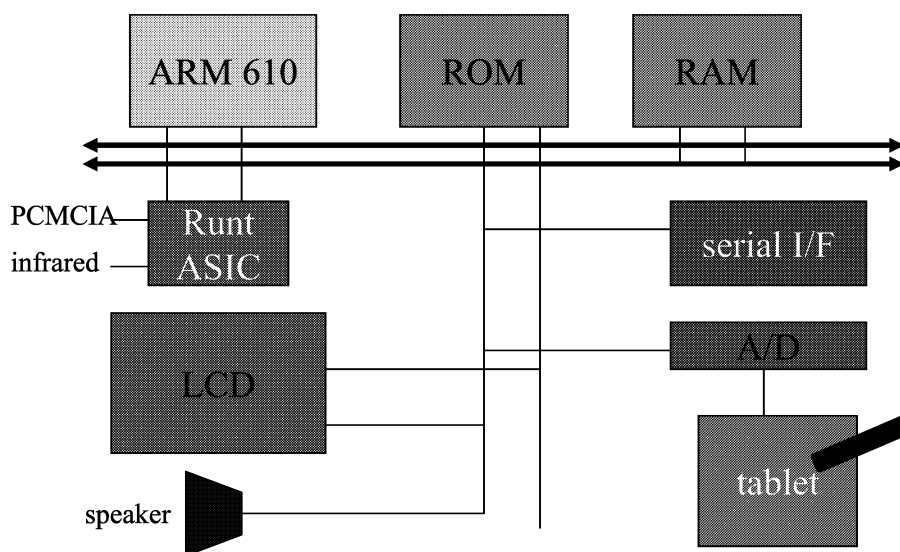
Multiprocessor systems-on-chips

- ⌘ Heterogeneous multiprocessors.
 - ☒ Optimized for area, power.
- ⌘ Processing elements may be CPUs or hardwired (or analog).
- ⌘ Custom memory architecture.
- ⌘ Moving toward networks-on-chips.
- ⌘ Need software development platforms.

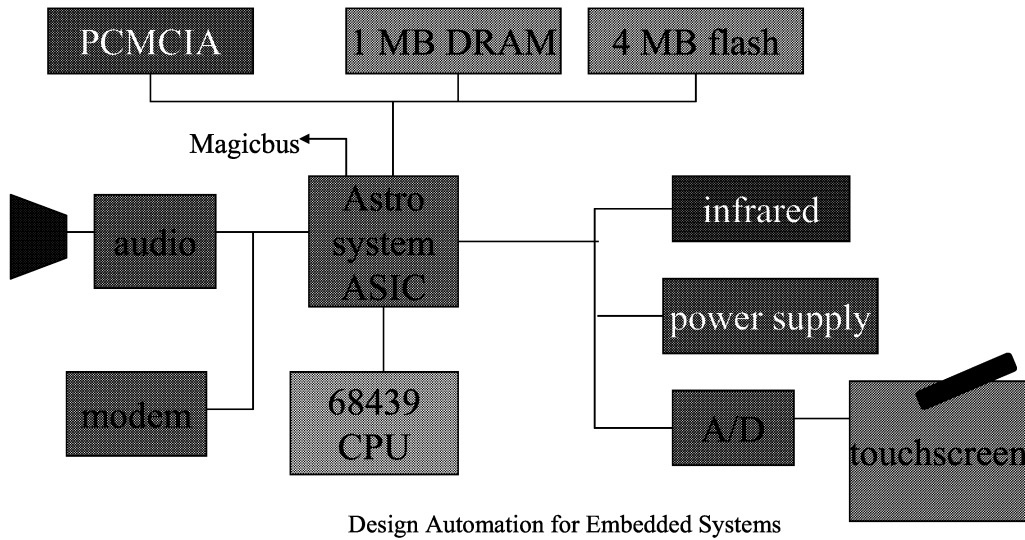
Viper set-top-box chip



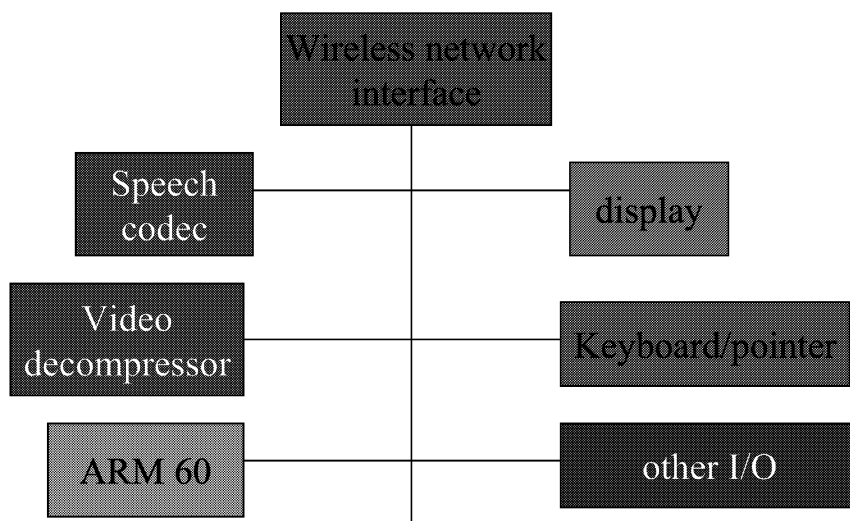
Apple Newton hardware architecture



Motorola Envoy hardware architecture



InfoPad hardware architecture



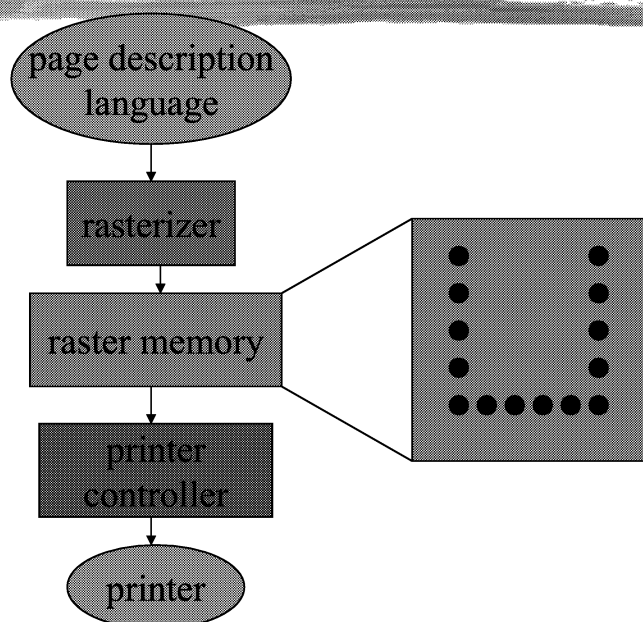
IEEE Trans. Computers

Printers

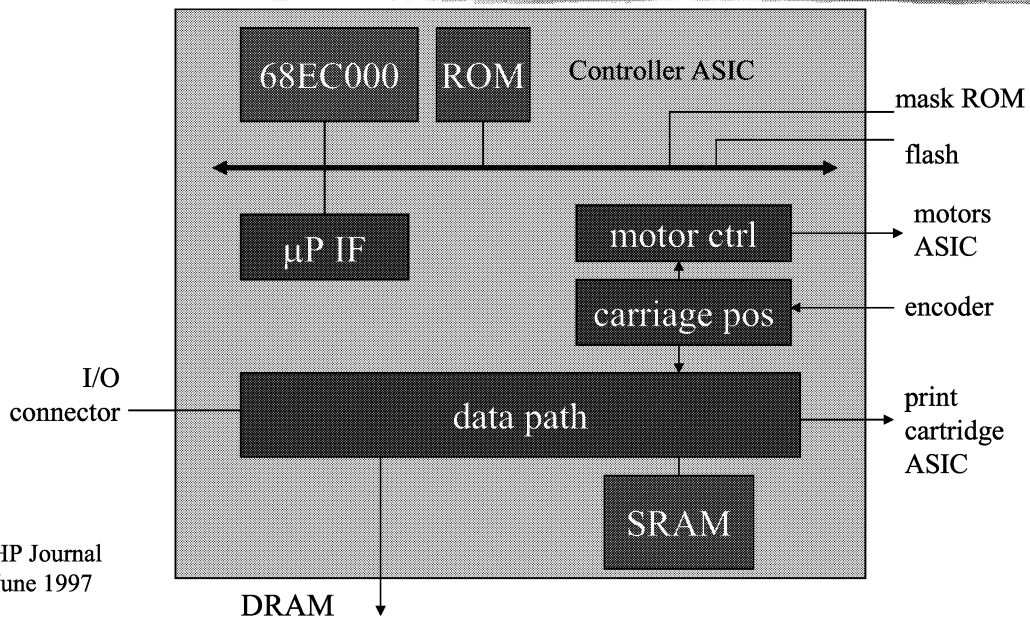
⌘ High-speed vs. low-cost:

- ☐ uniprocessor vs. multiprocessor hardware?
- ☐ multiprocessor requires more complex software.

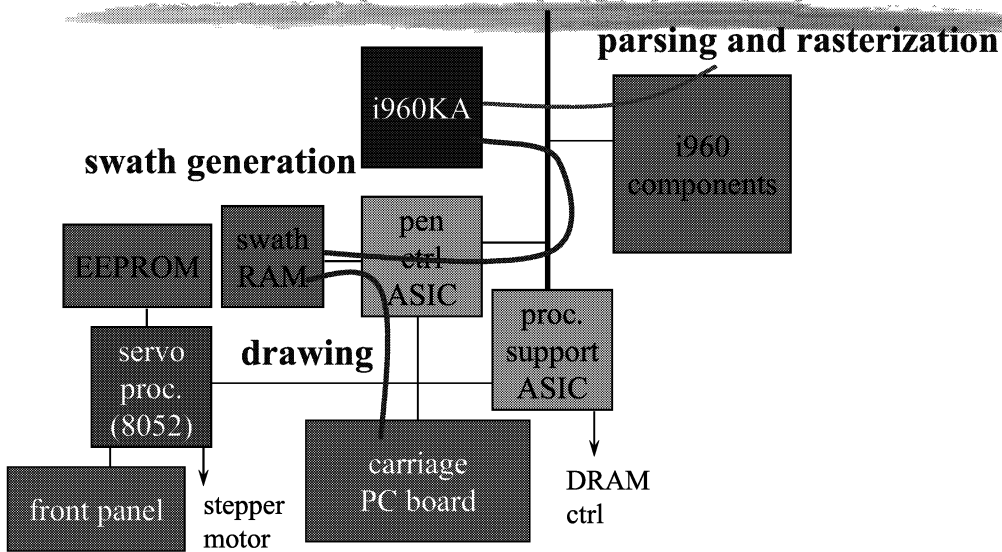
The printing process



HP PPA printer controller

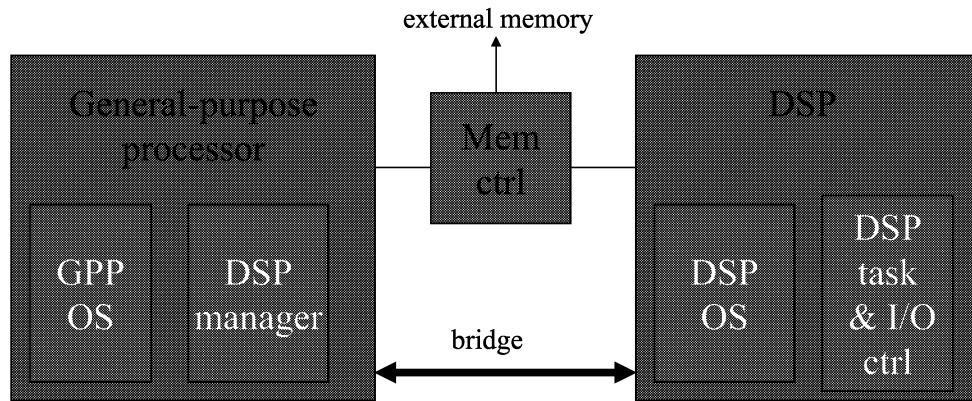


HP DesignJet hardware architecture



TI Open Multimedia Applications Platform

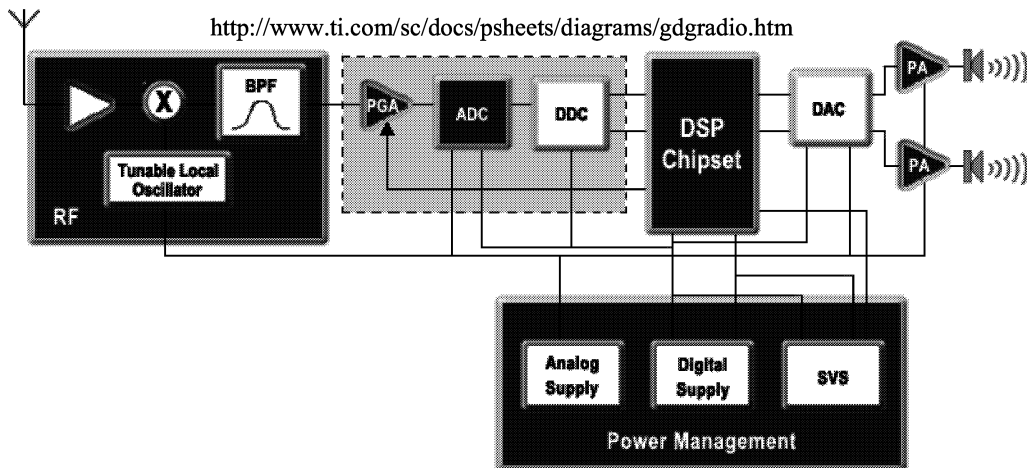
⌘ Dual-processor shared memory system:



<http://www.ti.com/sc/docs/apps/wireless/omap/overview.htm>

Wireless handset

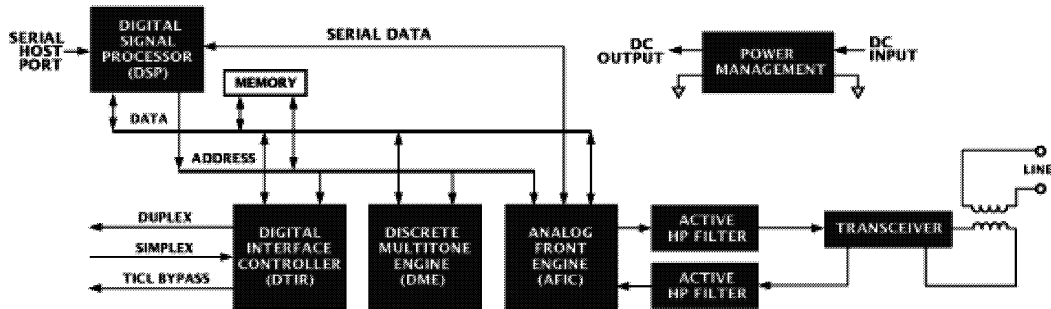
⌘ Generic handset (TI):



<http://www.ti.com/sc/docs/psheets/diagrams/gdgradio.htm>

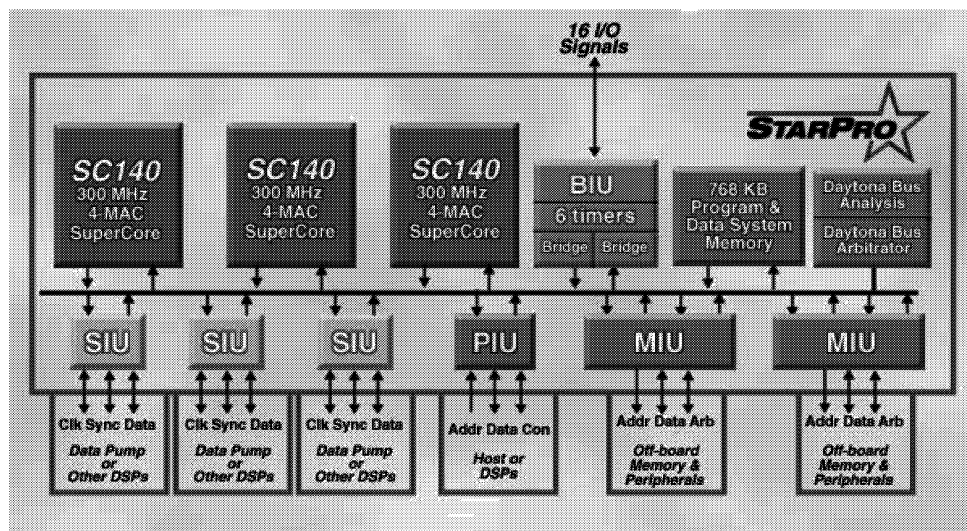
ADI ADSL engine

⌘ Bus-based multiprocessor (ADI):



http://www.analog.com/industry/signal_chains/auto/communications/comms_1.html

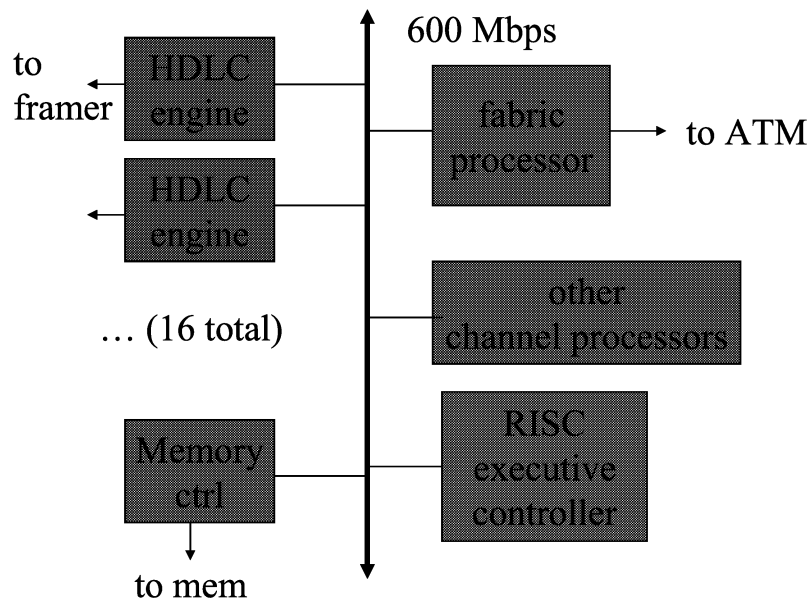
Agere StarPro platform



<http://www.lucent.com/micro/starpro/arch.html>

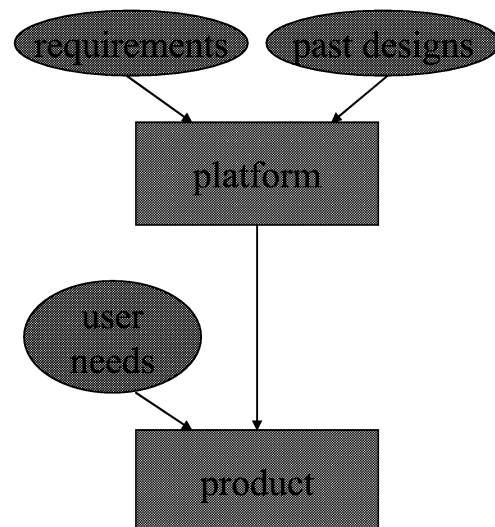
C-Port C5 network processor

<http://www.cportcorp.com/products/digital.htm>



Two phases of platform-based design

- ⌘ Design the platform.
- ⌘ Use the platform.



Division of labor

⌘ Platform design:

- ☒ choose, characterize hardware units;
- ☒ create the system architecture;
- ☒ optimize for performance, power.

⌘ Platform-based product design:

- ☒ modify hardware architecture;
- ☒ optimize programs.

Semiconductor vs. systems house

⌘ Semiconductor house designs the platform.

⌘ Systems house customizes the platform for its system:

- ☒ customization may be done in-house or by contractor.

Platform design challenges

- ⌘ Does it satisfy the application's basic requirements?
- ⌘ Is it sufficiently customizable? And in the right ways?
- ⌘ Is it cost-effective?
- ⌘ How long does it take to turn a platform into a product?

Platform design methodology

- ⌘ Size the problem.
 - ☐ How much horsepower? How much power?
- ⌘ Develop an initial architecture.
- ⌘ Evaluate for performance, power, etc.
- ⌘ Evaluate customizability.
- ⌘ Improve platform after each use.

Platform use challenges

- ⌘ How do I understand the platform's design?
- ⌘ How do I modify it to suit my needs?
- ⌘ How do I optimize for performance, power, etc.?

Platform use methodology

- ⌘ Start with reference design, evaluate differences required for your features.
- ⌘ Evaluate hardware changes.
- ⌘ Implement hardware and software changes in parallel.
- ⌘ Integrate and test.

Summary

- ⌘ Semiconductor houses are taking over system design functions.
- ⌘ Embedded multiprocessors solve a lot of problems.
 - ☒ But there is no one universal platform.

Hardware/Software Co-Design 1: Fundamentals and Beyond

Wayne Wolf
Dept. of Electrical Engineering
Princeton University

Outline

- Models
- Scheduling theory
- Tasks in co-synthesis
- Classic co-synthesis: hardware/software partitioning.
- Improvement to classic co-synthesis.
- Hot swapping.

What is co-synthesis?

- Use tools to simultaneously generate hardware architecture and the software which runs on it.
 - Architecture may contain programmable CPUs, ASICs.
- Different styles of co-synthesis use different levels of inputs, synthesis steps, types of results.

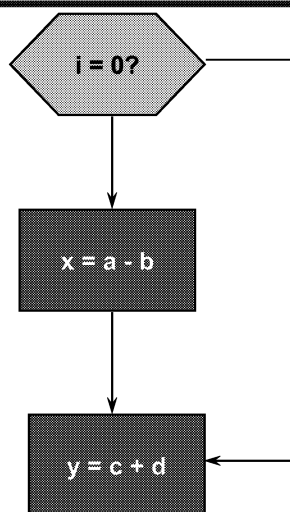
Uses for co-synthesis

- Design space exploration:
 - architecture planning;
 - planning of multi-generation products;
 - marketing planning.
- Prototype creation.
- Implementation generation.

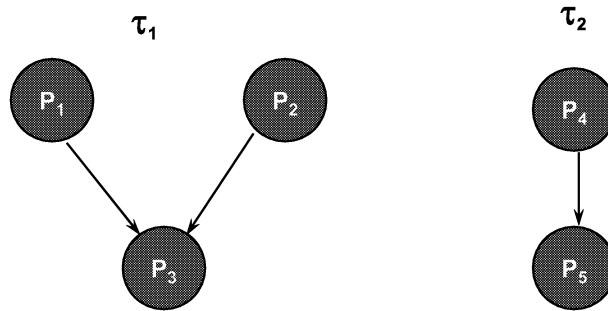
Terms

- thread, process: a single execution; thread usually implies no firewalls between threads
- PE: processing element, may be CPU, ASIC
- custom ASIC: designed on the fly
- catalog ASIC: characteristics known in advance

CDFG

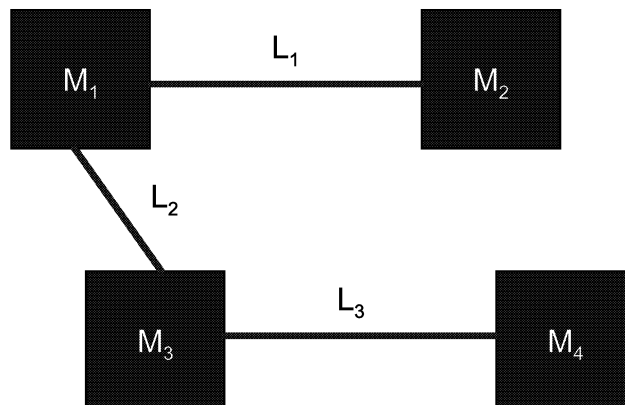


Task graph



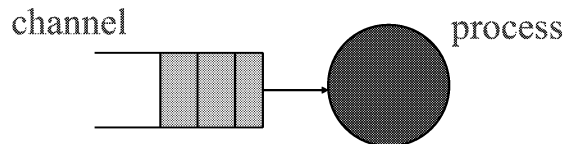
- Can model late arrivals, early departures by adding dummy processes.

Processor graph



Kahn process network

- Process has unbounded FIFO at each input:



- Each channel carries a possibly infinite sequence or stream.
- A process maps one or more input sequences to one or more output sequences.

Properties of processes

- Processes are usually required to be continuous: least upper boundedness can be moved across function boundary.
- Monotonicity:
 - $X \text{ in } X' \Rightarrow F(X) \text{ in } F(X')$

Networks of processes

- A network of processes relates the streams of several processes.
- If I = input sequences, X = internal sequences + outputs, then network behavior fixed point is
 - $X = F(X,I)$

Network properties

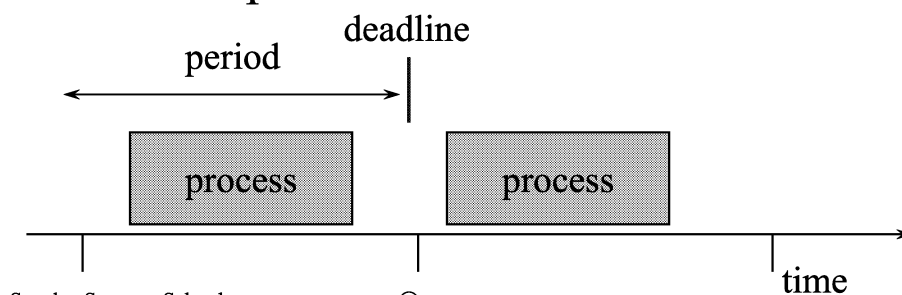
- A network of monotonic processes is a monotonic process.
 - Even in the presence of feedback loops.
- Can add nondeterminism in several ways:
 - allow process to test for emptiness;
 - allow process to be internally nondeterminate;
 - allow more than one process to consume data from a channel;

Processes as software objects

- A process is a unique execution of a program:
 - has its own state.
- A process may have a finite or infinite lifetime:
 - executed once or many times.
- A process may be executed periodically or aperiodically.

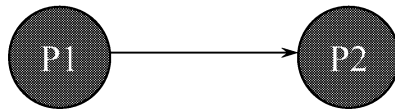
Process timing characteristics

- Period: interval between successive initiations.
- Deadline: time at which an initiated process must complete.



Process timing characteristics, cont'd

- Initiation time: time at which process goes into ready state. May come from:
 - external indeterminacy (jitter, etc.);
 - hidden data dependencies.



- Execution time: time required for process to complete assuming no interruptions.

Priority-driven scheduling

Each process is assigned a priority. Priority may be static or dynamic.

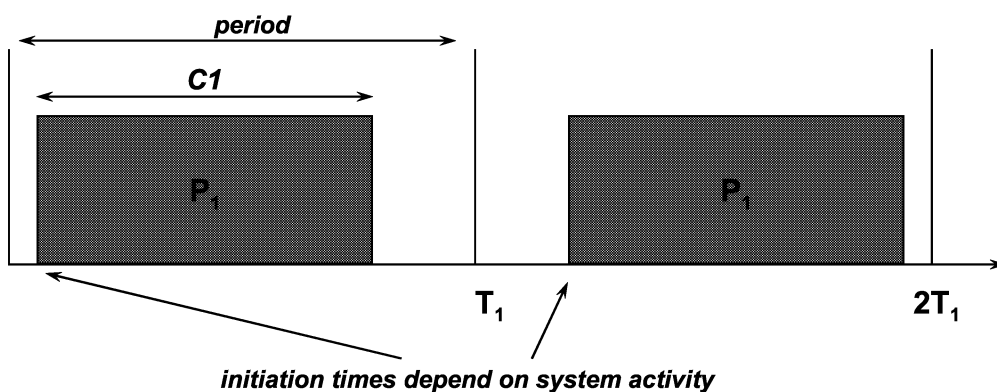
Highest-priority ready process becomes active.

A process runs until completion or preemption.

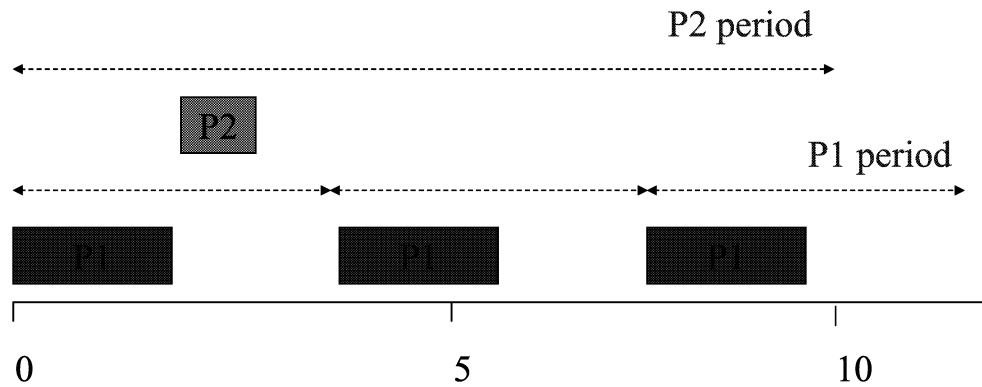
RMS model

- All processes run on one CPU.
- Static priority for each process.
- No data dependencies between processes.
- Must meet all deadlines for any valid combination of initiation times.
- Ignore context switch overhead.

RMS initiation times



RMS example

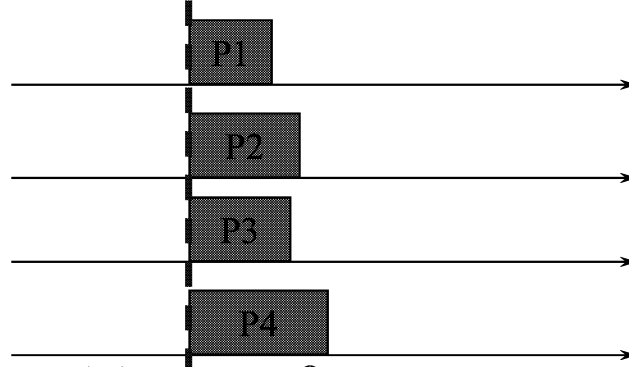


Rate monotonic analysis

- To meet deadlines, prioritize according to T_i 's, with shortest-period process given highest priority.
 - Fixed-priority scheme.
 - Independent of C_i 's.
- This priority assignment is optimal—no fixed priority scheme does better.

Critical instant

- The critical instant for a process occurs when the process and all higher-priority processes are initiated simultaneously.



CPU utilization under RMS

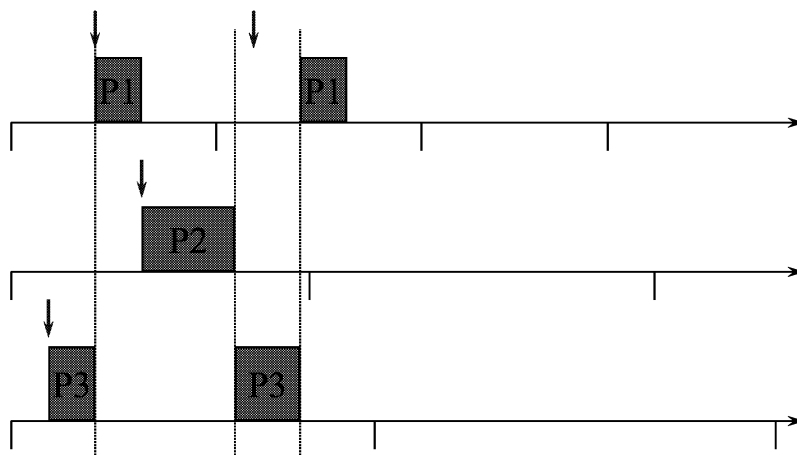
- Given task set of size m , utilization U :
 - $U = \sum_{1 \leq i \leq m} C_i/T_i$
- Least upper bound for utilization:
 - $U = m(2^{1/m} - 1)$
- Utilization asymptotically approaches 69%.
- Some CPU cycles cannot be utilized because the CPU must have enough cycles available to respond to the tightest critical instant for the highest-priority process.

Earliest-deadline first

Priority is dynamically calculated: process with deadline occurring soonest has highest priority.

Liu and Layland showed EDF can achieve 100% utilization until overload occurs. Cannot guarantee meeting deadlines for arbitrary data arrival times.

EDF example



Schedule unrolling

To prove that a complex scheduling algorithm works, must unroll the schedule to the least-common multiple of all the period times.

MPEG-1 example: 44.1 kHz x 30 Hz requires 132,300 time intervals.

After unrolling, must consider arrival time combinations.

Priority inversion

Problem arises when processes can request outside resources: bus, shared variable, etc.

Prioritization means that low-priority process can keep high-priority process from getting the resource, causing deadlock.

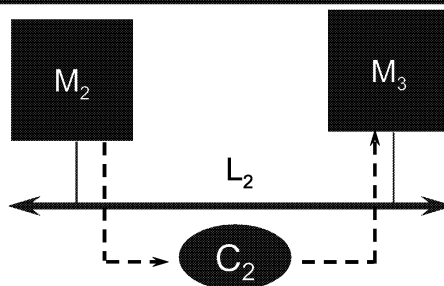
Priority inversion example

Deadlock scenario:

- low-priority process L gets resource;
- preempted by higher-priority process H;
- H requests same resource;
- H can't get resource until L finishes, but H won't let L continue.

Can be solved by priority inheritance: L gets higher priority while it has the resource.

Bus communication example



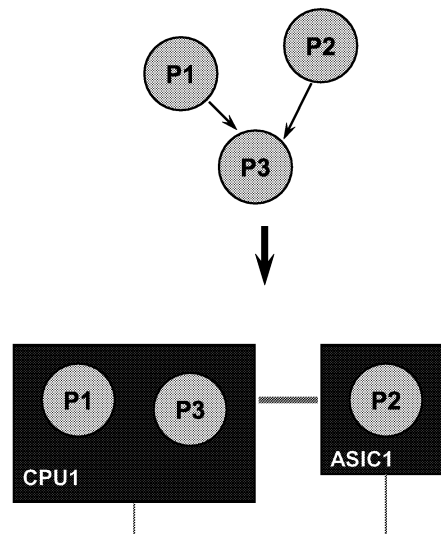
- Communication requires two resources: PE and channel.
- Long messages can hog communication resources.
- Bus communication must be prioritized.

Synthesis tasks

- *Scheduling*: make sure that data is available when it is needed.
- *Allocation*: make sure that processes don't compete for the PE.
- *Partitioning*: break operations into separate processes to increase parallelism, put serial operations in one process to reduce communication.
- *Mapping*: take PE, communication link characteristics into account.

Scheduling and allocation

- Must schedule/allocate
 - computation
 - communication
- Performance may vary greatly with allocation choice.

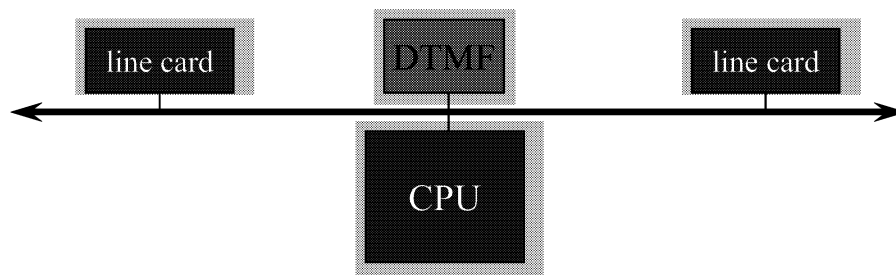


Problems in scheduling/allocation

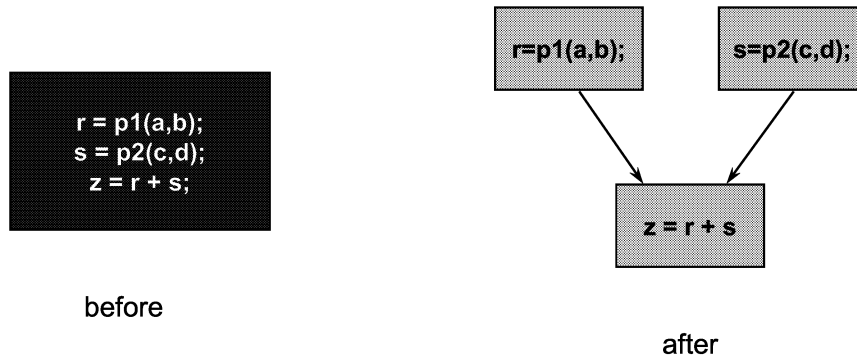
- Can multiple processes execute concurrently?
- Is the performance granularity of available components fine enough to allow efficient search of the solution space?
- Do computation and communication requirements conflict?
- How accurately can we estimate performance?
 - software
 - custom ASICs

TigerSwitch: allocation of DTMF

- Where do we put TouchTone (DTMF) detection?



Partitioning example

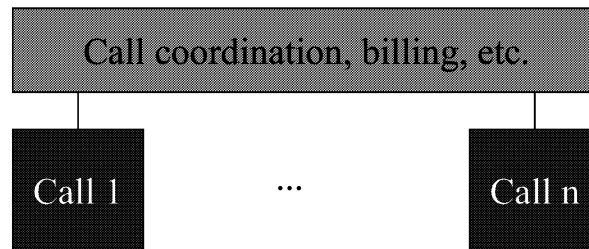


Problems in partitioning

- At what level of granularity must partitioning be performed?
- How well can you partition the system without an allocation?
- How does communication overhead figure into partitioning?

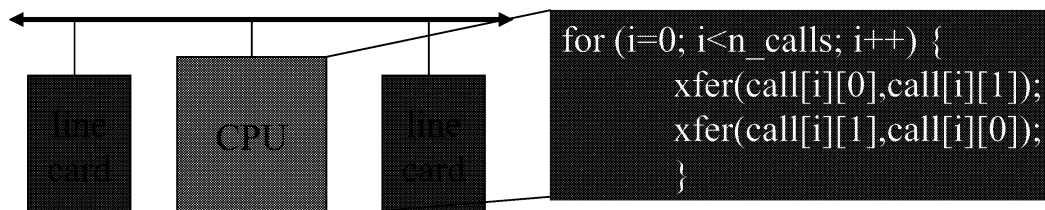
Tigerswitch partitioning problem: before

- Calls are parallel processes:



Tigerswitch partitioning problem: after

- Implement calls as array elements:

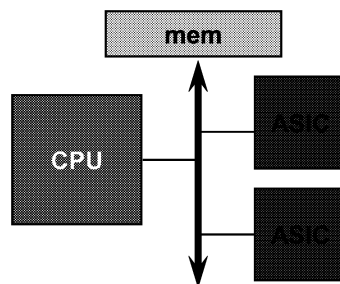


Problems in mapping

- Mapping and allocation are strongly connected when the components vary widely in performance.
- Software performance depends on bus configuration as well as CPU type.
- Mappings of PEs and communication links are closely related.

Hardware-software partitioning

Architectural template: CPU + 1 or more ASICs on a bus:



Properties of classic partitioning algorithms

- Single-rate.
- Single-threaded: CPU waits for ASIC.
- Type of CPU is known; ASIC is synthesized. Closely coupled to high-level synthesis.

Hardware/software partitioning styles

- Two major styles:
- Vulcan starts with all-ASIC solution and moves functions to software to reduce cost (primal method).
- COSYMA starts with all-software solution and moves functions to ASIC to meet performance goal (dual method).

Vulcan

- Gupta and De Micheli: Target architecture: CPU + ASICs on bus
- Break behavior into threads at nondeterministic delay points; delay of thread is bounded
- Software threads run under RTOS; threads communicate via queues

Specification and modeling

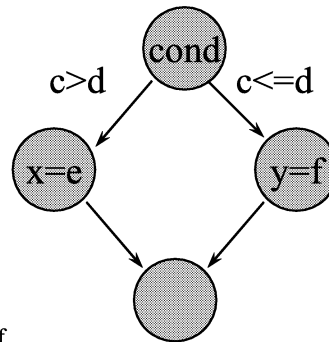
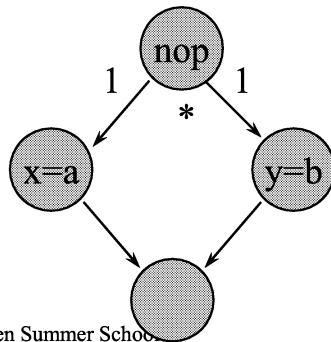
- Specified in Hardware C. Spec divided into threads at non-deterministic delay points.
- Hardware properties: size, # clock cycles.
- CPU/software thread properties:
 - thread latency
 - thread reaction rate
 - processor utilization
 - bus utilization

Vulcan thread modeling

Hardware C allows conjunctive, disjunctive execution:

conjunctive: $x=a; y=b;$

disjunctive: $\text{if } (c>d) x=e; \text{ else } y=f;$



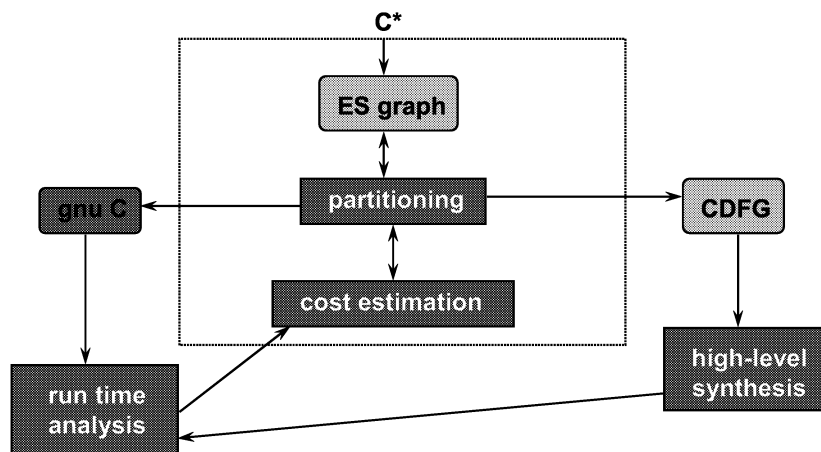
HW/SW allocation

- Start with unbounded-delay threads in CPU, rest of threads in ASIC.
- Optimization:
 - test one thread for move
 - if move to SW does not violate performance requirement, move the thread
 - feasibility depends on SW, HW run times, bus utilization
 - if thread is moved, immediately try moving its successor threads

COSYMA

- Ernst et al.: moves operations from software to hardware.
- Operations are moved to hardware in units of basic blocks.
- Estimates communication overhead based on bus operations and register allocation.
- Hardware and software communicate by shared memory.

COSYMA design flow



Cost estimation

- Speedup estimate for basic block b :
 - $\Delta c(b) = w(t_{HW}(b) - t_{SW}(b) + t_{com}(Z) - t_{com}(Z + b)) * It(b)$
 - $w = \text{weight}$, $It(b) = \# \text{ iterations taken on } b$
- Sources of estimates:
 - Software execution time (t_{SW}) is estimated from source code.
 - Hardware execution time (t_{HW}) is estimated by list scheduling.
 - Communiation time (t_{com}) is estimated by data flow analysis of adjacent basic blocks.

COSYMA optimization

- Goal: satisfy execution time. User specifies maximum number of function units in co-processor.
- Start with all basic blocks in software.
- Estimate potential speedup in moving a basic block to software using execution profiling.
- Search using simulated annealing. Impose high cost penalty for solutions that don't meet execution time.

Two-phase optimization

- Inner loop uses estimates to search through design space quickly.
- Outer loop uses detailed measurements to check validity of inner loop assumptions:
 - code is compiled and measured
 - ASIC is synthesized
- Results of detailed estimate are used to apply correction to current solution for next run of inner loop.

SOS ILP formulation

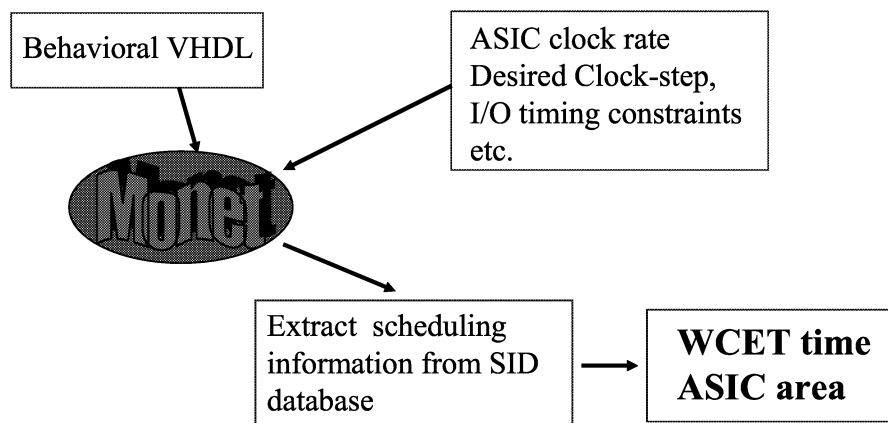
- Prakash and Parker:
 - variables represent schedule;
 - constraints represent process dataflow;
 - 0/1 variables represent allocation.
- Minimize system cost, satisfy performance requirement.

Co-synthesis and ASIC performance optimization

- Need a more accurate model of ASIC performance.
 - Use high-level synthesis in the loop to estimate.

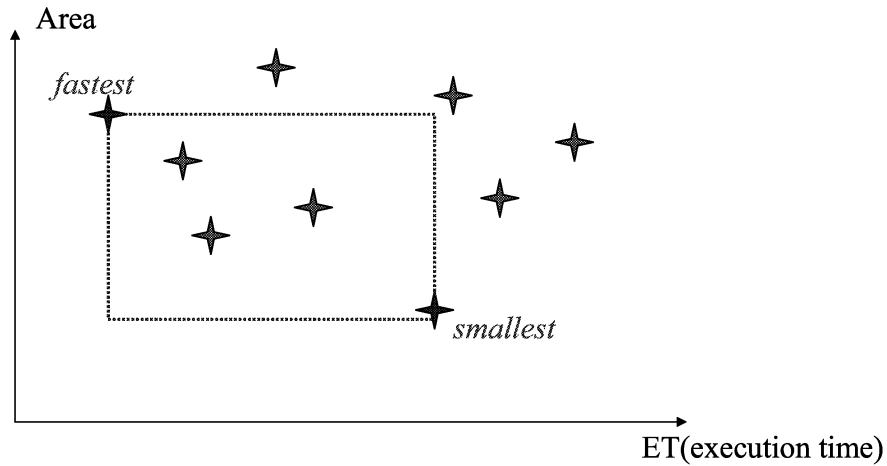
ASIC performance analysis

❖ **Monet**: a behavior-level *architectural exploration system* by Mentor Graphics, take behavioral description in VHDL or Verilog as input, output RTL description



ASIC performance analysis

❖ ASIC implementation design space

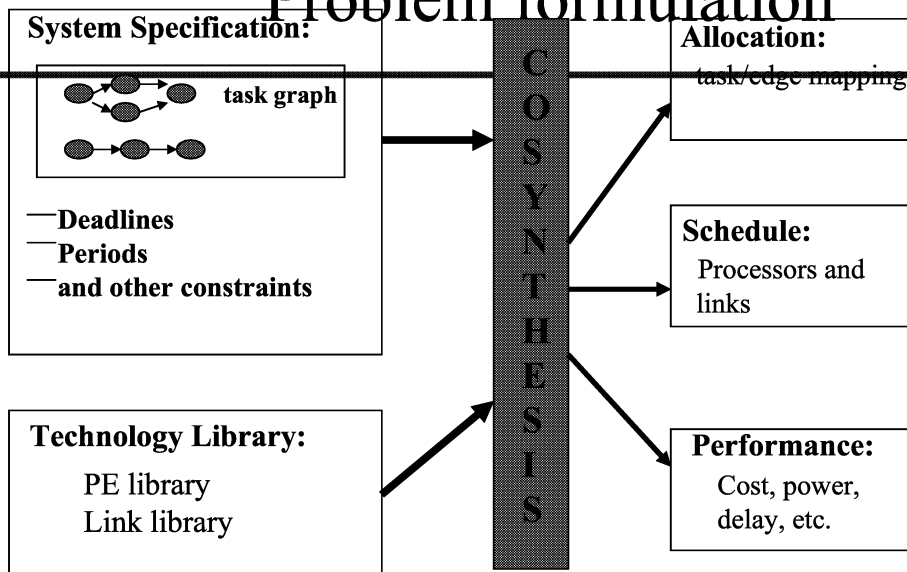


Find out 2 extremes and search intermediate solutions between these 2 extremes.

Sweden Summer School

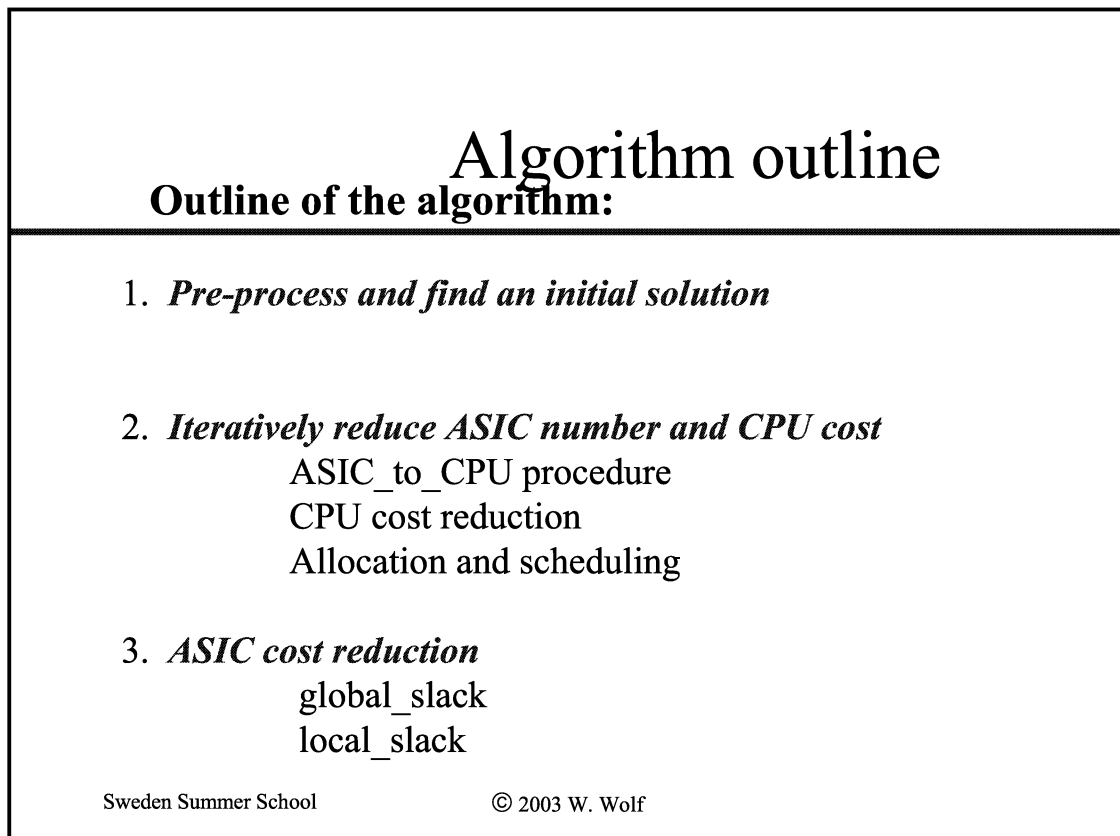
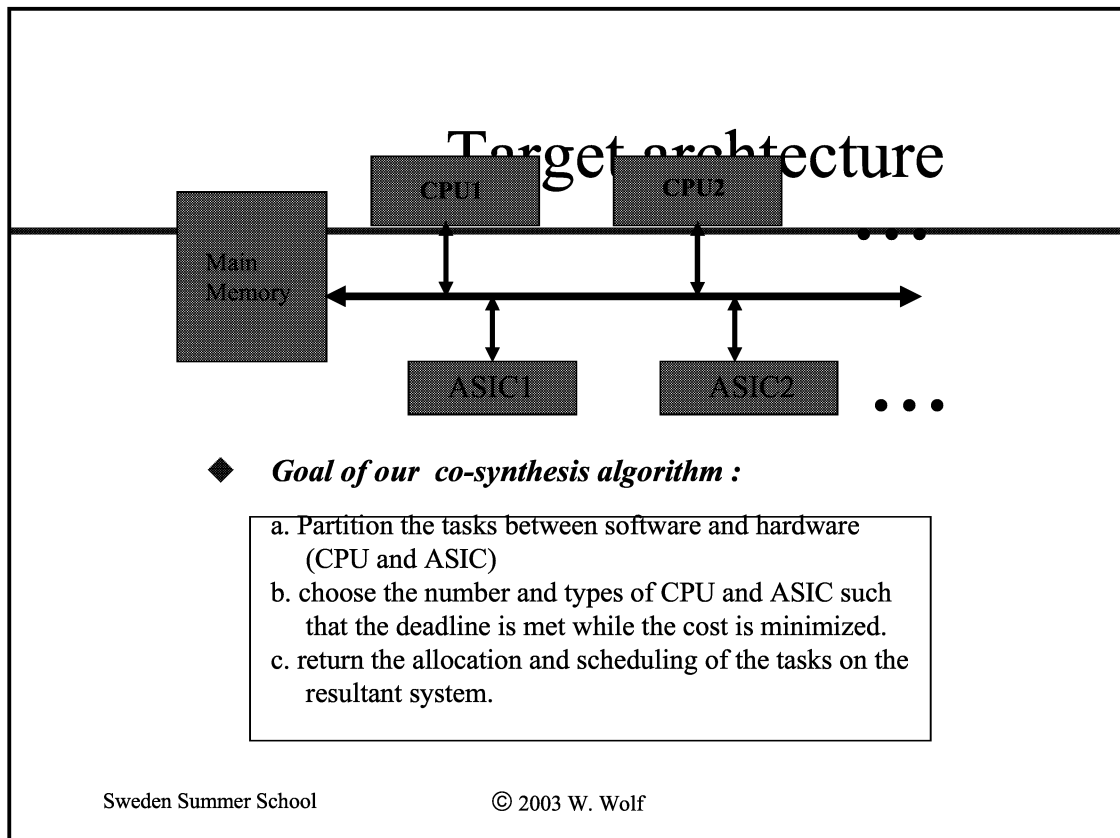
© 2003 W. Wolf

Problem formulation



Sweden Summer School

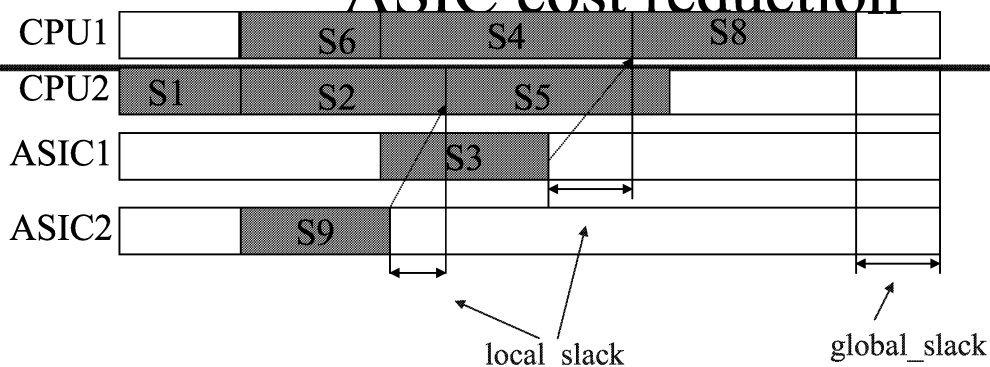
© 2003 W. Wolf



Iterative improvement

1. ASIC to CPU: 2 heuristics to move processes from ASIC to CPU
 - a. Difference of ASIC_ET and CPU_ET
 - b. Non-Critical path ASIC
2. CPU cost reduction: Reduce the existing CPU architecture cost.
 - a. reduce the number of CPU
 - b. replace the expensive CPU with a cheaper one
 - c. reduce cache cost
3. Allocation and scheduling
 - a. static urgency
 - b. dynamic urgency

ASIC cost reduction



$$\mathit{global_slack} = \text{Min}(\text{deadline} - \text{completion_time})$$

$$\mathit{local_slack} = \text{Min}(\text{start_time_of_successors_nodes}) - \text{completion_time_ASIC_j}$$

ASIC cost reduction (2)

1. Sort ASICs by *decreasing cost difference* $D = \text{cost}_{of_fastest} - \text{cost}_{of_smallest}$

2. For each ASIC_i in sorted list

replace the fastest ASIC with the smallest ASIC

if (meet deadline) use the smallest ASIC

else keep the fastest

3. Reduce the fastest ASIC cost:

for each ASIC_j{

a Set $ASIC_j_ET = (\text{fastest_ET} + \text{global_slack} + \text{local_slack}_j)$

b foreach ASIC_i in other ASICs

{ calculate $local_slack_i$ and

set $ASIC_i_ET = \text{fastest_ET} + \text{local_slack}_i$ }

c call ASIC analysis tool to get the total ASIC cost $COST(i)$ }

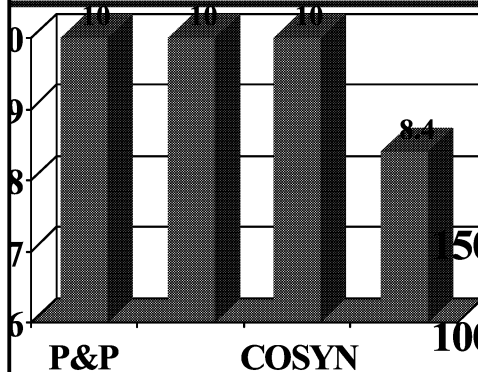
4. Select the minimal $COST(i)$ in step 3 and set corresponding ASIC_{ET}.

Sweden Summer School

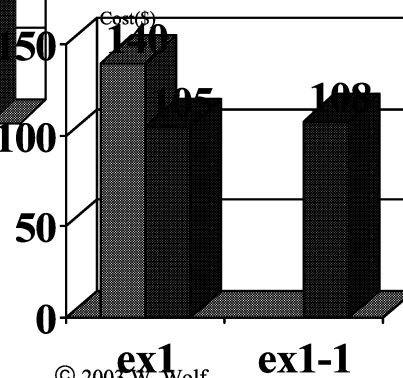
© 2003 W. Wolf

Results

PP9 example (ASICosyn uses ASICs)



ex1 example (Li/Wolf)



Sweden Summer School

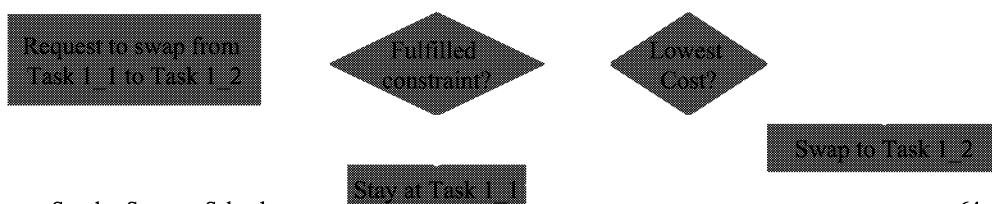
© 2003 W. Wolf

Characterization 1

- Lee/Henkel/Wolf: A certain task may have more than one implementation
- Different implementations of a task typically have different power consumption
- Swapping from one implementation of a certain task to another implementation of the same task implies an additional computation time and power consumption

Characterization 2

- The basic functionality of the various implementations of a certain task is the same. At least one implementation is a superset of the other implementations
- Swapping can be triggered by two requirements: **constraints** and **cost minimization**

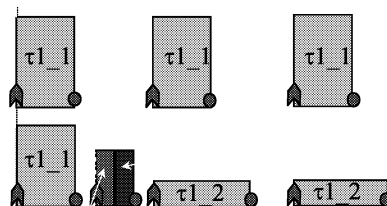


Goal

- It is unlikely that all the most costly implementations of all tasks are scheduled to run at the same time since the requirements for swapping are different for different tasks
- Achieve an advantage using hot swapping as opposed to a system that has only one fixed implementation of a task which is typically the most costly one (in order to cover the worst case)

Cost example

- RMS with Hot Swap
- Result: energy saving



Scheduling equations

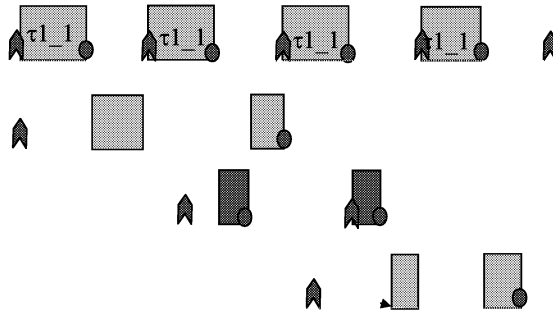
$$t_{1b} = \sum_{\forall \tau_j \in \{hp(i) \cup \tau_i\}} C_j \cdot \left\lceil \frac{t_{request} + 1}{T_j} \right\rceil$$
$$\vdots$$
$$t_{nb} = \sum_{\forall \tau_j \in \{hp(i) \cup \tau_i\}} C_j \cdot \left\lceil \frac{t_{nb-1} + 1}{T_j} \right\rceil$$

Feasibility test

- Case 1
 - Task τ_i has not yet started execution. Swapping is potentially possible if there is enough time left for conducting the swap (will be discussed later).
- Case 2
 - Task τ_i has already started execution. This case tells us that the feasibility test from above is not sufficient. We have to apply a second feasibility analysis

Feasibility test example

- Swapping task 2 from implementation 1 to 2



Scheduling Strategy 1

- *When* is it safe to swap
- *When* to actually perform the swap (assumed that there has already been initiated a request to swap a.s.a.p.)
- *How* to swap (e.g. using intermediate swaps etc.)

Scheduling Strategy 2

- ***At Design Time***
 - *Create database*
 - *Generate parameters*
- ***At Run time***
 - *Perform feasibility test*
 - *Decide to swap or not*
 - *Decide which implementation to swap*

Summary

- Co-design is one branch of embedded system design.
- Models include functionality, HW and SW targets, performance.
- Scheduling real-time embedded systems is hard.
- Classic co-synthesis partitions functionality onto a CPU+ASIC template.

Hardware/Software Co-Design 2: Techniques

Wayne Wolf
Dept. of Electrical Engineering
Princeton University

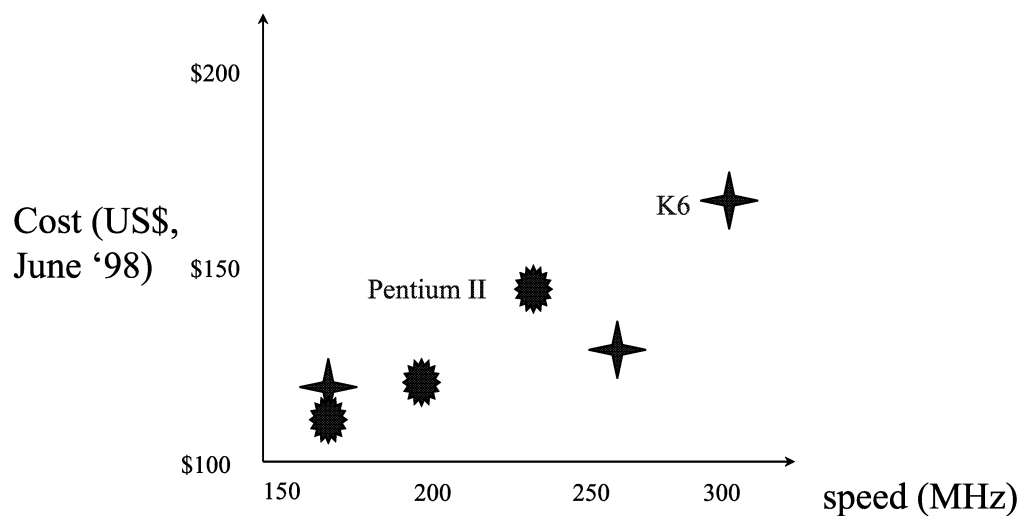
Outline

- Distributed system co-synthesis.
- Communication.
- Reactive system co-synthesis.
- Low-power design.
- Memory systems.

Why distributed systems?

- CPU cost is a non-linear function of performance.
 - Several small CPUs may be cheaper than one big one.
- Scheduling overhead must be paid for at the non-linear rate.

CPU cost vs. performance



Distributed system co-synthesis

- Can't take advantage of architectural template:
 - structure;
 - component characteristics.
- Generally multi-rate.

GCLP algorithm

Kalavade and Lee: global criticality/local phase; iterative algorithm.

Global criticality: critical path used to identify nodes to move onto ASIC.

Local phase: identify nodes which can be much more cheaply implemented in one medium than the other to reduce cost.

Successive-refinement co-synthesis

- Wolf: scheduling, allocation, and mapping are intertwined:
 - process execution time depends on CPU type selection
 - scheduling depends on process execution times
 - process allocation depends on scheduling
 - CPU type selection depends on feasibility of scheduling
- Solution: allocate and map conservatively to meet deadlines, then re-synthesize to reduce implementation cost.

A heuristic algorithm

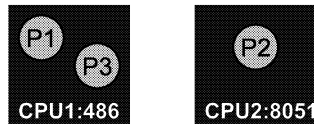
1. Allocate processes to CPUs and select CPU types to meet all deadlines.
2. Schedule processes based on current CPU type selection; analyze utilization.
3. Reallocate processes to CPUs to reduce cost.
4. Reallocate again to minimize inter-CPU communication.
5. Allocate communication channels to minimize cost.
6. Allocate devices, to internal CPU devices if possible.

Example

1—allocate and map for deadlines:



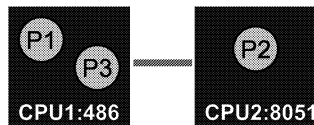
3—reallocate for cost:



4—reallocate for communication:



5—allocate communication:



PE cost reduction step

- Step 3 contributes most to minimizing implementation cost. Want to eliminate unnecessary PEs.
- Iterative cost reduction:
 - reallocate all processes in one PE;
 - pairwise merge PEs;
 - balance load in system.
- Repeat until system cost is not reduced.

Technology mapping: interrupt allocation

- Rhodes and Wolf: assign limited interrupts to processes:
 - **Allocate** each process to a resource (processor);
 - Determine the **priority** of each process;
 - Decide which arcs should be interrupt driven and which polled, subject to a limit per resource.

Analysis model used in Design

- POLLED data arrives “silently”
- INTERRUPT driven arc requires interrupt handling
 - Int. handlers assumed to have higher priority than any process
 - Interrupts are turned off while in an interrupt handler
 - Turned back on at conclusion
- **No context switch overhead**

Communication and reactive systems

- Communication is specified as systems of communicating finite-state machines.
- Finite-state machine model is important: determines what designer must write and what can be synthesized.

SOLAR/PARTIF

- Ismail et al: partitioning and refinement for communication-rich systems.
- Specification in terms of Design Units, written in StateChart style. Design Unit can be composed of Design Units and Communication Units. State Table can be used to specify a Design Unit.
- Three types of architectural components: hardware, software, communication. Units can be connected in various topologies.

SOLAR design process

- System is specified as a set of Design Units.
- DUs are assigned into hardware and software, but not allocated to particular units.
- Communication channels are allocated and bound.
- Design units are bound to particular architectural components.

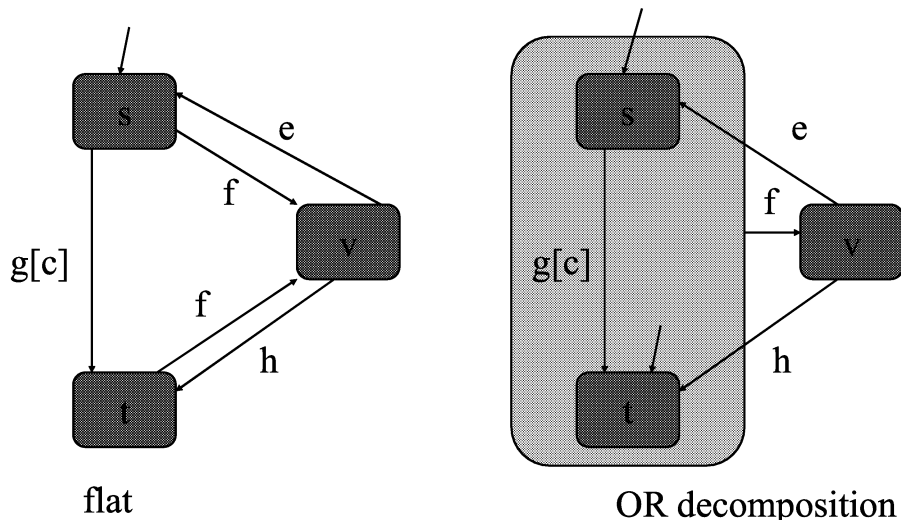
PARTIF

- Interactive partitioning tool. Provides design statistics to guide partitioning moves.
- Basic transformations:
 - move—transfer one process to another point in the hierarchy
 - merge—fuse two processes into one
 - split—turn a sequential machine into parallel machines
 - cut—cut a set of parallel states into interconnected Design Units
 - map—transform communicating systems into interconnected subsystems

Statecharts

- Harel: Early control-oriented specification model for reactive systems.
- Commercialized in the Statemate system.
- Provided hierarchical model for states.

Statechart OR state



Esterel

- Berry: language for reactive real-time systems.
- Synchronous specification:
- Assumption: zero-time communication.
- Implement by forming product machine.

TOSCA

Antoniazzi et al: targets control-dominated systems.

System modeled as communicating machines.

Use operators to explore design space:

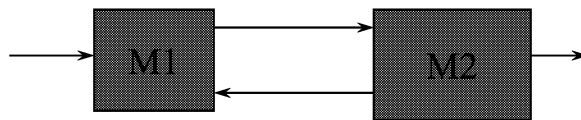
- unfold entry/exit;
- unfold FSM enabling condition;
- flatten hierarchy;
- replace timers with counters;
- collapse several processes into one.

POLIS

Sangiovanni-Vincentelli et al: targets multi-rate reactive systems.

System modeled as network of Co-design FSMs (CFSMs).

Uses zero-delay hypothesis: communication happens instantaneously.



Polis, cont'd

Communication can be analyzed by forming product of communicating machines.

Partitioning assigns functions to hardware and software.

Library components can be described as CFSMs.

Performance analysis for reactive systems

Balarin and S-V: validate schedule for reactive system.

Specification is DAG of tasks with priorities and execution times.

Events initiate computation. For every critical event from i to j , min time between 2 executions of i must be \geq max time between execution of i and j .

Compute partial loads to find answer.

Chinook

Borriello et al: targets control-dominated systems.

Major steps:

- HW/SW partitioning;
- device driver synthesis and low-level scheduling;
- I/O port allocation and interface synthesis;
- system-level scheduling;
- code generation.

Power analysis and optimization

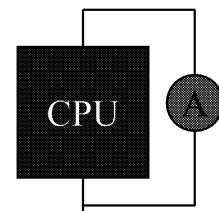
- Must estimate power/energy requirements:
 - multi-threaded
 - caches
- Are there tradeoffs between power and performance?

Single-process power analysis

Tiwari and Malik: measure average current for instructions.

First-order model gives energy per instruction.

Second-order model takes into account inter-instruction interactions.



TOSCA power estimation metrics

Types of systems and metrics:

- area-constrained: power*area
- performance-constrained
 - fixed-throughput: energy per operation
 - maximum throughput: energy per op/max throughput = Power/T^2
 - burst throughput: $(E_{\max} + E_{\text{idle}})/T_{\max}$

TOSCA power estimation

- Hardware:
 - IO section
 - data path
 - memory
 - control: primary inputs, state, combinational, outputs
- Software:
 - based on average current per instruction

Low-power synthesis

Kirovski and Potkonjak: synthesis concentrates on processor allocation and task assignment.

Architectural template: multiple PEs on common bus with common memory.

Uses mixed strategy: power consumption per process + voltage scaling.

Low power synthesis, cont'd

First step: allocate processors in template for minimum power for the task set:

- eliminate inferior PEs: more expensive, slower, or uses more energy
- iteratively improve to final configuration

Second step: assign tasks to PEs in the instantiated architecture.

Policy optimization

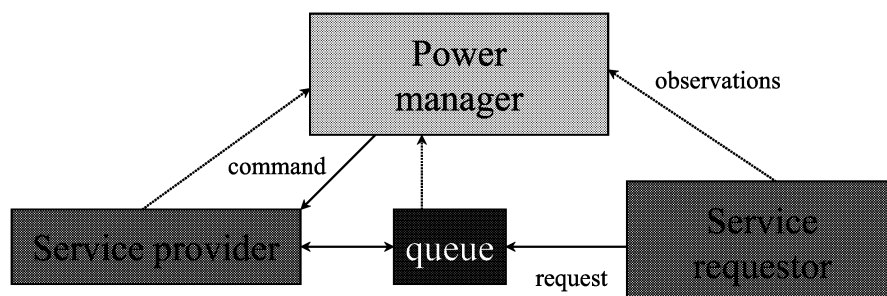
Paleologo et al: describe power management using stochastic FSMs.

Power management policy determines how power is managed.

Finding optimal policy to maximize average performance can be solved in polynomial time.

Stochastic model

Each component is a Markov chain:



Stochastic model components

Sample Markov chain for service requestor:

0 1
PSR = 0 | 0.9 0.1 |
1 | 0.2 0.8 | ← Probability of two requests in a row

Policy is the sequence of states taken by the power manager.

Optimal policy is stationary: depends only on current state of the system.

Memory is important

- Relative energy per operation (Cathoor et al):
 - memory transfer: 33
 - external I/O: 10
 - SRAM write: 9
 - SRAM read: 4.4
 - multiply: 3.6
 - add: 1

The cache's sweet spot

- Energy consumption has a sweet spot as cache size changes:
 - cache too small: program thrashes, burning energy on external memory accesses;
 - cache too large: cache itself burns too much power.

Energy dissipation for embedded systems

Li and Henkel: concentrates on power consumption in memory hierarchy.

Memory models for cache and main memory:

- array
- peripheral logic

Software energy and performance model

Based on behavioral simulation.

Total energy components:

- energy per instruction * instructions
- data cache write miss penalty
- data read miss
- instruction fetch miss

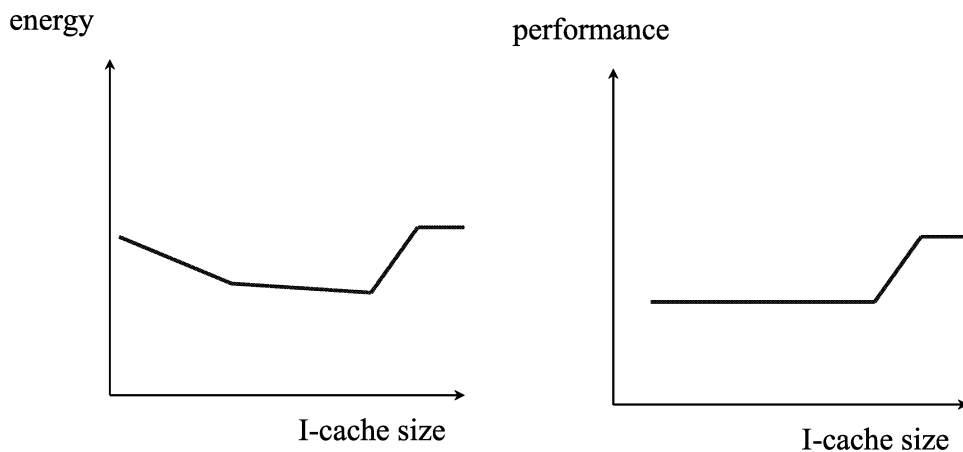
Design space exploration strategy

Software transformations:

- procedure in-lining
- loop unrolling

Cache may be made larger to increase performance at cost of higher energy utilization.

MPEG example



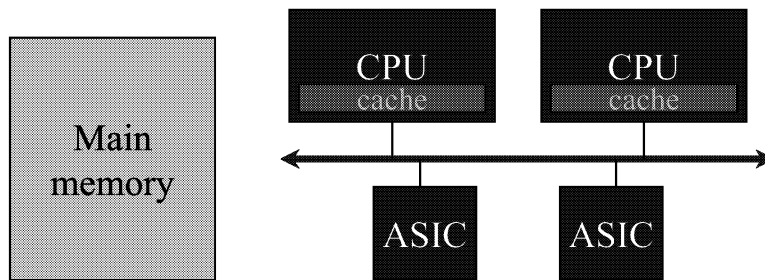
Memory system methodology

- Catthoor et al:
 - optimize storage and transfer between components;
 - then merge components and apply optimizations to global model;
 - use loop transformations, reordering of computations, data type refinement, etc.

Co-synthesis with memory hierarchies

Li and Wolf: use hierarchical memory model in co-synthesis system.

Target architecture:



System architecture

Given processes and characteristics, PE technology information

Find:

- schedule for processes and allocation to PEs;
- cache sizes and allocation of processes to cache.

Two phases: parameter extraction for processes; synthesis.

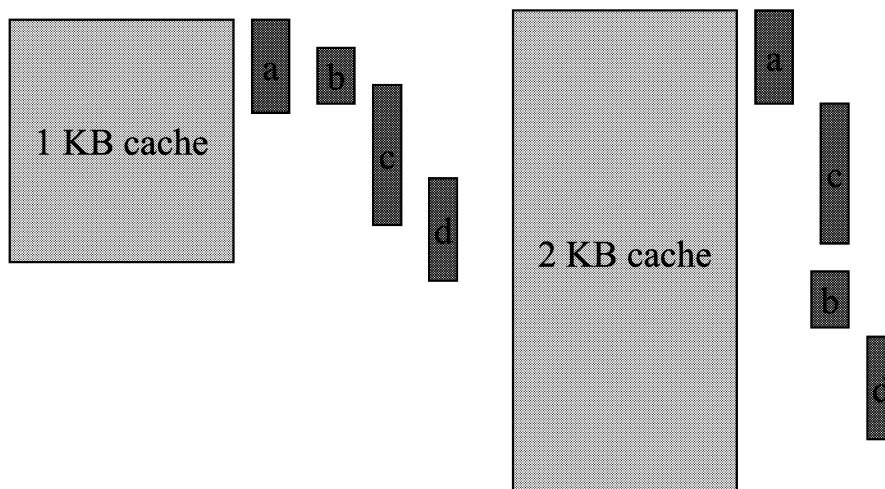
Cache assumptions

Only one level of cache; each process is well-behaved in that cache.

Caches are direct mapped; cache sizes are powers of two.

A task is mapped into a contiguous region of instruction cache.

Mapping of tasks onto cache



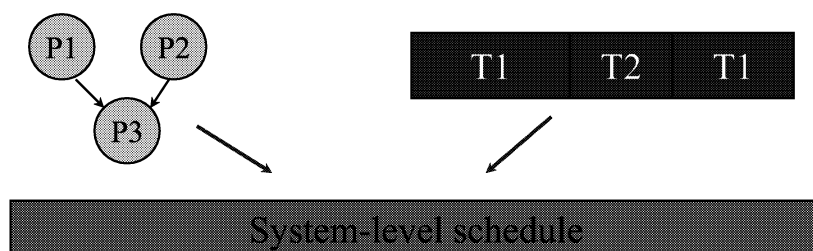
Process characterization

- Characterize each process independently:
 - trace-based analysis;
 - identify cache footprint.
- Process parameters:
 - worst-case execution time;
 - best-case execution time;
 - average-case execution time.

Task allocation and scheduling

Based on hierarchical scheduling algorithm:

- schedule each task independently;
- schedule system with task as atomic unit;
- instantiate tasks and optimize system schedule.



Results

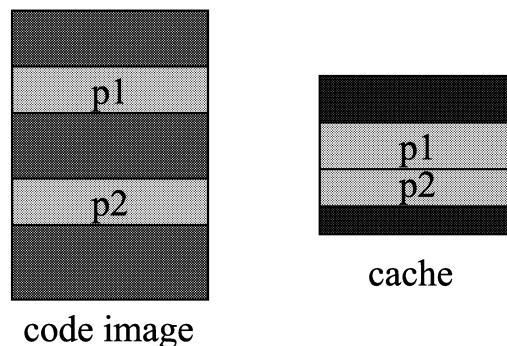
MPEG-2 encoder:

- without cache: 6 PEs, 121 CPU seconds
- fixed caches: 5 PEs, 157 CPU seconds
- synthesized caches: 4 PEs, 203 CPU seconds

Most CPU time goes into parameter evaluation, but this is outside synthesis loop and may be amortized across multiple designs.

Code placement

- Potkonjak et al:
 - place code to minimize dynamic cache interference through placement and scheduling.



Sponsored by:



Cycle-Accurate Joulemeter for CMOS VLSI Circuits

Soo-Ik Chae

Seoul National University, Korea



Cycle-Accurate Joulemeter for CMOS VLSI Circuits

Soo-Ik Chae
Seoul National University
chae@sdgroup.snu.ac.kr
2003. 8. 11



Contents

- Introduction
- CMOS Inverter Model
- Energy Estimation in Measurement System
 - Energy model 1
 - Energy model 2
 - Energy model 3
 - Energy model 4
 - Capacitance ratio constraint
- Second Order Effects
 - Leakage current
 - Overlapping current
 - Nonlinear capacitance
- Experimental Results
- Conclusion



Motivation for Low-Power Design

- Scaling of CMOS technology**
 - Higher functionality with smaller chips
 - Higher performance at lower cost
- Cool chips and cool systems**
 - Low-power design may ease the heat dissipation problem
- Recent portable compute-intensive applications**
 - Multimedia
 - Video display and capture
 - Notebook computer
 - Personal communication systems
 - Implantable medical electronics



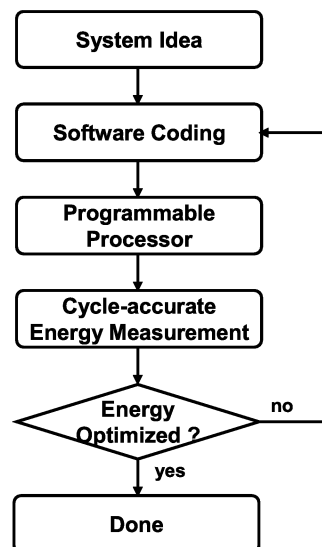
Power/Energy Estimation Techniques

- Low level simulators**
 - Precise estimation
 - Large amount of simulation time due to the increase of circuit complexity
- High level energy simulators**
 - Fast simulation time
 - Lower accuracy
- Measurement-based energy estimators**
 - Digital multimeters, wattmeters: only can measure average or rms value
 - Oscilloscope: little practical aspects in terms of time complexity and accuracy
- Need for cycle-accurate power/energy estimator**

Need for Cycle-Accurate Power/Energy Estimator

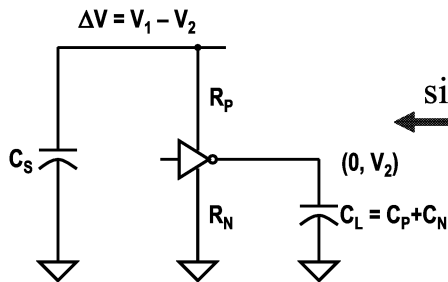
- ❑ Most digital systems consume very different amounts of energy for each cycle
- ❑ Need for cycle-accurate energy consumption profiles
- ❑ Software level energy optimization can be done
- ❑ Changing the energy sensitive factors while preserving the semantics of the original design

Measurement-Based Embedded Software Optimization



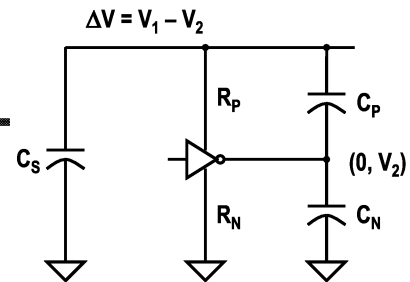
Energy Flow during a Transition of a CMOS Inverter

Inv Model A

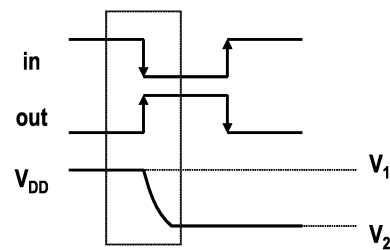
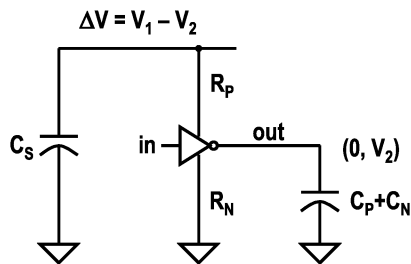


simplification

Inv Model B



Inv Model A: when the output is rising



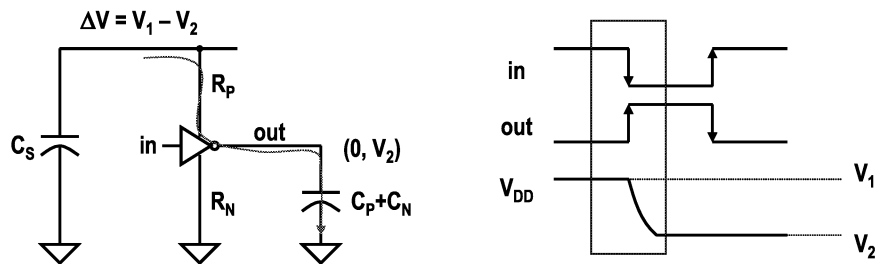
- Before transition,

$$Q_1 = C_S V_1 \quad E_1 = \frac{1}{2} C_S V_1^2$$

- After transition,

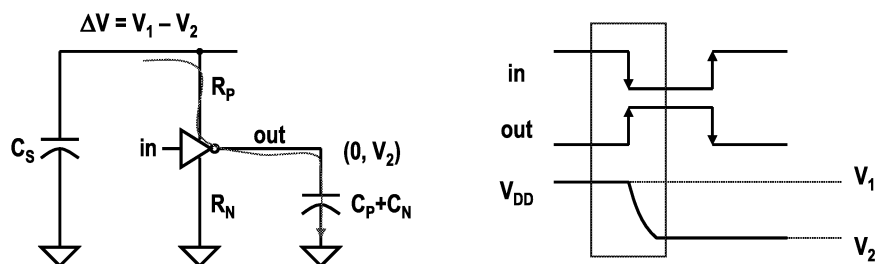
$$Q_2 = (C_S + C_P + C_N) V_2 \quad E_2 = \frac{1}{2} (C_S + C_P + C_N) V_2^2$$

Inv Model A: when the output is rising



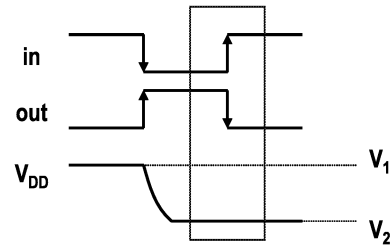
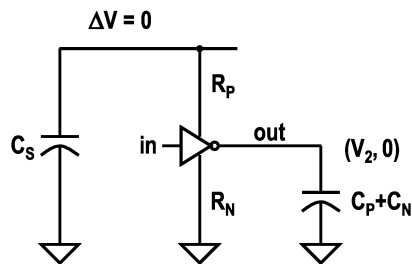
- Charge flow from C_S : $Q = (C_P + C_N)V_2$
- Charge discharged locally: $Q_{cancelled} = 0$
- Charge conservation law: $C_S V_1 = (C_S + C_P + C_N)V_2$

Inv Model A: when the output is rising



- Energy consumed:
$$\begin{aligned} \Delta E &= E_1 - E_2 = \frac{1}{2} C_S V_1^2 - \frac{1}{2} (C_S + C_P + C_N) V_2^2 \\ &= \frac{1}{2} C_S V_1 (V_1 - V_2) = \frac{1}{2} (C_P + C_N) V_1 V_2 \\ &= \frac{1}{2} \frac{C_S (C_P + C_N)}{C_S + C_P + C_N} V_1^2 = \frac{1}{2} \{C_S // (C_P + C_N)\} V_1^2 \\ &= \frac{1}{2} \frac{(C_P + C_N)(C_S + C_P + C_N)}{C_S} V_2^2 \end{aligned}$$

Inv Model A: when the output is falling



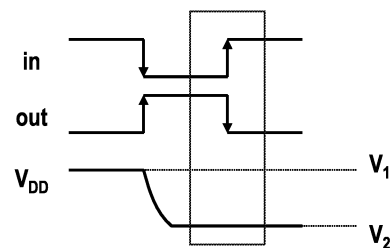
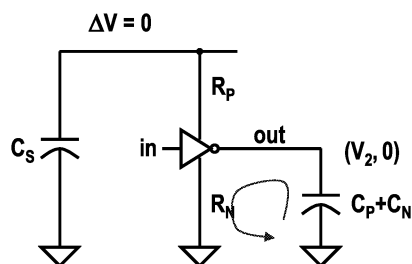
- Before transition,

$$Q_1 = (C_S + C_P + C_N)V_2 \quad E_1 = \frac{1}{2}(C_S + C_P + C_N)V_2^2$$

- After transition,

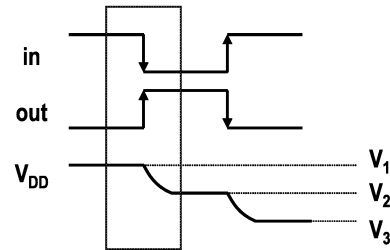
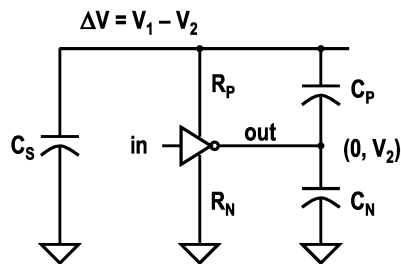
$$Q_2 = C_S V_2 \quad E_2 = \frac{1}{2}C_S V_2^2$$

Inv Model A: when the output falling



- Charge flow from C_S : $Q = 0$
- Energy consumed: $\Delta E = E_1 - E_2 = \frac{1}{2}(C_P + C_N)V_2^2$
- Charge discharged locally: $Q_{cancelled} = (C_P + C_N)V_2$
- Charge conservation law: $C_S V_2 = C_S V_2$

Inv Model B: when the output rising



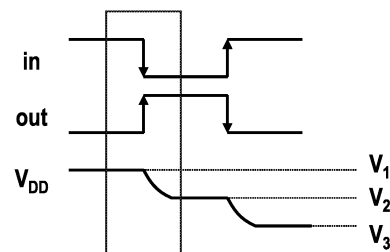
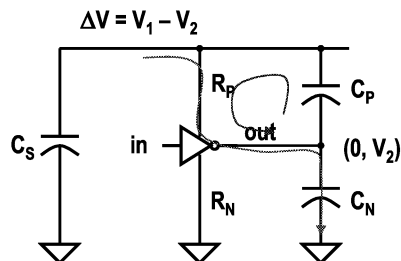
- Before transition,

$$Q_1 = (C_S + C_P)V_1 \quad E_1 = \frac{1}{2}(C_S + C_P)V_1^2$$

- After transition,

$$Q_2 = (C_S + C_N)V_2 \quad E_2 = \frac{1}{2}(C_S + C_N)V_2^2$$

Inv Model B: when the output is rising



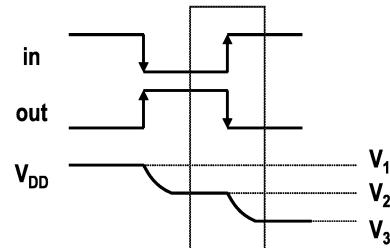
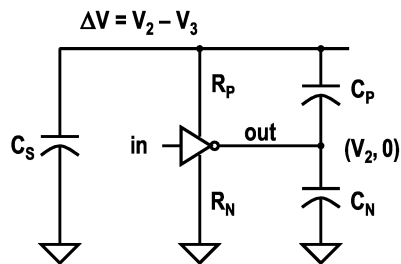
- Charge flow from C_S : $Q = C_N V_2$

- Charge discharged locally: $Q_{cancelled} = C_P V_1$

- Charge conservation law: $C_S V_1 = (C_S + C_N)V_2$

- Energy consumed: $\Delta E = E_1 - E_2 = \frac{1}{2}C_P V_1^2 + \frac{1}{2}C_N V_1 V_2$

Inv Model B: when the output is falling



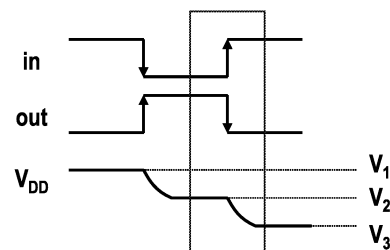
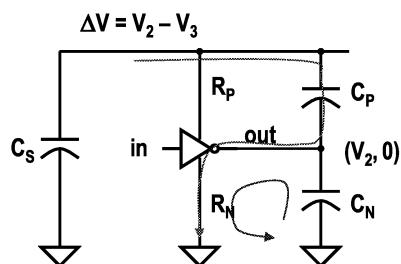
- Before transition,

$$Q_1 = (C_S + C_N)V_2 \quad E_1 = \frac{1}{2}(C_S + C_N)V_2^2$$

- After transition,

$$Q_2 = (C_S + C_P)V_3 \quad E_2 = \frac{1}{2}(C_S + C_P)V_3^2$$

Inv Model B: when the output is falling



- Charge flow from C_S : $Q = C_P V_3$

- Charge cancelled locally: $Q_{cancelled} = C_N V_2$

- Charge conservation law: $C_S V_2 = (C_S + C_P)V_3$

- Energy consumed: $\Delta E = E_1 - E_2 = \frac{1}{2}C_N V_2^2 + \frac{1}{2}C_P V_2 V_3$

CMOS Inverter Model A vs Model B

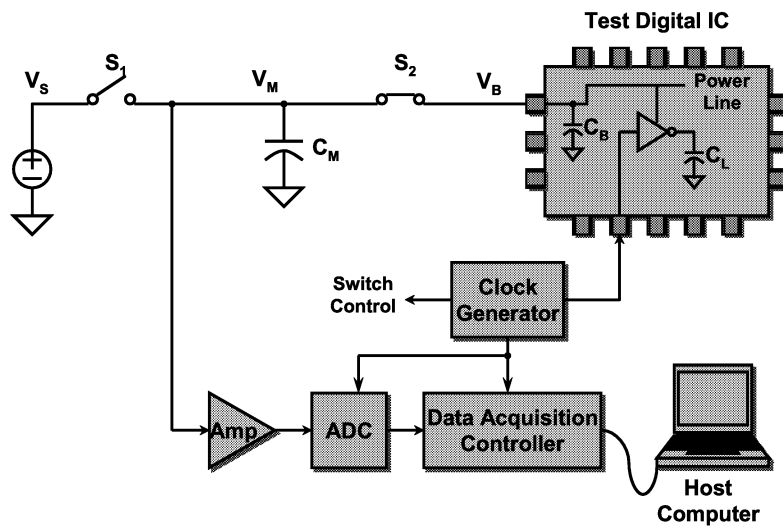
		Simple Model A	Model B
Output ↑	Charge from C_S	$(C_P+C_N)V_2$	$C_N V_2$
	Energy consumed	$1/2(C_P+C_N)V_1V_2$	$1/2C_P V_1^2 + 1/2C_N V_1V_2$
Output ↓	Charge from C_S	0	$C_P V_3$
	Energy consumed	$1/2(C_P+C_N)V_2^2$	$1/2C_N V_2^2 + 1/2C_P V_2V_3$

CMOS Inverter Model A vs Model B

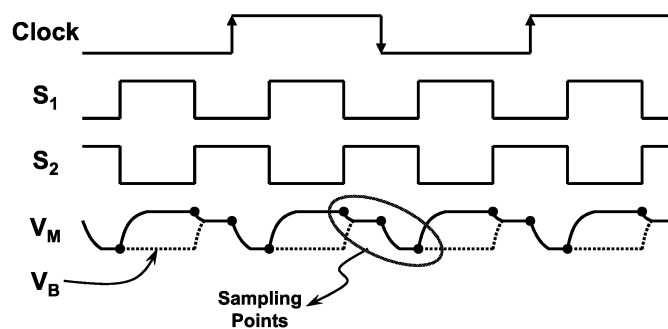
Assuming $C_s = \infty$

		Simple Model A	Model B
Output ↑	Charge from C_S	$(C_P+C_N)V$	$C_N V$
	Energy consumed	$1/2(C_P+C_N)V^2$	$1/2(C_P+C_N)V^2$
Output ↓	Charge from C_S	0	$C_P V$
	Energy consumed	$1/2(C_P+C_N)V^2$	$1/2(C_P+C_N)V^2$

Cycle-Accurate Energy Measurement System

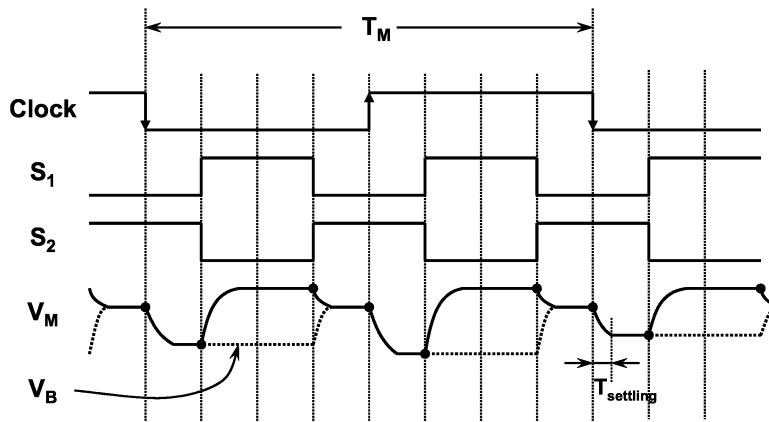


Waveforms of Energy Measurement System



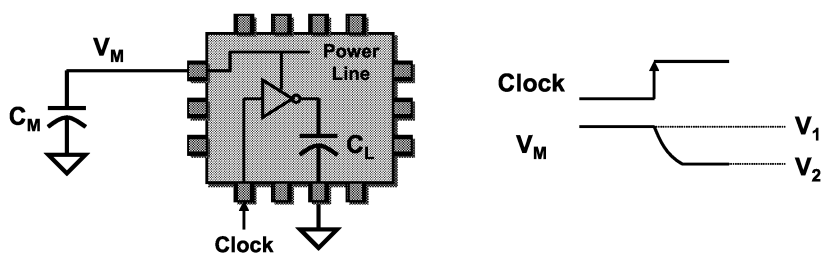
- Three sampling points for each clock transition

Settling Time Constraint



- All data points must be sampled after V_M is settled.
- $T_{\text{settling}} < T_M/8$
- $T_M = 1/f_M$, (f_M : operating frequency during measurement)
- Typically $f_M < f_O$ to meet the settling constraint. (f_O : actual operating frequency)

Energy Model 1: Transferred energy under measurement

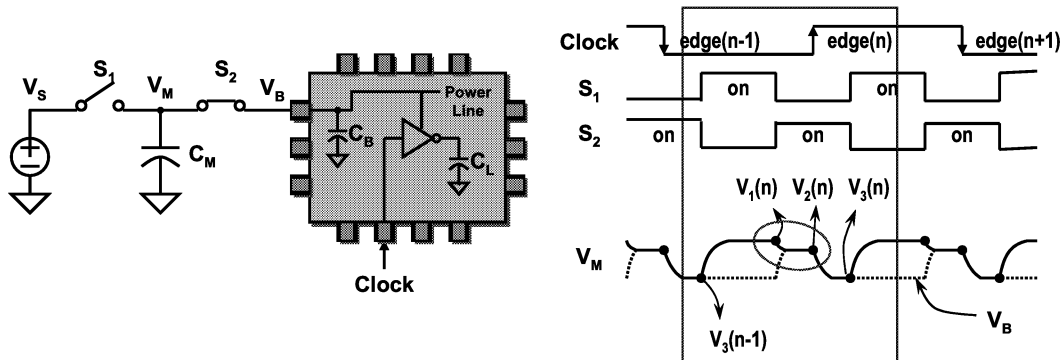


Difference of energy stored in C_M

$$E = \frac{1}{2} C_M V_1^2 - \frac{1}{2} C_M V_2^2$$

- Energy transferred from capacitor C_M
- Is this energy equal to the energy consumed in the chip?

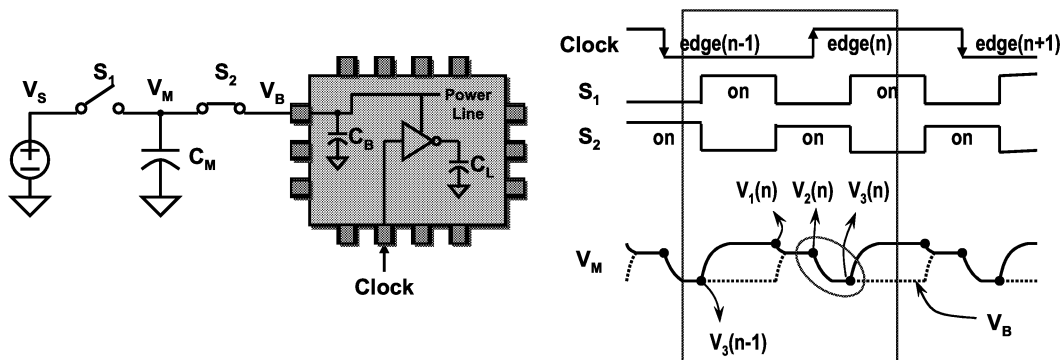
Energy Model 1: C_B Effect



- C_B : on-chip/off-chip bypass capacitor, parasitic capacitor on PCB or package
- Charge conservation law when S_2 on

$$C_M V_1(n) + C_B(n) V_3(n-1) = (C_M + C_B(n)) V_2(n) \quad \dots \textcircled{1}$$

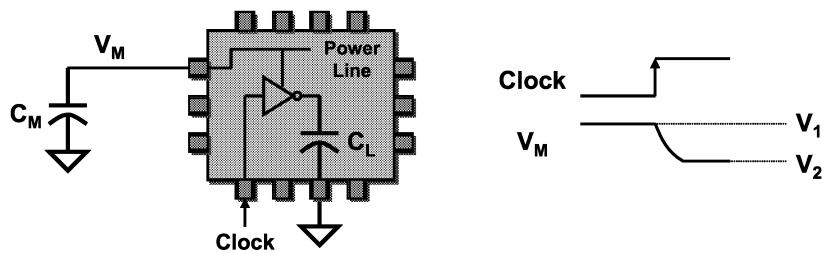
Energy Model 1: Calculate C_B & $E(n)$



$$C_B(n) = \frac{V_1(n) - V_2(n)}{V_2(n) - V_3(n-1)} \cdot C_M$$

$$E(n) = \frac{1}{2} (C_M + C_B(n)) (V_2^2(n) - V_3^2(n))$$

Energy Model 1: Problem

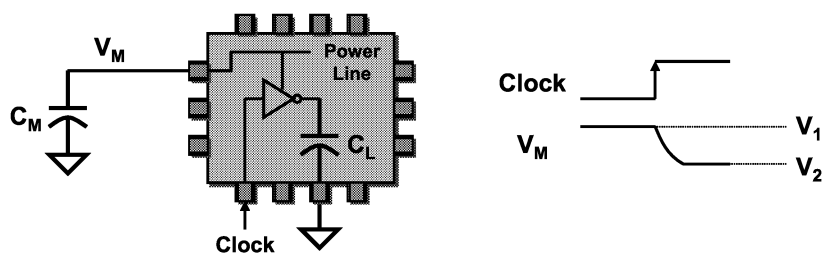


Difference of energy stored in C_M

$$E = \frac{1}{2} C_M V_1^2 - \frac{1}{2} C_M V_2^2$$

- Must consider the supply voltage drop during the energy flow
- Measured energy < real energy consumed with ideal supply

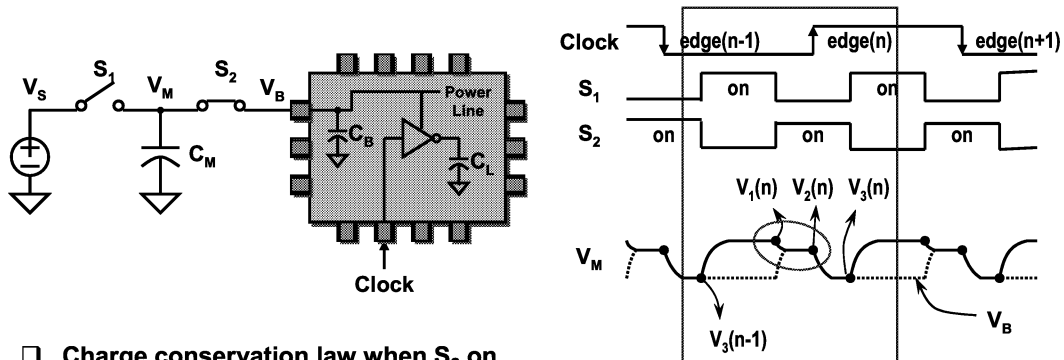
Energy Model 2: Consumed energy



$$E_{consumed} = C_L V_{DD}^2$$

- Energy consumption based on effective switching capacitance C_L
- Assume that capacitance is time invariant and voltage independent
- No error due to the supply voltage variation

Energy Model 2: Calculate C_B



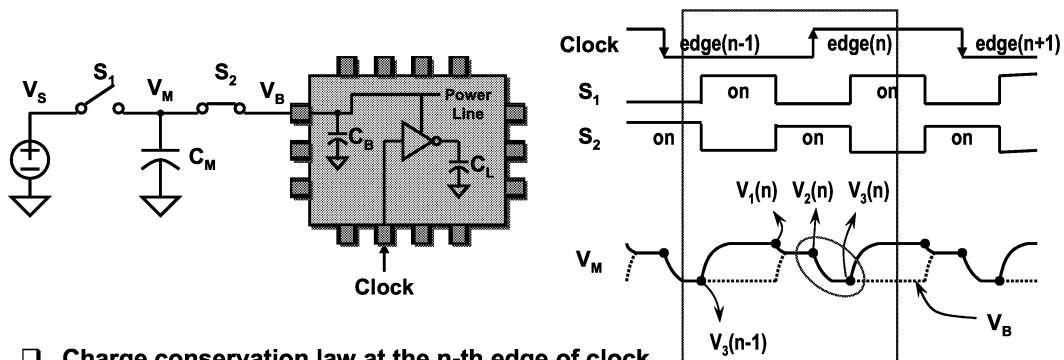
- Charge conservation law when S_2 on

$$C_M V_1(n) + C_B V_3(n-1) = (C_M + C_B) V_2(n) \quad \dots \textcircled{1}$$

- Same with energy model 1

$$C_B(n) = \frac{V_1(n) - V_2(n)}{V_2(n) - V_3(n-1)} \cdot C_M$$

Energy Model 2: Calculate C_L



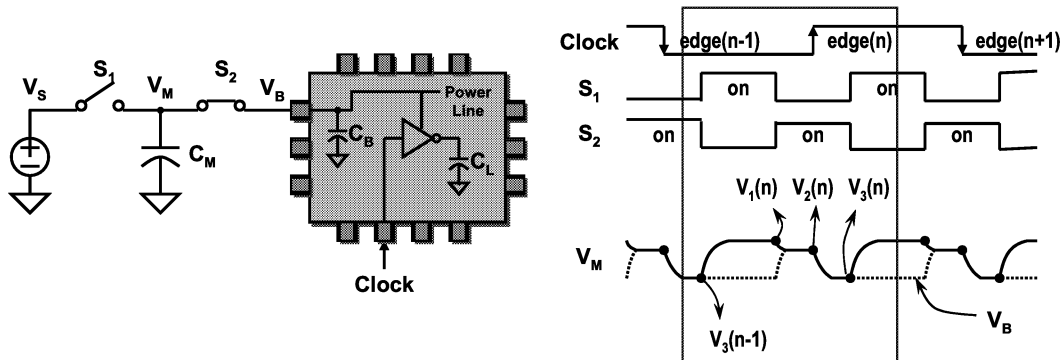
- Charge conservation law at the n -th edge of clock

$$(C_M + C_B(n)) V_2(n) = (C_M + C_B(n) + C_L(n)) V_3(n) \quad \dots \textcircled{2}$$

- Load capacitance for the n -th edge

$$C_L(n) = \frac{V_2(n) - V_3(n)}{V_3(n)} \cdot (C_M + C_B(n))$$

Energy Model 2: Calculate E(n)



- Energy consumption for the n-th edge

$$E(n) = C_L(n) \cdot V_{DD}^2$$

Comparison between Models 1 and 2

- Model 1:

$$E(n) = \frac{1}{2} (C_M + C_B(n)) (V_2^2(n) - V_3^2(n))$$

- Model 2:

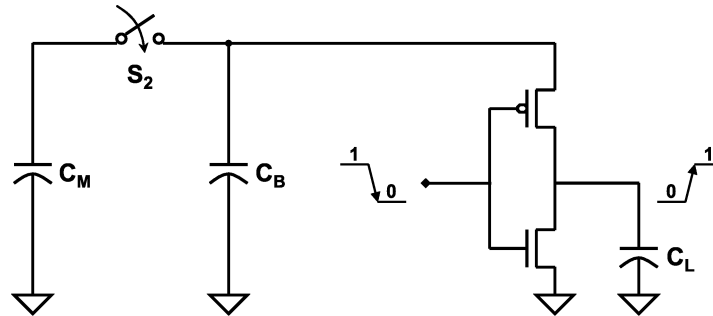
$$\begin{aligned} E(n) &= C_L(n) \cdot V_{DD}^2 \\ &= C_L(n) \cdot V_2^2(n) \quad (\text{assume } V_{DD} = V_2(n)) \\ &= (C_M + C_B(n)) \frac{V_2(n) - V_3(n)}{V_3(n)} \cdot V_2^2(n) \end{aligned}$$

- $\Delta E = E(n)$ of Model 1 - $E(n)$ of Model 2

$$\begin{aligned} \Delta E(n) &= \frac{1}{2} (C_M + C_B(n)) (V_2^2(n) - V_3^2(n)) - (C_M + C_B(n)) \frac{V_2(n) - V_3(n)}{V_3(n)} \cdot V_2^2(n) \\ &= \frac{1}{2} (C_M + C_B(n)) \cdot (\Delta V) \left(-3\Delta V - \frac{2(\Delta V)^2}{V_3(n)} \right) \quad (\Delta V = V_2(n) - V_3(n)) \\ &\cong -\frac{3}{2} (C_M + C_B(n)) \cdot (\Delta V)^2 \end{aligned}$$

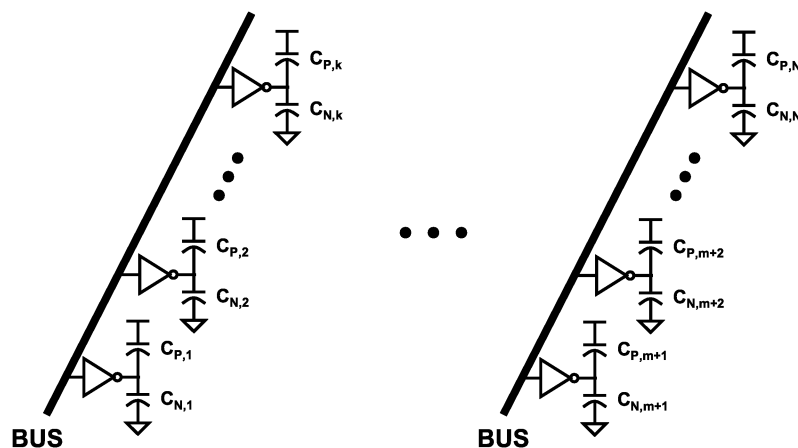
Model 1 underestimates the consumed energy.

Simplification In Energy Models 1 and 2



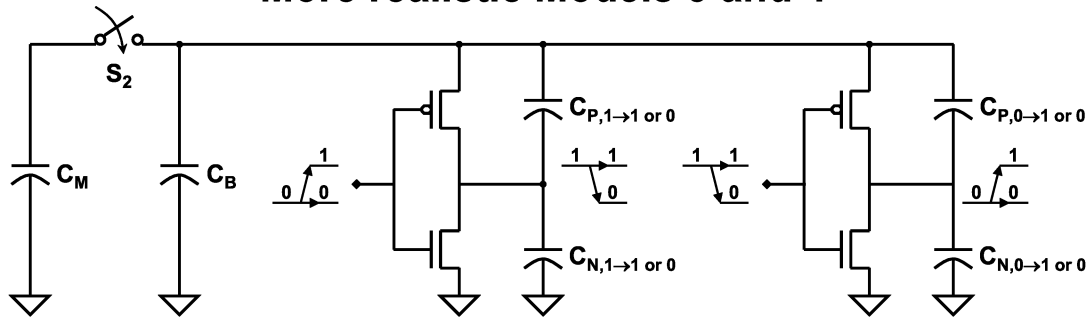
- Each load capacitor has only the capacitance to the ground line, not to the power line.
- CMOS inverter model A is used.

Load Capacitors in CMOS VLSI Circuits



- N: total number of load capacitors

More realistic Models 3 and 4

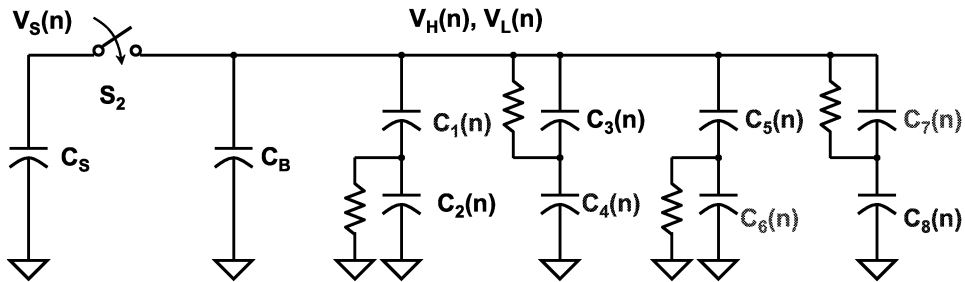


- All the output nodes considered.
 - Group 1: stay low
 - Group 2: stay high
 - Group 3: switch from low to high
 - Group 4: switch from high to low
- The load capacitors in the group 2 acts like C_B
- CMOS inverter model B is used

Classification of Load Capacitors

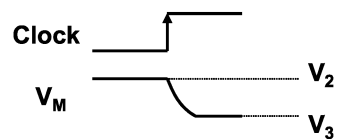
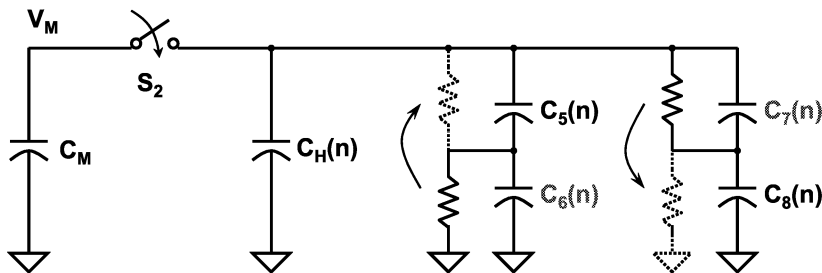
Group	Symbol		Input States
1	$C_{1N}(n) / C_{1P}(n)$	$C_2(n) / C_1(n)$	Stay in the low state
2	$C_{2N}(n) / C_{2P}(n)$	$C_4(n) / C_3(n)$	Stay in the high state
3	$C_{3N}(n) / C_{3P}(n)$	$C_6(n) / C_5(n)$	Switch from low to high
4	$C_{4N}(n) / C_{4P}(n)$	$C_8(n) / C_7(n)$	Switch from high to low

Energy Model 3: Transferred energy



- Group 1 (stay low) : $C_1(n), C_2(n)$
- Group 2 (stay high) : $C_3(n), C_4(n)$
- Group 3 (low to high): $C_5(n), C_6(n)$
- Group 4 (high to low): $C_7(n), C_8(n)$

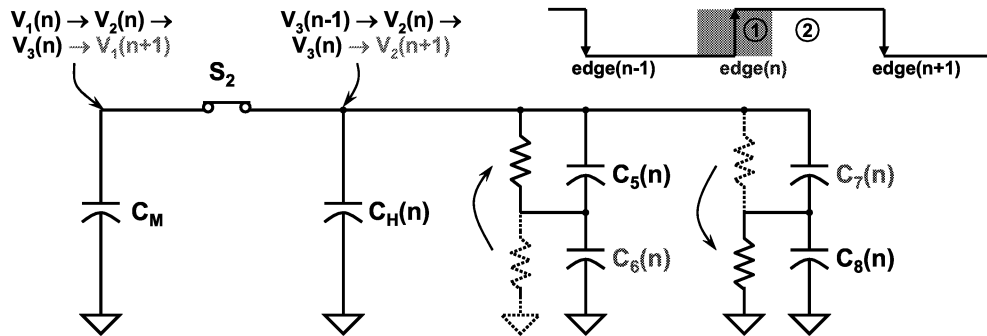
Energy Model 3: Simplified



$$E = (C_6 + C_7) \cdot V_{DD}^2$$

- $C_H(n) = C_B + C_1(n) + C_4(n)$

Energy Model 3: Calculate $C_H(n)$



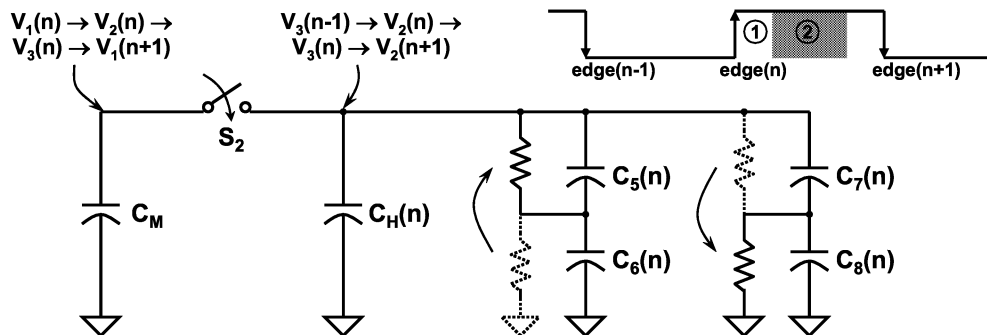
- Charge conservation law before and after the clock edge

$$(C_M + C_H(n))V_2(n) = (C_M + C_{H67}(n))V_3(n) \quad \dots \textcircled{1}$$

$$C_H(n) = \frac{(C_M + C_{H67}(n))V_3(n)}{V_2(n)} - C_M = \left[\frac{V_3(n)}{V_2(n)} \left(\frac{V_1(n+1) - V_3(n)}{V_2(n+1) - V_3(n)} \right) - 1 \right] \cdot C_M$$

- $C_{H67}(n) = C_H(n) + C_6(n) + C_7(n)$. It will be calculated in next phase.

Energy Model 3: Calculate $C_{H67}(n)$

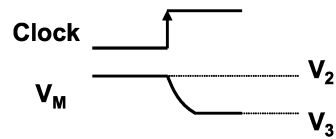
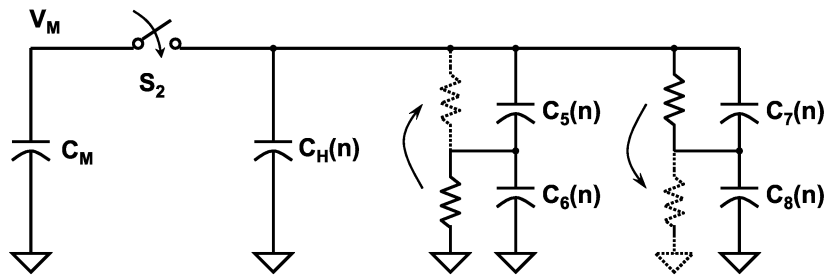


- Open S_2 and close S_1 to charge C_M
- Charge conservation law when S_2 is closed (S_1 open) after the edge n

$$C_M V_1(n+1) + C_{H67}(n) V_3(n) = (C_M + C_{H67}(n)) V_2(n+1) \quad \dots \textcircled{2}$$

$$C_{H67}(n) = \frac{V_1(n+1) - V_2(n+1)}{V_2(n+1) - V_3(n)} \cdot C_M$$

Energy Model 3: Calculate $C_L(n)$ & $E(n)$

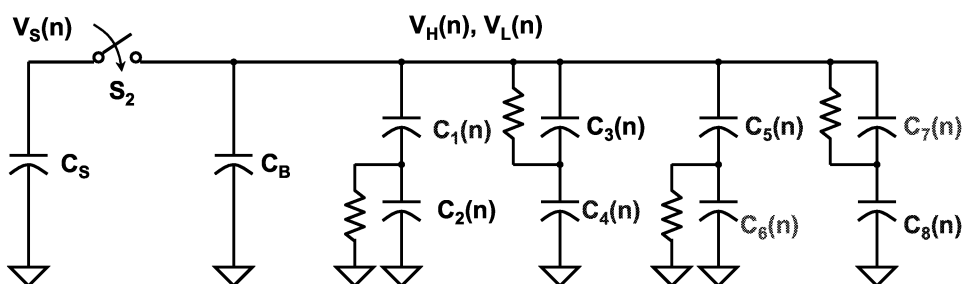


$$E = (C_6 + C_7) \cdot V_{DD}^2$$

□ From $C_{67}(n) = C_{H67}(n) - C_H(n)$

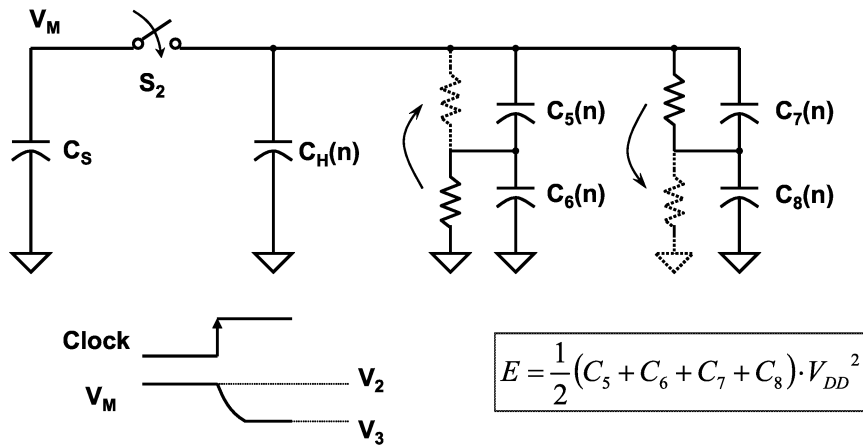
$$C_{67}(n) = \left(1 - \frac{V_3(n)}{V_2(n)}\right) \left(\frac{V_1(n+1) - V_3(n)}{V_2(n+1) - V_3(n)}\right) \cdot C_M$$

Energy Model 4: Consumed energy



- Group 1 (stay low) : $C_1(n), C_2(n)$
- Group 2 (stay high) : $C_3(n), C_4(n)$
- Group 3 (low to high): $C_5(n), C_6(n)$
- Group 4 (high to low): $C_7(n), C_8(n)$

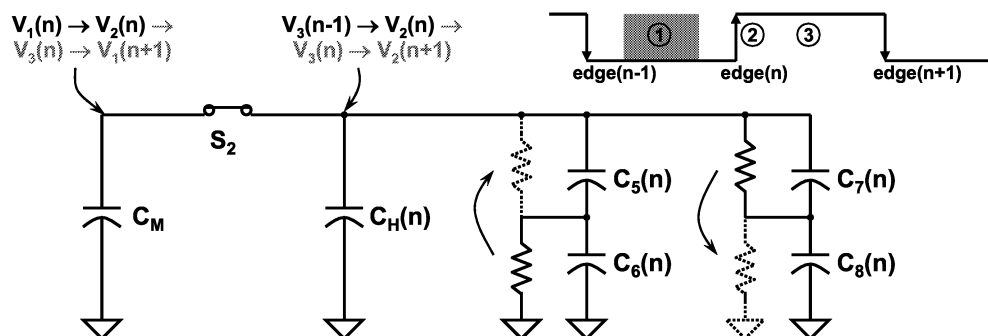
Energy Model 4: Simplified



$$E = \frac{1}{2} (C_5 + C_6 + C_7 + C_8) \cdot V_{DD}^2$$

- Heat dissipation
- $C_H(n) = C_B + C_1(n) + C_4(n)$

Phase 1 of Model 4: Calculate $C_{H58}(n)$

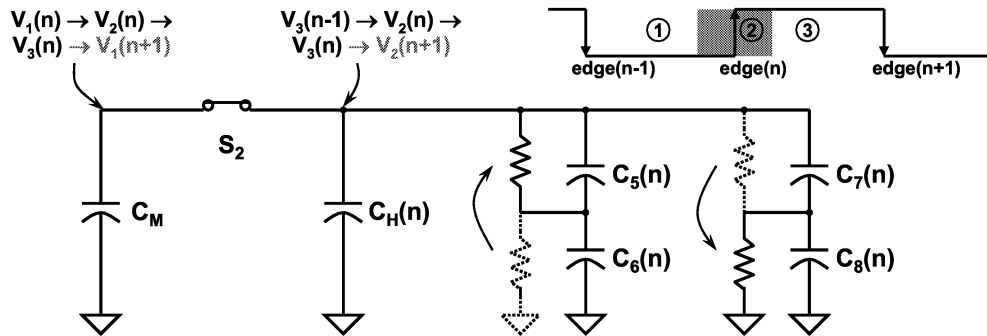


- $C_{H58}(n) = C_H(n) + C_5(n) + C_8(n)$
- Charge conservation law when S_2 is closed before the edge n

$$C_M V_1(n) + C_{H58}(n) V_3(n-1) = (C_M + C_{H58}(n)) V_2(n) \quad \dots \textcircled{1}$$

$$C_{H58}(n) = \frac{V_1(n) - V_2(n)}{V_2(n) - V_3(n-1)} \cdot C_M$$

Phase 2 of Model 4: Calculate $C_H(n)$



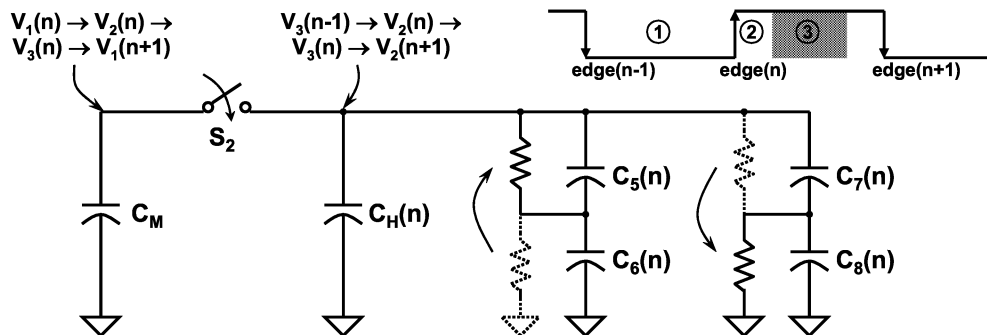
- Charge conservation law during the clock edge

$$(C_M + C_H(n))V_2(n) = (C_M + C_{H67}(n))V_3(n) \quad \dots \textcircled{2}$$

$$C_H(n) = \frac{(C_M + C_{H67}(n))V_3(n)}{V_2(n)} - C_M = \left[\frac{V_3(n)}{V_2(n)} \left(\frac{V_1(n+1) - V_3(n)}{V_2(n+1) - V_3(n)} \right) - 1 \right] \cdot C_M$$

- $C_{H67}(n) = C_H(n) + C_6(n) + C_7(n)$. It will be calculated in the next phase.

Energy Model 4: Calculate $C_{H67}(n)$



- Charge conservation law when S_2 is closed after the edge n

$$C_M V_1(n+1) + C_{H67}(n) V_3(n) = (C_M + C_{H67}(n)) V_2(n+1) \quad \dots \textcircled{3}$$

$$C_{H67}(n) = \frac{V_1(n+1) - V_2(n+1)}{V_2(n+1) - V_3(n)} \cdot C_M$$

Energy Model 4: Calculate $C_L(n)$ (n-th edge)

$$C_{H58}(n) = \frac{V_1(n) - V_2(n)}{V_2(n) - V_3(n-1)} \cdot C_M = \left[\frac{V_1(n) - V_3(n-1)}{V_2(n) - V_3(n-1)} - 1 \right] C_M$$

$$C_{H67}(n) = \frac{V_1(n+1) - V_2(n+1)}{V_2(n+1) - V_3(n)} \cdot C_M = \left[\frac{V_1(n+1) - V_3(n)}{V_2(n+1) - V_3(n)} - 1 \right] C_M$$

$$E = \frac{1}{2} C_L V_{DD}^2$$

$$C_H(n) = \left[\frac{V_3(n)}{V_2(n)} \left(\frac{V_1(n+1) - V_3(n)}{V_2(n+1) - V_3(n)} \right) - 1 \right] C_M$$

$$C_{58}(n) = C_{H58}(n) - C_H = \left[\frac{V_1(n) - V_3(n-1)}{V_2(n) - V_3(n-1)} - \frac{V_3(n)}{V_2(n)} \left(\frac{V_1(n+1) - V_3(n)}{V_2(n+1) - V_3(n)} \right) \right] C_M$$

$$C_{67}(n) = C_{H67}(n) - C_H = \left(1 - \frac{V_3(n)}{V_2(n)} \right) \left(\frac{V_1(n+1) - V_3(n)}{V_2(n+1) - V_3(n)} \right) C_M$$

$$C_L(n) = C_{5678}(n) = \left[\frac{V_1(n) - V_3(n-1)}{V_2(n) - V_3(n-1)} + \left(1 - \frac{2V_3(n)}{V_2(n)} \right) \left(\frac{V_1(n+1) - V_3(n)}{V_2(n+1) - V_3(n)} \right) \right] C_M$$

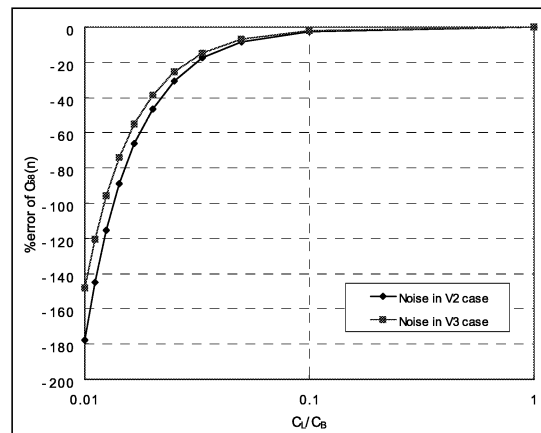
Energy Model 4: Capacitance Ratio Constraint (1)

- In the energy model 4, we obtained the following equation

$$\begin{aligned} C_{58}(n) &= \left[\frac{V_1(n) - V_3(n-1)}{V_2(n) - V_3(n-1)} - \frac{V_3(n)}{V_2(n)} \left(\frac{V_1(n+1) - V_3(n)}{V_2(n+1) - V_3(n)} \right) \right] C_M \\ &= \left[\Delta(n) - \frac{V_3(n)}{V_2(n)} \Delta(n+1) \right] C_M \end{aligned}$$

- But if the target chip is consists of large bypass capacitor C_B , relatively, the ratio of (C_L/C_B) becomes very small.
- And it results in very small signal level, that is, $V_2(n) - V_3(n) \cong 0$.
- If $V_2(n) - V_3(n) \cong 0$, then we may obtain the negative capacitance value of $C_{58}(n)$ for the small noise level.

Energy Model 4: Capacitance Ratio Constraint (2)



- ❑ This plot is for the fixed noise level of 0.2mV.
- ❑ Consequently, in case that (C_L/C_B) ratio is small, energy model 4 requires very precise measurement system.

Energy Model Summary

	Equation	Description
Model 1	$E = \frac{1}{2}C_M V_1^2 - \frac{1}{2}C_M V_2^2$	Energy transferred from capacitor C_M .
Model 2	$E = C_L V_{DD}^2$	Energy consumption based on the load capacitance C_L .
Model 3	$E = (C_6 + C_7) \cdot V_{DD}^2$	Energy consumed from supply.
Model 4	$E = \frac{1}{2}(C_5 + C_6 + C_7 + C_8) \cdot V_{DD}^2$	Energy consumption in the chip. Heat dissipation.

- ❑ Model 1, 2, 3 are related to the energy from supply, but model 4 is the energy consumed in the chip.

Appendix (1): Σ_0 vs Σ_1

□ **Definition**

- Σ_0 : total energy transferred from supply
- Σ_1 : total energy consumed in the chip

□ **Using our energy models, we can write Σ_0 and Σ_1 as follows**

$$\Sigma_0 = \sum_{n=1}^N E_3(n) = V_{DD}^2 \cdot \sum_{n=1}^N C_{67}(n)$$

$$\Sigma_1 = \sum_{n=1}^N E_4(n) = \frac{1}{2} V_{DD}^2 \cdot \sum_{n=1}^N C_{5678}(n)$$

Appendix (1): Σ_0 vs Σ_1

□ **For simplicity, we define as follows**

$$\alpha_n = \frac{V_3(n)}{V_2(n)}$$

$$\Delta_n = \frac{V_1(n) - V_3(n-1)}{V_2(n) - V_3(n-1)}$$

□ **Then the $C_{67}(n)$ and $C_{5678}(n)$ in the previous section can be rewritten,**

$$C_{67}(n) = (1 - \alpha_n) \Delta_{n+1} C_M$$

$$C_{5678}(n) = [\Delta_n + (1 - 2\alpha_n) \Delta_{n+1}] C_M$$

Appendix (1): Σ_0 vs Σ_1

$$\begin{aligned}
 \Sigma_1 &= \frac{1}{2} V_{DD}^2 \cdot \sum_{n=1}^{2N} C_{5678}(n) \\
 &= \frac{1}{2} V_{DD}^2 \cdot \sum_{k=1}^N [C_{5678}(2k-1) + C_{5678}(2k)] \\
 &= \frac{1}{2} V_{DD}^2 \cdot \sum_{k=1}^N [(\Delta_{2k-1} + (1-2\alpha_{2k-1})\Delta_{2k}) + (\Delta_{2k} + (1-2\alpha_{2k})\Delta_{2k+1})] C_M \\
 &= \frac{1}{2} V_{DD}^2 \cdot \sum_{k=1}^N [(\Delta_{2k-1} - \Delta_{2k} + 2(1-\alpha_{2k-1})\Delta_{2k}) + (\Delta_{2k} - \Delta_{2k+1} + 2(1-\alpha_{2k})\Delta_{2k+1})] C_M \\
 &= \frac{1}{2} V_{DD}^2 \cdot \sum_{k=1}^N [(\Delta_{2k-1} - \Delta_{2k+1}) C_M + 2C_{67}(2k-1) + 2C_{67}(2k)] \\
 &= V_{DD}^2 \cdot \sum_{k=1}^N [C_{67}(2k-1) + C_{67}(2k)] + \frac{1}{2} V_{DD}^2 \cdot \sum_{k=1}^N [(\Delta_{2k-1} - \Delta_{2k+1}) C_M] \\
 &= \Sigma_0 + \frac{1}{2} C_M V_{DD}^2 \cdot (\Delta_1 - \Delta_{2N+1})
 \end{aligned}$$

- Thus we can say that $\Sigma_0 \cong \Sigma_1$ for large N, that is, (total transferred energy)=(total consumed energy).

Appendix (2): Σ_1 vs Σ_2

□ **Definition**

- Σ_1 : total energy consumed in the chip
 - Σ_2 : total energy consumed in the chip when $V_2(n+1)$ is affected by noise
- $\Sigma_2 - \Sigma_1$ can be written as follows because the $C_{5678}(n)$ and $C_{5678}(n+1)$ are related with $V_2(n+1)$ and others are not.

$$\begin{aligned}
 \Sigma_2 - \Sigma_1 &= \sum_{n=1}^N E_4^*(n) - \sum_{n=1}^N E_4(n) \\
 &= \frac{1}{2} V_{DD}^2 \cdot \sum_{n=1}^N C_{5678}^*(n) - \frac{1}{2} V_{DD}^2 \cdot \sum_{n=1}^N C_{5678}(n) \\
 &= \frac{1}{2} V_{DD}^2 [C_{5678}^*(n) + C_{5678}^*(n+1) - C_{5678}(n) - C_{5678}(n+1)]
 \end{aligned}$$

Appendix (2): Σ_1 vs Σ_2

□ For simplicity, we define as follows

$$p_n = \frac{C_{58}(n)}{C_M} = \Delta_n - \alpha_n \Delta_{n+1}$$

$$q_n = \frac{C_{67}(n)}{C_M} = (1 - \alpha_n) \Delta_{n+1} = p_n + (\Delta_{n+1} - \Delta_n)$$

$$\Delta V_2 = V_2^*(n+1) - V_2(n+1) \quad (\Delta V_2 : \text{noise})$$

Appendix (2): Σ_1 vs Σ_2

□ Then $(\Sigma_2 - \Sigma_1)$ can be rewritten,

$$\begin{aligned} \Sigma_2 - \Sigma_1 &= \frac{1}{2} C_M V_{DD}^2 \left[(p_n^* + q_n^* + p_{n+1}^* + q_{n+1}^*) - (p_n + q_n + p_{n+1} + q_{n+1}) \right] \\ &= \frac{1}{2} C_M V_{DD}^2 \left[(\Delta_{n+1}^* - \Delta_n^* + 2p_n^* + \Delta_{n+2}^* - \Delta_{n+1}^* + 2p_{n+1}^*) - (\Delta_{n+1} - \Delta_n + 2p_n + \Delta_{n+2} - \Delta_{n+1} + 2p_{n+1}) \right] \\ &= \frac{1}{2} C_M V_{DD}^2 \left[(\Delta_{n+1}^* - \Delta_n + 2p_n^* + \Delta_{n+2} - \Delta_{n+1}^* + 2p_{n+1}^*) - (\Delta_{n+1} - \Delta_n + 2p_n + \Delta_{n+2} - \Delta_{n+1} + 2p_{n+1}) \right] \\ &= C_M V_{DD}^2 \left[(p_n^* + p_{n+1}^*) - (p_n + p_{n+1}) \right] \quad \dots \textcircled{1} \end{aligned}$$

□ Note that $\Delta_n^* = \Delta_n$ and $\Delta_{n+2}^* = \Delta_{n+2}$ because they have no dependency on $V_2(n+1)$.

Appendix (2): Σ_1 vs Σ_2

□ p_{n+1}^* can be written

$$\begin{aligned}
 p_{n+1}^* &= \Delta_{n+1}^* - \alpha_{n+1}^* \Delta_{n+2}^* \\
 &= \frac{\Delta_n^* - p_n^*}{\alpha_n^*} - \frac{V_3^*(n+1)}{V_2^*(n+1)} \Delta_{n+2}^* \\
 &= \frac{(p_n + \alpha_n \Delta_{n+1}) - p_n^*}{\alpha_n} - \frac{V_3(n+1)}{V_2(n+1) + \Delta V_2} \Delta_{n+2} \quad (\because p_n = \Delta_n - \alpha_n \Delta_{n+1}) \\
 &= \Delta_{n+1} + \frac{p_n - p_n^*}{\alpha_n} - \frac{V_3(n+1)}{V_2(n+1)} \cdot \frac{1}{1 + \frac{\Delta V_2}{V_2(n+1)}} \Delta_{n+2} \\
 &\cong \Delta_{n+1} + \frac{p_n - p_n^*}{\alpha_n} - \frac{V_3(n+1) \Delta_{n+2}}{V_2(n+1)} \cdot \left(1 - \frac{\Delta V_2}{V_2(n+1)}\right) \quad (\because \Delta V_2 \ll V_2(n+1)) \\
 &= \Delta_{n+1} - \alpha_{n+1} \Delta_{n+1} + \frac{p_n - p_n^*}{\alpha_n} + \alpha_{n+1} \Delta_{n+2} \cdot \frac{\Delta V_2}{V_2(n+1)} \\
 &= p_{n+1} + p_n - p_n^* + \left(1 - \frac{1}{\alpha_n}\right) (p_n^* - p_n) + \alpha_{n+1} \Delta_{n+2} \cdot \frac{\Delta V_2}{V_2(n+1)} \quad \dots \textcircled{2}
 \end{aligned}$$

Appendix (2): Σ_1 vs Σ_2

□ In order to calculate ΔV_2 in Eq. ②, we solve the following equation for $V_2(n+1)$

$$p_n = \Delta_n - \alpha_n \Delta_{n+1} = \Delta_n - \alpha_n \frac{V_1(n+1) - V_3(n)}{V_2(n+1) - V_3(n)}$$

□ Then

$$V_2(n+1) = V_3(n) + (V_1(n+1) - V_3(n)) \cdot \frac{\alpha_n}{\Delta_n - p_n}$$

□ In the same way

$$\begin{aligned}
 V_2^*(n+1) &= V_3^*(n) + (V_1^*(n+1) - V_3^*(n)) \cdot \frac{\alpha_n^*}{\Delta_n^* - p_n^*} \\
 &= V_3(n) + (V_1(n+1) - V_3(n)) \cdot \frac{\alpha_n}{\Delta_n - p_n}
 \end{aligned}$$

Appendix (2): Σ_1 vs Σ_2

□ ΔV_2 can be written

$$\begin{aligned}
 \Delta V_2 &= V_2^*(n+1) - V_2(n+1) \\
 &= \left[V_3(n) + (V_1(n+1) - V_3(n)) \cdot \frac{\alpha_n}{\Delta_n - p_n^*} \right] - \left[V_3(n) + (V_1(n+1) - V_3(n)) \cdot \frac{\alpha_n}{\Delta_n - p_n} \right] \\
 &= \alpha_n (V_1(n+1) - V_3(n)) \left(\frac{1}{\Delta_n - p_n^*} - \frac{1}{\Delta_n - p_n} \right) \\
 &= \alpha_n (V_1(n+1) - V_3(n)) \frac{1}{\Delta_n} \left(\frac{1}{1 - \frac{p_n^*}{\Delta_n}} - \frac{1}{1 - \frac{p_n}{\Delta_n}} \right) \\
 &\cong \alpha_n (V_1(n+1) - V_3(n)) \frac{1}{\Delta_n} \left(\frac{p_n^*}{\Delta_n} - \frac{p_n}{\Delta_n} \right) \quad (\because V_2(n) \cong V_3(n) \Rightarrow p_n \ll \Delta_n) \\
 &= \alpha_n (V_1(n+1) - V_3(n)) \frac{1}{\Delta_n} (p_n^* - p_n)
 \end{aligned}$$

Appendix (2): Σ_1 vs Σ_2

□ In Eq. ②,

$$\begin{aligned}
 \alpha_{n+1} \Delta_{n+2} \frac{1}{V_2(n+1)} \Delta V_2 &= \alpha_{n+1} \Delta_{n+2} \frac{1}{V_2(n+1)} \cdot \alpha_n (V_1(n+1) - V_3(n)) \frac{1}{\Delta_n} (p_n^* - p_n) \\
 &= (\Delta_{n+1} - p_{n+1}) \frac{1}{V_2(n+1)} \cdot \alpha_n (V_1(n+1) - V_3(n)) \frac{1}{\Delta_n} (p_n^* - p_n) \\
 &= \left(\frac{\Delta_n - p_n}{\alpha_n} - p_{n+1} \right) \frac{1}{V_2(n+1)} \cdot \alpha_n (V_1(n+1) - V_3(n)) \frac{1}{\Delta_n} (p_n^* - p_n) \\
 &= \left(1 - \frac{p_n + \alpha_n p_{n+1}}{\Delta_n} \right) \frac{V_1(n+1) - V_3(n)}{V_2(n+1)} \cdot \frac{1}{\Delta_n} (p_n^* - p_n)
 \end{aligned}$$

Appendix (2): Σ_1 vs Σ_2

□ From Eq. ① and ②, we write ($\Sigma_2 - \Sigma_1$),

$$\begin{aligned}\Sigma_2 - \Sigma_1 &= C_M V_{DD}^2 \left[(p_n^* + p_{n+1}^*) - (p_n + p_{n+1}) \right] \\ &= C_M V_{DD}^2 \left[\left(1 - \frac{1}{\alpha_n} \right) + \left(1 - \frac{p_n + \alpha_n p_{n+1}}{\Delta_n} \right) \frac{V_1(n+1) - V_3(n)}{V_2(n+1)} \cdot \frac{1}{\Delta_n} \right] (p_n^* - p_n)\end{aligned}$$

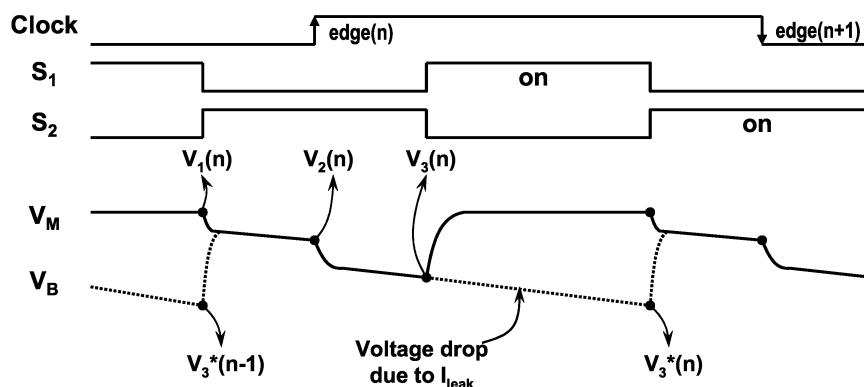
□ If $V_2(n) \cong V_3(n)$ then $\alpha_n \cong 1$ and $\Delta_n = L$. (L: large number)

□ And you can see that ($\Sigma_2 - \Sigma_1$) $\cong 0$.

□ Concluding remark

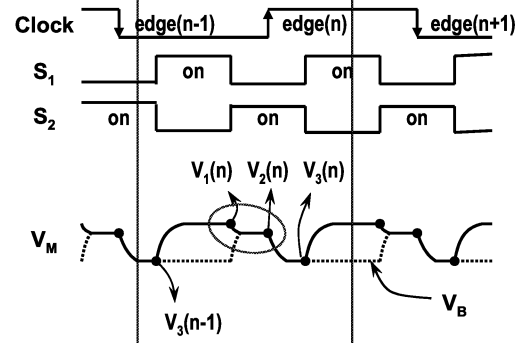
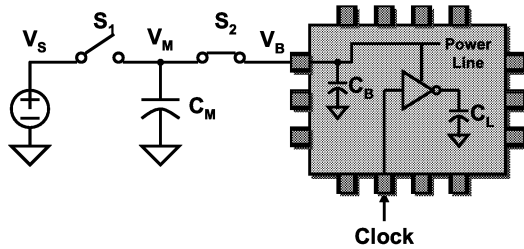
- If $V_2(n) \cong V_3(n)$, $C_{5678}(n)$ is very sensitive to noise (ΔV_2) and it sometimes goes to negative capacitance value.
- But the total consumed energy is almost not changed by the noise (ΔV_2).
- From this, we can fix the error of $C_{5678}(n)$ when it goes to the negative value.

Energy Model 2: Leakage Current Effect



□ I_{leak} : static current independent of clock. Leakage of static CMOS, analog sub-block current, and so on.

Energy Model 2: Calculate C_B with I_{leak}



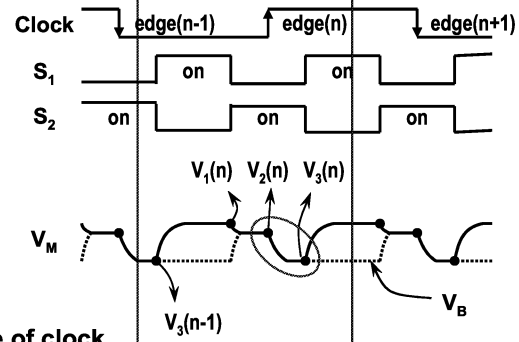
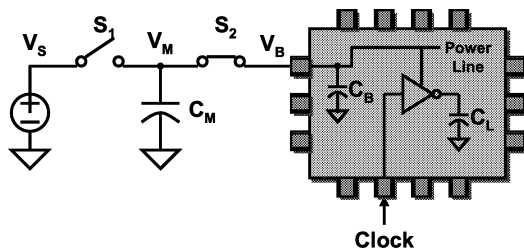
- Charge conservation law when S_2 on

$$C_M V_1(n) + C_B(n) V_3(n-1) = (C_M + C_B(n)) V_2(n) + I_{leak} \frac{3T_M}{8} \quad \dots \textcircled{1}$$

- Same with energy model 1

$$C_B(n) = \frac{C_M (V_1(n) - V_2(n)) - \frac{3I_{leak} T_M}{8}}{V_2(n) - V_3(n-1)}$$

Energy Model 2: Calculate C_L with I_{leak}



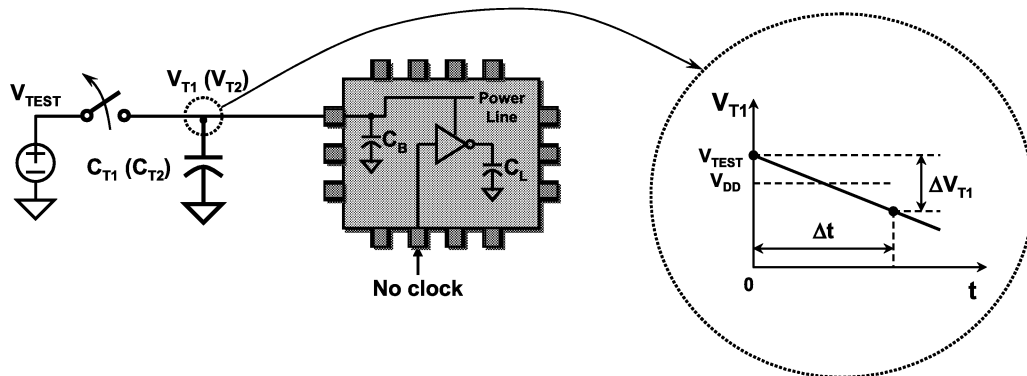
- Charge conservation law at the n-th edge of clock

$$(C_M + C_B(n)) V_2(n) = (C_M + C_B(n) + C_L(n)) V_3(n) + I_{leak} \frac{T_M}{8} \quad \dots \textcircled{2}$$

- Load capacitance for the n-th edge

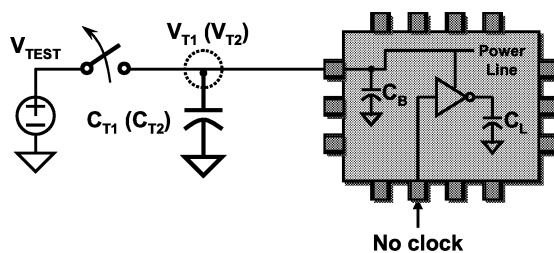
$$C_L(n) = \frac{(C_M + C_B(n))(V_2(n) - V_3(n)) - \frac{I_{leak} T_M}{8}}{V_3(n)}$$

Energy Model 2: I_{leak} Measurement



- Assume constant I_{leak} for simplicity
- No clocking
- Two measurements with two different capacitor C_{T1} , C_{T2} to determine C_B
- Voltage V_{TEST} slightly above V_{DD}

Energy Model 2: Calculate I_{leak}



$$(C_{T1} + C_B)\Delta V_{T1} = I_{leak}\Delta t$$

$$(C_{T2} + C_B)\Delta V_{T2} = I_{leak}\Delta t$$

$$C_B = \frac{C_{T2}\Delta V_{T2} - C_{T1}\Delta V_{T1}}{\Delta V_{T1} - \Delta V_{T2}}$$

$$I_{leak} = \frac{C_{T2} - C_{T1}}{\Delta V_{T1} - \Delta V_{T2}} \cdot \frac{\Delta V_{T1}\Delta V_{T2}}{\Delta t}$$

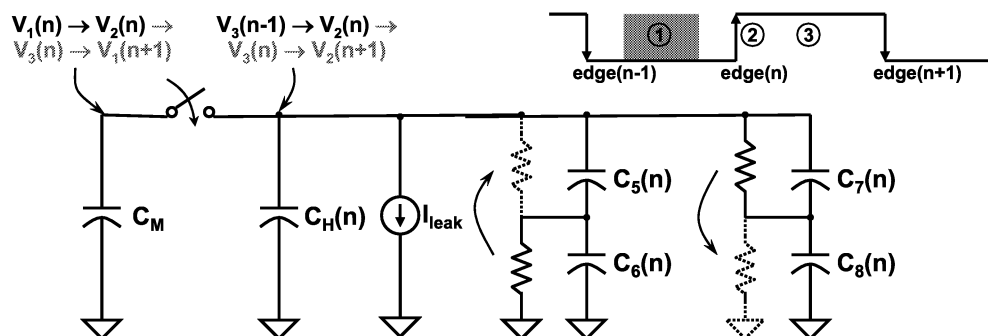
Energy Model 2: Frequency Compensation

- Energy consumption for the n-th clock edge

$$E(n) = C_L(n) \cdot V_{DD}^2 + \frac{1}{2} I_{leak} T_O V_{DD}$$

- $T_O = 1/f_O$, f_O : actual operating frequency
- $C_L(n)V_{DD}^2$: effective switching energy
- $(1/2)I_L T_O V_{DD}$: energy due to static current during a half period

Energy Model 4: Calculate $C_{H58}(n)$ with I_{leak}

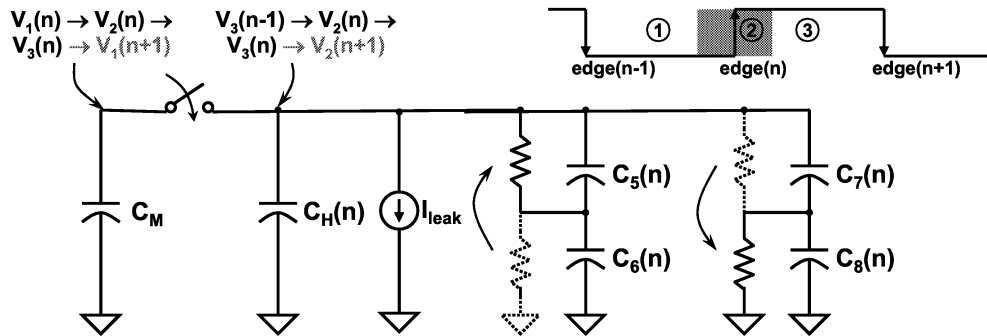


- Charge conservation law when S_1 on

$$C_M V_1(n) + C_{H58}(n) V_3(n-1) = (C_M + C_{H58}(n)) V_2(n) + \frac{3I_{leak} T_M}{8} \quad \dots \textcircled{1}$$

$$C_{H58}(n) = \frac{C_M (V_1(n) - V_2(n)) - \frac{3I_{leak} T_M}{8}}{V_2(n) - V_3(n-1)}$$

Energy Model 4: Calculate $C_H(n)$ with I_{leak}

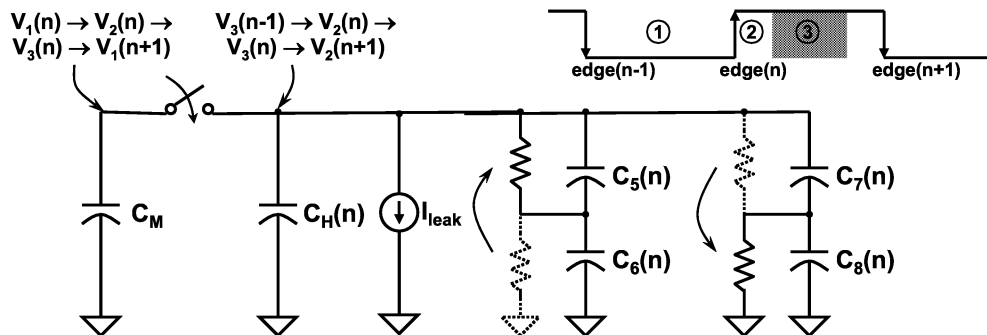


□ Charge conservation law during clock edge

$$(C_M + C_H(n))V_2(n) = (C_M + C_{H67}(n))V_3(n) + \frac{I_{leak}T_M}{8} \quad \dots \textcircled{2}$$

$$C_H(n) = \frac{(C_M + C_{H67}(n))V_3(n) + \frac{I_{leak}T_M}{8}}{V_2(n)} - C_M$$

Energy Model 4: Calculate $C_{H67}(n)$ with I_{leak}

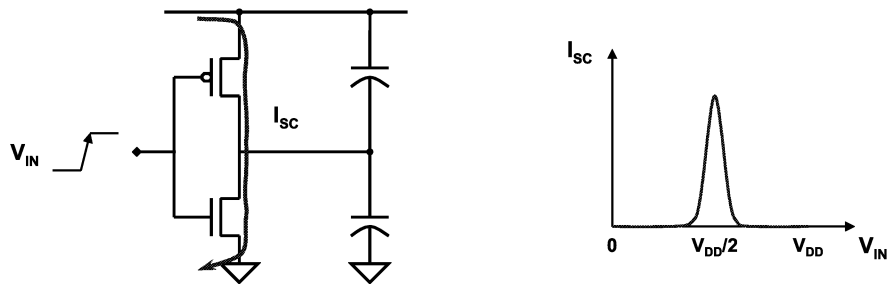


□ Charge conservation law when S_1 on

$$C_M V_1(n+1) + C_{H67}(n) V_3(n) = (C_M + C_{H67}(n)) V_2(n+1) + \frac{3I_{leak}T_M}{8} \quad \dots \textcircled{3}$$

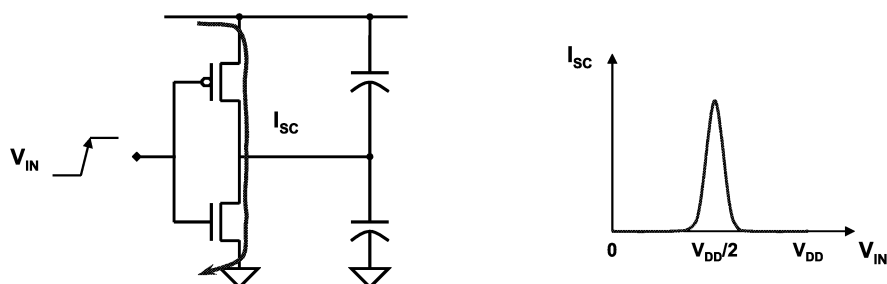
$$C_{H67}(n) = \frac{C_M (V_1(n+1) - V_2(n+1)) - \frac{3I_{leak}T_M}{8}}{V_2(n+1) - V_3(n)}$$

CMOS Short-Circuit Current



- ❑ During transition from 0 to 1 or from 1 to 0.
- ❑ Both NMOS and PMOS are on for a short period of time.
- ❑ This results in a short current pulse from V_{DD} to GND.
- ❑ I_{sc} may be misunderstood as load capacitance C_L in the measurement system.

E_{real}(n) with I_{sc}



- ❑ If we assume that the total charge loss by I_{sc} is Q_{sc}(n), the consumed energy E(n) in real operation

$$E_{real}(n) = C_{\delta 7,real}(n) \cdot V_{DD}^2 + Q_{sc}(n) \cdot V_{DD} \quad \dots \textcircled{1}$$

$E_{\text{meas}}(n)$ with I_{SC}

- $E_{\text{meas}}(n)$ can be written

$$E_{\text{meas}}(n) = C_{67,\text{eff}}(n) \cdot V_{DD}^2 \quad \dots \textcircled{2}$$

- In the CCL for the n-th edge

$$\begin{aligned} (C_M + C_H(n))V_2(n) &= (C_M + C_H(n) + C_{67,\text{eff}}(n))V_3(n) \\ &= (C_M + C_H(n) + C_{67,\text{real}}(n))V_3(n) + Q_{\text{SC}}(n) \quad \dots \textcircled{3} \end{aligned}$$

- Assume that

$$V_3(n) = V_{DD} - \Delta V_3$$

$E_{\text{meas}}(n)$ with I_{SC}

- From Eq.② and Eq.③

$$\begin{aligned} E_{\text{meas}}(n) &= C_{67,\text{eff}}(n) \cdot V_{DD}^2 \\ &= C_{67}(n) \cdot V_{DD}^2 + \frac{Q_{\text{SC}}(n)}{V_3(n)} \cdot V_{DD}^2 \end{aligned}$$

$$E_{\text{real}}(n) = C_{67}(n) \cdot V_{DD}^2 + Q_{\text{SC}}(n)V_{DD}$$

$$\Delta E(n) = E_{\text{eff}}(n) - E_{\text{real}}(n)$$

$$= \left(\frac{V_{DD}}{V_3(n)} - 1 \right) Q_{\text{SC}}(n)V_{DD}$$

$$= \left(\frac{1}{1 - \frac{\Delta V_3}{V_{DD}}} - 1 \right) Q_{\text{SC}}(n)V_{DD}$$

$$= Q_{\text{SC}}(n)\Delta V_3$$

$E_{\text{real}}(n)$ vs $E_{\text{meas}}(n)$ Example

□ Assumptions:

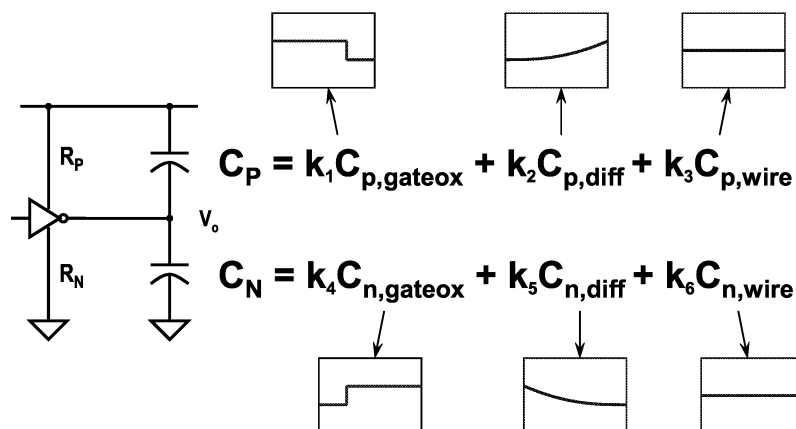
- $Q_{\text{SC}}(n) \cdot V_{\text{DD}}$ in Eq. ① is 10% of $E_{\text{real}}(n)$
- $V_{\text{DD}} = 3.0\text{V}$
- $V_3(n) = 2.8\text{V}$

□ Then, the % error

$$\% \text{ error} = \frac{E_{\text{meas}}(n) - E_{\text{real}}(n)}{E_{\text{real}}(n)} \cong 0.7\%$$

- Consequently, the error from calculating I_{SC} as C_L in our measurement system is not large.

Voltage dependent Load Capacitance

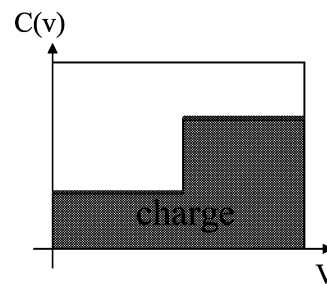
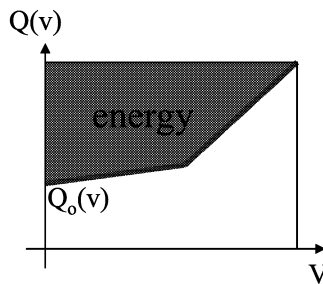


Differential Capacitance vs Static Capacitance

$$C_1 = \frac{dQ}{dV} \quad : \text{dynamic}$$

$$C_2 = \frac{Q}{V} \quad : \text{static}$$

□ Q(v) and C(v) curves



Differential Capacitance vs Static Capacitance

$$C_1 = \frac{dQ}{dV} \quad : \text{dynamic}$$

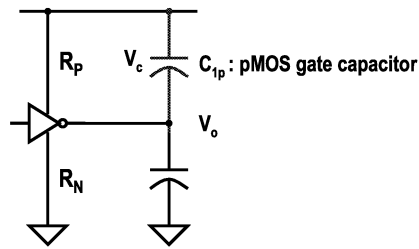
$$C_2 = \frac{Q}{V} \quad : \text{static}$$

□ Relationship between C_1 & C_2

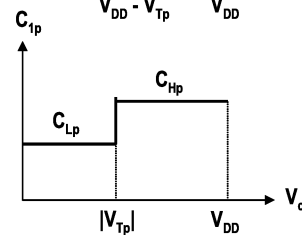
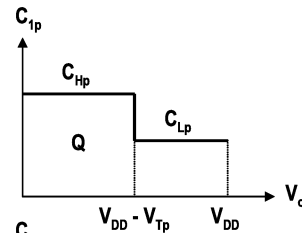
$$Q(V) = \int \frac{dQ}{dV} dV = \int C_1(V) dV + Q(0)$$

$$C_2(V) = \frac{Q(V)}{V} = \frac{\int C_1(V) dV + Q(0)}{V}$$

Energy stored in the pMOS capacitance

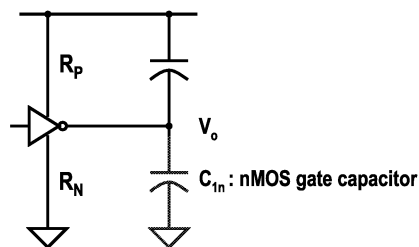


$$\begin{aligned}
 E_{p, \text{gateox}}^{\text{stored}} &= \int V_c dQ \\
 &= \int_0^{V_{DD}} C_{1p}(V) \cdot V_c dV_c \\
 &= \int_0^{|V_{Tp}|} C_{Lp} V dV + \int_{|V_{Tp}|}^{V_{DD}} C_{Hp} V dV \\
 &= \frac{1}{2} C_{Lp} |V_{Tp}|^2 + \frac{1}{2} C_{Hp} (V_{DD}^2 - |V_{Tp}|^2)
 \end{aligned}$$

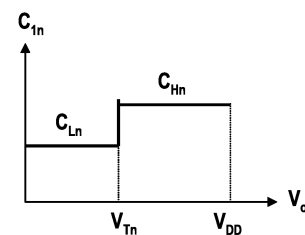


$$Q_{p, \text{gateox}}^{\text{transferred}} = C_{Hp} (V_{DD} - |V_{Tp}|) + C_{Lp} |V_{Tp}|$$

Energy stored in the nMOS capacitance

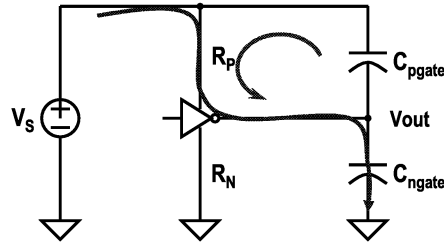


$$\begin{aligned}
 E_{n, \text{gateox}}^{\text{stored}} &= \int V_o dQ \\
 &= \int_0^{V_{DD}} C_{1n}(V) \cdot V_o dV_o \\
 &= \int_0^{V_{Tn}} C_{Ln} V dV + \int_{V_{Tn}}^{V_{DD}} C_{Hn} V dV \\
 &= \frac{1}{2} C_{Ln} V_{Tn}^2 + \frac{1}{2} C_{Hn} (V_{DD}^2 - V_{Tn}^2)
 \end{aligned}$$



$$Q_{n, \text{gateox}}^{\text{transferred}} = C_{Ln} V_{Tn} + C_{Hn} (V_{DD} - V_{Tn})$$

Energy Flow for Gate Capacitance Load

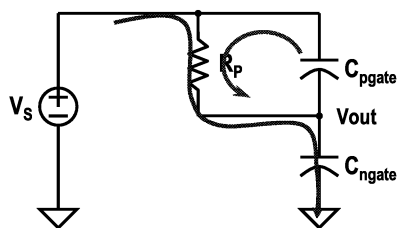


- When Vout is rising, $E_{gateox}^{loss} (\uparrow) = E_{p,gateox}^{stored} + E_{n,gateox}^{loss}$
- When Vout is falling, $E_{gateox}^{loss} (\downarrow) = E_{n,gateox}^{stored} + E_{p,gateox}^{loss}$

Energy Flow when the output is rising

Before transition

$$E_{p,gateox}^{stored} = \frac{1}{2} \left[C_{Lp} |V_{Tp}|^2 + C_{Hp} (V_{DD}^2 - |V_{Tp}|^2) \right]$$



After transition

$$E_{gateox}^{transferred} (\uparrow) = Q_{n,gateox}^{transferred} V_{DD} = \{C_{Ln} V_{Tn} + C_{Hn} (V_{DD} - V_{Tn})\} V_{DD}$$

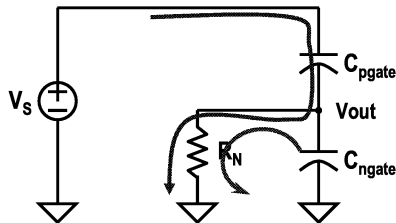
$$E_{n,gateox}^{stored} = \frac{1}{2} \left[C_{Ln} V_{Tn}^2 + C_{Hn} (V_{DD}^2 - V_{Tn}^2) \right]$$

$$E_{gateox}^{loss} (\uparrow) = E_{gateox}^{transferred} (\uparrow) - E_{n,gateox}^{stored} + E_{p,gateox}^{stored}$$

Energy Flow when the output is falling

Before transition

$$E_{n, \text{gateox}}^{\text{stored}} = \frac{1}{2} [C_{Ln} V_{Tn}^2 + C_{Hn} (V_{DD}^2 - V_{Tn}^2)]$$



After transition

$$E_{\text{gateox}}^{\text{transferred}} (\downarrow) = Q_{p, \text{gateox}}^{\text{transferred}} V_{DD} = \{C_{Lp} |V_{Tp}| + C_{Hp} (V_{DD} - |V_{Tp}|)\} V_{DD}$$

$$E_{p, \text{gateox}}^{\text{stored}} = \frac{1}{2} [C_{Lp} V_{Tp}^2 + C_{Hp} (V_{DD}^2 - V_{Tp}^2)]$$

$$E_{\text{gateox}}^{\text{loss}} (\downarrow) = E_{\text{gateox}}^{\text{transferred}} (\downarrow) - E_{p, \text{gateox}}^{\text{stored}} + E_{n, \text{gateox}}^{\text{stored}}$$

Approximation of Energy Related to CMOS Gate

□ For simplicity

- The load consists of the gate oxide capacitance only
- $V_T = V_{Tp} = V_{Tn}$
- $C_H = C_{Hp} = C_{Hn}$, $C_L = C_{Lp} = C_{Ln}$
- $(V_T/V_{DD}) = k$, $(C_L/C_H) = \alpha$

□ Then, for each transition

$$E_{\text{transferred}} = C_H V_{DD}^2 [(1-k) + \alpha k]$$

$$E_{\text{stored}} = \frac{1}{2} C_H V_{DD}^2 [\alpha k^2 + (1-k^2)]$$

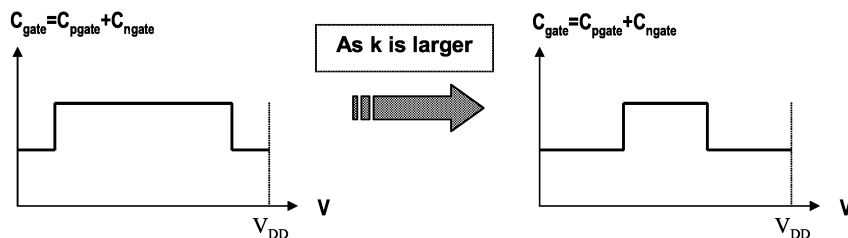
$$E_{\text{loss}} = E_{\text{transferred}}$$

E_{gate} vs k, α

	$k=0, \alpha=0$	$k=0.25, \alpha=0.5$
$E_{\text{transferred}}$	100%	87.5%
E_{stored}	50%	48.4%
E_{loss}	100%	87.5%

- With the typical MOS parameters today, we have $k=0.25, \alpha=0.5$

C_{gate} vs Supply Voltage Scaling



- With the technology scaling, the supply voltage has been lowered.
- then, $k(=V_{\text{T}}/V_{\text{DD}})$ becomes larger.
- Voltage dependency of C_{gate} is not negligible any longer.

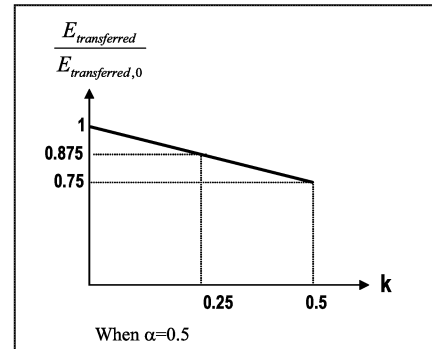
$E_{\text{transferred}}$ vs k

- If $V_T \ll V_{DD}$ or $k \approx 0$

$$E_{\text{transferred},0} = C_H V_{DD}^2$$

- Then, the ratio

$$\begin{aligned} \frac{E_{\text{transferred}}}{E_{\text{transferred},0}} &= (1-k) + \alpha k \\ &= 1 - (1-\alpha) \cdot k \end{aligned}$$



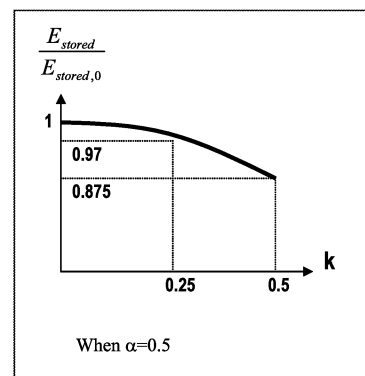
E_{stored} vs k

- If $V_T \ll V_{DD}$ or $k \approx 0$

$$E_{\text{stored},0} = \frac{1}{2} C_H V_{DD}^2$$

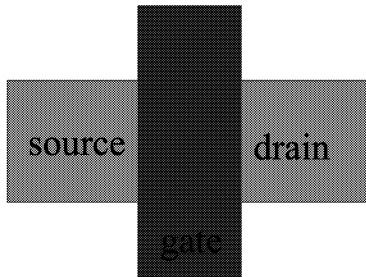
- Then, the ratio

$$\begin{aligned} \frac{E_{\text{stored}}}{E_{\text{stored},0}} &= \alpha k^2 + (1-k^2) \\ &= 1 - (1-\alpha) \cdot k^2 \end{aligned}$$



Diffusion Capacitors

- $C_{diff} = \text{Area} \cdot C_{bottom} + \text{perimeter} \cdot C_{sidewall}$
- Two types of junctions
 - Abrupt junction: bottom ($m=1/2$)
 - Graded junction: sidewall ($m=1/3$)



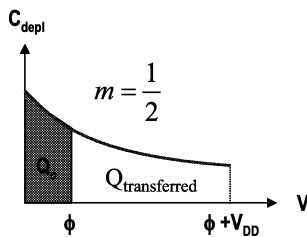
$$C_{depl}(V) = \frac{dQ}{dV} = \frac{\epsilon_{Si}}{t}$$

$$t_{abrupt} = k_{abrupt} \sqrt{V}, \quad k_{abrupt} = \sqrt{\frac{2\epsilon_{Si}}{e} \frac{N_D + N_A}{N_D N_A}}$$

$$t_{graded} = k_{graded} \sqrt[3]{V}, \quad k_{graded} = \sqrt[3]{\frac{12\epsilon_{Si}}{eG}}$$

$$\text{where } N_D - N_A = Gx$$

Capacitance for Abrupt Junctions



$$C_{diff}(V) = \frac{dQ}{dV} = \frac{k}{2\sqrt{V}}, \quad k = \sqrt{2e\epsilon_{Si} \frac{N_D N_A}{N_D + N_A}}$$

$$Q = k\sqrt{V}, \quad Q_{V_{DD}} = k\sqrt{\phi + V_{DD}}, \quad Q_0 = k\sqrt{\phi}$$

$$Q_{transferred} = Q_{V_{DD}} - Q_0 = k(\sqrt{\phi + V_{DD}} - \sqrt{\phi})$$

$$E_{transferred} = Q_{transferred} V_{DD}$$

$$= kV_{DD} (\sqrt{\phi + V_{DD}} - \sqrt{\phi})$$

$$E_{stored} = \int_{Q_0}^{Q_{V_{DD}}} V dQ$$

$$= \int_{Q_0}^{Q_{V_{DD}}} \frac{Q^2}{k^2} dQ = \frac{1}{3} [(\phi + V_{DD}) Q_{V_{DD}} - \phi Q_0]$$

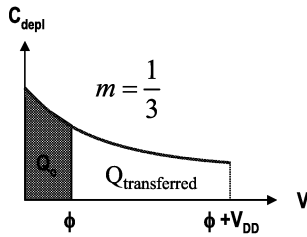
$$E_{stored}(V) = \int_{\phi}^{\phi + V_{DD}} C_{diff}(V) \cdot V dV$$

$$= \frac{k}{2} \int_{\phi}^{\phi + V_{DD}} \frac{V}{\sqrt{V}} dV = \frac{k}{2} \int_{\phi}^{\phi + V_{DD}} \sqrt{V} dV$$

$$= \frac{k}{3} [V\sqrt{V}]_{\phi}^{\phi + V_{DD}} = \frac{k}{3} [(\phi + V_{DD})\sqrt{\phi + V_{DD}} - \phi\sqrt{\phi}]$$

$$= \frac{1}{3} [(\phi + V_{DD}) Q_{V_{DD}} - \phi Q_0]$$

Capacitance for Graded Junctions



$$C_{diff}(V) = \frac{dQ}{dV} = \frac{2}{3} \frac{k}{\sqrt[3]{V}}, \quad k = \frac{3}{2} \sqrt[3]{\frac{e\epsilon_{Si}^2 G}{12}}$$

$$Q = k(V)^{\frac{2}{3}} \quad Q_{V_{DD}} = k(\phi + V)^{\frac{2}{3}}, \quad Q_0 = k(\phi)^{\frac{2}{3}}$$

$$Q_{transferred} = Q_{V_{DD}} - Q_0 = k \left[(\phi + V)^{\frac{2}{3}} - (\phi)^{\frac{2}{3}} \right]$$

$$E_{transferred} = Q_{transferred} V_{DD}$$

$$= k V_{DD} \left[(\phi + V_{DD})^{\frac{2}{3}} - (\phi)^{\frac{2}{3}} \right]$$

$$E_{stored} = \int_{Q_0}^{Q_{V_{DD}}} V dQ$$

$$= \int_{Q_0}^{Q_{V_{DD}}} \left(\frac{Q}{k} \right)^{\frac{3}{2}} dQ = \frac{2}{5} \left[(\phi + V_{DD}) Q_{V_{DD}} - \phi Q_0 \right]$$

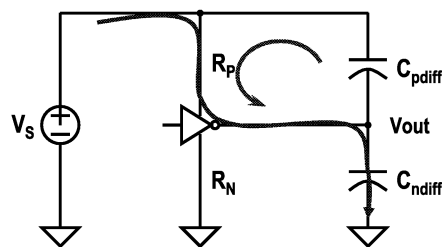
$$E_{stored}(V) = \int_{\phi}^{\phi + V_{DD}} V dQ = \int_{\phi}^{\phi + V_{DD}} C_{diff}(V) \cdot V dV$$

$$= \frac{2}{3} k \int_{\phi}^{\phi + V_{DD}} \frac{V}{\sqrt[3]{V}} dV = \frac{2}{3} k \int_{\phi}^{\phi + V_{DD}} (V)^{\frac{2}{3}} dV$$

$$= \frac{2k}{5} \left[V^{\frac{5}{3}} \right]_{\phi}^{\phi + V_{DD}} = \frac{2k}{5} \left[(\phi + V_{DD})^{\frac{5}{3}} - \phi^{\frac{5}{3}} \right]$$

$$= \frac{2}{5} \left[(\phi + V_{DD}) Q_{V_{DD}} - \phi Q_0 \right]$$

Energy Flow for Junction Capacitance Load

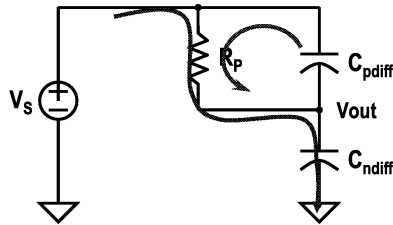


- When V_{out} is rising,
- When V_{out} is falling,

$$E_{diff}^{loss} (\uparrow) = E_{p,diff}^{stored} + E_{n,diff}^{loss}$$

$$E_{diff}^{loss} (\downarrow) = E_{n,diff}^{stored} + E_{p,diff}^{loss}$$

Energy Flow when the output is rising



$$Q_{p,m}^{V_{DD}} = k_{p,m} (\phi_p + V_{DD})^{1-m}, \quad Q_{p,m}^0 = k_{p,m} \phi_p^{1-m}$$

$$Q_{n,m}^{V_{DD}} = k_{n,m} (\phi_n + V_{DD})^{1-m}, \quad Q_{n,m}^0 = k_{n,m} \phi_n^{1-m}$$

Before transition

$$E_{p,diff,m}^{stored} = \frac{1-m}{2-m} [Q_{p,m}^{V_{DD}} (\phi_p + V_{DD}) - Q_{p,m}^0 \phi_p]$$

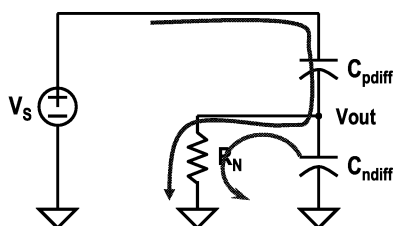
After transition

$$E_{diff,m}^{transferred} (\uparrow) = Q_{n,diff,m}^{transferred} V_{DD} = (Q_{n,diff,m}^{V_{DD}} - Q_{n,diff,m}^0) V_{DD}$$

$$E_{n,diff,m}^{stored} = \frac{1-m}{2-m} [Q_{n,diff,m}^{V_{DD}} (\phi_n + V_{DD}) - Q_{n,diff,m}^0 \phi_n]$$

$$E_{diff,m}^{loss} (\uparrow) = E_{diff,m}^{transferred} (\uparrow) - E_{n,diff,m}^{stored} + E_{p,diff,m}^{stored}$$

Energy Flow when the output is falling



$$Q_{p,m}^{V_{DD}} = k_{p,m} (\phi_p + V_{DD})^{1-m}, \quad Q_{p,m}^0 = k_{p,m} \phi_p^{1-m}$$

$$Q_{n,m}^{V_{DD}} = k_{n,m} (\phi_n + V_{DD})^{1-m}, \quad Q_{n,m}^0 = k_{n,m} \phi_n^{1-m}$$

Before transition

$$E_{n,diff,m}^{stored} = \frac{1-m}{2-m} [Q_{n,diff,m}^{V_{DD}} (\phi_n + V_{DD}) - Q_{n,diff,m}^0 \phi_n]$$

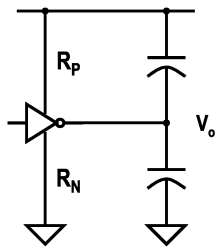
After transition

$$E_{diff,m}^{transferred} (\downarrow) = Q_{p,diff,m}^{transferred} V_{DD} = (Q_{p,diff,m}^{V_{DD}} - Q_{p,diff,m}^0) V_{DD}$$

$$E_{p,diff,m}^{stored} = \frac{1-m}{2-m} [Q_{p,diff,m}^{V_{DD}} (\phi_p + V_{DD}) - Q_{p,diff,m}^0 \phi_p]$$

$$E_{diff,m}^{loss} (\downarrow) = E_{diff,m}^{transferred} (\downarrow) - E_{p,diff,m}^{stored} + E_{n,diff,m}^{stored}$$

Load Capacitance



$$C_P = k_1 C_{p, \text{gateox}} + k_2 C_{p, \text{diff}, 0.5} + k_3 C_{p, \text{diff}, 0.3} + k_4 C_{p, \text{wire}}$$

$$C_N = k_5 C_{n, \text{gateox}} + k_6 C_{n, \text{diff}, 0.5} + k_7 C_{n, \text{diff}, 0.3} + k_8 C_{n, \text{wire}}$$

$$\begin{aligned} E_{\text{transferred}} &= E_{\text{transferred}}(\uparrow) + E_{\text{transferred}}(\downarrow) \\ &= E_{\text{gate}}^{\text{transferred}}(\uparrow) + E_{\text{gate}}^{\text{transferred}}(\downarrow) + E_{\text{wire}}^{\text{transferred}}(\uparrow) + E_{\text{wire}}^{\text{transferred}}(\downarrow) \\ &\quad + E_{\text{diff}, \frac{1}{2}}^{\text{transferred}}(\uparrow) + E_{\text{diff}, \frac{1}{2}}^{\text{transferred}}(\downarrow) + E_{\text{diff}, \frac{1}{3}}^{\text{transferred}}(\uparrow) + E_{\text{diff}, \frac{1}{3}}^{\text{transferred}}(\downarrow) \end{aligned}$$

We need to find 16 constants to find the voltage dependency of the energy consumption. ?!

Typical CMOS Load Capacitance (1)

		$E_{\text{transferred}}$	E_{stored}	E_{consumed}
C_{gate}	Rise	1.81	1.00	2.80
	Fall	3.58	1.99	2.59
C_{junction}	Rise	2.06	1.00	4.22
	Fall	6.38	3.16	4.22
C_{metal}	Rise	2.00	1.00	2.00
	Fall	2.00	1.00	2.00

Assumptions

- $V_{DD}=3.0V$
- $W_N:W_P=1:2$
- $V_{Tn}=0.5V, |V_{Tp}|=0.55V$

Typical CMOS Load Capacitance (2)

□ Assumptions

- $C_L = K_{\text{gate}} \cdot C_{\text{gate}} + K_{\text{junction}} \cdot C_{\text{junction}} + K_{\text{metal}} \cdot C_{\text{metal}}$
- $K_{\text{gate}} : K_{\text{junction}} : K_{\text{metal}} = 2 : 1 : 3$
- (# of 0→1 transition nodes) : (# of 0→1 transition nodes) = 1 : 1 for every edge.

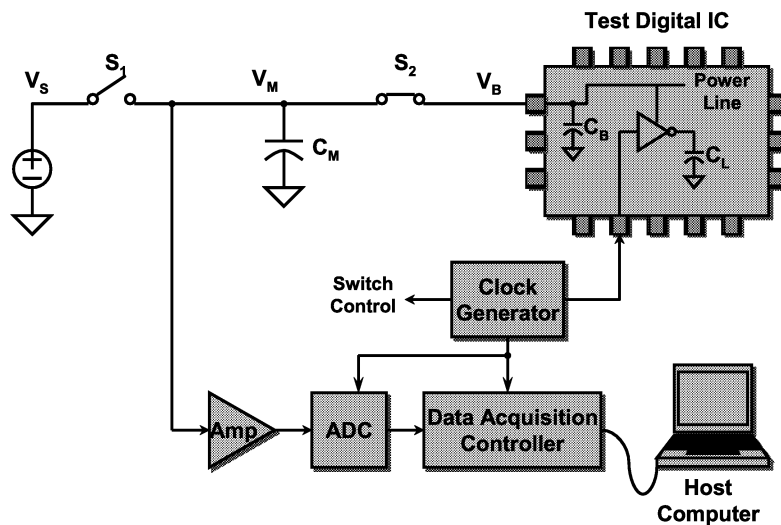
□ Then,

- $E_{\text{transferred}} : E_{\text{stored}} : E_{\text{consumed}} = 1.93 : 1 : 1.93$ for each edge.

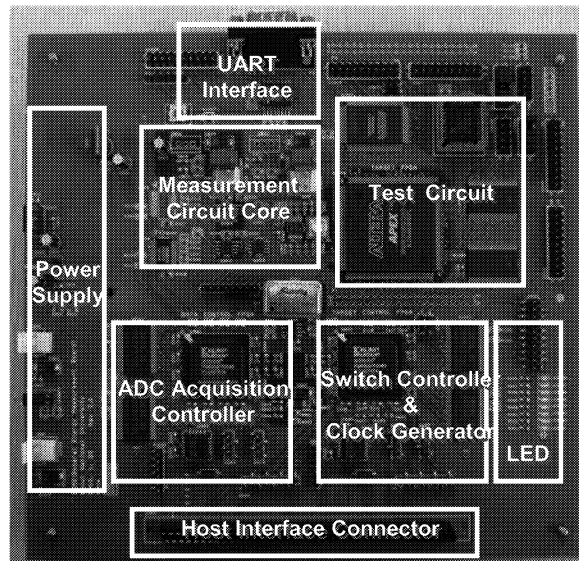
□ From this we can rewrite the consumed energy $E(n)$ in model 4, approximately,

$$E \cong \frac{1}{1.93} (C_5 + C_6 + C_7 + C_8) \cdot V_{DD}^2$$

Cycle-Accurate Energy Measurement System

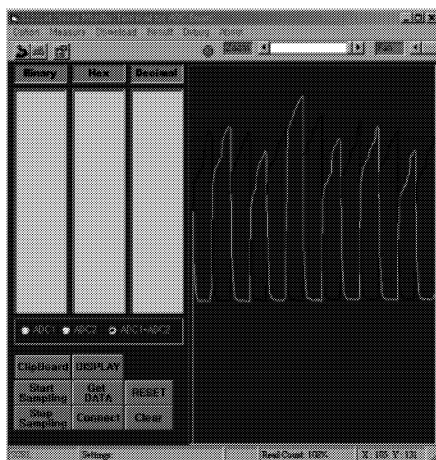


Cycle-Accurate Energy Measurement Board

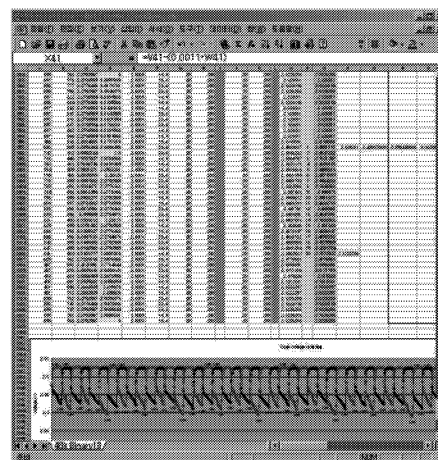


Data Processing Software

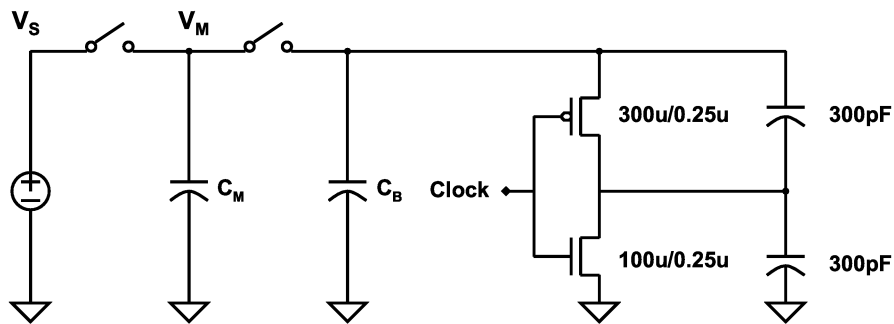
[Monitoring and Data Acquisition Program]



[Energy Calculation Excel Program]

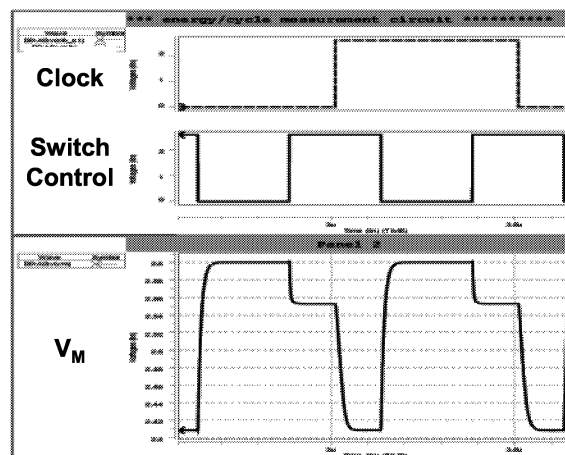


Simulation Setup



- In order to verify energy models, we simulated for an inverter with $C_{L,GND}$ and $C_{L,VDD}$.

Simulated Waveforms



Simulation Results (1): % Error vs V_S

V_S [V]	% error					
	Model 1		Model 2		Model 3, 4	
	C_L	Energy	C_L	Energy	C_L	Energy
2.3	-	-20.98	6.01	5.63	0.01	-0.35
2.4	-	-13.94	6.03	5.65	0.03	-0.33
2.5	-	-6.70	5.93	5.55	-0.06	-0.42
2.6	-	0.98	6.01	5.63	0.01	-0.35
2.7	-	8.92	6.03	5.65	0.03	-0.33
2.8	-	17.15	6.05	5.67	0.05	-0.31

- Model 1 is very sensitive to V_S variation.
- Model 3 and 4 provide small errors.

Simulation Results (2): % Error vs C_M

C_M [nF]	% error					
	Model 1		Model 2		Model 3, 4	
	C_L	Energy	C_L	Energy	C_L	Energy
2	-	-8.42	9.78	9.38	-0.19	-0.55
3	-	-2.03	7.47	7.08	-0.03	-0.39
4	-	0.98	6.01	5.63	0.01	-0.35
5	-	2.72	5.08	4.70	0.07	-0.29
6	-	3.82	4.41	4.04	0.12	-0.24
7	-	4.52	3.86	3.49	0.11	-0.25

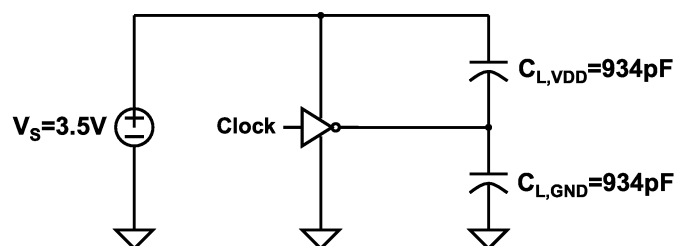
- Model 1 and 2 are very sensitive to C_M variation.
- Model 3 and 4 provide small errors.

Measurement Results: % Error vs C_M

C_M [nF]	% error					
	Model 1		Model 2		Model 3, 4	
	C_L	Energy	C_L	Energy	C_L	Energy
3.9	-	-10.0	8.2	8.2	0.8	0.8
5.3	-	-6.1	7.7	7.7	0.2	0.2
6.8	-	-5.3	5.4	5.4	0.5	0.5
8.2	-	-4.5	4.9	4.9	0.3	0.3
10	-	-3.8	3.5	3.5	0.5	0.5
22	-	-2.6	1.4	1.4	0.5	0.5

- Using our measurement system, we measured for an inverter with $C_{L,GND}$ and $C_{L,VDD}$.
- As the simulation results, model 3 and 4 provide small errors.

Comparison with Ammeter (1)



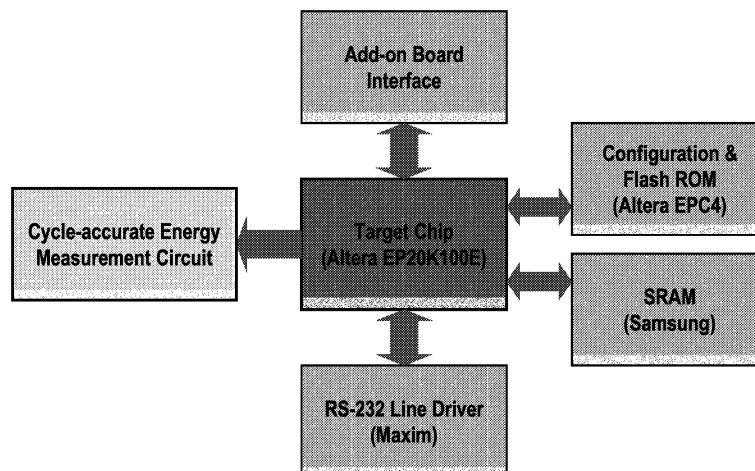
- In order to compare our measurement system with the ammeter, we measured an inverter with $C_{L,GND}$ and $C_{L,VDD}$.
- Theoretical $E_{\text{consumed}} = 11.441 \text{ nJ}$

Comparison with Ammeter (2)

	Theory	Ammeter	Our measurement system		
			Model 1	Model 2	Model 3, 4
Energy [nJ]	11.441	11.514	11.010	11.840	11.497
% error	-	0.63	-3.77	3.49	0.49

- Note that although ammeter shows small error but it cannot measure the cycle-accurate energy.

FPGA Test Setup



FPGA: Energy Consumed by Logic Operation

- If the circuits in FPGA are composed of rising edge triggered logic, we can write as follows

$$E_{rising} \cong E_{logic} - E_{clock}$$

$$E_{falling} \cong E_{clock}$$

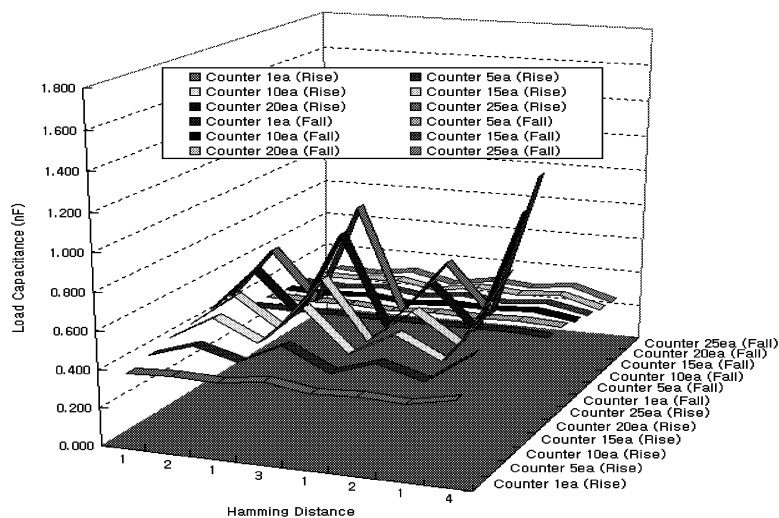
- Where

- E_{logic} : energy consumed by logic operation
- E_{clock} : energy consumed by clock tree circuit

- Then we can write

$$E_{logic} \cong E_{rising} - E_{falling}$$

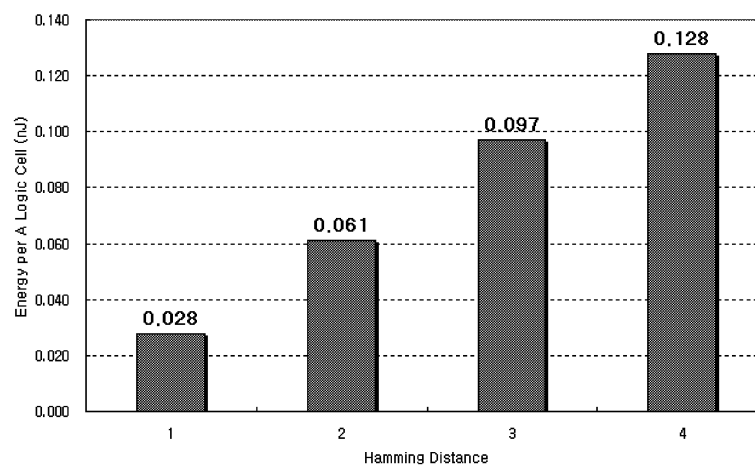
FPGA: 4-Bit Counter Test Results (1)



FPGA: 4-Bit Counter Test Results (2)

Case	Logic Cell Number	Energy Type	Energy by Hamming Distance (nJ)			
			1	2	3	4
Counter 1ea	4	Energy.rise	3.38	3.49	3.67	3.81
		Energy.fall	3.25	3.25	3.21	3.15
		Energy.logic	0.13	0.24	0.46	0.65
Counter 5ea	16	Energy.rise	3.71	4.33	4.75	5.22
		Energy.fall	3.22	3.19	3.14	3.09
		Energy.logic	0.48	1.15	1.61	2.14
Counter 10ea	34	Energy.rise	4.02	5.10	5.96	6.71
		Energy.fall	3.19	3.13	3.08	3.00
		Energy.logic	0.83	1.97	2.88	3.71
Counter 15ea	46	Energy.rise	4.15	5.62	6.77	7.79
		Energy.fall	3.15	3.08	3.01	2.93
		Energy.logic	1.00	2.54	3.76	4.85
Counter 20ea	61	Energy.rise	4.27	6.35	8.41	10.14
		Energy.fall	3.10	3.02	2.95	2.87
		Energy.logic	1.18	3.33	5.46	7.27
Counter 25ea	76	Energy.rise	4.43	6.85	9.23	11.38
		Energy.fall	3.06	2.95	2.87	2.79
		Energy.logic	1.37	3.89	6.36	8.59

FPGA: 4-Bit Counter Test Results (3)



□ Elogic vs Hamming distance plot



Conclusion

How to break software

James Whittaker

The Center for Information Assurance at Florida Tech

How to Break Software

Brought to you by:
The Center for Information Assurance (CIA) at Florida Tech

Your host today:
James A. Whittaker, Ph.D.
Professor of Computer Science

Users Use, Testers Test

- Any fool can stumble across bugs
- Testing requires:
 - *efficiency*: finding bugs faster than normal use
 - *effectiveness*: finding bugs that users care about and that developers will fix
 - *thoroughness*: leaving no stone unturned

Where are the Bugs?

- To find them you could:
 - Seek out the weak developers!
 - Seek out the bad managers!
 - Seek out the doomed projects!
 - Create them yourself!
- Understanding where bugs are requires that we understand *how and why software fails*

Software Fault Models

- Fault models can be based on:
 - Process maturity
 - Programming language constructs
 - Software behavior
- How do we understand problematic behavior?
 - Read bug reports
 - Recognize patterns of failure

Where Bugs Come From

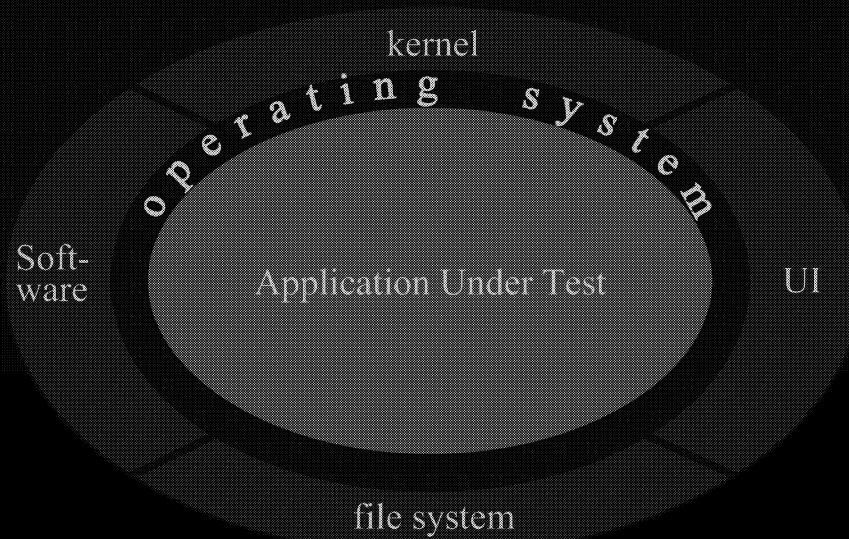
➤ Environment

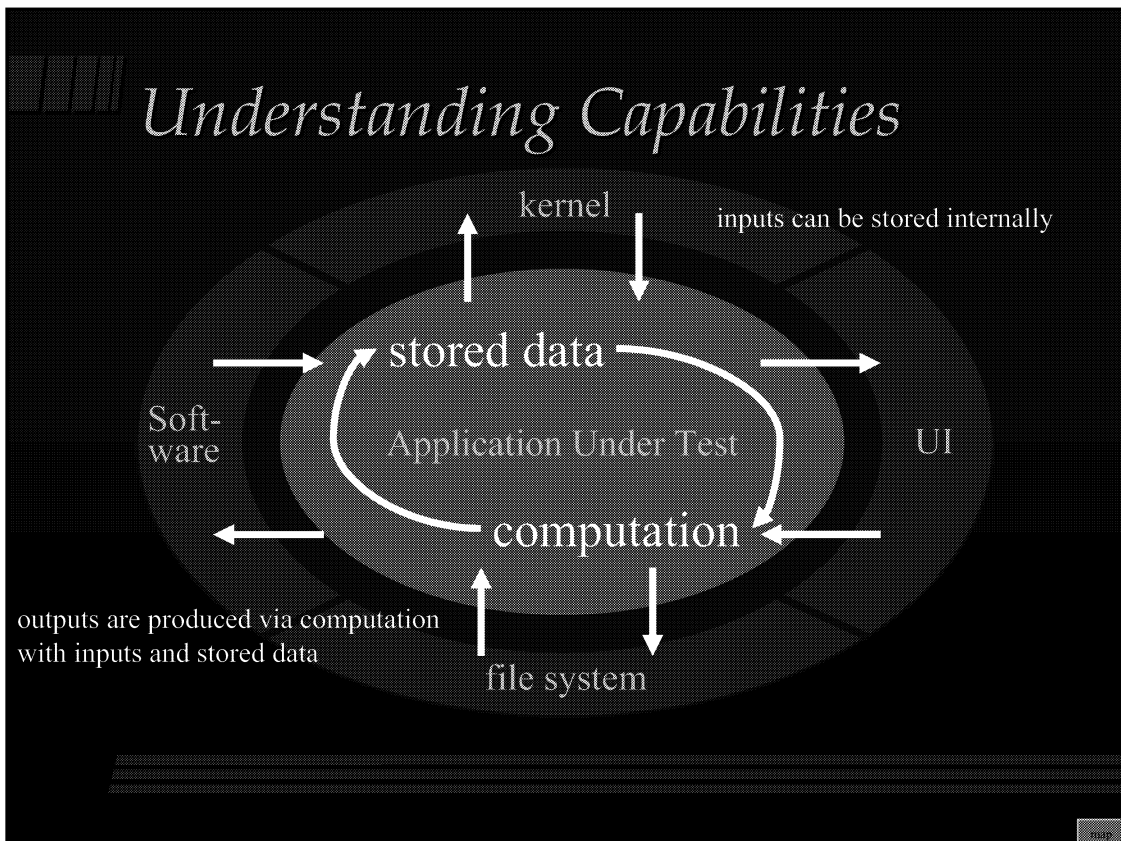
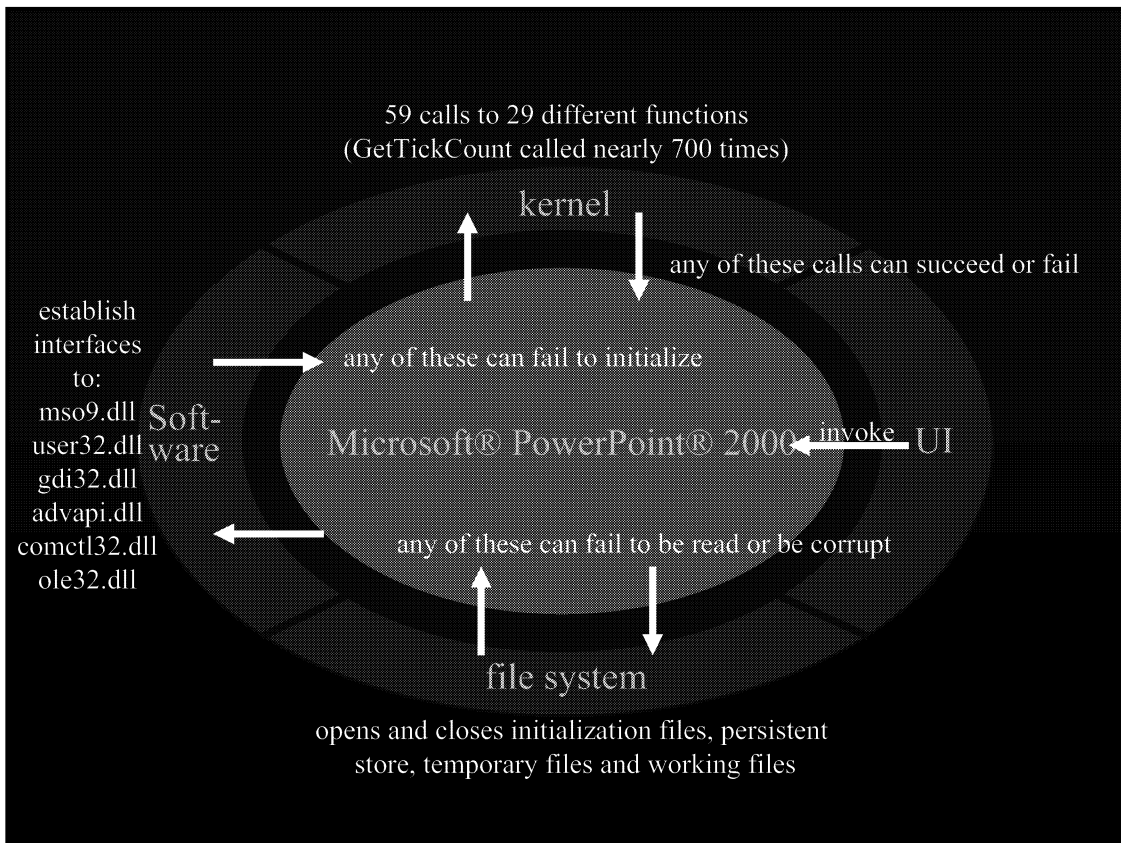
- The software misinterprets or cannot handle its environment
- What are all the environmental considerations that we must face?

➤ Capabilities

- The software incorrectly performs one or more of its capabilities
- What are software's general capabilities?

Understanding the Environment





Software Capabilities

- Although there are only four basic capabilities:
 - accepting input
 - producing output
 - storing data
 - performing computation
- Software can combine these capabilities to perform very complicated tasks

Testing Software Capabilities

- Confronting such complexity in a massive frontal testing assault is often unproductive
- Instead, exploratory testers wage small (winnable) battles until the enemy submits

Testing is War

- The enemy: bugs in the software
- Bugs prevent capabilities
- Drive capabilities and you find bugs
- Method:
 - » Determine your enemy's strengths and remove them
 - » Wage small wars on input capabilities, output capabilities, data storage and computation

Testing Input

- Software should only accept input that it can handle...

... But ensuring that this is the case is problematic

Testing Input

➤ How does software filter erroneous input?

- GUIs

1

- by preventing input data of incorrect type
- by preventing input data that is too small or too large
- by forcing the user down specific control flow paths

emap

Testing Input

➤ How does software filter erroneous input?

- Error checking code

2

- “if” statements
- by ensuring that inputs received can actually be processed
- but error checking code can also have errors!
 - writing error code might introduce errors
 - writing error code also means diverting your attention from the main-line code

emap

Testing Input

- How does software filter erroneous input?

3

- Exception handlers
 - failing gracefully is extremely difficult
 - error routines must reset state and cleanup side-effects...a very difficult endeavor

emap

Testing Output

- Software should generate only those outputs that are acceptable to its users
 - Displayed data must fit in its display area
 - Software cannot pass incorrect data types
 - Data must be correctly computed, software must *never* pass incorrect values to its users
 - Testing output requires domain expertise
- We must understand wrong answers and ensure that our software does not produce them

emap

Testing Data

- Inputs and computation results are often stored internally
- Software will fail if it stores illegal data
- Stored data values must be acceptable individually and in combination with other data

Testing Data

- The major difference between data and inputs is that data is *persistent*
- Persistence needs to be tested
 - data retrieval, data modification, data access
 - we must test that data structures can be operated on without failure
 - » accessed, retrieved, modified, overflowed, underflowed, ...

Testing Computation

- Software can correctly filter inputs, validate outputs and store data...
... and still fail
- $x=x+1$ will fail if it is executed enough times to overflow the value x
- Correct computation depends on operators, operands *and* result

Testing Computation

- Another aspect of computation is feature interaction
 - Features can interact in ways that affect computation
 - One feature can *get in the way* of another feature's computation

An Overview of the Methodology

- Software possesses 4 basic capabilities
- Attack each capability by staging situations that commonly cause failure
 - input attacks
 - output attacks
 - data attacks
 - computation attacks
- Determine which attacks apply to your app and apply them, one-at-a-time

emap

Exploring the Input Domain

- Banging on the keyboard is largely a waste of time, a strategy for rookies
- Each test should have a specific *purpose*
- A tester with clear goals is more likely to find a problem than a tester who is simply hacking away
- Testers must learn to target problematic input scenarios

emap

Exploring Inputs

1

- Explore the application under test
 - Apply the inputs that a user would apply to get real work done
- Observe the inputs
- Watch for attack opportunities

imap

Preview of Input Attacks

1. Force all error messages to occur
2. Force the software to establish default values
3. Explore allowable character sets and data types
4. Overflow input buffers
5. Find inputs that interact with other inputs
6. Repeatedly apply the same input/input sequence

imap

Documenting an Attack

- When to apply the attack
- What faults make the attack successful
- How to determine if the attack exposes failures
- How to conduct the attack
- Example and analysis

imap

First Attack:

Force Error Messages to Occur

- When to apply this attack:
 - Whenever an application must respond to erroneous input
 - Testers should consider the type of response:
 - » Input filter
 - » Input checking
 - » Exception handling

Attack 1

imap

Force Error Messages to Occur

- What faults make this attack successful:
 - Programming error cases is difficult
 - Additional code must be written – where there is code, there is often bad code
 - Writing error code takes the programmers attention away from writing functional code
 - Failing gracefully is difficult
 - What data needs to be saved?
 - What state is the app in?

Attack 1

imap

Force Error Messages to Occur

- How to determine if the software fails:
 - This attacks finds
 - Missing error cases
 - Nonsense error messages
 - Uninformative error messages
 - Thus, manifestation of the defect ranges from crash to fully functional code

Attack 1

imap

Force Error Messages to Occur

- How to conduct this attack:
 - Look thru the (ahem) specification for message definition
 - Consider *properties* of inputs
 - [type] entering invalid types often causes error messages
 - [length] a few too many characters in an input string will often elicit an error message
 - [boundary values] often represent special cases

Attack 1

map

Force Error Messages to Occur

- Target: Word® 2000
- Feature: Insert Index
- Reproducibility: Works on all PC versions of Word® and all versions of Windows® 9x, NT, 2000
- Synopsis: Probably a duplicated block of code, very sloppy programming

Attack 1: Example

map

Force Error Messages to Occur

Questions before we move to the next attack?

Attack 1: Example

Second Attack: Force Default Values

➤ When to apply this attack:

- All software uses variables
- The lifecycle of a variable is
 - › Declaration
 - › Initialization
 - › Use
- Variables must be initialized before they are used

Attack 2

Force Default Values

- What faults make this attack successful:
 - Compilers are often good at catching these mistakes
 - But the compilers must be used properly
 - Implicit declaration can often get a programmer into trouble
 - When users skip input fields or leave them blank, defaults must be established in case those variables are used in computation

Attack 2

map

Force Default Values

- How to determine if the software fails:
 - Best case (for ease of verification) is that use of an un-initialized variable causes a memory violation
 - Harder to detect is that a random value gets assigned to the variable
 - Look for garbage characters/"strange things"
 - Incorrect results
 - Too many values/too few values displayed
 - Incorrectly typed data being displayed

Attack 2

map

Force Default Values

- How to conduct this attack:
 - Determine the data that has defaults
 - Look for “options” or “configuration” buttons
 - Consult the source code’s declaration section
 - Force the app to use its defaults
 - Accept any displayed data as defaults
 - Enter a null value, if a value is displayed then delete it
 - Change settings from their default values to a valid value and then back again

Attack 2

map

Force Default Values

- Target: Word® 2000
- Feature: Insert Table of Contents
- Reproducibility: Works on all PC versions of Word® and all versions of Windows® 9x, NT, 2000
- Synopsis: Frankly, I would have difficulty coding this behavior if I *wanted* to!

Attack 2: Example

map

Force Default Values

Questions before we move to the next attack?

Attack 2: Example

Third Attack:

Explore Character Sets/Data Types

➤ *When to apply this attack:*

- Target: variable input
- Special cases based on:
 - Operating system reserved words
 - Programming language reserved words
 - Character set boundaries (ASCII, UNICODE, etc.)
 - spaces, quotation marks, delimiters, etc.

Attack 3

Explore Character Sets/Data Types

- What faults make this attack successful:
 - Special cases require special handling
 - Either:
 - Developers fail to recognize a special case
 - Developers put too much trust in interface controls
 - Developers fail to handle errors properly (we've already discussed that error code is hard to get right)

Attack 3

imap

Explore Character Sets/Data Types

- How to determine if the software fails:
 - Unhandled exceptions cause a system to crash
 - Generalized error handlers often present nonsensical or uninformative messages
 - Watch out for loss-of-state often caused by exception handlers
 - Since we are dealing with character data, watch for rendering problems
 - Watch for "Easter eggs"

Attack 3

imap

Explore Character Sets/Data Types

- How to conduct this attack:
 - Spend some time researching:
 - Operating system, programming language and character set keywords and ranges
 - Study documentation of each of the above
 - Beware outdated keywords

Attack 3

imap

Explore Character Sets/Data Types

- Target: Microsoft® Internet Explorer® v5.5
- Feature: File Open
- Reproducibility: Works only on early or recent versions of IE
- Synopsis: AUX is an old DOS device, device names were not considered when the file open program was created

Attack 3: Example

imap

Explore Character Sets/Data Types

Questions before we move to the next attack?

Attack 3: Example

Fourth Attack: Overflow Input Buffers

➤ When to apply this attack:

- The idea is simple: enter long strings into input fields
- Don't neglect APIs/exposed internal objects
- This is the hacker's choice because many buffer overflows create exploitable failure scenarios

Attack 4

Fourth Attack: Overflow Input Buffers

When to apply this attack:

- This is an important bug because:
 - copy/paste into inputs fields is a fairly common practice
 - Buffer overruns result in crashes, risking data loss and costly rework
- be sure to check with developers to find out tolerance for fixing long string bugs...thousands characters can get ridiculous

Attack 4

map

Overflow Input Buffers

What faults make this attack successful:

- Developers simply fail to constrain the amount of text the software will accept in an input sting
- When the text is read input memory, fixed-sized buffers are overflowed

Attack 4

map

Overflow Input Buffers

- How to determine if the software fails:
 - This bug almost always causes the software to crash
 - Other possibilities are extreme application instability (since memory is often overwritten by the long string)

Attack 4

imap

Overflow Input Buffers

- How to conduct this attack:
 - Identify where strings are read as input
 - Start small and then grow the string to its maximum length
 - This attack is very repetitive
 - It's helpful to count 12345678901234567890

Attack 4

imap

Overflow Input Buffers

- Target: Word® 2000
- Feature: Find/Replace
- Reproducibility: Works on all PC versions of Word® and all versions of Windows® 9x, NT, 2000
- Synopsis: Find field is string-length constrained but the Replace field is not

Attack 4: Example

map

Overflow Input Buffers

Questions before we move to the next attack?

Attack 4: Example

map

Fifth Attack: Find Interacting Inputs

- Up to now, we have dealt with inputs one-at-a-time
 - Find a spot where input is accepted and poke it until something breaks
- This next attack deals with *combinations* of inputs
 - Multiple inputs on a single input dialog
 - An API call with more than one input
- Constraining a single input is hard enough for developers...

Attack 5

map

Fifth Attack: Find Interacting Inputs

- When to apply this attack:
 - Some inputs affect other inputs
 - They might represent different properties of the same piece of data
 - They might be used in a single internal computation
 - Thus, individually correct inputs might be problematic when combined
 - These relationships should be watched for while executing the previous attacks

Attack 5

map

Find Interacting Inputs

- What faults make this attack successful:
 - Obviously, input relationships are hard to determine for both testers and developers
 - The logic involved in handling a single erroneous input is hard enough...
 - ...multiple error cases often require complex nested IF statements
 - Code changes make this situation worse

Attack 5

map

Find Interacting Inputs

- How to determine if the software fails:
 - Since inputs are often stored internally, slipping a bad input in means corruption of internal data
 - Once you suspect you tricked the app into accepting bad input, force that input to be used as much as possible
 - Carefully verify it every time it is displayed or used

Attack 5

map

Find Interacting Inputs

- How to conduct this attack:
 - Explore the app, identify possible relationships between inputs
 - Are they properties of a single data structure?
 - Are they used together in a single computation?
 - Select boundary and extreme combinations:
 - Large/small, small/large, large/large, small/small

Attack 5

imap

Find Interacting Inputs

- Target: Word® 2000
- Feature: Insert a Table
- Reproducibility: Works on all PC versions of Word® and all versions of Windows® 9x, NT, 2000
- Synopsis: These two parameters can be small-large, large-small, small-small but not large-large

Attack 5: Example

imap

Find Interacting Inputs

Questions before we move to the next attack?

Attack 5: Example

Sixth Attack:

Repeat Inputs Numerous Times

➤ When to apply this attack:

- This attack is applicable whenever the app accepts input inside a loop:

Accept an input

Process it

Repeat

Attack 6

Repeat Inputs Numerous Times

- What faults make this attack successful:
 - Repetition has the effect of gobbling up resources
 - Many applications are unaware of available resources and assume unlimited memory and storage
 - Low resources can cause undesirable side-effects

Attack 6

imap

Repeat Inputs Numerous Times

- How to determine if the software fails:
 - It is difficult to predict how memory stress will manifest
 - Watch for:
 - Screen refresh problems
 - Slow performance
 - Consider:
 - Using a memory leak detector

Attack 6

imap

Repeat Inputs Numerous Times

- How to conduct this attack:
 - Pick an input or a sequence of inputs
 - Apply them over and over to test for undesirable side-effects
- Or:
 - Pick an object and apply the same input to it over and over
 - Pick multiple objects and apply the same series of inputs to each object

Attack 6

imap

Repeat Inputs Numerous Times

- Target: Word®
- Feature: Equation Editor
- Reproducibility: Works on all PC versions of Word® and all versions of Windows® NT, 2000
- Synopsis: The number of parentheses is constrained at 10, but that number should, perhaps, be lowered

Attack 6: Example

imap

Repeat Inputs Numerous Times

Questions before we move to the next attack?

Attack 6: Example

Review of Input Attacks

1. Force all error messages to occur
2. Force the software to establish default values
3. Explore allowable character sets and data types
4. Overflow input buffers
5. Find inputs that interact with other inputs
6. Repeatedly apply the same input/input sequence

Exploring Outputs

2

- Some bugs are too difficult to find by concentrating on inputs alone
- Which inputs will generate incorrect results?
- Why not concentrate on what incorrect results *could* occur and then find the inputs to force them?

Preview of Output Attacks

7. Force different outputs to be generated for each input
8. Force invalid outputs to be generated
9. Force output properties to change
10. Force the screen to be refreshed

Seventh Attack:

Vary Outputs for Each Input

➤ When to apply this attack:

- Inputs are not independent of each other
- Applying some inputs before (or after) other inputs can affect behavior
- A good way to think about the most important cases is to ensure that you see every output that an input can cause

Attack 7

map

Vary Outputs for Each Input

➤ What faults make this attack successful:

- Input → Output is simple to code
- Inputs that cause different outputs require complex logic to be coded
- Complexity leads to bugs

Attack 7

map

Vary Outputs for Each Input

- How to determine if the software fails:
 - The main bug is that there are behaviors that the developer miscoded or forgot to code
 - These are usually severity 1 problems that get found and fixed before release

Attack 7

map

Vary Outputs for Each Input

- How to conduct this attack:
 - Testers must identify which inputs can cause multiple behaviors and understand the *context* in which these inputs can be applied

Attack 7

map

Vary Outputs for Each Input

- To illustrate, consider a telephone switch
- How many outputs can the input “pick up the receiver” cause?
 - The switch could generate a dial tone ...
... when the phone is idle
 - The switch could connect two callers...
... when the phone is ringing

these
are the
situations
to test!

Attack 7: Example

imap

Vary Outputs for Each Input

Questions before we move to the next attack?

Attack 7

imap

Eighth Attack: Force Invalid Outputs

When to apply this attack:

- Domain expertise is essential to effectively carry out this next attack
 - » It's hard to test a flight simulator without knowing how to fly an airplane
- Testers must know the product well enough to enumerate wrong answers...
- ...and then figure out how to drive the application to produce them

Attack 8

map

Force Invalid Outputs

What faults make this attack successful:

- Just as testers can misunderstand the problem domain...so can developers
 - » When this happens, they write bugs
- In most cases, the problem is over-looked special cases

Attack 8

map

Force Invalid Outputs

➤ How to determine if the software fails:

- This is a hard one...the software rarely fails in a spectacular (or even noticeable) manner
- Testers should ignore *type* and *format* and concentrate instead on the *value* being displayed

Attack 8

map

Force Invalid Outputs

➤ How to conduct this attack:

- Testers need to focus on *known bad results*
 - Enumerate wrong answers and then create the situation that forces them
- Learn as much about the problem domain as you can

Attack 8

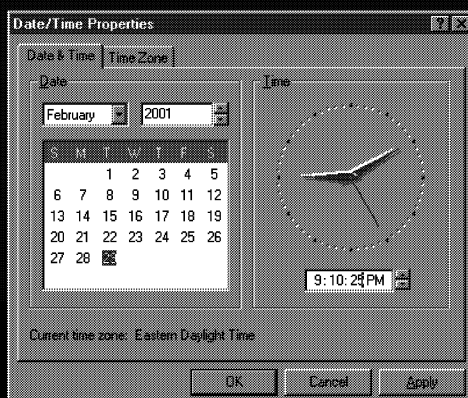
map

Force Invalid Outputs

- Target: Windows NT System Clock
- Feature: Calendar
- Reproducibility: Works on Windows® NT, Service Pack 4 and prior
- Synopsis: The calendar works fine as long as the cursor isn't fixed on day 29 of February

Attack 8: Example

Force Invalid Outputs



Windows NT4 Clock

The bug has since been fixed!

- Choose Month as February
- Choose Year as 2000
- Click on the 29th
- Change Year to 2001 (using spin control)

VOILA! We have a whole new calendar!

Attack 8: Example

Force Invalid Outputs

Questions before we move to the next attack?

Attack 8: Example

Ninth Attack: Change Output Properties

➤ When to apply this attack:

- This attack requires *persistent* outputs be present in the application (which often precludes its use on APIs)
- An output is generated and then we change some property of the output

Attack 9

Change Output Properties

- What faults make this attack successful:
 - Generating the output once tests that the software works with initial data settings
 - These are settings that the developer established
 - These are settings that the developer has anticipated
 - Changing the output ensures that the software will work user-defined setting
 - These settings might not be anticipated

Attack 9

imap

Change Output Properties

- How to determine if the software fails:
 - Since outputs are necessarily visually-intensive, this often requires manual verification
 - But since testers have had to determine output properties in advance, it is easy to determine what to look for

Attack 9

imap

Change Output Properties

- How to conduct this attack:
 - First determine the property of interest
 - Size, value, type, color, shape, direction, ...
 - Cause it to be displayed, then change it
 - Size (bigger to smaller, smaller to bigger,...)
 - Value(s) (high to low, positive to negative,...)
 - Type (char to int, int to float,...)
 - ...

Attack 9



Change Output Properties

- Target: PowerPoint® 2000
- Features: Insert WordArt
- Reproducibility: Works on all PC versions of PowerPoint® and Word® and all versions of Windows®
- Synopsis: First time through, two things happen; next time through, something is forgotten

Attack 9: Example



Change Output Properties

Questions before we move to the next attack?

Attack 9: Example

Tenth Attack:

Force the Screen to Refresh

➤ *When to apply this attack:*

- This attack is applicable only to GUI software with editable display areas
- The attack is most useful at the boundaries of screen objects
 - » Objects created by the user
 - » Partitions of the display area (frames, etc.) set by the software

Attack 10

Force the Screen to Refresh

- What faults make this attack successful:
 - Refreshing the contents of a window, after those contents have changed, is problematic
 - Refresh too often:
 - » Performance degrades
 - » The screen flickers to annoy the user
 - Refresh too seldom
 - » The screen becomes messy
 - Hitting the sweet spot is challenging

Attack 10

map

Force the Screen to Refresh

- How to determine if the software fails:
 - Sigh...you guessed it...labor-intensive visual verification is the only resort

Attack 10

map

Force the Screen to Refresh

- How to conduct this attack:
 - Add things to the screen
 - Move them around (varying the distance you move them)
 - Delete them
 - Edit them
- Do all these things with a mix of features and at or around object boundaries

Attack 10

Force the Screen to Refresh

- Target: Word® 2000
- Features: Entering text at the page boundary
- Reproducibility: Works on Word® 2000 on any OS. The problem is much worse on prior versions of Word
- Synopsis: Removal of a similar bug in Word 97 caused this one

Attack 10: Example

Force the Screen to Refresh

Questions before we move to the next attack?

Attack 10: Example

Review of Output Attacks

7. Force different outputs to be generated for each input
8. Force invalid outputs to be generated
9. Force output properties to change
10. Force the screen to be refreshed

Exploring Stored Data

3

- While executing the input and output attacks, keep notes on persistent (stored) data
 - Data you see over and over is stored
 - Data you see over multiple executions is persistent
- Think about where this data comes from and how it can get corrupted

Exploring Stored Data

- If you have the source code, data is easy to find
- If you don't have the source, you must be able to look through the interface and identify data
 - Put yourself in the shoes of the developer
 - How would you program it?

Preview of Stored Data Attacks

11. Apply inputs using a variety of initial conditions
12. Force a data structure to store too many/too few values
13. Investigate alternate ways to modify internal data constraints

map

Eleventh Attack: Vary Initial Conditions

➤ When to apply this attack:

- Inputs are often applicable in a variety of circumstances (states of the software)
- This attack investigates the application of inputs from a variety of initial states
- Try to pick states that might cause errant behavior
 - Pick states that are as different from each other as possible
 - Pick states that differ by only one data element

Attack 11

map

Vary Initial Conditions

- What faults make this attack successful:
 - Checking that inputs are valid is tricky
 - Checking that inputs – combined with internal state – are valid is even harder
 - The result is that some inputs only work on specific initial conditions

Attack 11

map

Vary Initial Conditions

- How to determine if the software fails:
 - Since some cases of this bug mean missing code (failure to consider a specific case)
 - › Watch for crashes
 - › Watch for nonsense error messages

Attack 11

map

Vary Initial Conditions

➤ How to conduct this attack:

- Enumerating states is a time consuming endeavor:
- Does an input cause different behavior in different circumstances?
- Is an input applicable sometimes but not other times?

For more information read about *model-based testing*
visit: www.model-based-testing.org

Attack 11

map

Vary Initial Conditions

- Target: Word® 2000
- Features: Draw Group/Ungroup
- Reproducibility: Works on all PC versions of Word® and all versions of Windows® 9x, NT, 2000
- Synopsis: Certain configurations of grouped objects do not ungroup cleanly

Attack 11: Example

map

Vary Initial Conditions

Questions before we move to the next attack?

Attack 11: Example

Twelfth Attack:

Over/Underflow Data Structures

➤ When to apply this attack:

- Fixed size structures can be overflowed by forcing the software to store too many values
- Testers must be able to detect size constraints and then find ways to violate them

Attack 12

Over/Underflow Data Structures

- What faults make this attack successful:
 - Developers are often good at checking content and failing to consider size
 - Every program that adds or removes elements from a structure must have code to check that size constraints are not broken as a result of the add/remove operation
 - It only takes one program to forget...

Attack 12

imap

Over/Underflow Data Structures

- How to determine if the software fails:
 - Reading or writing beyond the bounds of a data structures almost always causes the software to crash
 - But be alert for the usual signs of data corruption (performance, anomalous output/computation, etc.)

Attack 12

imap

Over/Underflow Data Structures

- How to conduct this attack:
 - A structure can be underflowed by removing more values than the structure contains
 - Strategy:
 - » add x number of values to a structure
 - » delete $x+1$ values from the structure
 - Try this on internal structures and also list boxes that store persistent data

Attack 12

map

Over/Underflow Data Structures

- Target: Word® 2000
- Features: Tables
- Reproducibility: Works on all PC versions of Word® and all versions of Windows® 9x
- Synopsis: failure to constrain the upper bound on the number of rows

Attack 12: Example

map

Over/Underflow Data Structures

Questions before we move to the next attack?

Attack 12: Example

Thirteenth Attack: Find Back Doors to Stored Data

➤ When to apply this attack:

- Whenever data is supposed to be constrained in any way:
 - A month must fall between 1 and 12
 - A year must fall between 1980 and 2095
 - The limit on undo operations is 20
 - ...

Attack 13

Find Back Doors to Stored Data

- What faults make this attack successful:
 - Plain and simple: a lack of good software engineering practice
 - Information hiding (object-orientation) cures this problem:
 - » All data is private to a set of access routines
 - » These access routines enforce data constraints
 - » A program must use the access routines to access or modify data

Attack 13

imap

Find Back Doors to Stored Data

- How to determine if the software fails:
 - Broken data constraints are serious, often resulting in system instability or crash
 - Also, look for:
 - » System sluggishness
 - » Incorrect error messages
 - » Incorrectly computed results

Attack 13

imap

Find Back Doors to Stored Data

- How to conduct this attack:
 - Identify data structures
 - Explore various ways to modify the contents or properties of the data structure

Attack 13

emap

Find Back Doors to Stored Data

- Target: PowerPoint® 2000
- Features: Insert Table
- Reproducibility: Works on all PC versions of PowerPoint® and all versions of Windows® 9x, NT, 2000
- Synopsis: Table size is constrained at creation but can be expanded on edit

Attack 13: Example

emap

Find Back Doors to Stored Data

Questions before we move to the next attack?

Attack 13: Example

Review of Stored Data Attacks

11. Apply inputs using a variety of initial conditions
12. Force a data structure to store too many/too few values
13. Investigate alternate ways to modify internal data constraints

Exploring Computation

4

- While executing the input and output attacks:
 - make a feature list
 - keep notes on what computation is happening inside the application
- Think about ways to “get in the way” of the computation, using inputs, outputs, data, or even other features

Preview of Computation Attacks

14. Experiment with invalid operand and operator combinations
15. Exploit recursion
16. Force computation results to be too large or too small
17. Find features that share data or interact poorly

Fourteenth Attack: Experiment w/ Operator/Operands

➤ When to apply this attack:

- Some operands cannot be used with certain operators
 - Divide by zero
 - Adding a character to a real number
- Some operators conflict with each other
 - Operators with an inverse relationship
 - Ex: $[\text{SQRT}(x)]^2$

Attack 14

map

Experiment w/ Operator/Operands

➤ What faults make this attack successful:

- Developers have to write error code to weed out invalid operator/operand combinations
- Consider divide-by-zero

```
if (x != 0)
    y = z / x;
else
    cout("cannot divide by zero");
```

map

Experiment w/ Operator/Operands

- How to determine if the software fails:
 - Become a problem domain expert to learn about the domain and range of the functions computed by your software
 - Apply invalid combinations and watch for:
 - » Crashes (for unhandled exceptions)
 - » Incorrect results
 - » Null or nonsensical values appearing as output

Attack 14

map

Experiment w/ Operator/Operands

- How to conduct this attack:
 - Must be able to identify computation (operators) and the data objects (operands) on which the computation operates
 - Computation isn't just assignments
 - » Graphical rendering
 - » Screen layout, WYSIWYG

Attack 14

map

Experiment w/ Operator/Operands

- Target: Windows® Calculator
- Features: Square Root/Exponent
- Reproducibility: Works on all versions of Windows® 9x, NT, 2000
- Synopsis: Round-off error

Attack 14: Example

map

Experiment w/ Operator/Operands

Questions before we move to the next attack?

Attack 14: Example

map

Fifteenth Attack: Exploit Recursion

➤ When to apply this attack:

- Recursion occurs when a function calls itself

```
long int factorial(long int n)
{
    if (n <= 1)
        return(1);
    else
        return(n * factorial(n - 1));
}
```

- If this occurs too many times, stack overflow will occur

Attack 15

map

Exploit Recursion

➤ What faults make this attack successful:

- Developers fail to write code that ensures loop or recursive call termination
- Since recursion only requires one line of code, constraining it often seems unnecessary to developers

Attack 15

map

Exploit Recursion

- How to determine if the software fails:
 - Infinite recursion causes a heap overflow in the computer's memory
 - This will almost always result in the application crashing or hanging

Attack 15

imap

Exploit Recursion

- How to conduct this attack:
 - Recursion can be thought about in terms of black box objects too:
 - Can a document reference itself?
 - Can a hyperlink point to itself?
 - Can an email message contain itself?
 - Can a program that spawns processes call itself?
 - Determine these objects and create the self-referencing situation

Attack 15

imap

Exploit Recursion

- This demo will show how such a program behaves on Windows 2000...
- ...However, we will wait until the end of the session because the computer will not be usable after this program executes

Attack 15: Example

map

Exploit Recursion

Questions before we move to the next attack?

Attack 15: Example

map

Sixteenth Attack:

Force Results to be too Large/Small

➤ When to apply this attack:

- Review:
 - Inputs need to be constrained
 - an app should never permit invalid input to enter the system
 - Data needs to be constrained
 - an app should never store invalid data
- But even if all inputs and data are correct, computation can still fail

Attack 16



Force Results to be too Large/Small

➤ What faults make this attack successful:

```
x=x+1;
```

- This simple line of code will fail if it is executed enough times
- Developers often focus on operators and operands and ignore the result

Attack 16



Force Results to be too Large/Small

- How to determine if the software fails:
 - Since the result of this attack is underflow or overflow, the software most often crashes

Attack 16

map

Force Results to be too Large/Small

- How to conduct this attack:
 - Usually, you have to force a computation to occur over and over and over
 - If you have control over the data used in the computation, rig it to begin at or around boundary values

Attack 16

map

Force Results to be too Large/Small

- Given the program:

```
#define count 2
main() {
    int sum, value[count];
    sum = 0;
    for (i = 0; i < count; ++i) {
        sum = sum + value[i];
    }
}
```

- Consider the values:
 - value[0] = 32700, value[1] = 70
 - count = 33000, value[0..32999] = 1

Attack 16: Example

map

Force Results to be too Large/Small

Questions before we move to the next attack?

map

Seventeenth Attack: Feature Interaction

➤ When to apply this attack:

- This is the attack that distinguishes novice testers from the pros
- Given two or more features that work fine individually, can you make them break when they work together
 - on the same data
 - at the same time
 - getting in the way of each other

Attack 17

map

Feature Interaction

➤ What faults make this attack successful:

- Features often work well in isolation because they are developed in isolation
- But two features may share data but require that different constraints on that data be enforced
- Failure to enforce such constraints causes this class of failure

Attack 17

map

Feature Interaction

- How to determine if the software fails:
 - Here is yet another example of painstaking manual verification
 - Keep a sharp eye out for:
 - » Improperly formatted output
 - » Incorrect computation
 - » Corrupted data
 - Failures often manifest long after the fault has been tripped over

Attack 17

imap

Feature Interaction

- How to conduct this attack:
 - Pick two or more features
 - » That might interact with the same user input
 - » That might contribute to computing a single output
 - » That might share the same data
 - Try to make them “get in each other’s way”

Attack 17

imap

Feature Interaction

- Target: Word® 2000
- Features: Footnotes/Dual Columns
- Reproducibility: Works on all PC versions of Word® and all versions of Windows® 9x, NT, 2000
- Synopsis: The footnote adopts the properties of the reference point, which can be incompatible with the page layout

Attack 17: Example

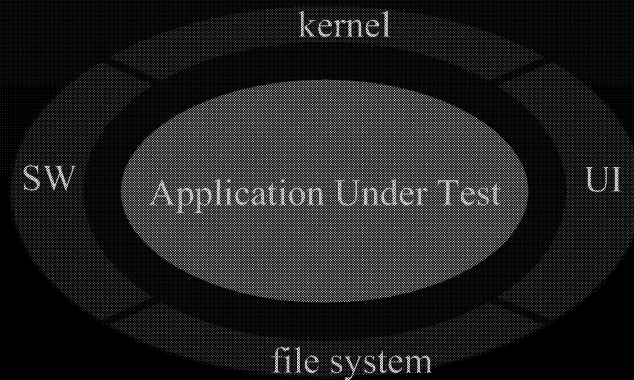
Review of Computation Attacks

14. Experiment with invalid operand and operator combinations
15. Exploit recursion
16. Force computation results to be too large or too small
17. Find features that share data or interact poorly

Summary

- Software testing requires understanding of and control over four interfaces:

- UI
- Kernel
- File System
- Software



Summary

- Each interface presents number of challenging problems
- Testing any of the interfaces is formidable
- Testing them thoroughly is exquisitely difficult
- Some interfaces are well-understood, some we are just beginning to study

Conclusions

➤ Testing each interface:

- The user interface
 - study the attacks
 - execute the attacks
 - meet often, share best practices

1

Conclusions

➤ Testing each interface:

- The kernel interface
 - acquire the tools
 - acquire the knowledge
 - study field bug reports and make sure you understand real stressed environments

2

Conclusions

Testing each interface:

- The file system interface
 - » understand your app's file formats
 - » understand how storage media can fail and determine your customer's tolerance for losing their data

3

Conclusions

Testing each interface:

- The software interface
 - » understand the other apps that your software depends upon
 - » understand the data that your app gets from these resources
 - » determine how this communication can fail
 - » determine if failure of the resource will cause failure of your app

4

Remember

- Know your app's environment
 - Understand what might go wrong and test that it doesn't
- Know your app's capabilities
 - Understand how it can screw things up, test that it doesn't
- Practice the attacks...*always* have a goal
- Brain on, eyes open, Test!

imap

How To Contact James Whittaker

The End

Questions?

Comments?

Suggestions?

Bug Stories?

Snail Mail

Department of Computer Sciences
150 W. University Boulevard
Melbourne, Florida 32901-6975

E-mail & Web

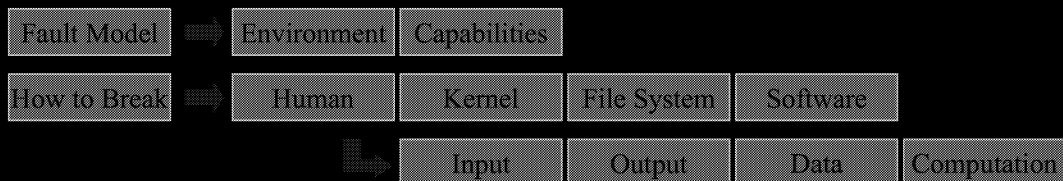
jw@cs.fit.edu
<http://www.se.fit.edu>
<http://www.howtobreaksoftware.com>

Telephone

321- 674 - 7638

Fax

321- 674 - 7046



Sponsored by:





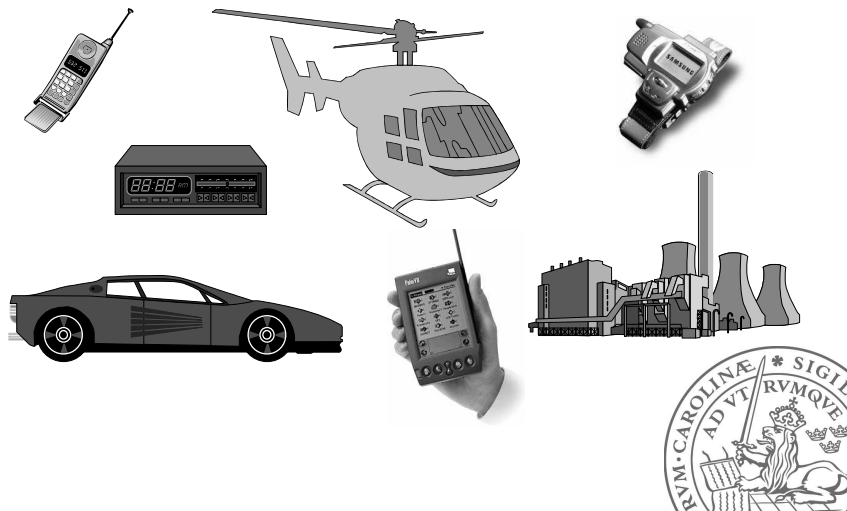
LUND INSTITUTE
OF TECHNOLOGY
Lund University

Design of Embedded Systems

Kris Kuchcinski
Dept. of Computer Science
Lund University
Sweden

<http://www.cs.lth/~kris>

Examples of Embedded Systems





LUND INSTITUTE
OF TECHNOLOGY
Lund University

Constraint Programming Approach

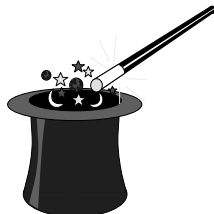
Kris Kuchcinski
Dept. of Computer Science
Lund University
Sweden

<http://www.cs.lth/~kris>

Quotations

"Constraint programming represents one of the closest approaches computer science has yet made to the Holy Grail of programming: the user states the problem, the computer solves it."

Eugene C. Freuder
CONSTRAINTS, April 1997



Introduction and Motivation

Synthesis of the following code
(inner loop of differential equation integrator)

```

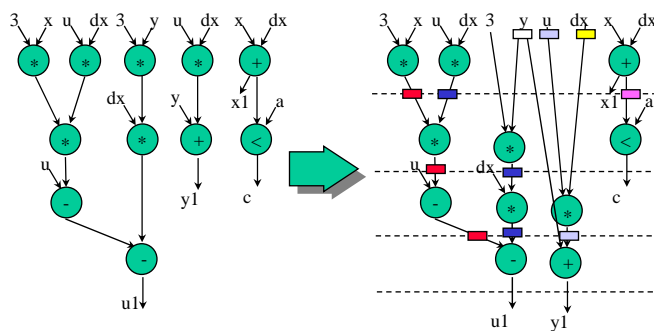
while c do
  begin
    x1 := x + dx;
    u1 := u - (3*x*u*dx);
    y1 := y + u*dx;
    c := x < a;
    x := x1; y := y1; u := u1;
  end;

```



3

Introduction and Motivation



data-flow graph

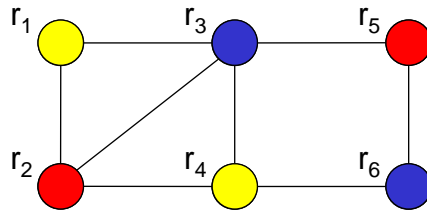
scheduled data-flow graph

register allocation

4



Register Allocation as Graph Coloring



Constraints:

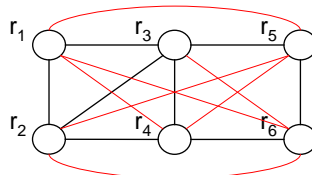
$[r_1, r_2, r_3, r_4, r_5, r_6] :: 0..2,$
 $r_1 \neq r_2, r_1 \neq r_3, r_2 \neq r_3,$
 $r_2 \neq r_4, r_3 \neq r_4, r_4 \neq r_6,$
 $r_5 \neq r_6.$



Register Allocation as Clique Finding

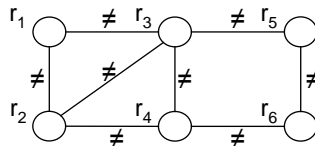
- for all r_i, r_j which are not connected by an edge:
 $r_i \neq 1 \vee r_j \neq 1$
- The maximal clique can be found by maximizing the following cost function:

$$\text{cost} = \sum_i r_i$$



Constraint Consistency

- All constraints are stored in the *constraint store*
- *Consistency* methods are applied to find inconsistent values and prune variables' domains
- Different types of consistency methods:
 - Node consistency
 - Arc consistency
 - Path consistency
 - ...



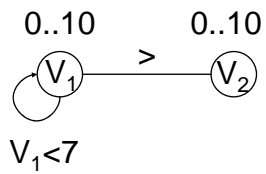
Consistency Properties

- Node consistency
 - A network is node consistent if in each node domain each value is consistent with unary constraint (e.g., $X > 7$)
- Arc consistency
 - A network is arc consistent if for each arc connecting variables V_i and V_j for each value in the domain of V_i there exist a value in the domain of V_j consistent with binary constraint (e.g., $X > Y$)

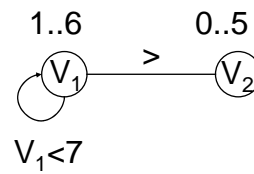


Node and Arc Consistency

- Example



Not node consistent
Not arc consistent

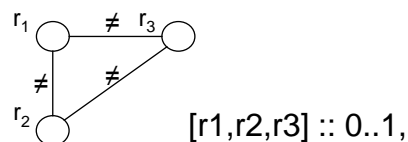


node consistent
arc consistent



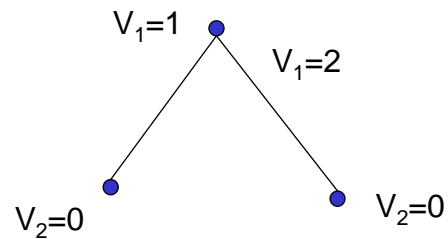
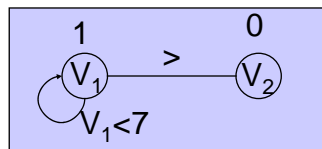
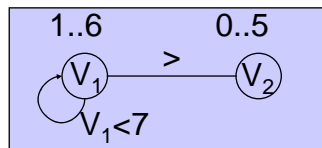
Need for search

- Node, arc and path consistency are in general not complete (complete for some problems with particular structures)
- Complete algorithm: N -consistency for N variable problems \rightarrow exponential complexity
- Example:



Search

- Solver is not complete and search for a solution is needed



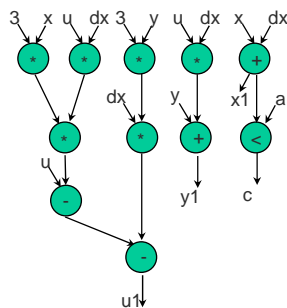
Constraint properties

- may specify partial information — need not uniquely specify the values of its variables,
- non-directional — typically one can infer a constraint on each present variable,
- declarative — specify relationship, not a procedure to enforce this relationship,
- additive — order of imposing constraints does not matter,
- rarely independent — typically they share variables



More realistic example Scheduling

Scheduling of the data-flow graph



Constraints:

- for all op_i and op_j such that op_i before op_j
 $T_i + D_i \leq T_j$
- for all op_i and op_j that can use the same resource
 $T_i + D_i \leq T_j \vee T_j + D_j \leq T_i \vee R_i \neq R_j$



Problems

- Constraint propagation for
 $T_i + D_i \leq T_j \vee T_j + D_j \leq T_i \vee R_i \neq R_j$ is weak
- Not all solvers support disjunctive constraints.
 - Other solution (reified constraints):

$$T_i + D_i \leq T_j \Leftrightarrow B1,$$

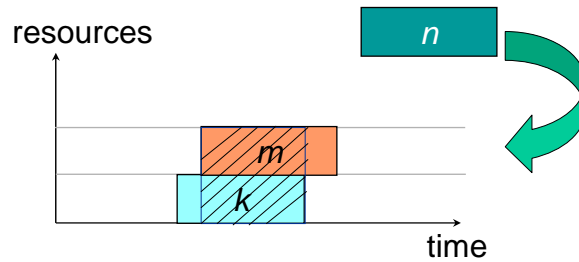
$$T_j + D_j \leq T_i \Leftrightarrow B2,$$

$$R_i \neq R_j \Leftrightarrow B3,$$

$$B1 + B2 + B3 \geq 1.$$



Propagation problems



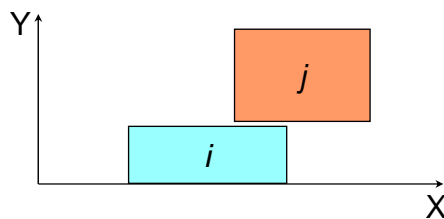
$$T_k + D_k \leq T_n \vee T_n + D_n \leq T_k \vee R_k \neq R_n$$

$$T_m + D_m \leq T_n \vee T_m + D_m \leq T_n \vee R_m \neq R_n$$



Global constraints

- Non-overlapping rectangles



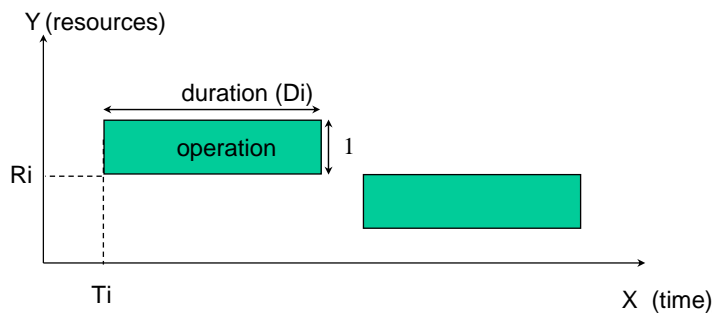
$$\text{diff2}([[X_i, Y_i, DX_i, DY_i], [X_j, Y_j, DX_j, DY_j]])$$

- All knowledge in one "place" – makes it possible to define good consistency methods (OR, mathematics, geometry, etc.)
- Specific algorithms for consistency – more efficient



Global Constraints – Scheduling

- diff2 constraint



diff2([[T1, R1, D1, 1], [T2, R2, D2, 1]], ...)



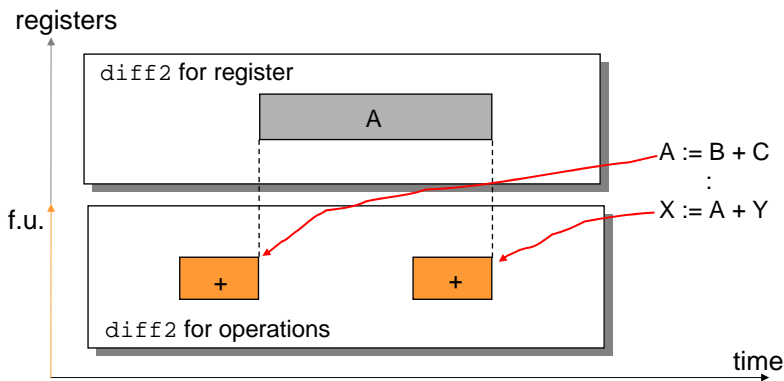
17

Scheduling Example Constraints

$T1 + 2 \leq T6$, $T2 + 2 \leq T6$,
 $T3 + 2 \leq T7$, $T4 + 2 \leq T8$,
 $T5 + 1 \leq T9$, $T6 + 2 \leq T10$,
 $T7 + 2 \leq T11$, $T10 + 1 \leq T11$,
 diff2([[T1,R1,2,1], [T2,R2,2,1], [T3,R3,2,1],
 [T4,R4,2,1], [T6,R6,2,1], [T7,R7,2,1],
 [T5,R5,1,1], [T8,R8,1,1], [T9,R9,1,1],
 [T10,R10,1,1], [T11,R11,1,1]]).



Registers



- can be done together with or after functional units allocation/binding and scheduling,



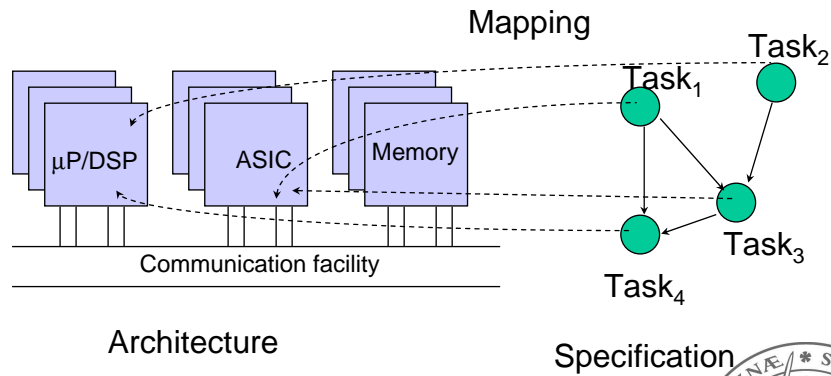
19

Other Synthesis Problems Defined with Constraints

- High-level synthesis:
 - Chaining,
 - Conditional execution,
 - Pipelined components,
 - Algorithmic pipelining,
 - Switching activity reduction (power consumption)
 - ...
- System design
 - different aspects of design space exploration
 - scheduling
 - component assignment
 - memory allocation/data assignment
 - power/energy consumption
 - ...



Design Space Exploration



21

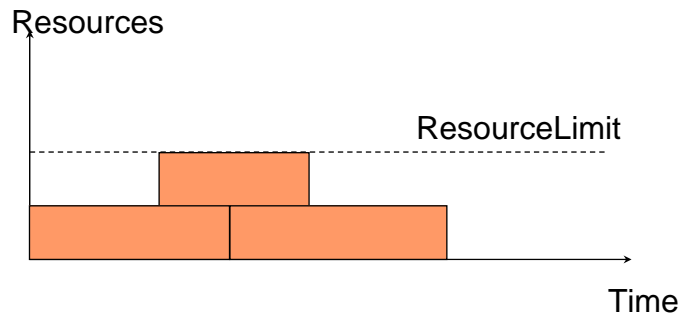
Additional Constraints element

- Element constraints
 - $\text{element}(N, [X_1, X_2, \dots, X_n], \text{Value})$
 - propagation from N to Value
 $N=i \rightarrow \text{Value} = X_i$
 - propagation from Value to N
 $\text{Value} = x \rightarrow N=i \text{ and } X_i = x \dots$
- Examples- $\text{element}(N, [2, 3, 4, 4], V)$
 - $N :: 1..2, V :: \{2, 3\}$
 - $V = 4, N :: 3..4$
- Used to define discrete cost functions and different relations



Additional constraints cumulative

Cumulative constraint

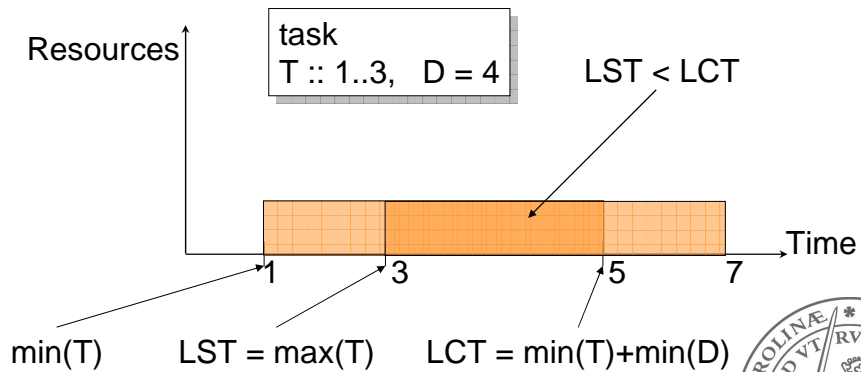


`cumulative([Tk, Tn, Tm], [Dk, Dn, Dm], [1, 1, 1], ResourceLimit)`

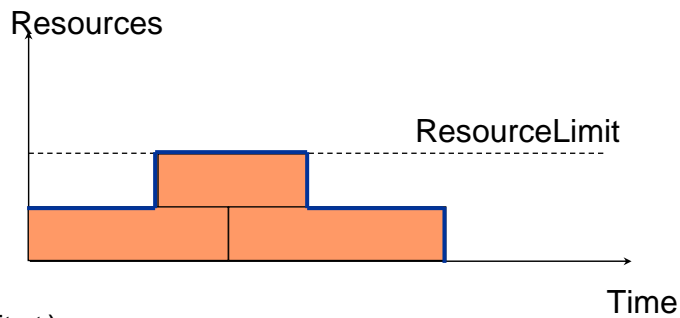


Cumulative propagations

- Execution interval which will always be occupied by a task.



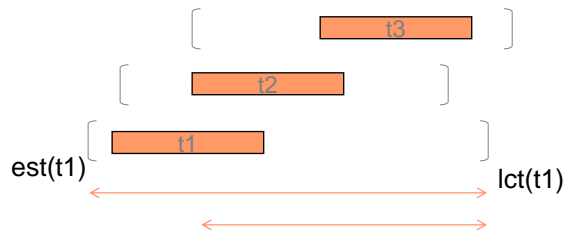
Cumulative propagations — profile based



for each $[t_i, t_j)$
 for each task_n whose exec. interval overlaps with $[t_i, t_j)$
 if $(ResourceLimit - resource_usage < task_n(resources))$
 T_n in $\{ complement(t_i - min(D_n) + 1 .. t_j - 1) \}$
 check $D_n \dots Res_n \dots$



Cumulative propagations — edge finding



t3 cannot be between t1 and t2 iff
 $lct(t1) - est(t1) < D1 + D2 + D3$

t3 cannot be before t1 and t2 iff
 $lct(t1) - est(t3) < D1 + D2 + D3$

t3 must be last !!!

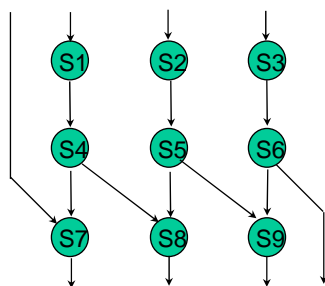


Edge Finding Algorithm

- Martin-Shmoys algorithm with $O(n^2)$ complexity.
- Up phase
 - for each unique lct we create a set $S = \{t \mid LCT(t) \leq lct\}$ and make checking whether a task can be the first or before
- Down phase
 - similar but using est and checking whether a task can be the last one or after all tasks.



System Synthesis Example



- original MILP formulation- 47 timing variables, 225 binary (bus 153) and 1081 constraints (bus 416)
- commercial linear programming package used to solve the problem (XLP, developed by XMP Software, Inc.)

Processor	Cost	Execution time								
		S1	S2	S3	S4	S5	S6	S7	S8	S9
P1	4	2	2	1	1	1	1	3	-	1
P2	5	3	1	1	3	1	2	1	2	1
P3	2	1	1	2	-	3	1	4	1	4



Modeling of cost and execution time

- Execution time
 $\text{element}(P1, [2, 3, 1], D1)$
 ...
 $\text{element}(P9, [1, 1, 4], D4)$
- Cost
 $(P1=1 \vee P2=1 \vee \dots \vee P9=1) \Leftrightarrow C1,$
 ...
 $(P1=6 \vee P2=6 \vee \dots \vee P9=6) \Leftrightarrow C6,$
 $\text{Cost} = 4 \cdot C1 + 4 \cdot C2 + 5 \cdot C3 + 5 \cdot C4 + 2 \cdot C5 + 2 \cdot C6$



System synthesis results

Design	Cost	Performance (time units)	Performance optimization		Cost optimization		
			MILP (s)	CLP (s)	B&B Nodes	CLP (s)	B&B Nodes
Bus	10	6	6438.00	0.43	84	0.55	92
	6	7	5371.80	0.53	114	0.68	144
	5	15	3691.20	0.43	68	0.70	103
point-to-point links	15	5	3732.00	0.43	20	1.67	125
	12	6	26710.20	1.42	98	2.18	169
	8	7	32320.20	1.00	58	2.59	198
	7	8	4510.80	1.64	75	2.02	112
	5	15	38501.20	1.50	32	1.48	77



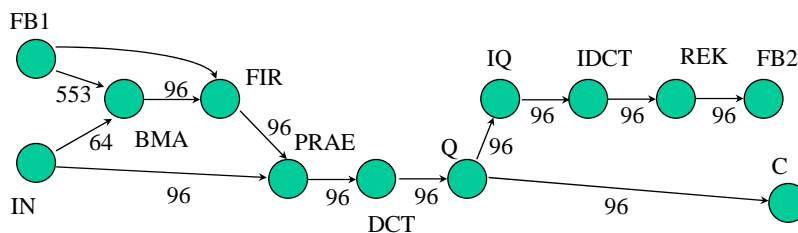
System synthesis results with local memory

Design	Cost	Performance (time units)	Performance optimization			Cost optimization	
			MILP (s)	CLP (s)	B&B Nodes	CLP (s)	B&B Nodes
Bus	28	6	6592.20	0.71	76	2.58	252
	23	7	5371.80	1.07	193	1.94	266
	22	8	123252.60	0.95	124	14.85	856
	21	10	316860.60	114.92	4 534	119.55	8 799
	18	11	236724.00	88.23	7 015	2.37	477
	17	12	138004.20	0.93	268	10.39	3 076
14	15	3581.40	0.54	22	9.89	3 076	
point-to-point links	38	5	-	0.56	24	2.08	107
	30	6	-	0.99	59	3.75	155
	25	7	-	1.60	79	5.58	314
	23	8	-	1.82	57	3.21	184
	22	10	-	4.50	84	59.25	855
	19	11	-	27.34	794	101.03	2 851
	18	12	-	97.72	2 686	8.66	1 047
14	15	-	1.18	14	4.95	328	



31

An Example



$$C + D_c \leq 2500$$

Video Coding Algorithm H.261



32

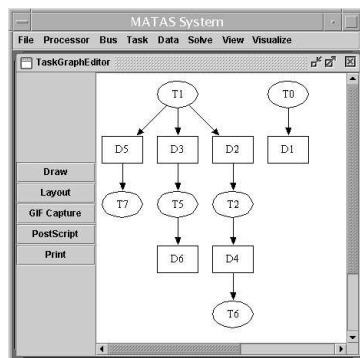
Task Mappings to Processors

Task	Uni- versal	BMA array	PAR1	DCT array	FIR array	BMA pipe	FIR seq	FIR pipe	DCT seq	DCT pipe
IN	-	-	-	-	-	-	-	-	-	-
FB1	-	-	-	-	-	-	-	-	-	-
BMA	7234	484	-	-	-	3617	-	-	-	-
FIR	7234	-	-	-	510	-	3461	1170	-	-
PRAE	1280	-	128	-	-	-	-	-	-	-
DCT	12312	-	-	132	-	-	-	-	6156	474
Q	-	-	-	-	-	-	-	-	-	-
IQ	-	-	-	-	-	-	-	-	-	-
IDCT	12312	-	-	132	-	-	-	-	6156	474
REK	1536	-	256	-	-	-	-	-	-	-
C	132	-	-	-	-	-	-	-	-	-
FB2	-	-	-	-	-	-	-	-	-	-

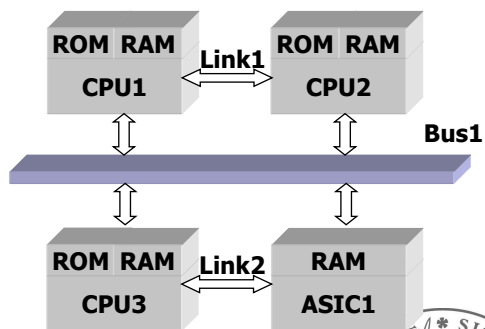


33

Scheduling with Memory Constraints



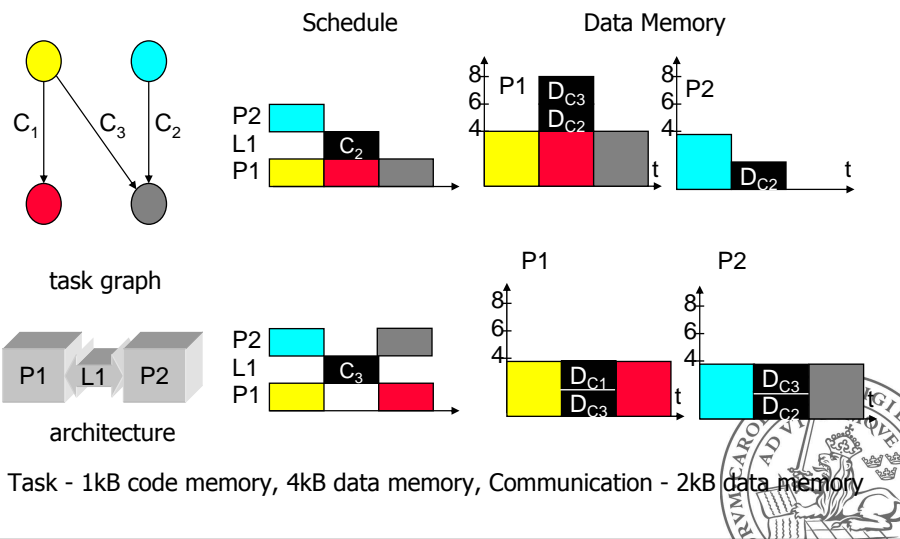
annotated task graph



target architecture



Memory importance



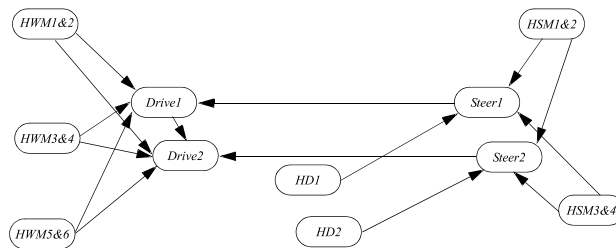
Experimental results H.261 example

EXP. #	ALGORITHM	PIPELINE DEGREE	DEADLINE	AVERAGE EXECUTION TIME	Σ DATA MEMORY	Δ TIME	Δ DATA MEM.
1	both	1	2871	2871	2683	-	-
2	greedy	4	6743	1686	3812	0	0
3	memory	4	6781	1696	3259	1%	-16%



Scheduling of Mars Path Finder under Power Consumption Constraints

The Mars rover operates on a very limited power supply. The power is provided by solar panels. The power obtained from solar panels was measured at different temperatures, and the results were as follows: 14.9W at -40°C, 12.0W at -60°C, and 9.0W at -80°C. There is also a battery power source, which provides a maximum of 10.0W, and it is not a replenishable energy source, so the battery power should be used as little as possible. The Mars rover has 6 driving and 4 steering motors, which need to be warmed up before their respective driving and steering can be performed.



Scheduling of Mars Path Finder under Power Consumption Constraints

Operation	Duration	
Heating steering motors (HSM1&2, HSM3&4)	5s	At least 5s and at most 50s before steering starts
Heating wheel motors (HWM1&2, HWM3&4, HWM5&6)	5s	At least 5s and at most 50s before driving starts
Hazard detection (HD1 & HD2)	10s	At least 10s before steering starts
Steering (Steer1, Steer2)	5s	At least 5s before driving
Driving (Drive1, Drive2)	10s	At least 10s before next hazard detection starts



Scheduling of Mars Path Finder under Power Consumption Constraints

Task	Duration	Power -40°C	Power -60°C	Power -80°C
Heat two motors	5s	7.6W	9.5W	11.3W
Drive	10s	7.5W	10.9W	13.8W
Steer	5s	4.3W	6.2W	8.1W
Hazard Detection	10s	5.1W	6.1W	7.3W
CPU	Constant	2.5W	3.1W	3.7W

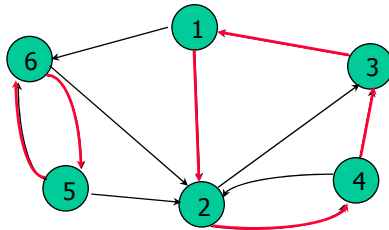


Modeling

- Precedence constraints:
 - $t_{hd1} + d_{hd1} \leq t_{steer1},$
 - ...
 - $t_{steer1} + d_{steer1} \leq t_{drive1},$
 - $t_{hwm12} \leq t_{steer1} + 50,$
- Power consumption constraints:
 - $cumulative([t_{hd1}, \dots, t_{sm12}],$
 - $[p_{hd1}, \dots, p_{sm12}],$
 - $[d_{hd1}, \dots, d_{sm12}], Power)$
- Optimize "Power"



Cycle (circuit) constraint



cycle(2, [[2,6], [3,4], [1], [2,3], [2,6], [2,5]])



[[2],[4], [1], [3], [6], [5]]



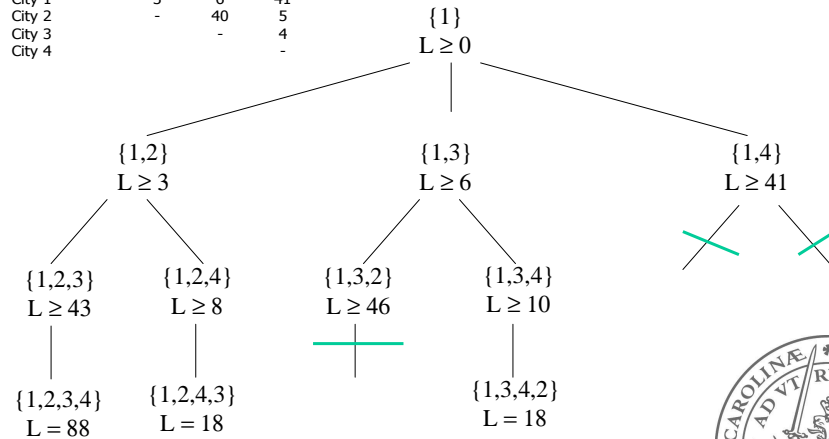
Search

- Standard search uses depth-first-search with backtracking.
- Optimization uses branch-and-bound or similar methods.



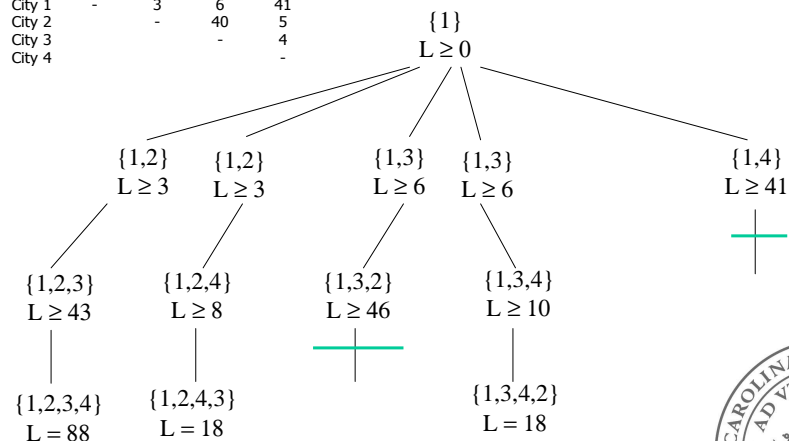
Typical branch and bound search (TSP problem)

	City 1	City 2	City 3	City 4
City 1	-	3	6	41
City 2	-	-	40	5
City 3	-	-	-	4
City 4	-	-	-	-



Search with restart (CLP typical)

	City 1	City 2	City 3	City 4
City 1	-	3	6	41
City 2	-	-	40	5
City 3	-	-	-	4
City 4	-	-	-	-



Search (cont'd)

[City1::2..4, City2::{1,3..4}, City3::{1..2,4}, City4::1..3]

- How to select order of variable assignment?
 - dynamic vs. static
 - criteria
- How to select values to be assigned from variable's domain?
 - a single value
 - sub-domain
 - ...



Variable Selection

- Static and dynamic
 - input order (static)
 - first-fail principle (smallest size of the domain)
 - smallest value in the domain
 - largest value in the domain
 - largest difference between the smallest and second smallest value in its domain
 - smallest max value in the domain
 - ...



Value Selection

- Single value
 - minimum in the domain and then upwards
 - maximum in the domain and then downwards
 - middle and then towards smallest and largest
 - random
 - ...
- Domain split
 - split into two sub-domains
 - split into N
 - ...

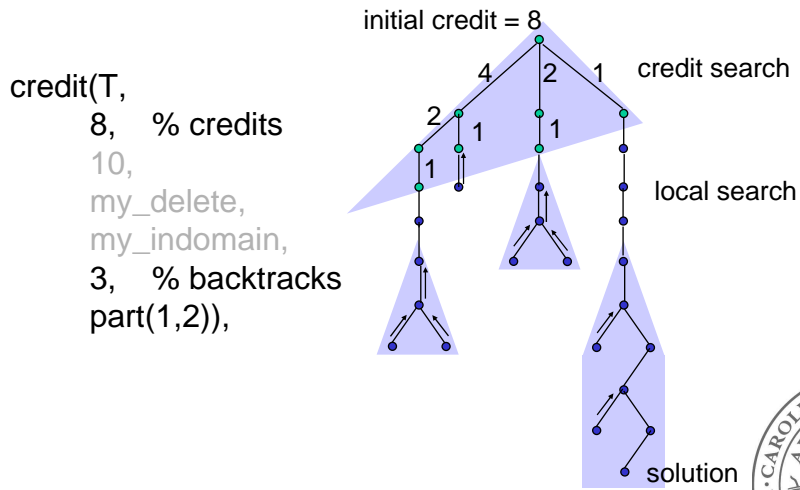


Search improvements

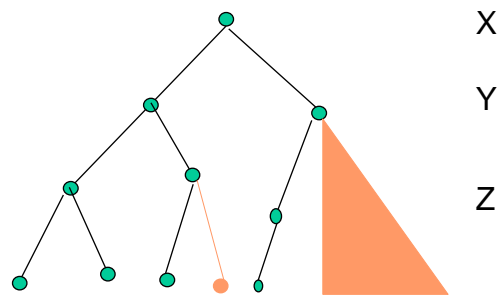
- Partial enumeration algorithms (instead of labeling)
 - Credit Search,
 - Limited Discrepancy Search (LDS).
- Assignment of subintervals instead of values to domain variables — possibly examines a bigger part of a solution space.
- Problem-dependent specific heuristics.
- Neighbourhood search...




Credit search



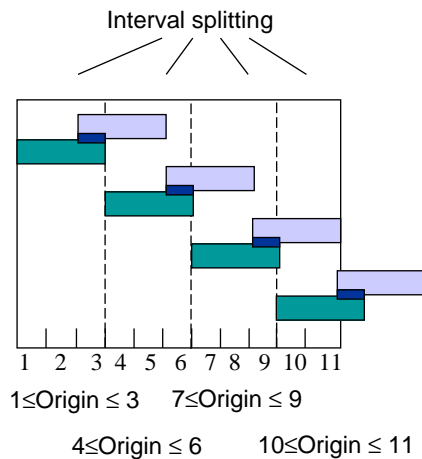
Limited discrepancy search



min_max(lds ([X,Y,Z], 1, input_order, indomain), Cost)



Interval splitting



For each task:

```
Origin :: min..max
duration

Rest    :: 0..duration-1,
Quotient :: 0..max,
Quotient*duration+Rest #= Origin.
```

Enumeration procedure:

```
labeling(Origins, Quotients) :-
    labeling(Quotients, first_fail, indomain),
    labeling(Origins, first_fail, indomain).
```

Summary and conclusions

- **Advantages:**
 - focus on a specification of the problem, not on a solution method.
 - unified framework for different algorithms to be used to solve a problem (by encapsulating them as constraints).
 - easy definition of problems with many heterogeneous constraints.
 - easy extension of a problem by adding new constraints.
 - collaboration of solvers and distributed computing

Summary and conclusions

- **Limitations:**
 - NP-hard problems.
 - often non-predictable behavior of a solver.
 - difficult to define and add new constraints:
 - into existing systems — interface problems.
 - new propagation algorithms need to be developed.
 - difficult to match constraints with actual problems.



CP finite domain systems

- SICStus Prolog
- CHIP from COSYTEC
- IF/Prolog
- ILOG
- Mozart/Oz
- Gnu Prolog
- JaCoP – Java based our own solver



Selected CP Web resources

- Constraints archive
<http://www.cs.unh.edu/ccs/archive>
- Guide to constraints programming
<http://kti.ms.mff.cuni.cz/~bartak/constraints>
- Sicstus manual
http://www.sics.se/isl/sicstus/sicstus_toc.html
- Gnu Prolog
<http://www.gnu.org/software/prolog/prolog.html>
- Mozart/Oz
<http://www.mozart-oz.org/>



Other resources

- Book
 - K. Mariott and P. J. Stuckey *Programming with Constraints: An Introduction*, The MIT Press, 1998.
- Conferences
 - Principles and Practice of Constraint Programming (CP)
 - The Practical Application of Constraint Technologies and Logic Programming (PACLP)
- Journal
 - Constraints (Kluwer Academic Publishers)



Selected Papers

- Kuchcinski, K., *Embedded System Synthesis by Timing Constraints Solving*, Proc. 10th International Symposium on System Synthesis, Antwerp, Belgium, September 17-19, 1997.
- Gruian, F. and Kuchcinski, K., *Operation Binding and Scheduling for Low Power Using Constraint Logic Programming*, Proc. 24th Euromicro Conference, Workshop on Digital System Design, Västerås, Sweden, August 25-27, 1998.
- Kuchcinski, K. and Wolinski, Ch., *Global Approach to Assignment and Scheduling of Complex Behaviours based on HCDG and Constraint Programming*, Journal of Systems Architecture, 2003, Elsevier Science.



Selected Papers (cont'd)

- Kuchcinski, K., *Constraints Driven Design Space Exploration for Distributed Embedded Systems*, Journal of Systems Architecture, vol. 47, no. 3-4, pp. 241-261, 2001, Elsevier Science.
- Szymanek, R. and Kuchcinski, K., *A Constructive Algorithm for Memory-Aware Task Assignment and Scheduling*, Proc. 9th International Symposium on Hardware/Software Codesign, Copenhagen, Denmark, Apr. 2001.
- Szymanek, R. and Kuchcinski, K., *Partial Assignment Technique for Task Graph Scheduling*, 40th DAC, Anaheim, USA, June 2003.
- Kuchcinski, K. *Constraint-driven scheduling and resource assignment*, ACM Trans. on Design Automation of Electronic Systems, vol. 8, no. 3, pp. 355-383, 2003.





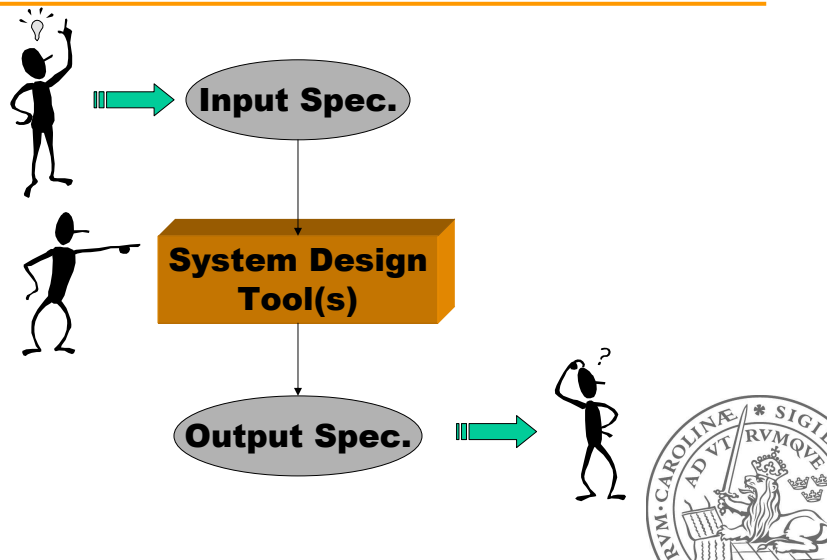
LUND INSTITUTE
OF TECHNOLOGY
Lund University

Methodologies for Embedded System design

Kris Kuchcinski
Dept. of Computer Science
Lund University
Sweden

<http://www.cs.lth/~kris>

General Methodology



Input to System Design

- Executable specification (functional requirements):
 - usually provided as interacting processes/tasks,
 - very often multi-language specifications,
 - can be simulated and verified,
 - can be used to perform analysis, e.g, estimation.
- Specification languages: C, C++, VHDL, Verilog, SystemC, Esterel, SDL, etc.
- Set of (non-functional) design requirements (cost, speed, I/O rate, power consumption, etc.).

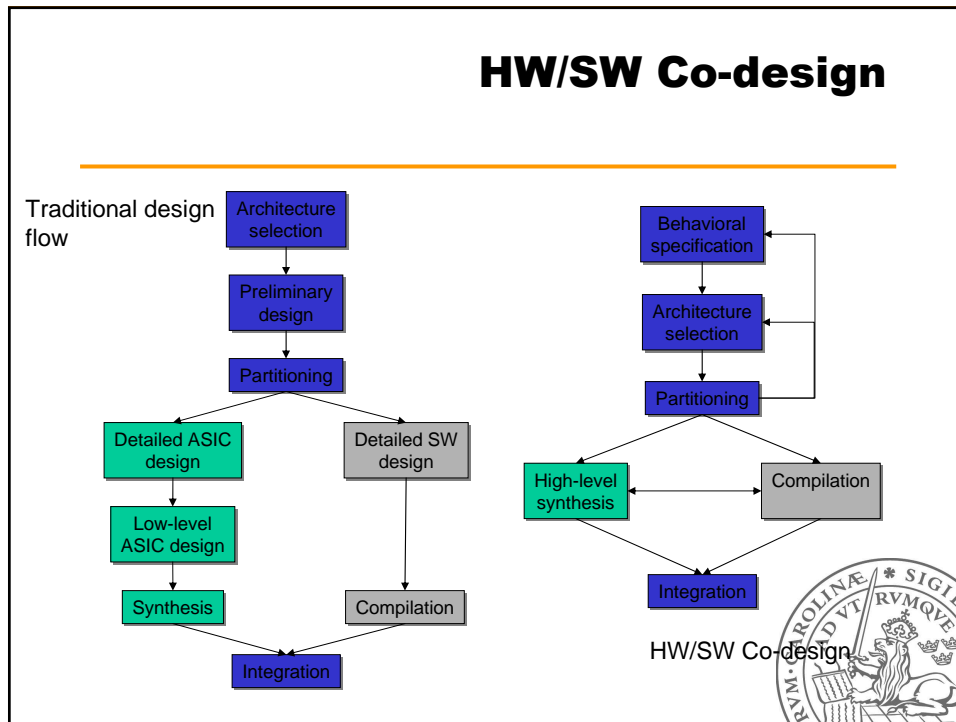


Output from System Design

- A set of system modules assigned to system components (CPU's, DSP's, ASIC's, etc.).
- Communication modules.
- Each module can be further synthesized to hardware using *high-level synthesis* or *compiled to software*.



HW/SW Co-design



Basic Characteristics of the Methodology

- *Behavioral specification* is given for the complete heterogeneous system, regardless of how different parts will be later implemented.
- *Analysis techniques* are provided; specially different estimation techniques.
- *Synthesis tools* are used to automatically explore a design space.
 - high-level synthesis, RTL synthesis,
 - compilers, cross-compilers,
 - interface generators,
 - etc.

Estimation of Design Parameters

- Estimation of parameters such as size, cost, power consumption.
- Does not need to be very precise but has to be “consistent” — follows real design parameters.
- Usually 15%-20% inaccurate.
- Trade-off between accuracy and estimation time.

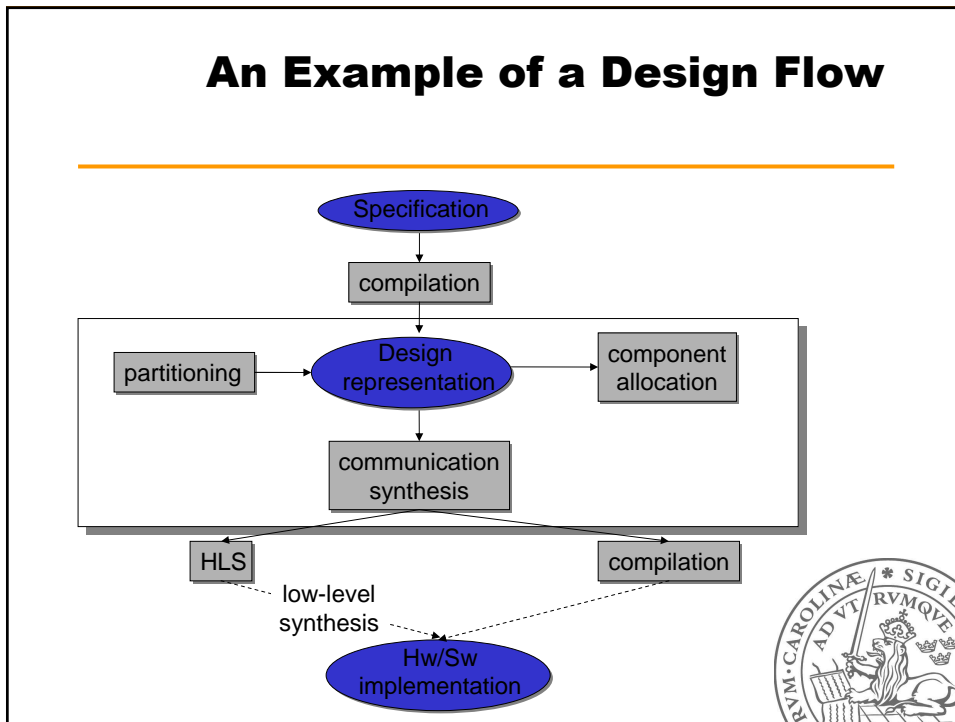


Improvements of the Design Process

- High-level specification is made before architecture selection and implementation *decisions can be made more accurate* (better exploration of architectures).
- A uniform description of HW and SW makes it possible to *move parts of the systems between HW and SW*.
- HW and SW development is moved closer and the *integration cost is reduced*.
- An *early evaluation of system characteristics* is possible.



An Example of a Design Flow



Specification Example

“Extended” VHDL

```

port(IP1,IP2:in INTEGER; OP1,OP2:out INTEGER);
...
signal S1,S2,S3,S4,S5,S6:INTEGER;
P1 : process
...
  receive(IP1);
...
  send(S1,...);
...
  send(S3,...);
...
  receive(S6);
...
end process P1;

P2 : process
...
  receive(IP2);
...
  receive(S1);
...
  send(OP2,...);
...
end process P2;

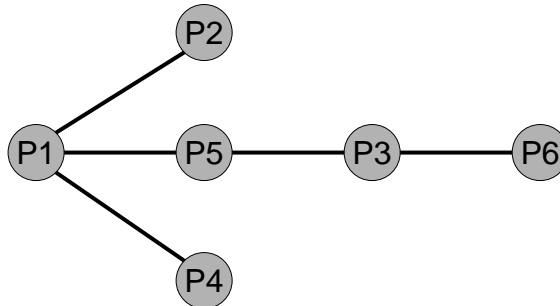
P3 : process
...
  receive(S4);
...
  send(S2,...);
...
end process P3;

P4 : process
...
  receive(S3);
...
  send(S5,...,S6,...);
...
end process P4;

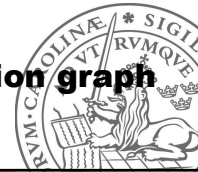
P5 : process
...
  receive(S1,S5);
...
  send(S4,...);
...
end process P5;

P6 : process
...
  receive(S2);
...
  send(OP1,...);
...
end process P6;
  
```


Representation Example



Process communication graph



Allocation of System Components

- Decides about the kind and number of components for implementation of the system
 - processing elements: μ procesosrs, micro-controllers, DSP's, ASIP's, ASIC's, FPGA's, etc.
 - storage elements: memories, register files, registers, etc.
 - communication devices: buses, point-to-point links, networks, etc.
 - specialized I/O devices: A/D, D/A, frame grabbers, etc.

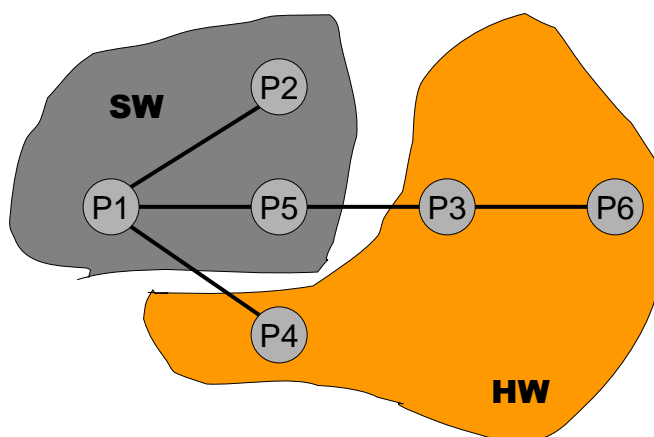


Partitioning

- *Functional* partitioning vs. *structural* partitioning.
- Abstraction level.
- Partitioning granularity (*fine* or *course*):
 - modules,
 - processes and procedures,
 - instructions.
- Partitioning objective:
 - performance,
 - minimal communication,
 - low power,
 - combination of several criteria.



Partitioning Example

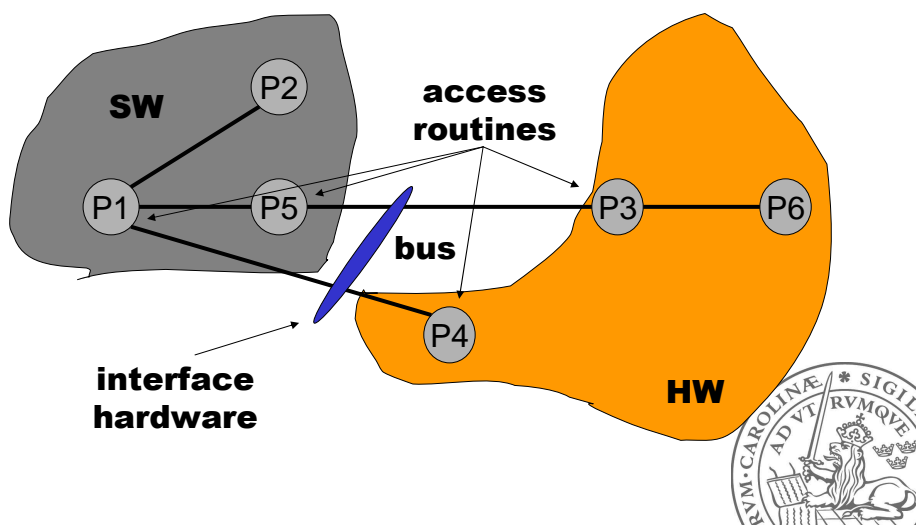


Communication Synthesis

- Creation of abstract communication channels by communication clustering.
- Communication refinement
 - selection of communication lines width,
 - protocol selection,
 - etc.
- Interface generation:
 - device drivers,
 - communication hardware,
 - etc.



Communication Synthesis Example



Design Decisions

- Different types of design decisions
 - selection of components, partitioning, assignments, scheduling, etc.
 - decisions regarding runtime system done off-line or are postponed to runtime (e.g., static vs. runtime scheduling)
- Design decisions are mutually dependent
- Huge design space



Design Automation

- Uses internal representations which are usually based on graphs.
- Graph algorithms (shortest path, Hamiltonian circuit, topological sort, depth-first-search, breadth-first-search, SAT, etc.).
- Optimization methods — (M)ILP, CLP, heuristics, etc.
- *Tractable* and *intractable* problems.
- *Decidable* and *undecidable* problems.
- *Decision problems* and *combinatorial optimization problems*.



Design Automation Consequences

- Most of the problems which need to be solved in design automation are NP-complete or NP-hard.
- Usually only small problems can be solved exactly.
- Need for algorithms which do not guarantee optimal solutions but “good enough” solutions
 - *approximation algorithms* — guarantee a solution with a cost that is within some margin of the optimum,
 - *heuristics* — algorithms that are constructed based on “rules-of-thumb”; nothing can be said in advance about the quality of the result.



Examples of Embedded Systems (cont'd)

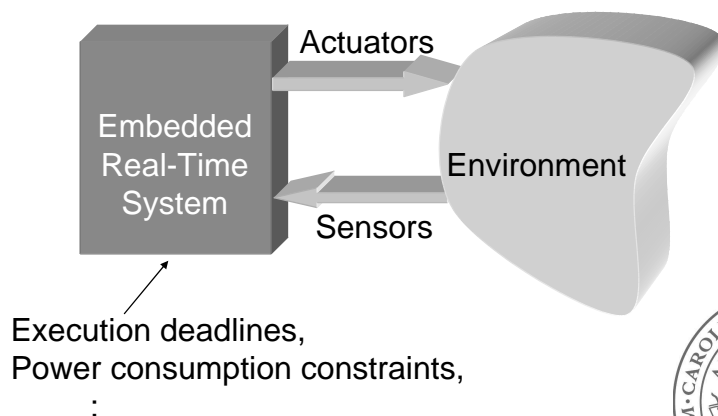
Anti-lock brakes	Modems
Auto-focus cameras	MPEG decoders
Automatic teller machines	Network cards
Automatic toll systems	Network switches/routers
Automatic transmission	On-board navigation
Avionic systems	Pagers
Battery chargers	Photocopiers
Camcorders	Point-of-sale systems
Cell phones	Portable video games
Cell-phone base stations	Printers
Cordless phones	Satellite phones
Cruise control	Scanners
Curbside check-in systems	Smart ovens/dishwashers
Digital cameras	Speech recognizers
Disk drives	Stereo systems
Electronic card readers	Teleconferencing systems
Electronic instruments	Televisions
Electronic toys/games	Temperature controllers
Factory control	Theft tracking systems
Fax machines	TV set-top boxes
Fingerprint identifiers	VCR's, DVD players
Home security systems	Video game consoles
Life-support systems	Video phones
Medical testing systems	Washers and dryers

Source: Embedded Systems Design: A Unified Hardware/Software Introduction, (c) 2000 Vahid/Givargis



Embedded Systems

"A device that includes a programmable computer but is not itself a general-purpose computer."



Embedded Systems (cont'd)

- Computing systems embedded within electronic devices
- Hard to define. Nearly any computing system other than a desktop computer
- Billions of units produced yearly, versus millions of desktop units
- Perhaps 50 per household and per automobile

Source: Embedded Systems Design: A Unified Hardware/Software Introduction, (c) 2000 Vahid/Givargis



Embedded Systems (cont'd)

- Non User-Programmable.
- Based on programmable components (e.g., Micro-controllers, DSP's...) but often contain application specific hardware (IC's, ASIC's).
- Reactive Real-Time Systems:
 - React to external environment,
 - Maintain permanent interaction,
 - Ideally never terminate,
 - Are subject to external timing constraints (real-time).

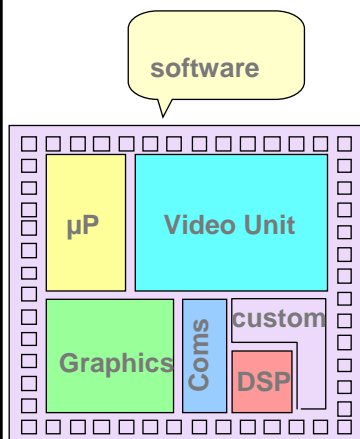


Characteristics of Embedded Systems

- Sophisticated functionality.
- Real-time operation.
- Low manufacturing cost.
- Low power.
- Designed to tight deadlines by small teams.
- “Resource conscious” vs. “Unlimited resources” programming



SoC Embedded System

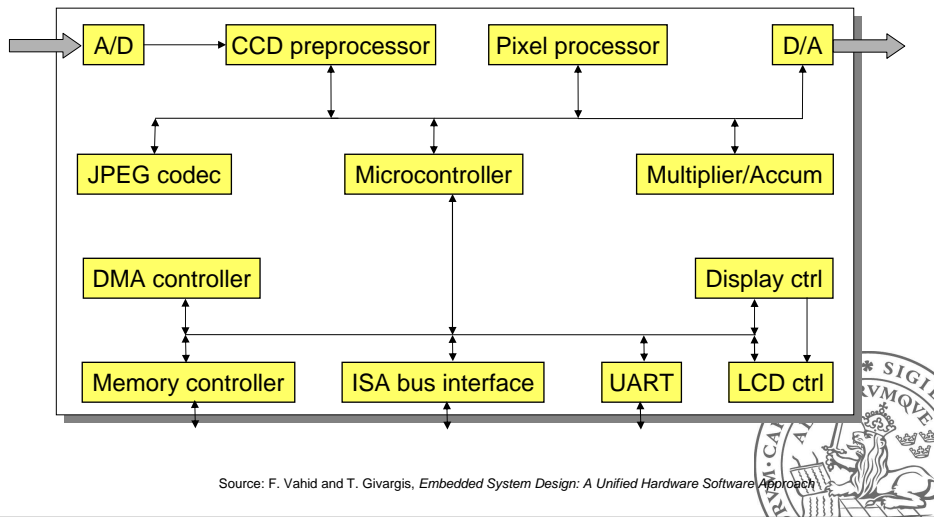


- Assembly of “prefabricated components” often purchased from external vendors (“IP”)
 - “black box” hierarchy
- Design & Verification at the System level
 - rather than the logic level
 - Interface and communication
- Great Importance of Software

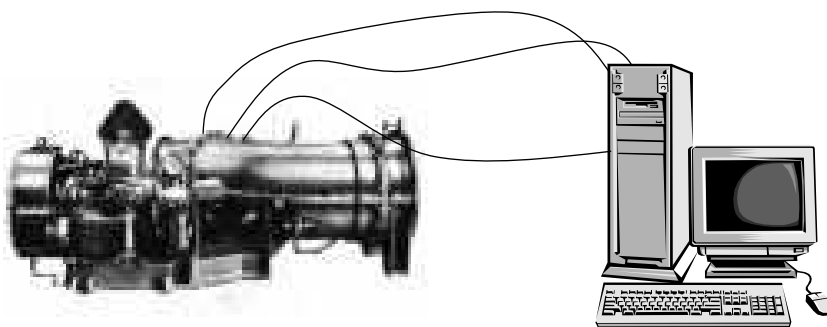
Source: Alberto Sangiovanni-Vincentelli, 35th DAC



A Digital Camera Example



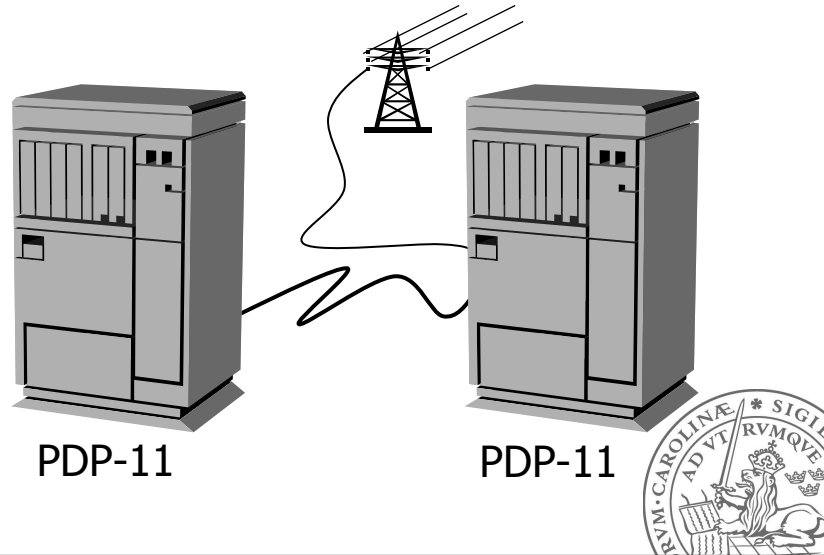
Real-time gas turbine testing system



MI-2 helicopter engine

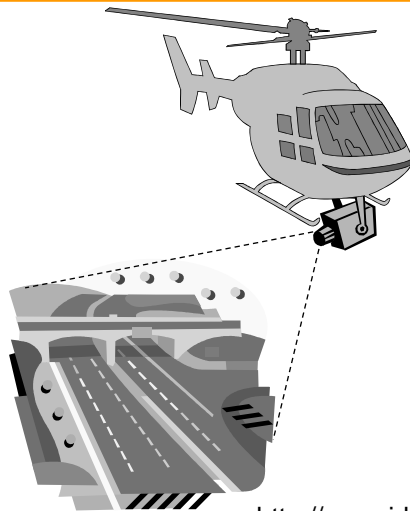
"Minicomputer"
8kB RAM
cassette tape

TELEX-I and TELEX-II systems



WITAS

WITAS project



- Autonomous system.
- Real-time system.
- Image processing.
- Mission planning.
- Incorporation of GIS systems.
- Interface with ground operator.
- ...

<http://www.ida.liu.se/ext/witas>



Typical Hardware Components of DSP System

Component class	Implements	Compiler	Specification
DSP processor	Low data-rate DSP Slow control loops Appl. Spec. alg.	(Retargetable) code generator High level synth.	Assembly C DFL
Microcontroller	User interface Slow control loops	C compiler	C
Hardware accelerator	High data-rate DSP	High level synth. RT level synth.	C, DFL VHDL
Communication blocks and memory	Internal & external communication Storage & buffering	Memory mgmt. (A)synchronous interface synth.	Data-sheets STG
Others	Usually FSM's - clock generators - DMA blocks	RT level synth. Asynchronous synth.	VHDL

Source: H. de Man, et. al. "Co-design of DSP Systems"
Hardware/Software Co-design, Kluwer 1995.

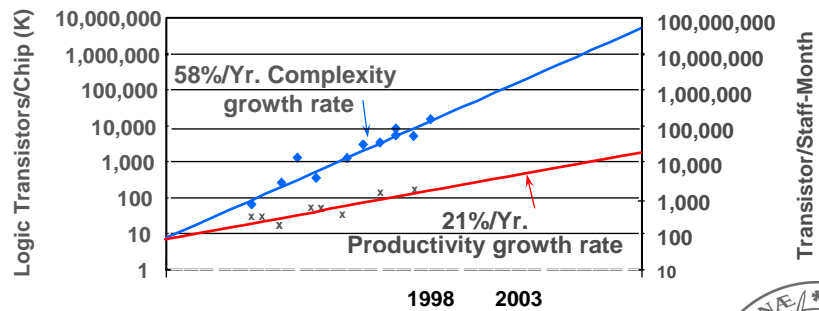


Importance of Embedded System Design Methodologies

- Hardware complexity.
- Heterogeneous systems containing hardware (both digital and analog) and software.
- Heterogeneous components (CPU's, DSP's, ASIC's, buses, point-to-point links, etc.).
- Heterogeneous requirements — performance, cost, power consumption, etc.
- System-on-chip.
- Shorter design cycles required by time-to-market constraints.
- ...



Design Complexity and Designer Productivity Gap



Source: Bryan Preas, Xerox PARC, 35th DAC



Software vs. Hardware Design short summary

- Software
 - flexibility,
 - reconfigurability, easy update, etc.,
 - complex functionality,
 - cost,
 - ...
- Hardware
 - speed,
 - power consumption,
 - cost in large volumes,
 - ...



Design of Embedded Systems

- Need to be done using high-level specification, programming and hardware description languages — not assembly languages and gate/transistor level design.
- Requires efficient design space exploration and synthesis/compilation tools.
- Different design requirements has to be taken into account, e.g., cost, performance, testability, quality of service, power consumption.
- Multi-language design framework.



Importance of High-Level Design Methods

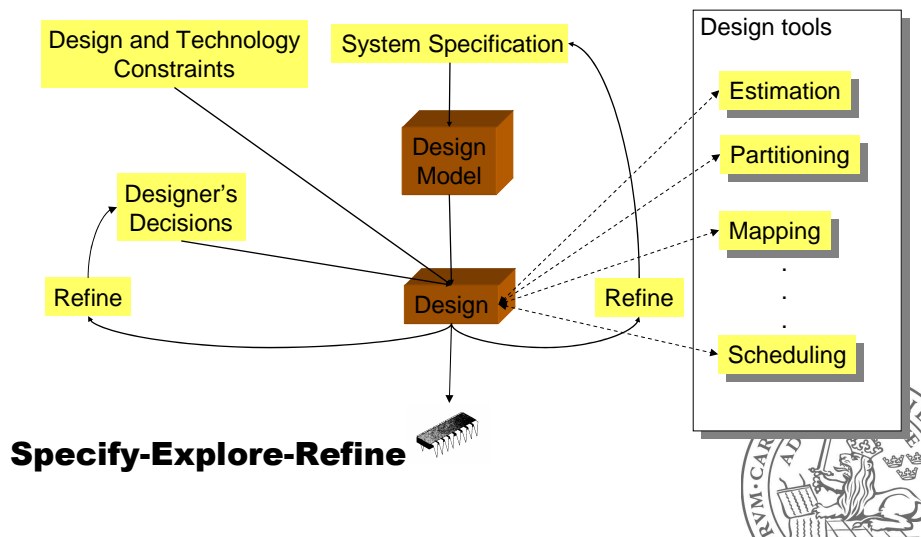
System Verification Processing Speeds

System Implementation	Processing time (s/frame)
Behavioral model	1 200 (20 min/frame)
RTL model	144 000 (1.6 days/frame)
Gate model	228 000 (2.6 days/frame)
Gate model on hardware accelerator	1 200
Rapid Prototype	0.5
Target Hardware	0.05

Source: Paul Clemente, Ron Crevier, Peter Runstadler "RTL and Behavioral Synthesis A Case Study", VHDL Times, vol. 5, no. 1.



General Design Flow



Specification and Programming

- Specification languages, such as UML, SDL.
- Programming languages, such as C, C++, Java, Esterel, assembly languages.
- Hardware description languages, such as VHDL, Verilog, SystemC.
- **Example:** combining SystemC and C++ gives unified simulation environment for hardware and software.



Hardware Description Languages

- Cover several levels of design abstraction as well as behavioral and structural description domain.
- Contain typical features of programming languages, such as data types and program statements.
- Special features:
 - time concept,
 - structure description,
 - parallelism.
- VHDL (IEEE standard), Verilog, SystemC.



Design Representations (Computational Model)

- Used to represent/model digital systems under design.
- Generated by a compiler from system specification or coded directly in the model.
- Represent the semantics, structure and timing of the system.
- Usually based on some kind of annotated graph representation.
- Used internally by design automation systems or by the modeler/designer.

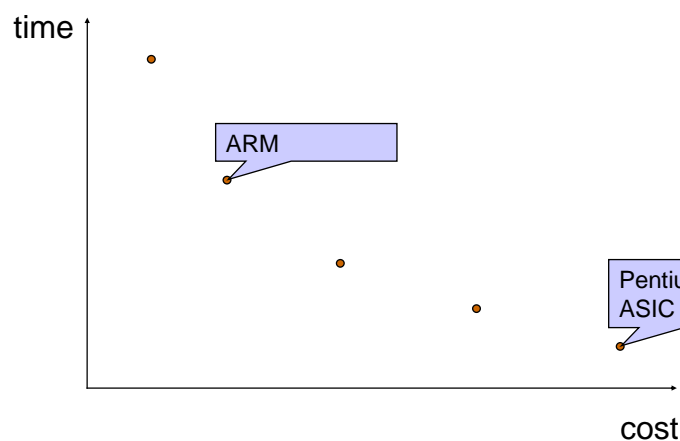


Design — Synthesis

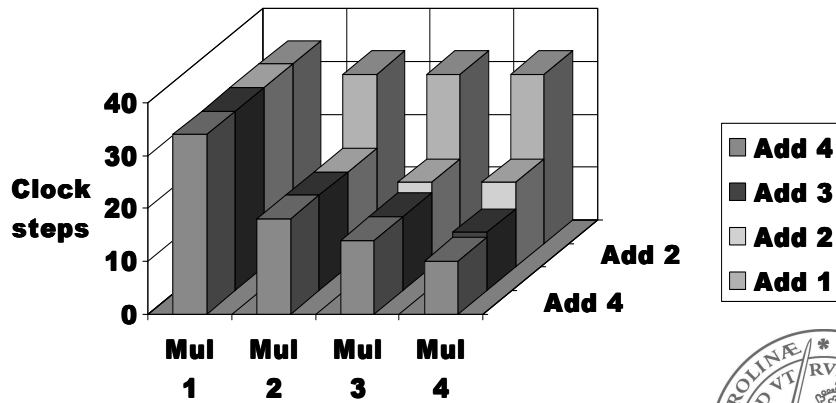
- *Software translation* into target code for a processor (real-time operating system might be used).
- *Hardware synthesis* — translation of a behavioral representation of a design into a structural one.
- *Communication synthesis* — generates hardware and software which interconnects system components.



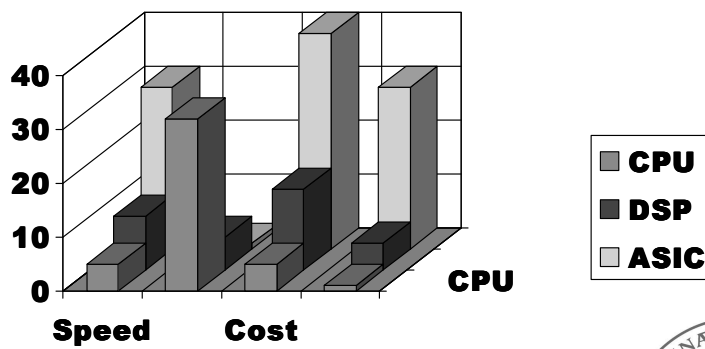
Pareto points



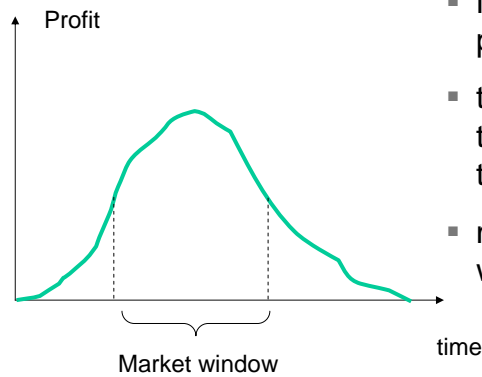
Discrete Cosine Transform Partial Design Space



Design Space Exploration



Time-to-market constraint



- Need time for new product development,
- the biggest profit is in the market window time,
- missing the market window can be costly.



Summary

- Embedded systems are important class of electronic systems which can be found everywhere,
- Combine hardware and software solutions,
- Cover several engineering and research areas:
 - microelectronics,
 - real-time systems,
 - software development,
 - etc.
- Need careful design which optimizes different design parameters.

