GEFÖRDERT VOM

Bundesministerium
für Bildung
und Forschung

Verifying the entire hardware of distributed real time systems

W. Paul
Universität Saarbrücken
wiss. Gesamtprojektleiter
bmb+f Projekt Verisoft

www.verisoft.de

You all know how to design hardware...

- Hardware verification is the process of explaining perfectly, why a piece of hardware works.

- If you don't know how to construct it, it is VERY hard to explain why it works....

Overview

- Verisoft Project (TPHOLs 2005)
  - ISA
  - memory management
  - maybe:
    - compiler
    - OS kernel
- Automotive Subproject (ICCD 2005; lecture notes ...)
  - ISA in (distributed) real time systems
  - serial interfaces
  - flex ray (like) bus interfaces
  - processors + interfaces on a bus
  - program correctness and worst case execution time (WCET)

Verisoft is...

- ...research supported by Dr. Reuse (bmb+f)
  - Transrapid
    - magnetic leverage train
  - Verbmobil
    - now speech technology of Siemens and Mercedes
  - **Verisoft**

Verisoft Mission

- Develop (paper and pencil theory,) tools and methods for pervasive system verification
  - hardware
  - system software
  - communication system
  - applications
- CLI stack style
- Demonstrate with applications of industrial interest

Verisoft

- Consortium
  - Infineon, T-Systems, BMW, AbsInt, OneSpin Solutions (2005: 14 Mio € venture capital)
  - TU Munich, Uni SB, TU Darmstadt, Uni Koblenz
  - DFKI, MPI, OFFIS
- Funding
  - 3.5 Mio €/year
  - now in 3rd year

Verisoft

- Consortium
  - Infineon, T-Systems, BMW, AbsInt, OneSpin Solutions (2005: 14 Mio € venture capital)
  - TU Munich, Uni SB, TU Darmstadt, Uni Koblenz
  - DFKI, MPI, OFFIS
- Funding
  - 3.5 Mio €/year
  - now in 3rd year
- Maximize: insight/€

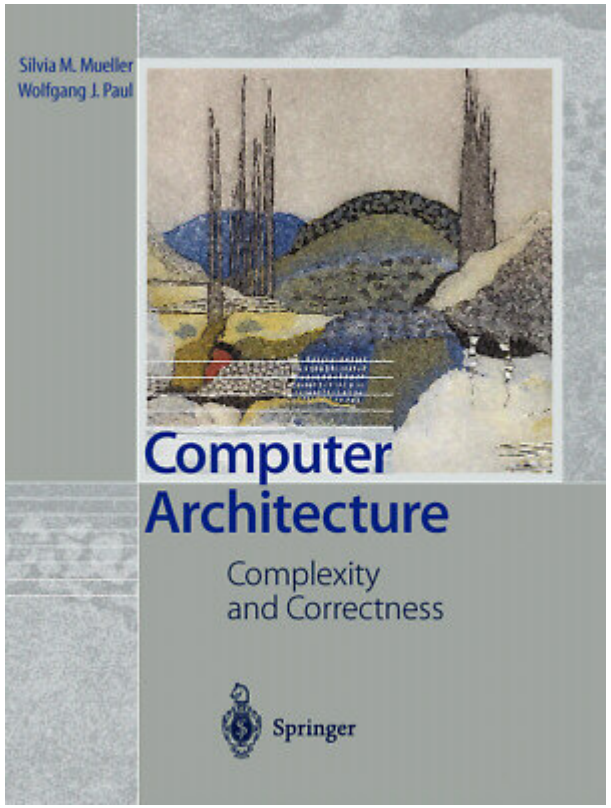
Project Structure

- Tools
  - interactive provers: Isabelle HOL and VSE
  - Hoare logic
  - integration of automatic methods
- Demonstrators
  - textbook (everything public)
  - hardware (infineon, OneSpin Solutions)
  - automotive (BMW, Absint)
  - biometric identification system (T-systems)

tools: example

- Hoare logic for C0 (PhD thesis Norbert Schirmer)
- int treated as natural numbers BUT
  - guards generated for each arithmetic operation (prove $x < 2^{32}$)
  - usually discharged automatically (like array bounds check)
- automatic termination analysis (A. Podelski)

textbook system

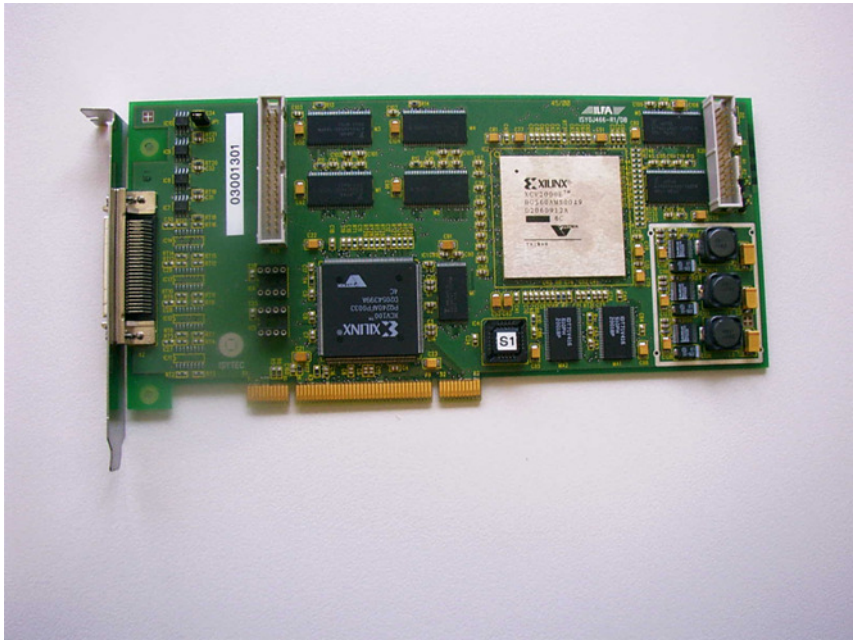


+ Diss. D. Kröning +...

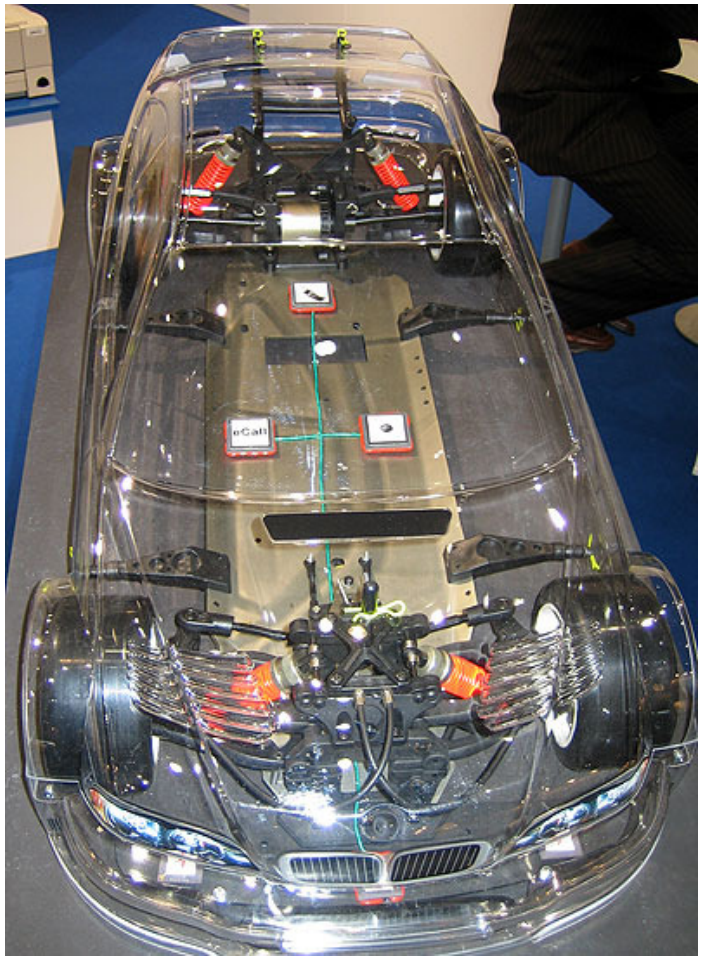
- underwent Lipton-DeMillo-Perlis screening
- VAMP processor (Charme 04/05)
  - out of order, precise maskable interrupts, IEEE compatible FPU, split cache, MMUs
- C0 compiler (SEFM 05)
- CVM generic operating system kernel (TPHOLs 05)
  - Disk and drivers (ICCD 05)
- Simple OS
- TCP/IP
- SMTP email client
- electronic signature

A side remark: VAMP hardware

- synthesized (Suggestion of C. Jacobi)
- v high end controller
- 1.5 Mio gate equivalents
- never tested
- up and runnig
- some results of multiplication of denormalized numbers ≠ results of certain (for normalized numbers verified) Intel fpu's

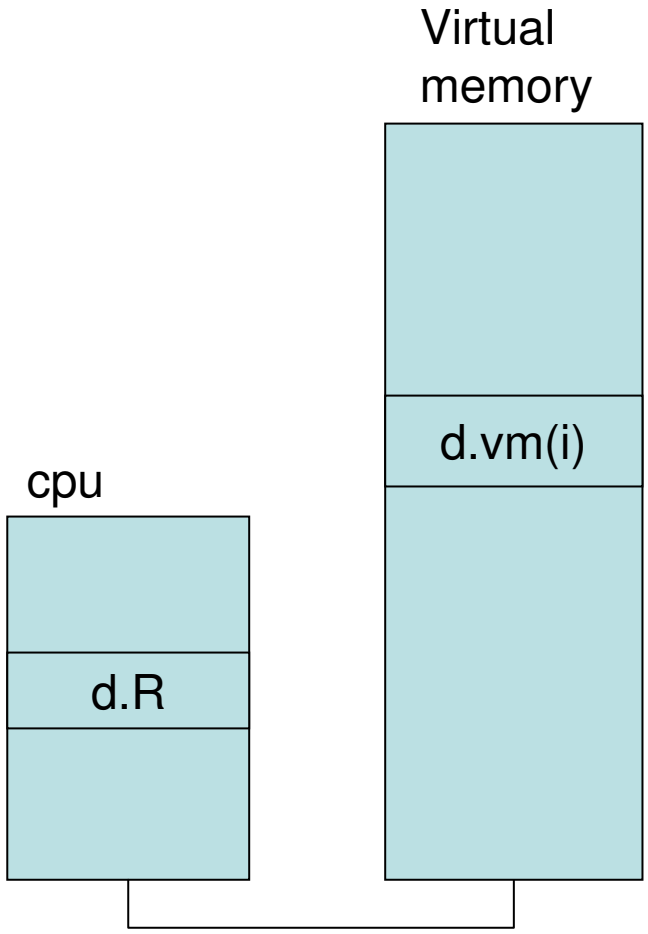


systems with industry partners



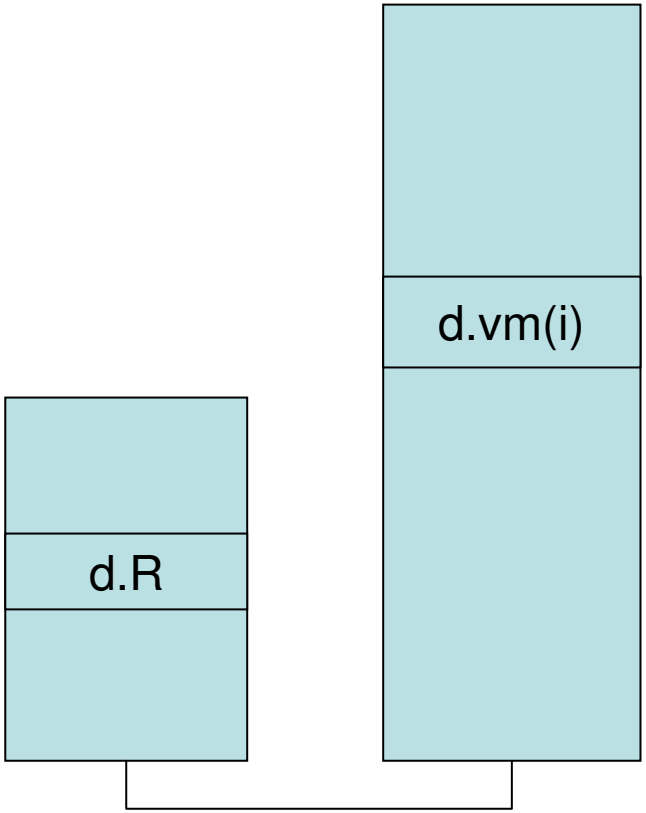
- Infineon, OneSpin Solutions: TriCore2 (high end controller)
- T-Systems:
  - PC (VAMP, C0, CVM, Simple OS)
  - card reader
  - chip card, biometric algorithms (not verified)
  - cryptographic protocols
- BMW, Absint: reverse engineered/public (ICCD 05)
  - VAMP/TriCore 2
  - FlexRay like bus interface (with SIO and clock synchronisation)
  - OSEKTime like real time OS (CVM dialect)
  - Worst case execution time (WCET)

virtual machines, configuration d



$$
\begin{aligned}
d &= (..., d.R, ..., d.vm) \\
d.R &: \text{cpu register} \quad R \\
R &\in \{PC, ..., GPR(i), ..., SR, ...\}
\end{aligned}
$$

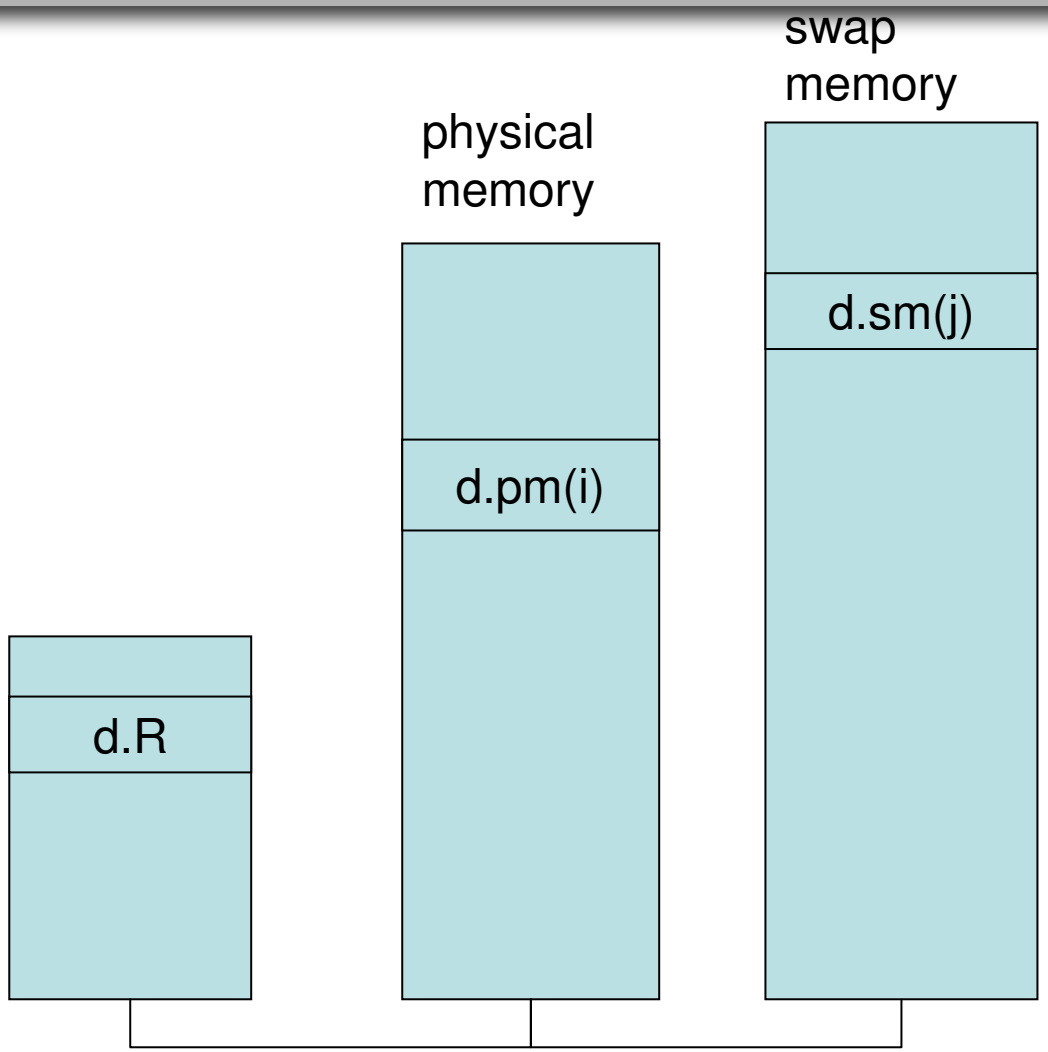
virtual machines, next state c‘, store word



$$
\begin{aligned}
I(d) &= d.vm_4(d.PC) \\
opc(d) &= I(d)[31:26] \\
sw(d) &= (opc(d) = 101010) \\
ea(d) &= d.gpr(RS1(d)) + imm(d) \\
d'.vm_4(ea(d)) &= d.gpr(RD(d))
\end{aligned}
$$

- no page fault interrupts

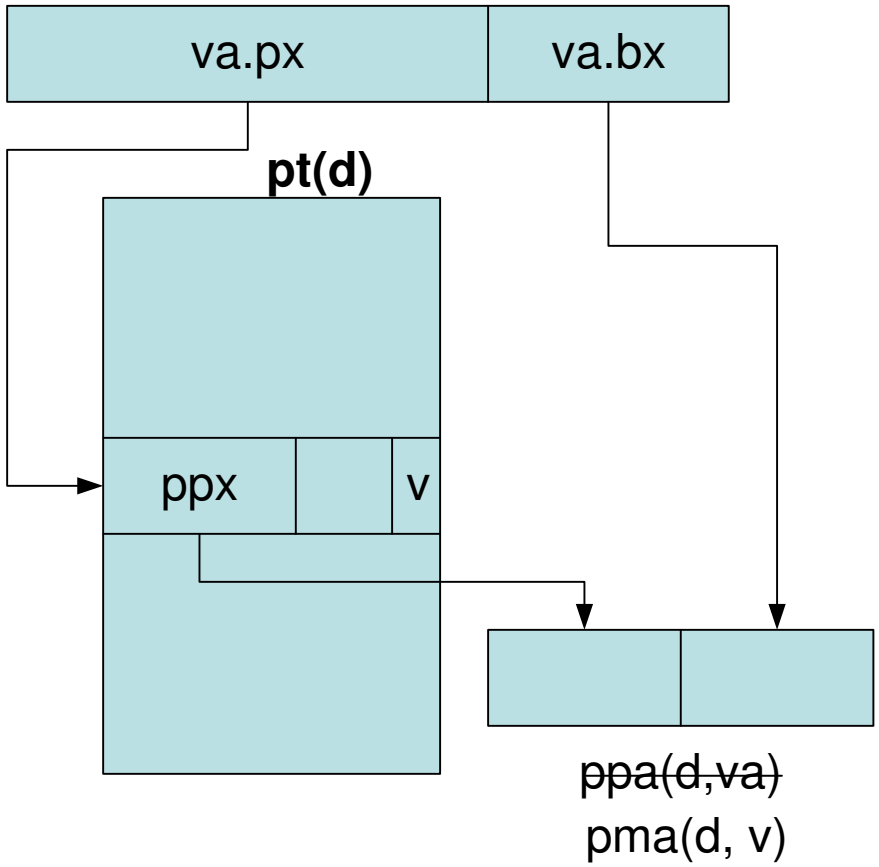
physical machines



- swap memory c.sm
- registers
  - d.mode
  - d.pto (page table origin)
  - d.ptl (page table length)
- adress translation if d.mode = 1
- page fault interrupts

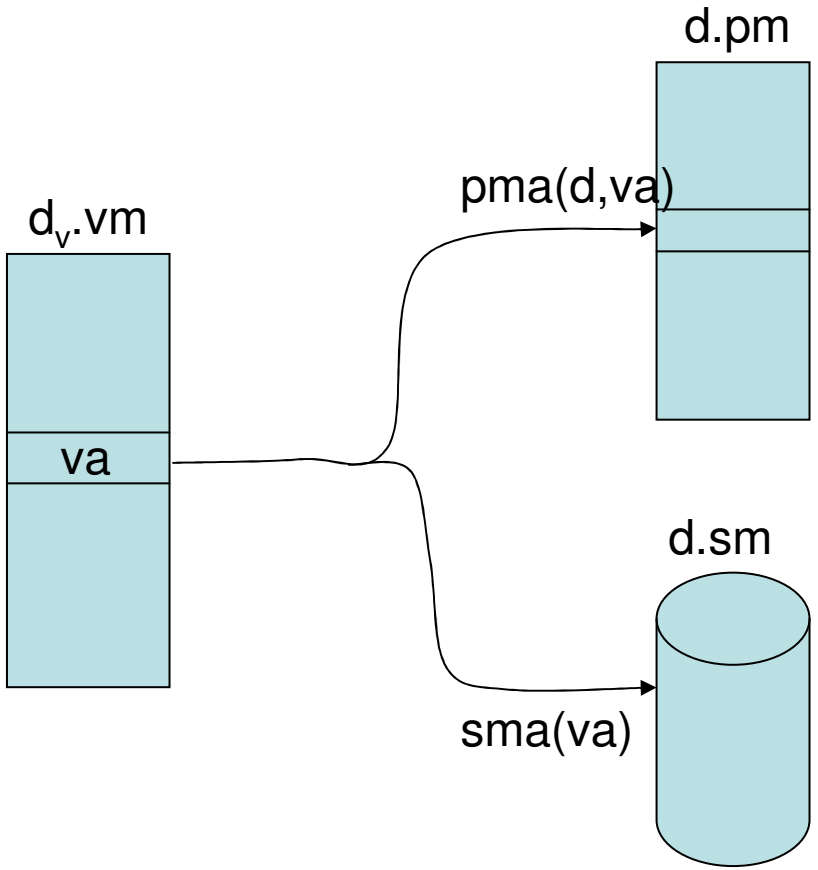
address translation (sequential)

- virtual address va
  - va = (va.px, va.bx)
  - px: page index,
  - bx: byte index
  - d DLX configuration
- hardware support by MMUs
  - pipelined realisation not trivial (self modifikation of page tables possible)
  - formally verified (Charme 05)



Simulation of virtual machines by phys. machines

- d_V.vm(va) =
  - d.pm(pma(va)):
    pt(d, va.px).v = 1
    (in Cache)
  - d.sm(sma(va)): otherwise
- d.pm is cache for d_V.vm
- theorem: phys. DLX + page fault handler simulate virtual DLX
- livesness: do not swap out most recently loaded page

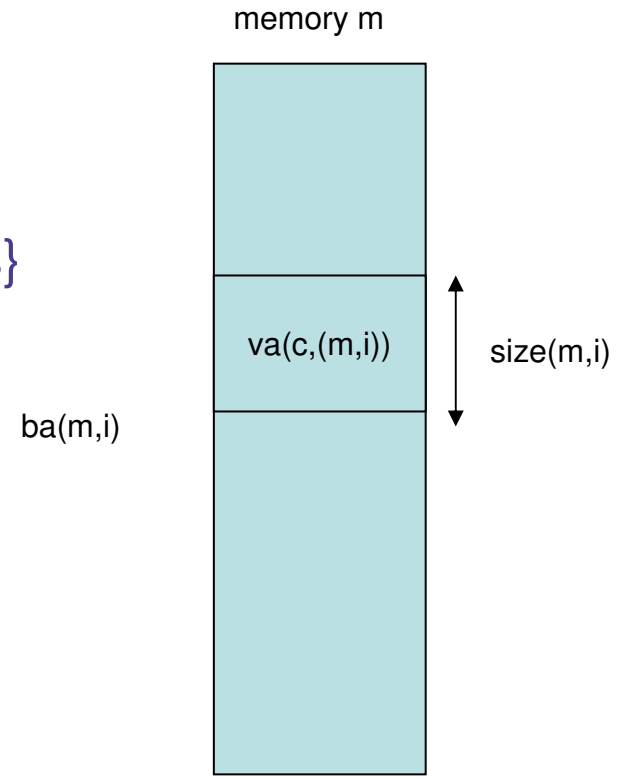


C0: Pascal with C Syntax

1. Hoare logic
   - Equivalent to big steps operational semantics
   - Shallow embedding in Isabelle-HOL very productive (1 page code/person week)
2. Small steps operational semantics
   - used for Interleaving of programs (kernel/several users)
   - imports results from Hoare logic

C0: configurations (~ M. Norrish)

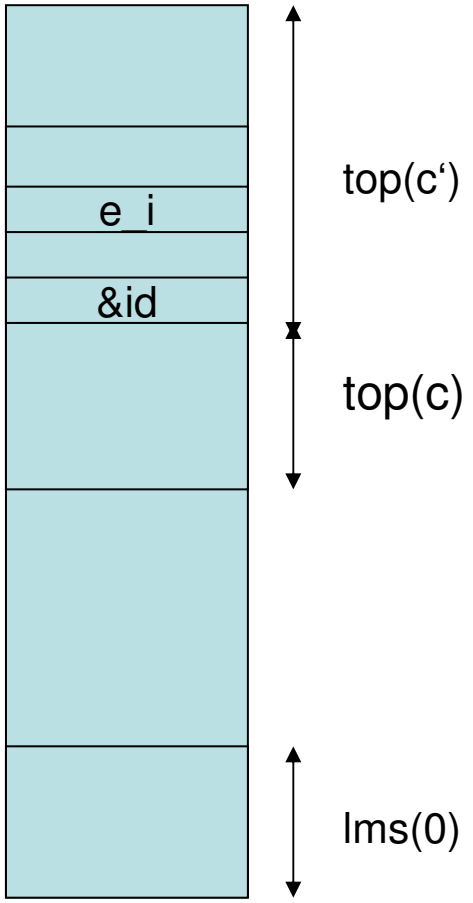
- c = (pr, rd, lms, hm)
  - pr program rest
  - rd recursion depth
  - lms: [0: recursion depth]![local memories}
  - hm: heap memory
- parameters
  - TT: {type names}![type descriptors}
  - FT:{function names}![types}X{bodies}
- subvariables
  - (m,i)[17].gpr[3]
- value of pointers: subvariables !



funktion call: semantics

rd: recursion depth

$$
\begin{aligned}
c.pr &: id = f(e_1, \ldots, e_n); r \\
c'.pr &: c.FT(f).body; r \\
c'.rd &= c.rd + 1 \\
top(c') &= c'.lms(c'.rd) \\
&\ldots \\
top(c')(i) &= va(c, e_i) \\
&\ldots \\
top(c')(0) &= \&(c, id)
\end{aligned}
$$



simulation relation consis(c, alloc, d)

r-consis: implementation
of stack and heap.

e-consis: (elementary subvariables)

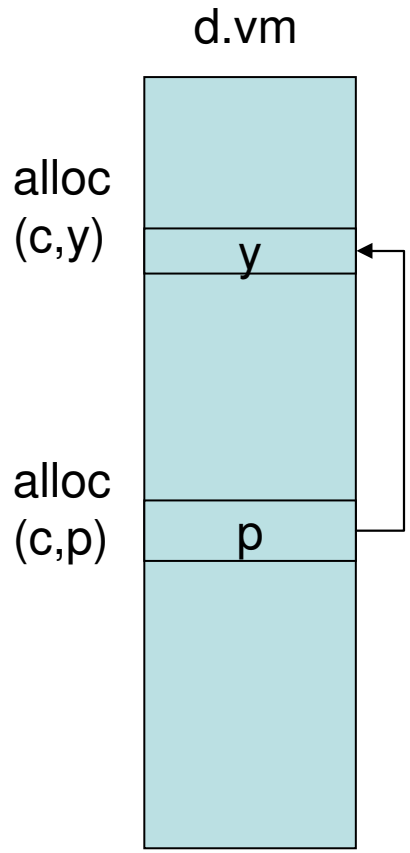
$$d.vm_{asize(x)}(alloc(c,x)) = va(c,x)$$

p-consis (pointer)
p, y (sub)variables, $va(c,p) = y$,
dann

$$d.vm_4(alloc(c,p)) = alloc(c,y)$$

c-consis(control)

$$d.pc = start(code(head(c.pr)))$$



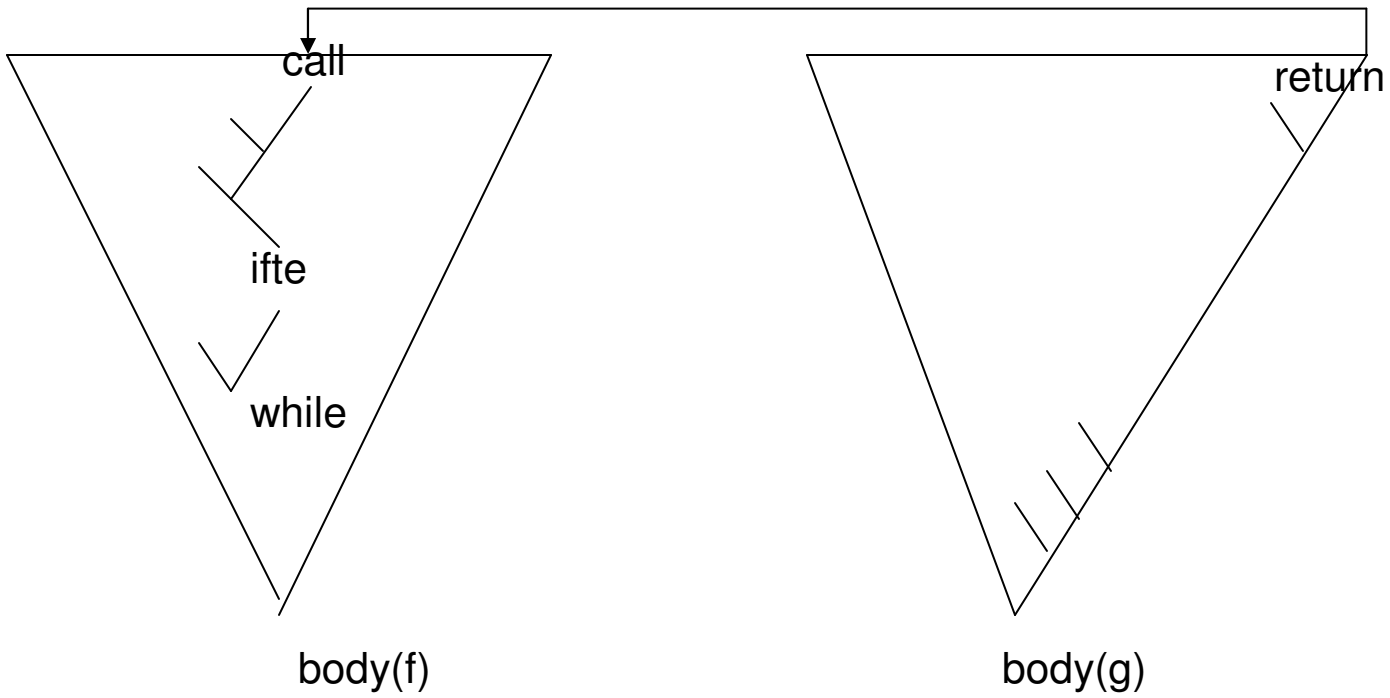
step by step simulation

For every c-computation $c^0, c^1, \ldots$ exist
DLX computation $d^0, d^1, \ldots$
step numbers $s^0, s^1, \ldots$
allocation funktions $alloc^0, alloc^1, \ldots$
such that for all T:
$consis(c^T, alloc^T, d^{s(T)})$

proof: induktion on T:

for c-consis: folklore theorem about second statement of program rest.

second statement of program rest



$C0_A$: C0 with in line assembler code

d: phys. DLX-configuration; Input

no in line code: $\delta_{C0A}(c, d) = \delta_C(c)$

new: $c.pr = asm(u); r'$:

$d \rightarrow_u^t d'$

$d = d_0 \rightarrow_{DLX} \ldots \rightarrow_{DLX} d_t = d'$

define $c_0, \ldots, c_j, \ldots c_t$:

$sw(d_j) \wedge ea(d_j) = alloc^0(c, x) \rightarrow$

$va(c_{j+1}, x) = d_j.GPR(RD(d_j))$

e.g. keeps data consistent

$\delta_{C0A}(c, d)^{i+1} = c_t$

CVM:Communicating Virtual Machines

- abstract (pseudo) parallel user model of the kernel
- cvm = (ca, ..., vm(i),...,vmsize(i),..., cp ,...)
  - ca: C0-configuration of abstract kernel k
  - vm(i): DLX-configuration of i'th user
  - cp = 0: kernel running (current process)
  - cp = i: vm(i) running
- parameter: kernel call definition
  - trap i calls funktion kcd(i) of kernel k
- No in line code in CVM: user processes visible in the parallel model !

CVM implementation: by konkrete kernel K ^{2} $C0_A$

- additional data structures of K
  - PCB[i]: process control block; save/restore registers
  - pt: page tables
  - spt: swap memory page tables
- formal theory of linking

...

CVM semantics and implementierung (1)

normal operation

$cp = 0$: $cvm'.ca = \delta_C(cvm.ca)$

$cp = i \wedge \not JISR(cvm.vm(i))$: $cvm'.vm(i) = \delta_D(cvm.vm(i))$

implementation

simulation of virtual maschines, possibly with

page fault handling,

CVM semantics and implementation (2)

start process $cp = 0 \wedge cvm.ca.pr = startnext; r'$:
cvm'.cp= va(cvm.ca, cp)
implementation
save registers of (compiled) kernel K in $PCB[0]$;
restore $vm(i)$ from $PCB[i]$; C0-Variable cp updated by scheduler

CVM semantics and implementation (3)

trap j by user i

let $f = kcd(j)$ handler for trap j

$n =$ number of parameters of f

$r = cvm.ca.pr$ old program rest

$$
\begin{aligned}
cvm'.ca.rd &= cvm'.ca.rd + 1 \\
cvm'.ca.pr &= cvm.ca.FT(f).body; r \\
top(cvm'.ca)(x) &= cvm.vm(i).GPR[x]
\end{aligned}
$$

trap has formal function call semantics

implementation

$f(PCB[i].GPR[1:n])$

trap implemented as function call

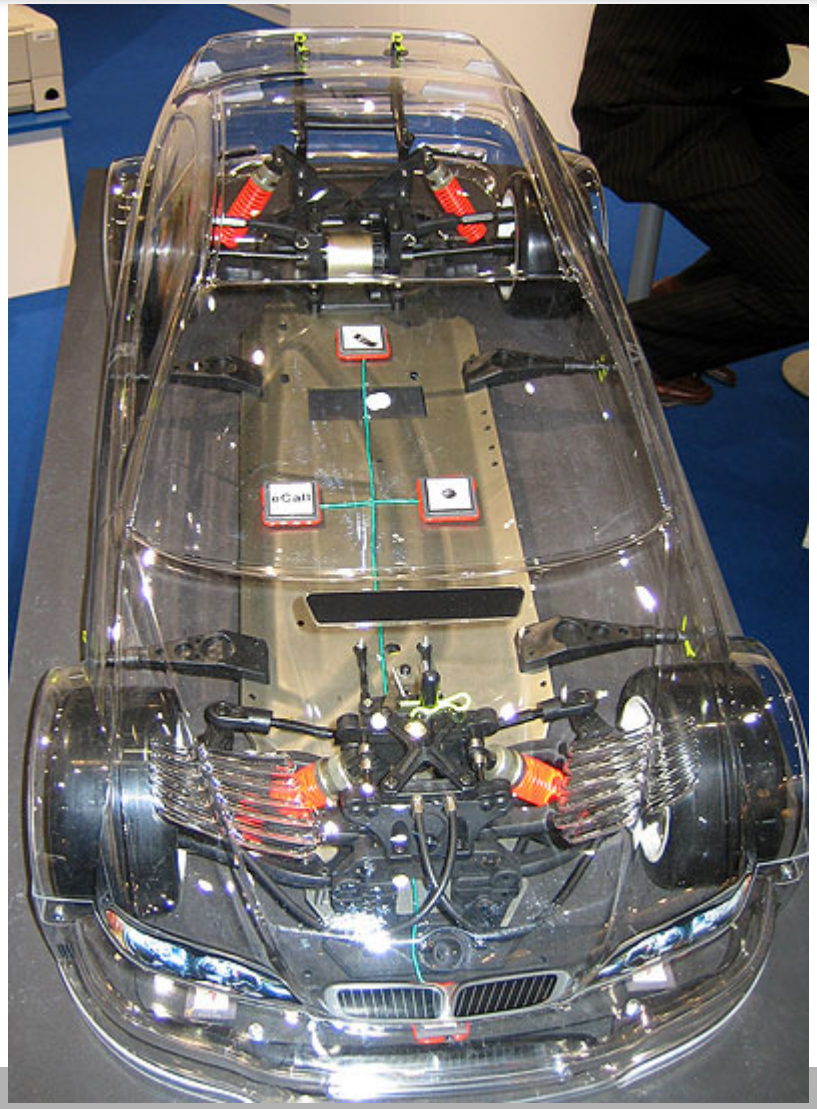
CVM correctness

- step by step simulation
- cp=0: compiler correctness
- cp>0: virtual memory simulation
- at borders (save/restore, startnext) or copy data between users: use in line assembler semantics
- induction with 3 computations:
  - cvm with abstr. kernel k and users vm(i)
  - phys. DLX
  - konkrete kernel K
- Formal induction hypothesis formulated

Induction step (4 dissertations)

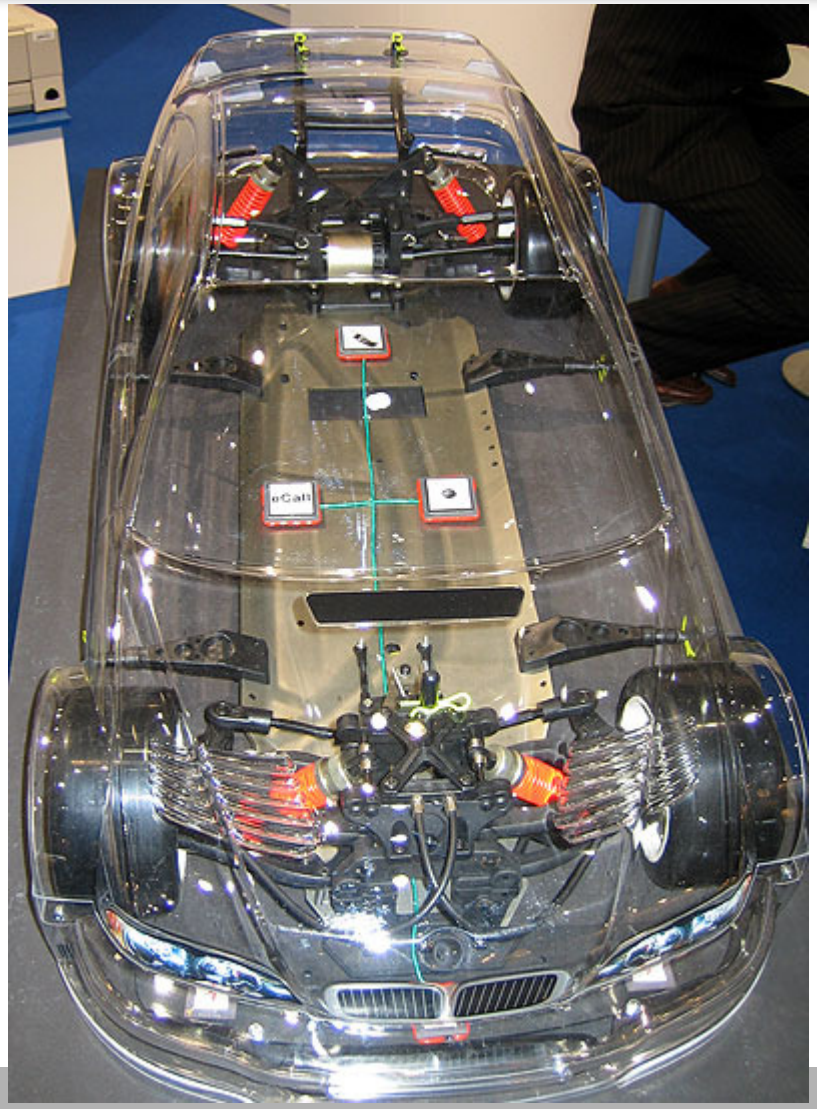
- Case: c.cp = c'.cp = 0 (system running)
  - C0 code: compiler + linker correctness
- Case: c.cp = c'.cp = u >0 (user running)
  - virtual memory simulation
  - case fault: handler/disk driver/C0 $_A$
- Case: c.cp ≠ c'.p (process switch user/system)
  - C0 $_A$ code
- Case: c.cp = 0; CVM primitive (e.g. copy)
  - C0 $_A$ code
- We are in the process of combining the formal proofs for the cases

automotive application e-call:automatic emergency call



- e-call exercises
  - CPUs
  - network interface (flex ray like)
  - drivers
  - real time operating system

Verisoft subproject: Automotive

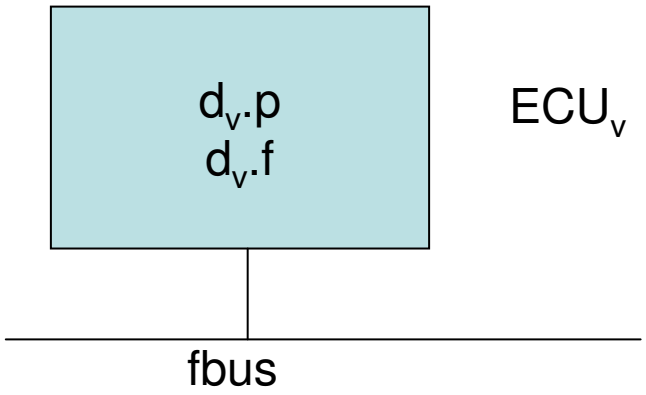


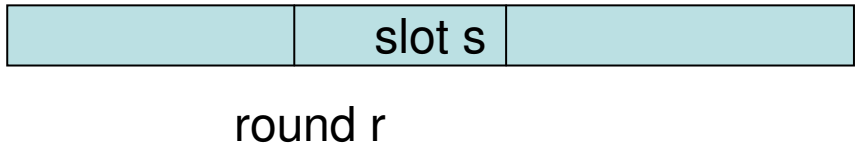
ecall (several ECU's)
FlexRay (like)
OSEKTime (like)
CVM (generic academic kernel)
TriCore2/VAMP processor
gates/registers

hardware details

- lecture notes
- my home page
  - http://www-wjp.cs.uni-sb.de
  - teaching
  - lectures
  - computer architecture 2 WS 05
  - bibliography
  - automotive
- these slides improve last lecture(s) of the notes

ISA programmers model I





FlexRay: ISA configuration of ECU_v

$d_v.p$: processor

$d_v.f$: f-interface

$sb(d_v)$: send buffer

$rb(d_v)$: receive buffer

Communication

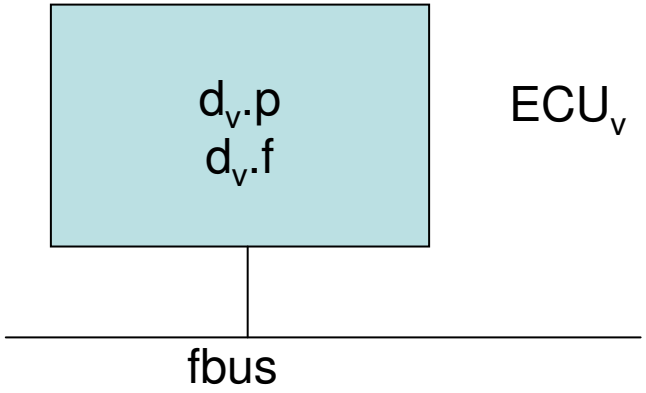
rounds r

slots $(r, 0) \ldots (r, ns - 1)$

Fixed schedule

$ECU_{send(s)}$ broadcasts send buffer in round (r, s) for all r

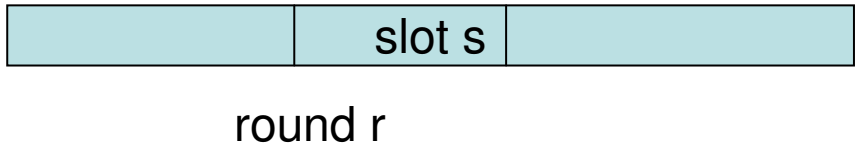
ISA programmers model II



Define for ECU_v in slot (r, s)
$d_v(r, s)$: first configuration
$e_v(r, s)$: last configuration

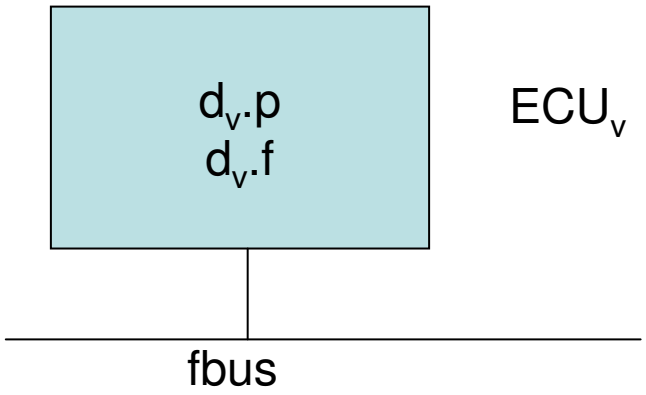
Want
1. for all u, r, s:
$rb(d_u((r, s) + 1))) = sb(e_{send(s)}((r, s) - 1))$

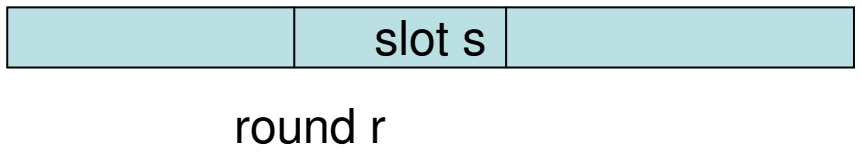
2.ISA correctness predicates
$Q(d_v(r, s), e_v(r.s))$



Goal:
justify this by hardware simulation theorem

solve 4 problems in 1 theory !





1. time:

τ_v: clock period on ECU_v

$|\tau_v - \tau| \leq \delta \cdot \tau$

$\rightarrow |\tau_u - \tau_v| \leq \Delta \cdot \tau_v$

$\delta = 0.15\%$

$\rightarrow$ serial interface

low level clock sync

2. f-interface

classical clock sync

3. timer interrupts

separate slots

cache penalties invisible in ISA

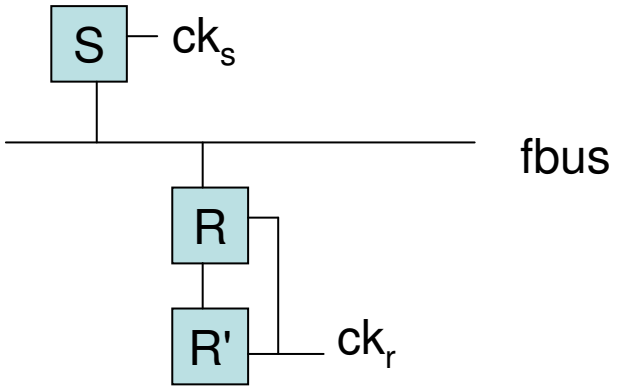
$\rightarrow$ ISA inherently nondeterministic

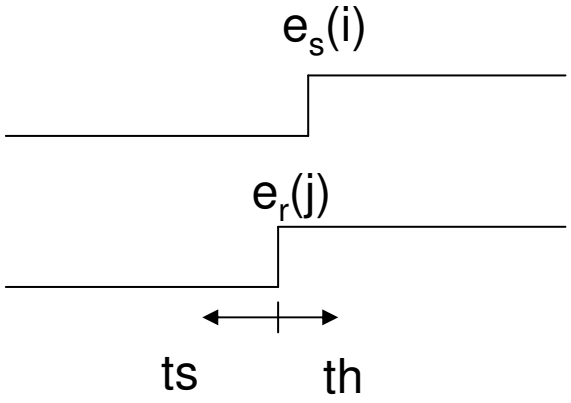
4. Remove nondeterminism by WCET

Pure WCET above RTL level of processor

- is either by measurements
  - guarantees usually nothing
- or
  - like guaranteeing a speed of at least 4.07 km/h for this car
- because:
  - cache penalties can affect execution time of an ISA intruction by factor 100

set up time, hold time, clock drift





R' removes metastability

clock edges

$e_v(i) = \alpha_v + i \cdot \tau_v$

relating cycles:

$cy_{s,r}(i) = \min\{j : e_r(j) + th > e_s(i)\}$

first receiver cycle possibly affected

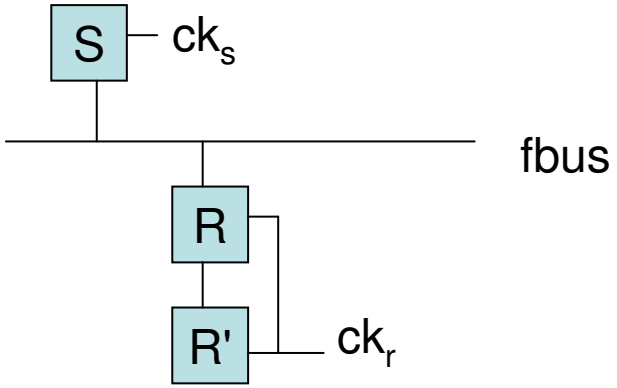
Lemmas:

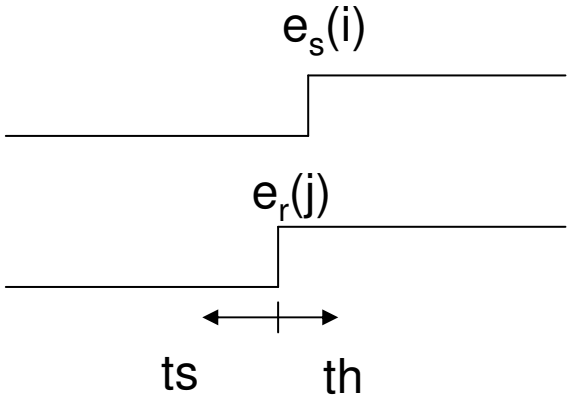
1. $S^i = \ldots = S^{i+7} \rightarrow$
$R^{cy(i)+k} = S^i$ for $k \in [1:6]$

2. $cy(i+1) \in [c(i) - 1 : cy(i) + 1]$

3. $cy(i+1) \neq cy(i) + 1 \rightarrow$
$cy(i+1+k) = cy(i+1) + k$ for $k \in [0:300]$

formal model for distributed hardware: discrete to continuous time





formal hardware model of distributed hardware:

v: index of an ECU

h_v: digital hardware configuration

t_v: number of cycle $(e_v(t_v), ev(t_v + 1)]$

$h_v^{t_v}$ hardware conf. in a cycle

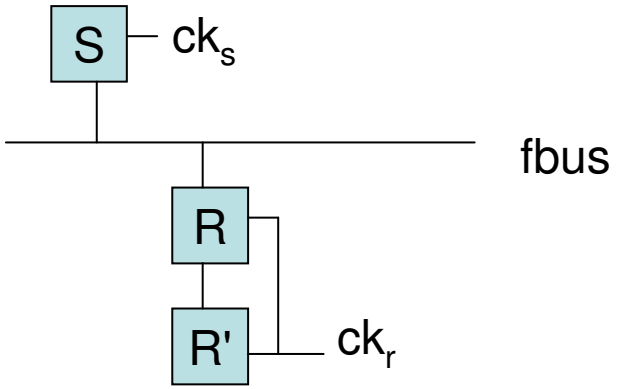
$S_v(t)$: sender register of ECU_v at time $t \in (e_v(t_v), ev(t_v + 1)]$

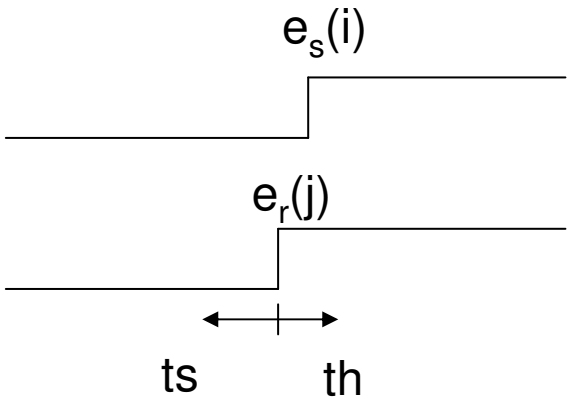
$$S_v(t) = \begin{cases} h_v^{t_v}.S & : \ /ce_v(h_v^{t_v}) \\ \Omega & : \ ce_v(h_v^{t_v}) \wedge t < e_v(t_v) + tpd \\ Sdin(h_v^{t_v}) & : \ t \geq e_v(t_v) + tpd \end{cases}$$

open collector bus:

$$fbus(t) = \bigwedge_v S_v(t)$$

formal model for distributed hardware: continuous time to discrete time





sampling into R_u at edge $e_u(t_u)$:
If $\exists x \in \{0,1\} \forall t \in (e_u(t_u) - ts, e_u(t_u) + th)$:
$fbus(t) = x$ then:

$$h_u^{t_u+1} = fbus(e_u(t_u))$$

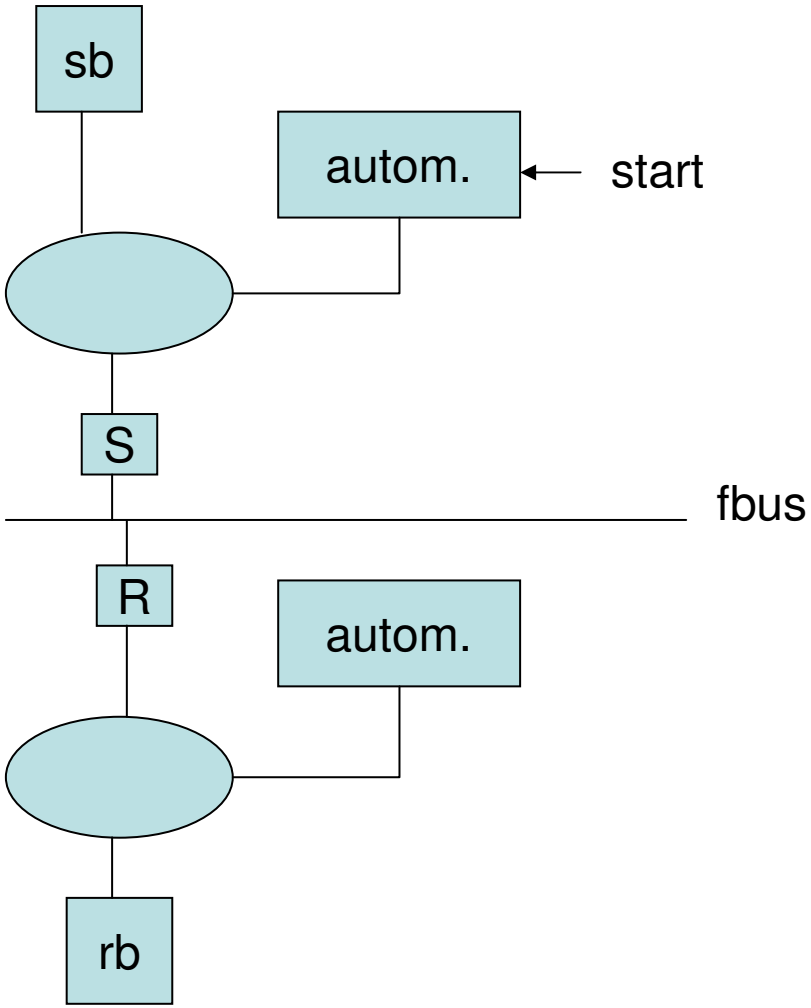
Otherwise

$$h_u^{t_u+1} = \Omega$$

ts: set up time
th: hold time

Demetastabilization
$h_u^{t_u}.R = \Omega \rightarrow h_u^{t_u+1}.R' \in \{0,1\}$

serial interface



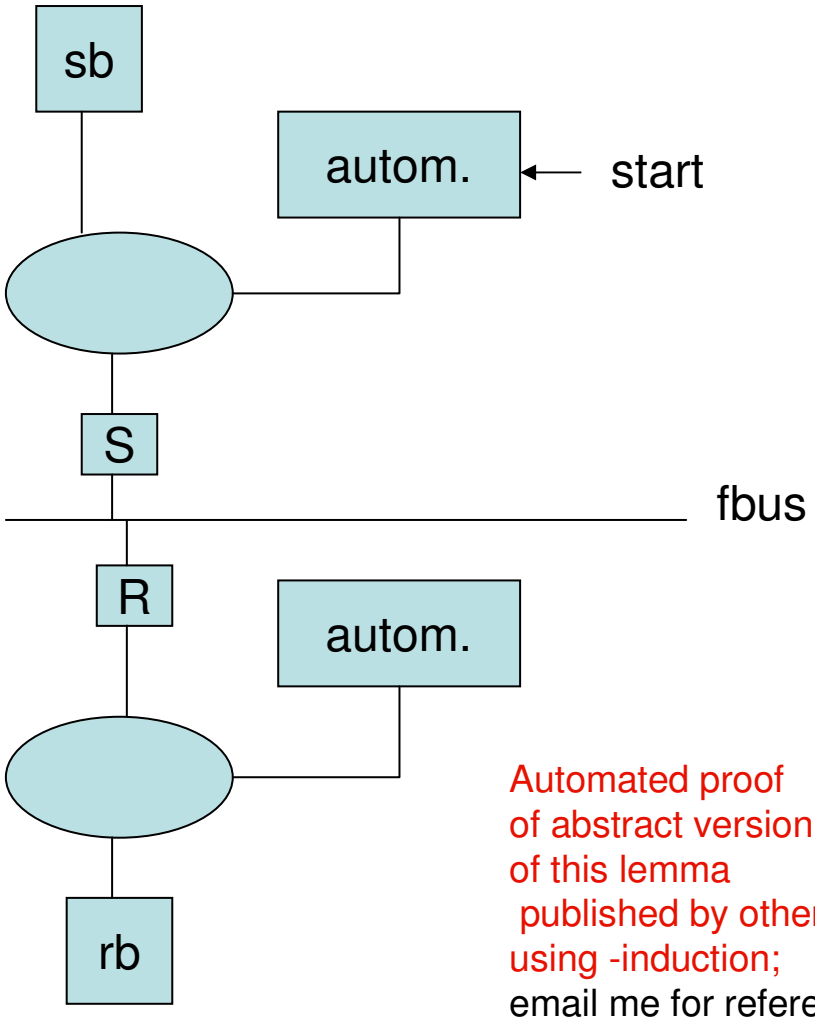
string $x[0:k-1]$ of bits:
$8x = x_0^8 \dots x_{k-1}^8$

message $m[0:\ell-1]$ of bytes:
$f(m) = 0110m[0]\dots10m[\ell-1]01$
sender sends $8f(m)$

define:
$sb(t), rb(t)$: at time t
t_0: activate start
t_1: done
$tc = 45 + 80 \cdot \ell$
Lemma:
1. $t_1 - t_0 = tc \cdot \tau_r$
2. $rb(t_1) = sb(t_0)$

proof: complex induction on
1. state (expecting sync edge)
2. sync (sampling in middle)
3. sampling

serial interface



Automated proof
of abstract version
of this lemma
published by others
using -induction;
email me for reference

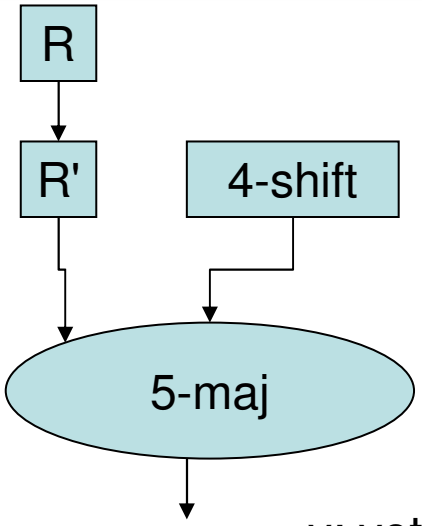
string $x[0:k-1]$ of bits:
$8x = x_0^8 \dots x_{k-1}^8$

message $m[0:\ell-1]$ of bytes:
$f(m) = 0110m[0] \dots 10m[\ell-1]01$
sender sends $8f(m)$

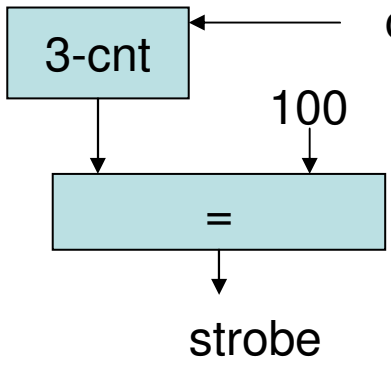
define:
$sb(t), rb(t)$: at time t
t_0: activate start
t_1: done
$tc = 45 + 80 \cdot \ell$
Lemma:
1. $t_1 - t_0 = tc \cdot \tau_r$
2. $rb(t_1) = sb(t_0)$

proof: complex induction on
1. state (expecting sync edge)
2. sync (sampling in middle)
3. sampling

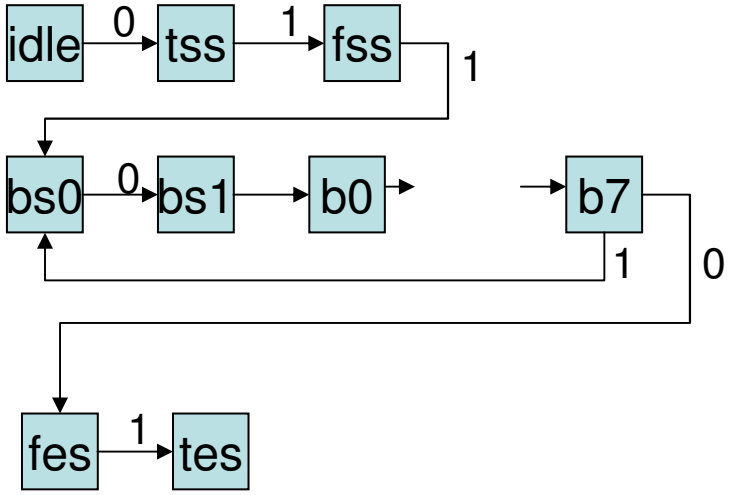
strobing a (voted) bit in 'middle' of 8 bits



v: voted bit



clear (= sync)



clock automaton with $strobe$
input of automaton: voted bit v

$$sync^t \leftrightarrow (idle^t \vee bs0^t) \wedge (v^t \wedge v^{t-1})$$