IAF0542

Sardsüsteemid III Loeng

Gert Jervan

Arvutitehnika instituut

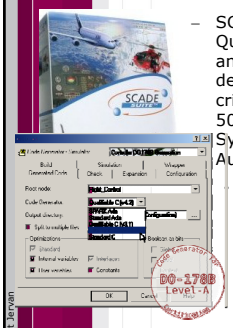
ati.ttu.ee/~gerje

www.pld.ttu.ee/IAF0542

Arvutusmudelid, millest räägime

Communication/ local computations	Shared memory	Message passing Synchronous | Asynchronous	
Undefined components	Plain text, use cases | (Message) sequence charts		
Communicating finite state machines	StateCharts		SDL
Data flow	(Not useful)		Kahn networks, SDF
Petri nets		C/E nets, P/T nets, …	
Discrete event (DE) model	VHDL, Verilog, SystemC, …	Only experimental systems, e.g. distributed DE in Ptolemy	
Von Neumann model	C, C++, Java	C, C++, Java with libraries CSP, ADA |	

Applications



- SCADE Suite, including the SCADE KCG Qualified Code Generator, is used by AIRBUS and many of its main suppliers for the development of most of the A380 and A400M critical on board software, and for the A340-500/600 Secondary Flying Command System, aircraft in operational use since August 2002.
  - François Pilarski, Systems Engineering Framework - Senior Manager Engineering,Systems & Integration Tests; Airbus France.

Instance of "model-based design"

Source: http://www.esterel-technologies.com/products/scade-suite/

SDL

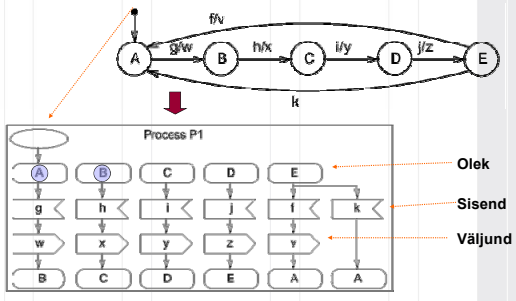
SDL

- Arvutusmudel, mis põhineb asünkroonsel sõnumite edastamisel
- Sobib muuhulgas ka hajussüsteemide jaoks

Kommunikatsioon/ arvutused	Jagatud mälu	Sõnumite edastamine	
		Blokeeriv	Mitteblokeeriv
FSM	StateCharts		SDL

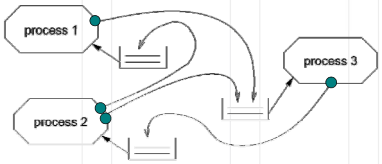
- Igat FSMi kutsutakse protsessiks
- SDL toetab operatsioone andmetega

FSMide/protsesside kujutamine SDL'iga



SDLi FSMide vaheline kommunikatsioon

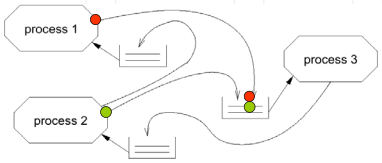
- FSMide (või „protsesside“) vaheline kommunikatsioon põhineb sõnumite edastamisel, eeldades lõpmatu suurusega FIFO puhvreid



- Iga protsess võtab FIFO järgmise elemendi
- Kontrollib, kas sisend vastab siirdele
- Kui jah: siire toimub
- Kui ei: sisendit ignoreeritakse

SDLi FSMide vaheline kommunikatsioon

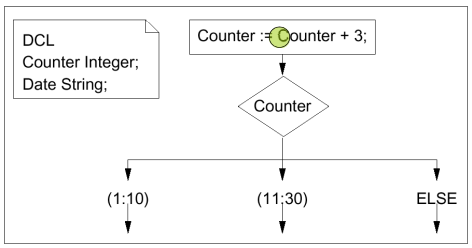
- Määramatus: samaaegse saabumise puhul on salvestamise järjekord teadmata



- Simulaatorid võivad anda erinevaid simulatsioonitulemusi, mis kõik võivad olla õiged

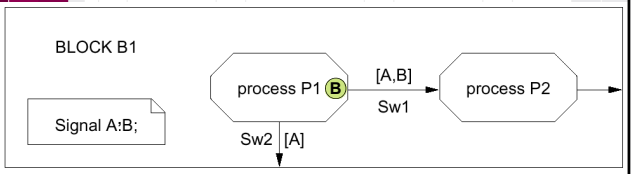
Operatsioonid andmetega

- Protsessidel võivad olla lokaalsed muutujad, tugi abstraktsetele andmetüüpidele



Protsesside suhtlusdiagrammid

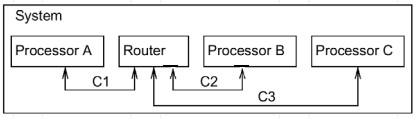
- Protsesside vahelise suhtlemise kirjeldamiseks võib kasutada suhtlusdiagramme (Blokkdiagrammide erijuht).
- Lisaks protsessidele, on nendel diagrammidel ka kanalid ja kohalikud signaalid

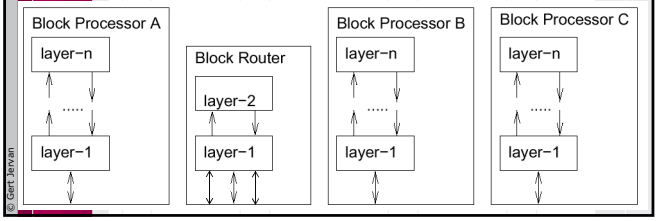


SDLi täiendavad võimalused

- Hierarhia
- Timerid
- Protseduurid
- Protsesside loomine ja lõpetamine
- Andmete kirjeldamine

Kasutamine: võrguprotokollide kirjeldamine

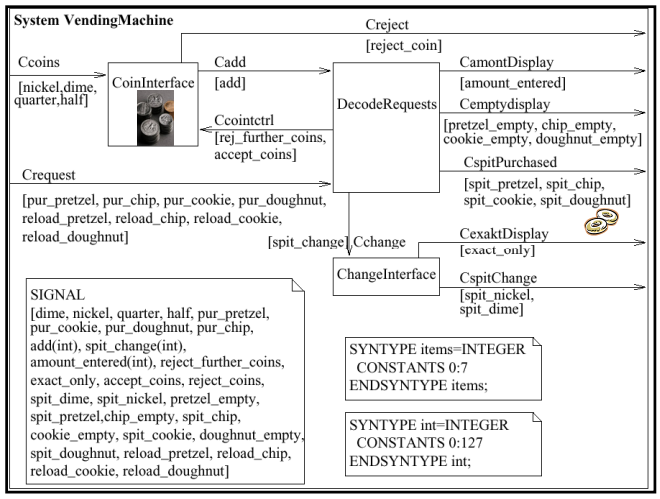




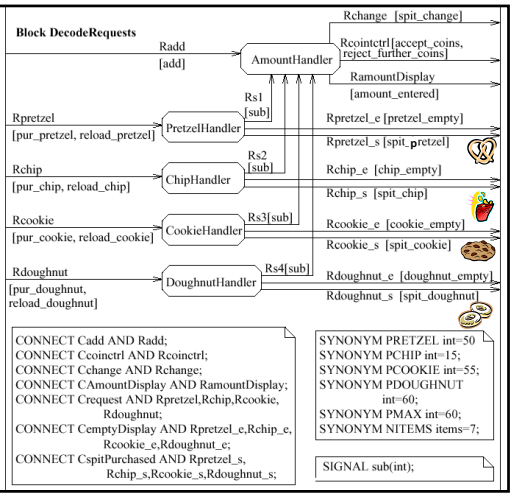
Suurema süsteemi näide: toiduautomaat

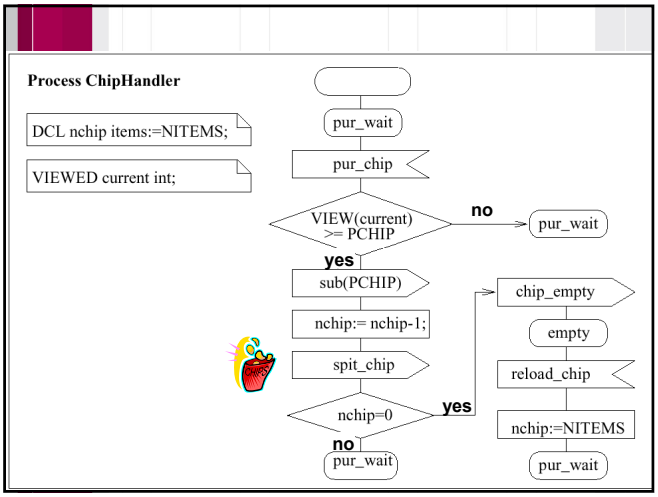
Masin, mis müüb erinevaid maiustusi
Aksepteerib erinevaid münte
Ei ole hajusrakendus

[J.M. Bergé, O. Levia, J. Roullard: High-Level System Modeling, Kluwer Academic Publishers, 1995]



Decode Requests





SDL

- Ideaalne hajusrakendustele (kasutati ISDNi spetsifitseerimisel),
- Tarkvara on saadaval: SINTEF, Telelogic, Cinderella (www.cinderella.dk).
- Ei ole täiesti deterministlik ja ei ole sünkroonne
- Implementatsiooni puhul on vaja teada FIFO maksimumsuurust – arvutamine võib olla väga keeruline
- Timeri põhimõte sobib vaid pehmetele reaalaja süsteemidele
- Hierarhiate kasutamine on limiteeritud
- Programmeerimiskeelte tugi on limiteeritud
- Mittefunktsionaalseid omadusi ei ole võimalik kirjeldada
- Rohkem infot: www.sdl-forum.org

Keelte võrdlus

Communication/ local computations	Shared memory	Message passing Synchronous | Asynchronous
Communicating finite state machines	StateCharts	SDL
Data flow model	Not useful	Kahn process networks
Von Neumann model	C, C++, Java	C, C++, Java with libraries CSP, ADA |
Discrete event (DE) model	VHDL, Verilog, SystemC	Only experimental systems, e.g. distributed DE in Ptolemy

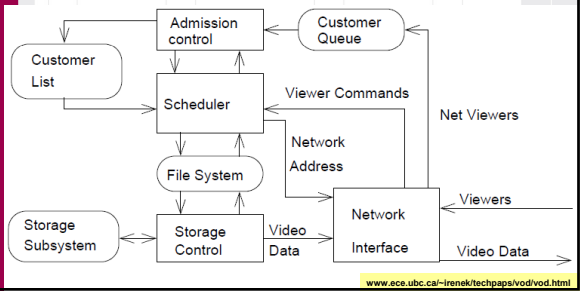
Andmevoo mudelid (Dataflow models)

Keelte võrdlus

Communication/ local computations	Shared memory	Message passing Synchronous | Asynchronous	
Communicating finite state machines	StateCharts		SDL
Data flow model	Not useful		Kahn process networks
Von Neumann model	C, C++, Java	C, C++, Java with libraries CSP, ADA |	
Discrete event (DE) model	VHDL, Verilog, SystemC	Only experimental systems, e.g. distributed DE in Ptolemy	

Andmevoo mudelid kui "loomulikud" mudelid

- Näide: VOD (video on demand)



Andmevoo mudelid

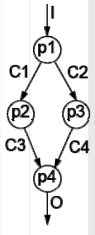
- Süsteemid on kirjeldatud, kui suunatud graafid, kus:
  - Sõlmed esitavad arvutusi (protsesse)
  - Kaared esitavad täielikult järjestatud andmevoogu
- Sõltuvalt semantikast on andmevoo põhjal defineeritud mitmeid erinevaid arvutusmudeleid:
  - Kahni protsessivõrgud (Kahn Process Networks - KPN)
  - Andmevoo protsessivõrgud (Dataflow process networks)
  - Sünkroonne andmevoog (Synchronous dataflow)
  - ...
- Andmete-põhine kattuvus

Andmevoo mudelid (2)

- Andmevoo mudelid on väga sobivad signaalitöötluses
  - Kodeerimine/dekodeerimine, filtreerimine, pakkimine jne
  - Perioodilised ja regulaarsed andmete lugemised
  - Tüüpiliselt on signaalitöötlusalgoritmid esitatud blokk-diagrammidena, mis sobib väga hästi andmevoo semantikaga

Andmevoo mudelid (3)

```
Process p1( in int a, out int x, out int y) {
.....
}
Process p2( in int a, out int x) {
.....
}
Process p3( in int a, out int x) {
.....
}
Process p4( in int a, in int b, out int x) {
.....
}
channel int I, O, C1, C2, C3, C4;
p1(I, C1, C2);
p2(C1, C3);
p3(C2, C4);
p4(C3, C4, O);
```

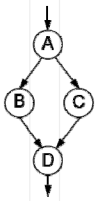


Protsesside sisemist andmetöötlust on võimalik kirjeldada suvalises programmeerimiskeeles (näiteks C)

Seda nimetatakse põhikeeleks (*host language*)

Aeg

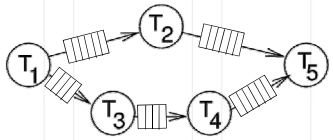
- Andmevoo mudelid on asünkroonselt kattuvad
  - Sündmused võivad toimuda igal ajal
  - On olemas sündmuste osaline järjestatavus
    - A poolt sümboli genereerimin toimub alati enne selle tarbimist B poolt
    - Ei ole mingit ette määratud järjekorda, kas sümboli tarbib enne B või C



KPN

- Protsesside suhtlemisel saadetakse andmeühikuid läbi ühesuunaliste FIFO kanalite
- Kanalisse kirjutamine on mitteblokeeriv
- Lugemine blokeeriv:
  - Protsess on blokeeritud kuni kanalis on piisav kogus andmeid
  - Protsess, mis proovib lugeda tühjast kanalist, peab ootama kuni andmed on saadaval. Ta ei saa enne lugemise alustamist, kas andmed on saadaval. Samuti ei saa ta ka tühja kanali korral lugemisest loobuda → **Determinism!**
  - Ainult üks lugeja ja kirjutaja per FIFO

KPN



- Kommunikatsioon ainult läbi FIFOde
- ≥1 sisendkanalist liigub informatsioon ≥1 väljundkanalisse
- Kanalite kaudu liigub informatsioon lõpliku kuid ettenägematu aja jooksul
- Üldjuhul on täitmisajad teadmata

KPN

- Kahni protsessivõrgud on deterministlikud:
  - Kindale sisendandmete kombinatsioonile vastab vaid üks võimalik väljundandmete kombinatsioon (sõltumata sellest, kui kaua võtavad mingid arvutused aega)
  - On võimalik vaid spetsifikatsiooni põhjal (teadmata midagi implementatsioonist) tuletada väljundjada, teades sisendandmeid
  - Ideaalne paljude sardsüsteemide disainiks kuna erinevad simulaatorid (aeglased ja kiired) ning riistvara prototüübid käituvad samamoodi

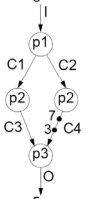

```
Process p1( in int a, out int x, out int y) {
int k;
loop
        k = a.receive();
        if k mod 2 = 0 then
                x.send(k);
        else
                y.send(k);
        end if;
end loop; }

Process p2( in int a, out int x) {
int k;
loop
        k = a.receive();
        x.send(k);
end loop; }

Process p3(in int a, in int b, out int x) {
int k; bool sw = true;
loop
        if sw then
                k = a.receive();
        else
                k = b.receive();
        end if;
        x.send(k);
        sw = !sw;
end loop; }

channel int I, O, C1, C2, C3, C4;
p1(I, C1, C2);
p2(C1, C3);
p2(C2, C4);
p3(C3, C4, O);
```

7 3 0 16 8 5 11 21 8 5



5 0 11 16 21 8 5 8

KPNide omadused

- KPNid on Turing-complete, i.e. kõike, mida saab arvutada, saab arvutada ka KPNidega
- Nende planeerimine on keeruline
- Nad on arvutuslikult võimsad kuid keerulised analüüsida (nnäiteks puhvrite suuruse küsimus)
- Protsesside arv on staatiline

Lisainfo ja katsetamine

- http://ls12-www.cs.tu-dortmund.de/daes/lehre/downloads/levi.html (Kahn Prozessnetzwerke Version 1.0.1 vom 14.11.2007 leviKPN.zip)
- http://en.wikipedia.org/wiki/Kahn_process_networks
- S. Edwards: http://www.cs.columbia.edu/~sedwards/classes/2001/w4995-02/presentations/dataflow.ppt

Petri võrgud (Petri Nets)

Petri võrgud

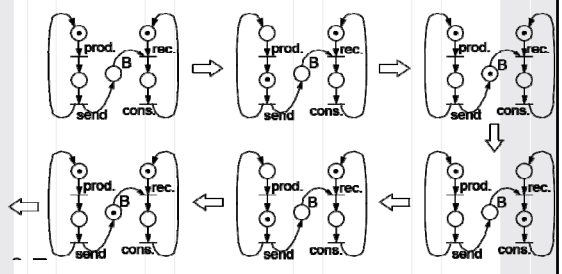
- Süsteem on spetsifitseeritud kui suunatud kahealuseline graaf, kus on kahte tüüpi sõlmi:
  - Koha sõlmed (places): Hoiavad hajutatud süsteemi olekut, mida väljendatakse märgi (token) olemasolu või puudumisega antud sõlmes
  - Üleminekud (transitions): Kasutatakse süsteemi toimimise tähistamiseks
- Süsteemi olek: kirjeldatakse koha sõlmede markeeringuga (märkide arv igas sõlmes)

Petri võrgud (2)

- Süsteemi dünaamiline areng on määratletud üleminekute käivitumisega
  - Üleminek võib käivituda kui kõik sellele eelnevad koha sõlmed on märgitud
  - Ülemineku käivitumisel likvideeritakse iga eelneva koha sõlme märgistus ja märgitakse kõik järgnevad sõlmed

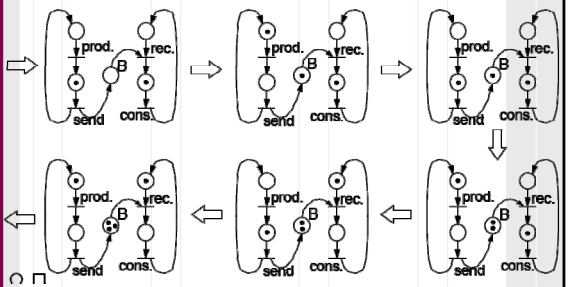
Petri võrgud näide

- Genereeriv ja vastuvõttev protsess suhtlevad läbi puhvri



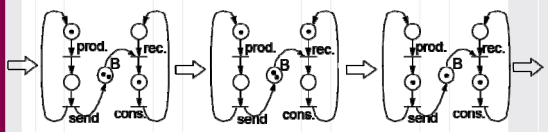
Petri võrgud näide (2)

- Näite jätk...



Petri võrgud näide (3)

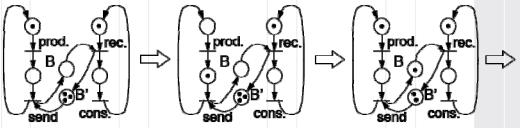
- Näite jätk...



- NB! Puhvri suurus on lõpmatu (märgid võivad sõlmes B koguneda)

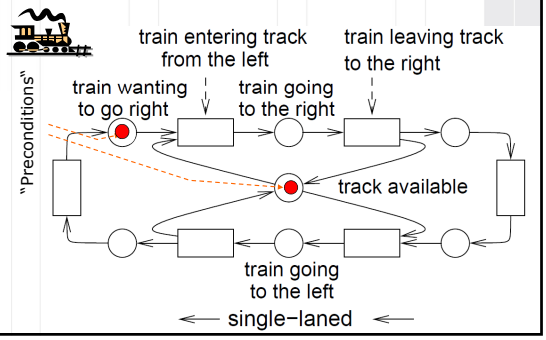
Petri võrgud näide (4)

- Sama näide, aga limiteeritud puhvriga. Puhvri suurus (esialgne märkide arv B'-is) on kolm

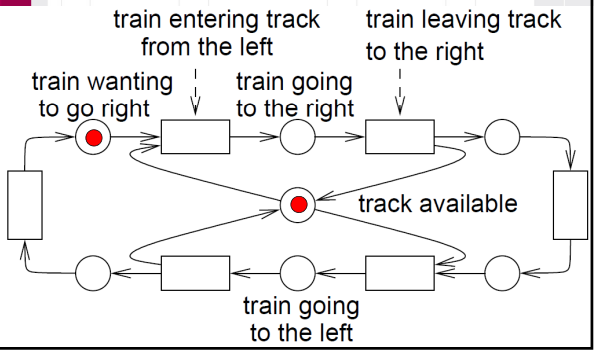


- Märkide koguarv B-s ja B'-s on konstantne

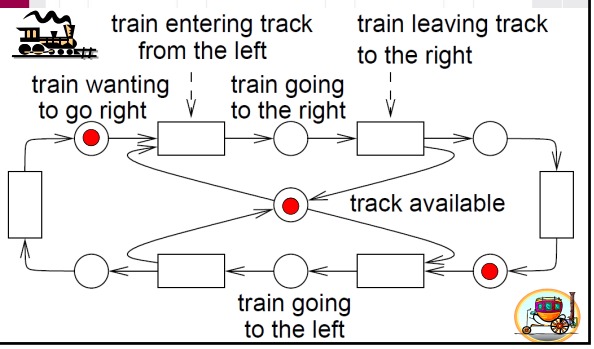
Example: Synchronization at single track rail segment



Playing the "token game"



Conflict for resource "track"



Petri võrkude tunnused ja kasutus

- Petri võrgud on intuitiivsed ja mitteinterpreteeritud mudel
- On palju kasutatud nii infosüsteemide arendamisel, kui ka arvutiarhitektuuride, operatsioonisüsteemide, hajussüsteemide ja riistvarasüsteemide loomisel

Petri võrkude omadused

- Saab kontrollide mitmeid süsteemi omadusi:
  - Piiratus (*Boundness*) – saab kontrollida, et etteantud ressursid ei oleks ületatud. Tokenite arv teatud kohas. Kui piirang on 1, siis seda kutsutakse vahel ka ohutuseks (safeness)
  - Elus olemine (*Liveness*) – Et vältida deadlock'e. Üleminek on elus, kui iga võimaliku märgistuse puhul on võimalik selle ülemineku aktiveerumine
  - Saavutatavus (*Reachability*) – Et jõutakse vajalikku olekusse või et mõnda olekusse kunagi ei jõutaks. Kas on võimalik liikuda ühest märgistusest teise?
- Spetsiaalsed matemaatilised töövahendid. Formaalne verifitseerimine.

Petri võrkude omadused (2)

- Petri võrgud on asünkroonselt samaaegsed
  - Sündmused võivad toimuda igal ajal
  - On olemas sündmuste osaline järjestus
- Laiendused:
  - Ajalised Petri võrgud (aja aspektide modelleerimiseks)
    - Ülekannetega on seotud ajad (aja intervallid)
    - Märgid kannavad ajamärgistust
  - Värvitud Petri võrgud
    - Märkidel on väärtused
    - Ülekannetega on seotud funktsioonid
- Petri võrke saab simuleerida, et süsteemi verifitseerida ja hinnata suutlikust

Discrete Event Models

Keelte võrdlus

Communication/ local computations	Shared memory	Message passing Synchronous | Asynchronous	
Communicating finite state machines	StateCharts		SDL
Data flow model	Not useful		Kahn process networks
Von Neumann model	C, C++, Java	C, C++, Java with libraries CSP, ADA |	
Discrete event (DE) model	VHDL, Verilog, SystemC	Only experimental systems, e.g. distributed DE in Ptolemy	

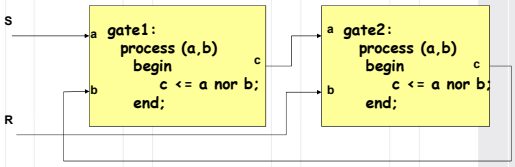
DEM

- Süsteem on protsesside kogum, mis reageerib sündmustele
- Iga sündmusega on seotud ajatempel, mis näitab selle sündmuse toimumise aega
- Ajatemplid on täielikult reastatud
- Diskreetne sündmusesimulaator peab globaalset sündmuste järjekorda, mis on sorteeritud ajatemplite põhiselt. Simulaator defineerib ka globaalse ühtse aja
- Mudelid on asünkroonsed ja samaaegsed
- Kasutusel riistvara kirjelduskeeltes, näiteks VHDL, Verilog, SystemC

VHDL

- VHDL = VHSIC hardware description language
- VHSIC = very high speed integrated circuit
- 1980: Def. started by US Dept. of Defense (DoD) in 1980
- 1984: first version of the language defined, based on ADA
  (which in turn is based on PASCAL)
- 1987: revised version became IEEE standard 1076
- 1992: revised IEEE standard
- 1999: VHDL-AMS: includes analog modeling
- 2006: Major extensions

Lihtne näide (VHDL)



- Protsessid ootavad muutusi sisendportidel
- Nende tekkimisel protsessid "ärkavad", teostavad oma arvutused ja salvestavad väljundsignaalide muutuse sündmuste ootejärjekorras (*event queue)* ja ootavad järgmist sündmust
- Kui kõik protsessid on ootamas, siis võetakse sündmuste ootejärjekorrast järgmine sündmus

VHDL processes

Delays allowed:

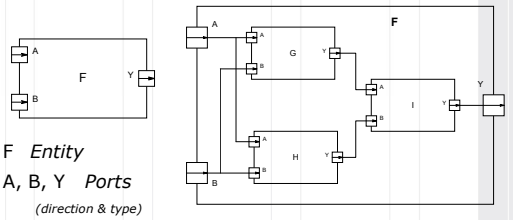
```
process (a,b)
 begin
   c <= a nor b after 10 ns;
 end;
```

Equivalent to:

```
process
    begin
     c <= a nor b after 10 ns;
     wait on a,b;
    end;
```

- <=: signal assignment operator
- Each executed signal assignment will result in adding entries in the projected waveform, as indicated by the (optional) delay time
- Implicit loop around the code in the body
- Sensitivity lists are a shorthand for a single wait on-statement at the end of the process body

VHDL - Describing Structure



F *Entity*

A, B, Y *Ports*

(direction & type)

G, H, I *Instances*

VHDL - Describing Behavior

- Not always the internal structure of the module is needed - only description of the function

$$Y = \overline{A} \bullet B + A \bullet \overline{B}$$

- Behavior of a digital system could be defined in terms of programming language notion

VHDL - Discrete Event Time Model

- Behavioral description can be executed by simulating the passage of time in discrete steps
- Some values will be feeded to the input
- New output values are calculated by the module - transactions
- Two phase simulation: initialization phase and two-stage simulation cycle
- System can be monitored during the simulation

VHDL - An Example

Two-bit counter

- Interface definition

```
entity count2 is
  generic (prop_delay : Time := 10 ns);
  port (clock : in bit;
        q1, q0 : out bit);
end count2;
```

VHDL - An Example

– Behavioral description

```
architecture behavior of count2 is
  begin
    count_up : process (clock)
      variable count_value : natural := 0;
    begin
      if clock = '1' then
        count_value := (count_value + 1) mod 4;
        q0 <= bit'val(count_value mod 2) after prop_delay;
        q1 <= bit'val(count_value / 2) after prop_delay;
      end if;
    end process count_up;
  end behavior;
```


VHDL - An Example

```
architecture structure of count2 is
  component t_flipflop
      port (ck : in bit; q : out bit);
  end component;

  component inverter
      port (a : in bit; y : out bit);
  end component;

  signal ff0, ff1, inv_ff0 : bit;

begin
  bit_0 : t_flipflop port map (ck => clock, q => ff0);
  inv : inverter port map (a => ff0, y => inv_ff0);
  bit_1 : t_flipflop port map (ck => inv_ff0, q => ff1);
  q0 <= ff0;
  q1 <= ff1;
end structure;
```


VHDL: Evaluation

- Behavioral hierarchy (procedures and functions),
- Structural hierarchy: through structural architectures,
  but no nested processes,
- No specification of non-functional properties,
- No object-orientation,
- Static number of processes,
- Complicated simulation semantics,
- Too low level for initial specification,
- Good as an intermediate "Esperanto" or "assembly" language for hardware generation.

Imperatiivsed keeled

C, SystemC, Java, ...

Keelte võrdlus

Communication/ local computations	Shared memory	Message passing Synchronous	Asynchronous
Communicating finite state machines	StateCharts		SDL
Data flow model	Not useful		Kahn process networks
Von Neumann model	C, C++, Java	C, C++, Java with libraries CSP, ADA	
Discrete event (DE) model	VHDL, Verilog, SystemC	Only experimental systems, e.g. distributed DE in Ptolemy	

C kasutamine sardsüsteemide loomisel

– Motivatsioon

- Paljud standardid (näiteks GSM, MPEG) on publitseeritud C programmidena
  - Riistvara kirjelduskeele (näiteks VHDL) kasutamiseks peaks standardeid hakkama "tõlkima"
- Paljude süsteemide funktsionaalsus eeldab nii riistvara kui ka tarkvara
  - Simuleerimine nõuaks vastavaid liideseid, kui just sama keelt ei kasutata
- On proovitud kirjeldada riistvara ja tarkvara, lähtudes samast keelest. See ei olegi nii lihtne → Erinevad C dialektid riistvara kirjeldamiseks

C/C++ puudused

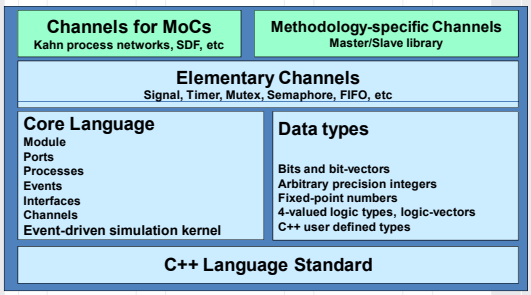
- C/C++ ei ole loodud riistvara disainiks
- C/C++ ei toeta:
  - Riistvara stiilis kommunikatsiooni – signaalid, protokollid
  - Aja mõistet – taktsignaal, operatsioonide ajaline järjestus
  - Kattuvus – Riistvara töötab paralleelselt
  - Reaktiivsus – Riistvara reageerib välistele andmetele, suhtleb keskkonnaga
  - Riistvaralised andmetüübid – bit, mitmeväärtuseline loogika
- Silumise käigus on ligipääs riistvarale keeruline

*** Good to know VHDL ☺**

SystemC

- Requirements, solutions for modeling HW in a SW language:
  - **C++ class library including required functions.**
  - **Concurrency**: via processes, controlled by sensivity lists* and calls to wait primitives.
  - **Time**: Floating point numbers in SystemC 1.0. Integer values in SystemC 2.0; Includes units such as ps, ns, µs etc*.
  - **Support of bit-datatypes**: bitvectors of different lengths; 2- and 4-valued logic; built-in resolution*)
  - **Communication**: plug-and-play (pnp) channel model, allowing easy replacement of intellectual property (IP)
  - Deterministic behavior not guaranteed.

SystemC arhitektuur



Transaction-based modeling

- Definition: "**Transaction-level modeling (TLM)** is a high-level approach to modeling digital systems where details of communication among modules are separated from the details of the implementation of functional units or of the communication architecture.
- Communication mechanisms such as buses or FIFOs are modeled as channels, and are presented to modules using SystemC interface classes. Transaction requests take place by calling interface functions of these channel models, which encapsulate low-level details of the information exchange.
- At the transaction level, the emphasis is more on the functionality of the data transfers - what data are transferred to and from what locations - and less on their actual implementation, that is, on the actual protocol used for data transfer.
- This approach makes it easier for the system-level designer to experiment, for example, with different bus architectures (all supporting a common abstract interface) without having to recode models that interact with any of the buses, provided these models interact with the bus though the common interface."

Grötker et al., 2002

Verilog

- HW description language competing with VHDL
- Standardized:
  - IEEE 1364-1995 (Verilog version 1.0)
  - IEEE 1364-2001 (Verilog version 2.0)
  - Features similar to VHDL:
    - Designs described as connected entities
    - Bitvectors and time units are supported
  - Features that are different:
    - Built-in support for 4-value logic and for logic with 8 strength levels encoded in two bytes per signal.
    - More features for transistor-level descriptions
    - Less flexible than VHDL.
    - More popular in the US (VHDL common in Europe)

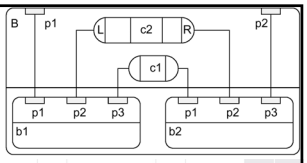
SystemVerilog

- Corresponds to Verilog versions 3.0 and 3.1. Includes:
  - Additional language elements for modeling behavior
  - C data types such as int
  - Type definition facilities
  - Definition of interfaces of HW components as entities
  - Mechanism for calling C/C++-functions from Verilog
  - Limited mechanism for calling Verilog functions from C.
  - Enhanced features for describing the testbench
  - Dynamic process creation.
  - Interprocess communication and synchronization
  - Automatic memory allocation and deallocation.
  - Interface for formal verification.

SpecC [Gajski, Dömer et. al. 2000]

- SpecC is based on the clear separation between communication and computation. Enables „plug-and-play" for system components; models systems as hierarchical networks of behaviors communicating through channels.
- Consists of behaviors, channels and interfaces.
- Behaviors include ports, locally instantiated components, private variables and functions and a public main function.
- Channels encapsulate communication. Include variables and functions, used for the definition of a communication protocol.
- Interfaces: linking behaviors and channels. Declare communication protocols (defined in a channel).

SpecC




```
interface L {void Write(int x);};
interface R {int Read (void);};
channel C implements L,R
 { int Data; bool Valid;
   void Write(int x)  {Data=x; Valid=true;}
   int Read(void) {while (!Valid) waitfor(10); return (Data);}
 };
behavior B1 (in int p1, L p2, in int p3)
  {void main(void)    {/*...*/ p2.Write(p1);} };
behavior B2 (out int p1, R p2, out int p3)
  {void main(void)   {/*...*/ p3=p2.Read();   } };
behavior B(in int p1, out int p2)
  { int c1; C c2; B1 b1(p1,c2,c1); B2 b2 (c1,c2,p2);
    void main (void)
    {par {b1.main();b2.main();}}
  };
```


Java

- Eelised
  - "Ohutu" keel
    - Puudub viidaaritmeetika
    - Toetab eranditöötlust (*exception handling*)
    - Kasutaja põhjustatud mälulekete puudumine
  - Toetab kattuvust (*light-weight processes*)
  - Platvormist sõltumatu
    - Väga kompaktne *byte-code* (kompaktsem, kui teiste keelte masinkood)

Java

- Puudused
  - *Run-time library'te* suurus
  - Ligipääs spetsiaalsele riistvarale (otsene I/O kontroll)
  - Automaatne mälukoristus
  - Mittedeterministlik threadide käivitamine (WCET hinnangud peavad olema väga pessimistlikud)
  - Real-aja mõiste hägusus

Veel keeli...

- Sequence diagrams
- UML
- Pearl
- Chill
- Estelle
- Silage
- Esterel
- Matlab
- Simulink
- StateFlow
- Lotos, Z

Levels of hardware modeling

- Possible set of levels (others exist)
  - System level
  - Algorithmic level
  - Instruction set level
  - Register-transfer level (RTL)
  - Gate-level models
  - Switch-level models
  - Circuit-level models
  - Device-level models
  - Layout models
  - Process and device models

System level

- Term not clearly defined.
- Here: denotes the entire embedded system, system into which information processing is embedded, and possibly also the environment.
- Models may include mechanics + information processing.
  May be difficult to find appropriate simulators.
  Solutions: VHDL-AMS, SystemC or MATLAB.
  MATLAB+VHDL-AMS support partial differential equations.
- Challenge to model information processing parts of the system such that the simulation model can be used for the synthesis of the embedded system.

Algorithmic level

- Simulating the algorithms that we intend to use within the embedded system.
- No reference is made to processors or instruction sets.
- Data types may still allow a higher precision than the final implementation.
- If data types have been selected such that every bit corresponds to exactly one bit in the final implementation, the model is said to be bit-true. non-bit-true → bit-true should be done with tool support.
- Single process or sets of cooperating processes.

Algorithmic level: Example: -MPEG-4 full motion search -

```
for (z=0; z<20; z++)
 for (x=0; x<36; x++) {x1=4*x;
  for (y=0; y<49; y++) {y1=4*y;
   for (k=0; k<9; k++) {x2=x1+k-4;
    for (l=0; l<9; ) {y2=y1+l-4;
     for (i=0; i<4; i++) {x3=x1+i; x4=x2+i;
      for (j=0; j<4;j++) {y3=y1+j; y4=y2+j;
       if (x3<0 || 35<x3||y3<0||48<y3)
        then_block_1; else else_block_1;
       if (x4<0|| 35<x4||y4<0||48<y4)
        then_block_2; else else_block_2;
}}}}}}
```


Instruction level

- Algorithms already compiled for the instruction set. Model allows counting the executed number of instructions.
- Variations:
  - Simulation only of the effect of instructions
  - Transaction-level modeling: each read/write is one transaction, instead of a set of signal assignments
  - Cycle-true simulations: exact number of cycles
  - Bit-true simulations: simulations using exactly the correct number of bits

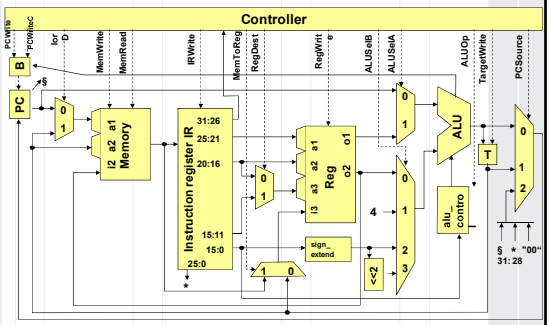
Instruction level: example

Assembler (MIPS)	Simulated semantics
`and $1,$2,$3`	`Reg[1]:=Reg[2] ∧ Reg[3]`
`or $1,$2,$3`	`Reg[1]:=Reg[2] ∨ Reg[3]`
`andi $1,$2,100`	`Reg[1]:=Reg[2] ∧ 100`
`sll $1,$2,10`	`Reg[1]:=Reg[2] << 10`
`srl $1,$2,10`	`Reg[1]:=Reg[2] >> 10`

Register transfer level (RTL)

- Modelling of all components at the register-transfer level, including
  - arithmetic/logic units (ALUs),
  - registers,
  - memories,
  - muxes and
  - decoders.
- Models at this level are always cycle-true.
- Automatic synthesis from such models is not a major challenge.

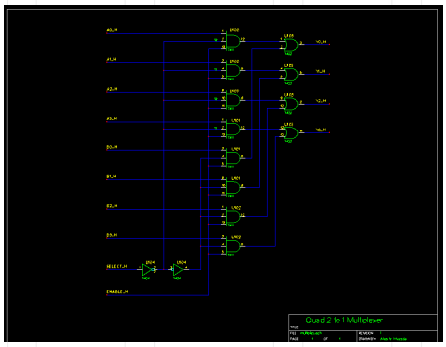
Register transfer level: example (MIPS)



Gate-level models

- Models contain gates as the basic components.
- Information about signal transition probabilities
  ☞ can be used for power estimations.
- Delay calculations can be more precise than for RTL. Typically no information about the length of wires (still estimates).
- Term sometimes also denotes Boolean functions (No physical gates; only considering the behavior of the gates).
  Such models should be called "Boolean function models".

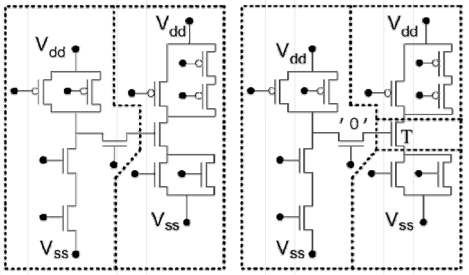
Gate-level models: Example



Switch-level models

- Switch level models use switches (transistors) as their basic components.
- Switch level models use digital values models.
- In contrast to gate-level models, switch level models are capable of reflecting bidirectional transfer of information.

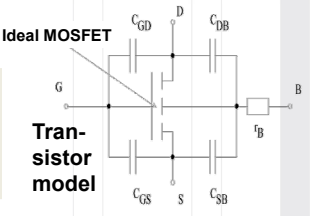
Switch level model: example



Circuit level models: Example

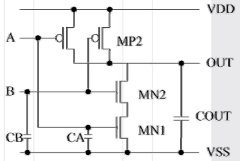
- Models circuit theory. Its components (current and voltage sources, resistors, capacitances, inductances and possibly macro-models of semiconductors) form the basis of simulations at this level.

Simulations involve partial differential equations.
Linear if and only if the behavior of semiconductors is linearized.



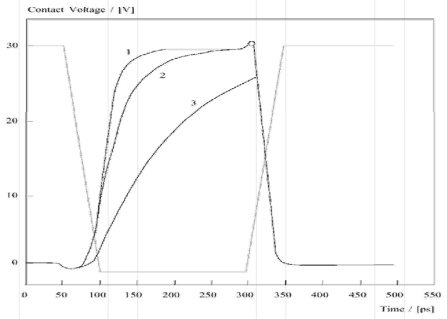
Circuit level models: SPICE

The most frequently used simulator at this level is SPICE [Vladimirescu, 1987] and its variants.
Example:



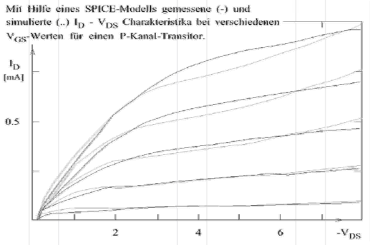

```
.SUBCKT NAND2 VDD VSS A B OUT
MN1 I1 A VSS VSS NFET W=8U L=4U AD=64P AS=64P
MN2 OUT B I1 VSS NFET W=8U L=4U AD=64P AS=64P
MP1 OUT A VDD VDD PFET W=16U L=4U AD=128P AS=128P
MP2 OUT B VDD VDD PFET W=16U L=4U AD=128P AS=128P
CA A VSS 50fF
CB B VSS 50fF
COUT OUT VSS 100fF
.ENDS
```


Circuit level models: sample simulation results



Device level

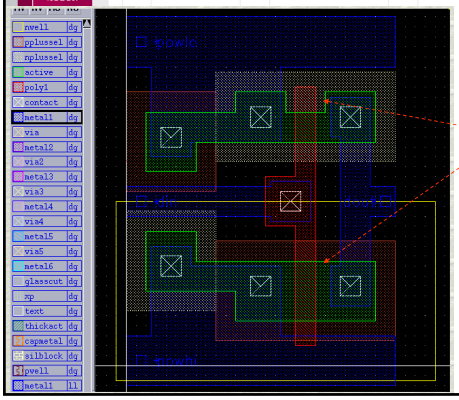
- Simulation of a single device (such as a transistor).
- Example (SPICE-simulation [IMEC]):

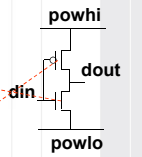


Layout models

- Reflect the actual circuit layout,
- include geometric information,
- cannot be simulated directly:
  behavior can be deduced by correlating the layout model with a behavioral description at a higher level or by extracting circuits from the layout.
- Length of wires and capacitances frequently extracted from the layout,
  back-annotated to descriptions at higher levels (more precision for delay and power estimations).

Layout models: Example

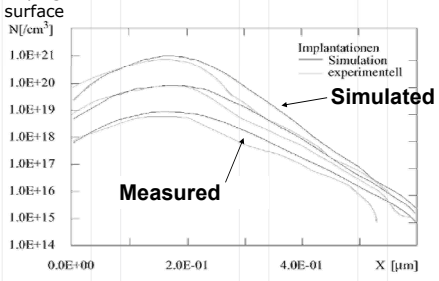




© Mosis (http://www.mosis.org/Technical/Designsupport/polyflowC.html); Tool: Cadence

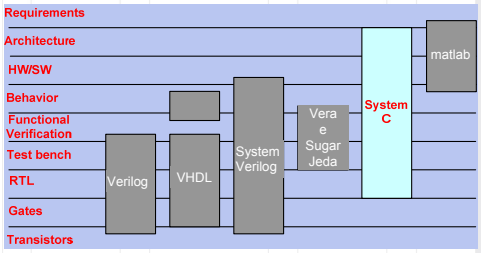
Process models

- Model of fabrication process; Example [IMEC]: Doping as a function of the distance from the surface



Movie (German)

Keeled ja abstraktsioonitasemed

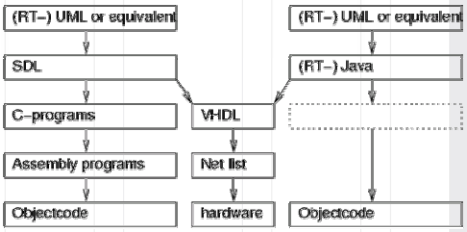


Keelde võrdlus

Language	Behavioral Hierarchy	Structural Hierarchy	Programming Language Elements	Exceptions Supported	Dynamic Process Creation
StateCharts	+	-	-	+	-
VHDL	+	+	+	-	-
SpecCharts	+	-	+	+	-
SDL	+-	+-	+-	-	+
Petri nets	-	-	-	-	+
Java	+	-	+	+	+
SpecC	+	+	+	+	+
SystemC	+	+	+	- (2.0)	- (2.0)
ADA	+	-	+	+	+

Keelte kasutamine praktikas

Erinevad lähenemised:



Milline modelleerimissuund valida?

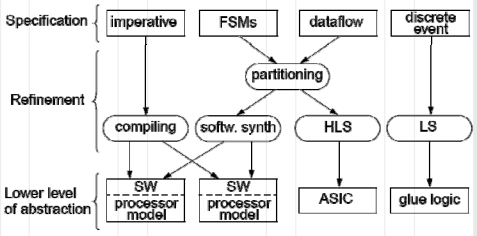
- Kõik sõltub loodavast süsteemist
  - Kas domineerib andmevoog (nagu näiteks DSPdes) või on tegemist kontrollile orienteeritud süsteemidga (reaktiivsed süsteemid)?
  - Sünkroonne või asünkroonne? Tsentraliseeritud või hajutatud?
  - Kui suur?
  - Milline on suhe aega?
  - ...
- Mida sa tahad mudeliga teha?
  - Simuleerimine
  - Formaalne verifitseerimine
  - Automaatne süntees

Milline modelleerimissuund valida? (2)

- Ära kasuta seda, mis võimaldab teha "kõike"!
- Loengust:
  - Suur väljendusvõime: imperatiivne mudel (näiteks C või Java piiranguteta kasutamine):
    - Võime kirjeldada "kõike"
    - Puudub võimalus formaalseks analüüsiks (või see on väga keerukas)
  - Piiratud väljendusvõime, mis põhineb hästi valitud arvutusmudelil:
    - Spetsifitseerida saab ainult valitud süsteeme
    - Formaalne analüüs on võimalik
    - Efektiivse (võib-olla isegi automaatse) sünteesi võimalus

Milline modelleerimissuund valida? (3)

- Suured sardsüsteemid on heterogeensed → mudelite segu



Keelte kasutamine

- Keele valik on suuresti seotud kasutatava modelleerimismeetodiga
  Seda seetõttu, et paljud keeled on väga tugevalt seotud mingi kindla arvutusmudeliga
  - Kommunikeeruvad asünkroonsed olekuautomaadid: SDL, Lotos
  - Sünkroonsed süsteemid: Esterel, StateCharts
  - Andmevoog ja pidevad arvutused: Matlab, Lustre, Silage

Keelte kasutamine (2)

- Osad keeled aga ei põhine ühelgil kindlal arvutusmudelil
  - Tavalised programmeerimiskeeled
    - Imperatiivsed: C, C++, Java, Ada
    - Funktsionaalsed: Lisp, Scheme, Haskell
    - Loogika: Prolog
  - Riistvara kirjelduskeeled
    - VHDL, Verilog, SystemC: imperatiivsed, diskreetsed sündmused
- Kui spetsifikatsioonide kirjeldamiseks kasutatakse teatud piiranguid ning suuniseid, siis nendes keeltes saab kirjeldada enamuste arvutsumudelite põhiseid spetsifikatsioone

Kokkuvõtteks

- Disaini võib vaadelda kui järjestike täpsustavate sammude kogumit, mis viib spetsifikatsioonist implementatsioonini
- Spetsifikatsioone kirjeldatakse spetsifikatsioonikeeltes
- Me tahaksime, et spetsifikatsioonikeelel oleks hästi defineeritud semantika
- Süsteemide kirjeldamiseks kasutatavate keelte semantika põhineb arvutusmudelitel

Kokkuvõtteks (2)

- Põhiküsimuste, nagu: "Kui keeruline on mudeli kirjeldamine?" ja "Mida me saame spetsifitseeritud mudeliga teha?" vastused on sõltuvad valitud arvutusmudelist
- Põhiline kompromiss tuleb teha väljendusvõimuse, formaalsete järelduste ning sünteesivõime vahel
- Põhilised aspektid, millest me olime huvitunud: kattuvus, kommunikatsioon ja sünkronisatsioon, aeg ja hierarhia

Küsimusi?

Gert Jervan
ati.ttu.ee/~gerje