1918 TALLINNA TEHNIKAÜLIKOOL
TALLINN UNIVERSITY OF TECHNOLOGY

Arvutitehnika instituut
ati.ttu.ee

Sardsüsteemid
(Embedded Systems)

II Loeng
Modelleerimine ja spetsifitseerimine

Gert Jervan
Arvutitehnika instituut

www.pld.ttu.ee/~gerje

Graphics: © Alexandra Nolte, Gesine Marwedel, 2003

Some materials: © Petru Eles

Ülevaade

- ✓ Spetsifikatsioonid
  - Nõudmised spetsifikatsioonidele
- ✓ Arvutusmudelid
- ✓ Modelleerimis- ja spetsifitseerimiskeeled
  - StateChart, Petri Net, jne...

Sissejuhatus

- ✓ Sardsüsteemid on tüüpiliselt:
  - Reaalajasüsteemid
  - Reaktiivsed
  - Paralleelsed (*concurrent)*
  - ...
- ✓ Süsteemide arendajad aga ei ole üheski nendest valdkondadest eriti tugevad...
  - Süsteemi kõik omadused on vaja inimesele ja tarkvarale vastuvõetavalt ära kirjeldada

Sissejuhatus

- ✓ Threads!
  - Probleemid:
    - Absoluutselt mittedeterminstlikud
    - Täitmisjärjekord tuleb jõuga paika panna (näiteks mutex-itega)
  - "*... **threads as a concurrency model are a poor match for embedded systems**. ... they work well only ... where best-effort scheduling policies are sufficient."*

Ed Lee: Absolutely Positively on Time, *IEEE Computer*, July, 2005

Von Neuman'i mudel on surnud!

- ✓ ***"The lack of timing in the core abstraction is a flaw, from the perspective of embedded software, ..."***

Ed Lee: Absolutely Positively on Time, *IEEE Computer*, July, 2005

- ✓ ***"Timing is everything"***

Frank Vahid, WESE 2008

- ✓ ***What is needed is nearly a reinvention of computer science.***

Ed Lee: Absolutely Positively on Time, *IEEE Computer*, July, 2005

Spetsifitseerimine

- ✓ Sardsüsteemide jaoks on vaja arvutusmudeleid, mis ei põhineks threadidel ja mis ei põhineks von Neumanni arvutusmudelil
- ✓ Millised on nõuded sardsüsteemide spetsifitseerimistehnikatele?

Nõudmised spetsifitseerimistehnikatele

- Hierarhia
  Inimesed ei suuda aru saada süsteemidest, milles on rohkem kui ca 5 objekti.
  Tegelikud süsteemid nõuavad palju enamat

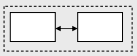
  - Käitumuslik hierarhia
    Näited: olekud, protsessid, protseduurid.

proc
proc
proc

  - Struktuurne hierarhia
    Näited: Protsessorid, räkid, trükkplaadid

Nõudmised spetsifitseerimistehnikatele (2)

- Struktuurne käitumine
  Peaks olema "kerge" alamsüsteemide käitumisest tuletada süsteemi, kui terviku käitumine

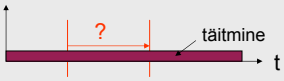


- Ajaline käitumine
  Esmaoluline sidumaks reaalse maailmaga

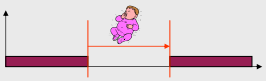
  - Igasugune lisainformatsioon (perioodid, sõltuvused, stsenaariumid) on teretulnud
  - Ka kasutatava platvormi ajaline käitumine (kiirus) peaks olema teada
  - Väga suur mõju disainiprotsessile!

Ajaline käitumine

- Neli nõuet spetsifikatsioonidele:
  1. Ligipääs timerile aja mõõtmiseks

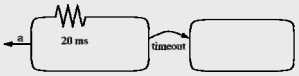


  2. Protsesside viivitamise võimaldamine

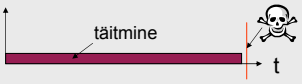


Ajaline käitumine

3. Timeoutide kirjeldamine

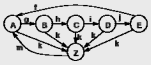


4. Meetodid deadline'ide ja planeeringute (*schedule*) kirjeldamiseks



Nõudmised spetsifitseerimistehnikatele (3)

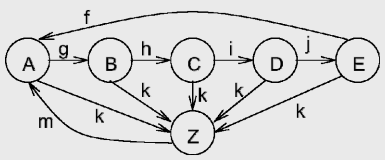
- Olekutele suunatud käitumine
  Vajalik reageerivatele süsteemidele;
  Tavaline automaadimudel ei ole piisav.



- Sündmuste käsitsemine
  (sisemised või välised sündmused)
- Ei ole piiranguid efektiivseks implementeerimiseks

Nõudmised spetsifitseerimistehnikatele (3)

- Tugi usaldusväärsete süsteemide loomiseks
  Üheselt mõistetavad semantikad, ...
- Eranditele suunatud käitumine
  Ei ole mõistlik kirjeldada erandeid iga oleku jaoks



Edaspidi näeme, kuidas kõik servad *k* on võimalik asendada ühe servaga

Nõudmised spetsifitseerimistehnikatele (4)

- Kattuvus (*Concurrency*)
  Reaalaja süsteemid on samaaegsed
- Sünkroniseerimine ja kommunikatsioon
  Komponendid peavad suhtlema!

- Programmeerimiselementide olemasolu
  Näiteks peaks olemas olema: aritmeetilised operatsioonid, tsüklid ja funktsioonide väljakutsed
- Täidetav (ei ole algebralisi spetsifikatsioone)
- Tugi suurte süsteemide kirjeldamiseks (☞ OO)
- Valdkonna-spetsiifilisus

Nõudmised spetsifitseerimistehnikatele (5)

- Loetavus
- Porditavus ja paindlikkus

- Lõpetamine
  Peaks olema selge, milliseks ajahetkeks on kõik arvutused lõppenud

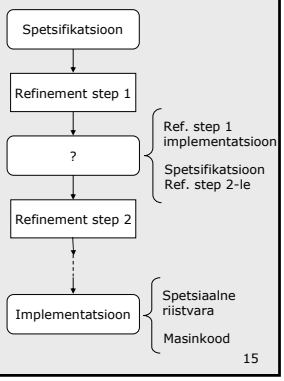
- Tugi mitte-standartsetele I/O seadmetele
  Otsene ligipääs lülititele, displeidele, ...

- Mitte-funktsionaalsed omadused
  veakindlus, kättesaadavus, EMC-omadused, kaal, suurus, kasutajasõbralikkus, laiendatavus, eluiga, võimsustarve, ...

- Sobiv arvutusmudel

Spetsifikatsioonid ja implementatsioonid

- **Spetsifikatsioon:**
  Süsteemi käitumuse ja muude omaduste kirjeldus
  - Projekteerija saab oma tööks (sisendiks) spetsifiaktsiooni ja loob selle põhjal implementatsiooni (toote). Toodet luuakse paljude täiustavate sammude jooksul (refinement steps)



Spetsifikatsioon

- Spetsifikatsioonid võivad olla:
  - Mitteformaalsed (loomulikus keeles)
  - Detailsemad ja ühetähenduslikumad, kasutades *spetsifikatsioonikeeli*
- Spetsifikatsioonikeeled peavad:
  - Olema võimelised hästi väljendama peamisi süsteem omadusi ja erinevaid aspekte sisutihedal ja selgel kujul
  - Sobima hästi nõuete täitmise kontrolliks ja implementatsioonide sünteesiks (soovitavalt automaatselt)
- Alati tuleb valida see keel, mis antud süsteemi jaoks sobiks kõige paremini!

Spetsifikatsioonikeeled

- Spetsifikatsioonikeeled võivad olla
  - Graafilised
  - Tekstilised
- Spetsifikatsioonikeeled võivad olla
  - Tavalised programmeerimiskeeled (C, C++)
  - Riistvara kirjelduskeeled (VHDL, Verilog)
  - Spetsiaalsed keeled, mida kasutatakse erinevates valdkondades süsteemide spetsifitseerimiseks. Tihti põhinevad need mingil *arvutusmudelil* (model of computation)

Süsteemide spetsifitseerimine

- Mida me tahame sardsüsteemi spetsifikatsiooniga peale hakata?

1. Valideerida süsteemi kirjeldust, et kontrollida, kas funktsionaalsus vastab soovitule ja et vajadused on kirjeldatud korrektselt. Selleks kasutatakse:
   - Formaalset verifitseerimist
   - Simuleerimist
2. Et sünteesida efektiivseid rakendusi

Formaalsed mudelid

- Me sooviksime, et spetsifitseerimiskeeled oleks hästi defineeritud semantikaga ➔ spetsifikatsioonid peaksid olema ühetähenduslikud
  - Semantika on reeglite kogu, mis seob tähenduse (interpretatsiooni) süntaktiliste keelekonstruktsioonidega (sümbolite kombinatsiooniga)
  - Semantika põhineb aluseks oleval arvutusmudelil

    Nimetatud mudel määrab ära, milliseid süsteeme saab selle keelega kirjeldada
    Arvutusmudel määrab ära keele väljendusvõime

Formaalsed mudelid (2)

- Kas me sooviksime kasutada suure väljendusvõimega keeli (et saaksime kirjeldada mida iganes)?
  Vahest mitte!
  - 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

1918 TALLINNA TEHNIKAÜLIKOOL TALLINN UNIVERSITY OF TECHNOLOGY

Arvutitehnika instituut ati.ttu.ee

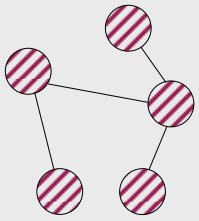
Arvutusmudelid
(Models of Computation)

Arvutusmudelid

- Arvutusmudelid (*models of computation*) käsitlevad keele täitmismudeli (*execution model*) loomiseks vajalikke teoreetiliste valikute kogumeid
  - Disain on esitatud kui komponentide kogum, mida võib vaadelda kui isoleeritud monoliitseid mooduleid (tihti kutsutakse neid protsessideks – *processes* või ülesanneteks – *tasks*), mis suhtlevad omavahel ja ümbruskonnaga
  - Arvutusmudel defineerib nende moodulite käitumise ning omavahelise suhtlemise

Arvutusmudelid (2)

- Arvutusmudelid esitavad:
  - Kuidas iga moodul (protsess või ülesanne) teostab oma sisemisi arvutusi
  - Kuidas moodulid vahetavad omavahel informatsiooni
  - Kuidas nad seostuvad omavahel kattuvuse mõistes
- Mõningad arvutusmudelid ei kajasta moodulite sisemust, vaid üksnes nende suhtlemist ja kattuvust

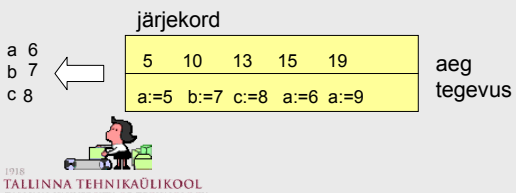


Komponendid

- Von Neumann'i mudel

  Järjestikune täitmine

- Diskreetsed sündmuses



Komponendid (2)

- Automaadid

- Differentsiaalvõrrandid

$$\frac{\partial^2 x}{\partial t^2} = b$$

Kattuvus (Concurrency)

- Süsteemid koosnevad tegevuste (protsessid või ülesanded) kogumist. Neid tegevusi võib potensiaalselt täita paralleelselt, ehk teisisõnu: nad on kattuvad (*concurrent*).
- Kuidas väljendada kattuvust?
  See on üks aspekt, milles arvutusmudelid erinevad
  - Andmete-põhine kattuvus
  - Kontrolli-põhine kattuvus

Andmete-põhine kattuvus

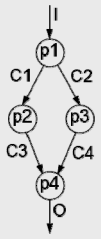
- Süsteem on kirjeldatud kui protsesside kogum ilma määratlemata täitmisjärjekorraga

- Protsesside täitmise järjekord (ja selle põhjal võib kaudselt teha järeldusi parallelismi kohta) on fikseeritud ainult andmete sõltuvuse põhjal
  - Väga tüüpiline paljudes DSP rakendustes

Andmete-põhine kattuvus (2)

```
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);
```

Ei ole oluline, mis järjekorras me need kirjutame



Kontrolli-põhine kattuvus

- Protsesside täitmise järjekord on üheselt kirjeldatud süsteemi spetsifikatsioonis
- Kasutatakse spetsiaalseid konstruktsioone et määrata ära täitmise järjekord ja kattuvus

Kontrolli-põhine kattuvus (2)

```
module p1:
.....
end module

module p2:
.....
end module

module p3:
.....
end module

module p4:
.....
end module

run p1;
[run p2 || run 3];
run p4
```

Siin on kirjutamise järjekord esmaoluline

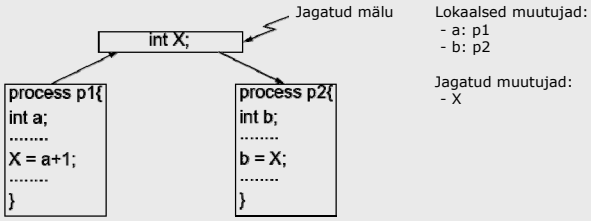
- See näide on kirjutatud ESTERELis
- Protsess p1 algab esimesena ja peab lõppema enne kui p2 ja p3 algavad
- p2 ja p3 algavad paralleelselt
- p2 ja p3 peavad mõlemad lõppema, enne kui p4 saab alustada

Kommunikatsioon

- Protsessid peavad info vahetuseks kommunikeeruma
- Erinevad arvutusmudelid kasutavad erinevaid arvutusmudeleid
- Jagatud mälu (*shared memory*)
- Sõnumite edastamine (*message passing*)
  - Blokeeriv
  - Mitte-blokeeriv

Jagatud mälu

- Iga saatev protsess kirjutab jagatud muutujatesse, mida saavad omakorda vastuvõtvad protsessid lugeda



Jagatud mälu (2)

- Võivd tekkida võidujooksud (race conditions) ☞ tagajärjeks vastuolulised andmed
  ☞ Kriitilised sektsioonid = sektsioonid, kus ressursile (näiteks jagatud mälu) tuleb garanteerida ainuõiguslik ligipääs

```
process a {
  ..
  P(S)  //lukustamine
  ..    //kriitiline sektsioon
  V(S)  //luku vabastamine
}
```

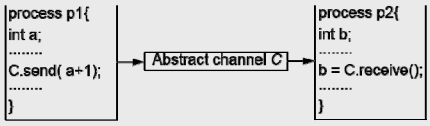
```
process b {
  ..
  P(S)  //lukustamine
  ..    //kriitiline sektsioon
  V(S)  //luku vabastamine
}
```

Ilma võidujooksuta ligipääs jagatud mälule, mida kaitseb lukk S

- Seda mudelit kasutavad:
  - Kriitiliste sektsioonide vastastikune välistamine
  - Cache coherency (jagatud vahemälu) protokollid

Sõnumite edastamine

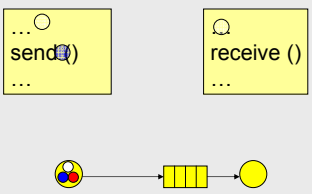
- Andmed edastatakse üle abstraktse kommunikatsioonikeskkonna, mida nimetatakse kanaliks



- See kommunikatsioonimudel on piisav kirjeldamaks hajussüsteeme (*distributed systems*)

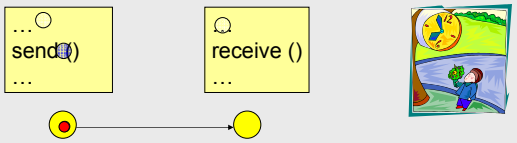
Mitteblokeeriv (asünkroonne)

- Saatja ei pea ootama kuni sõnum on kohale jõudnud; Potensiaalne probleem: Puhvri täitumine



Blokeeriv sõnumite edastamine – rendez-vous

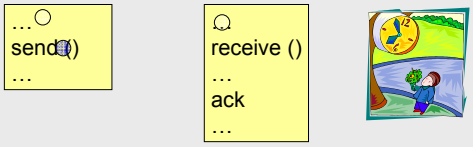
- Saatja ootab kuni vastuvõtja on sõnumi kätte saanud



- Protsess, mis suhtleb, blokeerib end (suspends), kuni teine protsess on valmis andmete ülekandeks
- Need kaks protsess peavad ennast enne andmete ülekannet sünkroniseerima

Laiendatud rendez-vous

- Vastuvõttev pool peab saatma spetsiaalse teate kättesaamise kohta. Vastu võttev pool võib teostada enne teate saatmist andmete kontrolli.



Sünkroniseerimine

- Sünkroniseerimist ei saa eraldada kommunikatsioonist
  - Iga protsesside vaheline suhtlemine eeldab mõningast kommunikatsiooni ja sünkroniseerimist
- Sünkroniseerimine: Üks protsess on seisatud (*suspended*) kuni teise täitmine jõuab mingi punktini
  - Kontrolli-põhine sünkroniseerimine
  - Andmete põhine sünkroniseerimine

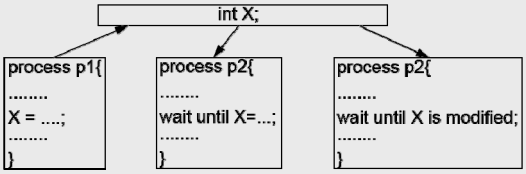
Kontrolli-põhine sünkroniseerimine

```
module p1:
.....
end module

module p2:
.....
end module

module p3:
.....
end module

module p4:
.....
end module

run p1;
[run p2 || run 3];
run p4
```

- Kontrolli-põhise sünkroniseerimise puhul tegeleb sünkroniseerimisega kontrollistruktuur
- Siin on mitmeid sünkroniseerimise punkte:
  - peale p1 lõpetamist ja enne p2, p3 algust
  - Peale p2 ja p3 lõppemist ning enne p4 algust

Andmete põhine sünkroniseerimine

- Kommunikatsioonimehhanismid väljendavad kaudselt ka sünkroniseerimist
- Jagatud mälu põhine sünkroniseerimine



Andmete põhine sünkroniseerimine (2)

- Sõnumite edastamise põhine sünkroniseerimine
  - Kommunikatsiooni blokeerimine sõnumitega tähendab automaatselt saatja ja vastuvõtja vahelist sünkroniseerimist

Protsesside omadused

- **Protsesside arv**
  - Staatiline;
  - Dünaamiline (Dünaamiliselt muutuv riistvaraplatvorm?)
- **Käitumuslik hierarhia**:
  - Protsesside rekursiivne deklareerimine (ADA, VHDL)
    ```
    process {
      process {
        process {
    }}}
    ```
  - või kõik deklareeritud samal tasemel (samm struktuurse hierahia suunas)
    ```
    process { … }
    process { … }
    process { … }
    ```

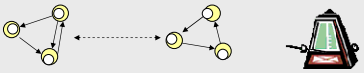
Protsesside omadused (2)

- Erinevad tehnikad protsesside loomiseks
  - Ilmutatud kujul koodis (vt. ADA)
    declare
      process P1 …
  - fork ja join (vt. Unix)
    id = fork();
  - spetsiaalsed protsessi loomise funktsioonid
    id = create_process(P1);

Sünkroonsed v. asünkroonsed keeled

- Mitmete protsesside kirjeldamine on paljudes keeltes mittedeterministlik:
  Ülesannete täitmise järjekord ei ole kindlaks määratud (võib mõjutada tulemust).
- Sünkroonsed keeled: põhinevad automaatide teoorial.
- „Sünkroonsete keelte eesmärgiks on pakkuda kõrgtaseme, modulaarseid konstruktsioone, et selliseid automaate oleks kergem luua" [Halbwachs].
- Sünkroonsed keeled kirjeldavad samaaegselt töötavaid automaate.

Sünkroonsed v. asünkroonsed keeled (2)



- Sünkroonsed keeled eeldavad (globaalset) taktsignaali. Igal taktil arvestatakse kõikide sisenditega, arvutatakse uued olekud ja väljundid ning alles siis tehakse siire.
- See eeldab levitamismehhanisme kõikidesse süsteemi osadesse.
- Ideaalne vaade üheaegsele toimimisele.
- Eeliseks on deterministliku käitumise tagamine.

Tüüpilised arvutusmudelid

- Olekudiagrammid (StateCharts)
- Andmevoo (dataflow) mudelid
- Petri võrgud (Petri Nets)
- Diskreetsed sündmused (Discerete events)
- (Sünkroonsed) lõplikud olekumasinad (Finite State Machines)
- Sünkroonsed/reaktiivsed keeled
- Koosdisaini lõplikud olekumasinad
- Timed Automata

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	

1918 TALLINNA TEHNIKAÜLIKOOL TALLINN UNIVERSITY OF TECHNOLOGY

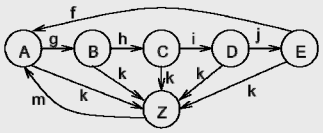
Arvutitehnika instituut
ati.ttu.ee

Olekudiagrammid (StateCharts)

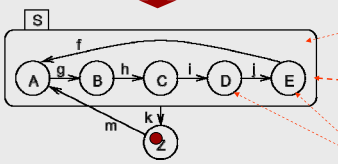
Olekudiagrammid

- Arvutusmudel, mis põhineb jagatud mäluga kommunikatsioonil
- Sobib ainult kohtrakendustele (mitte hajussüsteemidele)
- Klassikaline automaat ei ole sobiv keerukate süsteemide kirjeldamiseks (keerukaid graafe ei ole võimalik inimestel mõista)
- Hierarhia sisse toomine ☞ StateCharts [Harel, 1987]

Hierarhia



FSM on täpselt ühes S'i oleks kui S on aktiivne (kas A või B või ...)

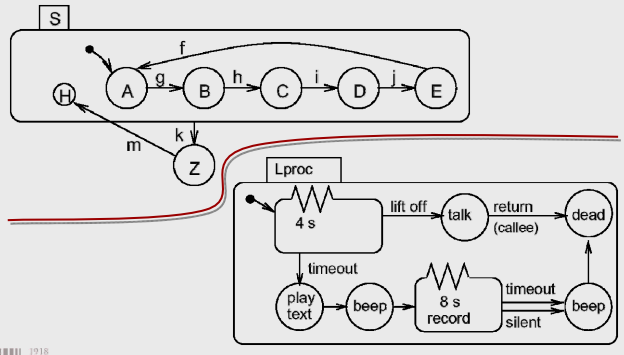


Superstate

Oleku E eellane (ancestor)

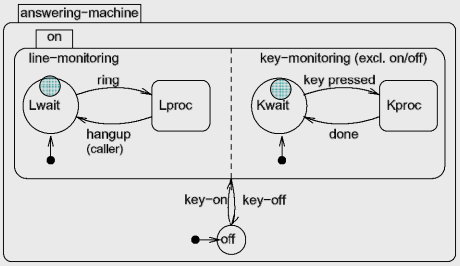
Alamolekud (substates)

Ajalugu, vaikimisi olek, timerid



Kattuvus

- AND-superstate



Hinnang StateChart'ile

- Pros:
  - Hierarhia lubab suvalist komplekti AND- ja OR-superstate'e.
  - Mitmed kommertstarkvarapaketid (StateMate, StateFlow, BetterState, ...)
  - On olemas „back-end" tarkvara StateChart'ide transleerimiseks C-sse või VHDLi, võimaldades sedasi tarkvaralisi ja riistvaralisi lahendusi.

Hinnang StateChart'ile (2)

- Cons:
  - Genereeritud C programmid ei ole alati efektiivsed
  - Ei sobi hajusrakendustele
  - Ei ole programmilisi konstruktsioone
  - Ei võimalda kirjeldada mitte-funktsionaalset käitumist
  - Ei ole objekt orienteeritud
  - Ei võimalda haarata struktuurset hierarhiat

Laiendused:
- Module charts struktuurse hierarhia kirjelduseks

1918 TALLINNA TEHNIKAÜLIKOOL TALLINN UNIVERSITY OF TECHNOLOGY

Arvutitehnika instituut ati.ttu.ee

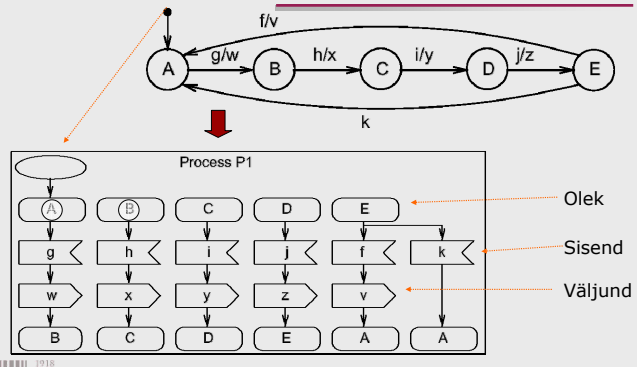
SDL

SDL

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

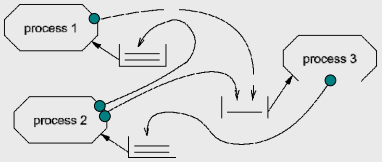
Kommunikatsioon/ arvutused	**Jagatud mälu**	**Sõnumite edastamine**	
		Blokeeriv	**Mitteblokeeriv**
FSM	StateCharts		SDL

FSMide/protsesside kujutamine SDL'iga



SDLi FSMide vaheline kommunikatsioon

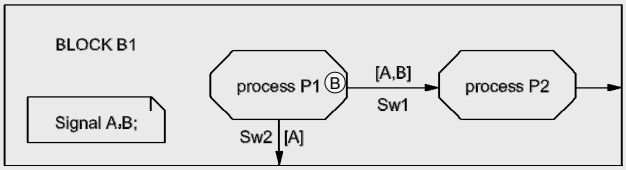
- ✓ FSMide (või „protsesside") vaheline kommunikatsioon põhineb sõnumite edastamisel, eeldades lõpmatu suurusega FIFO puhvreid



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

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

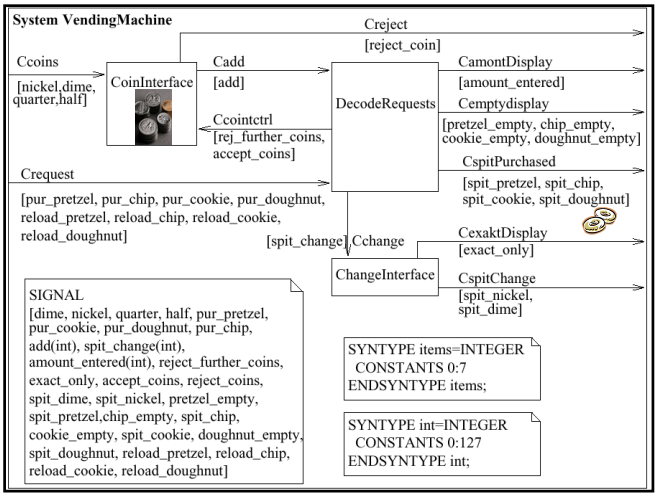
Süsteemi näide: toiduautomaat

Masin, mis müüb erinevaid maiustusi

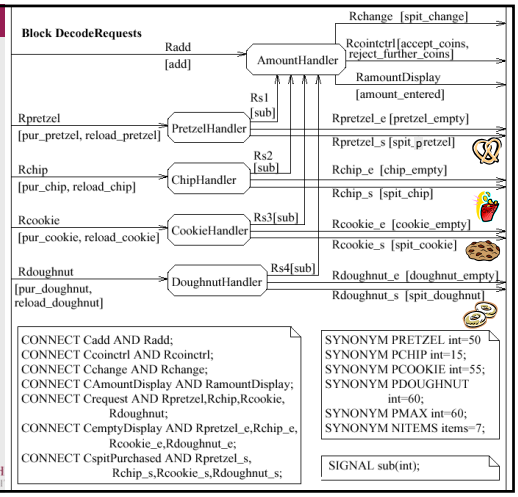
Aksepteerib erinevaid münte

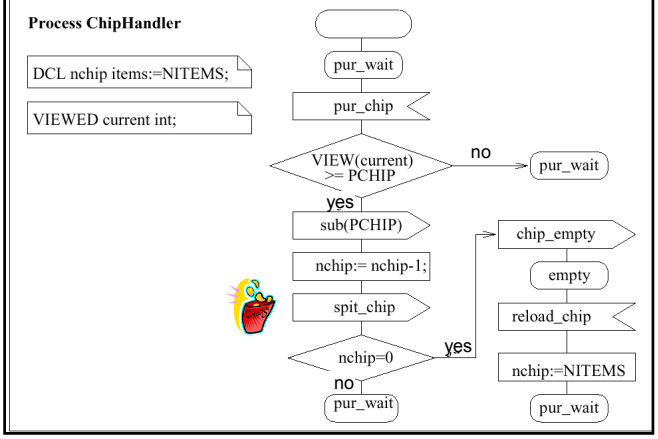
Ei ole hajusrakendus

[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	

TALLINNA TEHNIKAÜLIKOOL
TALLINN UNIVERSITY OF TECHNOLOGY

Arvutitehnika instituut
ati.ttu.ee

Andmevoo mudelid
(Dataflow models)

Andmevoo mudelid

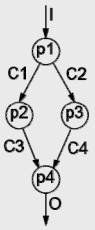
- Süsteemid on kirjeldatud, kui suunatud graafid, kus:
  - Sõlmed esitavad arvutusi (protsesse)
  - Kaared esitavad täielikult järjestatud andmevoogu
- Sõltuvalt semantikast on andmevoo põhjal defineeritud mitmeid erinevaid arvutusmudeleid:
  - Kahni protsessivõrgud (Kahn Process Networks)
  - Andmevoo 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*)

Kahni protsessivõrgud

- 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!**

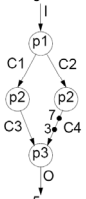
Kahni protsessivõrgud (2)

- 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

KPN

```
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);
```

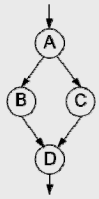
7 3 0 16 8 5 11 21 8 5



5 0 11 16 21 8 5 8

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



1918 TALLINNA TEHNIKAÜLIKOOL TALLINN UNIVERSITY OF TECHNOLOGY

Arvutitehnika instituut ati.ttu.ee

Petri võrgud
(Petri Nets)

Petri võrgud

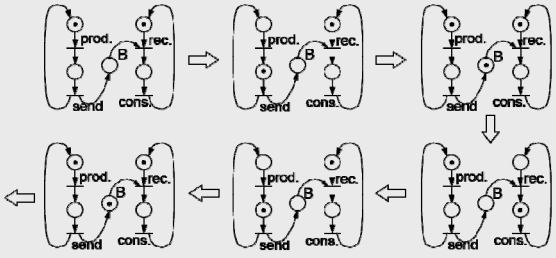
- ✓ Süsteem on spetsifitseeritud kui suunatud kahealuseline graaf, kus on kahte tüüpi sõlmi:
  - Koha sõlmed (places): Hoiavad hajutatud süsteemi olekut, mida väljendatakse märgi (token) olemasolu või puudumisega antud sõlmes
  - Üleminekud (transitions): Kasutatakse süsteemi toimimise tähistamiseks
- ✓ 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

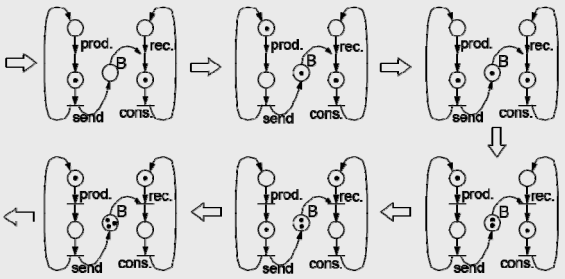
Petri võrgud näide

- ✓ Genereeriv ja vastuvõttev protsess suhtlevad läbi puhvri



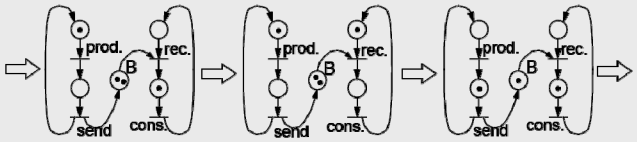
Petri võrgud näide (2)

- Näite jätk...



Petri võrgud näide (3)

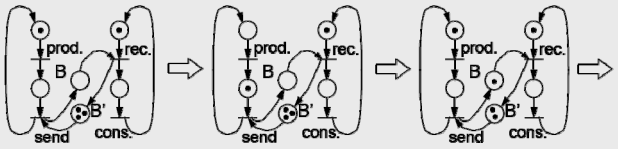
- Näite jätk...



- 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

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

1918 TALLINNA TEHNIKAÜLIKOOL TALLINN UNIVERSITY OF TECHNOLOGY

Arvutitehnika instituut
ati.ttu.ee

Discrete Event Models

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
- ✓ Näiteks: VHDL, Verilog

1918 TALLINNA TEHNIKAÜLIKOOL TALLINN UNIVERSITY OF TECHNOLOGY

Arvutitehnika instituut
ati.ttu.ee

Imperatiivsed keeled

C, SystemC, Java, ...

© Gert Jervan – TTÜ/ATI Arvutid II – Sardsüsteemid – Loeng 2

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	

© Gert Jervan – TTÜ/ATI Arvutid II – Sardsüsteemid – Loeng 2

C kasutamine sardsüsteemide loomisel

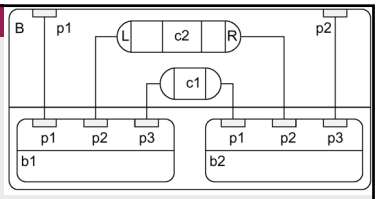
- ✓ Motivatsioon
  - Paljud standardid (näiteks GSM, MPEG) on publitseeritud C programmidena
    - Riistvara kirjelduskeele (näiteks VHDL) kasutamiseks peaks standardeid hakkama "tõlkima"
  - Paljude süsteemide funktsionaalsus eeldab nii riistvara kui ka tarkvara
    - Simuleerimine nõuaks vastavaid 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

© Gert Jervan – TTÜ/ATI Arvutid II – Sardsüsteemid – Loeng 2

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

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();}}
   };
```


* 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

Channels for MoCs Kahn process networks, SDF, etc	Methodology-specific Channels Master/Slave library
Elementary Channels Signal, Timer, Mutex, Semaphore, FIFO, etc	
Core Language Module Ports Processes Events Interfaces Channels **Event-driven simulation kernel**	Data types Bits and bit-vectors Arbitrary precision integers Fixed-point numbers 4-valued logic types, logic-vectors C++ user defined types
C++ Language Standard	

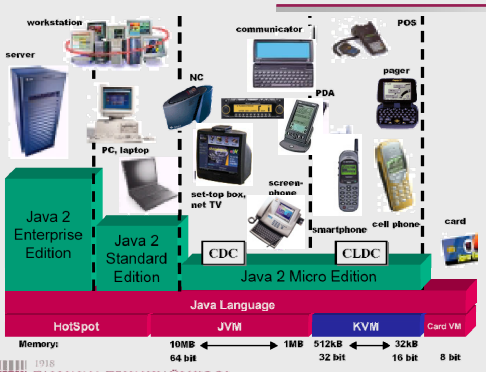
Java

- ✓ 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

Java 2



http://java.sun.com/products/cldc/wp/KVMwp.pdf

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 $\rightarrow$ 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

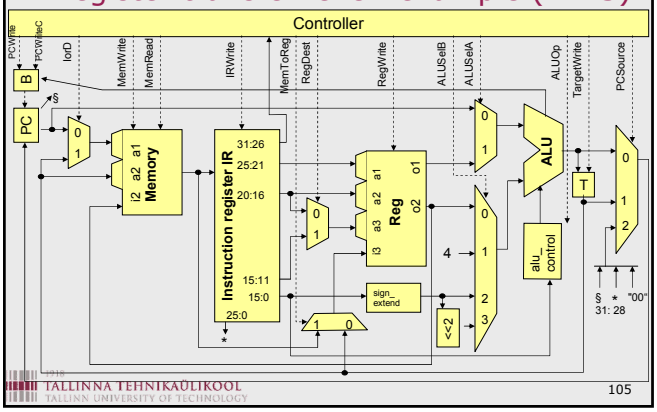
Instruction level: example

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.

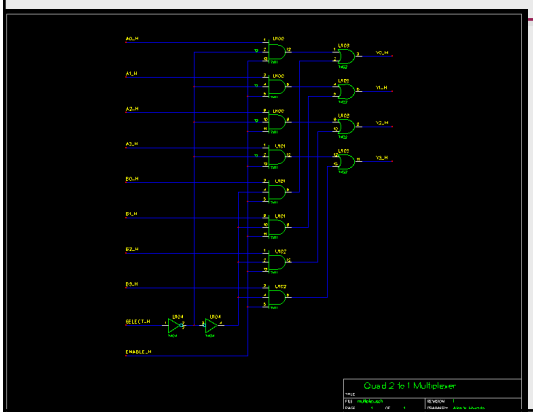
Register transfer level: example (MIPS)



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

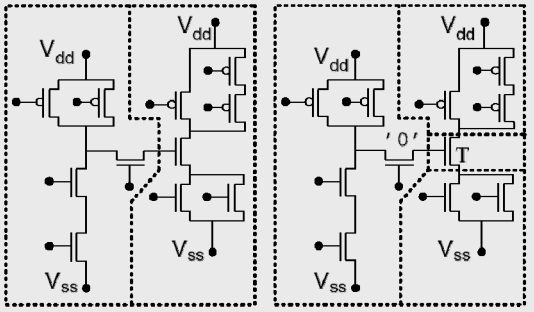


source: http://geda.seul.org/screenshots/screenshot-schem2.png

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

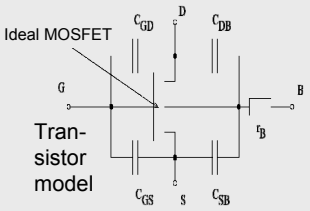


Source: http://vada1.skku.ac.kr/ClassInfo/ic/vlsicad/chap-10.pdf

Circuit level models: Example

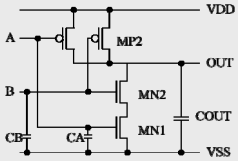
- Models circuit theory. Its components (current and voltage sources, resistors, capacitances, inductances and possibly macro-models of semiconductors) form the basis of simulations at this level.

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



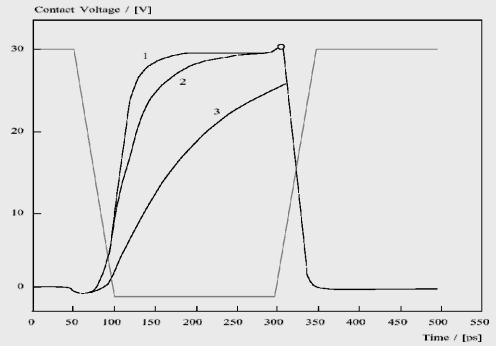
Circuit level models: SPICE

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



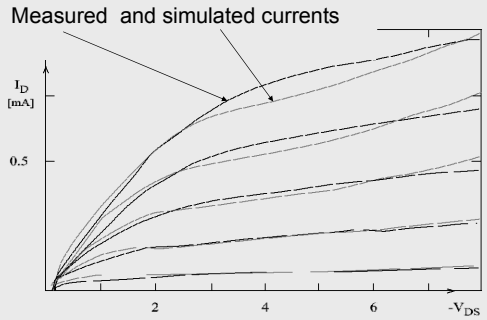

```
.SUBCKT NAND2 VDD VSS A B OUT
MN1 I1 A VSS VSS NFET W=8U L=4U AD=64P AS=64P
MN2 OUT B I1 VSS NFET W=8U L=4U AD=64P AS=64P
MP1 OUT A VDD VDD PFET W=16U L=4U AD=128P AS=128P
MP2 OUT B VDD VDD PFET W=16U L=4U AD=128P AS=128P
CA A VSS 50fF
CB B VSS 50fF
COUT OUT VSS 100fF
.ENDS
```


Circuit level models: sample simulation results



Device level

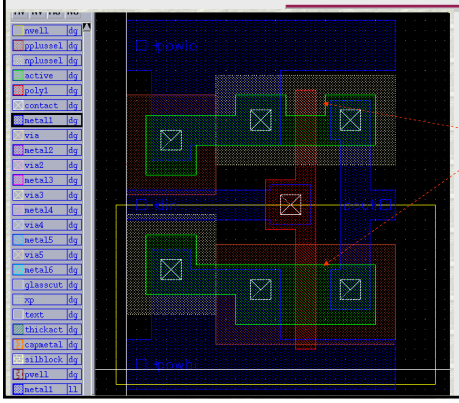
- Simulation of a single device (such as a transistor).
- Example (SPICE-simulation [IMEC]):



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).

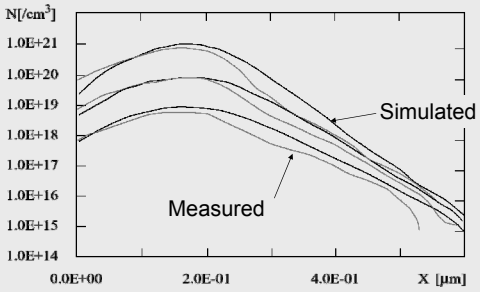
Layout models: Example



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

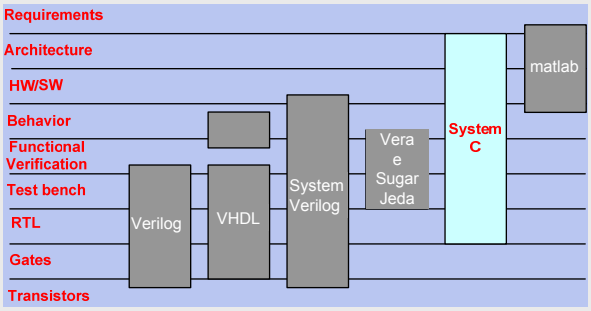
Process models

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



Movie (German)

Keeled ja abstraktsioonitasemed

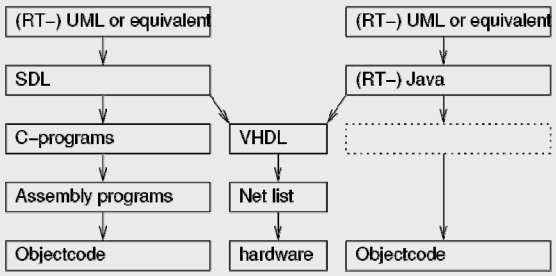


Keelde võrdlus

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?

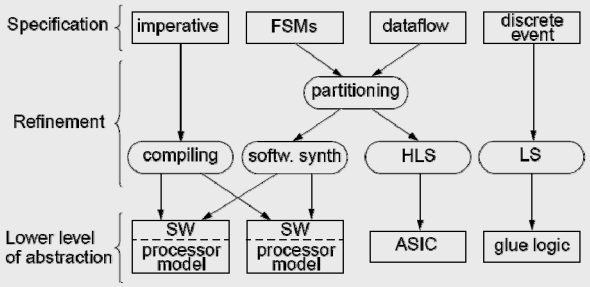
- Kõik sõltub loodavast süsteemist
  - Kas domineerib andmevoog (nagu näiteks DSPdes) või on tegemist kontrollile orienteeritud süsteemidga (reaktiivsed süsteemid)?
  - Sünkroonne või asünkroonne? Tsentraliseeritud või hajutatud?
  - Kui suur?
  - Milline on suhe aega?
  - ...
- Mida sa tahad mudeliga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

Milline modelleerimissuund valida? (2)

- Ära kasuta seda, mis võimaldab teha "kõike"!
- Eelmisest 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

Järgnevad loengud

- ✓ Sardsüsteemide riistvara
  - Reaalaja komponent
- ✓ Võimsus- ja energiatarbe optimeerimine
- ✓ Test ja verifitseerimine

1918 TALLINNA TEHNIKAÜLIKOOL TALLINN UNIVERSITY OF TECHNOLOGY

Arvutitehnika instituut
ati.ttu.ee

Küsimusi?

Gert Jervan
www.pld.ttu.ee/~gerje

Tallinna Tehnikaülikool
Arvutitehnika instituut