# Design og implementering af hardware-specifikationssprog

*Forende til simulationssystemet SimSys*

**Signe Reuss**

Datalogisk Institut, Københavns Universitet
4. marts 2002

**Resumé**

I dette speciale designes og implementeres hardware-specifikationssproget Hades med henblik på brug på kurset Dat1E på 2. år på datalogi på Københavns Universitet. Formålet med sproget er at give de studerende mulighed for at eksperimentere med design af mikroarkitekturer på komponentniveau (modsat transistorniveau) for derved at lære dem om opbygningen af en datamat.

Der laves et design for Hades, hvorefter det så vidt muligt dokumenteres, at dette design er brugbart på Dat1E. Herefter implementeres næsten hele sproget (det er desværre ikke muligt at nå at implementere det hele indenfor den givne tidsramme), og der laves en afprøvning for den implementerede delmængde. De udeladte dele af sproget mindsker ikke funktionaliteten, men gør blot nogle få konstruktioner en smule mindre elegante.

Implementeringen laves som en forende til simulationssystemet SimSys, der gør det muligt at lave køretidsstatistikker for simulationer, at køre dem samt at interagere med dem under kørsel.

Da Hades skal bruges på Dat1E, er der i specialet lagt vægt på, at det er tydeligt, hvad der mangler at blive implementeret, samt hvilke fejl der optræder, så det er nemt at færdiggøre oversætteren.

# Indhold

# Kapitel 1

# Forord

I vinteren/foråret år 2000 blev det tilbagevendende problem på kurset Dat1E bragt op i Datalogisk fagråd på Diku: Problemet med brugen af C++. I Dat1E bruges simulationssystemet SimSys til at implementere modeller til simulering af elektriske kredsløb. I SimSys betragtes digitallogiske komponenter som en slags byggeklodser, der kan sættes sammen til et kredsløb. SimSys er lavet som et klassebibliotek til C++, men brugen af systemet kræver ikke noget særligt kendskab til dette sprog, da de studerende stort set kan skrive af efter kursusbogen [And01]. Problemerne opstår først i det øjeblik, der rejses fejl af C++-oversætteren; at forstå disse fejl kræver enten et godt kendskab til C++ eller en relativ stor programmeringserfaring. Ingen af delene kan forventes af andenårsstuderende. Der er nogen undervisning i C++ på Dat1E, men der er ikke tilstrækkelig tid til at lære de studerende nok C++ til at kunne håndtere de problemer, der faktisk opstår i forbindelse med modelleringen af kredsløb i SimSys.

Dette problem er dukket op i fagrådet hvert år siden 1996, hvor SimSys og C++ for første gang blev brugt på Dat1E, og denne gang var vi fast besluttede på, at nu skal det løses. I den forbindelse diskuterede jeg problemet med den kursusansvarlige for Dat1E, Torben Mogensen, samt med en tidligere instruktor på kurset, Morten Fjord-Larsen. Det var Morten, der foreslog, at lave en forende (*eng: frontend*) til SimSys, så brugen af C++ kan undgås, mens Torben foreslog, at jeg lavede et speciale om dette.

Før det blev besluttet at lave en forende til SimSys, undersøgte Torben og jeg dog mulighederne for at bruge et allerede eksisterende maskinarkitektursprog (*eng: hardware description language*), da det ville løse problemet med brug af C++. Vi fandt, at hverken de relativt enkle relationelle og funktionelle sprog som Ruby [SR95], Hawk [Haw] og Lava [Lav] eller det noget ældre og også mere komplekse sprog Verilog [Ver] er brugbart. Førstnævnte kan ikke bruges, da de ikke indeholder mulighed for at lave køretidsstatistikker og ressourcemodellering for modellen af kredsløbet, og dette betragtes som en væsentlig del af opgaverne på Dat1E. Dette er muligt i Verilog, men dette sprog er til gengæld alt for maskinnært til, at det kan bruges på Dat1E. Derfor fandt vi, at den bedste løsning var at lave en forende til SimSys, så de studerende undgår brugen af C++. Samtidig kan simuleringsmotoren og køretidssystemet fra SimSys genbruges, hvilket vil gøre implementeringen noget nemmere og samtidig bevare muligheden for ressourceestimering.

Jeg har døbt mit sprog Hades (**ha***rdware* **de***sign* **s***prog*) og vil bruge dette navn i resten af rapporten.

## 1.1 Notation

I rapporten skrives eksempler i *kursiv*, hvis de ikke indgår som en integreret del af teksten.

## 1.2 Læserens forudsætninger

Det forventes, at læseren har et rudimentært kendskab til digitallogik svarende til en kort indføring i området. Det er også nødvendigt med et overfladisk kendskab til SimSys og med et basalt kendskab til oversætterteknik.

## 1.3 Tak til

Først og fremmest skylder jeg en stor tak til min mor, Anna Reuss, der i den sidste måneds tid op til afleveringen af dette speciale adskillige gange har lavet mad **og** bragt det ud til mig.

Finn Schiermer Andersen (ophavsmand til SimSys), Torben Mogensen og Martin Zachariasen (undervisere på Dat1E) har deltaget i adskillige diskussioner af mine forslag til design og har dermed været en stor hjælp i processen med at udarbejde et design, som er brugbart.

Også tak til Inge Li Gørtz, der har læst korrektur på specialet, og en ganske særlig tak til Søren Debois, der i meget høj grad har indrettet sit liv efter min specialeskrivning, og som desuden har ydet en ekstraordinær indsats ved korrektur af rapporten.

Sidst men ikke mindst tak til Morten Fjord-Larsen for at forslaget om at løse problemet på Dat1E ved at lave en forende til SimSys, da det muliggjorde dette speciale.

Og så selvfølgelig en tak til alle jer, der har hjulpet og støttet mig i forbindelse med specialeskrivningen. I ved, hvem I er ;).

# Kapitel 2

# Indledning

Nærværende speciale dokumenterer designet og, til en vis grad, implementeringen af en oversætter for hardware-specifikationssproget Hades.

Hades er et deklarativt sprog til specificering af digitallogiske kredsløb. Sprogets fokus er didaktisk; dets primære mål er at tillade datalogistuderende (på andet år) at eksperimentere med design af mikroarkitekturer på komponentniveau modsat transistorniveau. Hadesprogrammer lader sig således ikke umiddelbart omsætte til silicium.

Den implementerede oversætter tager Hadesprogrammer som inddata, og producerer SimSys-programmer som uddata. En almindelig C++-oversætter med SimSysklassebiblioteket inkluderet kan heraf producere en eksekverbar simulator. Hadesoversætteren kan håndtere hele Hades modulo nogle få undtagelser (se afsnit 9.3) samt nogle få kendte fejl (se afsnit 10.7), som det desværre ikke har været muligt at implementere og rette indenfor den givne tidshorisont. Implementeringen er dog opbygget, så de manglende konstruktioner nemt kan tilføjes.

## 2.1 Formål

Formålet med dette speciale er at designe og implementere et sprog til simulering af modeller af elektriske kredsløb til brug på kurset Dat1E. Det skal verificeres, at sproget er velegnet til brug på kurset, og den implementerede oversætter skal være brugbar og skal afprøves. Ved *brugbar* forstås, at funktionaliteten fuldt ud dækker behovene på Dat1E. Det tillades dog, at enkelte byggeklodser ikke er medtaget, da disse nemt kan inkluderes. Mangler og fejl skal fremgå klart af denne rapport, så oversætteren nemt kan færdiggøres.

## 2.2 Fremgangsmåde

For at opnå et på Dat1E anvendeligt sprog, skal de nuværende problemer og uhensigtsmæssigheder undgås. Disse afdækkes ved afprøvning af SimSys samt ved samtaler med undervisere, instruktorer og studerende på kurset samt med Finn Schiermer Andersen, der har lavet og vedligeholdt SimSys. Underviserne på Dat1E og Finn Schiermer Andersen har løbende givet feedback på mine forslag til design. Dette har været nødvendigt, da de har et indgående kendskab til problemstillingerne, niveauet og de studerende på Dat1E.

Implementeringen laves for en del af sproget ad gangen, og der satses på en høj grad af modularitet, så programmet er nemt at udvide og vedligeholde.

Det er muligt at udvide funktionaliteten af Hades, så en del af programmet kan skrives i SimSys, hvorefter der genereres passende SimSyskode ud fra det fulde program (inklusive Hadeskoden). Således vil også de elementer af SimSys, der er udeladt af Hades, kunne bruges af de studerende. Dette anses dog ikke som væsentligt i den første udgave af Hades.

## 2.3 Rapportens opbygning

- I kapitel 3 opsummeres SimSys' funktionalitet, og ud fra denne fastlægges den grundlæggende funktionalitet af Hades. Desuden diskuteres forslag til udvidelser af den nuværende funktionalitet.
- I kapitel 4 afdækkes problemer og uhensigtsmæssigheder ved brugen af SimSys og C++, og ud fra disse samt ud fra den ønskede funktionalitet sættes mål for, hvornår Hades er en succes.
- Både funktionaliteten samt ovennævnte mål tages i betragtning ved udformningen af Hades i kapitel 5. Her analyseres og diskuteres den generelle udformning af sproget, og sproget skitseres.
- Den fuldstændige grammatik og semantik for Hades gives i kapitel 6.
- Herefter gives et eksempel på et Hadesprogram i kapitel 7 for at illustrere sproget. Et faktorialkredsløb opbygges, og til brug i dette konstrueres også en hurtig multiplikator.
- I kapitel 8 dokumenteres det så vidt muligt, at designet for Hades opfylder målene for sprogets design opstillet i kapitel 4.
- I kapitel 9 beskrives implementeringen, og i afsnit 9.3 gennemgås, hvad der mangler at blive implementeret. Selve kildekoden kan ses i bilag E.
- I kapitel 10 dokumenteres en afprøvning af oversætteren, og i afsnit 10.7 opsummeres, hvilke fejl der er fundet.
- I konklusionen opridses resultaterne i denne rapport, og det beskrives kort, hvordan der bør følges op på dette arbejde.

# Kapitel 3

# Funktionalitet af Hades

I dette kapitel fastlægges funktionaliteten af Hades. Afsnit 3.1 giver en kort gennemgang af SimSys' funktionalitet, og med udgangspunkt heri fastlægges den grundlæggende funktionalitet af Hades i afsnit 3.2. Til sidst et kort afsnit 3.3 om udvidelser af den nuværende funktionalitet.

## 3.1 SimSys

SimSys består grundlæggende af fire elementer: byggeklodserne (*eng: building blocks*), faciliteter til at lime klodserne sammen, en simuleringsmotor og et køretidssystem. I dette afsnit gennemgås funktionaliteten af disse samt de dele af C++, der især bruges ved programmering i SimSys. Afsnittet er baseret på [And01].

**Byggeklodserne** i SimSys er klasser, der repræsenterer digitallogiske komponenter som f.eks. addere, multiplexere og flipflopper. En byggeklods består af en grænseflade og en implementering, hvor grænsefladen eksporterer nogle *stik* (*eng: channels*). Stik er en abstraktion af ind- og uddata til en komponent. Data til et stik skal have en bestemt bit-bredde. Byggeklodser forbindes ved at sammensætte hvert udstik fra en byggeklods med et indstik fra en anden.

*Byggeklodsen* `adder` *har fem stik:* `inA` *og* `inB` *(eng: addends),* `cIn` *(eng: carry in),* `sum` *og* `cOut` *(eng: carry out), hvor de tre første bruges til at føre bit ind i* `adder`*, og de to sidst modtager uddata fra klodsen.*

SimSys indeholder et bibliotek med prædefinerede byggeklodser (bl.a. `FlipFlop`, `Adder` og `Mux`), hvor kun grænsefladen er synlig for brugeren. Det er muligt at konstruere sine egne byggeklodser enten ved at sætte allerede eksisterende byggeklodser sammen til en såkaldt *sammensat byggeklods* (*eng: composite building block*) eller ved at konstruere en såkaldt *atomisk byggeklods* fra grunden (*eng: atomic building block*). I sidstnævnte tilfælde skal brugeren have en god model for den underliggende CMOS-implementering for at kunne lave en passende specifikation for klodsen (længste signalvej osv.).

Når en byggeklods eller komponent bruges, skal størrelsen samt eventuelle værdier for begyndelsestilstanden angives. Således bør der strengt taget skelnes mellem en byggeklods (f.eks. adder) og en instans af en byggeklods (f.eks. en 32-bit adder). I resten af rapporten bruges ordene byggeklods og komponent også om instanser af byggeklodser og komponenter, hvor det er klart af sammenhængen, hvad der er tale om.

Har man behov for mange byggeklodser af samme slags anvendt i serie, kan man arrangere disse i en vektor (*eng: array*). En vektor gør det nemmere at håndtere komponenterne samtidig i både instantiering og anvendelse. Specielt kan instantieringen af den enkelte byggeklods afhænge af dens

placering i vektoren. I en vektorbyggeklods samles alle stik af samme type til en såkaldt *stikvektor* (*eng: channel array*), der kan forbindes med en anden stikvektor. Nogle ikke-vektorbyggeklodser har også stikvektorer for dele af ind- eller uddata.

*Den indbyggede komponent* `mux` *har et indstik* `select` *og en indstikvektor* `in` *samt et udstik* `out`. `in` *består af et antal stik af samme størrelse, hvor* `select` *angiver, fra hvilket af disse uddata vælges.*

*Konstanter* (*eng: drivers*) bruges til at lede et konstant uddata, mens *afløb* (*eng: drains*) bruges til at bortlede uddata fra stik, der ellers ville lades uforbundne.

Brugeren kan bygge en model af et kredsløb ved parvis at **forbinde** stik eller stikvektorer for komponenter. Stik kan forbindes i deres helhed, men der findes også funktioner til at omdanne et stik ved at kopiere, udelade eller blot permutere bit i stikket, hvorved der fremkommer en såkaldt afledning (*eng: derivation*). En udstikvektor kan forbindes med en indstikvektor, hvis hvert udstik har samme størrelse som det tilsvarende indstik. En stikvektor kan transponeres (svarer til transponering af en matrix i matematik), før den forbindes med en anden. Endelig er det muligt at konvertere et stik til en stikvektor og vice versa.

Når byggeklodser sættes sammen, er det desuden muligt at opnå bestemte sideeffekter, når der løber data igennem en forbindelse af to stik. Disse faciliteter kaldes *skel* (*eng: boundaries*) og bruges, når der skal simuleres pipelinede mikroarkitekturer.

Givet en model af et kredsløb kontrollerer **simuleringsmotoren** aktiviteten af byggeklodserne og transporten af uddata fra et stik til et afhængigt indstik. Et indstik er *afhængigt* af et udstik, hvis stikkene er forbundet, da data fra udstikket skal flyde ind i indstikket bagefter. Ud fra afhængighederne mellem byggeklodserne tilrettelægges aktiviteten af de enkelte klodser samt transport af data mellem stik.

**Køretidssystemet** gør det muligt for brugeren at interagere med simulationen. Det er muligt at få udskrevet ind- og uddata fra stik under kørsel eller at undersøge tilstandene for tilstandselementer (*eng: building blocks with state*, f.eks. flipflopper). Der kan indsættes stoppunkter (*eng: data breakpoints*), så simuleringen standser, hvis betingelsen er opfyldt. Simulationen kan også blive spolet tilbage og kørt herfra eller blive kørt et skridt ad gangen.

**C++** stiller en del faciliteter til rådighed, men af disse bruges hovedsageligt mulighederne for at lave løkker eller betingede sætninger. Betingede sætninger bruges til at lave forskellige implementeringer af (dele af) kredsløb. Løkker bruges hovedsageligt til opbygning af strukturer som f.eks. træer, eller til at forbinde stikvektorer eller enkelte bit i et stik på en struktureret måde.

*Lad* `andarray` *være en vektor med* `antal` *2-bit andporte og* `mux` *en mux, hvor udstikket* `out` *er* `antal+1` *bit bred. Hvis den $j$'te andport i* `andarray` *skal modtage bit $j$ og $j+1$ fra* `out`*, for alle j, kan dette gøres ved*

```
for (int j=0; j<antal; j++) {
  mux.out.subset(j,2) >> andarray.in[j];
};
```

*hvor* `andarray.in[j]` *angiver $j$'te stik i indstikvektoren* `in` *for* `andarray`.

## 3.2 Grundlæggende funktionalitet af Hades

Den grundlæggende funktionalitet af Hades fastlægges i den efterfølgende diskussion, der tager sit udgangspunkt i funktionaliteten af SimSys. Systemet er blevet brugt på Dat1E siden 1996 og dækker

de funktionelle behov på kurset. Systemet er dog blevet udviklet og udbygget undervejs, så nogle dele nu kun bruges af få eller ingen studerende.

Hvilke dele af SimSys' funktionalitet, der medtages i Hades , vurderes ud fra følgende tre kriterier: nødvendighed, om de letter programmeringen, eller om de fremmer forståelsen af maskinens opbygning, hvor førstnævnte er en tilstrækkelig betingelse for, at en del medtages, mens de to sidstnævnte kriterier bruges, hvis dette ikke er opfyldt. De enkelte dele af SimSys vurderes ud fra programmering i sproget samt diskussioner med underviserne på Dat1E (Torben Mogensen og Martin Zachariasen) og med ophavsmanden til SimSys (Finn Schiermer Andersen, der også har opdateret og vedligeholdt strukturen), da de har et indgående kendskab til både opgaverne og de studerende på kurset.

Diskussionen i dette kapitel starter med byggeklodserne og forbindelser mellem disse og fortsætter med betingede sætninger og løkker (fra C++, men fokuseret på brugen i forbindelse med SimSys). Til sidst betragtes simuleringssmotoren og køretidssystemet, der ikke er en del af den sproglige grænseflade, men som skal virke i samspil med Hades, da formålet med at implementere Hades som en forende til SimSys er at kunne genbruge disse.

For indeværende indeholder byggeklodsbiblioteket i SimSys alle komponenter, der er nødvendige for at kunne løse opgaverne på Dat1E. Derfor bruger kun få eller ingen studerende på en årgang muligheden for at lave atomiske byggeklodser. Det er selvfølgelig lærerigt at konstruere sine egne komponenter (eventuelt ud fra specifikation af komponenter fra den virkelige verden), men dette bidrager mere til forståelsen af funktionaliteten af det indre af en komponent, end til forståelsen af maskinen og dennes opbygning (samspillet mellem komponenterne). Det går altså et niveau dybere, end formålet med kurset berettiger og udelades derfor af Hades. Til gengæld skal den fulde funktionalitet af byggeklodsbiblioteket dækkes, ligesom mulighederne for at lave konstanter eller lave vektorer af byggeklodser skal bevares.

Ved sammensætning af byggeklodserne i SimSys bruges alle de eksisterende funktioner til at lave afledninger, og denne funktionalitet skal altså også dækkes. Det samme gælder skel, da der i K1-opgaven laves en pipelinet version af en simulator, hvor disse bruges. Mulighederne for at lave forskellige implementeringer af samme grænseflade og at bygge strukturer som f.eks. træer er også vigtige at bevare.

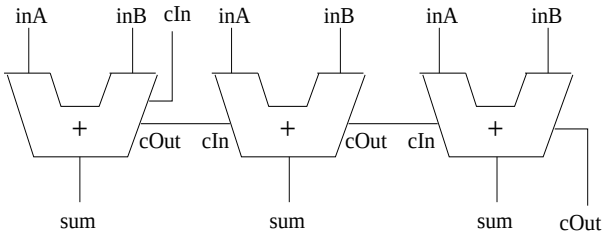
Simuleringsmotoren spiller ikke ind på udseendet af Hades, og da Hadesprogrammer oversættes til SimSysprogrammer, kan den bruges også med den nye forende. Når brugeren skal håndtere instanser af komponenter i køretidssystemet (f.eks. udskrive uddata fra bestemte stik), gøres dette via navnene på instanserne. Det er derfor vigtigt, at konstruktioner i Hades kan relateres direkte til de forskellige komponenter, køretidssystemet opererer med.

Med ovenstående funktionalitet bør alle opgaver i Dat1E kunne løses i Hades. De studerende, der måtte ønske at bruge faciliteter, der ikke er understøttet af Hades, har stadig mulighed for at lave deres programmer i SimSys.

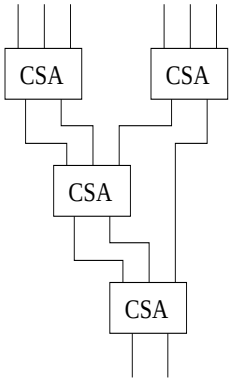
## 3.3 Udvidelser af funktionaliteten

Finn Schiermer Andersen er kommet med nedenstående to forslag til udvidelser.

Det første forslag er, at det skal være muligt at lave vektorer af komponenter, hvor disse er forbundet internt, dvs. hvor udstik for en blok i vektoren forbindes til indstik for næste blok i vektoren, som f.eks. i




Det andet forslag er, at det skal være nemt at lave træer og træniveauer (*eng: tree levels*). Et træniveau er en byggeklodsvektor, som indgår i et træ, og hvor antal bit i uddata er mindre end antal bit i inddata. Træniveauer kan sættes sammen til et træ ved, at uddata fra et niveau ledes ind i inddata for det næste som i f.eks.




der illustrerer et csatræ med kun tre niveauer. Bemærk, at hvis mængden af uddata fra et niveau ikke passer til inddata for næste, forbindes de resterende stik med niveauet efter. Herved adskiller træer indenfor digitallogik sig fra træer indenfor andre grene af datalogien.

*En multiplikation foretages ved at lægge addenderne for multiplikator og multiplikand sammen (se kapitel 7). Dette kan gøres med en vektor af addere, men tidsforbruget kan reduceres ved i stedet at bruge et træ. Eksempelvis giver brugen af et csatræ meget hurtig multiplikation.*

Af disse to udvidelser vil særligt den første være anvendelig, men de vil begge lette programmeringen og vil måske ydermere gøre det muligt at stille lidt mere avancerede opgaver på Dat1E. Derfor indarbejdes begge muligheder i Hades.

# Kapitel 4

# Mål

Da Hades skal bruges af omtrent 200 studerende på Dat1E, er det vigtigt, at risikoen for problemer ved brugen er minimal. Til dette formål opstilles i dette kapitel nogle mål for sprogets design. Disse skal sikre, at det opfylder behovene på kurset. Inden implementering af oversætteren dokumenteres det så vidt muligt, at målene er opfyldt, så eventuelle justeringer af uhensigtsmæssigheder kan foretages med det samme - og specielt inden Hades tages i brug.

Det er naturligvis vigtigt at undgå de nuværende problemer og uhensigtsmæssigheder ved brugen af SimSys, så i afsnit 4.1 afdækkes disse. Med udgangspunkt i dette og i funktionaliteten som beskrevet i kapitel 3 opstilles i afsnit 4.2 målene for succes.

## 4.1 Problemer og uhensigtsmæssigheder

Problemer og uhensigtsmæssigheder ved brugen af SimSys afdækkes ved programmering i SimSys samt ved diskussion med de studerende på Dat1E, med instruktorer og undervisere på kurset og med Finn Schiermer Andersen, der har skabt og vedligeholdt SimSys.

Afsnittet er delt i fire dele, hvor første afsnit beskriver problemet ved brugen af C++, som er det problem, der ligger til grund for overhovedet at lave et nyt sprog. De tre sidste afsnit omhandler uhensigtsmæssigheder i SimSys, som det vil være hensigtsmæssigt at undgå i Hades.

### 4.1.1 C++

Som nævnt i forordet kræver brugen af SimSys ikke noget videre kendskab til C++, mens forståelsen af fejlmeddelelserne fra C++-oversætteren kræver enten et indgående kendskab til C++ eller en solid programmeringserfaring. Ingen af delene kan forventes af studerende på andet år, og det er heller ikke muligt at nå at lære dem dette indenfor rammerne af Dat1E.

### 4.1.2 Syntaks

At SimSys er et C++-klassebibliotek gør, at syntaksen bliver unødigt omfattende og indviklet. Dette er ikke nødvendigt i et specialiseret sprog, der kun skal bruges til løsning af problemer indenfor et snævert domæne, og hvor de fleste sprogkonstruktioner fra et generelt højniveausprog som C++ altså er unødvendige.

### 4.1.3 Signalveje og overskuelighed af programmer

Når en model for et kredsløb implementeres i SimSys, splittes signalvejene op i forbindelser mellem enkelte ud- og indstik. Dette gør det meget vanskeligt at overskue, hvilket kredsløb et program simulerer.

### 4.1.4 Fejlrejsning

Af implementeringsmæssige grunde har det været nødvendigt at lave SimSys, så en del fejl først rejses på kørselstidspunktet, heriblandt fejl i forbindelser af stik (hvis f.eks. stik med forskellige størrelser forbindes). Disse fejl bør rejses allerede på oversættelsestidspunktet.

## 4.2 Mål for Hades

Overordnet ønskes, at Hades har den nødvendige funktionalitet, at målgruppen (studerende på andet år) finder sproget rimelig simpelt at lære og bruge, samt at de nuværende problemer og uhensigtsmæssigheder undgås. Med hensyn til førstnævnte skal sproget have en funktionalitet som specificeret i afsnit 3.2 og 3.3 - herunder kunne bruges sammen med køretidssystemet. Dette er tilstrækkeligt til at sikre et brugbart design.

Med ovenstående for øje formuleres følgende mål for Hades :

- Det skal være muligt at lave simulationer på et niveau svarende til niveauet på Dat1E:
  - De vejledende løsninger til G- og K1-opgaven år 1999 (svarer til G2- og K1-opgaven, 2001) kan skrives i Hades.
  - Laboratorieøvelse 2 og 3 på hjemmesiden [Sim] kan løses ved brug af Hades.
  - Alle dele af SimSys som beskrevet i manualen [And01] kapitlerne 1-3, 5 og 8 kan programmeres i Hades.

  *Ovenstående sikrer, at Hades har den nødvendige funktionalitet, men ikke at Hades kan bruges sammen med køretidssystemet, eller at de i afsnit 3.3 beskrevne udvidelser er en del af sproget. Det er ikke nødvendigt at løse opgaver af nyere dato, da laboratorieøvelserne altid er de samme, og da rapportopgaverne varierer minimalt fra år til år.*

- Der skal være en entydig sammenhæng mellem de forskellige konstruktioner i et Hadesprogram og de byggeklodser, der bruges i køretidssystemet.

  *Dette mål sikrer, at kommunikationen med køretidssystemet er forståelig for brugeren, da systemet ellers ikke i praksis kan anvendes sammen med Hades. Målet er tilstrækkeligt til at sikre brugbarhed af systemet, da Hadesprogrammer oversættes til SimSysprogrammer, så kommunikation mellem simulationen og køretidsystemet foregår mellem sidstnævnte og SimSysprogrammet.*

- Det skal være muligt at lave vektorer med interne forbindelser, og det skal være nemt at lave træer og træniveauer.

  *Hermed er samtlige udvidelser en del af Hades.*

- Hades skal være enkelt:

  - syntaks og semantik skal være intuitiv.
  - det skal være nemt at lave et program ud fra et kredsløbsdiagram.
  - Hades skal være kompakt - dog uden at det går ud over simpelheden.

  *Ovenstående begreber er ikke præcise, men formålet med disse punkter er at sikre, at Hades er nemt at lære og at bruge for de studerende; altså at det passer til målgruppen, der potentielt kun har kort erfaring med at programmere. Grunden til at kompakthed er medtaget er, at det er hurtigere at skrive et program, der fylder få linier (under forudsætning af, at det ikke er blevet tilsvarende komplekst), samt at korte programmer typisk er mere overskuelige.*

- Problemer og uhensigtsmæssigheder specificeret i afsnit 4.1 må ikke optræde i Hades:
  - Hades må ikke kræve kendskab til C++.
  - Det skal være let at overskue signalveje i et program for et kredsløb, så programmer generelt er nogenlunde overskuelige.
  - Fejl skal rejses på oversættelsestidspunktet.

  *At syntaksen skal være enkel er udeladt af dette mål, da det indgår i det foregående. For at sikre at fejl rejses på oversættelsestidspunktet, skal det kontrolleres, at SimSysfejl er en del af semantikken for Hades, da denne følges ved implementering af sproget.*

# Kapitel 5

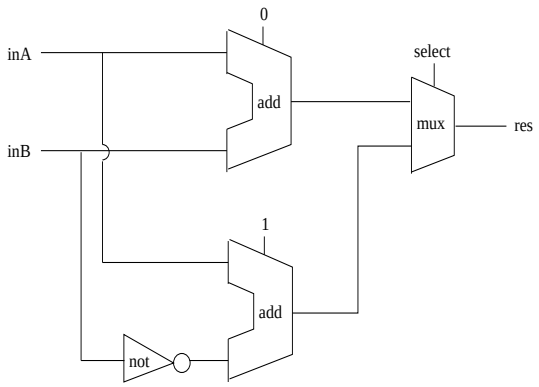
# Udformning af Hades

Baseret på den i kapitel 3 fastlagte funktionalitet og under hensyntagen til de i kapitel 4 opstillede mål for sproget, konstrueres Hades i dette kapitel. Der lægges vægt på, at sprogets udformning understøtter de studerendes forståelse af maskinen og sammenhængene i denne. Designet af Hades er grundigt overvejet, og Torben Mogensen og Martin Zachariasen (underviserne på Dat1E) har givet feedback på også de mindre detaljer. Disse er dog udeladt af denne analyse, så overblikket bevares.

Selvom Hades er et domænespecifikt sprog, kan abstraktionsniveauet variere fra et sprog, der minder om et (generelt) højniveausprog, til et sprog hvor der manipuleres med enkelte bit og strømme af bit. Derfor starter jeg med i afsnit 5.1 at fastlægge abstraktionsniveauet for Hades. Dernæst diskuterer jeg i afsnit 5.2 sprogets overordnede udformning, og i afsnittene 5.3, 5.4, 5.5, 5.6 og 5.7 diskuteres, hvordan forbindelser imellem komponenter nemmest udtrykkes. Herefter diskuteres i afsnit 5.8, hvordan et program skal se ud, og hvad det skal indeholde. I afsnit 5.9 overvejes brugen af komponenter i et kredsløb i Hades, og til sidst er der i afsnit 5.10 en kort diskussion af vektorer, træer og træniveauer.

## 5.1 Abstraktionsniveau

Lad os se på kredsløb




hvor `select` bruges til at vælge mellem værdierne `inA + inB` og `inA - inB`. Koden for dette kredsløb kan skrives noget i stil med

```
inA >> add0.inA
inA >> add1.inA
inB >> add0.inB
inB >> not.in
not.out >> add1.inB
0   >> add0.cIn
0   >> add1.cIn

add0.sum >> mux.in[0]
add1.sum >> mux.in[1]
select   >> mux.select

mux.out >> res
```

der ligger tæt op ad SimSys og illustrerer et abstraktionsniveau, hvor stikkene (*eng: channels*) for de enkelte komponenter forbindes parvis. Men abstraktionsniveauet i dette kodeeksempel er for lavt. Det giver mange liniers kode og så stor en detaljeringsgrad, at overblikket forstyrres. Det er rimelig nemt at skrive et program ud fra et kredsløbsdiagram men relativt svært at læse et program.

Et program kan i stedet være på et meget højere abstraktionsniveau, hvor brugeren ikke længere ser de enkelte komponenter. Et sådant program kan f.eks. skrives

```
res = case (select) of
        0: inA + inB
        1: inA - inB
```

Dette abstraktionsniveau virker umiddelbart fristende, da koden bliver kort og overskuelig. Der er dog den fare, at niveauet bliver så fjernt fra maskinniveau, at de studerende ”glemmer” maskinen og blot programmerer, som var det et højniveausprog. Dette er dog ikke et problem på Dat1E, da de skal lave et kredsløbsdiagram for deres simulator og på den måde ikke kan undgå at skulle overveje detaljerne i implementeringen. Men der er andre og mere væsentlige problemer.

Hvis køretidssystemet skal kunne bruges, og hvis den studerende skal kunne konstruere et program ud fra sit kredsløbsdiagram, skal der være en entydig korrespondance mellem komponenterne i diagrammet og de sproglige konstruktioner. Dette lader sig nemt gøre mellem en mux og en `case of`-konstruktion som illustreret ovenfor, men det bliver værre, når en operation som `-` betragtes. Udtrykket `A - B` er ækvivalent med det logiske udtryk `A + not B + 1`, hvor sidstnævnte er måden, `-` vil blive implementeret på, hvilket måske ikke er helt klart for den studerende, når vedkommende skriver sit program. Endnu værre bliver det, når man betragter et udtryk som `A + B + C`. Dette udtryk kan implementeres med to addere; men hvis `A`, `B` eller `C` er `1`, kan det implementeres mere effektivt med kun en adder. Altså vil oversætteren skjule implementeringsdetaljer for brugeren, eller også vil den i nogle tilfælde ikke lave den bedste løsning. Ingen af delene er acceptable, så jeg vælger et lavere abstraktionsniveau. Dette kan forhåbentlig også medvirke til samtidig at gøre det nemmere for de studerende at implementere deres modeller direkte ud fra deres kredsløbsdiagrammer.

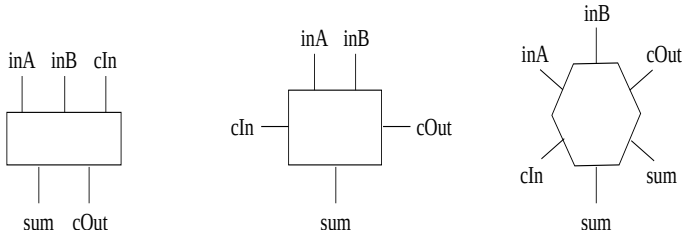
Ideen med at betragte komponenterne som byggeklodser er god, da den giver en tæt relation mellem et program og et kredsløbsdiagram. De mange liniers kode i det første af ovenstående eksempler skyldes hovedsageligt, at der er en linie kode pr. to stik. Da der som regel er mange stik i et kredsløb, giver dette mange liniers kode. Derfor hæves abstraktionsniveauet en smule, så komponenterne betragtes i deres hele. Hades konstrueres altså som et sprog til at samle byggeklodser. Det vil gøre det muligt at sammensætte to eller flere komponenter på én gang, og kan derved gøre et program mere kompakt uden forhåbentlig at ødelægge enkelheden. Med rette udformning kan det også gøre det nemt

at følge en datavej gennem et kredsløb og dermed at overskue, hvilken model et program simulerer, da datavejen ikke bliver splittet i mange små dele.

## 5.2 Sprogets udformning

At fortolke en komponent som en byggeklods betyder, at den opfattes som en klods med et antal sider, hvor hver side indeholder nul eller flere stik. Byggeklodserne samles, som byggeklodser normalt samles: ved at lægge dem oven på eller ved siden af hinanden. At sider fra to eller flere byggeklodser placeres ”op ad hinanden”skal forstås som, at stikkene i disse sider forbindes. Ved konstruktionen af Hades betragtes kun planare klodser, da programmer kun kan udtrykkes i ”to dimensioner”. Da der kun må laves forbindelser mellem ind- og udstik, betragtes desuden kun klodser, der ved overskæring kan omdannes til en klods med alle indstik og en med alle udstik, og hvor ingen side indeholder både ind- og udstik. Altså placeres ind- og udstik på hver sin del af klodsen. Det står dog stadig tilbage at afklare, hvordan klodsernes udformning skal opfattes samt ud fra denne at vælge den udformning af sproget, der mest optimalt understøtter, at klodserne samles.

*Nedenstående figur giver tre forskellige udformninger af byggeklodsen for en adder. Disse udformninger opfylder alle tre ovenstående krav.*

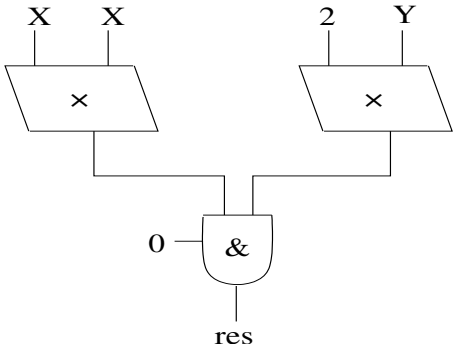



Figur 5.1: Tre forskellige byggeklodser for en adder.

Lad $i$ være antallet af sider i en byggeklods, der indeholder indstik, og definer tilsvarende $u$ som antal sider med udstik. *Dimensionen* af en byggeklods er så $\max(i, u)$, dvs. det maksimale antal sider med en type stik. Klodserne i figur 5.1 har altså dimensionerne en, to og tre.

Der er to oplagte måder at samle byggeklodser på. Sproget kan udformes, så det afspejler byggeklodsernes placering, og hvilke der forbindes. Dette er mest oplagt at bruge i forbindelse med klodser med to dimensioner. Dette illustreres nemmest ved et eksempel.

*Betragt kredsløbet*



*hvor indstik er placeret på venstre og på øverste side af byggeklodserne og udstik på de modstående sider. På de sider, der vender op og ned er placeret de stik, der typisk udgør en del af signalvejen. F.eks. er addenderne for additionsklodsen placeret på toppen af klodsen, mens "carry in" er placeret på siden. Dette kredsløb kan udtrykkes noget i stil med*

```
  x  x        2  y

multiply    multiply

0>    adder

       res
```

*At* `x x` *forbindes med den første multiplikationsklods og* `2 y` *med den næste kan nemt afgøres ud fra størrelserne af de enkelte klodser. Da en konstant (*`0`*) ikke tager inddata, vil udstikkene fra begge multiplikationsklodserne blive forbundet til indstikkene for additionsklodsen.* `>` *angiver, at* `0` *forbindes med "carry in" og ikke med* `res`.

Klodserne kan også samles i en dimension ad gangen på en måde, så det fremgår, hvilken dimension der samles hvornår. Ovenstående kredløb kan så i stedet udtrykkes som

```
vertical:
  x  x        2  y

multiply    multiply

      adder

       res

horisontal:
  0  adder
```

Først arbejdedes en del med at udforme et sprog, hvor todimensionale byggeklodser blev samlet i to dimensioner (som i ovenstående eksempel), dvs. hvor sproget selv er todimensionalt, og der blev konstrueret en skitse af et sprog. Denne udformning blev dog opgivet af to grunde: dels finder lærerne på Dat1E, at udformningen er for avanceret for de studerende, dels finder jeg selv, at de studerendes programmer typisk vil fylde ret meget, da det er nødvendigt at tænke på at lave kompakte programmer for at undgå dette. Se eksempler på et kompakt og et ikke-kompakt todimensionalt program i bilag A.

Dernæst eksperimenteredes med udformning af et endimensionalt sprog til at samle todimensionale klodser, men det blev hurtigt opgivet, da resultatet blev et enten alt for avanceret sprog eller et sprog, hvor programmer fylder for meget.

Sluttelig blev så konstrueret den endelige udformning af Hades: et sprog, hvor endimensionale klodser (den første klods i figur 5.1) samles i en dimension. Forbindelserne mellem byggeklodser laves så fra venstre mod højre som i SimSys men med den forskel, at flere klodser kan forbindes ad gangen, og at det tillades at forbinde ind- og udstik for en komponent i samme sætning. Dette gør det nemmere at lave sammenhængende definitioner af signalveje.

For en byggeklods defineres *indbitvektoren* til at bestå af alle bit i alle indstik i byggeklodsen. Rækkefølgen af stikkene i indbitvektoren og af bittene i de enkelte stik angiver rækkefølgen, så første bit i første stik er bit 0 osv.. *Udbitvektoren* for en byggeklods defineres tilsvarende. En *bitvektor* defineres som den sammensatte vektor bestående af en indbitvektor og en udbitvektor. En komponent kan altså opfattes som en bitvektor, hvor alle indstik er samlet i indbitvektoren, og alle udstik er samlet

i udbitvektoren. Når to bitvektorer sammensættes, danner de en ny bitvektor, der så kan sættes sammen med andre bitvektorer, og på den måde kan et kredsløb samles. Bittene ind i en bitvektor kaldes for bitvektorens *indbit*. Bitvektorens *udbit* defineres tilsvarende.

Følgende operationer gives til at samle to bitvektorer med:

```
A B
A , B
A >> B
```

hvor den første kaldes sekvens, den anden split og den sidste forbind.

*Sekvens* giver bitvektoren, hvor indbitvektoren består af indbitvektorerne for `A` og `B` lagt i forlængelse af hinanden, og udbitvektoren defineres tilsvarende.

I *split* skal `A`s og `B`s indbitvektorer have samme størrelse, og split giver så en indbitvektor af denne størrelse. Indbit til denne bitvektor kopieres og ledes til både `A` og `B`, mens udbitvektoren består af udbitvektorerne for `A` og `B` lagt i forlængelse.

*Forbind* er den eneste operation, der egentlig forbinder to bitvektorer. Det kræves, at `A`s udbitvektor og `B`s indbitvektor har samme størrelse. Alle udbit fra `A` ledes ind i alle indbit for `B`.

Sekvens har højere præcedens end split, der igen har højere præcedens end forbind. Brug af parenteser i bitvektorer tillades. Således vil bitvektoren

```
A >> B (C >> D) >> E
```

svare til bitvektorerne

```
A >> B C
C >> D
B D >> E
```

## 5.3 Afledninger af bitvektorer

Det er ikke nok at kunne forbinde komponenters bitvektorer i deres helhed. Det kan være nødvendigt at forbinde enkelte stik og sågar enkelte bit i bitvektorerne eller at kunne permutere bittene i en bitvektor (f.eks. i forbindelse med bitvise operationer). Derfor er det nødvendigt at kunne danne nye bitvektorer, hvor ind- eller udbitvektoren er en afledning af de oprindelige ind- og udbitvektorer.

Som tidligere nævnt er det vigtigt at bevare en funktionalitet, der er mindst på højde med SimSys', da denne har vist sig nødvendig og praktisk. Derfor skal alle afledninger i [And01, afsnit 2.5] bortset fra `split` og `join` understøttes. De to nævnte bruges blot til at ændre et stik til en vektor af stik og vice versa, men da der i Hades kun opereres med vektorer af bit, er disse operationer overflødige. En afledning af en bitvektor, er også en bitvektor. Dette giver selvsagt mulighed for at lave afledninger af afledninger, hvilket også er muligt i SimSys.

Det tillades, at der laves afledninger på alle bitvektorer, der består af en komponent eller er en sammensætning af komponenter. Det er ikke tilladt at lave afledninger på konstanter og afløb, da dette er overflødigt. Heller ikke afledninger på såkaldte *gennemløbsudtryk* godkendes. Et gennemløbsudtryk er en bitvektor, hvor ind- og udbitvektoren har samme størrelse, og hvor uddata er det samme som inddata. Gennemløbsudtryk bruges til at lede dele af en bitvektor forbi en komponent, når en signalvej angives.

Det er nødvendigt at kunne modificere ind- og udbitvektoren for en bitvektor separat. En modificeret indbitvektor kaldes for en *indafledning* og en modificeret udbitvektor for en *udafledning*. Ind- og udafledninger kaldes også for *bitafledninger* eller blot *afledninger*.

Følgende notation bruges

```
indafledning bitvektor udafledning
```

hvor udeladelse af en bitafledning vil betyde at bitvektoren bruges i sin helhed.

*Lad* `A`, `B` *og* `C` *være bitvektorer og* `ud0`, `ind1` *og* `ind2` *afledninger. Så betyder et udtryk som*

```
A ud0 >> ind1 B >> ind2 C
```

*at udbit fra* `A` *som udvalgt af* `ud0` *forbindes med indbit i* `B` *som udvalgt af* `ind1`*, og at alle udbit fra* `B` *tilsvarende forbindes med indbittene i* `ind2 C`.

Det tillades ikke, at der laves mellemrum mellem en bitvektor og eventuelle bitafledninger, da der så kan opstå situationer, hvor det ikke er klart, hvilken bitvektor en given bitafledning tilhører (da det jo er tilladt at undlade at bruge afledninger på en bitvektor).

## 5.4 Grundlæggende bitafledninger

Den grundlæggende bitafledning tager bit ud af bitvektoren, og til denne bruges notationen

```
[bitmønster]
```

hvor `bitmønster` angiver, hvilke bit der tages ud. Følgende bitmønstre tillades:

```
bit
bit0..bitN
stik
bitmønster , bitmønster
```

hvor `bit`, `bit0` og `bitN` skal være heltalsudtryk, og `stik` skal være navnet på et stik ind i eller ud af den komponent, afledningen tages på (dette bitmønster er ikke tilladt i afledninger, der anvendes på bitvektorer bestående af mere end en byggeklods). `bit` angiver en enkelt bit, og `bit0..bitN` sekvensen af bit, der starter med `bit0` og slutter med `bitN`. Hvis `bit0` $>$ `bitN`, er bittene i den afledte bitvektor spejlede i forhold til bittene i den oprindelige bitvektor. Et stik for en komponent udgør en sekvens af bit i ind- eller udbitvektoren for komponenten, og `stik` er altså blot syntaktisk sukker for `bit0..bitN`, hvor `bit0` er første bit i feltet for stikket og `bitN` sidste. Denne notation medtages, da den i mange tilfælde gør det nemmere at specificere eller at gennemskue en forbindelse. `bitmønster , bitmønster` angiver, at de to bitmønstre tages ud af bitvektoren og herefter lægges i forlængelse og danner en ny bitvektor. `,` har lavere præcedens end `...`

*En adder har stikkene* `inA`, `inB` *og* `cIn` *ind og stikkene* `out` *og* `cOut` *ud (i nævnte rækkefølge). Hvis* `add` *er en 32-bit adder, er* `[inA]add` *og* `[0..31]add` *ækvivalente, og* `add[cOut]` *og* `add[32]` *er ækvivalente.*

Et eksempel på en tilladt bitafledning er

```
[1..3, 32..63, 2, inA]
```

Bemærk, at det altså er tilladt at kopiere bit. Dette tillades dog kun i udafledninger, da en indbit kun må forbindes med netop en udbit.

For at lette programmeringen tillades det, at `bit0` og/eller `bitN` udelades af bitmønsteret

```
bit0..bitN
```

Udeladelse af `bit0` betyder første (mindst betydende) bit i bitvektoren, og udeladelse af `bitN` betyder tilsvarende sidste bit i bitvektoren. Disse forkortelser godkendes, da de er relativt intuitive, og da situationer, hvor de kan bruges, jævnligt optræder.

## 5.5 Feltvise bitafledninger

For at dække funktionaliteten af afledninger i SimSys, skal det være muligt at lave en afledning af en bitvektor ved at dele denne i felter, der hver består af samme antal bit, og så tage bestemte bit ud fra hvert af disse felter.

*Antag at man fra bitvektoren* `0123456789abcdef` *ønsker at tage første, femte, niende og tretende bit. Dette kan gøres ved at dele den i fire (lige store) dele og tage første bit fra hver af disse.*

Ved deling af bitvektorer i felter bruges følgende notation

```
/udtryk[bitmønster0 ; ... ; bitmønsterN]
```

`udtryk` er et heltalsudtryk $> 0$, og det skal gå op i det samlede antal bit i den ind- eller udbitvektor, afledningen laves på. Ovenstående bitafledning betyder, at bitvektoren deles i `udtryk` felter. Først tages `bitmønster0` fra hvert felt, så `bitmønster1` og så videre. Bitvektoren må gerne være en afledning af en bitvektor. Bitafledningen

```
[0..7]/4[0;1]
```

vil ændre `0123456789abcdef` til `02461357`. Forkortelsen

```
/udtryk
```

tillades og er en forkortelse for

```
/udtryk[bit0 ; ... ; bitN]
```

hvor `bit0` er første og `bitN` sidste bit i et felt. Dette gør det nemt at lave bitvise operationer. Hvis f.eks. `inA` og `inB` har størrelsen `N` vil

```
(inA inB)/2 >> andarray
```

svare til

```
(inA inB)/2[0;1;...;N-1] >> andarray
```

dvs. til at sende `inA` og `inB` ind i et `BWAndArray`.

## 5.6 Kopiering af bit i afledninger

Også kopiering (kun af udbit) indgår i SimSysafledninger og skal derfor kunne lade sig gøre i Hades. Det er allerede muligt at lave kopiering ved hjælp af operationen split, men det vil lette programmeringen i en del tilfælde, hvis der også laves en afledning til kopiering.

Følgende notation bruges:

```
*udtryk
```

hvor udbitvektoren som før kan være en afledning af en bitvektor. `udtryk` er et heltalsudtryk $> 0$ og angiver, hvor mange gange udbitvektoren kopieres. F.eks. vil bitafledningen

```
[9..12]*3
```

brugt på `0123456789abcdef` give `9abc9abc9abc9abc`.

## 5.7 Sammensætning af afledninger

Det kan være brugbart at kunne sammensætte flere bitafledninger. Til dette bruges notationen

```
bitafledning + bitafledning
```

hvor `+` binder svagere end både `*` og `/`. De resulterende bitvektorer lægges blot i forlængelse af hinanden og danner herved en bitvektor. Eksempelvis giver bitafledningen

```
[10..]*2+[..4]
```

brugt på `0123456789abcdef` bitvektoren `abcdefabcdef01234`.

Parenteser i bitafledninger er ulovlige, da de er unødvendige og blot forplumrer overblikket. Den ovenfor beskrevne funktionalitet af bitafledninger dækker fuldt ud funktionaliteten i SimSys (se afsnit 8.1) - og mere til.

To ting kan måske siges at være lidt mere besværligt at udtrykke i Hades end i SimSys: at manipulere et enkelt stik og at tage et enkelt felt ud af en vektor af stik. F.eks. udtrykkes

```
adder.inB.subset(0,3)
```

i Hades som

```
[32..34]adder
```

hvis `adder` er en 32-bit adder; og værre endnu udtrykkes

```
mux.in[3]
```

i Hades som

```
[96..127]mux
```

eller eventuelt som

```
[3*32..4*32-1]mux
```

for en 32-bit mux. Disse udtryk er dog værd at give køb på for til gengæld at få en notation, som er relativt simpel, og hvor det er nemt at manipulere en eller flere komponenter ad gangen.

## 5.8 Programmer i Hades

Nu er udformningen af kredsløb i Hades på plads, men selve opbygningen af programmer i sproget mangler stadig at blive fastlagt. Det gøres i dette afsnit.

Opbygningen af et program bliver som i SimSys, så det består altså af et hovedprogram og af definitioner af sammensatte komponenter. Dog skilles erklæring og definition for en sammensat komponent ikke, da begge dele typisk vil blive rimelig kortfattede (se f.eks. kapitel 7, hvor et Hadesprogram for en hurtig multiplikationsklods opbygges). Hovedprogrammet indeholder modellen for det kredsløb, der skal simuleres.

Hovedprogrammet og de sammensatte komponenter kaldes alle for *programblokke*. En programblok består af et hoved og en krop, hvor hovedet indeholder parametre for kroppen og eventuelt (hvis programblokken er en sammensat byggeklods) stik ud af og ind i byggeklodsen samt disses størrelse. Kroppen indeholder koden for det kredsløb, programblokken simulerer. Hvis nogle af parametrene til en programblok bruges til beregning af størrelserne af et eller flere stik, kaldes de *størrelsesvariable*

og skal være ikke-negative heltalsvariable. Alle andre parametre skal enten være boolske variable eller heltalsvariable. Typen fremgår af brugen af variablen i kroppen. Værdierne for parametre til hovedprogrammet skal specificeres på oversættelsestidspunktet.

Hoved og krop for en programblok er et virkefelt, og indlejrede virkefelter er ikke tilladt. Kroppen består af sætninger, hvor en sætning typisk består af en bitvektor. I disse bitvektorer tillades brugen af instanser af sammensatte komponenter og af indbyggede komponenter. Hades kommer til at indeholde stort set de samme indbyggede komponenter som byggeklodsbiblioteket i SimSys (se en liste over grænsefladerne for de indbyggede komponenter i Hades i afsnit 6.7). Kun vektorer af byggeklodser bliver ikke indbyggede, da disse nemt kan defineres af brugeren (se næste afsnit). Rekursion er tilladt, dvs. det er tilladt at bruge instanser af en sammensat byggeklods i definitionen af byggeklodsen selv. Også indbyrdes rekursion tillades. Der stilles blot det ene krav til rekursion, at når en instans af en byggeklods optræder i en rekursion, skal den være mindre end sidste gang en instans af denne byggeklods optrådte. Hvis $par0, ..., parN$ er parametre for den pågældende byggeklods, gælder $(par0, ..., parN) < (par0', ..., parN')$, hvis $(par1, ..., parN) < (par1', ..., parN')$ og $par0$ er et boolsk udtryk, eller hvis $par0$ er et heltalsudtryk, og hvis $par0 < par0'$ eller hvis $par0 = par0'$ og $(par1, ..., parN) < (par1', ..., parN')$.

Dette krav til rekursion sikrer, at programmer i Hades altid terminerer, da størrelsen for en komponent skal være positiv. Det er desuden stadig muligt at lave de konstruktioner, der ønskes, som det fremgår af afsnit 8.1.

En sætning i kroppen for en programblok kan også indeholde en betinget sætning. En sådan får syntaksen

```
if boolskUdtryk then krop else krop
```

hvor `boolskUdtryk` kun kan være en simpel sammenligning. Ud over dette er det nødvendigt at tillade definition af planer til brug i PLAer. Desuden tillades definition af såkaldte *bitvariable*:

```
var = bitvektor
```

hvor `bitvektor` ikke må indeholde >>. Bitvariable er egentlig ikke nødvendige, men de er praktiske, hvis en bitvektor sammensat af mange komponenter skal bruges flere steder, eller hvis en bitvektor er så stor, at den forplumrer en signalvej.

## 5.9 Brug af komponenter

Når en instans af en komponent som f.eks. en flipflop bruges, skal størrelsen og den initielle værdi fastlægges, og eventuelt skal byggeklodsen (altså flipflop), i denne sammenhæng også kaldet typen, angives. Instansen skal med andre ord erklæres. Dette kan gøres et vilkårligt sted i den programblok, hvor komponenten bruges, men må dog kun gøres en gang. Syntaksen for en erklæring bliver

```
navn'type andreParametre'
```

hvor der ikke må være mellemrum mellem instansens navn og dens erklæring (se sidst i dette afsnit). Kun for en PLA er erklæringen en smule anderledes, idet `type` udelades, og i stedet angives navnet på den plan, PLAen skal bruge.

Det er tilladt at udelade `type`, hvis typen fremgår af instansens navn. F.eks. vil `mux` og `minMux` begge være multiplexorer, med mindre andet angives i en erklæring. Dette gør, at f.eks. en notport slet ikke behøver blive erklæret, hvis dens type fremgår af navnet. Hvis navnene på to byggeklodser overlapper, vil oversætteren ved typeinferens vælge det navn, der matcher tidligst i strengen. Hvis to forskellige match starter samme sted, vil den længste streng, der matcher, blive valgt.

*Antag at brugeren har defineret to sammensatte byggeklodser med navnene* `minmux` *og* `mux2`. *Disse navne er overlappende og overlapper desuden med navnet på den indbyggede komponent* `mux`. *Hvis nu typen på en instans af en byggeklods skal udledes af dennes navn, vil instansen* `minmux2` *have typen* `minmux`, `mux12` *typen* `mux` *og* `mux2` *typen* `mux2`.

Det tillades også at specifikation af størrelsesvariable udelades, hvis størrelsen af en komponent kan infereres af programmet. Hvis f.eks. `inA` er 32 bit bred, kan det ud fra

```
inA >> flipflop
```

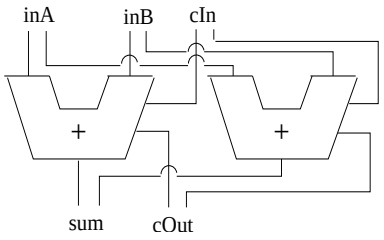
udledes, at `flipflop` er en 32-bit flipflop. Det er ikke tilladt at udelade andre parametre[1], så i ovenstående eksempel skal den initielle værdi for flipfloppen angives. Hvis værdien er 0, bliver erklæringen af flipfloppen altså `flipflop'0'`.

En byggeklods kan også være en konstant eller et afløb (skrives _). For disse angives kun en størrelse, men notationen er som ved andre byggeklodser. Størrelsen behøver kun at blive angivet, hvis den ikke kan udledes. Afløb beholdes i Hades, da det kræves, at alle udbit skal være forbundet.

Endelig tillades gennemløbsudtryk. Disse angives blot ved størrelsen, altså `'udtryk'`. Derfor skal erklæringen for en komponent stå sammen med dennes navn (uden mellemrum), da den ellers kan forveksles med et gennemløbsudtryk.

## 5.10 Vektorer og træer

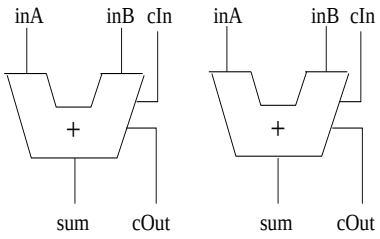
Vektorer af byggeklodser, hvor der ikke er forbindelser byggeklodserne imellem, konstrueres blot ved at tilføje `array` til typeerklæringen (som f.eks. i `muxarray`) samt tilføje antallet af byggeklodser i vektoren. Det er tilladt at bruge variablen `index` i resten af erklæringen for byggeklodsen, hvor `index` har samme værdi som klodsens placering i vektoren, startende med 0. En byggeklods bestående af en vektor af komponenter har samme stik ud og ind, som en af de indgående komponenter. Hvert af disse stik indeholder det tilsvarende stik fra hver af komponenterne som illustreret i




Ovenstående addervektor har altså tre indstik: `inA`, `inB` og `cIn`, og to udstik: `sum` og `cOut`. `inA` består af `inA` for de to addere i vektoren lagt i forlængelse af hinanden. De resterende stik er defineret tilsvarende.

I illustationer i resten af rapporten føres ledninger direkte ud af vektoren, og ovennævnte opfattelse lades implicit. Det betyder, at ovenstående vektor vil blive tegnet som

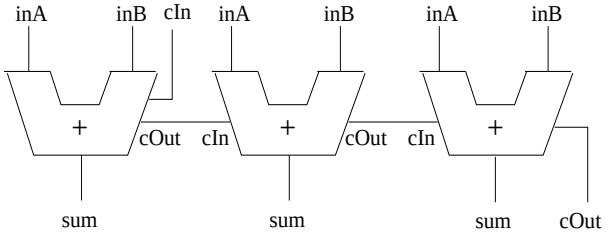
[1]Der laves ikke normalværdier (*eng: default values*) for disse, da det kan foranledige de studerende til blot at bruge en blok uden en erklæring og glemme en eventuel initialisering. Dette kan give mærkelige resultater, hvor fejlen kan være svær at finde.




Skal komponenterne forbindes internt, tilføjes et ekstra felt til erklæringen af vektoren, hvor disse forbindelser angives. Dette felt diskuteres længere nede.

Der bliver ikke en særlig konstruktion i sproget til at lave træer og træniveauer. Træniveauer kan nemt defineres som en sammensat komponent ved hjælp af vektorer og rekursion, og disse kan så samles til et træ (se i afsnit 7.4 hvordan f.eks. `CSATreeLevel` og `CSATree` kan defineres).

Feltet til at forbinde komponenter i vektorer med hinanden beskrives nemmest ud fra et eksempel. Betragt

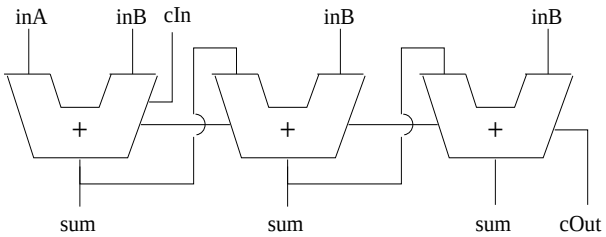



Her udtrykkes de interne forbindelser ved

```
cIn >> cOut
```

Som det ses af ovenstående bruges navnene på stikkene for de enkelte komponenter i vektoren som bitvektorer og forbindes som sådanne. `>>` må kun bruges en gang i denne slags bitvektorer. Der er den lille krølle, at når et udstik forbindes med et indstik, betyder det, at udstikket for komponent $i$ forbindes med indstikket for komponent $i + 1$; således lades første indbitvektor og sidste udbitvektor uforbundne. Når et stik (eller en del af et stik, for bitafledninger tillades også her) forbindes internt, løber dette ikke ind i eller ud af vektoren. Ind- og udbitvektorerne for vektoren er som for andre vektorer, hvor blot de internt forbundne stik ikke er en del af stikkene. Ovenstående vektor har altså indbitvektor `[inA,inB,cIn]`, hvor `inA` er alle `inA` for de indgående addere lagt i forlængelse af hinanden, `inB` er tilsvarende, mens `cIn` består af `cIn` for første komponent (mindst betydende). Dette gør det nemt at lave bitvist inddata, da det kan laves på kun `inA` og `inB`. Udbitvektoren er tilsvarende.

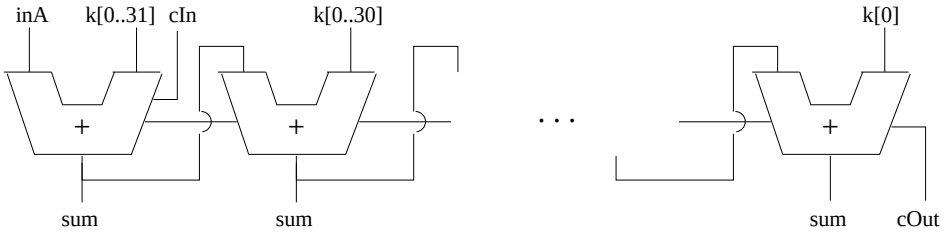
Der er andre problematikker. Betragt f.eks.




Her er problemet, at nogle stik skal forbindes både internt og eksternt. Her indfører jeg det specielle element `{o}`, der betyder forbindelse ud af vektoren. Således udtrykkes ovenstående som

```
cOut >> cIn ; out >> inB ; out >> {o}
```

Et tredje eksempel afslører en ny problematik. Lad adderne i nedenstående vektor have størrelse `32-index`.




Her er det umiddelbare problem, at der ikke er løkker eller lignende i Hades, så `k` udefra kan forbindes på passende måde til `inB`. Det er muligt at gøre dette ved at definere en sammensat byggeklods til at ændre `k` til `k[..31,..30,...,0]` ved brug af rekursion; men jeg tillader også, at forbindelsesfeltet indeholder forbindelser ud af vektoren, dvs. forbindelser som kun er delvist interne. For disse kræves, at den ene side er helt ekstern og den anden side helt intern. Udstik i vektoren, der forbindes således, indgår heller ikke i udbitvektoren.

Nu kan ovenstående udtrykkes

```
k[0..31-index] >> inB ; sum >> inA ; sum >> {o} ; cIn >> cOut
```

Her kommer indbitvektoren for addervektoren så kun til at bestå af `inA` og `cIn` for den første adder i vektoren.

# Kapitel 6

# Grammatik og semantik for Hades

I dette kapitel gives grammatik og semantik for Hades. Grammatikken er beskrivende, så der er ikke lagt vægt på entydighed. Semantikken er beskrevet uformelt ved de enkelte produktioner.

I afsnit 6.1 beskrives opbygningen af et program og i 6.2, hvordan kroppen for en programblok ser ud. Herefter beskrives bitvektorer i afsnit 6.3 og så bitafledninger i 6.4. Instantiering for komponenter, afløb mm. gives i afsnit 6.5 og tal, udtryk og lignende i afsnit 6.6. Til sidst gives grænsefladerne for alle indbyggede komponenter i afsnit 6.7 og diverse småting i afsnit 6.8.

## 6.1 Program

*program* ::= *kompdefsEvt hovedprogram kompdefsEvt*

Et program består af et hovedprogram og en eller flere definitioner af sammensatte byggeklodser. Rækkefølgen er ligegyldig. En sammensat byggeklods kan altså godt bruges i f.eks. hovedprogrammet og først blive defineret efter dette.

*hovedprogram* ::= `main (` *strengeEvt* `)` *krop* `end`

*strengeEvt* er parametre til hovedprogrammet og kan bruges i *krop*, der udgør et virkefelt. Typen af en variabel afgøres af brugen i kroppen og skal være enten et heltal eller en boolsk værdi. Værdierne for parametrene angives på oversættelsestidspunktet, og der rejses fejl, hvis de har en ulovlig type, eller hvis der er et forkert antal. I kroppen specificeres det kredsløb, programmet skal simulere.

*kompdefsEvt* ::= $\epsilon$ | *kompdef kompdefsEvt*
*kompdef* ::= *streng* `(` *stikEvt kommaStrEvt* `) >> (` *stikEvt* `)` *krop* `end`

*streng* angiver navnet for den sammensatte byggeklods og må ikke være det samme som navnet på en anden sammensat byggeklods eller på en indbygget komponent (se afsnit 6.7). Første *stikEvt* angiver eventuelle stik ind i komponenten og andet *stikEvt* eventuelle stik ud samt størrelsen på disse. En sammensat komponent skal have mindst et stik. *kommaStrEvt* angiver navnene for eventuelle parametre til byggeklodsen, og disse må bruges som variable både i *stikEvt* og i *krop*, da den samlede definition af en byggeklods udgør et virkefelt. Hvis en sammensat byggeklods ikke bruges, kontrolleres dens krop ikke for andre fejl end de syntaktiske.

I kroppen specificeres kredsløbet indeni byggeklodsen. I dette kredsløb er det lovligt at bruge instanser af byggeklodsen selv, hvis blot værdien af parametrene til det nye kald er blevet mindre.

At parametrene $par0, ..., parN$ for en byggeklods er mindre end parametrene $par0', ..., parN'$ betyder, at hvis $par0$ er et boolsk udtryk, gælder $(par1, ..., parN) < (par1', ..., parN')$, ellers gælder $par0 < par0'$, og ellers $par0 = par0'$ og $(par1, ..., parN) < (par1', ..., parN')$.

*stikEvt* ::= $\epsilon$ | *strenge udtryk stikEvt*

*strenge* angiver navnene på et eller flere stik og *udtryk* størrelsen for disse. Ingen stik må have samme navn (hverken ind- eller udstik), og størrelsen for et stik skal være positiv. Et udtryk kan også være en streng (en variabel), men i så fald skal denne være en parameter til den pågældende programblok. Der rejses fejl, hvis en variabel i et udtryk ikke er erklæret.

*kommaStrEvt* ::= $\epsilon$ | `,` *strenge*

`strenge` giver navnene på parametre for komponenten. Typen bestemmes ud fra brugen i kroppen og er enten heltal eller bool. Navnet på en parameter må ikke være det samme som navnet på en anden parameter eller på et stik.

## 6.2 Krop

*krop* ::= *sætning* | *sætning* `;` *krop*

Kroppen angiver det kredsløb, en programblok simulerer, og indeholder forbindelser mellem komponenter mm.. Der rejses fejl, hvis en bit i en komponent eller en bit i et stik ind i eller ud af programblokken ikke er forbundet med en bit i en anden bitvektor, eller hvis en indbit er forbundet mere end en gang. Desuden rejses fejl, hvis en kreds af komponenter i kroppen ikke indeholder et tilstandselement. Sætningerne i en krop specificerer tilsammen det kredsløb, som kroppen simulerer. En parameter til en programblok har som ovenfor nævnt en type og skal bruges i overensstemmelse med denne (en parameter må altså ikke et sted bruges som en boolsk variabel og et andet sted som en heltalsvariabel).

*sætning* ::= `if` *boolskUdtryk* `then` *krop* `else` *krop*

På oversættelsestidspunktet bestemmes hvilken del af en betinget sætning, der skal bruges, og den anden del elimineres af programmet. Ingen kontrol af korrekthed vil blive udført på den udeladte del af programmet.

*sætning* ::= *bitvektor*

En bitvektor specificerer en del af (eller hele) det kredsløb, der skal simuleres.

*sætning* ::= *streng* `=` *bitvektorUPil*

Dette angiver erklæring af en bitvariabel med navn *streng* og værdi *bitvektorUPil*, hvor *bitvektorUPil* angiver en bitvektor, hvor `>>` ikke optræder. En bitvariabel kan bruges i hele kroppen (også før definition) og bruges som en byggeklods. Dens værdi substitueres ind alle steder, hvor den bruges. To bitvariable må ikke have samme navn, og navnet må heller ikke være det samme som for en komponent i kroppen. Det er tilladt at bruge andre bitvariable i *bitvektorUPil*, men rekursion er ikke tilladt.

*sætning* ::= *streng* `=` `"` *plan* `"`

*streng* er navnet på et sæt af and- og orplaner (kaldes fremover en plan), der bruges til at specificere en PLA. Navnet må ikke falde sammen med navne på komponenter, andre planer eller stik i programblokken.

*plan* ::= *andplan orplan* | *andplan orplan* `,` *plan*

*andplan* angiver den streng, inddata til PLAen skal matche, mens *orplan* specificerer de ettaller i uddata, en given andplan forårsager. Et inddata kan godt matche flere andplaner, og i givet fald vil uddata have ettaller på alle de pladser, de tilhørende orplaner har ettaller. Alle andplaner skal have samme længde, og det samme gælder alle orplaner.

*andplan* ::= `0` | `1` | `-` | `0` *andplan* | `1` *andplan* | `-` *andplan*

`0` og `1` i en andplan skal matche `0` henholdsvis `1` på den tilsvarende position i inddata til PLAen. `-` matcher hvad som helst.

*orplan* ::= `1` | `-` | `1` *orplan* | `-` *orplan*

`1` bidrager med et ettal på den tilsvarende position i uddata for PLAen, mens `-` ikke yder noget. De positioner i uddata, hvor der ikke ledes `1`, vil automatisk være `0`.

## 6.3 Bitvektorer

*bitvektor* ::= *bitvektor* >> *bitvektor*

To bitvektorer forbindes ved at uddata fra den første ledes ind i inddata for den anden, og udbitvektoren for første bitvektor skal have samme størrelse som indbitvektoren for den næste. Resultatet bliver en ny bitvektor bestående af indbitvektoren for den første bitvektor og udbitvektoren for den sidste. Hvis der indgår skel i en bitvektor, skal et sådant stå alene mellem to >>, og for udtrykket *bitvektor* >> *skel* >> *bitvektor'* bliver den resulterende bitvektor som for udtrykket *bitvektor* >> *bitvektor'*. Udbittene for *bitvektor* ledes ind i indbittene for *bitvektor'* via skellet, så det kræves stadig, at ind- og udbitvektorerne har samme størrelse.

*bitvektor* ::= *bitvektor* `,` *bitvektor*

To bitvektorer sættes sammen (uden at forbindes). De to indbitvektorer skal have samme størrelse $\neq 0$, og den nye indbitvektor får også denne størrelse. Alle indbit til den samlede bitvektor ledes til både indbitvektoren for første og indbitvektoren for anden bitvektor. Den samlede udbitvektor består af de to udbitvektorer lagt i forlængelse af hinanden. Denne operation har højere præcedens end >>.

*bitvektor* ::= *bitvektor bitvektor*

To bitvektorer i forlængelse af hinanden giver en bitvektor, hvor indbitvektoren består af de to indbitvektorer lagt i forlængelse af hinanden, og udbitvektoren består af de to udbitvektorer lagt i forlængelse af hinanden. Denne operation har højere præcedens end de to foregående.

*bitvektor* ::= *indafledningEvt* ( *bitvektor* ) *udafledningEvt*

Der kan laves afledninger af en sammensat bitvektor ved at omgive denne med parenteser. Der rejses fejl, hvis *indafledningEvt* eller *udafledningEvt* angiver bit udenfor *bitvektor*. Der må ikke være mellemrum mellem *indafledningEvt* og ( og mellem ) og *udafledningEvt*.

*bitvektor* ::= *indafledningEvt streng typeEvt udafledningEvt*

En bitvektor kan også bestå af en indbygget eller sammensat komponent, af et skel eller af et stik for programblokken. *streng* angiver navnet på denne. For et stik må ikke angives en type, mens en komponent eller et skel skal erklæres højst en gang, og dette kan gøres et vilkårligt sted i programblokken. Stik og komponenter kan altså godt bruges, før de er erklæret. En erklæring kan i nogle tilfælde være overflødig (se afsnit 6.5). Komponenter og skel må ikke have samme navn eller samme navn som en plan, en bitvariabel eller et stik for den pågældende programblok. Der må ikke laves afledninger på skel, og der rejses desuden fejl, hvis en bit i en afledning ligger udenfor komponentens bitvektor. Der må ikke være mellemrum mellem *indafledningEvt* og *streng*, mellem *streng* og *typeEvt* og mellem *typeEvt* og *udafledningEvt*.

*bitvektor* ::= *tal størrelseEvt*

En konstant har en tom indbitvektor og et konstant uddata indeholdende værdien af *tal*. Udbitvektorens bredde specificeres af *størrelseEvt* og kan udelades, hvis den kan deduceres fra resten af programmet. Der må ikke være mellemrum mellem *tal* og *størrelseEvt*.

*bitvektor* ::= _ *størrelseEvt*

_ angiver et afløb, dvs. en bitvektor med tom udbitvektor. Afløb bruges til afledning af uforbundne udbit, så fejlrejsning undgås. Hvis størrelsen af afløbet kan udregnes ud fra resten af programmet, kan denne udelades. Der må ikke være mellemrum mellem _ og *størrelseEvt*.

*bitvektor* ::= *størrelse*

Dette er notationen for en gennemløbsvariabel. Denne giver en bitvektor, der er *størrelse* bit bred, og hvor inddata og uddata er det samme. Den kan skelnes fra en type ved, at den ikke står op ad en streng.

*bitvektorUPil* ::= *bitvektorUPil* , *bitvektorUPil*
| *bitvektorUPil bitvektorUPil*
| *indafledningEvt* ( *bitvektorUPil* ) *udafledningEvt*
| *indafledningEvt streng typeEvt udafledningEvt*
| *tal størrelseEvt*
| _ *størrelseEvt*
| *størrelse*

*bitvektorUPil* er fuldstændig som *bitvektor* med den ene forskel, at >> ikke er tilladt. Den bruges kun til at angive værdien af en bitvariabel.

## 6.4 Afledninger

*indafledningEvt* ::= $\epsilon$ | `[]` | *indafledning*

Hvis der ikke er nogen afledning, betyder det, at indbitvektoren bruges i sin fulde udstrækning. `[]` betyder, at indbitvektoren udelades, mens *indafledning* angiver, hvilke bit i indbitvektoren der bruges.

*indafledning* ::= *indafledning* `+` *indafledning*

Hver indafledning tages på indbitvektoren og lægges herefter i forlængelse af hinanden og danner en ny indbitvektor.

*indafledning* ::= `[` *bitmønster* `]` *irestEvt*

*bitmønster* angiver position og rækkefølge af de bit, der tages af indbitvektoren. Herved dannes en ny indbitvektor. *irestEvt* angiver eventuelle operationer, der skal udføres på denne nye indbitvektor. Der må ikke være mellemrum mellem `]` og *irestEvt* og mellem *irestEvt* og den bitvektor, indafledningen tages på. Hvis en bit i *bitmønster* er udenfor indbitvektoren, rejses fejl.

*indafledning* ::= *irest*

*irest* angiver operationer, der udføres på den samlede indbitvektor for herefter at danne en ny. Der må ikke være mellemrum mellem *irest* og bitvektoren.

*irestEvt* ::= $\epsilon$ | *irest*
*irest* ::= `/` *udtryk* `[` *felter* `]` *irestEvt*

*udtryk* angiver, hvor mange felter indbitvektoren skal deles i, og dette tal skal være $> 0$ og gå op i antal bit i indbitvektoren. *felter* angiver, hvordan bittene fra disse felter skal sættes sammen til en ny indbitvektor. Der må ikke være mellemrum mellem *udtryk* og `[` og mellem `]` og *irestEvt*.

*irest* ::= `/` *udtryk irestEvt*

er en forkortelse for `/` *udtryk* `[0;1;2;...]`.

*udafledningEvt* ::= $\epsilon$ | `[]` | *udafledning*
*udafledning* ::= *udafledning* `+` *udafledning*
| `[` *bitmønster* `]` *urestEvt*
| *urest*
*urestEvt* ::= $\epsilon$ | *urest*
*urest* ::= `*` *udtryk urestEvt*
| `/` *udtryk urestEvt*
| `/` *udtryk* `[` *felter* `]` *urestEvt*

Udafledninger fungerer stort set som indafledninger; kun med den undtagelse, at der findes det ekstra udtryk `*` *udtryk*, der betyder, at udbitvektoren gentages *udtryk* gange. Her skal gælde *udtryk* $> 0$.

*felter* ::= *bitmønster*

Bittene specificeret af *bitmønster* tages fra første felt, så fra andet felt osv., og alle disse bit lægges i forlængelse og danner derved en ny ind- eller udbitvektor. Der rejses fejl, hvis en bit i *bitmønster* ligger udenfor (størrelsen af) et felt.

*felter* ::= *bitmønster* `;` *felter*

På samme måde som ovenfor beskrevet danner bittene specificeret af *bitmønster*, taget fra hvert felt, en bitvektor, og denne bruges som starten af en ny bitvektor, hvor resten af den angives af *felter*. Der rejses fejl, hvis en bit i *bitmønster* ligger udenfor (størrelsen af) et felt.

*bitmønster* ::= *bitmønster* `,` *bitmønster*

Bittene angivet af andet *bitmønster* lægges i forlængelse af bittene angivet af første *bitmønster*.

*bitmønster* ::= *udtryk*

Hvis *udtryk* er en streng, der er navnet på et stik for den komponent, afledningen laves på (kan ikke bruges, hvis afledningen er på en bitvektor omgivet af parenteser), er ovenstående syntaktisk sukker for `bit0..bitN` (se nedenfor), hvor `bit0` er første og `bitN` sidste bit i stikket. Ellers udregnes *udtryk* til et heltal, der angiver nummeret på den bit, der vælges fra ind- eller udbitvektoren. Første bit i en bitvektor er bit 0.

*bitmønster* ::= *udtrykEvt* `..` *udtrykEvt*

Giver en sekvens af bit startende fra bit *udtrykEvt* og til og med bitten angivet af andet *udtrykEvt*. Hvis andet udtryk er mindre end første, er bittene i omvendt rækkefølge. Hvis første udtryk udelades, svarer det til at skrive `0`, mens at udelade andet udtryk angiver den sidste bit i den pågældende ind- eller udbitvektor.

## 6.5 Typer og størrelser

*typeEvt* ::= $\epsilon$

Komponenten eller skellet er erklæret andetsteds, eller en erklæring er overflødig (se nedenfor); eller også hører typeerklæringen til et stik for programblokken.

*typeEvt* ::= `'` *skelEvt streng* `[` *udtryk* `]` `'`

Typen for et skel angiver, hvilken slags skel det drejer sig om (*skelEvt*) samt hvorfra den udbit, der kontrollerer skellet, kommer. *streng* angiver den pågældende komponents navn og *udtryk* hvilken bit, der tages. Denne bit skal ligge indenfor udbitvektoren for komponenten.

*typeEvt* ::= `'` *strengEvt udtrykEvt udtrykFlereEvt forbindelserEvt* `'`

*strengEvt* angiver komponenttypen og kan udelades, hvis typen fremgår af navnet på komponenten (som i f.eks. `mux` eller `andarray`). En type fremgår af et navn, hvis navnet på typen er en delstreng i navnet. Hvis to typenavne optræder i navnet, vælges den første, og hvis de starter samme sted, vælges den med længst navn. Et byggeklodsnavn efterfulgt af `array` vil lave en vektor bestående af denne slags byggeklodser. Hvis navnet for denne vektor er `navn`, vil navnet for hver klods i vektoren blive `navn0`, `navn1` osv.. Udledes typen af klodsens navn, vil denne være en vektor, hvis blot `array` indgår i dette; `array` behøver altså ikke stå i forlængelse af navnet for typen af komponenter for vektoren (f.eks. vil `mux3array` og `arrayOfMux` være vektorer af muxe med mindre andet angives). Typen for en byggeklods i et kredsløb kan være en af de indbyggede komponenter (se afsnit 6.7) eller en af brugeren defineret byggeklods. Hvis en byggeklods er en vektor, er bitvektoren for klodsen som

følger: den indeholder samme indstik og udstik som en af de indgående komponenter, men hvert stik består af stikket af denne type fra hver af de indgående komponenter i samme rækkefølge som disse indgår. Desuden kan *udtrykFlereEvt* indeholde variablen `index`, der for hver komponent er dennes placering i bitvektoren (`index` $= 0, 1, ...$).

*udtrykEvt* udelades, hvis byggeklodsen ikke er en vektor. Ellers skal det være positivt og angiver antallet af komponenter i vektoren. *udtrykFlereEvt* angiver værdier til parametrene for komponenten (eller for de indgående komponenter, hvis byggeklodsen er en vektor). Det er lovligt at udelade værdier for størrelsesvariable, hvis disse kan udledes af programmet. Der rejses fejl, hvis der angives flere eller færre værdier, end der er parametre til den type byggeklods, hvis en værdi har en forkert type, hvis en størrelsesvariabel tildeles en negativ værdi, eller hvis størrelsesvariable ikke tildeles værdier, og disse ikke kan udregnes. Bortset fra for komponenterne `IdealMemIOSys` og `MemIOSys` er kun typerne heltal og bool tilladt. For de to nævnte blokke skal tredje henholdsvis syvende parameter være en streng (nemlig et filnavn).

*forbindelserEvt* må kun specificeres for vektorer og angiver eventuelle interne forbindelser mellem de indgående komponenter.

*skelEvt* ::= $\epsilon$ | `pipestage` | `pipestageboundary` | `zap` | `zapboundary`

Typen af skellet kan udelades, hvis `pipestage`, `pipestageboundary`, `zap` eller `zapboundary` indgår i navnet for byggeklodsen.

*forbindelserEvt* ::= $\epsilon$ | `;` *ud* `>>` *ind forbindelserEvt*

Angiver forbindelser mellem komponenterne i en vektor af byggeklodser og eventuelt også ud af byggeklodserne. Udbittene fra bitvektoren *ud* ledes ind i indbitvektoren for *ind*, så disse skal have samme størrelse. Hele *ud* eller hele *ind* skal være stik fra komponenterne i vektoren, og hvis *ud* eller *ind* indeholder komponenter udefra, skal den være helt ekstern. Hvis en forbindelse er helt intern (begge sider er stik fra komponenterne i vektoren), angiver den, at de specificerede udstik for første komponent forbindes til de specificerede indstik for anden komponent og så fremdeles, så de specificerede udstik til sidst er forbundet for alle interne komponenter bortset fra den sidste. Ligeledes er alle indstikkene forbundet for de interne komponenter bortset fra den første. Er en forbindelse ekstern (dvs. en ekstern og en intern side), forbindes stikkene angivet i den "interne side" for alle komponenterne i vektoren. De forbundne stik fjernes alle fra byggeklodsens bitvektor, der altså bliver mindre. Variablen `index` kan indgå i bitafledninger i *ud* og *ind* og er 0 for første komponent, 1 for næste osv..

*ud* ::= *ud ud*

De to bitvektorer lægges i forlængelse af hinanden til en ny bitvektor.

*ud* ::= `(` *ud* `)` *udafledningEvt*

Bittene *udafledningEvt* tages fra *ud* og laves til en ny bitvektor. Alle bit skal være indenfor *ud*s udbitvektor.

*ud* ::= *streng udafledningEvt*

*streng* angiver en komponent, en bitvariabel, et stik for programblokken eller et stik for den komponenttype, vektoren er opbygget af. De tre første fungerer som almindeligvis i bitvektorer, mens et stik angiver en del af udbitvektoren for en intern komponent.

*ud* ::= *konstant*

Der kan ikke angives størrelser for konstanter, da `'` afslutter typeerklæringen for vektorbyggeklodsen. Dette er heller ikke nødvendigt, da angivelse af interne forbindelser er ret simple, og det vil derfor typisk være meget nemt at inferere størrelsen.

*ind* ::= `{o}`
| *indrest*
*indrest* ::= *ind* `,` *ind*
| *ind ind*
| *indafledningEvt* `(` *ind* `)`
| *indafledningEvt streng*
| `_`

*indrest* fungerer stort set som *ud* med den forskel, at også *ind* `,` *ind* tillades (og betyder det samme som i andre bitvektorer), samt at afløb tillades i stedet for konstanter. `,` har lavere præcedens end mellemrum. I *ind* `>>` `{o}` skal *ind* være interne stik, og udtrykket betyder, at disse stik føres ud af komponenten, selv om de også er forbundet internt.

*størrelseEvt* ::= $\epsilon$ | *størrelse*
*størrelse* ::= `'` *udtryk* `'`

*udtryk* angiver størrelsen af en konstant, et afløb eller et gennemløbsudtryk og skal være positiv.

## 6.6 Udtryk, strenge og tal

*udtrykFlereEvt* ::= $\epsilon$ | *udtryk udtrykFlereEvt*
*udtrykEvt* ::= $\epsilon$ | *udtryk*
*udtryk* ::= *udtryk* `+` *udtryk* | *udtryk* `-` *udtryk* | *udtryk* `*` *udtryk*
| *udtryk* `/` *udtryk* | *udtryk* `^` *udtryk* | *udtryk* `mod` *udtryk*
| *streng* | *tal*

De sædvanlige præcedens- og associativitetsregler gælder for regneoperationerne. `/` dividerer to heltal, og fungerer som `div` i ML. Der rejses fejl, hvis en variabel i et udtryk ikke er erklæret, eller hvis den har en ulovlig type.

*boolskUdtryk* ::= *udtryk* `<` *udtryk* | *udtryk* `<=` *udtryk* | *udtryk* `=` *udtryk*
| *udtryk* `>=` *udtryk* | *udtryk* `>` *udtryk*
| *streng* | `true` | `false`

Der rejses fejl, hvis variablen *streng* ikke er erklæret eller ikke er en boolsk variabel.

*strengeEvt* ::= $\epsilon$ | *strenge*
*strenge* ::= *streng strengeEvt*
*strengEvt* ::= $\epsilon$ | *streng*
*streng* ::= *bogstav strengRest*
*strengRest* ::= $\epsilon$ | *bogstav strengRest* | *ciffer strengRest* | `_` *strengRest*
*tal* ::= `0x` *hextal* | *dectal*

*dectal* ::= *ciffer* | *ciffer dectal*
*hextal* ::= *hexciffer* | *hexciffer hextal*
*ciffer* ::= *0* | ... | *9*
*hexciffer* ::= *ciffer* | `a` | ... | `f` | `A` | ... | `F`
*bogstav* ::= `a` | ... | `z` | `A` | ... | `Z`

## 6.7 Indbyggede komponenter

Dette afsnit indeholder navn og grænseflade for alle de indbyggede komponenter. Funktionaliteten af disse er den samme som i SimSys, så beskrivelser kan ses i [And01, kap. 8]. For nogle komponenter er der angivet to navne (f.eks. `notgate/not`). Dette betyder, at begge navne kan bruges til at betegne denne type komponent.

- `addends (multiplier height multiplicand width, height width) >> (addends height*width)`
- `adder/add (inA inB sz cIn 1, sz) >> (sum sz cOut 1)`
- `andgate/and (in width) >> (out 1)`
- `nandgate/nand (in width) >> (out 1)`
- `orgate/or (in width) >> (out 1)`
- `norgate/nor (in width) >> (out 1)`
- `xorgate/xor (in width) >> (out 1)`
- `csadder/csadd (inA inB inC sz, sz) >> (sumA sumB sz)`
- `csatree (addends arguments*width, arguments maxLevels width) >> (sums results*width)`
- `csatreeLevel (in nrOfInMembers*wordSz, nrOfInMembers wordSz) >> (out (2*(nrOfInMembers/3)+nrOfInMembers mod 3)*wordSz)`
- `decoder (in selectsz, selectsz) >> (out 2^selectsz)`
- `encoder (in 2^logsz, logsz) >> (out logsz valid 1)`
- `fastmux (in selectsz*datawidth select selectsz, selectsz datawidth) >> (out datawidth)`
- `flipflop/ff (in sz, sz init) >> (out sz)`
- `fulladder/fulladd (inA inB inC 1) >> (outA outB 1)`
- `fulladderarray (inA inB width cIn 1, width) >> (sum width cOut 1)`
- `halfadder/halfadd (inA inB 1) >> (outA outB 1)`
- `idealmemiosys (iAddrIn 32 iRd 1 dAddrIn 32 dRd dWr 1 dDataIn 32 , logmemsz accesstime fname) >> (iDataOut dDataOut 32)`

- memiosys (iAddrIn 32 iRd 1 dAddrIn 32 dRd dWr 1 dDataIn 32
  , sets idxsz blocksz memidxsz memfstchunk memNxtChunk fname iportsz)
  >> (iDataOut 32*2^iportsz dDataOut 32 holdAndWait 1)
- merger (streamIn streamsz dataIn datasz mergeMask streamsz/datasz
  , streamsz datasz) >> (streamOut streamsz)
- multiply/mult (multiplier height multiplicand width, height width)
  >> (product width)
- mux (in 2^selectsz*datawidth select selectsz
  , selectsz datawidth) >> (out datawidth)
- notgate/not (in 1) >> (out 1)
- pgnode (pIn gIn width cIn 1, width) >> (cOut width pOut gOut 1)
- pgtree (pIn gIn width cIn 1, width nodewidth)
  >> (carries width cOut 1)
- pla (in insz, insz outsz height planes) >> (out outsz)
- prioritizer (in sz, sz) >> (out sz valid 1)
- priorityencoder (in 2^logsz, logsz) >> (out logsz valid 1)
- qualifieddecoder(in selectsz qualifier 1, selectsz)
  >> (out 2^selectsz)
- qualifiedsearch (keys fields*fieldwidth key fieldwidth valid fields
  , fields fieldwidth) >> (hits fields)
- registerfile/regfile (wrSelect wrports*selectsz wrEnable wrports
  wrData wrports*datasz rdSelect rdports*selectsz
  , wrports rdports selectsz datasz) >> (rdData rdports*datasz)
- search (keys fields*fieldwidth key fieldwidth
  , fields fieldwidth) >> (hits fields)
- serialadder/serialadd (inA inB sz cIn 1, sz) >> (sum sz cOut 1)
- sram (wrEnable wrports*lines dataIn wrports*datasz
  rdEnable rdports*lines, wrports rdports lines datasz)
  >> (dataOut rdports*datasz)
- validatewrite (idxIn ports*idxsz dataIn ports*datasz wrEnable ports
  , ports idxsz datasz domain) >> ()
- wrenflipflop/wrenff (in sz wrEnable 1, sz init) >> (out sz)

## 6.8 Diverse

Hades er ikke følsomt overfor størrelsen af et bogstav (dvs. `inA` og `ina` er ækvivalente). Følgende ord er reserverede: `main`, `end`, `mod`, `if`, `then`, `else`, `true` og `false`. Som start på etlinies kommentarer bruges `//`, mens afsluttede kommentarer omsluttes af `/* */`.

# Kapitel 7

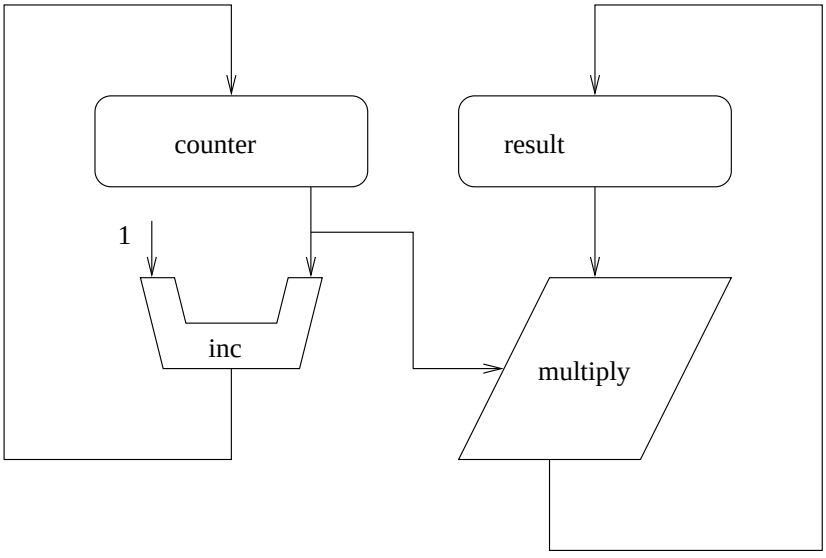
# Hurtig multiplikation i Hades

I dette kapitel illustreres brugen af Hades ved et eksempel. Som eksempel bruges faktorialkredsløbet fra kursusbogen [And01, afs. 2.9 og kap. 5], da Hadesprogrammet for dette illustrerer vektorer med interne forbindelser, træer og træniveauer samt feltvise afledninger, dvs. de mest avancerede konstruktioner i sproget. De to figurer i kapitlet er taget fra kursusbogen. Byggeklodserne konstrueret i kapitlet er indbyggede i Hades.

I afsnit 7.1 bygges hovedprogrammet for et kredsløb, der implementerer faktorialfunktionen og i afsnit 7.2 multiplikationsklodsen, der bruges i dette kredsløb. I afsnit 7.3 laves multiplikator og multiplikand om til addender, som lægges sammen ved brug af et csatræ, som konstrueres i 7.4. Til sidst konstrueres en csadder (”carry-save adder”) i afsnit 7.5.

## 7.1 Faktorial-kredsløbet

Hovedprogrammet beregner faktorialfunktionen iterativt, og kredsløbet er som følger



Dette kredsløb kan modelleres lige ud ad landevejen med programmet

```
main ()
  1'10' counter'ff 10 1' 0'1' >> inc'add 10' >> counter _'1';
  counter result'ff 100 1' >> multiply'10 100' >> result
end
```

`0` i første linie er inddata til `cIn` for adderen `inc`, og `_` er tilsvarende uddata fra `cOut`.

Ovenstående program er lavet ud fra kredsløbsdiagrammet og afspejler meget godt strukturen af dette. Det var simpelt at lave, og signalvejene er rimelig nemme at følge i programmet.

## 7.2 Multiplikationsbyggeklodsen

Selve multiplikationen består af tre skridt. Først genereres addenderne ved for hver bit i multiplikatoren enten at sende 0 igennem eller multiplikanden bitskiftet passende i forhold til positionen (se næste afsnit). Dernæst bruges et csatræ til at reducere antallet af partielle summer, indtil der kun er to tilbage, og disse lægges sammen med en almindelig adder. I figur 7.1 illustreres csatræet og den afsluttende adder. Addenderne, csatræet og csadderne konstrueres i de følgende afsnit. Kredsløbet for multiplikatoren kan laves i Hades som

```
multiply (multiplier h multiplicand w, h w) >> (product w)
  multiplier multiplicand >> addends'h w' >> csatree'h w' 0'1'
    >> adder'w' >> product _'1'  // 0>>[cIn]adder, adder[cOut]>>afloeb
end
```

Sætningen i dette program følger strukturen i ovenstående beskrivelse nært, og byggeklodserne mellem hvert par af `>>` angiver et af de tre skridt. I `addends >> csatree 0` forbindes `addends` kun med `csatree`, da konstanter har en tom indbitvektor.

## 7.3 Addenderne

Addenderne produceres af byggeklodsen `addends`. Denne tager multiplikator og multiplikand til en multiplikation og laver dem om til lige så mange addender, som der er bit i multiplikatoren. Værdien af den $j$'te addend er

| | |
|---|---|
| 0 | hvis bit $j$ i multiplikator er 0 |
| multiplikand<<$j$ | hvis bit $j$ i multiplikator er 1 |

hvor multiplikand<<$j$ betyder, at multiplikand venstreskiftes $j$ gange. Til at vælge mellem de to værdier bruges en vektor af muxe, hvor der er en mux for hver bit i multiplikator. Multiplikanden forbindes nu ved at bruge muligheden for at lave interne forbindelser, da variablen `index` så kan bruges til at vælge de korrekte bit i multiplikanden. Dette programmeres som følger

```
addends (multiplier h multiplicand w, h w) >> (addends h*w)
  muxarray'h 1 w ; 0 multiplicand[index..] >> in';  // foranstil 0er
  multiplier >> muxarray >> addends
end
```

I ovenstående program er `in` forbundet for hver mux i `muxarray` og indgår derfor ikke i indbitvektoren for vektoren. Den indeholder altså kun bittene i `select`, og disse forbindes med `multiplier` (multiplikator).

## 7.4 Csatræ

Csatræet konstrueres ved først at lave et csatræniveau og dernæst at samle csatræniveauer til et træ. Et csatræniveau kan programmeres ved

```
CSATreeLevel (in nr*wordSz, nr wordSz)
          >> (out (2*(nr/3)+(nr mod 3))*wordSz)
  CSAddarray'nr/3 wordSz';
  if nr mod 3 = 0 then in >> CSAddarray >> out
  else in >> CSAddarray '(nr mod 3)*wordSz' >> out
end
```

Indstikket `in` modtager `nr` addender hver bestående af `wordSz` bit. Addenderne skal forbindes med indstikkene i et antal csaddere (se næste afsnit), og udbittene fra disse ledes ud af træniveauet. Addenderne forbindes tre ad gangen til en csadder, så hvis antallet af addender ikke kan divideres med tre, bliver der en eller to addender i "overskud". Bittene i disse føres direkte til uddata.[1]

Strengt taget svarer `CSATreeLevel` som defineret ovenfor ikke helt til et niveau i figur 7.1 på side 42, da overskydende bit i et niveau i førstnævnte altid er de sidste bit i indbitvektoren, mens de i sidstnævnte kan være de første eller sidste bit. Dette giver dog kun den forskel, at længste signalvej i sidstnævnte i nogle tilfælde indeholder lidt færre CSAddere end i førstnævnte (betragt f.eks. et csatræ med $5$ addender ind).

Nu kan csatræet konstrueres ved at sætte csatræniveauer sammen. Definitionen kan laves ved

```
CSATree (addends args*w, args w) >> (sums 2*w)
  if args = 2 then addends >> sums
  else addends >> CSATreeLevel'args w'
         >> CSATree'2*(nr/3)+(nr mod 3) w' >> sums
end
```

Csatræet modtager `args` addender, der er `w` bit brede. Træet opbygges ved hjælp af rekursion. Først ledes indbittene ind i et csatræniveau og herefter videre ind i næste instans af csatræklodsen, som er mindre end den foregående. Rekursionen stopper, når der kun er to addender tilbage.

## 7.5 CSAdder

I csatræet bruges en csadder. Denne opbygges som

```
CSAdder (inA inB inC sz, sz) >> (sumA sumB sz)
  inA inB inC >> /(sz)xorarray'sz 3';
  inA inB >> /(sz)and01'andarray sz 2';
  inA inC >> /(sz)and02'andarray sz 2';
  inB inC >> /(sz)and12'andarray sz 2';
  and01 and02 and12 >> 0'1' /(sz)orarray'sz 3'[0..sz-2] >> sumB _'1';
  xor >> sumA
end
```

`xorarray` består af `sz` 3-bit xorporte. `/(sz)` angiver, at indbitvektoren for disse består af bit $0$ fra hver port efterfulgt af bit $1$ fra hver port og til sidst bit $2$ fra hver port. Altså svarer

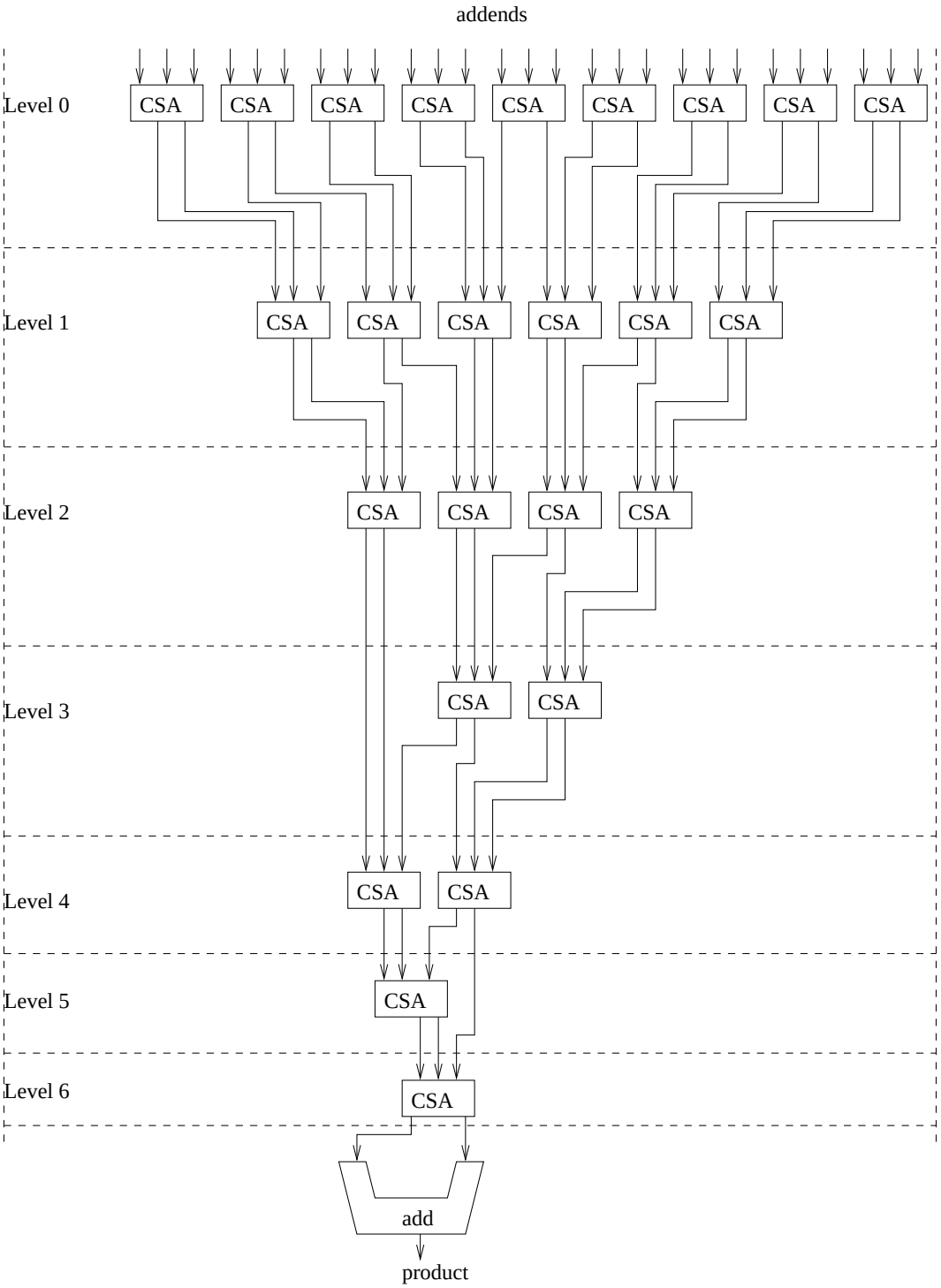
[1] Hvis størrelsen $0$ tillades for et gennemløbsudtryk, kan programmet oplagt simplificeres. Dette vil være en hensigtsmæssig forbedring af Hades.

`/(sz)xorarray`

til

`[0,3,6,...,3*sz-3,1,4,...,3*sz-2,2,5,...,3*sz-1]`

De resterende forbindelser af vektorer fungerer tilsvarende.




Figur 7.1: csatræ og afsluttende adder.

# Kapitel 8

# Kontrol af design

I dette kapitel argumenteres så vidt muligt for, at målene for Hades (kapitel 4) er opfyldt med nærværende design. Målene er af meget varierende beskaffenhed, lige fra meget konkret mål (det skal være muligt at lave vektorer med interne forbindelser) til uhåndgribelige mål (Hades skal være enkelt), der kræver en subjektiv vurdering. Denne vurdering bør laves af studerende fra målgruppen. Det har dog ikke vist sig muligt at lave en brugerafprøvning, så jeg vil argumentere løst for, at også disse mål er opfyldt.

Ved læsning af dette kapitel er det hensigtsmæssigt, at have SimSysmanualen [And01] ved hånden.

Hvert mål har fået sit eget afsnit, så det i 8.1 verificeres, at der i Hades kan laves modeller for simulationer svarende til Dat1E-niveau. I afsnit 8.2 argumenteres kort for, at hver komponent i Hades svarer til en bestemt komponent eller til en vektor af bestemte komponenter i SimSys, så køretidssystemet kan bruges sammen med Hades. I afsnit 8.3 diskuteres vektorer med interne forbindelser og træer, og afsnit 8.4 er om enkelhed af sproget. Til sidst verificeres i afsnit 8.5, at de nuværende problemer med SimSys undgås med nærværende design af Hades.

## 8.1 Simulationer på Dat1E-niveau

**Rapportopgaverne**

De vejledende løsninger til G- og K1-opgaven i 1999 (svarer nogenlunde til G2- og K1-opgaven i 2001) er i bilag B, og Hadesprogrammerne for disse er ligeledes vedlagt rapporten i bilag C.

Det springer umiddelbart i øjnene, at SimSysløsningerne fylder 23 sider, mens Hadesløsningerne kun fylder 3. Disse tal kan dog ikke sammenlignes direkte, da der i SimSysprogrammerne er ca. 3 siders kommentarer og 4 siders specifikation af elementer, der ikke er medtaget i Hades. Der er altså ca. 14 siders "reel" SimSyskode. I Hadesprogrammerne er der ingen kommentarer. Programmerne fylder altså mere end 4 gange så meget i SimSys, som de gør i Hades.

Hovedprogrammerne for de to løsninger i Hades er programmeret ud fra de tilsvarende hovedprogrammer i SimSys. De vil kunne laves noget mere læselige ud fra et kredsløbsdiagram, da programmerne så kan opbygges ud fra strukturen af kredsløbet og hermed bedre afspejle denne. Betragtes Hadesprogrammerne for byggeklodserne `shifter`, `relop`, `alu` og `bypass`, der er lavet ud fra kredsløbsdiagrammerne for komponenter, er koden for disse da også meget mere overskuelig, og signalvejene er rimelig nemme at aflæse.

Især har `shifter` (se side 72 og 93) kunnet konstrueres elegant ved hjælp af vektorer med interne forbindelser.

### Laboratorieøvelserne

Løsningerne til disse er i bilag D.

### Manualen

I dette afsnit vil jeg argumentere uformelt for, at de kredsløb, der kan udtrykkes i SimSys, også kan udtrykkes i Hades. Nedenfor konstrueres for de forskellige konstruktioner og programeksempler i [And01] tilsvarende konstruktioner og programmer i Hades. Ved hver del af beskrivelsen er det relevante afsnit eller kapitel i SimSysmanualen angivet.

**kap. 1** `main () flipflip >> not >> flipflip end`

**afs. 2.1-2.3** Et program består af et hovedprogram på formen

```
main (parametre) krop end
```

samt eventuelt af definitioner af sammensatte byggeklodser på formen

```
kompnavn (indstik...indstik stikstr ...,parametre)
      >> (udstik...udstik stikstr ...)
  krop end
```

**afs. 2.4-2.5.1** To stik kan forbindes ved `komp0[stik0] >> [stik1]komp1`. Der kan laves afledninger på stikket på venstresiden ved `(komp0[stik0])udafledning` og på stikket på højresiden ved `indafledning([stik1]komp1)`. De enkelte SimSysafledninger kan konstrueres som følger i Hades:

**subset(int bit0,int lgd)** `[bit0..bit0+lgd-1]`.

**subset(int bit0,int lgd,int felter)** `/felter[bit0..bit0+lgd-1]`.

**operator[](int k)** `[k]`.

**split(int elm,int elmStr)** Er ikke relevant i Hades, da der ikke findes vektorer af stik i dette sprog (fungerer som almindelige stik).

**reverse()** `[bitN..bit0]`, hvor `bit0` er første og `bitN` sidste bit i stikket.

**reverse(int felter)** `/felter[bitN..bit0]`, hvor `bit0` er første og `bitN` sidste bit i et felt.

**repeat(int kopier)** `*kopier`.

**afs. 2.5.2** SimSysoperationen `operator+` til at samle flere stik laves i Hades som følger: Hvis stikkene kommer fra samme komponent, kan de vælges med `[bitmønster,...,bitmønster]`, hvor hvert `bitmønster` angiver bittene for de enkelte stik. Kommer stikkene fra forskellige byggeklodser, samles disse til en bitvektor, der omgives med parenteser. Af denne bitvektor udtages på samme måde som ovenfor de relevante bit.

**afs. 2.5.3** Hades har ikke vektorer af stik, men derfor kan det godt være relevant at kunne lave tilsvarende operationer på f.eks. stikket `in` i en mux. Afledninger på vektorer af stik kan konstrueres som følger i Hades (hvor vektoren af stik udgør en bitvektor):

**operator[](int k)** `[bit0..bitN]`, hvor `bit0` er første og `bitN` sidste bit i stik `k`.

**transpose()** `/antal`, hvor `antal` er antal stik i vektoren.

**afs. 2.6** Hades indeholder både `pipeStageBoundary` og `ZapBoundary`. Erklæring og brug af skel kan f.eks. ske ved

```
komp >> ZAP'mux[0]' >> PIPE'pipestage mux[1]' >> komp2
```

**afs. 2.7** Konstanter og afløb skrives `værdi'str'` henholdsvis `_'str'` eller blot `værdi` henholdsvis `_`, hvis størrelserne kan udledes.

**afs. 2.8** Vektorerne i afsnittet kan erklæres `notarray'100'` og `andarray'100 102-index'` i Hades.

**afs. 2.9** I Hadesprogrammet for multiplikatorkredsløbet i afsnittet har jeg valgt at lave en særskilt komponent for multiplikationsdelen for at gøre programmet mere overskueligt.

```
main (height width)
  counter'ff height 1' 1'height' >> inc'serialadder height'
    >> counter;
  counter result'ff width 1' >> minMultiply'height width' >> result
end

minMultiply (multiplier h multiplicand w, h w) >> (product w)
  muxarray'h 1 w-index ; 0 multiplicand[index..] >> in' ;
  serialadderarray'h-1 w ; 0 >> [..index]inB ; out >> inA' ;
  multiplier >> muxarray 0'w'  // kun select uforbundet
    >> serialadderarray        // foerste inA, noget inB, cIn
    >> product _'w-2'          // sum, cOut
end
```

**kap. 3** De to eksempler på sammensatte byggeklodser i kapitlet vises i Hades nedenfor og er tilstrækkeligt til at illustrere, at sammensatte byggeklodser kan laves i dette sprog.

```
nand (in w, w) >> (out 1)
  in >> and'w' >> not >> out
end

flipflop2 (in sz, sz) >> (out sz)
  ff'1 0' >> not >> ff , [wrenable]wrenff'sz 0';
  in >> [in]wrenff >> out
end
```

**kap. 5** Dette eksempel er allerede lavet i kapitel 7.

**kap. 8** De fleste indbyggede komponenter i SimSys er også indbyggede i Hades med samme funktionalitet (se afsnit 6.7). De resterende konstrueres her.

**AndArray, NandArray, OrArray, NorArray, XorArray** `AndArray` kan f.eks. erklæres ved `andarray'gates width'`. De resterende vektorer erklæres tilsvarende.

**BWAndArray, BWNandArray, BWOrArray, BWNorArray, BWXorArray** `BWAndArray` med `in[width,gates]` erklæres og bruges ved `/(gates)andarray'width gates'`. De resterende vektorer erklæres og bruges tilsvarende.

**BWNotArray** Erklæres ved `notarray'gates'`.

**CLAdder** Ikke medtaget, da den er identisk med `Adder`.

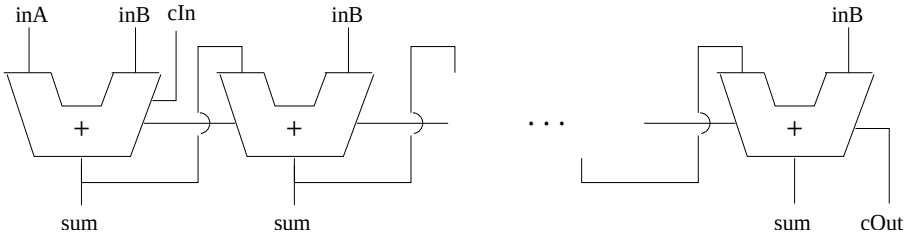
**NotArray** Erklæres ved `notarray'gates'`.

## 8.2 Hades og køretidssystemet

Da Hades består af byggeklodser på samme måde som SimSys, er det oplagt, at det er muligt at bruge køretidssystemet for SimSys.

## 8.3 Vektorer og træer

Af grammatikken fremgår det, at der kan laves vektorer med interne forbindelser. Eksempelvis kan vektoren




programmeres i Hades som

```
addarray'antal str ; cout >> cin ; sum >> inA ; sum >> {o}'
```

Det er muligt at konstruere træniveauer i Hades ved at bruge en vektor i definitionen for den sammensatte byggeklods for træniveauet. Træer kan nu konstrueres ved brug af rekursion: indbit til træet ledes ind i et træniveau og herfra videre ind i en mindre instans af træet (eller der stoppes, hvis der er et passende antal udbit tilbage). Eksempelvis kan et csatræniveau og et csatræ programmeres som

```
CSATreeLevel (in nr*wordSz, nr wordSz)
          >> (out (2*(nr/3)+(nr mod 3))*wordSz)
  CSAddarray'nr/3 wordSz';
  if nr mod 3 = 0 then in >> CSAddarray >> out
  else in >> CSAddarray '(nr mod 3)*wordSz' >> out
end
```

og

```
CSATree (addends args*w, args w) >> (sums 2*w)
  if args = 2 then addends >> sums
  else addends >> CSATreeLevel'args w'
         >> CSATree'2*(nr/3)+(nr mod 3) w' >> sums
end
```

Disse programmer(stumper) er både rimelig kompakte og rimelig nemme at læse og giver altså en fornemmelse af, at det er relativt nemt at konstruere træer i Hades (selv om det ikke er et bevis).

## 8.4 Enkelhed

Dette mål er svært at verificere uden en decideret brugerafprøvning, men i dette afsnit vises ved eksempler, at (designet for) Hades giver noget enklere programmer end SimSys. Ud fra dette argumenteres løst for, at målet om enkelhed af sproget er opfyldt.

Lad os først betragte kredsløbet, hvor uddata for en flipflop ledes ind i en notport og herfra tilbage i flipfloppen. I SimSys udtrykkes dette program ved

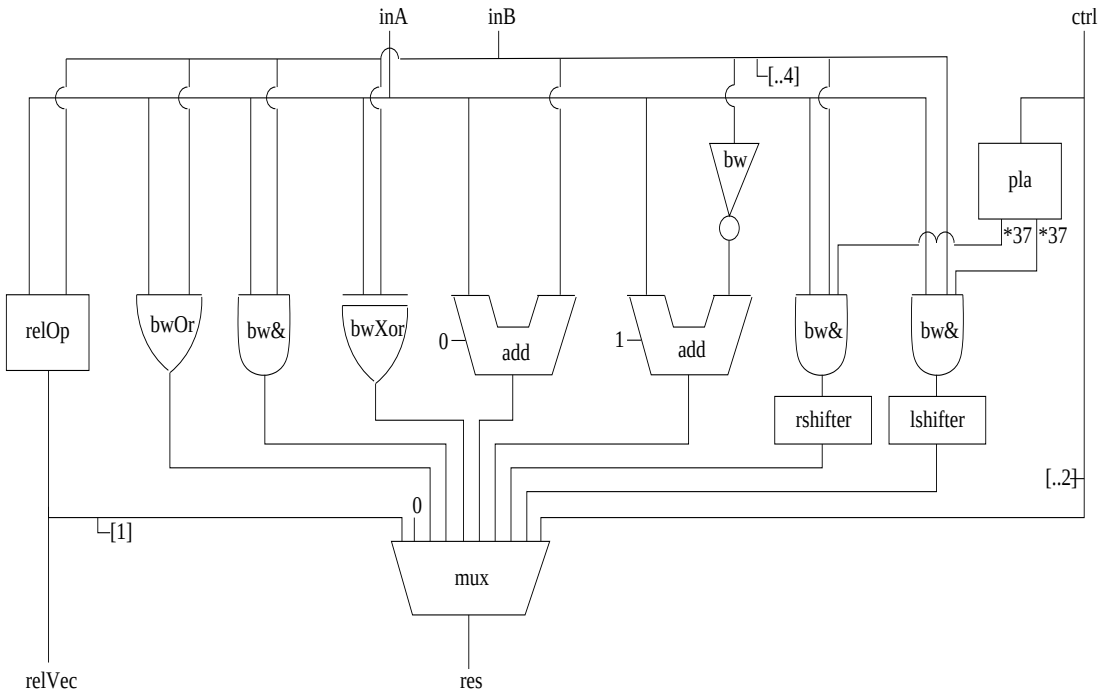
```
#include "simlib.h"

class MySimApp : public SimApp {
public:
  virtual void BuildModel();
  MySimApp() {}
};

SimApp* SimApp::mk(int argc, char *argv[], char *envp[])
{ return MySimApp; };

void MySimApp::BuildModel() {
  D(FlipFlop,myFlipFlop,1,0);
  D(NotGate,myNotGate);
  myFlipFlop.out >> myNotGate.in;
  myNotGate.out >> myFlipFlop.in;
};
```

Der skal en del ekstra kode til grundet brugen af C++. I modsætning til dette er der i Hades ikke en masse overflødig syntaks, og ovenstående program kan skrives som

```
main ()
  myFlipFlop >> myNotGate >> myFlipFlop
end
```

I dette program er koden intuitiv og enkel, men det illustrerede kredsløb er også mindre, end et kredsløb normalt vil være. Desuden består et væsentligt element i en model for et kredsløb af at definere sammensatte byggeklodser.

Betragt derfor følgende kredsløb for en 32-bit alu:




I SimSys skal der laves både en erklæring (lægges typisk for sig selv i en `.h`-fil) og en definition for en byggeklods. Hvis det antages, at byggeklodserne (typerne) i kredsløbet er defineret, kan ovenstående udtrykkes ved

```
#include "interface.h"

struct ALU {
  INABSTRACTION& inA;     // [32]
  INABSTRACTION& inB;     // [32]
  INABSTRACTION& ctrl;    // [4]
  OUTABSTRACTION& res;    // [32]
  OUTABSTRACTION& relVec; // [10]
  ALU(INABSTRACTION& inA,
      INABSTRACTION& inB,
      INABSTRACTION& ctrl,
      OUTABSTRACTION& res,
      OUTABSTRACTION& relVec)
    : inA(inA), inB(inB), ctrl(ctrl), res(res), relVec(relVec) {};
  static ALU& mk(String home);
};

#endif

#include "composite.h"

ALU& ALU::mk(String home) {
  D(INPAD,ctrl,4);
  D(INPAD,inA,32);
  D(INPAD,inB,32);
  D(OUTPAD,res,32);
  D(OUTPAD,relVec,10);

  // build add/sub path
  D(Mux,resMux,3,32);
  D(BWNotArray,negator,32);
  D(Adder,adder,32);
  D(Adder,suber,32);
  inA >> adder.inA >> suber.inA;
  inB >> adder.inB >> negator.in[0]; negator.out >> suber.inB;
  "0 [1]" >> adder.cIn;
  "1 [1]" >> suber.cIn;
  adder.sum >> resMux.in[4];
  suber.sum >> resMux.in[5];
  ctrl.subset(0,3) >> resMux.select;
  adder.cOut >> 0;
  suber.cOut >> 0;

  // build slt path
  D(RelOp,relOp);
  inA >> relOp.inA;
  inB >> relOp.inB;
  relOp.relVec >> relVec;
  relOp.relVec[1] >> resMux.in[0].subset(0,1);
  Const("0",31)   >> resMux.in[0].subset(1,31);

  // build And, Or and Xor paths
  D(BWAndArray,and,32,2);
```

```
  D(BWOrArray,or,32,2);
  D(BWXorArray,xor,32,2);
  or.out  >> resMux.in[1];
  and.out >> resMux.in[2];
  xor.out >> resMux.in[3];
  inA >> and.in[0]
      >> or.in[0]
      >> xor.in[0];
  inB >> and.in[1]
      >> or.in[1]
      >> xor.in[1];

  // build lshift and rshift paths
  D(Shifter,lshft,5,Shifter::LEFT);
  D(Shifter,rshft,5,Shifter::RIGHT);
  static char* shftSaverPlane[] = { "0110 1-",
                                    "0111 -1" };
  D(PLA,shftSaver,4,2,2,shftSaverPlane);
  ctrl >> shftSaver.in;
  D(BWAndArray,enableLshft,37,2);
  D(BWAndArray,enableRshft,37,2);
  shftSaver.out[0].repeat(37) >> enableRshft.in[0];
  shftSaver.out[1].repeat(37) >> enableLshft.in[0];
  inA >> enableRshft.in[1].subset(0,32)
      >> enableLshft.in[1].subset(0,32);
  inB.subset(0,5) >> enableRshft.in[1].subset(32,5)
                  >> enableLshft.in[1].subset(32,5);
  enableRshft.out.subset(0,32) >> rshft.in;
  enableLshft.out.subset(0,32) >> lshft.in;
  enableRshft.out.subset(32,5) >> rshft.amount;
  enableLshft.out.subset(32,5) >> lshft.amount;
  lshft.out >> resMux.in[6];
  rshft.out >> resMux.in[7];

  // result
  resMux.out >> res;

  // get out of here
  return *new ALU(inA.outside(), inB.outside(),
                  ctrl.outside(), res.outside(), relVec.outside());
};
```

Dette laves noget simplere i Hades ved

```
alu (inA inB 32 ctrl 4) >> (res 32 relvec 10)
  shftsaverPlane = " 0110 1-,0111 -1 ";
  ctrl >> (shftsaver'shftsaverPlane'[1]*37+[0]*37
           (inA inB[..4])*2)/2[..36;37..]
    >> /37enableLshft'andarray 37 2' /37enableRshft'andarray 37 2'
    >> lshifter'5 true' rshifter'5 false'
    >> [6*32..8*32-1]resmux;
  inA inB >> relop*1+[1] 0'31' ctrl >> relvec [..31,select]resmux'3 32';
  inA inB >> /2orarray'32 2' , /2andarray'32 2' , /2xorarray'32 2'
    , [..63]add[out] , ('32' notarray'32' >> [..63]sub'add')
```

```
    >> [32..6*32-1]resmux >> res;
  0'1' 1'1' >> [cIn]add[cOut] [cIn]sub[cOut] >> _
end
```

Ovenstående program er nogenlunde kompakt, men stadig simpelt. Desuden er det rimelig nemt at konstruere ud fra kredsløbsdiagrammet.

## 8.5 Undgå nuværende problemer

**Ingen C++-kode**

At programmere i Hades kræver oplagt intet kendskab til C++, da sprogets er design ikke hviler på C++.

**Signalveje**

I programeksemplet med aluen i afsnit 8.4 er det rimelig nemt at følge signalvejen igennem kredsløbet. Også programmet i kapitel 7 taler for, at signalvejene i et kredsløb er nemme at følge i det tilsvarende program, hvis dette er fremstillet ud fra kredsløbsdiagrammet.

**Fejlrejsning**

Alle syntaktiske fejl fanges af Hadesoversætteren, så det skal blot kontrolleres, at de relevante semantiske fejl (ikke fejl fra f.eks. køretidssystemet), der fanges i SimSys (altså i programmet for klassebiblioteket) fanges allerede i Hadesoversætteren. Jeg har derfor gennemset kildekoden for SimSysoversætteren og fundet, at der rejses nedenstående fejl. Fejl af samme type er slået sammen, mens fejl i konstruktioner, der ikke indgår i Hades, er udeladt (f.eks. fejl vedrørende vektorer af stik).

- Stik, planer eller komponenter med samme navn.
- Kreds uden tilstandselement (*eng: building block with state*).
- Komponent ikke defineret.
- Stik med specificeret navn eksisterer ikke for komponent.
- Bit i et stik ikke forbundet.
- Bit i indstik forbundet mere end en gang.
- Bit i afledning for stik er udenfor stikket.
- Bit i `subset(fst,len,fields)` er udenfor et felt.
- Afledning med 0 bit.
- Stik med forskellig størrelse forbindes.
- Størrelse for stik $\leq 0$.
- Feltstørrelsen i `subset(fst,len,fields)` eller `reverse(fields)` $\leq 0$.

- Feltstørrelsen i `subset(fst,len,fields)` eller `reverse(fields)` går ikke op i den totale størrelse.
- `antal` $< 1$ i `repeat(antal)`.

Som det fremgår af kapitel 6 indgår alle disse fejl i semantikken for Hades, og de vil derfor ikke optræde i det resulterende SimSysprogram.

# Kapitel 9

# Implementering

Hades er som nævnt i indledningen lavet som en forende til SimSys, da dette giver mulighed for at genbruge både simuleringsmotoren og køretidssystemet. I implementeringen oversættes Hades til en intern repræsentation (i resten af dette kapitel skrives mellemkode i stedet for intern repræsentation), der herefter oversættes til SimSys. Dette sikrer, at der kan ændres i Hades eller i SimSys, uden at hele forenden skal revideres.

Lexeren og parseren er fuldstændige, men af tidsmæssige grunde er nogle få dele af sproget ikke behandlet i resten af oversætteren. Programmet er opbygget modulært for at gøre det nemt at ændre og udvide dele af det. Specielt er der taget hensyn til, at de manglende konstruktioner nemt skal kunne indsættes. Alle strukturer tilgås desuden via en grænseflade, så indholdet kan udskiftes med en hurtigere datastruktur uden at ændre i resten af programmet. Generelt går oversættelse af programmer dog rimelig hurtigt på Dikus maskiner.

Formålet med dette kapitel er at give et overblik over oversætteren, så i afsnit 9.1 beskrives oversættelsen til mellemkode, mellemkoden og ganske kort oversættelsen til SimSys. Herefter beskrives programmet og programmets opbygning i afsnit 9.2, og til sidst gives i afsnit 9.3 en kort gennemgang af, hvad der mangler at blive implementeret.

## 9.1 Oversættelsen

Et Hadesprogram består af et hovedprogram og eventuelt af en eller flere brugerdefinerede byggeklodser. Når et program oversættes, angives parametrene til hovedprogrammet. Derfor kendes alle værdier i et program på oversættelsestidspunktet. Alle fejl i et Hadesprogram fanges i oversættelsen fra Hades til mellemkode.

Oversætteren starter med at læse alle erklæringer af sammensatte byggeklodser og gemme værdierne for disse. Disse skal bruges, når typeerklæringer for instanser af byggeklodser i kroppene skal læses. Samtidig erstattes eventuelle bitvariable i kroppen for en programblok med deres værdi, og også plandefinitioner fjernes fra kroppen, der nu kun indeholder specifikation af kredsløbet i programblokken. Denne gemmes sammen med værdierne af eventuelle plandefinitioner.

Herefter oversættes selve hovedprogrammet. Dette gøres ved, at alle parametre for programmet indsættes i kroppen for programblokken, og i eventuelle betingede sætninger elimineres den del, der ikke skal bruges. Nu indsamles typeerklæringerne for alle instanser af byggeklodser, der bruges i kroppen, samt størrelserne for alle konstanter, afløb og gennemløb. For vektorer af byggeklodser laves en instans for hver byggeklods i vektoren. Hver bitvektor deles herefter i forbindelser, hvor der kun

indgår en udbitvektor og en indbitvektor. Udbitvektoren indeholder en konstant, en afledning af et indstik for programblokken eller en afledning for et enkelt stik ud af en (instans af) byggeklods. Tilsvarende indeholder indbitvektoren et afløb, en afledning af et udstik for programblokken eller en afledning for et enkelt stik ind i en byggeklods. Størrelsen af konstanter og afløb indgår, hvor de bruges. Gennemløb fjernes, så forbindelser via gennemløb foregår direkte.

Mellemkoden for hovedprogrammet består af typeerklæringerne for alle instanser af byggeklodser, der bruges i kroppen, af eventuelle definitioner af planer og af alle de ovennævnte forbindelser. Dette kan nemt oversættes til SimSys, hvor hver byggeklods skal instantieres og alle planer defineres inden brug, og hvor forbindelser kun kan laves mellem enkelte stik.

Hvis der i kroppen for hovedprogrammet bruges instanser af sammensatte (brugerdefinerede) byggeklodser, genereres der mellemkode for hvert forskelligt sæt værdier for hver af disse på samme måde som mellemkoden blev frembragt for kroppen for selve hovedprogrammet. Blot gemmes også stikkene ind i og ud af disse programblokke. En sammensat byggeklods kan altså godt blive oversat til flere forskellige sammensatte byggeklodser, der alle gives et entydigt navn. De enkelte byggeklodser har ikke længere parametre, da alle værdier for parametre er indsat i kroppen under oversættelsen til mellemkode.

Hvis der i kroppene for nogen af disse byggeklodser bruges sammensatte byggeklodser med værdisæt, som ikke er konstrueret endnu, laves der mellemkode for disse. Dette fortsætter, til der er genereret mellemkode for alle forskellige sæt værdier for alle forskellige typer sammensatte byggeklodser, som der bruges i programmet. Bruges en byggeklods ikke, genereres der altså ikke mellemkode for den.

*Antag brugeren har defineret en byggeklods* `minmux`*, der tager en parameter, som skal være et heltal. Hvis der i et program bruges tre instanser af* `minmux` *med værdien* `3`*,* `4` *og* `3`*, vil der i mellemkoden blive genereret to byggeklodser, hvor den ene indeholder mellemkoden for* `minmux`*, hvor parameteren er* 3*, og den anden mellemkoden for* `minmux` *med parameteren* 4*.*

Mellemkoden for en (instans af) en sammensat byggeklods består af dennes navn (der genereres et entydigt navn for hver sammensat byggeklods), type og værdier, af stikkene og af mellemkoden for kroppen, der er som den for hovedprogrammet. Type og værdier bruges til opslag, når der skal laves en typeerklæring for en komponent i en krop.

I et SimSysprogram består en sammensat byggeklods af en erklæring af grænsefladen for denne (dvs. stikkene ind og ud), samt af en definition, hvor stik, planer og instanser af byggeklodser erklæres og herefter sættes sammen ved forbindelser mellem enkelte stik. Det er derfor nemt at oversætte mellemkoden til et SimSysprogram.

## 9.2 Programmets opbygning

Forenden er modulært opbygget, og hver del af programmet er placeret i en fil. I dette afsnit beskrives disse filer. Afsnittene har navn efter de(n) fil, de beskriver (eksklusive endelserne `.sml` og `.sig`).

### main

Formålet med hovedprogrammet er at læse filnavnene for filerne indeholdende Hadesprogrammet samt værdierne for eventuelle parametre til hovedprogrammet fra kommandolinien og sørge for, at disse laves om til et passende syntakstræ, der gives videre til oversætteren. Desuden rejses her passende fejlmeddelelser, hvis der opstår fejl under oversættelsen.

**oversaetter**

Sørger for at oversætte først til mellemkode og herfra til SimSys. Uddata skrives i filen `main.cc`.

**typ**

Læser alle erklæringer for sammensatte byggeklodser i programmet og gemmer de data, der skal bruges, når der stødes på typeerklæringer i kroppen for en programblok. Desuden substitueres bitvariable med deres værdi, og planer gemmes i mellemkoden.

**mellemkode**

Parametrene til hovedprogrammet indsættes, og der genereres mellemkode for kroppen af denne programblok. Herefter genereres mellemkode for alle instanser af sammensatte komponenter, der bruges i hovedprogrammet, så af alle instanser, der bruges i disse instanser osv.. Der genereres dog kun en instans for hver brug af en byggeklods med et bestemt sæt værdier for parametrene.

**typer**

Her indsamles udtryk for typer samt værdier for parametre for alle instanser af komponenter og udtryk for størrelser af alle afløb, konstanter og gennemløbsudtryk.

**stoerrelser**

Udtrykkene for størrelser og værdier til parametre omregnes til heltal eller bool, og hvis typen for en instans ikke er angivet, udregnes denne. For vektorbyggeklodser konstrueres instanser for hver komponent i klodsen.

**afled**

Afledninger ændres til en form, hvoraf det præcist fremgår, hvilke bit afledningen vedrører, og bitmønstrene splittes, så hver del kun vedrører bit fra et stik.

**split**

Forbindelserne i kroppen splittes i forbindelser, der hver angiver en forbindelse mellem (dele af) et indstik og (dele af) et udstik.

**simsys**

Mellemkoden oversættes til et SimSysprogram, der gemmes i filen `main.cc`.

**udtryk, plan, bitvektor, liste, fejl, syntaks, konst, fkt**

Herudover findes diverse filer med funktioner til håndtering af udtryk, planer mm.. Formålet fremgår af navnene. `fkt` indeholder diverse generelle funktioner.

## 9.3 Mangler i oversætteren

Indenfor den givne tidsramme har det ikke været muligt at nå at implementere alle dele af Hades. Ved prioriteringen er udeladt to dele af sproget, der ikke er strengt nødvendige for at opnå den fulde funktionalitet, men som blot gør sproget nemmere at arbejde med. Dermed kan der stadig skrives fungerende programmer i Hades. Desuden er byggeklodserne `CSATree`, `CSATreeLevel`, `IdealMemIOSys` og `MemIOSys` endnu ikke inkluderet. De to sidstnævnte er inkluderet i byggeklodsbiblioteket, men der mangler mulighed for at læse parametre, der er strenge (altså ikke er tal eller bool), da disse to byggeklodser har en sådan parameter. Desuden fanges det ikke af Hadesoversætteren, hvis der i en model af et kredsløb optræder en kreds uden tilstandselementer.

De udeladte konstruktioner er inferens af størrelser samt mulighed for at definere interne forbindelser i vektorer. Førstnævnte kan oplagt undværes, da størrelser blot kan angives. Sidstnævnte kan undværes, da sådanne vektorer kan konstrueres (dog med noget mere besvær) som en sammensat komponent ved brug af rekursion. Eksempelvis kan

```
myaddarray'32 32 ; cOut >> cIn ; 0 mux[index+1..] >> inA'
```

udtrykkes ved

```
myaddarray (inA antal*32 cIn 1 mux antal, antal) >> (out antal*32 cOut 1)
  if antal = 1 then
    inA 0'32' cIn >> add'32' >> out cOut; mux >> _
  else
    inA[..31] 0'32-antal' mux[1..] cIn >> add'32'[sum] >> [..31]out;
    inA[32..] add[cOut] mux[1..] >> myaddarray'antal-1'; mux[0] >> _'1'
  ;
end
```

hvor `myaddarray` kaldes med `antal` $= 32$. Konstruktionen bør dog tilføjes implementeringen inden brug, da ovenstående er uelegant.

# Kapitel 10

# Afprøvning

I dette kapitel dokumenteres en afprøvning af Hadesoversætteren. Den er tilrettelagt ud fra grammatikken og semantikken for Hades som beskrevet i kapitel 6 og starter med at sandsynliggøre, at helt simple strukturer som udtryk virker. Herfra arbejdes opad indtil konstruktionen af et helt program.

Afprøvningen består af en række check af, at de forskellige konstruktioner virker efter hensigten, samt at der rejses korrekte fejl ved ulovlig brug. For hvert check er angivet, hvilket program der bruges til at kontrollere dette, og disse programmer samt uddata fra oversætteren (det kan være en fejlmeddelelse eller et SimSysprogram) ved kørsel af dem er vedlagt i bilag F. For hvert check er det desuden med + og ÷ angivet, om uddata bør være et fungerende program eller en fejlmeddelelse, og til sidst er det specificeret ved ”ok” eller ”fejl”, om uddata er som forventet. Hvis uddata ikke er korrekt, beskrives afvigelsen kort under tabellen. De ikke-implementerede dele af Hades er ikke medtaget i denne afprøvning. For en god ordens skyld oversætter og starter jeg de velfungerende (SimSys)programmer for at kontrollere, at der ikke gemmer sig uopdagede, syntaktiske fejl.

Det primære mål har været, at Hadesoversætteren kan oversætte korrekte Hadesprogrammer til fungerende SimSysprogrammer. Som det ses i kapitlet er dette stort set lykkedes, idet de eneste fejl der findes i oversætteren er, at der til tider rejses forkerte fejlmeddelelser, samt at syntaksen for den indbyggede komponent `sram` er ulovlig.

I afsnit 10.1 kontrolleres udtryk og i 10.2 type- og størrelseserklæringer for komponenter mm.. Herefter afprøves afledninger i afsnit 10.3 og så selve opbygningen af bitvektorer i afsnit 10.4. Til sidst følger så afprøvning af kroppe i afsnit 10.5 og af hele programmer i 10.6. I afsnit 10.7 opsummeres kort, hvilke fejl der blev fundet under afprøvningen.

## 10.1 Udtryk

**Heltalsudtryk**

| | | | |
|---|---|---|---|
| Prøv variabel. | F.1.1 | + | ok |
| Prøv tal. | F.1.1 | + | ok |
| Prøv hexadecimalt tal. | F.1.6 | + | ok |
| Prøv +. | F.1.2 | + | ok |
| Prøv -. | F.1.1 | + | ok |
| Prøv *. | F.1.1 | + | ok |
| Prøv /. | F.1.2 | + | ok |
| Prøv `mod`. | F.1.2 | + | ok |
| Prøv ^. | F.1.8 | + | ok |
| Prøv variabel, der ikke er erklæret. | F.2.1 | ÷ | ok |
| Prøv variabel med ulovlig type. | F.2.2 | ÷ | ok |

**Boolsk udtryk**

| | | | |
|---|---|---|---|
| Prøv variabel. | F.1.1 | + | ok |
| Prøv `true`. | F.1.5 | + | ok |
| Prøv `false`. | F.1.6 | + | ok |
| Prøv $<$. | F.1.3 | + | ok |
| Prøv $\leq$. | F.1.3 | + | ok |
| Prøv $=$. | F.1.3 | + | ok |
| Prøv $\geq$. | F.1.6 | + | ok |
| Prøv $>$. | F.1.3 | + | ok |
| Prøv variabel, der ikke er erklæret. | F.2.3 | ÷ | ok |
| Prøv variabel med ulovlig type. | F.2.4 | ÷ | ok |

## 10.2 Typer og størrelser

**Skel**

| | | | |
|---|---|---|---|
| Prøv alle skel (`pipestageboundary`, `zapboundary`). | F.1.6 | + | ok |
| Erklæring af skel: `'`*skelnavn komp* `[`*bit*`]'`. | F.1.6 | + | ok |
| Erklæring af skel: `'`*komp* `[`*bit*`]'`, hvor skel kan udledes af navnet. | F.1.6 | + | ok |
| Erklæring af skel: `'`*skelnavn komp* `[`*bit*`]'`, hvor *skelnavn* ikke er et eksisterende skel. | F.2.5 | ÷ | ok |
| Erklæring af skel: `'`*komp* `[`*bit*`]'`, hvor skel ikke kan udledes af navnet. | F.2.6 | ÷ | ok |
| Erklæring af skel, hvor *komp* ikke findes. | F.2.7 | ÷ | ok |
| Erklæring af skel, hvor `bit` er udenfor komponentens udbitvektor. | F.2.8 | ÷ | ok |

**Komponent**

| | | | |
|---|---|---|---|
| Prøv `sram` | F.1.11 | + | fejl |
| Prøv de resterende indbyggede komponenter (se afsnit 6.7). | alle | + | ok |
| Prøv sammensat byggeklods (med parametre). | F.1.1 | + | ok |
| Erklæring af komponent: *'type parametre'*. | F.1.2 | + | ok |
| Erklæring af komponent: *'parametre'*, hvor *type* kan udledes. | F.1.1 | + | ok |
| Erklæring af komponent: *'parametre'*, hvor *type* ikke kan udledes. | F.2.9 | ÷ | ok |
| Erklæring af komponent, hvor der angives en parameter for meget. | F.2.10 | ÷ | ok |
| Erklæring af komponent, hvor der angives en parameter for lidt. | F.2.11 | ÷ | ok |
| Tildel størrelsesvariabel værdi $0$. | F.2.12 | ÷ | ok(!) |
| Tildel størrelsesvariabel værdi $< 0$. | F.2.13 | ÷ | ok(!) |

Når et Hadesprogram, hvor komponenten `sram` bruges, oversættes til et SimSysprogram, sker dette uden fejl, men når SimSysprogrammet herefter oversættes, rejses fejlen „SRam' undeclared", som om denne komponent ikke kendes i SimSys. De to sidste programmer giver korrekte fejlmeddelelser, men fejlens position angives som $(0, 0)$.

**Vektor**

| | | | |
|---|---|---|---|
| Erklæring af vektor: *'type antal parametre'*. | F.1.4 | + | ok |
| Erklæring af vektor: *'antal parametre'*, hvor typen kan udledes. | F.1.1 | + | ok |
| Erklæring af vektor: *'antal parametre'*, hvor typen ikke kan udledes. | F.2.14 | ÷ | ok |
| Erklæring med en parameter for meget. | F.2.15 | ÷ | ok |
| Erklæring med en parameter for lidt. | F.2.16 | ÷ | ok |
| Tildel antal værdi $0$. | F.2.17 | ÷ | ok |
| Tildel antal værdi $< 0$. | F.2.18 | ÷ | ok |
| Tildel størrelsesvariabel værdi $0$. | F.2.19 | ÷ | ok(!) |
| Tildel størrelsesvariabel værdi $< 0$. | F.2.20 | ÷ | ok(!) |

De to sidste programmer giver korrekte fejlmeddelelser, men fejlens position angives som $(0, 0)$.

**Størrelse**

Vedrører størrelser for konstanter, afløb og gennemløbsudtryk.

| | | | |
|---|---|---|---|
| Prøv størrelse $> 0$. | F.1.1 | + | ok |
| Prøv `2'0'`. | F.2.21 | ÷ | ok |
| Prøv `_'0'`. | F.2.22 | ÷ | fejl |
| Prøv `'0'`. | F.2.23 | ÷ | fejl |

For ulovlig størrelse af afløb rejses en intern fejlmeddelelse, mens ulovlig størrelse af gennemløb giver fejlmeddelelsen "in-bit already connected".

## 10.3 Afledninger

**Bitmønster**

| | | | |
|---|---|---|---|
| Prøv bitmønster *bitmønster*`,`*bitmønster*. | F.1.7 | + | ok |
| Prøv bitmønster *stik*. | F.1.1 | + | ok |
| Prøv bitmønster *bit*. | F.1.7 | + | ok |
| Prøv bitmønster *bit0*`..`*bitN*, hvor *bit0* $<$ *bitN*. | F.1.4 | + | ok |
| Prøv bitmønster *bit0*`..`*bitN*, hvor *bit0* $=$ *bitN*. | F.1.4 | + | ok |
| Prøv bitmønster *bit0*`..`*bitN*, hvor *bit0* $>$ *bitN*. | F.1.3 | + | ok |
| Prøv bitmønster `..`*bit*. | F.1.4 | + | ok |
| Prøv bitmønster *bit*`...` | F.1.1 | + | ok |
| Prøv bitmønster, der går over mindst to stik. | F.1.1 | + | ok |
| Prøv bitmønster *stik*, hvor `stik` ikke findes for denne komponenttype. | F.2.24 | ÷ | ok |
| Prøv bitmønster *stik* på bitvektor, der ikke er en komponent. | F.2.25 | ÷ | ok |
| Prøv bitmønster, hvor en bit er $< 0$. | F.2.26 | ÷ | fejl |
| Prøv bitmønster, hvor en bit er $>$ sidste bit. | F.2.27 | ÷ | ok |

Hvis en bit i et bitmønster er $< 0$, gives fejlmeddelelsen, at bit 0 ikke er forbundet.

**Felter**

| | | | |
|---|---|---|---|
| Prøv *bitmønster*. | F.1.7 | + | ok |
| Prøv *bitmønster*`; ... ;`*bitmønster*. | F.1.8 | + | ok |
| Prøv felter, hvor en bit $>$ sidste bit i feltet. | F.2.28 | ÷ | ok |

**Indafledning**

| | | | |
|---|---|---|---|
| Prøv med ingen indafledning. | F.1.1 | + | ok |
| Prøv med `[]`. | F.1.4 | + | ok |
| Prøv *indafledning*+*indafledning*. | F.1.8 | + | ok |
| Prøv `[`*bitmønster*`]`*bitvektor*. | F.1.1 | + | ok |
| Prøv `/`*udtryk*`[`*felter*`]`*bitvektor*. | F.1.9 | + | ok |
| Prøv `/`*udtryk bitvektor*. | F.1.8 | + | ok |
| Prøv indafledning med `/`*udtryk*, hvor *udtryk* = 0. | F.2.29 | ÷ | fejl |
| Prøv indafledning med `/`*udtryk*, hvor *udtryk* $< 0$. | F.2.30 | ÷ | fejl |
| Prøv indafledning med `/`*udtryk*, hvor *udtryk* ikke går op i antal bit i indbitvektoren. | F.2.31 | ÷ | ok |

Hvis *udtryk* $\leq 0$ i `/`*udtryk*, rejses en intern fejl.

**Udafledning**

| | | | |
|---|---|---|---|
| Prøv med ingen udafledning. | F.1.1 | + | ok |
| Prøv med `[]`. | F.1.9 | + | ok |
| Prøv *udafledning+udafledning*. | F.1.7 | + | ok |
| Prøv *bitvektor*`[`*bitmønster*`]`. | F.1.3 | + | ok |
| Prøv *bitvektor*`/`*udtryk*`[`*felter*`]`. | F.1.7 | + | ok |
| Prøv *bitvektor*`/`*udtryk*. | F.1.3 | + | ok |
| Prøv *bitvektor*`*`*udtryk*. | F.1.1 | + | ok |
| Prøv udafledning med `/`*udtryk*, hvor *udtryk* = 0. | F.2.32 | ÷ | fejl |
| Prøv udafledning med `/`*udtryk*, hvor *udtryk* $< 0$. | F.2.33 | ÷ | fejl |
| Prøv udafledning med `*`*udtryk*, hvor *udtryk* = 0. | F.2.34 | ÷ | ok |
| Prøv udafledning med `*`*udtryk*, hvor *udtryk* $< 0$. | F.2.35 | ÷ | ok |
| Prøv udafledning med `/`*udtryk*, hvor *udtryk* ikke går op i antal bit i udbitvektoren. | F.2.36 | ÷ | fejl |

Hvis *udtryk* $\leq 0$ i `/`*udtryk*, rejses en intern fejl. Hvis `/`*udtryk*, hvor *udtryk* ikke går op i antal bit i udbitvektoren gives fejlmeddelelsen, at ind- og uddatastørrelsen i forbind ikke er ens.

## 10.4 Bitvektorer

**Konstanter, afløb, gennemløb**

| | | | |
|---|---|---|---|
| Prøv konstant `0x`*hextal*. | F.1.2 | + | ok |
| Prøv konstant *decimaltal*. | F.1.1 | + | ok |
| Prøv konstant, der er for stor til den specificerede størrelse. | F.2.37 | ÷ | ok |
| Prøv *bitvektor* >> *bitvektor konstant* >> *bitvektor*. | F.1.1 | + | ok |
| Prøv afløb. | F.1.1 | + | ok |
| Prøv *bitvektor* >> *afløb bitvektor* >> *bitvektor*. | F.1.2 | + | ok |
| Prøv gennemløbsudtryk. | F.1.3 | + | ok |
| Prøv *bitvektor* >> *gennemløb bitvektor* >> *bitvektor*. | F.1.3 | + | ok |
| Prøv *gennemløb* >> *bitvektor*. | F.2.38 | ÷ | fejl |
| Prøv *bitvektor* >> *gennemløb*. | F.2.39 | ÷ | fejl |

Når gennemløbsudtryk bruges ulovligt, rejses en intern fejl.

**Komponenter, skel, stik**

| | | | |
|---|---|---|---|
| Brug komponent. | F.1.1 | + | ok |
| Brug vektor. | F.1.1 | + | ok |
| Brug skel. | F.1.6 | + | ok |
| Brug bitvariabel. | F.1.4 | + | ok |
| Brug stik for programblokken. | F.1.1 | + | ok |
| Definer to komponenter med samme navn. | F.2.40 | ÷ | ok |
| Definer komponent med samme navn som plan. | F.2.41 | ÷ | ok |
| Definer komponent med samme navn som bitvariabel. | F.2.42 | ÷ | ok |
| Definer komponent med samme navn som stik for programblokken. | F.2.43 | ÷ | ok |
| Definer komponent og skel med samme navn. | F.2.44 | ÷ | ok |
| Definer to skel med samme navn. | F.2.45 | ÷ | ok |
| Definer skel med samme navn som plan. | F.2.46 | ÷ | ok |
| Definer skel med samme navn som bitvariabel. | F.2.47 | ÷ | ok |
| Definer skel med samme navn som stik for programblokken. | F.2.48 | ÷ | ok |

**Forbind**

| | | | |
|---|---|---|---|
| Prøv *bitvektor* >> *bitvektor* >> *bitvektor*. | F.1.1 | + | ok |
| Prøv *bitvektor* >> *skel* >> *skel* >> *bitvektor*. | F.1.6 | + | ok |
| Prøv *bitvektor0* >> *bitvektorN*, hvor størrelsen af udbitvektoren for *bitvektor0* $\neq$ størrelsen af indbitvektoren for *bitvektorN*. | F.2.49 | ÷ | ok |
| Prøv *bitvektor0* >> *bitvektorN*, hvor størrelsen af udbitvektoren for *bitvektor0* = størrelsen af indbitvektoren for *bitvektorN* = 0. | F.2.50 | ÷ | fejl |
| Prøv *bitvektor0* >> *skel* >> *bitvektorN*, hvor størrelsen af udbitvektoren for *bitvektor0* $\neq$ størrelsen af indbitvektoren for *bitvektorN*. | F.2.51 | ÷ | ok |
| Prøv *skel* >> *bitvektor*. | F.2.52 | ÷ | ok |
| Prøv *bitvektor* >> *skel*. | F.2.53 | ÷ | ok |

Hvis størrelserne i forbind er 0, rejses fejlmeddelelsen, at ind- og uddatastørrelserne er forskellige.

**Split**

| | | | |
|---|---|---|---|
| Prøv *bitvektor*, *bitvektor*. | F.1.4 | + | ok |
| Prøv *bitvektor* >> *bitvektor*, *bitvektor* >> *bitvektor*. | F.1.5 | + | ok |
| Prøv *bitvektor* >> *skel*, *bitvektor* >> *bitvektor*. | F.2.54 | ÷ | ok |
| Prøv *bitvektor*, *bitvektor*, hvor indbitvektorernes størrelse er forskellig. | F.2.55 | ÷ | ok |
| Prøv *bitvektor*, *bitvektor*, hvor indbitvektorernes har størrelse 0. | F.2.56 | ÷ | ok |

**Sekvens**

| | | | |
|---|---|---|---|
| Prøv *bitvektor bitvektor*. | F.1.1 | + | ok |
| Prøv *bitvektor* >> *bitvektor bitvektor* >> *bitvektor*. | F.1.1 | + | ok |
| Prøv *bitvektor* >> *skel bitvektor* >> *bitvektor*. | F.2.57 | ÷ | ok |

**Parenteser**

| | | | |
|---|---|---|---|
| Prøv (*bitvektor*). | F.1.2 | + | ok |
| Prøv (*bitvektor bitvektor*). | F.1.8 | + | ok |
| Prøv (*bitvektor*, *bitvektor*). | F.1.4 | + | ok |
| Prøv (*bitvektor* >> *bitvektor*). | F.1.5 | + | ok |
| Prøv *bitvektor* >> (*bitvektor*, *bitvektor*) *bitvektor* >> *bitvektor*. | F.1.4 | + | ok |
| Prøv *bitvektor* >> *bitvektor* (*bitvektor* >> *bitvektor*) >> *bitvektor*. | F.1.7 | + | ok |
| Prøv *bitvektor* >> *bitvektor*, (*bitvektor* >> *bitvektor*) >> *bitvektor*. | F.1.5 | + | ok |
| Prøv *bitvektor* >> (*skel*) >> *bitvektor*. | F.2.58 | ÷ | ok |

## 10.5 Krop

**Plandefinition**

| | | | |
|---|---|---|---|
| Prøv definition af plan. | F.1.4 | + | ok |
| Definer to planer med samme navn. | F.2.59 | ÷ | ok |
| Definer plan med samme navn som stik for programblokken. | F.2.60 | ÷ | fejl |
| Definer plan med andplaner med forskellig længde. | F.2.61 | ÷ | ok |
| Definer plan med orplaner med forskellig længde. | F.2.62 | ÷ | ok |

Hvis plan og stik har samme navn, rejses en intern fejl.

**Definition af bitvariable**

| | | | |
|---|---|---|---|
| Prøv definition af bitvariabel. | F.1.4 | + | ok |
| Definer to bitvariable med samme navn. | F.2.63 | ÷ | ok |
| Definer to bitvariable, hvor den ene indgår i den andens værdi. | F.1.5 | + | ok |
| Definer to bitvariable, hvor de indgår i hinandens værdi. | F.2.64 | ÷ | ok |
| Definer en bitvariabel, der indgår i sin egen værdi. | F.2.65 | ÷ | ok |

**Betingede sætninger**

| | | | |
|---|---|---|---|
| Lav betinget sætning, hvor det boolske udtryk giver `true`. | F.1.1 | + | ok |
| Lav betinget sætning, hvor det boolske udtryk giver `false`. | F.1.1 | + | ok |

**Kroppen**

| | | | |
|---|---|---|---|
| Lav en krop. | F.1.1 | + | ok |
| Lav krop, hvor en bit i en indbitvektor ikke forbindes. | F.2.66 | ÷ | ok |
| Lav krop, hvor en bit i en udbitvektor ikke forbindes. | F.2.67 | ÷ | ok |
| Lav krop, hvor en bit i en indbitvektor forbindes to gange. | F.2.68 | ÷ | ok |
| Lav krop, hvor rekursion bruges på lovlig vis. | F.1.3 | + | ok |
| Lav krop, hvor rekursion bruges uden indgående komponenter bliver mindre. | F.2.69 | ÷ | ok |

## 10.6 Program

**Hovedprogram**

| | | | |
|---|---|---|---|
| Prøv hovedprogram uden parametre. | F.1.2 | + | ok |
| Prøv hovedprogram med parametre. | F.1.1 | + | ok |
| Oversæt med korrekt antal parametre. | F.1.1 | + | ok |
| Oversæt med en parameter for lidt. | F.2.70 | ÷ | ok |
| Oversæt med en parameter for meget. | F.2.71 | ÷ | ok |
| Lav program uden hovedprogram. | F.2.72 | ÷ | ok |
| Lav program med to hovedprogrammer. | F.2.73 | ÷ | ok |

**Stik**

| | | | |
|---|---|---|---|
| Prøv definition af stik. | F.1.1 | + | ok |
| Definer stik, hvor størrelsen er 0. | F.2.74 | ÷ | ok |
| Definer stik, hvor størrelsen er $\leq 0$. | F.2.75 | ÷ | ok |
| Definer *stik stik størrelse*. | F.1.1 | + | ok |

**Komponentdefinition**

| | | | |
|---|---|---|---|
| Brug sammensat komponent med to forskellige værdisæt. | F.1.1 | + | ok |
| Definer komponent med ingen indstik. | F.1.2 | + | ok |
| Definer komponent med $\geq 1$ indstik. | F.1.1 | + | ok |
| Definer komponent med ingen udstik. | F.1.2 | + | ok |
| Definer komponent med $\geq 1$ udstik. | F.1.1 | + | ok |
| Definer komponent uden stik. | F.2.76 | ÷ | ok |
| Definer komponent med ingen parametre. | F.1.1 | + | ok |
| Definer komponent med $\geq 1$ parametre. | F.1.1 | + | ok |
| Lav komponentdefinitioner før og efter hovedprogrammet. | F.1.2 | + | ok |
| Definer komponent med samme navn som indbygget komponent. | F.2.77 | ÷ | ok |
| Definer to komponenter med samme navn. | F.2.78 | ÷ | ok |

## 10.7 Fejl i oversætteren

Bortset fra `sram`-komponenten virker Hadesoversætteren, så alle korrekte programmer oversættes til fungerende SimSysprogrammer, mens der rejses fejl ved oversættelse af Hadesprogrammer med fejl i. Blot rejses der nogle gange forkerte fejl, eller der opgives forkert position for fejlen.

Der rejses forkerte fejl ved følgende fejl:

- Størrelsesvariable $\leq 0$.
- Størrelse for afløb eller gennemløbsudtryk $= 0$.
- Udtryk $< 0$ i bitmønster.
- Hvis *udtryk* $\leq 0$ i `/`*udtryk*.
- Hvis *udtryk* i `/`*udtryk* i en udbitvektor ikke går op i antal bit i udbitvektoren.

- Gennemløb bruges ulovligt.
- Hvis den indgående ind- og udbitvektor i en forbindelse mellem to bitvektorer har størrelsen 0.
- Plan og stik for programblok har samme navn.

# Kapitel 11

# Konklusion

I dette speciale er sproget Hades blevet designet med henblik på brug på andenårskurset Dat1E. Det er dokumenteret, at sproget opfylder kursets krav til funktionalitet, og der er argumenteret for, at Hades er nemt at bruge og derfor har en passende sværhedsgrad til at blive brugt af andenårsstuderende.

Der er implementeret en oversætter for det meste af sproget. Det var ikke muligt at nå at implementere hele sproget indenfor den givne tidsramme, men der er lagt vægt på, at det skal være nemt at tilføje de resterende konstruktioner. Manglerne er opremset i afsnit 9.3.

Der er dokumenteret en afprøvning af oversætteren. Denne afslører kun ganske få fejl, der primært består af rejsning af forkerte fejlmeddelelser. Der rejses altså fejl, når der skal rejses fejl. Blot er fejlmeddelelserne til tider ikke korrekte. Der har ikke været tid til at rette fejlene, men de er opsummeret i afsnit 10.7, så det er klart, hvad der skal ændres i oversætteren.

Før Hades bruges på Dat1E, skal fejlene i den nuværende Hadesoversætter rettes, og de resterende dele af Hades skal tilføjes. Ideelt set bør der også laves en brugerafprøvning af studerende.

På længere sigt kan funktionaliteten af Hades udvides, så det bliver muligt at specificere dele af et program i SimSys. Det vil gøre det muligt at bruge de konstruktioner i SimSys, som er udeladt i Hades.

# Bilag A

# Eksempler på todimensionalt program

Til illustration af en skitse af et todimensionalt sprog bruges en 32-bit `alu`, der minder om aluen i de vejledende løsninger til rapportopgaverne på side 47. Blot laves der altid venstre- og højrebitskift. Kredsløbet er det samme som i afsnit 8.4.

Først laves et eksempel på et program, der er lavet lige ud ad landevejen. Som det ses, giver dette et temmelig omfattende program. Bagefter laves der en kompakt version af det samme program. Denne fylder meget lidt, men er måske nok for avanceret for den almindelige studerende.

Syntaksen i eksemplerne minder relativt meget om Hadessyntaks.

## A.1 Et ikke-kompakt program

```
alu (in v: inA inB 32) >> (out v: res, out h: relVec)

                    (inA inB)
                    *6

relOp  bworarray’32 2’  bwandarray’32 2’  bwxorarray’32 2’  0>add>_  ’32’ notarray’32’
[1]

’1’  0’31’  ’4*32’                                                    1>sub’add 32’>_

                       [..6*32-1]
                       mux
;
relOp

relVec
;
ctrl

pla
;
inA  inB    pla     inA  inB    pla
     [..4]  [0]*37       [..4]  [1]*37

bwandarrayR’32 2’    bwandarrayL’32 2’

rshifter            lshifter  ctrl
                              [..2]
            [6*32..]
            mux

            res
```

`;` markerer, at byggeklodserne i linien over `;` ikke forbindes med byggeklodserne i linien under. Udtryk som `*6` og lignende over og under en linie hører til den komponent, det står lige under eller over, og angiver hvilke og hvor mange bit der tages fra komponenten. Er der en parentes om flere komponenter, laves operationen på dem alle. Således betyder `*6` under `(inA inB)`, at alle bit i `inA` og `inB` lægges i rækkefølge, og dette gøres `6` gange. Et eller flere udtryk omgivet af `'` angiver typen (f.eks. `adder` - kun nødvendig, hvis den ikke kan udledes) og værdierne for komponenten. Specielt svarer `'1'` til, at en bit flyder igennem det pågældende lag - altså til en slags identitetsfunktion. `0>add>_` markerer, at `cIn` forbindes med `0` og `cOut` med et afløb.

En meget mere kompakt variant af samme program kan skrives som

```
alu (in v: inA inB 32) >> (out v: res,out h: relVec)

                       (inA   inB)                                          ctrl
                       6+2[..36]                                            1+[..2]

relOp bworarray'32 2' bwandarray'32 2' bwxorarray'32 3' 0>add>_ '32' notarray'32' pla '2'
1+[1]                                                                        37[0]37[1]
                                              /2[..36,37..]
relVec  '1' 0'31'           1>sub'add 32'>_  (bwandarrayR'32 2' bwandarrayL'32 2')

                      '6*32'                    rshifter  lshifter

                                   mux

                                   res
```

Her betyder `/2[..36;37..]`, at først tages bit 0 til 36 fra `bwandarrayR` og så fra `bwandarrayL`, derefter tages bit 37 indtil sidste bit fra først `bwandarrayR` og så `bwandarrayL`. I dette eksempel er hele programmet skrevet "ud i et" (uden `;`). Derfor er det på få liniers kode. Der skal dog tænkes noget mere, når den del af kredsløbet, der indeholder `rshifter` og `lshifter` specificeres samtidig med resten af kredsløbet.

Begge ovenstående programeksempler er læselige, når man lige vænner sig til syntaksen. De fleste studerende vil formodentlig programmere i nogenlunde samme stil, som det er gjort i første eksempel, og da dette fylder (for) mange linier, blev idéen med at bruge et todimensionalt sprog aflivet.

# Bilag B

# Vejledende løsninger til G- og K1-opgaven, 1999

I dette bilag er vejledende løsninger til G- og K1-opgaverne på Dat1E i 1999. Disse opgaver svarer nogenlunde til de nuværende G2- og K1-opgaver på Dat1E, og de tilsvarende programmer i Hades er i bilag C.

Der er en fælles kerne i løsningerne, som jeg starter med i afsnit B.1. Herefter kommer først løsningen til G-opgaven i afsnit B.2 og dernæst til K1-opgaven i afsnit B.3.

## B.1 Grundlæggende byggeklodser

### mipsdecoder

```
// "decode.h" -- MIPS instruction decoder
// (C) Finn Schiermer Andersen, 1998

#ifndef __DECODE_H__
#define __DECODE_H__

#include "interface.h"

struct MipsDecoder : public AutoDelete {
  // input side
  INABSTRACTION& invalid;   //! [1]     inst is invalid ?
  INABSTRACTION& inst;      //! [32]    instruction
  INABSTRACTION& nextPc;    //! [30]    address of inst + 4.

  // output side
  OUTABSTRACTION& exDstValid; //! [1]   dstRegNr is valid after ex (not zero)
  OUTABSTRACTION& mDstValid;  //! [1]   dstRegNr is valid after m  (not zero)
  OUTABSTRACTION& dstRegNr;   //! [5]   write result to this register
  OUTABSTRACTION& opARegNr;   //! [5]   read opA from this register
  OUTABSTRACTION& opBRegNr;   //! [5]   read opB from this register
  OUTABSTRACTION& opCRegNr;   //! [5]   read opC from this register
  OUTABSTRACTION& needRegA;   //! [1]   is opA required for EX (stall on use)
  OUTABSTRACTION& needRegB;   //! [1]   is opB required for EX (stall on use)
  OUTABSTRACTION& needRegC;   //! [1]   is opC required for M (no stall on use)
  OUTABSTRACTION& value;      //! [32]  embedded/derived value
  OUTABSTRACTION& aluCtrl;    //! [7]   control-bits for alu stage
  OUTABSTRACTION& mCtrl;      //! [2]   control-bits for m stage
  OUTABSTRACTION& isJorJal;   //! [1]   is a J or a Jal instruction
  OUTABSTRACTION& isStop;     //! [1]   is a STOP instruction
  OUTABSTRACTION& isBne;      //! [1]   is a Bne instruction
```

```
  OUTABSTRACTION& isBeq;      //! [1]   is a Beq instruction
  OUTABSTRACTION& isJr;       //! [1]   is a Jr instruction

  // Instantiation
  MipsDecoder(INABSTRACTION& invalid,
              INABSTRACTION& inst,
              INABSTRACTION& nextPc,
              OUTABSTRACTION& exDstValid,
              OUTABSTRACTION& mDstValid,
              OUTABSTRACTION& dstRegNr,
              OUTABSTRACTION& opARegNr,
              OUTABSTRACTION& opBRegNr,
              OUTABSTRACTION& opCRegNr,
              OUTABSTRACTION& needRegA,
              OUTABSTRACTION& needRegB,
              OUTABSTRACTION& needRegC,
              OUTABSTRACTION& value,
              OUTABSTRACTION& aluCtrl,
              OUTABSTRACTION& mCtrl,
              OUTABSTRACTION& isJorJal,
              OUTABSTRACTION& isStop,
              OUTABSTRACTION& isBne,
              OUTABSTRACTION& isBeq,
              OUTABSTRACTION& isJr)
    : invalid(invalid), inst(inst), nextPc(nextPc), exDstValid(exDstValid),
      mDstValid(mDstValid), dstRegNr(dstRegNr), opARegNr(opARegNr),
      opBRegNr(opBRegNr), opCRegNr(opCRegNr), needRegA(needRegA),
      needRegB(needRegB), needRegC(needRegC), value(value), aluCtrl(aluCtrl),
      mCtrl(mCtrl), isJorJal(isJorJal),
      isStop(isStop), isBne(isBne), isBeq(isBeq), isJr(isJr)
    {};

    static MipsDecoder& mk(String name);
};

#endif

// "decode.cc" -- Mips instruction decoder
// (C) Finn Schiermer Andersen, 1998

#include "decode.h"
#include "composite.h"
#include "simlib.h"
#include "adder.h"  // sinus,2002: bruges ikke

MipsDecoder& MipsDecoder::mk(String home) {
  // PADs
  D(INPAD,invalid,1);
  D(INPAD,inst,32);
  D(INPAD,nextPc,30);
  D(OUTPAD,exDstValid,1);
  D(OUTPAD,mDstValid,1);
  D(OUTPAD,dstRegNr,5);
  D(OUTPAD,opARegNr,5);
  D(OUTPAD,opBRegNr,5);
  D(OUTPAD,opCRegNr,5);
  D(OUTPAD,needRegA,1);
  D(OUTPAD,needRegB,1);
  D(OUTPAD,needRegC,1);
  D(OUTPAD,value,32);
  D(OUTPAD,aluCtrl,4);
  D(OUTPAD,mCtrl,2);
  D(OUTPAD,isJorJal,1);
  D(OUTPAD,isStop,1);
  D(OUTPAD,isBne,1);
  D(OUTPAD,isBeq,1);
  D(OUTPAD,isJr,1);
```

```
// Production of the proper const/derived value.
// 0 PC+4+(sign extended imm << 2)         for branch
// 1 PC+4                                  for jal
// 2 sign extended imm                     for signed imm and ld/st
// 3 zero extended imm                     for unsigned imm
// 4 imm<<16                               for lui
D(FastMux,valMux,5,32);
D(Adder,bTargetAdd,32);
//D(CSAdder32,bTargetAdd);
nextPc                      >> bTargetAdd.inA.subset(2,30);
Const("0",2)                >> bTargetAdd.inA.subset(0,2)
                            >> bTargetAdd.inB.subset(0,2);
inst.subset(0,16)           >> bTargetAdd.inB.subset(2,16);
inst[15].repeat(14)         >> bTargetAdd.inB.subset(18,14);
Const("0",1)                >> bTargetAdd.cIn; bTargetAdd.cOut >> 0;
bTargetAdd.sum              >> valMux.in[0];
nextPc+Const("0",2)         >> valMux.in[1];
inst.subset(0,16)           >> valMux.in[2].subset(0,16)
                            >> valMux.in[3].subset(0,16)
                            >> valMux.in[4].subset(16,16);
inst[15].repeat(16)         >> valMux.in[2].subset(16,16);
Const("0",16)               >> valMux.in[3].subset(16,16)
                            >> valMux.in[4].subset(0,16);
valMux.out                  >> value;

// Valid register specifier for ex and m stages
D(OrGate,dstOr,5);
D(AndGate,exDstAnd,2);
dstOr.out    >> mDstValid    >> exDstAnd.in[0];
exDstAnd.out >> exDstValid;

// Destination Register selector
// 0 const $0.
// 1 const $31.
// 2 field d
// 3 field t
D(FastMux,dstMux,4,5);
Const("0",5)      >> dstMux.in[0];
Const("1F",5)     >> dstMux.in[1];
inst.subset(11,5) >> dstMux.in[2];
inst.subset(16,5) >> dstMux.in[3];
dstMux.out        >> dstRegNr        >> dstOr.in;

inst.subset(21,5) >> opARegNr;
inst.subset(16,5) >> opBRegNr >> opCRegNr;


// The decoder/control pla:
// 13 inputs: Op, func, and invalid fields.
// Outputs: (24)
// Control of muxes above:
// valMuxCtrl, 4
// dstMuxCtrl, 4, also used for mDstValid
// opAMuxCtrl, 2, also used for needRegA
// opBMuxCtrl, 2, also used for needRegB
// opCMuxCtrl, 2, also used for needRegC
// exDstValid, 1
// aluCtrl, 7 (4:beq, 5:bne, 6:jr, 0:3 alu control, stop=beq+bne
//   alu control: 0000 slt
//   alu control: 0001 or
//   alu control: 0010 and
//   alu control: 0011 xor
//   alu control: 0100 +
//   alu control: 0101 -
//   alu control: 0110 <<
```

```
//   alu control: 0111 >>
//   alu control: 1xxx unused
// mCtrl, 2 (0: rd, 1: wr)
//                                      isStop
//                                      |valMuxCtrl (lui-zext-sext-jal-bra)
//                                      ||||||dstMuxCtrl (t-d-31-0)
//                                      ||||||||||isJorJal
//                                      |||||||||||needRegA
//                                      ||||||||||||needRegB
//                                      |||||||||||||needRegC
//                               invalid ||||||||||||||exDstValid
//                                     | ||||||||||||||isJr
//                                     | |||||||||||||||isBne
//                                     | ||||||||||||||||isBeq
//                           funcField| ||||||||||||||||AluCtrl
//                         opField|||||| ||||||||||||||||||||mCtrl
//                          |||||||||||| |^^^^^||||^|||^|^|^^^^||
static char* dSpec[] = { "00000001000000 -------1---11-1----1----",/*add R*/
                         "00000001000100 -------1---11-1----1-1--",/*sub R*/
                         "00000001010100 -------1---11-1---------",/*slt R*/
                         "00000001001000 --1----1---11-1-----1---",/*and R*/
                         "00000001001010 --1----1---11-1------1--",/*or R */
                         "00000001001100 --1----1---11-1-----11--",/*xor R*/
                         "00000000010000 ---------1-1---1--------",/*jr */
                         "100011------0 ---1--1----1-------1---1",/*lw*/
                         "101011------0 ---1-----1-1-1-----1--1-",/*sw*/
                         "001000------0 ---1--1----1--1----1----",/*add I*/
                         "001010------0 ---1--1----1--1---------",/*slt I*/
                         "001100------0 --1---1----1--1-----1---",/*and I*/
                         "001101------0 --1---1----1--1------1--",/*or I*/
                         "001110------0 --1---1----1--1-----11--",/*xor I*/
                         "001111------0 -1----1-------1------1--",/*lui I*/
                         "000100------0 -----1---1-11----1------",/*beq*/
                         "000101------0 -----1---1-11---1-------",/*bne*/
                         "000010------0 ----------11-------------",/*j */
                         "000011------0 ----1---1-1---1------1--",/*jal*/
                         "111111------0 1-----------------------",/*stop*/
                         ------------1 ------------------------",/*inv*/
};
D(PLA,dec,13,24,20,dSpec);
inst.subset(26,6) >> dec.in.subset(7,6);
inst.subset(0,6)  >> dec.in.subset(1,6);
invalid           >> dec.in.subset(0,1);
dec.out.subset(18,5) >> valMux.select;
dec.out.subset(14,4) >> dstMux.select;
dec.out.subset(9,1)  >> exDstAnd.in[1];
dec.out.subset(2,4)  >> aluCtrl;
dec.out.subset(0,2)  >> mCtrl;
dec.out[23] >> isStop;
dec.out[6]  >> isBeq;
dec.out[7]  >> isBne;
dec.out[8]  >> isJr;
dec.out[13] >> isJorJal;
dec.out[12] >> needRegA;
dec.out[11] >> needRegB;
dec.out[10] >> needRegC;

// Exporting the interface..

return * new MipsDecoder(invalid.outside(), inst.outside(), nextPc.outside(),
                         exDstValid.outside(), mDstValid.outside(),
                         dstRegNr.outside(), opARegNr.outside(),
                         opBRegNr.outside(), opCRegNr.outside(),
                         needRegA.outside(), needRegB.outside(),
                         needRegC.outside(), value.outside(),
                         aluCtrl.outside(), mCtrl.outside(),
```

```
                          isJorJal.outside(),
                          isStop.outside(), isBne.outside(),
                          isBeq.outside(), isJr.outside());
};
```

### shifter

```
// "shifter.h" -- Shifter circuit.
// (C) Finn Schiermer Andersen, 1998

#ifndef __SHIFTER_H__
#define __SHIFTER_H__

#include "interface.h"

struct Shifter : public AutoDelete {
  INABSTRACTION& in;    // [1<<shft] number to be shifted
  OUTABSTRACTION& out;  // [1<<shft] result
  INABSTRACTION& amount;// [shft]
  Shifter(INABSTRACTION& in,
          OUTABSTRACTION& out,
          INABSTRACTION& amount)
    : in(in), out(out), amount(amount) {};
    enum Direction { LEFT, RIGHT };
  static Shifter& mk(String home, int shft, Direction dir);
};

#endif

// "shifter.cc" -- Implementation of shift circuit
// (C) Finn Schiermer Andersen, 1998

#include "shifter.h"
#include "simlib/mux.h"
#include "composite.h"
#include <assert.h>

Shifter& Shifter::mk(String home ,int shft, Direction dir) {
  assert(shft<32);
  int width = 1<<shft;

  // PADs
  D(INPAD,in,width);
  D(INPAD,amount,shft);
  D(OUTPAD,out,width);

  // interior
  // first layer need special care
  OUTABSTRACTION* tmp;
  if (dir==LEFT) {
    // build a left-shift-stage:
    D1(Mux,lshft,0,1,width);
    amount.subset(0,1) >> lshft.select;
    in >> lshft.in[0];
    in.subset(0,width-1) + Const("0",1) >> lshft.in[1];
    tmp = & lshft.out;
  } else {
    // build a right-shift-stage:
    D1(Mux,rshft,0,1,width);
    amount.subset(0,1) >> rshft.select;
    in >> rshft.in[0];
    Const("0",1) + in.subset(1,width-1) >> rshft.in[1];
    tmp = & rshft.out;
  };
  for (int j=1; j<shft; j++) {
    if (dir==LEFT) {
```

```
      // build a left-shift-stage:
      D1(Mux,lshft,j,1,width);
      amount.subset(j,1) >> lshft.select;
      *tmp >> lshft.in[0];
      tmp->subset(0,width-(1<<j)) + Const("0",1<<j) >> lshft.in[1];
      tmp = & lshft.out;
    } else {
      // build a right-shift-stage:
      D1(Mux,rshft,j,1,width);
      amount.subset(j,1) >> rshft.select;
      *tmp >> rshft.in[0];
      Const("0",1<<j) + tmp->subset(1<<j,width-(1<<j)) >> rshft.in[1];
      tmp = & rshft.out;
    };
  };
  *tmp >> out;
  return * new Shifter(in.outside(),out.outside(),amount.outside());
};
```

## relop

```
// "relop.h" -- interface to relation operator
// (C) Finn Schiermer Andersen, 1998

#ifndef __RELOP_H__
#define __RELOP_H__

#include "interface.h"

/*
//+
RelOp

This building block examines two operands and outputs a bit vector indicating
the relationship between them. The relations detected are:

  bit     symbol     meaning
  0       ne         opA != opB
  1       lts        opA <  opB (signed)
  2       gts        opA >  opB (signed)
  3       ltu        opA <  opB (unsigned)
  4       gtu        opA >  opB (unsigned)
  5       eq         opA == opB
  6       ges        opA >= opB (signed)
  7       les        opA <= opB (signed)
  9       geu        opA >= opB (unsigned)
  9       leu        opA <= opB (unsigned)

//-
*/

struct RelOp : public AutoDelete {  //!
  INABSTRACTION& inA;      //! [32]
  INABSTRACTION& inB;      //! [32]
  OUTABSTRACTION& relVec;  //! [10]

  RelOp(INABSTRACTION& inA,
        INABSTRACTION& inB,
        OUTABSTRACTION& relVec)
    : inA(inA), inB(inB), relVec(relVec)
    {};

    static RelOp& mk(String name);
};

#endif
```

```
// "relop.cc" -- implementation of relational operators.
// (C) Finn Schiermer Andersen, 1998

#include "relop.h"
#include "composite.h"

#include "simlib/priority.h"
#include "simlib/pla.h"
#include "simlib/mux.h"
#include "simlib/gates.h"  // well, not that Gates :-)

//                      B gtu
//                     A| |ltu
//               isSign|| ||gts
//               valid||| |||lts
//                   |||| ||||
char *plaSpec[] = { "1001 -1-1",
                    "1010 1-1-",
                    "1101 -11-",
                    "1110 1--1" };

RelOp& RelOp::mk(String home) {
  // pads
  D(INPAD,inA,32);
  D(INPAD,inB,32);
  D(OUTPAD,relVec,10);

  // interior:
  D(BWXorArray,xor,32,2);
  D(FastMux,muxA,32,1);
  D(FastMux,muxB,32,1);
  D(PLA,pla,4,4,4,plaSpec);
  D(BWNotArray,not,5);
  inA >> xor.in[0];
  inB >> xor.in[1];
  D(Prioritizer,firstDiff,32);
  xor.out >> firstDiff.in;
  firstDiff.out >> muxA.select >> muxB.select;
  inA >> muxA.in.join();
  inB >> muxB.in.join();
  firstDiff.out[31] >> pla.in[2]; /* isSign */
  muxA.out >> pla.in[1]; /* opA */
  muxB.out >> pla.in[0]; /* opB */
  firstDiff.valid >> pla.in[3] >> relVec[0] >> not.in[0][0];
  pla.out >> relVec.subset(1,4) >> not.in[0].subset(1,4);
  not.out >> relVec.subset(5,5);
  // interface:
  return * new RelOp(inA.outside(), inB.outside(), relVec.outside());
};
```

## alu

```
// "alu.h" -- Generic 32-bit arithmetic/logic unit, but no flags!
// (C) Finn Schiermer Andersen, 1998

#ifndef __ALU_H__
#define __ALU_H__

#include "interface.h"

/*
  Control      Operation
  0000          res = slt inA, inB
  0001          res = inA or inB
  0010          res = inA and inB
  0011          res = inA xor inB
```

```
  0100          res = inA + inB
  0101          res = inA - inB (two cpl)
  0110          res = inA << inB (unsigned)
  0111          res = inA >> inB (unsigned)
  1---          reserved for future extensions

  As an optimisation the entire relation vector is output along with the
  result of the operation. This is handy when implementing conditional
  branches outside the ALU.
*/

struct ALU {
  INABSTRACTION& inA;     // [32]
  INABSTRACTION& inB;     // [32]
  INABSTRACTION& ctrl;    // [4]
  OUTABSTRACTION& res;    // [32]
  OUTABSTRACTION& relVec; // [10]
  ALU(INABSTRACTION& inA,
      INABSTRACTION& inB,
      INABSTRACTION& ctrl,
      OUTABSTRACTION& res,
      OUTABSTRACTION& relVec)
    : inA(inA), inB(inB), ctrl(ctrl), res(res), relVec(relVec) {};
  static ALU& mk(String home);
};

#endif
```

```
// "alu.cc" -- Implementation of generic 32-bit ALU
// (C) Finn Schiermer Andersen, 1998

#include "alu.h"
#include "simlib/gates.h"
#include "simlib/adder.h"
#include "simlib/mux.h"
#include "shifter.h"
#include "simlib/pla.h"
#include "composite.h"
#include "relop.h"

// uncomment define below to enable dynamic shifters
//#define SHIFTERS

ALU& ALU::mk(String home) {
  D(INPAD,ctrl,4);
  D(INPAD,inA,32);
  D(INPAD,inB,32);
  D(OUTPAD,res,32);
  D(OUTPAD,relVec,10);

  // build add/sub path
  D(Mux,resMux,3,32);
  D(BWNotArray,negator,32);
  D(Adder,adder,32);
  D(Adder,suber,32);
  inA >> adder.inA >> suber.inA;
  inB >> adder.inB >> negator.in[0]; negator.out >> suber.inB;
  "0 [1]" >> adder.cIn;
  "1 [1]" >> suber.cIn;
  adder.sum >> resMux.in[4];
  suber.sum >> resMux.in[5];
  ctrl.subset(0,3) >> resMux.select;
  adder.cOut >> 0;
  suber.cOut >> 0;

  // build slt path
  D(RelOp,relOp);
```

```
  inA >> relOp.inA;
  inB >> relOp.inB;
  relOp.relVec >> relVec;
  relOp.relVec[1] >> resMux.in[0].subset(0,1);
  Const("0",31)   >> resMux.in[0].subset(1,31);

  // build And, Or and Xor paths
  D(BWAndArray,and,32,2);
  D(BWOrArray,or,32,2);
  D(BWXorArray,xor,32,2);
  or.out  >> resMux.in[1];
  and.out >> resMux.in[2];
  xor.out >> resMux.in[3];
  inA >> and.in[0]
      >> or.in[0]
      >> xor.in[0];
  inB >> and.in[1]
      >> or.in[1]
      >> xor.in[1];

#ifdef SHIFTERS
  // build lshift and rshift paths -- Warning the data path below is usually
  // not time-critical. I've therefore optimized it for faster simulation, by
  // selectively powering down the shifters. This is done by filtering their
  // inputs, feeding only zeros, when their results are not needed. This makes
  // the alu faster for usual use (few shifts) but will make it slower, if
  // shifts are heavily used.
  D(Shifter,lshft,5,Shifter::LEFT);
  D(Shifter,rshft,5,Shifter::RIGHT);
  static char* shftSaverPlane[] = { "0110 1-",
                                    "0111 -1" };
  D(PLA,shftSaver,4,2,2,shftSaverPlane);
  ctrl >> shftSaver.in;
  D(BWAndArray,enableLshft,37,2);
  D(BWAndArray,enableRshft,37,2);
  shftSaver.out[0].repeat(37) >> enableRshft.in[0];
  shftSaver.out[1].repeat(37) >> enableLshft.in[0];
  inA >> enableRshft.in[1].subset(0,32)
      >> enableLshft.in[1].subset(0,32);
  inB.subset(0,5) >> enableRshft.in[1].subset(32,5)
                  >> enableLshft.in[1].subset(32,5);
  enableRshft.out.subset(0,32) >> rshft.in;
  enableLshft.out.subset(0,32) >> lshft.in;
  enableRshft.out.subset(32,5) >> rshft.amount;
  enableLshft.out.subset(32,5) >> lshft.amount;
  lshft.out >> resMux.in[6];
  rshft.out >> resMux.in[7];
#else
  Const("0",32) >> resMux.in[6] >> resMux.in[7];
#endif // of IFDEF SHIFTERS


  resMux.out >> res;

  // get out of here
  return *new ALU(inA.outside(), inB.outside(),
                  ctrl.outside(), res.outside(), relVec.outside());
};
```

## B.2 Løsning til G-opgaven, 1999

```
// "single.cc" -- Single Cycle MIPS microarch
// (C) Finn Schiermer Andersen, 1999

#include <strstream.h>
```

```
#include "simlib.h"
#include "alu.h"
#include "decode.h"
#include "disasm.h"


class MySimApp : public SimApp {
public:
  virtual void BuildModel();
  virtual int  Cycle(unsigned long CurrentCycle);
  virtual void PreRun() { Stop = Channel("dec.isStop");}
  MySimApp(
           int outputEnabled,
           char* name)
    : outputEnabled(outputEnabled),
      name(name)
  {};
protected:
  int outputEnabled;
  char* name;
  CHANNEL Stop;
};

SimApp*
SimApp::mk(int argc, char *argv[], char *envp[])
{
  int outputEnabled = -1;
  if (argc!=3) {
    cerr << "Wrong number of arguments." << endl;
    cerr << " Use " << argv[0]
         << " fname printLevel" << endl;
    exit(1);
  };
  if (argc==3) {
    istrstream ss(argv[2],10);
    ss >> outputEnabled;
    if (outputEnabled >= 0 && outputEnabled <= 2)
      argv[2] = 0;
    else
      cout << "Error parsing printLevel." << endl;
  }
  else
    outputEnabled=0;
  char* fname = argv[1];
  argv[1] = 0;
  return new MySimApp(outputEnabled,fname);
}

void MySimApp::BuildModel() {
  /*
    ---------------------------------------------------------------------
    MicroArchitecture Starts
    ---------------------------------------------------------------------
  */

  /*
    Memory subsystem
  */
  D(IdealMemIOSys,mem, 20,1,name);
  IdealMemIOSys::setIOManager(0xF,* new IntIO(cin,cout));
  /*
    Shared register file
  */
  D(RegisterFile,reg, 1,3,5,32);
  D(ValidateWrite,val,1,5,32);
```

```
/*
  Instruction fetch & global control
*/
D(FlipFlop,pc, 32,0);
D(Adder,pcInc, 32);  "0 [1]" >> pcInc.cIn; "4 [20]" >> pcInc.inB;
pc.out >> pcInc.inA >> mem.iAddrIn;
"1 [1]" >> mem.iRd;
D(FastMux,pcMux, 4,32);
pcInc.sum >> pcMux.in[0];
pcInc.cOut >> 0;
pcMux.out >> pc.in;

/*
  Decode & register select
*/
D(MipsDecoder,dec);
mem.iDataOut >> dec.inst;
"0 [1]" >> dec.invalid;
pcInc.sum.subset(2,30) >> dec.nextPc;
dec.dstRegNr >> reg.wrSelect[0] >> val.idxIn[0];
D(Mux,valAMux, 1,32); dec.needRegA >> valAMux.select;
D(Mux,valBMux, 1,32); dec.needRegB >> valBMux.select;
D(Mux,valCMux, 1,32); dec.needRegC >> valCMux.select;
dec.opARegNr >> reg.rdSelect[0]; reg.rdData[0] >> valAMux.in[1];
dec.opBRegNr >> reg.rdSelect[1]; reg.rdData[1] >> valBMux.in[1];
dec.opCRegNr >> reg.rdSelect[2]; reg.rdData[2] >> valCMux.in[1];
dec.value >> valAMux.in[0] >> valBMux.in[0] >> valCMux.in[0];
dec.exDstValid >> 0;
dec.isStop >> 0;
/*
  Execute
*/
D(ALU,alu);
valAMux.out >> alu.inA; valBMux.out >> alu.inB;
dec.aluCtrl >> alu.ctrl;
dec.value >> pcMux.in[1];
valAMux.out >> pcMux.in[2];
pc.out.subset(28,4) >> pcMux.in[3].subset(28,4);
mem.iDataOut.subset(0,26) >> pcMux.in[3].subset(2,26);
"0 [2]" >> pcMux.in[3].subset(0,2);

//                        isJr
//                    isJorJal|
//                         eq|| jal
//                       isEq||| |jr
//                        ne|||| ||bra
//                    isBne||||| |||nextPc
//                        ||||||  ||||
static char *ctrl[] = { "11---- --1-",
                        "10---- ---1",
                        --11-- --1-",
                        --10-- ---1",
                        ----1- 1---",
                        -----1 -1--",
                        "0-0-00 ---1" };

D(PLA,ctrlPla, 6,4,7,ctrl);
dec.isJr     >> ctrlPla.in[0];
dec.isJorJal >> ctrlPla.in[1];
dec.isBeq    >> ctrlPla.in[3];
dec.isBne    >> ctrlPla.in[5];
alu.relVec[0]>> ctrlPla.in[4];
alu.relVec[5]>> ctrlPla.in[2];
ctrlPla.out  >> pcMux.select;

/*
```

```
    Memory access
  */
  alu.res     >> mem.dAddrIn;
  valCMux.out >> mem.dDataIn;
  dec.mCtrl[0] >> mem.dRd; dec.mCtrl[1] >> mem.dWr;
  D(Mux,resMux, 1,32); dec.mCtrl[0] >> resMux.select;
  mem.dDataOut >> resMux.in[1]; alu.res >> resMux.in[0];

  /*
    Write back
  */
  dec.mDstValid >> reg.wrEnable[0]
                >> val.wrEnable[0];
  resMux.out >> reg.wrData[0]
             >> val.dataIn[0];

  /*
    ---------------------------------------------------------------
    MicroArchitecture Ends
    ---------------------------------------------------------------
  */

};



int MySimApp::Cycle(unsigned long currentCycle) {
  if (outputEnabled > 1) {
    cout << currentCycle << "  ";
    cout << Channel("pc.out").getData()
         << " : "
         << Channel("dec.inst").getData();
    cout << "  " << disassemble(Channel("dec.inst").getData()) << "  ";
    if (Channel("mem.dWr").getData()!=0)
      cout << "mem["
           << Channel("mem.dAddrIn").getData() << "] <- "
           << Channel("mem.dDataIn").getData() << "  ";
    if (Channel("reg.wrEnable[0]").getData()!=0)
      cout << "$"
           << Channel("reg.wrSelect[0]").getData() << " <- "
           << Channel("reg.wrData[0]").getData();
    cout << endl;
  }
  if (Stop.getData()!=Data(1,0)) return 0;
  return SimApp::Cycle(currentCycle);
};
```

## B.3 Løsning til K1-opgaven, 1999

### bypass

```
// "bypass.h" -- Generic bypass module
// (C) Finn Schiermer Andersen, 1998

#ifndef __BYPASS_H__
#define __BYPASS_H__

#include "interface.h"

/*
//+
Bypass ("bypass.h")

     int sz      number of entries
```

```
     int dataSz  number of bits in bypassed word
     int idSz    number of bits in an identifier

The Bypass object selects among a prioritized set of inputs. The input
with highest priority, valid set and an id matching the requset is
supplied as output. If no matching input exist, the output is undefined.
slot 0 is lowest priority, slow entries-1 has highest priority.
//-
*/

struct Bypass : public virtual AutoDelete {
  INARRAYABSTRACTION& dataIn;   //! [sz,dataSz]
  INARRAYABSTRACTION& wrSelect; //! [sz,idSz]
  INABSTRACTION&      validIn;  //! [sz]
  INABSTRACTION&      rdSelect; //! [idSz]
  OUTABSTRACTION&     dataOut;  //! [dataSz]
  Bypass(INARRAYABSTRACTION& dataIn,
         INARRAYABSTRACTION& wrSelect,
         INABSTRACTION&      validIn,
         INABSTRACTION&      rdSelect,
         OUTABSTRACTION&     dataOut)
    : dataIn(dataIn), wrSelect(wrSelect), validIn(validIn),
      rdSelect(rdSelect), dataOut(dataOut) {};
  static Bypass& mk(String home, int sz, int dataSz, int idSz);
};


#endif

// "bypass.cc" -- implementation of generic bypass
// (C) Finn Schiermer Andersen, 1998

#include "bypass.h"
#include "simlib/gates.h"
#include "simlib/priority.h"
#include "simlib/mux.h"
#include "composite.h"
#include "simlib/search.h"

/*
Bypass& Bypass::mk::operator(,int sz, int dataSz, int idSz) {
*/
Bypass& Bypass::mk(String home, int sz, int dataSz, int idSz) {
  // PADs
  D(INPADARRAY,dataIn,sz,dataSz);
  D(INPADARRAY,wrSelect,sz,idSz);
  D(INPAD,validIn,sz);
  D(INPAD,rdSelect,idSz);
  D(OUTPAD,dataOut,dataSz);

  // Generate valid matches
  D(QualifiedSearch, matcher,sz,idSz);
  wrSelect.join() >> matcher.keys;
  validIn         >> matcher.valid;
  rdSelect        >> matcher.key;

  // Prioritize among valid matches, propagates highest priority match
  D(Prioritizer, pri,sz);
  matcher.hits >> pri.in;
  D(FastMux, mux,sz,dataSz);
  dataIn    >> mux.in;
  pri.out   >> mux.select;
  pri.valid >> 0;
  mux.out   >> dataOut;
  return * new Bypass(dataIn.outside(),
                      wrSelect.outside(),
                      validIn.outside(),
```

```
                      rdSelect.outside(),
                      dataOut.outside());
};
```

### mips

```
// "mips.cc" -- Mips pipeline
// (C) Finn Schiermer Andersen, 1998

#include "simlib.h"

#include "decode.h"
#include "alu.h"
#include "bypass.h"
#include "disasm.h"
#include <strstream.h>

#include "utility/boundary.h"


void prtReg(char* s, Data valid, Data nr);

#ifdef USEGC
#include "sgc.h"
#endif



class MySimApp : public SimApp {
public:
  virtual void BuildModel();
  virtual int  Cycle(unsigned long CurrentCycle);
  virtual void PreRun() { Stop = Channel("decode.isStop.IDZap.IDEX.EXM");}
  MySimApp(
           int outputEnabled,
           int assoc,
           int indexSz,
           int blockSz,
           char* name)
    : outputEnabled(outputEnabled),
      assoc(assoc),
      indexSz(indexSz),
      blockSz(blockSz) ,
      name(name)
  {};
protected:
  int outputEnabled;
  int assoc;
  int indexSz;
  int blockSz;
  char* name;
  CHANNEL Stop;
};

SimApp*
SimApp::mk(int argc, char *argv[], char *envp[])
{
  int outputEnabled;
  int assoc = 7;
  int indexSz = 5;
  int blockSz = 5;
  if (argc!=2 && argc!=3 && argc!=6) {
    cerr << "Wrong number of arguments." << endl;
    cerr << " Use " << argv[0]
         << " fname [prt [assoc indexSz blockSz]]" << endl;
    exit(1);
```

```
  };
  if (argc==3 || argc==6) {
    istrstream ss(argv[2],10);
    ss >> outputEnabled;
    argv[2] = 0;
  }
  else
    outputEnabled=0;
  if (argc==6) {
    istrstream ss1(argv[3],10);
    ss1 >> assoc;
    argv[3] = 0;
    istrstream ss2(argv[4],10);
    ss2 >> indexSz;
    argv[4] = 0;
    istrstream ss3(argv[5],10);
    ss3 >> blockSz;
    argv[5] = 0;
  }
  char* fname = argv[1];
  argv[1] = 0;
  return new MySimApp(outputEnabled,assoc,indexSz,blockSz,fname);
}


void MySimApp::BuildModel()
{

  D(OpenSram,os,1,1,33,6);
  Const("1",33) >> os.wrEnable[0];
  Const("2",33) >> os.rdEnable[0];
  os.dataOut[0] >> os.dataIn[0];
  os.outState   >> os.inState;
  Const("1",1)  >> os.wrState;

  /*
    Memory sub system
  */

#ifdef USEGC
  gc_limits(0.20,0.30);
#endif

  D(MemIOSys,mem,assoc,indexSz,blockSz,20,100,3,name);
  MemIOSys::setIOManager(0xF,* new IntIO(cin,cout));
  /*
    Shared register file
  */

  D(RegisterFile,reg,1,3,5,32);

  /*
    Global stall/flow controller. The pipe operates as follows:
    0: Normal flow. All segments move on.
    1: Normal stall. All segments stay put.
    2: Hazard stall. IF & ID segments stall, EX, M & WB move
    3: Branch squash. IF & ID bubble, all segments move
    4: Jmp. No squash. All segements move. Jumps are handled in IF stage.
    5: Jr. Squash IF. All segments move. Jr is handled in ID stage

    Outputs:     IFIDMove EXMWBMove IDBubble IFBubble PCSource
    flow         1        1         0        0        from PC+4
    stall        0        0         0        0        dont care
    hazard       0        1         1        0        dont care
    branch       1        1         1        1        from EX
    jmp          1        1         0        0        from inst in IF
```

```
  jr          1       1        0       1       from ID

  Inputs:
  iMiss:    The icache lookup failed
  dMiss:    The dcache lookup failed
  hazard:   The ID stage detected a data hazard
  bcc:      A branch has been taken from EX stage.
  jr:       A jr has been taken from ID stage.
  ifOpcode: The opcode field from the instruction fetched from I$ in IF

*/
//                                        IFIDStall
//                                  ifOpcode |EXMWBStall
//                                  bcc|||||  ||IFIDMove
//                                  jr||||||  |||
//                             hazard|||||||  |||EXMWBMove
//                        holdAndWait||||||||  ||||IDBubble
//                                  |||||||||  |||||IFBubble
//                                  |||||||||  ||||||PCMuxCtrl
//                                  ||||^^^^^  ||||||^^^^
static char *stallCtrlSpec[]  = { "0--1-----  --1111----", // bcc
                                  "01-0-----  1--11-----", // hazard
                                  "0010-----  --11-1----", // jr
                                  "0000-----  --11------", // jmp/flow
                                  "1--------  11--------", // holdAndWait
                                  ---1-----  ---------1", // bcc
                                  --10-----  ------1---", // jr
                                  --0000001  --------1-", // jmp
                                  --001----  -------1--", // flow/hazard
                                  --00-1---  -------1--", // flow/hazard
                                  --00--1--  -------1--", // flow/hazard
                                  --00---1-  -------1--", // flow/hazard
                                  --00----0  -------1--"  // flow/hazard
};
/*
  Note that the PLA  is partitioned so that the pcmux control
  is independent on hit/miss/hazard status. This results in
  about 20% performance increase compared to non-partitioned
  control. The pcmux is still on the critical path, however
*/
D(PLA,stallCtrl,9,10,13,stallCtrlSpec);
INABSTRACTION& holdAndWait = stallCtrl.in[8];
INABSTRACTION& hazard      = stallCtrl.in[7];
INABSTRACTION& jr          = stallCtrl.in[6];
INABSTRACTION& bcc         = stallCtrl.in[5];
INABSTRACTION& ifOpcode    = stallCtrl.in.subset(0,5);
OUTABSTRACTION& IFIDStall  = stallCtrl.out[9];
OUTABSTRACTION& EXMWBStall = stallCtrl.out[8];
OUTABSTRACTION& IFIDMove   = stallCtrl.out[7];
OUTABSTRACTION& EXMWBMove  = stallCtrl.out[6];
OUTABSTRACTION& IDBubble   = stallCtrl.out[5];
OUTABSTRACTION& IFBubble   = stallCtrl.out[4];
IFIDStall  >> 0;
EXMWBStall >> 0;

// The PC multiplexor is positioned here as well...
D(FastMux,pcMux,4,30);
stallCtrl.out.subset(0,4) >>  pcMux.select;
INABSTRACTION& pcJrIn       = pcMux.in[3];
INABSTRACTION& pcFlowIn     = pcMux.in[2];
INABSTRACTION& pcJmpIn      = pcMux.in[1];
INABSTRACTION& pcBccIn      = pcMux.in[0];
```

```
/*****************************************************************
 * IF Stage. This is the PC, the PC incrementer and the cache.
 *****************************************************************/

// We're connecting to ID, so the latches between IF & ID goes first
D(PipeStageBoundary,IFID,IFIDMove);

// PC Section
D(WrEnFlipFlop,pc,30,0);         IFIDMove >> pc.wrEnable;
D(Adder,pcInc,30);
pcMux.out     >> pc.in; pc.out >> pcInc.inA >> mem.iAddrIn.subset(2,30);
Const("0",2)  >> mem.iAddrIn.subset(0,2);
Const("1",30) >> pcInc.inB;
Const("0",1)  >> pcInc.cIn;
pcInc.cOut    >> 0;
pcInc.sum     >> pcFlowIn   /*   >> IFIDnextPc.in */;

// ICache Access Section
"1 [1]"       >> mem.iRd;
mem.holdAndWait >> holdAndWait;

// Jmp & Jal execution
mem.iDataOut.subset(0,26) >> pcJmpIn.subset(0,26);
pc.out.subset(26,4)       >> pcJmpIn.subset(26,4);
mem.iDataOut.subset(27,5) >> ifOpcode;




/*****************************************************************
 * ID Stage. This is the decoder, the bypass networks,
 * the hazard detector and a nullifier mux.
 *****************************************************************/

D(MipsDecoder, decode);
decode.isJorJal >> 0;
IFBubble     >> IFID >> decode.invalid;
mem.iDataOut >> IFID >> decode.inst;
pcInc.sum    >> IFID >> decode.nextPc;
// The IFID flipflops are declared in the IF stage section.
// We're connecting to EX, so the latches between ID & EX goes first.
D(WrEnFlipFlop,IDEXopA,32,0);       EXMWBMove >> IDEXopA.wrEnable;
D(WrEnFlipFlop,IDEXopB,32,0);       EXMWBMove >> IDEXopB.wrEnable;
D(WrEnFlipFlop,IDEXopC,32,0);       EXMWBMove >> IDEXopC.wrEnable;
D(PipeStageBoundary,IDEX,EXMWBMove);

// EX/M/WB -> ID bypass paths. Inputs to these paths are:
// slot 5: constant/derived value from decode
// slot 4: result from EX stage
// slot 3: result from d$ in M stage
// slot 2: result from alu passed through stage
// slot 1: result from WB stage
// slot 0: result from register file readout (always valid)
// slot 0 & 5 are setup below. The other slots are setup by their
// respective sourcing stage. Note the clever trick used to choose
// between constants and register file readouts without seperate
// muxing!
/*
D(Bypass,opAby,6,32,5);             opAby.dataOut >> IDEXopA.in;
D(Bypass,opBby,6,32,5);             opBby.dataOut >> IDEXopB.in;
D(Bypass,opCby,6,32,5);             opCby.dataOut >> IDEXopC.in;
*/
D(Bypass,opAby,6,32,5);             opAby.dataOut >> IDEXopA.in;
```

```
D(Bypass,opBby,6,32,5);              opBby.dataOut >> IDEXopB.in;
D(Bypass,opCby,6,32,5);              opCby.dataOut >> IDEXopC.in;
D(NotGate,needValA);                    needValA.in << decode.needRegA;
D(NotGate,needValB);                    needValB.in << decode.needRegB;
D(NotGate,needValC);                    needValC.in << decode.needRegC;
opAby.dataIn[0] << reg.rdData[0];  opAby.validIn[0] << Const("1",1);
opBby.dataIn[0] << reg.rdData[1];  opBby.validIn[0] << Const("1",1);
opCby.dataIn[0] << reg.rdData[2];  opCby.validIn[0] << Const("1",1);
opAby.dataIn[5] << decode.value;   opAby.validIn[5] << needValA.out;
opBby.dataIn[5] << decode.value;   opBby.validIn[5] << needValB.out;
opCby.dataIn[5] << decode.value;   opCby.validIn[5] << needValC.out;
decode.opARegNr >> opAby.wrSelect[5] >> opAby.wrSelect[0]
                >> opAby.rdSelect    >> reg.rdSelect[0];
decode.opBRegNr >> opBby.wrSelect[5] >> opBby.wrSelect[0]
                >> opBby.rdSelect    >> reg.rdSelect[1];
decode.opCRegNr >> opCby.wrSelect[5] >> opCby.wrSelect[0]
                >> opCby.rdSelect    >> reg.rdSelect[2];

// Bubbling. This simply means passing a null instruction to next stage.
// We must avoid ld/st, branching and reg wb.
D(ZapBoundary,IDZap,IDBubble);

// Hazard detection. Note: Hazards on opC is ignored. They will be resolved
// by the LD/ST bypass in the next cycle instead.
D(QualifiedSearch,hazardMatch,2,5);
D(OrGate,anyMatch,2);
D(NotGate,holdOnUse);
D(AndGate,matchValid,3);
hazardMatch.valid << decode.needRegA + decode.needRegB;
//  hazardMatch.keys  << IFIDInst.out.subset(16,10);
// ^^^ Too slow to wait for decoder.. And we now the numbers anyway :-)
hazardMatch.keys  << decode.opARegNr + decode.opBRegNr;
anyMatch.in       << hazardMatch.hits;
decode.dstRegNr   >> IDEX >> hazardMatch.key;
decode.exDstValid >> IDZap >> IDEX >> holdOnUse.in;
decode.mDstValid  >> IDZap >> IDEX >> matchValid.in[2];
matchValid.in[0]  << anyMatch.out;
matchValid.in[1]  << holdOnUse.out;
matchValid.out    >> hazard;

// Jump Register
decode.isJr               >> jr;
opAby.dataOut.subset(2,30) >> pcJrIn;




/*****************************************************************
 * EX Stage. This is the ALU and dedicated branch circuitry for
 * BEQ and BNE, as well as the LD/ST bypass
 *****************************************************************/

// The IDEX flipflops are declared as part of the ID stage.
// We're connecting to M, so we'll declare the EX/M flipflops below
D(WrEnFlipFlop,EXMopC,32,0);            EXMWBMove >> EXMopC.wrEnable;
D(PipeStageBoundary,EXM,EXMWBMove);

// The ALU proper
D(ALU,alu);
IDEXopA.out                >> alu.inA;
IDEXopB.out                >> alu.inB;
decode.aluCtrl     >> IDZap >> IDEX >> alu.ctrl;
```

```
// Feeding the bypasses to ID
alu.res            >> opAby.dataIn[4]
                   >> opBby.dataIn[4]
                   >> opCby.dataIn[4];
decode.dstRegNr             >> IDEX >> opAby.wrSelect[4]
                                    >> opBby.wrSelect[4]
                                    >> opCby.wrSelect[4];
decode.exDstValid  >> IDZap >> IDEX >> opAby.validIn[4]
                                    >> opBby.validIn[4]
                                    >> opCby.validIn[4];

// Branching
//                             isNe
//                            isEq|
//                           isBeq||
//                          isBne||| bcc
//                              |||| |
static char* branchSpec[] = { "1--1 1",
                              -11- 1" };
D(PLA,branchCtrl,4,1,2,branchSpec);
IDEXopC.out.subset(2,30)    >> pcBccIn;
branchCtrl.out[0]           >> bcc;
decode.isBeq       >> IDZap >> IDEX >> branchCtrl.in[2];
decode.isBne       >> IDZap >> IDEX >> branchCtrl.in[3];

alu.relVec[5]               >> branchCtrl.in[1];
alu.relVec[0]               >> branchCtrl.in[0];

// The LD/ST bypass. slot 0: source from ID, slot 1: bypass from M
//  D(Bypass,ldstBy,2,32,5);
D(Bypass,ldstBy,2,32,5);
IDEXopC.out      >> ldstBy.dataIn[0];
Const("1",1)     >> ldstBy.validIn[0];
decode.opCRegNr           >> IDEX >> ldstBy.wrSelect[0] >> ldstBy.rdSelect;
decode.mDstValid >> IDZap >> IDEX >> EXM >> ldstBy.validIn[1];
decode.dstRegNr           >> IDEX >> EXM >> ldstBy.wrSelect[1];
EXMopC.in        << ldstBy.dataOut;
INABSTRACTION& ldstPass = ldstBy.dataIn[1];




/*********************************************************************
 * M stage. This is were the cache is accessed. For a ST opC delivers
 * the value to store. Result from EX stage is used to address memory.
 *********************************************************************/

// The M stage connects to the WB stage, so we'll start out with the
// MWB flipflops.
D(WrEnFlipFlop,MWBmDstValid,1,0);       EXMWBMove >> MWBmDstValid.wrEnable;
D(WrEnFlipFlop,MWBdstRegNr,5,0);        EXMWBMove >> MWBdstRegNr.wrEnable;
D(WrEnFlipFlop,MWBresult,32,0);         EXMWBMove >> MWBresult.wrEnable;
D(WrEnFlipFlop,MWBstop,1,0);            EXMWBMove >> MWBstop.wrEnable;
MWBstop.out >> 0;

// Memory access & result selection
D(Mux,resMux,1,32);
(decode.mCtrl >> IDZap >> IDEX >> EXM )[0] >> mem.dRd >> resMux.select;
(decode.mCtrl >> IDZap >> IDEX >> EXM )[1] >> mem.dWr;
//  EXMmCtrl.out[0] >> mem.dRd >> resMux.select;
//  EXMmCtrl.out[1] >> mem.dWr;
alu.res         >> EXM >> mem.dAddrIn
                       >> resMux.in[0];
EXMopC.out      >> mem.dDataIn;
```

```
mem.dDataOut    >> resMux.in[1];
resMux.out      >> MWBresult.in;

// Passing args on to WB
decode.dstRegNr          >> IDEX >> EXM >> MWBdstRegNr.in;
decode.mDstValid >> IDZap >> IDEX >> EXM >> MWBmDstValid.in;
decode.isStop    >> IDZap >> IDEX >> EXM >> MWBstop.in;

// Feeding the bypasses to ID
mem.dDataOut     >> opAby.dataIn[3]
                 >> opBby.dataIn[3]
                 >> opCby.dataIn[3];
decode.dstRegNr  >> IDEX >> EXM >> opAby.wrSelect[3]
                                >> opBby.wrSelect[3]
                                >> opCby.wrSelect[3];
decode.dstRegNr  >> IDEX >> EXM >> opAby.wrSelect[2]
                                >> opBby.wrSelect[2]
                                >> opCby.wrSelect[2];
(decode.mCtrl    >> IDZap >> IDEX >> EXM)[0] >> opAby.validIn[3]
                                             >> opBby.validIn[3]
                                             >> opCby.validIn[3];
decode.mDstValid >> IDZap >> IDEX >> EXM >> opAby.validIn[2]
                                         >> opBby.validIn[2]
                                         >> opCby.validIn[2];
alu.res >> EXM >> opAby.dataIn[2]
               >> opBby.dataIn[2]
               >> opCby.dataIn[2];

// Feeding the ld/st bypass to EX
ldstPass << resMux.out;

OUTABSTRACTION& EXMmDstValid = decode.mDstValid >> IDZap >> IDEX >> EXM;
OUTABSTRACTION& EXMdstRegNr  = decode.dstRegNr           >> IDEX >> EXM;
OUTABSTRACTION& EXMresult    = alu.res                           >> EXM;
OUTABSTRACTION& EXMwbStop    = decode.isStop    >> IDZap >> IDEX >> EXM;



/*****************************************************************
 * WB Stage. This is the final stage in the pipeline. The destination
 * register is written, and the result bypassed as well.
 *****************************************************************/

// Lets validate agains a trace if possible
D(ValidateWrite,validator,1,5,32);

// Writing to the register file:
MWBmDstValid.out >> reg.wrEnable[0] >> validator.wrEnable[0];
MWBdstRegNr.out  >> reg.wrSelect[0] >> validator.idxIn[0];
MWBresult.out    >> reg.wrData[0]   >> validator.dataIn[0];

// Feeding the bypass to ID
MWBresult.out    >> opAby.dataIn[1]
                 >> opBby.dataIn[1]
                 >> opCby.dataIn[1];
MWBdstRegNr.out  >> opAby.wrSelect[1]
                 >> opBby.wrSelect[1]
                 >> opCby.wrSelect[1];
MWBmDstValid.out >> opAby.validIn[1]
                 >> opBby.validIn[1]
                 >> opCby.validIn[1];
```

```
  /**************************************************************************
   *
   *                     END OF PIPE
   *
   **************************************************************************/

}

int MySimApp::Cycle(unsigned long currentCycle) {
  if (Stop.getData()!=Data(1,0)) return 0;
  return SimApp::Cycle(currentCycle);
}

#if 0

  SimSys.Prepare();
  if (outputEnabled > 3)
    SimSys.Inspect();

  int hStall = 0, memStall = 0;
  int numInsts = 0, numBcc = 0, numJr = 0;
  int numLds = 0, numSts = 0, numBccT = 0;
  int numCycles = 0, numJ = 0;
  int numRegOutputs = 0;
  int numMemOutputs = 0;
  do {
    SimSys.Cycle();
    numCycles++;

    if (outputEnabled == -1 || outputEnabled == -2) {
      if (EXMWBMove.getData().toLong()
          && mem.dWr.getData().toLong()) {
        numMemOutputs++;
        cout << numMemOutputs << " mem[" << mem.dAddrIn.getData()
             << "] <- " << mem.dDataIn.getData();
        if (outputEnabled == -2)
          cout << "   cycle: " << numCycles;
        cout << endl;
      }
    }
    if (outputEnabled == -3 || outputEnabled == -4) {
      if (EXMWBMove.getData().toLong()
          && reg.wrEnable[0].getData().toLong()) {
        numRegOutputs++;
        cout << numRegOutputs << " reg[" << reg.wrSelect[0].getData().toLong()
             << "] <- " << reg.wrData[0].getData();
        if (outputEnabled == -4)
          cout << "   cycle: " << numCycles;
        cout << endl;
      };
    };
    if (outputEnabled > 0) {
      if (EXMWBMove.getData().toLong()) {
        numInsts++;
        int exCtrl = branchCtrl.in.getData().toLong();
        int mCtrl  = (decode.mCtrl >> IDZap >> IDEX).getData().toLong();
        if ((exCtrl & (1<<2)) || (exCtrl & (1<<3)))  numBcc++;
        else if (mCtrl == 1) numLds++;
        else if (mCtrl == 2) numSts++;
        if (IFIDMove.getData().toLong()==0)
          { hStall++; numInsts--; }
        else
          {
            int pcS = pcMux.select.getData().toLong();
```

```
        if (pcS & 1)      { numBccT++; numInsts-=2; }
        else if (pcS & 2) { numJ++; }
        else if (pcS & 8) { numJr++; numInsts--; }
      };
  } else memStall++;
}
if (outputEnabled > 1) {
  /*
   * Generating output
   */
  if (outputEnabled > 2)
    cout << ------------------------------------------------------"
         << -----------------"
         << ----------------------------------------------------" << endl;

  if (IFIDMove.getData().toLong())
    if (pcMux.select.getData().toLong() & 9)
      cout << "  * Sqsh * ";
    else
      cout << "PC:" << pc.out.getData();
  else
    if (EXMWBMove.getData().toLong()==0)
      cout << "  * Miss * ";
    else
      cout << "  * Hzrd * ";
  if (IFBubble.getData().toLong())
    cout << " : <bubble>           ";
  else
    cout << " : " << disassemble(mem.iDataOut.getData());
  if (hazard.getData().toLong()==0
    && (IDBubble.getData().toLong()
        || decode.invalid.getData().toLong()))
    cout << "  ID: <bubble>           ";
  else
    cout << "  ID: " << disassemble(decode.inst.getData());
  cout << "  EX: ";
  int exCtrl = alu.ctrl.getData().toLong();
  int exmCtrl = (decode.mCtrl >> IDZap >> IDEX).getData().toLong();
  if (exCtrl==0
      && exmCtrl==0
      && (decode.isStop >> IDZap >> IDEX).getData()==0
      && (decode.mDstValid >> IDZap >> IDEX).getData()==0)
    cout << ≪bubble>      ";
  else {
    if ((decode.isStop >> IDZap >> IDEX).getData().toLong())
      cout << "STOP ";
    else if (exCtrl & (1<<4)) cout << "BEQ  ";
    else if (exCtrl & (1<<5)) cout << "BNE  ";
    else if (exmCtrl & 1) cout << "LW   ";
    else if (exmCtrl & 2) cout << "SW   ";
    else
      cout << "     ";
    if (exCtrl & (7<<4)) {
      if (bcc.getData().toLong())
        cout << "[t]  "; else cout << "[nt] ";
    }       else cout << "     ";
    switch (exCtrl & 0xF) {
    case 0: cout << "SLT "; break;
    case 1: cout << "OR  "; break;
    case 2: cout << "AND "; break;
    case 3: cout << "XOR "; break;
    case 4: cout << "ADD "; break;
    case 5: cout << "SUB "; break;
    case 6: cout << "SL  "; break;
    case 7: cout << "SR  "; break;
    default: cout << "****";
```

```
  };
};
cout << "  M: ";
int mmCtrl = (decode.mCtrl >> IDZap >> IDEX >> EXM).getData().toLong();
if (mmCtrl==0) {
  if (EXMmDstValid.getData().toLong())
    if (EXMdstRegNr.getData().toLong() > 9)
      cout << "reg[" << EXMdstRegNr.getData().toLong() << "] = "
           << EXMresult.getData();
    else
      cout << "reg[ " << EXMdstRegNr.getData().toLong() << "] = "
           << EXMresult.getData();
  else
    cout << "                 ";
}
if (mmCtrl==1) cout << "LW  ";
if (mmCtrl==2) cout << "SW  ";
if (mmCtrl!=0)
  cout << EXMresult.getData() << "      ";
cout << "     WB: ";
if (MWBmDstValid.out.getData().toLong())
  if (MWBdstRegNr.out.getData().toLong() > 9)
    cout << "reg[" << MWBdstRegNr.out.getData().toLong() << "] = "
         << MWBresult.out.getData();
  else
    cout << "reg[ " << MWBdstRegNr.out.getData().toLong() << "] = "
         << MWBresult.out.getData();
cout << endl;

if (outputEnabled>2) {
  /*
    Line 2-4
  */
  cout << "              ";
  if (mem.holdAndWait.getData().toLong())
    cout << "hold ";
  else
    cout << "run  ";

  cout << "                ID: ";
  prtReg("opA: ",decode.needRegA.getData(),decode.opARegNr.getData());
  prtReg("  dst: ",decode.mDstValid.getData(),decode.dstRegNr.getData());
  cout << "     EX: ";
  cout << "a: " << IDEXopA.out.getData();
  cout << "     M:                        WB:" << endl;

  cout << "                                    ID: ";
  prtReg("opB: ",decode.needRegB.getData(),decode.opBRegNr.getData());
  prtReg("  opC: ",decode.needRegC.getData(),decode.opBRegNr.getData());
  cout << "     EX: ";
  cout << "b: " << IDEXopB.out.getData();
  cout << "     M: ";
  if (mmCtrl==1) cout << "loads   " << mem.dDataOut.getData();
  else if (mmCtrl==2) cout << "stores  " << mem.dDataIn.getData();
  else cout << "                 ";
  cout << "       WB:" << endl;

  cout << "                                    ID: ";
  cout << "val: " << decode.value.getData();
  cout << "        EX: ";
  cout << "c: " << IDEXopC.out.getData();
  cout << "     M: ";
  if (mmCtrl) {
    if (mem.holdAndWait.getData().toLong())
      cout << "hold ";
    else
```

```
          cout << "run  ";
        } else
          cout << "     ";
        cout << "                 WB:" << endl;
      };
    };
  } while (EXMwbStop.getData()==0);
  if (outputEnabled > 0) {
    cout << endl;
    cout << "Execution statistics:" << endl;
    cout << "  Instructions: " << numInsts << endl;
    cout << "    Bcc:    " << numBcc << " (" << numBccT << " taken)" << endl;
    cout << "    Jr:     " << numJr << endl;
    cout << "    J/Jal:  " << numJ << endl;
    cout << "    Loads:  " << numLds << endl;
    cout << "    Stores: " << numSts << endl;
    cout << "    Others: " << numInsts-numBcc-numLds-numSts-numJr-numJ << endl;
    cout << "  Cycles: " << numCycles << endl;
    cout << "    Stalled for Memory: " << memStall << endl;
    cout << "    Data Hazard Stalls: " << hStall << endl;
    cout << "  CPI: " << numCycles*1.0/numInsts << endl;
    cout << endl;
  };

  SimSys.Done();
};

void prtReg(char* s, Data valid, Data nr)
{
  cout << s;
  if (valid.toLong()==0) cout << --";
  else cout << nr;
};
#endif
```

# Bilag C

# Vejledende løsninger i Hades

Dette bilag indeholder løsninger til G- og K1-opgaven anno 1999 programmeret i Hades, og disse er lavet ud fra løsningerne i bilag B. Selve hovedprogrammerne for de to opgaver er skrevet stort set af herfra, da det ville tage (uforholdsmæssigt) lang tid at gennemskue, præcis hvilket kredsløb de simulerer. Derimod er Hadesprogrammerne for de andre byggeklodser lavet ud fra et kredsløbsdiagram, og de er noget simplere end de oprindelige SimSysprogrammer. Der er nogle få afvigelser i forhold til SimSysløsningerne (markeret med `// NB!` i programmerne), men der ændres ikke på den grundlæggende funktionalitet. Programmerne er ikke kommenteret, da kommentarer kan læses i de oprindelige programmer.

Som i sidste bilag indeholder afsnit C.1 de byggeklodser, der bruges i begge løsninger, mens afsnit C.2 indeholder hovedprogrammet for G1-opgaven og afsnit C.3 en ekstra sammensat byggeklods samt hovedprogrammet for K-opgaven.

## C.1 Grundlæggende byggeklodser

**mipsDecoder**

```
mipsDecoder (invalid 1 inst 32 nextPc 30)
         >> (exDstValid mDstValid 1 dstRegNr opARegNr opBRegNr opCRegNr 5
             needRegA needRegB needRegC 1 value 32 aluCtrl 4 mCtrl 2
             isJorJal isStop isBne isBeq isJr 1)
  0'2' nextPc 0'2' inst[..15]+[15]*14 0'1'
    >> bTargetAdd'32'[cOut,sum] nextPc 0'2' inst[..15]+[15]*16+[..15] 0'32'
       inst[..15] []dec[18..22]
    >> valmux'fastmux 5 32' >> value;
  inst[16..25] >> (opBRegNr , opCRegNr) opARegNr;
  0x1F'10' inst[11..20] dec[14..17] >> dstMux'fastmux 4 5'
    >> dstRegNr , dstOr'5' >> mDstValid , '1' []dec[9] >> exDstAnd'2' >> exDstValid;
  invalid inst[..5,26..] >> dec'dspec'[..8,10..13,23]
    >> mCtrl aluCtrl isBeq isBne isJr needRegC needRegB needRegA isJorJal isStop;
  dspec = " 0000001000000 -------1---11-1----1----, /*add R*/
            0000001000100 -------1---11-1----1-1--, /*sub R*/
            0000001010100 -------1---11-1---------, /*slt R*/
            0000001001000 --1----1---11-1-----1---, /*and R*/
            0000001001010 --1----1---11-1------1--, /*or R */
            0000001001100 --1----1---11-1-----11--, /*xor R*/
            0000000010000 ---------1-1---1--------, /*jr */
            100011------0 ---1--1----1-------1---1, /*lw*/
            101011------0 ---1-----1-1-1-----1--1-, /*sw*/
            001000------0 ---1--1----1--1----1----, /*add I*/
            001010------0 ---1--1----1--1---------, /*slt I*/
            001100------0 --1---1----1--1-----1---, /*and I*/
```

```
            001101------0 --1---1----1--1------1--, /*or I*/
            001110------0 --1---1----1--1-----11--, /*xor I*/
            001111------0 -1----1-------1------1--, /*lui I*/
            000100------0 -----1---1-11----1------, /*beq*/
            000101------0 -----1---1-11---1-------, /*bne*/
            000010------0 ---------11-------------, /*j */
            000011------0 ----1---1-1---1------1--, /*jal*/
            111111------0 1-----------------------, /*stop*/
            ------------1 ------------------------ "/*inv*/
end
```

### shifter

```
// NB! isLeft er en boolean. I SimSysprogrammet er det en enumereret type.
shifter (in 2^shft amount shft, shft isLeft) >> (out 2^shft)
  if isLeft then
    shifter'muxarray shft 1 2^shft ; out 0 out[..2^shft-2^index-1] >> in'
  else  // rightshift
    shifter'muxarray shft 1 2^shft ; out*1+[2^index..] 0 >> in'
  ;
  in amount >> shifter >> out
end
```

### relop

```
relop (inA inB 32) >> (relvec 10)
  (inA inB)/2 >> xorarray'32 2' >> firstdiff'prioritizer 32';
  inB inA firstdiff[out]*2+[31,32]
    >> /2[..31;32..](muxB'fastmux 32 1' muxA'fastmux 32 1') '2'
    >> pla'plaspec';
  (firstdiff[valid] pla)*2 >> '5' notarray'5' >> relvec;
  plaspec = " 1001 -1-1,1010 1-1-,1101 -11-,1110 1--1 "
end
```

### alu

```
// NB! withshift bruges til at markere, om der skal skiftes. I SimSysprogrammet
// saettes den i selve programmet og er altsaa ikke en parameter til aluen.
alu (inA inB 32 ctrl 4, withshift) >> (res 32 relvec 10)
  if withshift then
    shftsaverPlane = " 0110 1-,0111 -1 ";
    ctrl >> (shftsaver'shftsaverPlane'[1]*37+[0]*37
             (inA inB[..4])*2)/2[..36;37..]
      >> /37enableLshft'andarray 37 2' /37enableRshft'andarray 37 2'
      >> lshifter'5 true' rshifter'5 false'
      >> [6*32..8*32-1]resmux
  else 0 >> [6*32..8*32-1]resmux
  ;
  inA inB >> relop*1+[1] 0'31' ctrl
    >> relvec [..31,select]resmux'3 32' >> res;
  inA inB >> /2orarray'32 2' , /2andarray'32 2' , /2xorarray'32 2'
    , [..63]add[out] , ('32' notarray'32' >> [..63]sub'add')
    >> [32..6*32-1]resmux;
  0'1' 1'1' >> [cIn]add[cOut] [cIn]sub[cOut] >> _
end
```

## C.2 Løsning til G-opgaven, 1999

```
\\ NB! FILE er navnet paa fil og kan ikke som i SimSys gives som inddata til main
main (withShift)
  (dec'mipsdecoder'[value]*3 reg'registerfile 1 3 5 32')/2[..31;32..63;64..]
    >> [in]valAmux'1 32' [in]valBmux'1 32' [in]valCmux'1 32'[] []dec[aluctrl]
```

```
    >> alu'withshift';
  dec[22..24] >> [64]valAmux [64]valBmux [64]valCmux;
  pc'ff 32 0' 4'32' 0'1' >> pcinc'add 32';
  pc 1'1' alu[res] dec[mctrl] valCmux >> mem'idealMemIOSys 20 1 FILE';
  dec[1..21] resmux >> [5,..4,38..,6..37]reg'registerfile 1 3 5 32';
  0'1' mem[..31] pcinc[2..31] >> dec;
  dec[dstregnr] resmux'1 32' dec[1] >> val'validatewrite 1 5 32';
  alu[res] mem[32..] dec[61] >> resmux;
  dec[63,65..67] alu[32,37] >> [1,5,3,0,4,2]ctrlPla'ctrl'
    >> [select]pcMux'fastmux 4 32';
  pcinc[cOut,sum] dec[value] valAmux 0'2' mem[..25] pc[28..]
    >> _'1' [in]pcMux >> pc;
  dec[0,64] >> _;
  ctrl = " 11---- --1-,10---- ---1,--11-- --1-,--00-- ---1,
           ----1- 1---,-----1 -1--,0-0-00 ---1 "
end
```

## C.3 Løsning til K1-opgaven, 1999

### bypass

```
bypass (dataIn sz*dataSz wrSelect sz*idSz validIn sz rdSelect idSz
       , sz dataSz idSz) >> (dataOut dataSz)
  wrSelect rdSelect validIn >> matcher'qualifiedSearch sz idSz'
    >> dataIn prioritizer'sz' >> fastmux'sz dataSz' _'1' >> dataOut
end
```

### mips

```
\\ NB! FILE er navnet paa fil og kan ikke som i SimSys gives som inddata til main
main (assoc indexsz blocksz)
  mem'memIOsys assoc indexSz blockSz 20 100 3 FILE' reg'registerfile 1 3 5 32';
  stallCtrlSpec = " 0--1----- --1111----,01-0----- 1--11-----,
                    0010----- --11-1----,0000----- --11------,
                    1-------- 11--------,---1----- ---------1,
                    --10----- ------1---,--0000001 --------1-,
                    --001---- -------1--,--00-1--- -------1--,
                    --00--1-- -------1--,--00---1- -------1--,
                    --00----0 -------1-- ";
  pcmux'fastmux 4 30' stallctrl'stallctrlSpec'[7] >> pc'wrenff 30 0' 1'30' 0'1'
    >> pcinc'adder 30'[cout] >> _'1';
  IDEXopc[2..] mem[..25] pc[26..29] pcinc[sum] opaby[2..31] stallctrl[..3] >> pcmux;
  decode[85..87] >> 7'3' needValA'not' needValB'not' needValC'not'
    >> /3[222;227](opAby'bypass 6 32 5' opBby'bypass 6 32 5' opCby'bypass 6 32 5')
       []stallCtrl[6]*3
    >> /3[..31;32](IDEXopA'wrenff 32 0' IDEXopB'wrenff 32 0' IDEXopC'wrenff 32 0');
  reg[rddata] decode[value]*3+[70..84]*4
    >> /3[..31;160..191;217..221;192..196;228..232](opAby opBby opCby) [rdselect]reg;
  decode[dstregnr] >> IDEX'pipestage stallctrl[6]'
    >> [key]hazardMatch'qualifiedSearch 2 5';
  decode[70..79,85,86] >> [..9,16..]hazardMatch >> anymatch'or 2'
    >> [0]matchvalid'and 3' >> [7]stallctrl;
  decode[63,64] >> IDzap'stallctrl[5]' >> IDEX >> holdOnUse'not' '1'
    >> [1,2]matchvalid;
  decode[isjr] >> [6]stallctrl[8,9] >> _;
  decode[aluctrl,exDstValid] >> IDzap >> IDEX
    >> [ctrl]alu ([226]opAby , [226]opBby , [226]opCby);
  IDEXopA IDEXopB >> [..63]alu[32,37]+[res]*3
    >> [0,1]branchCtrl'brachspec' /3[128..159](opAby opBby opCby);
  branchspec = " 1--1 1,-11- 1";
  decode[dstregnr]*3 >> IDEX >> /3[212..216](opAby opBby opCby);
  decode[isBeq,isBne] >> IDzap >> IDEX >> [2,3]branchCtrl >> [5]stallctrl;
  0'2' pc 1'1' >> [..32]mem[27..31,64] >> [..4,8]stallctrl;
```

```
  stallctrl[4] mem[mdataOut] pcinc[sum] >> IFID'pipestage stallctrl[7]'
    >> [invalid,inst,nextPc]decode'mipsDecoder'[isJorJal] >> _;
  IDEXopC 1'1' >> [..31,74]ldstby'bypass 2 32 5' []stallctrl[6]
    >> EXMopC'wrenff 32 0';
  decode[opCRegNr]*2 >> IDEX >> [64..68,76]ldstby;
  decode[mDstValid] >> IDzap >> IDEX >> EXM >> [75]ldstBy;
  decode[dstregnr] >> IDEX >> EXM'pipestage stallctrl[6]' >> [69..73]ldstBy;
  stallctrl[6] >> [1]MWBstop , [5]MWBdstregnr , [32]MWBresult , [1]MWBmdstvalid
    >> _'1' ([..4,6..37,5]reg , validator'validateWrite 1 5 32');
  decode[dstregnr]*7 >> IDEX >> EXM
    >> [in]MWBdstregnr'wrenff 5 0' /3[202..211](opAby opBby opCby);
  decode[mDstValid,isStop] >> IDzap >> IDEX >> EXM
    >> [in]MWBmdstvalid'wrenff 1 0' [in]MWBstop'wrenff 1 0';
  mem[ddataOut] MWBresult'wrenff 32 0' MWBdstregnr MWBdstvalid
    >> /3[96..127;32..63;197..201;223]{opAby opBby opCby);
  alu[res] >> EXM >> ([daddrIn]mem[ddataOut] , '32') >> [in]resmux'1 32'
    >> [in]MWBresult , [32..63]ldstBy;
  decode[mctrl] >> IDzap >> IDEX >> EXM >> ([dRd]mem , [32]resmux) [dWr]mem;
  EXMopC >> [ddataIn]mem;
  alu[res]*3 >> EXM >> /3[64..95](opAby opBby opCby);
  decode[64,124]*3 >> IDzap >> IDEX >> EXM >> /3[224,225](opAby opBby opCby)
end
```

# Bilag D

# Laboratorieøvelse 2 og 3 i Hades

Laboratorieøvelserne [Sim] i Dat1E bruges som en introduktion til SimSys og indeholder opgaver, der lærer de studerende at bruge køretidssystemet samt at programmere i SimSys. Kun øvelse 2 og 3 indeholder programmeringsopgaver, så kun disse opgaver medtages i dette bilag. Opgaver og delspørgsmål, der ikke er programmeringsøvelser, overspringes. Herved dokumenteres, at laboratorieøvelserne også kan løses ved brug af Hades. Den relevante del af opgaveteksten til de løste opgaver er medtaget.

### 2.1

Byg et kredsløb der hver cyklus adderer konstanten 42 (hex) til værdien i et register.

```
main ()
  flipflop’64 0’ 0x42’64’ 0’1’ >> adder’64’ >> flipflop _’1’
end
```

### 2.2

I det følgende modificerer vi vores additionskredsløb fra øvelse 2.1 så bredden af datavejen (register og adder) kan angives som parameter på kørselstidspunktet.

```
// NB! Parametre skal angives paa oversaettelsestidspunktet i Hades
main (bit)
  flipflop’bit 0’ 0x42’bit’ 0’1’ >> adder’bit’ >> flipflop _’1’
end
```

### 2.4

Lav derpå et nyt program [...] ved at kopiere [...] [programmet fra 2.2] og anvende en ”ripple carry adder”.

```
main (bit)
  flipflop’bit 0’ 0x42’bit’ 0’1’ >> serialadder’bit’ >> flipflop _’1’
end
```

**2.5**

Byg [...] [et 4-bit nedtællingskredsløb] af et fire bit register (type FlipFlop) og en PLA med fire indgående bit og fire udgående bit.

```
main ()
  ff'4 0' >> pla'plan' >> ff;
  plan = " ---0 ---1,--00 --1-,--11 --1-,-000 -1--,-11- -1--,
           -101 -1--,0000 1---,11-- 1---,101- 1---,1001 1--- "
end
```

**3.1**

Implementer løsningen af opgave S-95-2 i Kursusbog bind 1, delopgave 2.3.

```
main ()
  ff'3 0' >> x0 , x1 , x2 >> ff
end

x0 (in 3) >> (out 1)
  in >> not _'2' >> out
end

x1 (in 3) >> (out 1)
  in >> xor'2' _'1' >> out
end

x2 (in 3) >> (out 1)
  in >> and'2' '1' >> xor'2' >> out
end
```

**3.2**

[...] lav[...] en inkrementer i stedet for en dekrementer. [...] inkrementeren [skal laves] som en selvstændig byggeklods hvis størrelse kan angives som parameter ved instantieringen.

```
main (bit)
  ff'bit 0' >> inc'bit' >> ff
end

inc (in bit, bit) >> (out bit)
  in 1'bit' 0'1' >> adder'bit' >> out
end
```

**3.3**

[...] lav[...] en inkrementer der sparker røv.

Du observerer at for enhver bit position j i det tal der skal tælles op gælder at netop denne bit skal inverteres, hvis samtlige mindre betydende bits alle er sat. Du ser straks en løsning med en række af And-porte og en række af Xor-porte. Bits 0..j-1 føres til And-port j. Resultatet fra And-port j føres til Xor port j. Bit j i inddata føres også til Xor port j.

Rækken af Xor-porte kan konstrueres med den indbyggede byggeklods BWXorArray. Rækken af And-porte har forskelligt antal indgående bits. And-port 4 har 3 indgående bits, And-port j har j-1 indgående bits.

```
inc (in bit, bit) >> (out bit)
  stikind = in;  // noedvendig
  andarray'bit-1 index+1 ; stikind[0..index+1] >> in';
  in andarray >> not xorarray'bit-1 2' >> out
end
```

### 3.5

[Lav en inkrementer, der opfylder:]

- En 256-bit inkrementer må maximalt bruge 20.000 transistorer.
- Samme 256-bit inkrementer skal kunne bruges i kredsløb med en cyklustid på 1.0 ns (1GHz) og kunne tælle op hver cyklus.

Det har ikke været muligt at opdrive en vejledende løsning til denne opgave eller på kort tid at lave en løsning selv. Implementering af oversætteren er blevet prioriteret højere end løsning af denne opgave, så derfor er den ikke medtaget her.

### 3.6, 3.7, 3.8, 3.9

Disse opgaver går ud på at bygge dele af det kredsløb, der bruges i G- og K-opgaverne. Løsningerne til disse, programmeret i Hades, er i bilag C og er derfor udeladt her.

# Bilag E

# Program

Først gives kildekoden for lexer og parser og derefter for resten af oversætteren. Rækkefølgen af sidstnævnte filer er den samme som i afsnit 9.2.

### lexer.lex

```
{
open  Lexing;

val currentLine = ref 1
val lineStartPos = ref [0]
val planL = ref [[]]
val kommentarSt = ref (0,0)
val kommentarDybde = ref 0

(* efterSplit bruges til at markere, om [ eller ' eller / kommer efter et
 * splittegn (>> eller ) eller ,) eller indeni en maengde af komponenter. I
 * foerstnaevnte tilfaelde skal [ hhv. ' hhv. / laeses som VKPARM hhv.
 * ANFOERM hhv. DIVM og i sidstnaevnte som VKPAR hhv. ANFOER hhv. DIV.
 * efterSplit starter med at vaere false, da jeg ikke tillader bitafledninger
 * i starten af en linie, og da jeg ikke tillader gennemloebsvariable i
 * starten af en saetning. *)

  val efterSplit = ref false

 exception fejl_leksikalsk of syntaks.streng  (* (message, (line, column)) *)
 exception fejlIntern_lexer

fun getPos lexbuf = getLineCol (getLexemeStart lexbuf)
                               (!currentLine)
                               (!lineStartPos)

and getLineCol pos line (p1::ps) =
       if pos>=p1 then (line, pos-p1)
       else getLineCol pos (line-1) ps
  | getLineCol pos line [] =
       raise fejlIntern_lexer

     fun lexerFejl lexbuf s = raise fejl_leksikalsk (s, getPos lexbuf)

     fun keyword (s, pos) =
     let val str = String.map Char.toLower s in
     case str of
         "main"         => parser.MAIN pos
       | "end"          => parser.END pos
       | "if"           => parser.IF pos
       | "then"         => parser.THEN pos
```

```
      | "else"         => parser.ELSE pos
      | "true"         => parser.TRUE pos
      | "false"        => parser.FALSE pos
      | "mod"          => parser.MOD pos
      | _              => (efterSplit:=false;parser.STRENG (str,pos))
    end;
 }

let ws = [‘ ‘ ‘\t‘ ‘\r‘]
let nl = [‘\n‘ ‘\012‘]

rule token = parse
    "//" [^‘\n‘ ‘\012‘]*{ token lexbuf }  (* comment *)
  | "/*"               { kommentarSt := getPos lexbuf;
                         kommentarDybde := 1;
                         kommentarSkip lexbuf; token lexbuf }
  | ‘(‘                { efterSplit:=true;parser.VPAR(getPos lexbuf) }
  | ‘)‘                { parser.HPAR(getPos lexbuf) }
  | ws ‘[‘             { efterSplit:=false;parser.VKPARM(getPos lexbuf) }
  | nl ‘[‘             { currentLine := !currentLine+1;
                         lineStartPos :=  getLexemeStart lexbuf
                                       :: !lineStartPos;
                         efterSplit:=false;parser.VKPARM(getPos lexbuf)}
  | ‘[‘                { if !efterSplit then
                           (efterSplit:=false;parser.VKPARM(getPos lexbuf))
                         else parser.VKPAR(getPos lexbuf)}
  | ‘]‘                { parser.HKPAR (getPos lexbuf) }
  | ws ‘’‘             { efterSplit:=false;parser.ANFOERM (getPos lexbuf) }
  | nl ‘’‘             { currentLine := !currentLine+1;
                         lineStartPos :=  getLexemeStart lexbuf
                                       :: !lineStartPos;
                         efterSplit:=false;parser.ANFOERM (getPos lexbuf) }
  | ‘’‘                { if !efterSplit then
                           (efterSplit:=false;parser.ANFOERM(getPos lexbuf))
                         else parser.ANFOER (getPos lexbuf) }
  | ‘"‘                { planL:=[[]]; plan lexbuf }
  | ‘,‘                { efterSplit:=true ; parser.KOMMA (getPos lexbuf) }
  | ">>"               { efterSplit:=true ; parser.PIL (getPos lexbuf) }
  | ".."               { parser.TIL (getPos lexbuf) }
  | ‘;‘                { parser.SEMI (getPos lexbuf) }
  | ‘+‘                { parser.PLUS (getPos lexbuf) }
  | ‘-‘                { parser.MINUS (getPos lexbuf) }
  | ‘*‘                { parser.GANGE (getPos lexbuf) }
  | nl "//" [^‘\n‘ ‘\012‘]*
                       { currentLine := !currentLine+1;
                         lineStartPos :=  getLexemeStart lexbuf
                                       :: !lineStartPos;
                         token lexbuf }  (* comment *)
  | ws "//" [^‘\n‘ ‘\012‘]*
                       { token lexbuf }  (* comment *)
  | ws "/*"            { kommentarSt := getPos lexbuf;
                         kommentarDybde := 1;
                         kommentarSkip lexbuf; token lexbuf }
  | nl "/*"            { currentLine := !currentLine+1;
                         lineStartPos :=  getLexemeStart lexbuf
                                       :: !lineStartPos;
                         kommentarSt := getPos lexbuf;
                         kommentarDybde := 1;
                         kommentarSkip lexbuf; token lexbuf }
  | ws ‘/‘             { efterSplit:=false;parser.DIVM(getPos lexbuf) }
  | nl ‘/‘             { currentLine := !currentLine+1;
                         lineStartPos :=  getLexemeStart lexbuf
                                       :: !lineStartPos;
                         efterSplit:=false;parser.DIVM(getPos lexbuf) }
  | ‘/‘                { if !efterSplit then
                           (efterSplit:=false;parser.DIVM(getPos lexbuf))
```

```
                        else parser.DIV(getPos lexbuf) }
  | ‘^‘                 { parser.HAT (getPos lexbuf) }
  | "="                 { parser.LIG (getPos lexbuf) }
  | ‘<‘                 { parser.LT (getPos lexbuf) }
  | "<="                { parser.LEQ (getPos lexbuf) }
  | ">="                { parser.GEQ (getPos lexbuf) }
  | ">"                 { parser.GT (getPos lexbuf) }
  | "_"                 { efterSplit:=false;parser.AFLOEB (getPos lexbuf) }
  | "{o}"               { efterSplit:=false;parser.UD (getPos lexbuf) }
  | "0x"([‘0‘-‘9‘] | [‘a‘-‘f‘] | [‘A‘-‘F‘])+
                        { efterSplit:=false;parser.HEX(
                            String.map Char.toLower
                                 (String.extract(getLexeme lexbuf,2,NONE)),
                            getPos lexbuf) }
  | [‘0‘-‘9‘]+          { case Int.fromString (getLexeme lexbuf) of
                            NONE   => lexerFejl lexbuf "Bad integer"
                          | SOME i => (efterSplit:=false;parser.TAL(i,getPos lexbuf))}
  | ([‘a‘-‘z‘] | [‘A‘-‘Z‘]) ([‘a‘-‘z‘] | [‘A‘-‘Z‘] | [‘0‘-‘9‘] | "_")*
                        { keyword (getLexeme lexbuf, getPos lexbuf) }
  | ws                  { token lexbuf }
  | nl                  { currentLine := !currentLine+1;
                          lineStartPos :=  getLexemeStart lexbuf
                                        :: !lineStartPos;
                          token lexbuf }
  | eof                 { parser.EOF (getPos lexbuf) }
  | _                   { lexerFejl lexbuf "Illegal symbol in input" }
  and plan = parse  (* BEMAERK: planen er bagvendt *)
    (‘0‘|‘1‘|‘-‘)+      { planL :=
                          ((getLexeme lexbuf, getPos lexbuf)::hd(!planL))::tl(!planL);
                          plan lexbuf }
  | ‘,‘                 { planL := []::(!planL); plan lexbuf }
  | ws                  { plan lexbuf }
  | nl                  { currentLine := !currentLine+1;
                          lineStartPos :=  getLexemeStart lexbuf
                                        :: !lineStartPos;
                          plan lexbuf }
  | ‘"‘                 { parser.PLANL(!planL) }
  | _                   { lexerFejl lexbuf "illegal symbol in plan" }
and kommentarSkip = parse
    "*/"                { kommentarDybde := !kommentarDybde-1;
                          if !kommentarDybde = 0 then
                            ()
                          else kommentarSkip lexbuf }
  | "/*"                { kommentarDybde := !kommentarDybde+1;
                          kommentarSkip lexbuf }
  | eof                 { raise fejl_leksikalsk("comment not terminated",!kommentarSt) }
  | nl                  { currentLine := !currentLine+1;
                          lineStartPos :=  getLexemeStart lexbuf
                                        :: !lineStartPos;
                          kommentarSkip lexbuf }
  | _                   { kommentarSkip lexbuf }

;
```

### parser.grm

```
%token <syntaks.pos> MAIN END EOF SEMI
%token <syntaks.pos> VPAR HPAR VKPAR VKPARM HKPAR ANFOER ANFOERM
%token <syntaks.pos> KOMMA PIL LIG TIL  // PIL = >>, TIL = ..
%token <syntaks.pos> PLUS MINUS GANGE DIV DIVM MOD HAT LT LEQ GT GEQ
%token <syntaks.pos> IF THEN ELSE TRUE FALSE
%token <syntaks.pos> AFLOEB UD  // UD = {o}
%token <syntaks.streng> STRENG HEX
%token <syntaks.streng list list> PLANL
%token <int * syntaks.pos> TAL
```

```
%left PIL
%left KOMMA
%left PLUS MINUS
%left GANGE DIV DIVM MOD
%right HAT

%start program
%type <syntaks.programblok list> program
%type <syntaks.programblok list> kompdefs
%type <syntaks.programblok> kompdef
%type <syntaks.streng list> kommaStrenge

%type <syntaks.saetning list> krop
%type <syntaks.saetning> saetning

%type <syntaks.bitvektor> bitvektor
%type <syntaks.bitvektor> bitvUdenPil
%type <syntaks.bitvektor> komponenter
%type <syntaks.bitvektor> komponent

%type <syntaks.typen> type
%type <syntaks.typen> typeEvt
%type <syntaks.udtryk> stoerrelseEvt
%type <(syntaks.bitvektor * syntaks.bitvektor) list> forbindelser
%type <syntaks.bitvektor> udledninger
%type <syntaks.bitvektor> udledning
%type <syntaks.bitvektor> indledninger
%type <syntaks.bitvektor> indl
%type <syntaks.bitvektor> indledning

%type <syntaks.bitafled list> bitafledV
%type <syntaks.bitafled list> bitafledVEvt
%type <syntaks.bitmoenster list> bitV
%type <syntaks.bitafled list> bitafledH
%type <syntaks.bitafled list> bitafledHEvt
%type <syntaks.bitmoenster list> bitH
%type <syntaks.bitmoenster list> bit
%type <syntaks.bitmoenster list> bitEvt
%type <(syntaks.udtryk * syntaks.udtryk) list> bitmoenster
%type <(syntaks.udtryk * syntaks.udtryk)> bitm
%type <(syntaks.udtryk * syntaks.udtryk) list list> bitmFelt
%type <(syntaks.udtryk * syntaks.udtryk) list list> bitmFeltEvt

%type <syntaks.streng list> strenge
%type <syntaks.streng list> strengeEvt
%type <syntaks.udtryk> udtryk
%type <syntaks.udtryk> udtrykUS
%type <syntaks.udtryk> udtrykEvt
%type <syntaks.udtryk list> udtrykFlere
%type <syntaks.udtryk list> udtrykFlereEvt
%type <syntaks.udtryk> udtrykba
%type <syntaks.udtrykmBool list> udtrykMedBoolFlere
%type <syntaks.udtrykmBool> udtrykMedBool
%type <syntaks.udtrykmBool> udtrykMedBoolUS
%type <syntaks.udtrykBool> udtrykBool

%type <unit> ping

%%

// Alle strenge er med smaa bogstaver.

// ***************** Programblokke *****************

program:
          kompdefs EOF                                  {$1}
```

```
kompdefs:
                                                    {[]}
        | kompdef kompdefs                          {$1::$2}

// Et Hadesprogram kan spredes over mere end en fil, og da disse parses
// separat, er der ikke noedvendigvis et hovedprogram i et "program"

kompdef:
          MAIN VPAR strengeEvt HPAR krop END        {syntaks.MAIN($3,$5)}
        | STRENG VPAR udtrykFlereEvt kommaStrenge HPAR
            PIL VPAR udtrykFlereEvt HPAR
            krop END                                {syntaks.KOMPDEF
                                                     ($1,$3,$4,$8,$10)}

kommaStrenge:
                                                    {[]}
        | KOMMA strenge                             {$2}


// ***************** Krop *****************

krop:
          saetning                                  {[$1]}
        | saetning SEMI krop                        {$1::$3}

saetning:  /* BEMAERK: planer er bagvendte */
          STRENG LIG bitvUdenPil                    {syntaks.VAR($1,$3)}
// ??? fjern ogsaa ,?
        | STRENG LIG PLANL                          {syntaks.PLAN($1,$3)}
        | bitvektor                                 {syntaks.SAETNING($1)}
        | IF udtrykBool THEN krop ELSE krop END     {syntaks.IF($2,$4,$6)}


// ***************** Bitvektorer *****************

bitvektor:
          bitvektor PIL bitvektor                   {syntaks.FORBIND($1,$3)}
        | bitvUdenPil                               {$1}

bitvUdenPil:
          bitvUdenPil KOMMA bitvUdenPil             {syntaks.SPLIT($1,$3)}
        | komponenter                               {$1}

komponenter:
          komponent                                 {$1}
        | komponent komponenter                     {syntaks.SEKV($1,$2)}

komponent:
          ANFOERM udtryk ping                       {syntaks.GENNEMLOEB($2,$1)}
        | TAL stoerrelseEvt                         {syntaks.KONST(
                                                       konst.talTilKonst(#1 $1),
                                                       #2 $1, $2)}
        | HEX stoerrelseEvt                         {syntaks.KONST(
                                                       konst.hexTilKonst(#1 $1),
                                                       #2 $1, $2)}
        | AFLOEB stoerrelseEvt                      {syntaks.AFLOEB($2,$1)}
        | STRENG typeEvt bitafledHEvt               {syntaks.KOMP($1,$2,[],$3)}
        | bitafledV STRENG typeEvt bitafledHEvt     {syntaks.KOMP($2,$3,$1,$4)}
        | bitafledVEvt VPAR bitvektor HPAR bitafledHEvt {syntaks.BITV($3,$1,$5)}


// ***************** Type og størrelser *****************

type:
          ANFOER STRENG VKPAR udtryk HKPAR ping     {syntaks.SKELTYPE((,(0,0))
```

```
                                                                    ,$2,$4)}
        | ANFOER STRENG STRENG VKPAR udtryk HKPAR ping  {syntaks.SKELTYPE($2,$3,$5)}
        | ANFOER STRENG forbindelser ping               {syntaks.TYPE([syntaks.udt(
                                                          syntaks.STR($2))],$3)}
        | ANFOER udtrykMedBoolUS forbindelser ping      {syntaks.TYPE([$2],$3)}
        | ANFOER STRENG udtrykMedBoolFlere forbindelser ping
                                                       {syntaks.TYPE((syntaks.udt(
                                                          syntaks.STR($2)))::$3,$4)}
        | ANFOER udtrykMedBoolUS udtrykMedBoolFlere forbindelser ping
                                                       {syntaks.TYPE($2::$3,$4)}

typeEvt:
                                                       {syntaks.NIL}
        | type                                         {$1}

stoerrelseEvt:
                                                       {syntaks.BLANK}
        | ANFOER udtryk ping                           {$2}

forbindelser:
                                                       {[]}
        | SEMI udledninger PIL indledninger forbindelser{($2,$4)::$5}

udledninger:
          udledning                                    {$1}
        | udledning udledninger                        {syntaks.SEKV($1,$2)}
//???tilfoej 'str' paa tal og afloeb, naar forbindelser indfoeres
udledning:
          STRENG bitafledHEvt                          {syntaks.KOMP($1,syntaks.NIL,[],$2)}
        | VPAR udledninger HPAR bitafledHEvt           {syntaks.BITV($2,[],$4)}
        | TAL                                          {syntaks.KONST(
                                                          konst.talTilKonst(#1 $1),
                                                          #2 $1, syntaks.BLANK)}
        | HEX                                          {syntaks.KONST(
                                                          konst.hexTilKonst(#1 $1),
                                                          #2 $1, syntaks.BLANK)}

indledninger:
          indl                                         {$1}
        | indledninger KOMMA indledninger              {syntaks.SPLIT($1,$3)}
//???tillad ikke KOMMA?
indl:
          indledning                                   {$1}
        | indledning indl                              {syntaks.SEKV($1,$2)}

indledning:
          UD                                           {syntaks.UD($1)}  // UD = {o}
        | STRENG                                       {syntaks.KOMP($1,syntaks.NIL,[],[])}
        | bitafledV STRENG                             {syntaks.KOMP($2,syntaks.NIL,$1,[])}
        | bitafledVEvt VPAR indledninger HPAR          {syntaks.BITV($3,$1,[])}
        | AFLOEB                                       {syntaks.AFLOEB(syntaks.BLANK,$1)}


// ***************** Bitafledninger *****************

bitafledV:
          VKPARM HKPAR                                 {[syntaks.TOM $1]}
        | bitV                                         {[syntaks.BITAF($1)]}
        | bitV PLUS bitafledH                          {syntaks.BITAF($1)::$3}

bitafledVEvt:
                                                       {[]}
        | bitafledV                                    {$1}

bitV:
          VKPARM bitmoenster HKPAR bitEvt              {syntaks.BITM($2)::$4}
```

```
        | DIVM udtrykba bitmFeltEvt bitEvt              {syntaks.FELTER($2,$3)::$4}

bitafledH:
          VKPAR HKPAR                                   {[syntaks.TOM $1]}
        | bitH                                          {[syntaks.BITAF($1)]}
        | bitH PLUS bitafledH                           {syntaks.BITAF($1)::$3}

bitafledHEvt:
                                                        {[]}
        | bitafledH                                     {$1}

bitH:
          VKPAR bitmoenster HKPAR bitEvt                {syntaks.BITM($2)::$4}
        | bit                                           {$1}

bit:
          DIV udtrykba bitmFeltEvt bitEvt               {syntaks.FELTER($2,$3)::$4}
        | GANGE udtrykba bitEvt                         {syntaks.KOPIER($2)::$3}

bitEvt:
                                                        {[]}
        | bit                                           {$1}

bitmoenster:
          bitm                                          {[$1]}
        | bitm KOMMA bitmoenster                        {$1::$3}

bitm:
          udtryk                                        {($1,$1)} // kan vaere ledn
        | udtrykEvt TIL udtrykEvt                       {($1,$3)}

bitmFelt:
          bitmoenster                                   {[$1]}
        | bitmoenster SEMI bitmFelt                     {$1::$3}

bitmFeltEvt:
                                                        {[]}
        | VKPAR bitmFelt HKPAR                          {$2}


// ***************** Diverse *****************

strenge:
          STRENG strengeEvt                             {$1::$2}

strengeEvt:
                                                        {[]}
        | strenge                                       {$1}

udtryk:
          STRENG                                        {syntaks.STR($1)}
        | TAL                                           {syntaks.TAL($1)}
        | HEX                                           {syntaks.HEX($1)}
        | VPAR udtryk HPAR                              {$2}
        | udtryk PLUS udtryk                            {syntaks.PLUS($1,$3)}
        | udtryk MINUS udtryk                           {syntaks.MINUS($1,$3)}
        | udtryk GANGE udtryk                           {syntaks.GANGE($1,$3)}
        | udtryk DIV udtryk                             {syntaks.DIV($1,$3)}
        | udtryk DIVM udtryk                            {syntaks.DIV($1,$3)}
        | udtryk MOD udtryk                             {syntaks.MOD($1,$3)}
        | udtryk HAT udtryk                             {syntaks.HAT($1,$3)}

udtrykUS:
          TAL                                           {syntaks.TAL($1)}
        | HEX                                           {syntaks.HEX($1)}
        | VPAR udtryk HPAR                              {$2}
```

```
        | udtryk PLUS udtryk                          {syntaks.PLUS($1,$3)}
        | udtryk MINUS udtryk                         {syntaks.MINUS($1,$3)}
        | udtryk GANGE udtryk                         {syntaks.GANGE($1,$3)}
        | udtryk DIV udtryk                           {syntaks.DIV($1,$3)}
        | udtryk DIVM udtryk                          {syntaks.DIV($1,$3)}
        | udtryk MOD udtryk                           {syntaks.MOD($1,$3)}
        | udtryk HAT udtryk                           {syntaks.HAT($1,$3)}

udtrykEvt:
                                                      {syntaks.BLANK}
        | udtryk                                      {$1}

udtrykFlere:
          udtryk udtrykFlereEvt                       {$1::$2}

udtrykFlereEvt:
                                                      {[]}
        | udtrykFlere                                 {$1}

udtrykba:
          STRENG                                      {syntaks.STR($1)}
        | TAL                                         {syntaks.TAL($1)}
        | HEX                                         {syntaks.HEX($1)}
        | VPAR udtryk HPAR                            {$2}

udtrykMedBoolFlere:
          udtrykMedBool                               {[$1]}
        | udtrykMedBool udtrykMedBoolFlere            {$1::$2}

udtrykMedBool:
          udtryk                                      {syntaks.udt($1)}
        | TRUE                                        {syntaks.udtb(syntaks.TRUE($1))}
        | FALSE                                       {syntaks.udtb(syntaks.FALSE($1))}

udtrykMedBoolUS:
          udtrykUS                                    {syntaks.udt($1)}
        | TRUE                                        {syntaks.udtb(syntaks.TRUE($1))}
        | FALSE                                       {syntaks.udtb(syntaks.FALSE($1))}

udtrykBool:
          STRENG                                      {syntaks.STRB($1)}  // boolsk var
        | TRUE                                        {syntaks.TRUE($1)}
        | FALSE                                       {syntaks.FALSE($1)}
        | udtryk LT udtryk                            {syntaks.LT($1,$3)}
        | udtryk LEQ udtryk                           {syntaks.LEQ($1,$3)}
        | udtryk LIG udtryk                           {syntaks.LIG($1,$3)}
        | udtryk GEQ udtryk                           {syntaks.GEQ($1,$3)}
        | udtryk GT udtryk                            {syntaks.GT($1,$3)}

// resten er nødvendige pga brugen af mellemrum

ping:
          ANFOER                                              {}
        | ANFOERM                                             {}
```

### main.sml

```
structure main =
  struct

    exception fejlMain

    fun createLexerStream ( is : BasicIO.instream ) =
          Lexing.createLexer (fn buff => fn n => Nonstdio.buff_input is buff 0 n)

    fun split (--"::filer) = ([],filer)
```

```
      | split (arg::argsfiler) =
          let val (args,filer) = split argsfiler
          in (arg::args,filer) end
      | split [] = raise fejlMain  (* sker ikke *)

    fun vaerdierLav [] = []
      | vaerdierLav ("true"::args) = (syntaks.B true)::(vaerdierLav args)
      | vaerdierLav ("false"::args) = (syntaks.B false)::(vaerdierLav args)
      | vaerdierLav (arg::args) =
          let val t = Int.fromString arg in
            (case t of
               SOME tal => (syntaks.I tal)::(vaerdierLav args)
             | NONE     => raise fejl.fejl(fejl.mainVaerdiIkkeTalBool))
          end

    fun filerLaes [] = []
      | filerLaes (filnavn::filer) =
          let val lexbuf = createLexerStream (BasicIO.open_in filnavn)
            val abstrae = (parser.program lexer.token lexbuf)
          in abstrae@(filerLaes filer) end

    fun compile argsfiler =
          let val (args,filer) = split argsfiler
            val vaerdier = vaerdierLav args
            val abstrae = filerLaes filer
            val (hfiler,ccfiler) = oversaetter.oversaet abstrae vaerdier
            fun skrivFiler [] = ()(*???slet*)
              | skrivFiler (s::ss) = (TextIO.output(TextIO.stdOut, s^" ")
                                      ; skrivFiler ss)
          in (TextIO.output(TextIO.stdOut,"\n")) end

    fun fejlSkriv s = ((TextIO.output (TextIO.stdErr,s ^ "\n")); BasicIO.exit 1)

    val _ = (compile (List.tl (Mosml.argv ()))
             handle Parsing.yyexit ob => fejlSkriv "Parser-exit\n"
                  | Parsing.ParseError ob =>
                      let val Location.Loc (p1,p2)
                              = Location.getCurrentLocation ()
                          val (lin,col)
                              = lexer.getLineCol p2 (!lexer.currentLine)
                                                    (!lexer.lineStartPos)
                      in fejlSkriv ("Parse-error at line "
                                    ^ makestring lin ^ ", column " ^ makestring col)
                      end
                  | lexer.fejl_leksikalsk (mess,(lin,col)) =>
                      fejlSkriv ("Lexical error: " ^mess^ " at line "
                                 ^ makestring lin ^ ", column " ^ makestring col)
                  | fejl.syntaks_strEjErklaeret (ln,sjl) =>
                      fejlSkriv ("Error at line "^makestring ln^", col "
                                 ^makestring sjl^":\nSize not declared")
                  | fejl.fejl meddelelse =>
                      fejlSkriv (fejl.oversaet meddelelse)
                  | SysErr (s,_) => fejlSkriv ("Exception: " ^ s))
      handle Empty => fejlSkriv ("no program files specified")

  end
```

### oversaetter.sig

```
signature oversaetter =
  sig

    (* Ud fra det abstrakte syntakstrae og de indlaeste vaerdier (vaerdier
     * til hovedprogrammet) laves og gemmes en liste med mellemkode for alle
     * brugte komponenttyper (forskellige stoerrelser giver forskellige
     * komponenttyper).
```

```
     * Mellemkoden oversaettes herefter til en fil med SimSyskode.
     * Returnerer navnene paa h-filerne og paa cc-filerne. *)

    val oversaet : syntaks.programblok list -> syntaks.intboolstreng list
                -> string list * string list
                 (* (abstrakt syntakstrae,vaerdier) -> (h-filer,cc-filer) *)

  end
```

### oversaetter.sml

```
structure oversaetter :> oversaetter =
  struct

    (* Den fulde oversaettelse fra hades til mellemkode og videre til
     * SimSys. *)

    fun oversaet abstrae vaerdier =
          (* laes alle erklaeringer i programmet *)
          let val (varnavne,planer,krop) = typ.laes abstrae  (* vaerdier for main *)
            (* oversaet kroppen for main, og rekursivt for alle dens boern osv,
             * til mellemkode (indstik, udstik, ophav = []) *)
            val () = mellemkode.oversaet vaerdier varnavne planer krop
          (* skriv til sidst det hele i h- og cc-filer *)
          in simsys.oversaet() end

  end
```

### typ.sig

```
signature typ =
  sig

    (* Laes alle erklaeringer i programmet og gem typnavn, varliste, ind- og
     * udstik, planer og kroppen, hvor plan- og bitvardefinitioner er fjernet
     * og sidstnaevnte erstattet med deres vaerdi. Planerne faas herfra.
     * Ovenstaaende gemmes sammen med tilsvarende vaerdier for de indbyggede
     * komponenter, hvor krop = [].
     * Lav til sidst dag med alle typnavne (inklusive indbyggede komponenter). *)

    (* Rejser fejl, hvis der er ingen eller to hovedprogrammer, eller hvis
     * to (komp)typer, planer eller bitvar har samme navn, eller hvis der er
     * er fejl i en plan eller i listen af stik. *)
    val laes : syntaks.programblok list
            -> syntaks.streng list * (string * plan.typ) list * syntaks.saetning list
             (* program -> (varnavne, planer, krop) for main *)

    (* Henter vaerdierne for en komponenttype. Rejser fejl, hvis ingen type
     * af dette navn findes. *)
    val hent : syntaks.streng -> (string * syntaks.vaerditype) list
             * (string * string * syntaks.udtryk * syntaks.udtryk) list
             * (string * string * syntaks.udtryk * syntaks.udtryk) list
             * (string * plan.typ) list * syntaks.saetning list
             (* typ -> (varliste,indstik,udstik,planer,krop) *)

    (* For et komponentnavn findes den komponenttype, der er foerste og
     * stoerste (i naevnte prioritetsfoelge) delstreng i navnet.
     * Der rejses fejl, hvis ingen komptype er en delstreng i navnet. *)
    val findMaxMatch : syntaks.streng -> string

    (* Hent SimSys-navnet for en indbygget komponent. *)
    val hentNavn : string -> string

  end
```

### typ.sml

```
structure typ :> typ =
  struct

    local open syntaks fejl
    in

      (* Laes alle erklaeringer i programmet og gem typnavn, varliste, ind- og
       * udstik, planer og kroppen, hvor plan- og bitvardefinitioner er fjernet
       * og sidstnaevnte erstattet med deres vaerdi. Planerne faas herfra.
       * Ovenstaaende gemmes sammen med tilsvarende vaerdier for de indbyggede
       * komponenter, hvor krop = [].
       * Lav til sidst dag med alle typnavne (inklusive indbyggede komponenter). *)

      exception fejlTyp_erStik of streng
      exception fejlTyp_ikkeBitvar
      exception fejlTyp

      (***** Diverse *****)

      fun indsaetN navn data [] = [(navn,data)]
        | indsaetN (navn as (s,_)) data ((elm as ((s',_),_))::liste) =
            if (s < s') then (navn,data)::elm::liste
            else if (s = s') then raise fejl(bitvarDobbeltDef navn)
                 else elm::(indsaetN navn data liste)

      fun opdaterN _ _ [] = raise fejlTyp
        | opdaterN (navn as (s,_)) data ((elm as ((s',_),_))::rest) =
            if (s = s') then (navn,data)::rest
            else elm::(opdaterN navn data rest)

      fun hentN _ [] = raise fejlTyp_ikkeBitvar
        | hentN (navn as (s,_)) ((elm as ((s',_),data))::rest) =
            if (s = s') then data else (hentN navn rest)

      (***** Haandtering af strukturen med de forskellige komponenttyper
       * og af dag'en med komptypenavnene. *****)

      (* struktur kommer til at vaere [((komptyp,pos),data)], hvor
       * data = (varliste,indstik,udstik,planer,krop), hvor kroppen er uden
       * definitioner af planer og bitvar.
       * stik = [(stiknavn,vektorfeltstr,str)], hvor vektorfeltstr angiver
       * stoerrelsen af de enkelte felter, hvis stikket er ARRAYABSTRACTION. *)
      val struktur = ref [("add",
                           ([("sz",stoertype)],[("ina","inA",BLANK,STR("sz",(0,0))),
                                                ("inb","inB",BLANK,STR("sz",(0,0))),
                                                ("cin","cIn",BLANK,TAL(1,(0,0)))],
                            [("sum","sum",BLANK,STR("sz",(0,0))),
                             ("cout","cOut",BLANK,TAL(1,(0,0)))],[],[])),
                          ("addends",
                           ([("h",stoertype),("w",stoertype)],
                            [("multiplier","multiplier",BLANK,STR("h",(0,0))),
                             ("multiplicand","multiplicand",BLANK,STR("w",(0,0)))],
                            [("addends","addends",STR("w",(0,0)),
                              GANGE(STR("w",(0,0)),STR("h",(0,0))))],[],[])),
                          ("adder",
                           ([("sz",stoertype)],[("ina","inA",BLANK,STR("sz",(0,0))),
                                                ("inb","inB",BLANK,STR("sz",(0,0))),
                                                ("cin","cIn",BLANK,TAL(1,(0,0)))],
                            [("sum","sum",BLANK,STR("sz",(0,0))),
                             ("cout","cOut",BLANK,TAL(1,(0,0)))],[],[])),
                          ("and",
                           ([("w",stoertype)],[("in","in",BLANK,STR("w",(0,0)))],
                            [("out","out",BLANK,TAL(1,(0,0)))],[],[])),
                          ("andgate",
                           ([("w",stoertype)],[("in","in",BLANK,STR("w",(0,0)))],
```

```
  [("out","out",BLANK,TAL(1,(0,0)))],[],[])),
("csadd",
 ([("sz",stoertype)],
  [("ina","inA",BLANK,STR("sz",(0,0))),
   ("inb","inB",BLANK,STR("sz",(0,0))),
   ("inc","inC",BLANK,STR("sz",(0,0)))],
  [("suma","sumA",BLANK,STR("sz",(0,0))),
   ("sumb","sumB",BLANK,STR("sz",(0,0)))],[],[])),
("csadder",
 ([("sz",stoertype)],
  [("ina","inA",BLANK,STR("sz",(0,0))),
   ("inb","inB",BLANK,STR("sz",(0,0))),
   ("inc","inC",BLANK,STR("sz",(0,0)))],
  [("suma","sumA",BLANK,STR("sz",(0,0))),
   ("sumb","sumB",BLANK,STR("sz",(0,0)))],[],[])),
("decoder",
 ([("sz",stoertype)],[("in","in",BLANK,STR("sz",(0,0)))],
  [("out","out",BLANK,HAT(TAL(2,(0,0)),STR("sz",(0,0))))],
  [],[])),
("encoder",
 ([("sz",stoertype)],
  [("in","in",BLANK,HAT(TAL(2,(0,0)),STR("sz",(0,0))))],
  [("out","out",BLANK,STR("sz",(0,0))),
   ("valid","valid",BLANK,TAL(1,(0,0)))],
  [],[])),
("fastmux",
 ([("sz",stoertype),("w",stoertype)],
  [("in","in",STR("w",(0,0)),
    GANGE(STR("w",(0,0)),STR("sz",(0,0)))),
   ("select","select",BLANK,STR("sz",(0,0)))],
  [("out","out",BLANK,STR("w",(0,0)))],[],[])),
("ff",
 ([("sz",stoertype),("init",taltype)],
  [("in","in",BLANK,STR("sz",(0,0)))],
  [("out","out",BLANK,STR("sz",(0,0)))],[],[])),
("flipflop",
 ([("sz",stoertype),("init",taltype)],
  [("in","in",BLANK,STR("sz",(0,0)))],
  [("out","out",BLANK,STR("sz",(0,0)))],[],[])),
("fulladd",
 ([],[("ina","inA",BLANK,TAL(1,(0,0))),
      ("inb","inB",BLANK,TAL(1,(0,0))),
      ("inc","inC",BLANK,TAL(1,(0,0)))],
  [("outa","outA",BLANK,TAL(1,(0,0))),
   ("outb","outB",BLANK,TAL(1,(0,0)))],
  [],[])),
("fulladder",
 ([],[("ina","inA",BLANK,TAL(1,(0,0))),
      ("inb","inB",BLANK,TAL(1,(0,0))),
      ("inc","inC",BLANK,TAL(1,(0,0)))],
  [("outa","outA",BLANK,TAL(1,(0,0))),
   ("outb","outB",BLANK,TAL(1,(0,0)))],[],[])),
("halfadd",
 ([],[("ina","inA",BLANK,TAL(1,(0,0))),
      ("inb","inB",BLANK,TAL(1,(0,0)))],
  [("outa","outA",BLANK,TAL(1,(0,0))),
   ("outb","outB",BLANK,TAL(1,(0,0)))],
  [],[])),
("halfadder",
 ([],[("ina","inA",BLANK,TAL(1,(0,0))),
      ("inb","inB",BLANK,TAL(1,(0,0)))],
  [("outa","outA",BLANK,TAL(1,(0,0))),
   ("outb","outB",BLANK,TAL(1,(0,0)))],
  [],[])),
("idealmemiosys",
 ([("logMemSz",taltype),("accessTime",taltype),
```

```
   ("fname",strengtype)],
  [("iaddrin","iAddrIn",BLANK,TAL(32,(0,0))),
   ("ird","iRd",BLANK,TAL(1,(0,0))),
   ("daddrin","dAddrIn",BLANK,TAL(32,(0,0))),
   ("drd","dRd",BLANK,TAL(1,(0,0))),
   ("dwr","dWr",BLANK,TAL(1,(0,0))),
   ("ddatain","dDataIn",BLANK,TAL(32,(0,0)))],
  [("idataout","iDataOut",BLANK,TAL(32,(0,0))),
   ("ddataout","dDataOut",BLANK,TAL(32,(0,0)))],[],[])),
("memiosys",
 ([("Sets",taltype),("IdxSz",taltype),("BlockSz",taltype),
   ("memIdxSz",taltype),("memFirstChunk",taltype),
   ("memNextChunk",taltype),("fname",strengtype),
   ("iPortSz",stoertype)],
  [("iaddrin","iAddrIn",BLANK,TAL(32,(0,0))),
   ("ird","iRd",BLANK,TAL(1,(0,0))),
   ("daddrin","dAddrIn",BLANK,TAL(32,(0,0))),
   ("drd","dRd",BLANK,TAL(1,(0,0))),
   ("dwr","dWr",BLANK,TAL(1,(0,0))),
   ("ddatain","dDataIn",BLANK,TAL(32,(0,0)))],
  [("idataout","iDataOut",BLANK,TAL(32,(0,0))),
   ("ddataout","dDataOut",BLANK,TAL(32,(0,0))),
   ("holdandwait","holdAndWait",BLANK,TAL(1,(0,0)))],[],[])),
("merger",
 ([("streamsz",stoertype),("datasz",stoertype)],
  [("streamin","streamIn",BLANK,STR("streamsz",(0,0))),
   ("datain","dataIn",BLANK,STR("datasz",(0,0))),
   ("mergemask","mergeMask",BLANK,
    DIV(STR("streamsz",(0,0)),STR("datasz",(0,0))))],
  [("streamout","streamOut",BLANK,STR("streamsz",(0,0)))],
  [],[])),
("multiply",
 ([("h",stoertype),("w",stoertype)],
  [("multiplier","multiplier",BLANK,STR("h",(0,0))),
   ("multiplicand","multiplicand",BLANK,STR("w",(0,0)))],
  [("product","product",BLANK,STR("w",(0,0)))],[],[])),
("mux",
 ([("sz",stoertype),("w",stoertype)],
  [("in","in",STR("w",(0,0)),
    GANGE(STR("w",(0,0)),HAT(TAL(2,(0,0)),STR("sz",(0,0)))),
   ("select","select",BLANK,STR("sz",(0,0)))],
  [("out","out",BLANK,STR("w",(0,0)))],[],[])),
("nand",
 ([("w",stoertype)],[("in","in",BLANK,STR("w",(0,0)))],
  [("out","out",BLANK,TAL(1,(0,0)))],[],[])),
("nandgate",
 ([("w",stoertype)],[("in","in",BLANK,STR("w",(0,0)))],
  [("out","out",BLANK,TAL(1,(0,0)))],[],[])),
("nor",
 ([("w",stoertype)],[("in","in",BLANK,STR("w",(0,0)))],
  [("out","out",BLANK,TAL(1,(0,0)))],[],[])),
("norgate",
 ([("w",stoertype)],[("in","in",BLANK,STR("w",(0,0)))],
  [("out","out",BLANK,TAL(1,(0,0)))],[],[])),
("not",
 ([],[("in","in",BLANK,TAL(1,(0,0)))],
  [("out","out",BLANK,TAL(1,(0,0)))],
  [],[])),
("notgate",
 ([],[("in","in",BLANK,TAL(1,(0,0)))],
  [("out","out",BLANK,TAL(1,(0,0)))],
  [],[])),
("or",
 ([("w",stoertype)],[("in","in",BLANK,STR("w",(0,0)))],
  [("out","out",BLANK,TAL(1,(0,0)))],[],[])),
("orgate",
```

```
 ([("w",stoertype)],[("in","in",BLANK,STR("w",(0,0)))],
  [("out","out",BLANK,TAL(1,(0,0)))],[],[])),
("pgnode",
 ([("w",stoertype)],
  [("pin","pIn",BLANK,STR("w",(0,0))),
   ("gin","gIn",BLANK,STR("w",(0,0))),
   ("cin","cIn",BLANK,TAL(1,(0,0)))],
  [("cout","cOut",BLANK,STR("w",(0,0))),
   ("pout","pOut",BLANK,TAL(1,(0,0))),
   ("gout","gOut",BLANK,TAL(1,(0,0)))],[],[])),
("pgtree",
 ([("w",stoertype),("nodeWidth",taltype)],
  [("pin","pIn",BLANK,STR("w",(0,0))),
   ("gin","gIn",BLANK,STR("w",(0,0))),
   ("cin","cIn",BLANK,TAL(1,(0,0)))],
  [("carries","carries",BLANK,STR("w",(0,0))),
   ("cout","cOut",BLANK,TAL(1,(0,0)))],[],[])),
("prioritizer",
 ([("sz",stoertype)],[("in","in",BLANK,STR("sz",(0,0)))],
  [("out","out",BLANK,STR("sz",(0,0))),
   ("valid","valid",BLANK,TAL(1,(0,0)))],
  [],[])),
("priorityencoder",
 ([("logsz",stoertype)],[("in","in",BLANK,
                          HAT(TAL(2,(0,0)),STR("logsz",(0,0))))],
  [("out","out",BLANK,STR("logsz",(0,0))),
   ("valid","valid",BLANK,TAL(1,(0,0)))],
  [],[])),
("qualifieddecoder",
 ([("sz",stoertype)],
  [("in","in",BLANK,STR("sz",(0,0))),
   ("qualifier","qualifier",BLANK,TAL(1,(0,0)))],
  [("out","out",BLANK,
    HAT(TAL(2,(0,0)),STR("sz",(0,0))))],[],[])),
("qualifiedsearch",
 ([("fields",stoertype),("w",stoertype)],
  [("keys","keys",BLANK,
    GANGE(STR("fields",(0,0)),STR("w",(0,0)))),
   ("key","key",BLANK,STR("w",(0,0))),
   ("valid","valid",BLANK,STR("fields",(0,0)))],
  [("hits","hits",BLANK,STR("fields",(0,0)))],[],[])),
("regfile",
 ([("wr",stoertype),("rd",stoertype),("selsz",stoertype),
   ("sz",stoertype)],
  [("wrselect","wrSelect",STR("selsz",(0,0)),
    GANGE(STR("selsz",(0,0)),STR("wr",(0,0)))),
   ("wrenable","wrEnable",TAL(1,(0,0)),STR("wr",(0,0))),
   ("wrdata","wrData",STR("sz",(0,0)),
    GANGE(STR("sz",(0,0)),STR("wr",(0,0)))),
   ("rdselect","rdSelect",STR("selsz",(0,0)),
    GANGE(STR("selsz",(0,0)),STR("rd",(0,0))))],
  [("rddata","rdData",STR("sz",(0,0)),
    GANGE(STR("sz",(0,0)),STR("rd",(0,0))))],[],[])),
("registerfile",
 ([("wr",stoertype),("rd",stoertype),("selsz",stoertype),
   ("sz",stoertype)],
  [("wrselect","wrSelect",STR("selsz",(0,0)),
    GANGE(STR("selsz",(0,0)),STR("wr",(0,0)))),
   ("wrenable","wrEnable",TAL(1,(0,0)),STR("wr",(0,0))),
   ("wrdata","wrData",STR("sz",(0,0)),
    GANGE(STR("sz",(0,0)),STR("wr",(0,0)))),
   ("rdselect","rdSelect",STR("selsz",(0,0)),
    GANGE(STR("selsz",(0,0)),STR("rd",(0,0))))],
  [("rddata","rdData",STR("sz",(0,0)),
    GANGE(STR("sz",(0,0)),STR("rd",(0,0))))],[],[])),
("search",
```

```
                      ([("fields",stoertype),("w",stoertype)],
                       [("keys","keys",BLANK,
                         GANGE(STR("fields",(0,0)),STR("w",(0,0)))),
                        ("key","key",BLANK,STR("w",(0,0)))],
                       [("hits","hits",BLANK,STR("fields",(0,0)))],[],[])),
                     ("serialadd",
                      ([("sz",stoertype)],
                       [("ina","inA",BLANK,STR("sz",(0,0))),
                        ("inb","inB",BLANK,STR("sz",(0,0))),
                        ("cin","cIn",BLANK,TAL(1,(0,0)))],
                       [("sum","sum",BLANK,STR("sz",(0,0))),
                        ("cout","cOut",BLANK,TAL(1,(0,0)))],
                       [],[])),
                     ("serialadder",
                      ([("sz",stoertype)],
                       [("ina","inA",BLANK,STR("sz",(0,0))),
                        ("inb","inB",BLANK,STR("sz",(0,0))),
                        ("cin","cIn",BLANK,TAL(1,(0,0)))],
                       [("sum","sum",BLANK,STR("sz",(0,0))),
                        ("cout","cOut",BLANK,TAL(1,(0,0)))],
                       [],[])),
                     ("sram",
                      ([("wr",stoertype),("rd",stoertype),("lines",stoertype),
                        ("sz",stoertype)],
                       [("wrenable","wrEnable",STR("lines",(0,0)),
                         GANGE(STR("lines",(0,0)),STR("wr",(0,0)))),
                        ("datain","dataIn",STR("sz",(0,0)),
                         GANGE(STR("sz",(0,0)),STR("wr",(0,0)))),
                        ("rdenable","rdEnable",STR("lines",(0,0)),
                         GANGE(STR("lines",(0,0)),STR("rd",(0,0))))],
                       [("dataout","dataOut",STR("sz",(0,0)),
                         GANGE(STR("sz",(0,0)),STR("rd",(0,0))))],[],[])),
                     ("validatewrite",
                      ([("ports",stoertype),("idxsz",stoertype),("sz",stoertype),
                        ("Domain",taltype)],
                       [("idxin","idxIn",STR("idxsz",(0,0)),
                         GANGE(STR("idxsz",(0,0)),STR("ports",(0,0)))),
                        ("datain","dataIn",STR("sz",(0,0)),
                         GANGE(STR("sz",(0,0)),STR("ports",(0,0)))),
                        ("wrenable","wrEnable",TAL(1,(0,0)),STR("ports",(0,0)))],
                       [],[],[])),
                     ("wrenff",
                      ([("sz",stoertype),("init",taltype)],
                       [("in","in",BLANK,STR("sz",(0,0))),
                        ("wrenable","wrEnable",BLANK,TAL(1,(0,0)))],
                       [("out","out",BLANK,STR("sz",(0,0)))],[],[])),
                     ("wrenflipflop",
                      ([("sz",stoertype),("init",taltype)],
                       [("in","in",BLANK,STR("sz",(0,0))),
                        ("wrenable","wrEnable",BLANK,TAL(1,(0,0)))],
                       [("out","out",BLANK,STR("sz",(0,0)))],[],[])),
                     ("xor",
                      ([("w",stoertype)],[("in","in",BLANK,STR("w",(0,0)))],
                       [("out","out",BLANK,TAL(1,(0,0)))],[],[])),
                     ("xorgate",
                      ([("w",stoertype)],[("in","in",BLANK,STR("w",(0,0)))],
                       [("out","out",BLANK,TAL(1,(0,0)))],[],[]))]

(* Indeholder SimSys-navnene for de indbyggede komponenter. *)
val navne = ref [("add","Adder"),("addends","Addends"),("adder","Adder"),
                 ("and","AndGate"),("andgate","AndGate"),
                 ("csadd","CSAdder"),("csadder","CSAdder"),
                 ("decoder","Decoder"),("encoder","Encoder"),
                 ("fastmux","FastMux"),("ff","FlipFlop"),("flipflop","FlipFlop"),
                 ("fulladd","FullAdder"),("fulladder","FullAdder"),
                 ("halfadd","HalfAdder"),("halfadder","HalfAdder"),
```

```
                ("idealmemiosys","IdealMemIOSys"),("memiosys","MemIOSys"),
                ("merger","Merger"),("multiply","Multiply"),("mult","Multiply"),
                ("mux","Mux"),("nand","NandGate"),("nandgate","NandGate"),
                ("nor","NorGate"),("norgate","NorGate"),
                ("not","NotGate"),("notgate","NotGate"),
                ("or","OrGate"),("orgate","OrGate"),
                ("pgnode","PGNode"),("pgtree","PGTree"),
                ("prioritizer","Prioritizer"),
                ("priorityencoder","PriorityEncoder"),
                ("qualifieddecoder","QualifiedDecoder"),
                ("qualifiedsearch","QualifiedSearch"),
                ("regfile","RegisterFile"),("registerfile","RegisterFile"),
                ("search","Search"),("serialadd","SerialAdder"),
                ("serialadder","SerialAdder"),("sram","SRam"),
                ("validatewrite","ValidateWrite"),("wrenff","WrEnFlipFlop"),
                ("wrenflipflop","WrEnFlipFlop"),
                ("xor","XorGate"),("xorgate","XorGate")]

(* dag kommer til at indeholde fkt med dag med alle komptyper. Denne
 * bruges til at finde den komptyp, der har foerste (og stoerste)
 * overlap med et komponentnavn. *)
val dag = ref (fn _ => )

fun gem navn data =
      ((struktur := liste.indsaet (#1 navn) data (!struktur))
       handle liste.fejlListe_alleredeIListe =>
         raise fejl(typAlleredeDef navn))

fun hent (s,pos) =
      ((liste.hent s (!struktur))
       handle liste.fejlListe_ikkeIListe =>
         raise fejl(typIkkeDef (s,pos)))

fun dagLav () = dag := fkt.dagDelstreng(map (fn (s,_) => s) (!struktur))

fun findMaxMatch (s,pos) =
      let val typ = (!dag) s in
        if typ <>  then typ
        else raise fejl(navnMatcherIkkeTyp (s,pos))
      end

fun hentNavn typ = liste.hent typ (!navne)

(***** Lav stikliste af formen [(navn,udtryk)], hvor udtryk er
 * stoerrelsen. *****)

(* Kontrollerer at alle variable i et stoerrelsesudtryk er
 * defineret. Hvis udtryk = streng, hvor streng ikke er en variabel,
 * rejses fejlTyp_erStik, da udtrykket saa potentielt er et stik og
 * altsaa ikke en stoerrelse. *)
fun checkOmStoer (STR (s,pos)) varliste =
      ((liste.opdater s stoertype varliste)
       handle liste.fejlListe_ikkeIListe => raise fejlTyp_erStik (s,pos))
  | checkOmStoer u varliste =
      ((let val (v,h) = udtryk.par u
        in checkOmStoer h (checkOmStoer v varliste) end)
          handle fejlTyp_erStik navn       => raise fejl(varIkkeDef navn)
               | udtryk.fejlUdtryk_ikkePar => varliste)  (* u er TAL|HEX|BLANK *)

(* Check om u er stik eller stoerrelse. Tilfoej den til stik,
 * hvis den er stik, og ellers aendr stoerrelsen til denne. *)
fun stikLav' (u, (varliste,stik,stoer,naesteErStoer)) =
      ((let val nyvarliste = checkOmStoer u varliste in
          (* u er en stoerrelse *)
          if naesteErStoer then  (* naeste udtryk er ogsaa en stoerrelse *)
            raise fejl(stoerrelseUdenStik (udtryk.pos stoer))
```

```
          (* naeste udtryk er ikke en stoerrelse *)
          else (nyvarliste,stik,u,true)
        end)  (* u er et stik *)
          handle fejlTyp_erStik (navn as (s,_)) =>
            (case stoer of
               TAL (0,(0,0)) =>  (* sidste elm i listen *)
                 raise fejl(stikIngenStoerrelse navn)
             | _             => (varliste,(s,s,BLANK,stoer)
                                     ::stik,stoer,false)))

(* Listen med udtryk, der skal omdannes til stik, laeses fra hoejre
 * mod venstre, saa stoerrelser laeses foer stik. *)
fun stikLav varliste uliste =
      let val (nyvarliste, stik, stoer, naesteErStoer) =
            foldr stikLav' (varliste, [], TAL(0,(0,0)), false) uliste
      in
        if naesteErStoer then raise fejl(stoerrelseUdenStik (udtryk.pos stoer))
        else (nyvarliste, stik)
      end

(***** Erstat bitvar med deres vaerdi *****)

(* Erstatter bitvars med deres vaerdi i hoejresiden af bitvar. Navn
 * fra den bitvar, hvor bitvars erstattes, skal med, for at der kan
 * rejses fejl, hvis der optraeder rekursion. *)
fun erstatBitvarsIVaerdi navn (BITV(t,ind,ud)) vardefs =
      let val (t',tAendret) = erstatBitvarsIVaerdi navn t vardefs
      in (BITV(t',ind,ud),tAendret) end
  | erstatBitvarsIVaerdi navn (k as KOMP(navn',typen,ind,ud)) vardefs =
      if #1 navn = #1 navn' then  (* bitvar indgaar i egen vaerdi *)
        raise fejl(rekursionIBitvarDef navn)
      else  (* erstat eventuel bitvar med vaerdi *)
        ((let val vaerdi = hentN navn' vardefs in
            if typen <> NIL then raise fejl(bitvarDefSomKomp navn')
            else (BITV(vaerdi,ind,ud),true)
          end)  (* ikke bitvar *)
            handle fejlTyp_ikkeBitvar => (k,false))
  | erstatBitvarsIVaerdi navn bv vardefs =
      ((let val (v,h) = bitvektor.par bv
            val (v',vAendret) = erstatBitvarsIVaerdi navn v vardefs
            val (h',hAendret) = erstatBitvarsIVaerdi navn h vardefs
        in (bitvektor.saml bv (v',h'),vAendret andalso hAendret) end)
          handle bitvektor.fejlBitvektor_ikkeSammensat => (bv,false))

(* Erstatter bitvars i vaerdierne for alle bitvars. Hvis en bitvar
 * erstattes i en vaerdi, opdateres dennes vaerdi, og den tilfoejes
 * desuden til listen af aendrede bitvardef. Naar denne er tom,
 * returneres den nye liste med bitvars og deres vaerdier. *)
fun erstatBitvarsIBitvars'([], vardefs) = vardefs
  | erstatBitvarsIBitvars'(aendrede, vardefs) =
      let fun erstat ((navn,bv), (aendrede,vardefs)) =
        let val (nybv,erAendret) = erstatBitvarsIVaerdi navn bv vardefs in
          if erAendret then
            ((navn,nybv)::aendrede,opdaterN navn nybv vardefs)
          else (aendrede,vardefs)
        end
      in erstatBitvarsIBitvars'(List.foldr erstat ([],vardefs) aendrede) end

(* Erstatter bitvars paa hoejresiden af en bitvardefinition med deres
 * vaerdi, indtil der ikke bitvars paa nogen hoejreside. *)
fun erstatBitvarsIBitvars vardefs = erstatBitvarsIBitvars'(vardefs, vardefs)

(* Erstatter bitvar i en bitvektor med deres vaerdi. *)
fun erstatBitvarsIBitvektor bitvar bv =
      ((let val (v,h) = bitvektor.par bv in
          bitvektor.saml bv (erstatBitvarsIBitvektor bitvar v,
```

```
                          erstatBitvarsIBitvektor bitvar h)
        end)
          handle bitvektor.fejlBitvektor_ikkeSammensat =>
            (case bv of
               BITV(bv',i,u) =>
                 BITV(erstatBitvarsIBitvektor bitvar bv',i,u)
             | KOMP(navn,t,i,u) =>
                 ((let val vaerdi = hentN navn bitvar in
                     if t = NIL then BITV(vaerdi,i,u)
                     else raise fejl(bitvarDefSomKomp navn)
                   end)
                     handle fejlTyp_ikkeBitvar => bv) (* ej bitvar *)
             | _ => bv))  (* afloeb, gennemloeb, konst, {o} *)

(* Erstatter bitvars i saetninger med deres vaerdi. *)
fun erstatBitvarsISaetn bitvar (SAETNING bv) =
      SAETNING (erstatBitvarsIBitvektor bitvar bv)
  | erstatBitvarsISaetn bitvar (IF (bool,k0,k1)) =
      IF (bool, map (erstatBitvarsISaetn bitvar) k0,
                map (erstatBitvarsISaetn bitvar) k1)
  | erstatBitvarsISaetn _ _ = raise fejlTyp  (* sker ikke *)


(***** Fjern bitvar og planer i krop og erstat bitvar med deres
 * vaerdier. *****)

(* Fjerner og returnerer plan- og bitvardefinitioner fra en krop.
 * Bitvarlisten har formen [(navn,vaerdi)], hvor vaerdi er en
 * bitvektor. Rejser fejl, hvis der er fejl i planen, eller hvis en
 * plan eller bitvar er defineret to gange. *)
fun fjern [] planer bitvar = ([],planer,bitvar)
  | fjern ((VAR(navn,bv))::krop) planer bitvar =
      fjern krop planer (indsaetN navn bv bitvar)
  | fjern ((PLAN(navn,plan))::krop) planer bitvar =
      ((fjern krop (liste.indsaet (#1 navn)  (plan.lav plan navn) planer) bitvar)
       handle liste.fejlListe_alleredeIListe =>
         raise fejl(planDobbeltDef navn))
  | fjern ((IF(u,k0,k1))::krop) planer bitvar =
      let val (nyk0,nyp,nyb) = fjern k0 planer bitvar
        val (nyk1,nyp',nyb') = fjern k1 nyp nyb
        val (nykrop,nyplaner,nybitvar) = fjern krop nyp' nyb'
      in ((IF(u,nyk0,nyk1))::nykrop,nyplaner,nybitvar) end
  | fjern (saetning::krop) planer bitvar =  (* bitvektor *)
      let val (nykrop,nyplaner,nybitvar) = fjern krop planer bitvar
      in (saetning::nykrop,nyplaner,nybitvar) end

(* Fjerner planer og bitvar fra krop og erstatter bitvar med
 * deres vaerdi. *)
fun fjernBitvarOgPlaner krop =
      let val (nykrop',planer,bitvar) = fjern krop [] []
        val nybitvar = erstatBitvarsIBitvars bitvar
        val nykrop = if nybitvar = [] then nykrop'
                     else map (erstatBitvarsISaetn nybitvar) nykrop'
      in (nykrop,planer) end

(***** Laes alle erklaeringer i programmet *****)

(* Laes erklaeringerne af alle programblokke. Gem vaerdierne for de
 * sammensatte komponenter, og returner dem for hovedprogrammet. *)
fun laes' ((MAIN(varnavne,krop))::rest) =
      (* laes resten af programmet *)
      if (laes' rest) <> ([],[],[]) then raise fejl(toHovedprogrammer)
      else  (* fjern planer og bitvar fra krop samt lav varliste *)
        let val (nykrop,planer) = fjernBitvarOgPlaner krop
        in (varnavne,planer,nykrop) end
  | laes' ((KOMPDEF(navn,ind,varnavne,ud,krop))::rest) =
```

```
            let val varliste = map (fn (s,_) => (s,ukendt)) varnavne
              val (nyvarliste',indstik) = stikLav varliste ind
              val (nyvarliste,udstik) = stikLav nyvarliste' ud
              val (nykrop,planer) = fjernBitvarOgPlaner krop
            in if ind = [] andalso ud = [] then raise fejl(kompUdenStik navn)
               else (gem navn (nyvarliste,indstik,udstik,planer,krop)
                     ; laes' rest) end
       | laes' [] = ([],[],[])

      fun laes program =
            let val vaerdi = laes' program in
              if vaerdi = ([],[],[]) then raise fejl(intetHovedprogram)
              else (dagLav() ; vaerdi)
            end

    end
  end
```

### mellemkode.sig

```
signature mellemkode =
  sig

    exception fejlMellemkode_indbyggetKomp

    (* Oversaetter hovedprogrammets krop til mellemkode, samt rekursivt
     * alle ikke-indbyggede komponenter i kroppen osv. *)

    val oversaet : syntaks.intboolstreng list -> syntaks.streng list
                -> (string * plan.typ) list -> syntaks.saetning list
                -> unit
                   (* (vaerdier,varnavne,planer,krop) -> unit *)

    (* Henter mellemkoden for programmet. *)
    val hent : unit ->
               (string
               * ((string * int) list * (string * int) list
                 * (string * plan.typ) list
                 * (string * (syntaks.streng * syntaks.intboolstreng list)) list
                 * (string * string * string * string * string * int) list
                 * (split.bvElm * split.bvElm) list)) list
            (* () -> [(typnavn,(indstik,udstik,planer,komp,skel,krop))],
             * hvor skel = [(navn,typ,kompn,stik,stiknr,bitNr)]. *)

    (* Returnerer navnet paa komponenten af specificeret komptype og
     * med angivne vaerdier. Rejser fejlMellemkode_indbyggetKomp, hvis
     * komponenten ikke er brugerdefineret. *)
    val hentEgenKomp : string -> syntaks.intboolstreng list -> string

  end
```

### mellemkode.sml

```
structure mellemkode :> mellemkode =
  struct

    local open syntaks fejl
    in

    (* Oversaetter hovedprogrammets krop til mellemkode, samt rekursivt
     * alle ikke-indbyggede komponenter i kroppen osv. *)

      exception fejlMellemkode
      exception fejlMellemkode_indbyggetKomp
```

```
(***** Vaerdier *****)

fun lig ((I tal)::vaerdier) ((I tal')::vaerdier') =
      (tal = tal') andalso (lig vaerdier vaerdier')
  | lig ((B bu)::vaerdier) ((B bu')::vaerdier') =
      (bu = bu') andalso (lig vaerdier vaerdier')
  | lig [] [] = true
  | lig _ _ = raise fejlMellemkode

fun stoerre ((I tal)::vaerdier) ((I tal')::vaerdier') =
      (tal > tal') orelse ((tal = tal') andalso (stoerre vaerdier vaerdier'))
  | stoerre (_::vaerdier) (_::vaerdier') = stoerre vaerdier vaerdier'
  | stoerre [] [] = false
  | stoerre _ _ = raise fejlMellemkode  (* sker ikke *)

(***** Haandtering af strukturen med mellemkoden *****)

(* Der er to strukturer, som kun indeholder data for de sammensatte
 * (egne) komponenter:
 * typstruktur = [(typ,vaerdier,navn)], hvor navn=typ, hvis der kun
 * findes en slags af denne komponenttype,
 * struktur = [(navn,(indstik,udstik,planer,komp,krop))].
 * For hver sammensat komponent findes alle de slags, der bruges i
 * programmet, samt "main" i begge strukturer. *)

val typstruktur = ref []  (* Er sorteret efter typ, men ikke efter vaerdier. *)
val struktur = ref []

val kompnavn = ref
val kompnr = ref 0

fun indsaetTyp' styp vaerdier s ((elm as (styp',vaerdier',_))::rest) =
      if styp > styp' then elm::(indsaetTyp' styp vaerdier s rest)
      else if styp < styp' then (kompnavn := s ; (styp,vaerdier,s)::elm::rest)
           else  (* styp = styp' *)
             if (lig vaerdier vaerdier') then elm::rest
             else elm::(indsaetTyp' styp vaerdier s rest)
  | indsaetTyp' styp vaerdier s [] = (kompnavn := s ; [(styp,vaerdier,s)])

(* Indsaetter (styp,vaerdier,s) i typstrukturen, hvis (styp,vaerdier)
 * ikke allerede findes i denne. s er et unikt navn for denne komptype.
 * Returnerer , hvis (styp,vaerdier) allerede findes; ellers s. *)
fun indsaetTyp "main" vaerdier =
      (typstruktur := indsaetTyp' "main" vaerdier "main" (!typstruktur) ; "main")
  | indsaetTyp styp vaerdier =
      let val s = "komp"^(Int.toString (!kompnr)) in
        (kompnr := (!kompnr) + 1 ; kompnavn :=
         ; typstruktur := indsaetTyp' styp vaerdier s (!typstruktur)
         ; !kompnavn)
      end

fun hentTyp typ vaerdier ((typ',vaerdier',typnavn)::rest) =
      if typ < typ' then raise fejlMellemkode_indbyggetKomp
      else if typ = typ' andalso (lig vaerdier vaerdier') then typnavn
           else hentTyp typ vaerdier rest  (* typ>typ' ell. vaerdier forsk *)
  | hentTyp _ _ [] = raise fejlMellemkode_indbyggetKomp

fun hentEgenKomp typ vaerdier = hentTyp typ vaerdier (!typstruktur)

(* En komptyp med dette navn findes ikke i forvejen. *)
fun indsaet styp data =
      struktur := liste.indsaet styp data (!struktur)

fun hent () = (!struktur)

(***** Diverse *****)
```

```
fun ophavIndsaet styp vaerdier [] = [(styp,vaerdier)]
  | ophavIndsaet styp vaerdier ((styp',vaerdier')::ophav) =
      if styp < styp' then (styp,vaerdier)::(styp',vaerdier')::ophav
      else if styp > styp' then
             (styp',vaerdier')::(ophavIndsaet styp vaerdier ophav)
           else  (* styp = styp' *)
             if (stoerre vaerdier' vaerdier) then (styp,vaerdier)::ophav
             else raise fejl(komponentRekursionUdenFald styp)

(* Behoever ikke kontrollere typer, da dette blev checket, da
 * stoerrelserne blev infereret. *)
fun varLav ((navn,_)::varliste) (vaerdi::vaerdier) =
      (navn,vaerdi)::(varLav varliste vaerdier)
  | varLav [] [] = []
  | varLav _ _ = raise fejlMellemkode

(***** Oversaettelsen til mellemkode *****)

fun kroppeTilMellemkode [] _ = ()
  | kroppeTilMellemkode ((_,(typ,vaerdier))::komp) ophav =
      (((kropTilMellemkode typ vaerdier (typ.hent typ) ophav)
        handle fejl(typIkkeDef _) => ())  (* typ er PLA *)
       ; kroppeTilMellemkode komp ophav)

and kropTilMellemkode _ _ (_,_,_,_,[]) _ = ()  (* indbygget komp *)
  | kropTilMellemkode (s,_) vaerdier (varliste,ind,ud,planer,krop) ophav =
      (* rejs fejl, hvis komponenten har sig selv som ophav med
       * hoejst de samme vaerdier *)
      let val nytophav = ophavIndsaet s vaerdier ophav
        val typs = indsaetTyp s vaerdier
      in
        if typs =  then ()  (* typ med disse vaerdier findes allerede *)
        else  (* lav kroppen til mellemkode *)
          let val var = varLav varliste vaerdier
            val indstik = map (fn (s,_,_,str) => (s,udtryk.udregn var str)) ind
            val udstik = map (fn (s,_,_,str) => (s,udtryk.udregn var str)) ud
            (* indsaml typer i kroppen og eliminer bet. saetninger
             * (kroppen er nu liste af bitvektorer) *)
            val (nykrop',typer) = typer.indsaml krop var planer
            (* inferer stoerrelsen af komp mm. Kontrollerer at
             * der er rette antal vaerdier i erklaeringerne. *)
            val (komp,kompsAndet,kompStikSkel)
              = stoerrelser.udregn nykrop' var typer indstik udstik planer
            (* Udfyld alle afledninger og kontroller stoerrelser *)
            val nykrop'' = afled.udfyld nykrop' var indstik udstik kompsAndet
            (* lav selve oversaettelsen. Fjern g.loeb, kontroller
             * ud- og indbit korrekt forbundet *)
            val (bvvl,skel)
              = split.tilMellemkode nykrop'' indstik udstik kompStikSkel
          in (indsaet typs (indstik,udstik,planer,komp,skel,bvvl)
              ; kroppeTilMellemkode komp nytophav)
          end
      end

(* Oversaetter hovedprogrammets krop til mellemkode, samt rekursivt
 * alle ikke-indbyggede komponenter i kroppen osv.
 * Modtager vaerdierne for main. *)
fun oversaet vaerdier varnavne planer krop =
      let val varliste = map (fn (s,_) => (s,ukendt)) varnavne in
        if (List.length vaerdier) <> (List.length varnavne) then
          raise fejl(hovedprogramForkertAntalVar (List.length varnavne))
        else (kropTilMellemkode ("main",(0,0)) vaerdier
              (varliste,[],[],planer,krop) [])
      (* indstik, udstik, ophav = [] *)
      end
```

```
    end
  end
```

### typer.sig

```
signature typer =
  sig

    (* Indsamler typerne for komp, afloeb, konst og gennemloeb. Samtidig
     * elimineres den del af hver betinget saetning, der ikke skal bruges.
     * Der goeres intet ved typerne eller ved bitvektorerne i kroppen.
     * Det kontrolleres, at ingen komponenter har samme navn, eller har
     * samme navn som en plan. *)

    val indsaml : syntaks.saetning list -> (string * syntaks.intboolstreng) list
                -> (string * plan.typ) list -> syntaks.bitvektor list
                 * ((syntaks.streng * syntaks.typen) list
                   * (syntaks.pos * syntaks.udtryk) list)
              (* (krop,var,planer) -> krop * komp = ([(navn,typen)],[(pos,str)]) *)

  end
```

### typer.sml

```
structure typer :> typer =
  struct

    local open syntaks fejl
    in

    (* Indsamler typerne for komp, afloeb, konst og gennemloeb. Samtidig
     * elimineres den del af hver betinget saetning, der ikke skal bruges.
     * Der goeres intet ved typerne eller ved bitvektorerne i kroppen. *)

      exception fejlTyper

      (***** Diverse *****)

      fun posUdtryk (GENNEMLOEB(u,pos)) = (pos,u)
        | posUdtryk (AFLOEB(u,pos)) = (pos,u)
        | posUdtryk (KONST(_,pos,u)) = (pos,u)
        | posUdtryk _ = raise fejlTyper  (* kaldes ikke med UD *)

      fun checkNavneoverlap komp [] = ()
        | checkNavneoverlap [] _ = ()
        | checkNavneoverlap ((komp as ((s,pos),_))::komps) ((plan as (s',_))::planer) =
            if s = s' then raise fejl(kompDefSomPlan (s,pos))
            else if s < s' then checkNavneoverlap komps (plan::planer)
                 else checkNavneoverlap (komp::komps) planer

      (***** Haandtering af komp-strukturen *****)

      (* Indsaet eller opdater typen for en komponent. *)
      fun indsaetOpdater navn typen [] = [(navn,typen)]
        | indsaetOpdater (n as (s,_)) typen ((n' as (s',_),typen')::komps) =
            if s < s' then (n,typen)::(n',typen')::komps
            else if s > s' then (n',typen')::(indsaetOpdater n typen komps)
                 else (* s = s' *)
                   if typen = NIL then (n',typen')::komps
                   else if typen' = NIL then (n,typen)::komps
                        else raise fejl(kompDobbeltDef n)

      (***** Indsaml komponenter mm. *****)
```

```
    (* Indsaml typerne i en bitvektor og indsaet i komp. *)
    fun indsamlIBitvektor (BITV(bv,_,_)) komp =
          indsamlIBitvektor bv komp
      | indsamlIBitvektor (KOMP(navn,typen,_,_)) (komps,andet) =
          (indsaetOpdater navn typen komps,andet)
      | indsamlIBitvektor bv (komp as (komps,andet)) =
          ((let val (v,h) = bitvektor.par bv
            in indsamlIBitvektor h (indsamlIBitvektor v komp) end)
              handle bitvektor.fejlBitvektor_ikkeSammensat =>
                let val (pos,u) = posUdtryk bv
                in (komps,liste.indsaetPos pos u andet) end)

    (* Indsaml typer for komponenter, afloeb, gennemloeb og konstanter,
     * og eliminer den del af betingede saetninger, der ikke bruges. *)
    fun indsaml' [] _ komp = ([],komp)  (* (krop,komp) *)
      | indsaml' ((IF(ub,k0,k1))::krop) var komp =
          if (udtryk.udregnBool var ub) then indsaml' (k0@krop) var komp
          else indsaml' (k1@krop) var komp
      | indsaml' ((SAETNING bv)::krop) var komp =
          let val (nykrop,nykomp) = indsaml' krop var (indsamlIBitvektor bv komp)
          in (bv::nykrop,nykomp) end
      | indsaml' _ _ _ = raise fejlTyper  (* faar ikke plan- og bitvardefs *)

    fun indsaml krop var planer =
          let val kropTyper = indsaml' krop var ([],[])
            val _ = checkNavneoverlap (#1 (#2 kropTyper)) planer
          in kropTyper end

  end
end
```

### stoerrelser.sig

```
signature stoerrelser =
  sig

    (* ???Der mangler indtil videre at blive infereret stoerrelser. Dvs.
     * det paa nuvaerende tidspunkt er noedvendigt at angive stoerrelser
     * for alle komponenter, afloeb mm. *)

    (* Typerne for komponenter, skel, afloeb mm. oversaettes til vaerdier, og
     * hvis nogle stoerrelser ikke er angivet, infereres disse. Hvis en
     * komponents type (f.eks. mux) ikke er angivet, udledes dette af
     * komponentens navn. Tre strukturer returneres indeholdende typer,
     * vaerdier og stoerrelser. Den foerste af disse skal gemmes i
     * i mellemkoden (skal bruges ved erklaering af komponenterne), den
     * naeste naar alle afledninger skal udfyldes, og den sidste naar
     * selve kroppen skal oversaettes til mellemkode ved at blive splittet
     * i elementer.
     * For vektorer laves komponenterne i vektoren og i selve programmet
     * skal disse erstatte vektoren. Har vektoren navn s, faar hver af
     * komponenterne navnet s^idx (dvs. f.eks. "andarray0"). Kun i
     * kompsAndet indgaar selve vektoren for at denne kan erstattes med
     * de enkelte komponenter i selve kroppen. *)

    val udregn : syntaks.bitvektor list -> (string * syntaks.intboolstreng) list
              -> ((syntaks.streng * syntaks.typen) list
                  * (syntaks.pos * syntaks.udtryk) list)
              -> (string * int) list -> (string * int) list -> (string * plan.typ) list
              -> (string * (syntaks.streng * syntaks.intboolstreng list)) list
               * ((string * ((string * (int * int) list) list * int
                             * (string * (int * int) list) list * int
                             * syntaks.bitvektor)) list
                  * (syntaks.streng * string * syntaks.streng * (int * syntaks.pos)) list
                  * (syntaks.pos * int) list)
               * ((string * ((string * string * int * int) list * int
```

```
                        * (string * string * int * int) list * int)) list
                 * (syntaks.streng * string * syntaks.streng * (int * syntaks.pos)) list)
               (* (krop,var,typer,indstik,udstik,planer)
                  -> (komp,kompsAndet,kompStikSkel),
                * hvor komp = [(navn,(typ,vaerdier))]
                * og skal gemmes, mens kompsAndet = (komps,skel,[(pos,str)]),
                * hvor komps = [(navn,([(stikn,ibit0,ibitN)],istr,
                *               [(stikn,ubit0,ubitN)],ustr,vektorBv))]
                * og skal bruges, naar afledningerne skal udfyldes. Hvis navn er en
                * vektor, er vektorBv = SEKV _, og ellers er vektorBv noget andet.
                * kompStikSkel = ([(navn,istik,istr,ustik,ustr)],skel),
                * hvor stik = [(stikn,stiknr,bit0,bitN)],
                * hvor string er stiknavn og int er sidste bit i dette stik. stiknr
                * er , hvis stikket ikke er et vektorstik. Ellers er det "0","1",...
                * Skal bruges, naar kroppen laves til mellemkode.
                * skel = [(navn,typ,komp,bitnr)] hhv [(navn,typ,komp,stik,stiknr,bitnr)].
                * De tre strukturer indeholder data for alle komponenter, der ikke
                * er vektorer, samt for alle komponenter indeni vektorer (f.eks.
                * "addarray0"). Derudover indeholder kompsAndet alle
                * vektorkomponenter. *)

  end
```

### stoerrelser.sml

```
structure stoerrelser :> stoerrelser =
  struct

    local open syntaks fejl
    in

      exception fejlStoerrelser
      exception fejlStoerrelser_ikkeVektor

      (***** Diverse *****)

      (* Returner praefiks af streng, der ender paa "array". Rejs
       * fejlStoerrelser_ikkeVektor, hvis strengen ikke ender paa "array". *)
      fun typArray s =
            let val str = String.size s in
              if str > 5 andalso String.extract(s,str-5,NONE) = "array" then
                String.extract(s,0,SOME(str-5))
              else raise fejlStoerrelser_ikkeVektor
            end

      fun varLav n ((s,ukendt)::varliste) (vaerdi::vaerdier) =
            (s,vaerdi)::(varLav n varliste vaerdier)
        | varLav n ((s,strengtype)::varliste) ((vaerdi as S _)::vaerdier) =
            (s,vaerdi)::(varLav n varliste vaerdier)
        | varLav n ((_,strengtype)::_) (_::_) =
            raise fejl(varTildelesIkkeStreng n)
        | varLav n ((s,booltype)::varliste) ((vaerdi as B _)::vaerdier) =
            (s,vaerdi)::(varLav n varliste vaerdier)
        | varLav n ((_,booltype)::_) (_::_) =
            raise fejl(varTildelesIkkeBool n)
        | varLav n ((s,_)::varliste) ((vaerdi as I _)::vaerdier) =  (* stoer/tal *)
            (s,vaerdi)::(varLav n varliste vaerdier)
        | varLav n ((_,_)::_) (_::_) =  (* stoer/tal *)
            raise fejl(varTildelesIkkeTal n)
        | varLav _ [] [] = []
        | varLav n [] _ = raise fejl(flereVaerdierEndVar n)
        | varLav n _ [] = raise fejl(flereVarEndVaerdier n)

      fun flet (fraTil::rest) ((stikn,stikL)::restL) =
            (stikn,fraTil@stikL)::(flet rest restL)
        | flet [] [] = []
```

```
  | flet _ _ = raise fejlStoerrelser

fun vektorBvLav 0 s = KOMP((s^"0",(0,0)),NIL,[],[])
  | vektorBvLav idx s = SEKV(vektorBvLav (idx-1) s,
                            KOMP((s^(Int.toString idx),(0,0)),NIL,[],[]))

fun vektorLav' iBitSt uBitSt inavne unavne [] = (inavne,iBitSt,unavne,uBitSt)
  | vektorLav' iBitSt uBitSt inavne unavne ((istikL,istr',ustikL,ustr')::rest) =
      let val (indstik,istr,udstik,ustr)
              = vektorLav' (iBitSt+istr') (uBitSt+ustr') inavne unavne rest
          val ind = map (fn (sn,fraTilL)
                         => map (fn (fra,til) => (fra+iBitSt,til+iBitSt)) fraTilL
                         ) istikL
          val ud = map (fn (sn,fraTilL)
                        => map (fn (fra,til) => (fra+uBitSt,til+uBitSt)) fraTilL
                        ) ustikL
          val istikny = flet ind indstik
          val ustikny = flet ud udstik
      in (istikny,istr,ustikny,ustr) end

fun vektorLav ((elm as (istikL,_,ustikL,_))::rest) =
      let val inavne = map (fn (s,_) => (s,[])) istikL
        val unavne = map (fn (s,_) => (s,[])) ustikL
      in vektorLav' 0 0 inavne unavne (elm::rest) end
  | vektorLav [] = raise fejlStoerrelser

(***** Haandtering af stik *****)

fun stikLav' posSt posSl idx feltStr s s' =
      if posSt = posSl then []
      else (s,s',Int.toString idx,posSt,posSt+feltStr-1)
        ::(stikLav' (posSt+feltStr) posSl (idx+1) feltStr s s')

(* Lav bit for hvert stik i en liste. Hvis et stik er ARRAYABSTRACTION,
 * deles det i flere, der hver faar et feltnavn ("0","1",...).  *)
fun stikLav _ [] = []
  | stikLav stPos ((s,s',0,str)::stik) =
      (s,s',,stPos,stPos+str-1)::(stikLav (stPos+str) stik)
  | stikLav stPos ((s,s',feltstr,str)::stik) =  (* vektorstik *)
      (stikLav' stPos (str+stPos) 0 feltstr s s')@(stikLav (stPos+str) stik)

(* Lav ovenstaaende om til liste, hvor hvert stik er slaaet sammen til
 * (stikn,[(fra,til)]). *)
fun stiklLav ((s,_,,fra,til)::rest) = (s,[(fra,til)])::(stiklLav rest)
  | stiklLav ((elm as (s,_,_,_,_))::rest) =  (* vektorstik *)
      let fun split s ((elm' as (s',_,_,fra,til))::rest') =
                if s <> s' then ([],elm'::rest')
                else let val (st,sl) = split s rest'
                     in ((fra,til)::st,sl) end
            | split _ [] = ([],[])
          val (fraTilL,sl) = split s (elm::rest)
      in (s,fraTilL)::(stiklLav sl) end
  | stiklLav [] = []

fun stikStr [] = 0
  | stikStr stik = (fn (_,_,_,_,t) => t+1) (List.last stik)

fun stikUdregn var (s,s',BLANK,u) =
      let val str = udtryk.udregn var u
      in if str <= 0 then raise fejl(stikStoerrelseLE0 (udtryk.pos u))
         else (s,s',0,str)
      end
  | stikUdregn var (s,s',feltu,u) =
      let val str = udtryk.udregn var u
        val feltstr = udtryk.udregn var feltu
      in if feltstr = 0 then raise fejl(stikStoerrelseLE0 (udtryk.pos feltu))
```

```
         else
           if str mod feltstr <> 0 then
             raise fejl(feltinddelingUlovlig (udtryk.pos feltu))
           else (s,s',feltstr,str)
      end

(***** Oversaettelse af typer for komponenter og vektorer *****)

fun kompTypen' varKrop navn uliste varliste indstik udstik =
      let val vaerdier = map (udtryk.udregnTalBoolS varKrop) uliste
        val var = varLav navn varliste vaerdier
        val ind = map (stikUdregn var) indstik
        val ud = map (stikUdregn var) udstik
        val istik' = stikLav 0 ind
        val ustik' = stikLav 0 ud
        val istr = stikStr istik'
        val ustr = stikStr ustik'
        val istikL = stiklLav istik'
        val ustikL = stiklLav ustik'
        val istik = map (fn (_,s',stiknr,bit0,bitN)
                         => (s',stiknr,bit0,bitN)) istik'
        val ustik = map (fn (_,s',stiknr,bit0,bitN)
                         => (s',stiknr,bit0,bitN)) ustik'
      in (vaerdier,istikL,istr,ustikL,ustr,istik,ustik) end

(* Oversaet typen for en komponent til de noedvendige data. *)
fun kompTypen varKrop (navn as (s,_)) typ ubs =
      let val (varliste,indstik,udstik,_,_) = typ.hent typ
        val (vaerdier,istikL,istr,ustikL,ustr,istik,ustik)
          = kompTypen' varKrop navn ubs varliste indstik udstik
      in ([(s,(typ,vaerdier))],[(s,(istikL,istr,ustikL,ustr,UD(0,0)))],
          [],[(s,(istik,istr,ustik,ustr))])
      end

fun vektorKompsLav idx antal varKrop (s,pos) typ ubs varliste indstik udstik =
      if idx = antal then ([],[],[],[])
      else
        let val navn = s^(Int.toString idx)
          val (vaerdier,istikL,istr,ustikL,ustr,istik,ustik)
            = kompTypen' (("index",I(idx))::varKrop) (navn,pos) ubs varliste
                         indstik udstik
          val (komp,komps,kompStik,vektor)
            = vektorKompsLav (idx+1) antal varKrop (s,pos) typ ubs varliste
                             indstik udstik
        in ((navn,(typ,vaerdier))::komp,
            (navn,(istikL,istr,ustikL,ustr,UD(0,0)))::komps,
            (navn,(istik,istr,ustik,ustr))::kompStik,
            (istikL,istr,ustikL,ustr)::vektor)
        end

(* Oversaet typen for en vektor til de noedvendige data. *)
fun vektorTypen varKrop (navn as (s,_)) typ ((udt u)::ubs) =
      let val antal = udtryk.udregn varKrop u
        val vektorBv = if antal <= 0 then raise fejl(vektorUdenElm navn)
                       else vektorBvLav (antal-1) s
        val (varliste,indstik,udstik,_,_) = typ.hent typ
        val (komp,komps,kompStik,vektor)
          = vektorKompsLav 0 antal varKrop navn typ ubs varliste
                           indstik udstik
        val (istikL,istr,ustikL,ustr) = vektorLav vektor
      in (komp,(s,(istikL,istr,ustikL,ustr,vektorBv))::komps,[],kompStik) end
  | vektorTypen _ navn _ _ = raise fejl(vektorUdenAntal navn)

(***** Laes og oversaet alle typer *****)

fun laesTypen' var _ navn NIL =
```

```
      kompTypen var navn (typ.findMaxMatch navn,#2 navn) []
  | laesTypen' var planer navn (TYPE ((udt(STR(s,pos)))::us,_)) =
      ((let val (_,istr,ustr) = liste.hent s planer
            val navns = #1 navn in  (* s er plan *)
          if us <> [] then raise fejl(flereVaerdierEndVar navn)
          else ([(navns,((s,pos),[I(istr),I(ustr)]))],
                [(navns,[("in",[(0,istr-1)])],istr,
                         [("out",[(0,ustr-1)])],ustr,UD(0,0)))],[],
                [(navns,[("in",,0,istr-1)],istr,[("out",,0,ustr-1)],ustr))])
        end) handle liste.fejlListe_ikkeIListe =>
          if (liste.findes s var) then  (* s er variabel *)
            if fkt.delstreng "array" (#1 navn) then
               vektorTypen var navn (typ.findMaxMatch navn,
                                     #2 navn) ((udt(STR(s,pos)))::us)
             else kompTypen var navn (typ.findMaxMatch navn,
                                      #2 navn) ((udt(STR(s,pos)))::us)
          else ((vektorTypen var navn (typArray s,pos) us)  (* s er vektor *)
                handle fejlStoerrelser_ikkeVektor =>
                  kompTypen var navn (s,pos) us))  (* s er komponent *)
  | laesTypen' var _ (navn as (s,pos)) (TYPE (ubs,_)) =
      if fkt.delstreng "array" s then
        vektorTypen var navn (typ.findMaxMatch navn,pos) ubs
      else kompTypen var navn (typ.findMaxMatch navn,pos) ubs
  | laesTypen' var _ (navn as (s,_)) (SKELTYPE((t,pos'),kompn,u)) =
      let val typ
        = (if t = "zap" orelse t = "zapboundary"
             orelse (t =  andalso fkt.delstreng "zap" s) then "ZapBoundary"
           else
             if t = "pipestage" orelse t = "pipestageboundary"
               orelse (t =  andalso fkt.delstreng "pipestage" s)
               then "PipeStageBoundary"
             else if t =  then raise fejl(navnMatcherIkkeSkel navn)
                  else raise fejl(skelFindesIkke(t,pos')))
      in ([],[],[(navn,typ,kompn,(udtryk.udregn var u,udtryk.pos u))],[]) end

fun laesTypen _ _ _ _ ((navn,TYPE(_,(bv,_)::_))::_) =
      (* lige nu er interne forbindelser ulovlige *)
      raise syntaks_internForbindelse (bitvektor.pos bv)
  | laesTypen var indstik udstik planer ((navn as (s,_),typen)::typer) =
      if (liste.findes s indstik) then  (* indstik *)
        if typen <> NIL then raise fejl(stikDefSomKomp navn)
        else laesTypen var (List.tl indstik) udstik planer typer
      else
        if (liste.findes s udstik) then  (* udstik *)
          if typen <> NIL then raise fejl(stikDefSomKomp navn)
          else laesTypen var indstik (List.tl udstik) planer typer
        else  (* navn er PLA, komp eller vektor *)
          let val (komp',komps',skel',kompStik') = laesTypen' var planer navn typen
            val (komp,komps,skel,kompStik)
              = laesTypen var indstik udstik planer typer
          in (komp'@komp,komps'@komps,skel'@skel,kompStik'@kompStik) end
  | laesTypen _ _ _ _ [] = ([],[],[],[])

(***** Global funktion *****)

(* Lav lister med typerne for komponenter, afloeb mm. om til
 * lister med vaerdierne for komponenterne og stoerrelser for
 * afloeb mm.
 * Kroppen skal bruges, hvis stoerrelser skal infereres. *)
fun udregn (krop:bitvektor list) var (komps',andet') indstik udstik planer =
      let val andet = map (fn (pos,BLANK) => raise syntaks_strEjErklaeret pos
                           | (pos,u)     => (pos,udtryk.udregn var u)) andet'
          val (komp,komps,skel,kompStik)
            = laesTypen var indstik udstik planer komps'
      in (komp, (komps,skel,andet), (kompStik,skel)) end
```

```
    end
  end
```

### afled.sig

```
signature afled =
  sig

    (* Alle afledninger i kroppen udfyldes; udtryk udregnes, vaerdier
     * indsaettes i stedet for stik, [..a], [a..] udfyldes og /t laves
     * til /t[0;1;...].
     * Alle afledninger aendres til at have formen BITAF([BITM bitl,...]),
     * hvor bitl er splittet i eventuelle stik (en afledning kan selvfoelgelig
     * ogsaa vaere [TOM]).
     * Det kontrolleres, at stoerrelserne ved alle forbindelser passer,
     * samt at alle bitafledninger ligger indenfor den bitvektor, de
     * afledes fra (det kontrolleres ikke, at hver indbit forbindes netop en
     * gang og hver udbit mindst en gang).
     * Alle skel laves til SKEL(navn,pos) i stedet for KOMP, og der rejses
     * fejl, hvis et skel bruges ulovligt (maa kun bruges som bv >> skel >> bv),
     * eller hvis der laves afledninger paa skel. *)

    val stoerrelse : (syntaks.udtryk * syntaks.udtryk) list -> int

    val udfyld : syntaks.bitvektor list -> (string * syntaks.intboolstreng) list
              -> (string * int) list -> (string * int) list
              -> (string * ((string * (int * int) list) list * int
                            * (string * (int * int) list) list * int
                            * syntaks.bitvektor)) list
                 * (syntaks.streng * string * syntaks.streng * (int * syntaks.pos)) list
                 * (syntaks.pos * int) list
              -> syntaks.bitvektor list
               (* (krop,var,indstik,udstik,(komps,skel,andet)) -> krop,
                * hvor komps indeholder (istik,istr,ustik,ustr), hvor
                * stik = [(stiknavn,[(fra,til)])] for ind- og udstikkene
                * for hver komponent. Grunden til at det er lister er, at i
                * vektorer kommer hvert stik af stikkene i de enkelte
                * elementer (f.eks. kan "inA" vaere [(0..31),(65..96)]. *)

  end
```

### afled.sml

```
structure afled :> afled =
  struct

    local open syntaks fejl
    in

      (* Alle afledninger i kroppen udfyldes; udtryk udregnes, vaerdier
       * indsaettes i stedet for stik, [..a], [a..] udfyldes og /t laves
       * til /t[0;1;...].
       * Alle afledninger aendres til at have formen BITAF([BITM bitl,...]),
       * hvor bitl er splittet i eventuelle stik (en afledning kan selvfoelgelig
       * ogsaa vaere [TOM]).
       * Det kontrolleres, at stoerrelserne ved alle forbindelser passer,
       * samt at alle bitafledninger ligger indenfor den bitvektor, de
       * afledes fra (det kontrolleres ikke, at hver indbit forbindes netop en
       * gang og hver udbit mindst en gang). *)

      exception fejlAfled
      exception fejlAfled_ikkeStik

      (***** Diverse *****)
```

```
fun position ((BITAF((BITM ((u,_)::_))::_))::_) = udtryk.pos u
  | position ((BITAF((FELTER(u,_))::_))::_) = udtryk.pos u
  | position ((BITAF((KOPIER u)::_))::_) = udtryk.pos u
  | position ((TOM pos)::_) = pos
  | position _ = raise fejlAfled

(* Find stoerrelsen af et udfyldt bitmoenster. *)
fun stoerrelse ((TAL(t,_),TAL(t',_))::rest) =
      (if t <= t' then (t'-t+1) else (t-t'+1)) + (stoerrelse rest)
  | stoerrelse [] = 0
  | stoerrelse _ = raise fejlAfled

(* Vend bitmoenster. *)
fun vend bitl = foldl (fn ((u,u'),bitl) => (u',u)::bitl) [] bitl

(* Find et stik ud fra et udtryk. *)
fun findStik (STR(s,_)) stikL =
      ((let val fraTilL = liste.hent s stikL
        in map (fn (fra,til) => (TAL(fra,(0,0)),TAL(til,(0,0)))) fraTilL end)
          handle liste.fejlListe_ikkeIListe => raise fejlAfled_ikkeStik)
  | findStik _ _ = raise fejlAfled_ikkeStik

fun stikTilBitl stikL =
      let val bitl' = List.concat(map (fn (_,fraTilL) => fraTilL) stikL)
      in map (fn (fra,til) => (TAL(fra,(0,0)),TAL(til,(0,0)))) bitl' end

(***** Del bit i stik *****)

(* Lav [t..t'], hvor t <= t', til [bit], hvor hver bit er
 * indeholdt i et stik. [t..t'] er indenfor bitvektoren, og listen af
 * stik daekker denne. *)
fun delIStik bit0 bitN ((_,[])::stikL) = delIStik bit0 bitN stikL
  | delIStik bit0 bitN ((s,(fra,til)::rest)::stikL) =
      let val bitSl = til - fra in  (* sidste bit i fra->til *)
        if bit0 > bitSl then
          delIStik (bit0-bitSl-1) (bitN-bitSl-1) ((s,rest)::stikL)
        else  (* overlap *)
          if bitN <= bitSl then [(TAL(fra+bit0,(0,0)),TAL(fra+bitN,(0,0)))]
          else ((TAL(fra+bit0,(0,0)),TAL(til,(0,0)))
                ::(delIStik 0 (bitN-bitSl-1) ((s,rest)::stikL)))
      end
  | delIStik _ _ _ = raise fejlAfled

(***** Udfyld bitafledning *****)

(* Udfyld bitinterval. Hvis u = u', kan u dog vaere et stik og skal
 * findes i stikL. *)
fun udfyldOgDelIStik var stikL str (u,u') =
      ((let val t = udtryk.udregn var u
            (* hvis u' er BLANK, skal t' vaere sidste bit *)
            val t' = if u' = BLANK then (str-1) else udtryk.udregn var u'
        in if t >= str then raise fejl(afledUdenforBitvektor (udtryk.pos u))
           else if t' >= str
                  then raise fejl(afledUdenforBitvektor (udtryk.pos u'))
                else  (* intervallet er indenfor bitvektoren *)
                  if t <= t' then delIStik t t' stikL
                  else vend (delIStik t' t stikL)
        end) handle fejl(varIkkeDef navn) =>  (* kan vaere stik for komp *)
          (if u <> u' then raise fejl(varIkkeDef navn)
           else findStik u stikL)
             handle fejlAfled_ikkeStik => raise fejl(varIkkeDef navn))

(* Udfyld [u] (u kan vaere stik) og [u..u'] (u,u' kan vaere BLANK). *)
fun udfyldBitl var stikL str bitl =
      let val nybitl = List.concat(map (udfyldOgDelIStik var stikL str) bitl)
      in (nybitl,stoerrelse nybitl) end
```

```
(* Udfyld bitll i FELTER(_,bitll). str er stoerrelsen af et felt. *)
fun udfyldBitll _ str [] =
      (List.tabulate(str, (fn i => [(TAL(i,(0,0)),TAL(i,(0,0)))])),str)
  | udfyldBitll var str bitll =
      let val bitllStr = map (udfyldBitl var [(,[(0,str-1)])] str) bitll
      in foldr (fn ((bitl,str'),(felter,str)) => (bitl::felter,str+str'))
               ([],0) bitllStr
      end

(* Udfyld resten af en afledning (det der kommer efter et eventuelt
 * BITM). Rejs fejl, hvis KOPIER optraeder i indafledning. *)
fun udfyldBitml _ _ true ((KOPIER u)::_) =
      raise fejl(afledIndMedKopier (udtryk.pos u))
  | udfyldBitml var str false ((KOPIER u)::rest) =
      let val t = udtryk.udregn var u
        val (nyrest,nystr) = udfyldBitml var (t*str) false rest
      in ((KOPIER(TAL(t,udtryk.pos u)))::nyrest, nystr) end
  | udfyldBitml var str erInd ((FELTER(u,bitll))::rest) =
      let val t = udtryk.udregn var u in
        if (str mod t) <> 0 then
          raise fejl(feltinddelingUlovlig (udtryk.pos u))
        else
          let val (nybitll,nystr') = udfyldBitll var (str div t) bitll
            val (nyrest,nystr) = udfyldBitml var (t*nystr') erInd rest
          in ((FELTER(TAL(t,udtryk.pos u),nybitll))::nyrest,nystr) end
      end
  | udfyldBitml _ str _ [] = ([],str)
  | udfyldBitml _ _ _ _ = raise fejlAfled  (* faar ikke BITM *)

(* Udfyld en enkelt afledning fra en liste af afledninger. Hver
 * afledning bliver til BITAF([BITM bitl,...]), hvor bitl er
 * splittet om komponentens stik.
 * Der rejses fejl ved TOM, da denne ikke boer bruges mellem
 * ikke-tomme afledninger. *)
fun udfyldAfled _ _ _ _ (TOM pos) =
      raise fejl(afledBrugIkkeTomMedAndre pos)
  | udfyldAfled var stikL str erInd (BITAF((BITM bitl)::rest)) =
      let val (nybitl,nystr') = udfyldBitl var stikL str bitl
        val (nyrest,nystr) = udfyldBitml var nystr' erInd rest
      in (BITAF((BITM nybitl)::nyrest),nystr) end
  | udfyldAfled var stikL str erInd (BITAF rest) =
      let val (nyrest,nystr) = udfyldBitml var str erInd rest
      in (BITAF((BITM (stikTilBitl stikL))::nyrest),nystr) end

(* Udfyld en afledning for en komponent, et stik eller BITV.
 * Alle afledninger BITAF bliver til BITAF([BITM bitl,...]), hvor
 * bitl er splittet om de forskellige stik. *)
fun udfyldAfledl _ _ _ _ [TOM pos] = ([TOM pos], 0)
  | udfyldAfledl _ _ 0 _ afled =
      if afled = [] then ([TOM (0,0)], 0)
      else raise fejl(afledPaaTomBitvektor (position afled))
  | udfyldAfledl _ stikL str _ [] =
      ([BITAF[BITM (stikTilBitl stikL)]],str)
  | udfyldAfledl var stikL str erInd afled =
      let val afledStr = map (udfyldAfled var stikL str erInd) afled in
        foldr (fn ((afled',str'),(afled,acc))
               => (afled'::afled,str'+acc)) ([],0) afledStr
      end

(* Udfyld afledningerne i en bitvektor og kontroller, at alle
 * afledningers stoerrelser i kompositioner af bitvektorer passer,
 * og at alle bit i en afledning falder indenfor bitvektoren. *)
fun udfyldBv _ _ _ (_,_,andet) (AFLOEB(_,pos)) =
      let val str = liste.hentPos pos andet
      in (AFLOEB(TAL(str,pos),pos),str,0) end
```

```
| udfyldBv _ _ _ (_,_,andet) (GENNEMLOEB(_,pos)) =
    let val str = liste.hentPos pos andet
    in (GENNEMLOEB(TAL(str,pos),pos),str,str) end
| udfyldBv _ _ _ (_,_,andet) (KONST(vaerdi,pos,_)) =
    ((let val str = liste.hentPos pos andet
      in (KONST(konst.tilStoerrelse vaerdi str,pos,TAL(str,pos)),0,str) end)
        handle konst.fejlKonst_konstForKort =>
          raise fejl(konstLaengereEndStoerrelse pos))
| udfyldBv var indstik udstik (komps,skel,_) (KOMP(n as (s,_),_,ind,ud)) =
    if List.exists (fn ((s',_),_,_,_) => s' = s) skel then
      raise fejl(skelUlovligBrug n)
    else
      ((let val (istr',ustr') = ((liste.hent s udstik, 0)
                                 handle liste.fejlListe_ikkeIListe =>
                                   (0, liste.hent s indstik)) (* n er stik *)
            val (nyind,istr)
              = udfyldAfledl var [(,[(0,istr'-1)])] istr' true ind
            val (nyud,ustr)
              = udfyldAfledl var [(,[(0,ustr'-1)])] ustr' false ud
        in (KOMP(n,NIL,nyind,nyud),istr,ustr) end)  (* stik *)
          handle liste.fejlListe_ikkeIListe =>  (* n er komponent *)
            let val (indL,istr',udL,ustr',vektorBv) = liste.hent s komps
              val (nyind,istr) = udfyldAfledl var indL istr' true ind
              val (nyud,ustr) = udfyldAfledl var udL ustr' false ud
              val komp = (case vektorBv of
                            SEKV _ =>  (* vektor *)
                              let val (bv,_,_) = udfyldBv [] [] []
                                                          (komps,skel,[]) vektorBv
                              in BITV(bv,nyind,nyud) end
                          | _ => KOMP(n,NIL,nyind,nyud))  (* ikke vektor *)
            in (komp, istr, ustr) end)
| udfyldBv var indstik udstik kompsAndet (BITV(bv,ind,ud)) =
    let val (nybv,istr',ustr') = udfyldBv var indstik udstik kompsAndet bv
      val (nyind,istr) = udfyldAfledl var [(,[(0,istr'-1)])] istr' true ind
      val (nyud,ustr) = udfyldAfledl var [(,[(0,ustr'-1)])] ustr' false ud
    in (BITV(nybv,nyind,nyud), istr, ustr) end
| udfyldBv var indstik udstik (kompsAndet as (_,skel,_))
      (FORBIND(bv,k as KOMP(n as (s,_),_,i,u))) =
    let val (nybv,istr,ustr) = udfyldBv var indstik udstik kompsAndet bv in
      if List.exists (fn ((s',_),_,_,_) => s' = s) skel then  (* k er skel *)
        if i <> [] then raise fejl(skelMedAfled (position i))
        else if u <> [] then raise fejl(skelMedAfled (position u))
             else  (* "overspring" skellet *)
               (FORBIND(nybv,SKEL n),istr,ustr)
      else  (* k er ikke skel *)
        let val (nyk,istr',ustr') = udfyldBv var indstik udstik kompsAndet k in
          if ustr <> istr' then raise fejl(udIndForskIForbind(bitvektor.pos k))
          else (FORBIND(nybv,nyk),istr,ustr')
        end
    end
| udfyldBv var indstik udstik kompsAndet (FORBIND(bv,bv')) =
    let val (nybv,istr,ustr') = udfyldBv var indstik udstik kompsAndet bv
      val (nybv',istr',ustr) = udfyldBv var indstik udstik kompsAndet bv'
    in if ustr' <> istr' then raise fejl(udIndForskIForbind (bitvektor.pos bv'))
       else (FORBIND(nybv,nybv'), istr, ustr)
    end
| udfyldBv var indstik udstik kompsAndet (SPLIT(bv,bv')) =
    let val (nybv,istr,ustr) = udfyldBv var indstik udstik kompsAndet bv
      val (nybv',istr',ustr') = udfyldBv var indstik udstik kompsAndet bv'
    in if istr <> istr' then raise fejl(indForskISplit (bitvektor.pos bv'))
       else (SPLIT(nybv,nybv'), istr, ustr+ustr')
    end
| udfyldBv var indstik udstik kompsAndet (SEKV(bv,bv')) =
    let val (nybv,istr,ustr) = udfyldBv var indstik udstik kompsAndet bv
      val (nybv',istr',ustr') = udfyldBv var indstik udstik kompsAndet bv'
    in (SEKV(nybv,nybv'), istr+istr', ustr+ustr') end
```

```
        | udfyldBv _ _ _ _ _ = raise fejlAfled

      (***** Hovedfunktionen *****)

      fun udfyld krop var indstik udstik kompsAndet =
            let fun fejlVSkel (FORBIND(_,SKEL n),_,_) = raise fejl(skelUlovligBrug n)
                  | fejlVSkel (bv,_,_) = bv
            in map (fn bv
                    => (fejlVSkel (udfyldBv var indstik udstik kompsAndet bv))) krop
            end

    end
  end
```

## split.sig

```
signature split =
  sig

    (* Kroppen forvandles til en liste af (bvElm,bvElm), der specificerer
     * forbindelsen af to stik. Gennemloeb udskiftes, og det kontrolleres
     * desuden, at alle indbit er forbundet netop en gang og alle udbit
     * mindst en gang.
     * Kroppen forventes at vaere paa en form, hvor alle bitl i afledninger
     * er splittet i stik (er add f.eks. 32-bit adder, skal [0..35] vaere
     * paa formen [0..31,32..35]). Desuden skal alle afledninger vaere
     * udfyldt, saa de er fuldt specificeret ([..2], [2..] og /2 er ulovlige)
     * og ikke indeholder variable eller stik ([2*t] og [in] er ulovlige).
     * Alle afledninger skal starte med [t...,...t'], dvs. vaere paa formen
     * BITAF[BITM bitl,...].
     * Udfylder ogsaa en liste med skel med stiknavn og eventuelt stiknr samt
     * kontrollerer, at komponenten faktisk eksisterer og at bitNr er indenfor
     * komponentens udbit. *)

    (* Afledninger i mellemkoden. *)
    datatype simsysAfled =
      subset of int * int         (* (bit0,lgd) *)
    | reverse

    (* Elementer i mellemkoden. Stiknavn er navnet paa stikket, f.eks. "in",
     * man kan ogsaa vaere , hvis komponenter selv er et stik. stiknr er
     * , hvis stikket stiknavn ikke er ARRAYABSTRACTION, ellers er det "0",
     * "1" osv. og angiver, hvilket stik i vektoren, der skal bruges, f.eks.
     * mux.in[0]. *)
      datatype bvElm =
        kelm of string * string * string * simsysAfled list * string list
      (* komp: navn,stiknavn,stiknr,afled,skel *)
      | aelm                                        (* afloeb *)
      | belm of konst.typ * string list             (* konst: vaerdi,skel *)

    (* Oversaetter kroppen til liste af (bvElm,bvElm), der angiver uddata
     * fra et stik til inddata for et andet stik. Der rejses fejl, hvis
     * ind- eller udbit for en komponent eller et stik ikke er forbundet.
     * Gennemloeb fjernes. *)
    val tilMellemkode : syntaks.bitvektor list
                    -> (string * int) list -> (string * int) list
                    -> ((string * ((string * string * int * int) list * int
                                * (string * string * int * int) list * int)) list
                        * (syntaks.streng * string * syntaks.streng
                           * (int * syntaks.pos)) list)
                    -> (bvElm * bvElm) list
                     * (string * string * string * string * string * int) list
                      (* (krop,indstik,udstik,kompStikSkel) -> (nykrop,nyskel)
                       * hvor nykrop = [(bvElm,bvElm)],
                       * kompStik = [(navn,([(istiknavn,stiknr,bit0,bitN)],istr,
                                           [(ustiknavn,stiknr,bit0,bitN)],ustr))],
```

```
                      * hvor navn er komponent i kroppen.
                      * nyskel = [(navn,typ,komp,stikn,stiknr,bitnr)]. *)

  end
```

### split.sml

```
structure split :> split =
  struct

    local open syntaks fejl
    in

      (* Kroppen forvandles til en liste af (bvElm,bvElm), der specificerer
       * forbindelsen af to stik. Gennemloeb udskiftes, og det kontrolleres
       * desuden, at alle indbit er forbundet netop en gang og alle udbit
       * mindst en gang.
       * Kroppen forventes at vaere paa en form, hvor alle bitl i afledninger
       * er splittet i stik (er add f.eks. 32-bit adder, skal [0..35] vaere
       * paa formen [0..31,32..35]). Desuden skal alle afledninger vaere
       * udfyldt, saa de er fuldt specificeret ([..2], [2..] og /2 er ulovlige)
       * og ikke indeholder variable eller stik ([2*t] og [in] er ulovlige).
       * Alle afledninger skal starte med [t...,...t'], dvs. vaere paa formen
       * BITAF[BITM bitl,...]. *)

      exception fejlSplit
      exception fejlSplit_ingenGloeb

      (* Gem her (s,pos,[(bit0,bitN)]) for hhv. ind- og
       * udbit for alle komponenter. Bittene findes, naar K konverteres til kelm
       * samt i skelerklaeringerne. Der skal vaere pos paa, da der foerst
       * rejses fejl til sidst, hvis bittene ikke er forbundet. pos er foerste
       * pos, hvor split stoeder paa s. Her laegges ogsaa (s,[]) ind i indbit
       * for at sikre, at der ogsaa rejses fejl, hvis ingen indbit er angivet. *)
      val indbit = ref []
      val udbit = ref []

      (* Afledninger i mellemkoden. *)
      datatype simsysAfled =
        subset of int * int         (* (bit0,lgd) *)
      | reverse

      (* Elementer i mellemkoden. Stiknavn er navnet paa stikket, f.eks. "in",
       * man kan ogsaa vaere , hvis komponenter selv er et stik. stiknr er
       * , hvis stikket stiknavn ikke er ARRAYABSTRACTION, ellers er det "0",
       * "1" osv. og angiver, hvilket stik i vektoren, der skal bruges, f.eks.
       * mux.in[0]. *)
      datatype bvElm =
        kelm of string * string * string * simsysAfled list * string list
      (* komp: navn,stiknavn,stiknr,afled,skel *)
      | aelm                                        (* afloeb *)
      | belm of konst.typ * string list             (* konst: vaerdi,skel *)

      (* Type noedvendig til intern repraesentation (string list er skel). *)
      datatype splitElm =
        K of string * pos * int * int * string list  (* kompnavn,bit0,bitN,skel *)
      | A
      | BIN of konst.typ * string list
      | G of pos * int * int * string list           (* pos,bit0,bitN,skel *)
      | S of (splitElm * int) list list

      (***** Haandter indbit og udbit *****)

      fun iubitStart s pos =
            let fun bitStart s pos ((e as (s',_,_))::bit) =
                      if s < s' then (s,pos,[])::e::bit
```

```
              else if s = s' then e::bit else e::(bitStart s pos bit)
            | bitStart s pos [] = [(s,pos,[])]
      in (indbit := bitStart s pos (!indbit) ;
          udbit := bitStart s pos (!udbit)) end

(* Tilfoej et interval til udbitlisten [(bit0,bitN)]. Soerg for at
 * overlappende intervaller slaas sammen. bit0 <= bitN. *)
fun utilfoej bit0 bitN [] = [(bit0,bitN)]
  | utilfoej bit0 bitN ((bit0',bitN')::rest) =
      if bitN < bit0'-1 then (bit0,bitN)::(bit0',bitN')::rest
      else if bit0 > bitN'+1 then (bit0',bitN')::(utilfoej bit0 bitN rest)
           else utilfoej (Int.min(bit0,bit0')) (Int.max(bitN,bitN')) rest

fun ubitTilfoej s bit0 bitN =
      let fun tilfoej s ((e as (s',pos,ubit))::rest) =
                if s = s' then (s,pos,utilfoej (Int.min(bit0,bitN))
                                (Int.max(bit0,bitN)) ubit)::rest
                else e::(tilfoej s rest)
            | tilfoej _ [] = raise fejlSplit
      in (udbit := tilfoej s (!udbit)) end

(* Tilfoej et interval til indbitlisten [(bit0,bitN)]. Rejs fejl ved
 * overlap. Soerg for at tilstoedende intervaller slaas sammen.
 * bit0 <= bitN. *)
fun itilfoej pos bit0 bitN [] = [(bit0,bitN)]
  | itilfoej pos bit0 bitN ((bit0',bitN')::rest) =
      if bitN < bit0'-1 then (bit0,bitN)::(bit0',bitN')::rest
      else if bit0 > bitN'+1 then (bit0',bitN')::(itilfoej pos bit0 bitN rest)
           else if bitN = bit0'-1 then (bit0,bitN')::rest
                else if bit0 = bitN'+1 then itilfoej pos bit0' bitN rest
                     else raise fejl(indbitOverlapper pos)

fun ibitTilfoej s pos bit0 bitN =
      let fun tilfoej s ((e as (s',pos',ibit))::rest) =
                if s = s' then (s,pos',itilfoej pos (Int.min(bit0,bitN))
                                (Int.max(bit0,bitN)) ibit)::rest
                else e::(tilfoej s rest)
            | tilfoej _ [] = raise fejlSplit
      in (indbit := tilfoej s (!indbit)) end

fun iubitTilfoej s pos bit0 bitN erInd =
      if erInd then ibitTilfoej s pos bit0 bitN
      else ubitTilfoej s bit0 bitN

(* Checker om bit 0->(str-1) alle er forbundet. Returnerer -1, hvis
 * dette er tilfaeldet og ellers en uforbunden bit. *)
fun uforbundet 0 _ = ~1
  | uforbundet str [(0,bitN)] = if bitN = str-1 then ~1 else bitN+1
  | uforbundet _ ((_,bitN)::(_,_)::_) = bitN+1
  | uforbundet _ _ = 0  (* bitl = [] ell. [(b0,_)] hvor b0>0 *)

fun iubitCheck' _ _ _ [] [] = ()
  | iubitCheck' kompStik indstik udstik ((s,pos,ibit)::ibitl)
        ((s',pos',ubit)::ubitl) =
      if s <> s' then raise fejlSplit
      else
        let val (_,istr,_,ustr) =
          (((liste.hent s kompStik) handle liste.fejlListe_ikkeIListe =>
            ([],0,[],liste.hent s indstik)) handle liste.fejlListe_ikkeIListe =>
            ([],liste.hent s udstik,[],0))
          val ib = uforbundet istr ibit
          val ub = uforbundet ustr ubit
        in if ib >= 0 then raise fejl(indbitIkkeForbundet(s,pos,ib))
           else if ub >= 0 then raise fejl(udbitIkkeForbundet(s,pos',ub))
                else iubitCheck' kompStik indstik udstik ibitl ubitl
        end
```

```
  | iubitCheck' _ _ _ _ _ = raise fejlSplit

(* Kontroller at alle ind- og udbit er forbundet. *)
fun iubitCheck kompStik indstik udstik =
      iubitCheck' kompStik indstik udstik (!indbit) (!udbit)

(***** Haandter gloebliste = [(pos,[(st,elm)])] *****)

(* Indsaet (st,skel,elm) paa rette plads i liste sorteret efter bitSt.
 * GLOEB >> skel >> elm. *)
fun gloebIndsaet' st skel elm [] = [(st,skel,elm)]
  | gloebIndsaet' st skel elm ((st',skel',elm')::rest) =
      if st < st' then (st,skel,elm)::(st',skel',elm')::rest
      else (st',skel',elm')::(gloebIndsaet' st skel elm rest)  (* st > st' *)

(* Indsaet elm paa rette plads i vektoren for g.loeb pos. *)
fun gloebIndsaet pos st skel elm ((pos',stElm)::gloebL) =
      if pos = pos' then (pos,gloebIndsaet' st skel elm stElm)::gloebL
      else if (fkt.lessPos pos pos')
             then (pos,[(st,skel,elm)])::(pos',stElm)::gloebL
           else (pos',stElm)::(gloebIndsaet pos st skel elm gloebL)
  | gloebIndsaet pos st skel elm [] = [(pos,[(st,skel,elm)])]

(***** Diverse *****)

(* Find stik indeholdende bit t for komponenten s. *)
fun findStik s t kompStik erInd =
      let val (istik,_,ustik,_) = liste.hent s kompStik
        fun findStik' t ((stik as (_,_,_,sl))::rest) =
              if t<=sl then stik else findStik' t rest
          | findStik' _ [] = (test "0";raise fejlSplit)
      in if erInd then (findStik' t istik) else (findStik' t ustik) end

fun konverterElm _ _ _ (A,_) = aelm
  | konverterElm _ _ _ (BIN(vaerdi,skel),_) = belm(vaerdi,skel)
  | konverterElm stik kompStik erInd (K(s,pos,bit0,bitN,skel),_) =
      let val ssafled' = if bit0 > bitN then [reverse] else []
        val (min,max) = (Int.min(bit0,bitN),Int.max(bit0,bitN))
        val str = max - min + 1
      in if liste.findes s stik  (* stik ind i ell. ud af programblokken *)
           then (iubitTilfoej s pos bit0 bitN erInd
                 ; kelm(s,,,(subset(min,str))::ssafled',skel))
         else  (* komp i blokken *)
           let val (stikn,stiknr,bitSt,bitSl) = findStik s bit0 kompStik erInd
             val ssafled = if bit0 = bitSt andalso bitN = bitSl then ssafled'
                           else (subset(min-bitSt,str))::ssafled'
           in (iubitTilfoej s pos bit0 bitN erInd
               ; kelm(s,stikn,stiknr,ssafled,skel)) end
      end
  | konverterElm _ _ _ _ = (test "8";raise fejlSplit)

(* Find stikdata for skel. Rejs fejl, hvis oplysninger i skellet er
 * ukorrekte. *)
fun skelLav kompStik ((s,_),typ,(komp as (komps,_)),(bitnr,pos)) =
       ((let val (stikn,stiknr,bit0,_) = findStik komps bitnr kompStik false
             val nybitnr = bitnr - bit0
        in (ubitTilfoej komps nybitnr nybitnr
            ; (s,typ,komps,stikn,stiknr,nybitnr)) end)
          handle liste.fejlListe_ikkeIListe => raise fejl(kompFindesIkke komp)
               | fejlSplit => raise fejl(afledUdenforBitvektor pos))

fun skelPaaElm (A,str) _ = (A,str)
  | skelPaaElm (BIN(vaerdi,skel),str) skel' = (BIN(vaerdi,skel@skel'),str)
  | skelPaaElm (K(s,pos,bit0,bitN,skel),str) skel' =
      (K(s,pos,bit0,bitN,skel@skel'),str)
  | skelPaaElm _ _ = (test "7";raise fejlSplit)
```

```
(* Vend "bitvektoren" [(elm,str)] samt hvert element. *)
fun vendBL BL = foldl (fn (elm,BL) => (vendElm elm)::BL) [] BL
and vendElm (K(s,pos,bit0,bitN,skel),str) = (K(s,pos,bitN,bit0,skel),str)
  | vendElm (A,str) = (A,str)
  | vendElm (BIN(vaerdi,skel),str) = (BIN(konst.spejles vaerdi,skel),str)
  | vendElm (S(BLL),str) = (S(map vendBL BLL),str)
  | vendElm (G(pos,bit0,bitN,skel),str) = (G(pos,bitN,bit0,skel),str)

(* Split BL lige efter bit b. *)
fun splitBLskel b ((skel,elm as (_,str))::rest) =
      if b = str then ([(skel,elm)],rest)
      else
        if b < str then
          ([(skel,intervalFraElm elm 0 (b-1))],
           (skel,intervalFraElm elm b (str-1))::rest)
        else let val (BLst,BLsl) = splitBLskel (b-str) rest
             in ((skel,elm)::BLst,BLsl) end
  | splitBLskel _ _ = (test "9";raise fejlSplit)

(***** Tag interval t->t' fra BL, hvor t' kan vaere mindre end t *****)

(* Tag delmaengde af et (bvElm,str), t<=t' og t'<str. *)
and intervalFraElm (S BLL,_) t t' = (S(map (intervalFraBL' t t') BLL),t'-t+1)
  | intervalFraElm (BIN(vaerdi,skel),_) t t' =
      (BIN(konst.delKonst vaerdi t t',skel),t'-t+1)
  | intervalFraElm (A,_) t t' = (A,t'-t+1)
  | intervalFraElm (K(s,pos,bit0,bitN,skel),_) t t' =
      if bit0 <= bitN then (K(s,pos,bit0+t,bit0+t',skel),t'-t+1)
      else  (K(s,pos,bit0-t,bit0-t',skel),t'-t+1)
  | intervalFraElm (G(pos,bit0,bitN,skel),_) t t' =
      if bit0 <= bitN then (G(pos,bit0+t,bit0+t',skel),t'-t+1)
      else  (G(pos,bit0-t,bit0-t',skel),t'-t+1)

(* Tag bit t->t' ud af [(splitElm,str)], t<=t'. *)
and intervalFraBL' t t' ((elm as (_,str))::rest) =
      if t >= str then intervalFraBL' (t-str) (t'-str) rest
      else  (* tag noget af (eller hele) elm *)
        if t' < str then [intervalFraElm elm t t']
        else (intervalFraElm elm t (str-1))::(intervalFraBL' 0 (t'-str) rest)
  | intervalFraBL' t t' [] = (test "1";raise fejlSplit)

(* Tag interval t->t' fra BL. Muligvis er t > t'. *)
fun intervalFraBL t t' BL =
      if t <= t' then intervalFraBL' t t' BL
      else vendBL (intervalFraBL' t' t BL)

(***** Tag afledning fra BL eller komponent *****)

(* Tager bit ud af [(splitElm,str)]. *)
fun bitFraBL BL (TAL(t,_),TAL(t',_)) = intervalFraBL t t' BL
  | bitFraBL _ _ = (test "2";raise fejlSplit)  (* alle afledninger er udregnet *)

(* Lav antal felter og tag fra disse felter bittene specificeret af
 * afledningen bitl. str er stoerrelsen af hvert felt efter dette
 * er gjort.  *)
fun tagFelter [(A,_)] antal str _ = [(A,antal*str)]
  | tagFelter [(S BLL,_)] antal str bitl =
      [(S(map (fn BL => tagFelter BL antal str bitl) BLL),antal*str)]
  | tagFelter BL antal _ bitl =
      let val str = (List.foldl (fn ((_,str'),acc) => str'+acc) 0 BL) div antal in
        (* lav antal felter, og fra hvert felt tages bitl *)
        List.concat(List.tabulate
                    (antal,(fn i =>
                            let val nyBL = intervalFraBL' (i*str) ((i+1)*str-1) BL
                            in List.concat(map (bitFraBL nyBL) bitl) end)))
```

```
      end

(* Tag bitm ud af [(splitElm,str)]. *)
fun bitmAfBL(KOPIER(TAL(t,_)), BL) = List.concat(List.tabulate(t, (fn _ => BL)))
  | bitmAfBL(BITM bitl, BL) = List.concat(map (bitFraBL BL) bitl)
  | bitmAfBL(FELTER(TAL(t,_),bitll), BL) =
      let val str = afled.stoerrelse (hd bitll)
      in List.concat(map (tagFelter BL t str) bitll) end
  | bitmAfBL _ = (test "3";raise fejlSplit)  (* alle afledninger er udregnet *)

(* Tag bitml ud af liste [(elm,str)]. *)
fun afledAfBL BL (BITAF bitml) = foldl bitmAfBL BL bitml
  | afledAfBL _ (TOM _) = (test "4";raise fejlSplit)  (* TOM fjernet i afled.sml *)

(* Lav komponent med afledning om til [(elm,str)],  *)
fun afledAfKomp s pos skel (BITAF ((BITM bitl)::rest)) =
      let fun kompTilBL s pos skel (TAL(t,_),TAL(t',_)) =
                [(K(s,pos,t,t',skel), Int.max(t,t')-Int.min(t,t')+1)]
            | kompTilBL _ _ _ _ = raise fejlSplit
          val BL = List.concat(map (kompTilBL s pos skel) bitl)
      in afledAfBL BL (BITAF rest) end
  | afledAfKomp _ _ _ _ = (test "6";raise fejlSplit)

(***** Split bitvektor i ([(elmUd,str)],[(elmInd,str)]) *****)

(* Lav ind i _ >> ind til [(elmInd,str)]. *)
fun ind (FORBIND(v,_)) = ind v
  | ind (SPLIT(v,h)) =
      ((let val BLv = ind v
            val BLh = ind h
        in case BLv of
          [(S(BLL),str)] => [(S(BLL@[BLh]),str)]
        | _              => [(S([BLv,BLh]),foldl (fn ((_,str'),str)
                                                  => str'+str) 0 BLv)]
        end) handle fejl(bitvektorIndErTom pos) =>
          raise fejl(bitvektorISplitTom pos))
  | ind (SEKV(v,h)) =
      ((((ind v)@(ind h)) handle fejl(bitvektorIndErTom _) => ind h)
        handle fejl(bitvektorIndErTom _) => ind v)
        (* hvis v og h begge tomme, rejses fejlen i v *)
  | ind (BITV(bv,[TOM pos],_)) = raise fejl(bitvektorIndErTom pos)
  | ind (BITV(bv,afled,_)) =
      let val BL = ind bv
      in List.concat(map (afledAfBL BL) afled) end
  | ind (KONST(_,pos,_)) = raise fejl(bitvektorIndErTom pos)
  | ind (AFLOEB(TAL(t,_),_)) = [(A,t)]
  | ind (KOMP((_,pos),_,[TOM _],_)) = raise fejl(bitvektorIndErTom pos)
  | ind (KOMP((s,pos),_,afled,_)) =
      List.concat(map (afledAfKomp s pos []) afled)
  | ind (GENNEMLOEB(TAL(t,_),pos)) = [(G(pos,0,t-1,[]),t)]
  | ind (SKEL n) = raise fejl(skelUlovligBrug n)
  | ind _ = (test "10";raise fejlSplit)

(* Lav ud i ud >> _ til [(elmUd,str)]. *)
fun ud skel (FORBIND(_,h)) = ud skel h
  | ud _ (BITV(_,_,[TOM pos])) = raise fejl(bitvektorUdErTom pos)
  | ud skel (BITV(bv,_,afled)) =
      let val BL = ud skel bv in List.concat(map (afledAfBL BL) afled) end
  | ud _ (AFLOEB(_,pos)) = raise fejl(bitvektorUdErTom pos)
  | ud _ (KOMP((_,pos),_,_,[TOM _])) = raise fejl(bitvektorUdErTom pos)
  | ud skel (KOMP((s,pos),_,_,afled)) =
      List.concat(map (afledAfKomp s pos skel) afled)
  | ud skel (KONST(vaerdi,_,TAL(t,_))) = [(BIN(vaerdi,skel),t)]
  | ud skel (GENNEMLOEB(TAL(t,_),pos)) = [(G(pos,0,t-1,skel),t)]
  | ud _ (SKEL n) = raise fejl(skelUlovligBrug n)
  | ud skel bv =
```

```
      let val (v,h) = bitvektor.par bv in
        ((((ud skel v)@(ud skel h))
             handle fejl(bitvektorUdErTom _) => ud skel h)
             handle fejl(bitvektorUdErTom _) => ud skel v)
      end

(* Split bitvektor i ([(elmUd,str)],[(elmInd,str)]). *)
fun split skel (FORBIND(FORBIND(v,SKEL (s,_)),h)) =
      split (s::skel) (FORBIND(v,h))
  | split skel (FORBIND(v,h)) = (ud skel v,ind h)::(split [] v)@(split [] h)
  | split skel (BITV(bv,_,_)) = split skel bv
  | split _ (KOMP((s,pos),_,_,_)) = (iubitStart s pos ; [])
  | split skel bv =
      ((let val (v,h) = bitvektor.par bv in
          (split skel v)@(split skel h)
        end) handle bitvektor.fejlBitvektor_ikkeSammensat => [])

(***** Lav (BL,BL) til [(elmUd,elmInd)], hvor S (split) er
 * elimineret *****)

fun tilElmElm(BLu, (S(BLL),str)::BLi) =
      let val BLuSt' = intervalFraBL 0 (str-1) BLu
        val kopier = List.length(BLL) - 1  (* kopier start af BLu passende *)
        val BLuSt = List.concat(List.tabulate(kopier,(fn _ => BLuSt')))
      in tilElmElm(BLuSt@BLu,(List.concat BLL)@BLi) end
  | tilElmElm((e as (_,str))::BLu, (e' as (_,str'))::BLi) =
      if str = str' then (e,e')::(tilElmElm(BLu, BLi))
      else if str < str' then
             (e,intervalFraElm e' 0 (str-1))
             ::(tilElmElm(BLu, (intervalFraElm e' str (str'-1))::BLi))
           else  (* str > str' *)
             (intervalFraElm e 0 (str'-1),e')
             ::(tilElmElm((intervalFraElm e str' (str-1))::BLu, BLi))
  | tilElmElm([],[]) = []
  | tilElmElm _ = (test "11";raise fejlSplit)

(***** Fjern gennemloeb og konverter til bvElm *****)

(* Split BL (paa hoejresiden af G >> A...B) om gennemloeb. *)
fun gloebSplitOm ((skel,(G(pos,st,_,skel'),str))::rest) =
      ([],(pos,st,st+str-1,skel@skel'),rest)
  | gloebSplitOm (elm::rest) =
      let val (BLst,gloeb,BLsl) = gloebSplitOm rest
      in (elm::BLst,gloeb,BLsl) end
  | gloebSplitOm [] = raise fejlSplit_ingenGloeb

(* Eliminer eventuelle gennemloeb paa hoejresiden af G >> A...B. Den
 * relevante del flyttes fra G' (i A...B) til hoejresiden af G>>... *)
fun gloebFjern' [] alleGloeb = alleGloeb
  | gloebFjern' ((pos,BL)::gloebV) alleGloeb =
      (let val (BLst,(pos',st',sl',skel),BLsl) = gloebSplitOm BL
           val BL' = map (fn (skel',e)
                          => (skel@skel',e)) (liste.hentPos pos' alleGloeb)
           val (st,sl) = if st'<=sl' then (st',sl') else (sl',st')
           val (BLst',BLrest) = splitBLskel st BL'
           val (BLm',BLsl') = splitBLskel (sl-st+1) BLrest
           (* vend BLm', hvis st'>sl' *)
           val BLm = if st'<=sl' then BLm'
                     else map (fn (skel,e) => (skel,vendElm e)) BLm'
       in gloebFjern' ((pos,BLst@BLm@BLsl)::gloebV)
         (liste.opdaterPos pos' (BLst'@BLsl') alleGloeb)
       end) handle fejlSplit_ingenGloeb => gloebFjern' gloebV alleGloeb

fun gloebTilElmElm ((e as (_,str))::BLu) ((skel,e' as (_,str'))::BLi) =
      if str = str' then (skelPaaElm e skel,e')::(gloebTilElmElm BLu BLi)
      else if str < str' then
```

```
              (skelPaaElm e skel,intervalFraElm e' 0 (str-1))
              ::(gloebTilElmElm BLu ((skel,intervalFraElm e' str (str'-1))::BLi))
            else  (* str > str' *)
              (skelPaaElm (intervalFraElm e 0 (str'-1)) skel,e')
              ::(gloebTilElmElm ((intervalFraElm e str' (str-1))::BLu) BLi)
    | gloebTilElmElm [] [] = []
    | gloebTilElmElm _ _ = (test "12";raise fejlSplit)

  (* Flet lister med [(G(pos)>>A...B)] og [(C...D>>G(pos))] sammen til
   * [((C...D)>>(A...B))] og split denne i [(elmUd,elmInd)]. *)
  fun gloebFlet [] [] eeLL = List.concat eeLL
    | gloebFlet ((pos,BL)::gloebV) ((pos',BL')::gloebH) eeLL =
      if pos <> pos' then (test "13";raise fejlSplit)
      else gloebFlet gloebV gloebH ((gloebTilElmElm BL' BL)::eeLL)
    | gloebFlet _ _ _ = raise (test "14";fejlSplit)

  (* Fjern alle g.loeb fra [(elm,elm)] samt oversaet alle elm til bvElm.
   * For alle g.loeb G med A...B >> G, G >> C...D laves [(elm,elm)] for
   * A...B >> C...D, og denne tilsaettes slutlisten. *)
  fun gloebFjern indstik udstik kompStik (((G(pos,st,_,skel),_), e)::eeL)
          (gloebV,gloebH) =
        gloebFjern indstik udstik kompStik eeL
          (gloebIndsaet pos st skel e gloebV,gloebH)
    | gloebFjern indstik udstik kompStik ((e, (G(pos,st,_,_),_))::eeL)
          (gloebV,gloebH) =
        gloebFjern indstik udstik kompStik eeL
          (gloebV,gloebIndsaet pos st [] e gloebH)
    | gloebFjern indstik udstik kompStik ((e, e')::eeL) gloeb =
        (konverterElm indstik kompStik false e,
         konverterElm udstik kompStik true e')
        ::(gloebFjern indstik udstik kompStik eeL gloeb)
    | gloebFjern _ _ _ [] ([],[]) = []
    | gloebFjern indstik udstik kompStik [] (gloebV,gloebH) =
        if gloebV = [] orelse gloebH = [] then raise (test "15";fejlSplit)
        else  (* fkt til at lave [(pos,[(bit0,elm)])] til [(pos,BL)] *)
          let val nygloebH = map (fn (pos,stElm)
                                   => (pos,map (fn (_,_,e) => e) stElm)) gloebH
            val nygloebV' = map (fn (pos,stElm)
                                 => (pos,map (fn (_,skel,e)
                                              => (skel,e)) stElm)) gloebV
            val nygloebV = gloebFjern' nygloebV' nygloebV'
            val elmelmL = gloebFlet nygloebV nygloebH []
          in map (fn (e,e')
                  => (konverterElm indstik kompStik false e,
                      konverterElm udstik kompStik true e')) elmelmL
          end

  (***** Oversaet kroppen til mellemkode *****)

  (* Oversaet liste af bitvektorer til liste med (elmUd,elmInd), hvor
   * et elm er en komponent, et afloeb eller en konstant. Gennemloeb
   * erstattes med de vaerdier, deres data loeber fra og til. *)
  fun tilMellemkode bvlist indstik udstik (kompStik,skel) =
        (* split kroppen (bvlist) i [(ud,ind)] uden SPLIT paa yderste niveau *)
        let val () = (indbit := [] ; udbit := [])
          val BLBLL
            = List.concat(map (split []) bvlist)
          (* lav [([Bud],[Bind])] til [(elmUd,elmInd)] *)
          val eeL = List.concat(map tilElmElm BLBLL)
          val nyskel = map (skelLav kompStik) skel
          val bvvl = gloebFjern indstik udstik kompStik eeL ([],[])
        in (iubitCheck kompStik indstik udstik ; (bvvl, nyskel)) end
end
end
```

### simsys.sig

```
signature simsys =
  sig

    (* Oversaetter mellemkoden til fil med SimSyskode. *)

    val oversaet : unit -> string list * string list
                (* () -> (h-filer,cc-filer???) *)

  end
```

### simsys.sml

```
structure simsys :> simsys =
  struct

    (***** Oversaetter mellemkoden til fil med SimSyskode. *****)

    (***** Diverse *****)

    val G = "\  (* " *)

    fun D typ s vaerdier =
          typ^"& "^s^" = "^typ^"::mk("^G^s^G
          ^String.concat(List.map (fn v => ", "^v) vaerdier)^");\n"

    fun vaerdiTilStreng (syntaks.I t) = Int.toString t
      | vaerdiTilStreng (syntaks.B b) = Bool.toString b
      | vaerdiTilStreng (syntaks.S s) = s

    (* Udskriv liste af vaerdier i henhold til fkt, der oversaetter
     * elementer i listen til strenge. *)
    fun udskrivL udstroem fkt liste =
          map (fn elm => TextIO.output(udstroem, fkt elm)) liste

    (***** Lav h-filer med erklaeringerne af komponenterne *****)

    fun erklaeringer udstroem (("main", (_,_,_,_,_,_))::program) =
          erklaeringer udstroem program
      | erklaeringer udstroem ((navn,(indstik,udstik,_,_,_,_))::program) =
          let val ind = String.concat(map (fn (s,_) => ", INABSTRACTION& "^s) indstik)
            val ud = String.concat(map (fn (s,_) => ", OUTABSTRACTION& "^s) udstik)
            val iu = String.extract(String.concat(map (fn (s,_) => ", "^s^"("^s^")")
                                                  (indstik@udstik)),2,NONE)
          in (TextIO.output(udstroem,"struct "^navn^" : public virtual AutoDelete {\n\n")
              ; udskrivL udstroem (fn (s,_) => "INABSTRACTION& "^s^";\n") indstik
              ; udskrivL udstroem (fn (s,_) => "OUTABSTRACTION& "^s^";\n") udstik
              ; TextIO.output(udstroem, navn^"("^String.extract(ind^ud,2,NONE)
                              ^")\n  : "^iu^" {};\n"
                              ^"static "^navn^"& mk(String home);\n};\n")
              ; (navn^".h")::(erklaeringer udstroem program))(*???slet alt filhalloejet*)
          end
      | erklaeringer udstroem [] = []

    (***** Lav cc-filer med definitionerne af komponenterne *****)

    (* Udskriv stik i cc-fil. *)
    fun padsSkriv _ [] [] = ()
      | padsSkriv udstroem indstik udstik =
          (TextIO.output(udstroem, "// PADs\n")
           ; udskrivL udstroem (fn (s,str) => D "INPAD" s [Int.toString str]) indstik
           ; udskrivL udstroem (fn (s,str) => D "OUTPAD" s [Int.toString str]) udstik
           ; TextIO.output(udstroem, "\n"))

    fun kompErkl udstroem planer ((s,((typ,_),vaerdier))::komp) =
```

```
      let val (nytyp,nyvaerdier) =
        ((let val (plan,istr,ustr) = liste.hent typ planer  (* s er plan *)
              val h = List.length plan
          in ("PLA",(map (Int.toString) [istr,ustr,h])@[typ]) end)
            handle liste.fejlListe_ikkeIListe =>
              ((mellemkode.hentEgenKomp typ vaerdier,[])
               handle mellemkode.fejlMellemkode_indbyggetKomp =>
                 (typ.hentNavn typ,map vaerdiTilStreng vaerdier)))
      in (TextIO.output(udstroem, D nytyp s nyvaerdier)
          ; kompErkl udstroem planer komp)
      end
  | kompErkl _ _ [] = ()

fun skelErkl udstroem [] = ()
  | skelErkl udstroem ((s,typ,komp,stik,stiknr,bitNr)::skel) =
      let val vaerdi = kompˆ(if stik =  then  else "."ˆstik)
        ˆ(if stiknr =  then  else "["ˆstiknrˆ"]")ˆ"["ˆInt.toString bitNrˆ"]"
      in (TextIO.output(udstroem, D typ s [vaerdi])
          ; skelErkl udstroem skel)
      end

(* Skriv erklaeringerne af planer og komponenter i en krop for
 * en komponent i filen specificeret af udstroem. *)
fun erklSkriv _ [] [] [] = ()
  | erklSkriv udstroem planer komp skel =
      let fun planS plan =
        String.extract(String.concat(map (fn (i,u) => ", "ˆGˆiˆ" "ˆuˆG) plan),
                       2, NONE)
      in (TextIO.output(udstroem,"// Declarations \n\n")
          ; udskrivL udstroem (fn (s,(plan,_,_)) => "char *"ˆsˆ"[] = { "
                                ˆ(planS plan)ˆ" };\n") planer
          ; kompErkl udstroem planer komp
          ; skelErkl udstroem skel
          ; TextIO.output(udstroem,"\n"))
      end

fun afledTilS (split.subset(st,lgd)) =
      ".subset("ˆ(Int.toString st)ˆ","ˆ(Int.toString lgd)ˆ")"
  | afledTilS (split.reverse) = ".reverse()"

(* Udskriv forbindelserne i kroppen. *)
fun elmS (split.aelm) = "0"
  | elmS (split.belm(vaerdi,skel)) =
      "Const("ˆGˆ(konst.tilHex vaerdi)ˆGˆ","
      ˆ(Int.toString (konst.stoerrelse vaerdi))ˆ")"
      ˆ(String.concat(List.map (fn s => " >> "ˆs) skel))
  | elmS (split.kelm(navn,stik,stiknr,afled,skel)) =
      navnˆ(if stik =  then  else "."ˆstik)
      ˆ(if stiknr =  then  else "["ˆstiknrˆ"]")
      ˆ(String.concat(map afledTilS afled))
      ˆ(String.concat(List.map (fn s => " >> "ˆs) skel))

fun stikStreng ((s,_)::indstik) udstik =
      (","ˆsˆ".outside()")ˆ(stikStreng indstik udstik)
  | stikStreng [] ((s,_)::udstik) =
      (","ˆsˆ".outside()")ˆ(stikStreng [] udstik)
  | stikStreng [] [] =

fun returnerObjekt _ "main" _ _ = ()
  | returnerObjekt udstroem navn indstik udstik =
      let val stik' = stikStreng indstik udstik
        val stik = if stik' =  then  else String.extract(stik',1,NONE)
      in TextIO.output(udstroem,"  return *new "ˆnavnˆ"("ˆstikˆ");\n") end

(* Udskriv definitionen for en programblok. *)
fun definition udstroem navn hoved indstik udstik planer komp skel krop =
```

```
        ( TextIO.output(udstroem, "\n" ^ hoved)
         ; padsSkriv udstroem indstik udstik
         ; erklSkriv udstroem planer komp skel
         ; TextIO.output(udstroem,"// Connections\n\n")
         ; udskrivL udstroem (fn (u,i) => (elmS u)^" >> "^(elmS i)^";\n") krop
         ; returnerObjekt udstroem navn indstik udstik
         ; TextIO.output(udstroem, "};\n")
         ; (navn^".cc"))

  fun definitioner udstroem (("main",(_,_,planer,komp,skel,krop))::program) =
        let val hoved = "class MySimApp : public SimApp {\npublic:\n  virtual void BuildModel();\n  MySimApp(
        in (definition udstroem "main" hoved [] [] planer komp skel krop)
          ::(definitioner udstroem program)
        end
    | definitioner udstroem ((navn,(indstik,udstik,planer,komp,skel,krop))::program) =
        let val hoved = navn^"& "^navn^"::mk(String home) {\n\n"
        in (definition udstroem navn hoved indstik udstik planer komp skel krop)
          ::(definitioner udstroem program)
        end
    | definitioner _ [] = []

  (***** Oversaet fra mellemkode til filer med SimSys/C++-kode. *****)

  fun oversaet () =
        let val mlkode = mellemkode.hent ()
          val udstroem = TextIO.openOut "main.cc"
          val () = TextIO.output(udstroem, "#include "^G^"simlib.h"^G^"\n")
          val () = TextIO.output(udstroem, "#include "^G^"interface.h"^G^"\n")
          val () = TextIO.output(udstroem, "#include "^G^"composite.h"^G^"\n\n\n")
          val hfiler = erklaeringer udstroem mlkode
          val ccfiler = definitioner udstroem mlkode
        in (TextIO.closeOut udstroem ; (hfiler,ccfiler)) end

end
```

### udtryk.sig

```
signature udtryk =
  sig

    exception fejlUdtryk_ikkePar
    exception fejlUdtryk_ulovligtUdtryk

    (* Rejser fejlUdtryk_ulovligtUdtryk, hvis inddata er BLANK. *)
    val pos : syntaks.udtryk -> syntaks.pos

    (* Returnerer venstre og hoejre del af et sammensat udtryk. Er
     * udtrykket ikke sammensat, rejses fejlUdtryk_ikkePar. *)
    val par : syntaks.udtryk -> syntaks.udtryk * syntaks.udtryk

    (* Returnerer vaerdien af et udtryk. Returnerer 0, hvis BLANK. *)
    val udregn : (string * syntaks.intboolstreng) list -> syntaks.udtryk -> int

    (* Returnerer vaerdien af et boolsk udtryk. *)
    val udregnBool : (string * syntaks.intboolstreng) list  (* (var, *)
                  -> syntaks.udtrykBool -> bool       (*  udtryk) -> bool *)

    (* Udregner udtryk, der enten er en streng med en heltals-,
     * boolsk eller strengvaerdi eller et udtryk. *)
    val udregnTalBoolS : (string * syntaks.intboolstreng) list -> syntaks.udtrykmBool
                      -> syntaks.intboolstreng

  end
```

### udtryk.sml

```
structure udtryk :> udtryk =
  struct

    local open syntaks fejl
    in

      exception fejlUdtryk_ikkePar
      exception fejlUdtryk_ulovligtUdtryk
      exception fejlUdtryk  (* fejl, der ikke kan ske *)

      (* Returnerer underudtrykkene i et sammensat udtryk. *)
      fun par (PLUS vh) = vh
        | par (MINUS vh) = vh
        | par (GANGE vh) = vh
        | par (DIV vh) = vh
        | par (MOD vh) = vh
        | par (HAT vh) = vh
        | par _ = raise fejlUdtryk_ikkePar

      (* Returnerer underudtrykkene i et sammensat boolsk udtryk. *)
      fun parBool (LT vh) = vh
        | parBool (LEQ vh) = vh
        | parBool (LIG vh) = vh
        | parBool (GEQ vh) = vh
        | parBool (GT vh) = vh
        | parBool _ = raise fejlUdtryk_ikkePar

      (* Returnerer operatoren for et udtryk. *)
      fun ope (PLUS _) = op+
        | ope (MINUS _) = op-
        | ope (GANGE _) = op*
        | ope (DIV _) = op div
        | ope (MOD _) = op mod
        | ope (HAT _) = op fkt.oploeft
        | ope _ = raise fejlUdtryk_ikkePar

      (* Returnerer operatoren for et boolsk udtryk. *)
      fun opeBool (LT _) = op<
        | opeBool (LEQ _) = op <=
        | opeBool (LIG _) = op=
        | opeBool (GEQ _) = op >=
        | opeBool (GT _) = op>
        | opeBool _ = raise fejlUdtryk_ikkePar

      (* Returnerer (start)positionen for et udtryk. *)
      fun pos (STR(_,p)) = p
        | pos (TAL(_,p)) = p
        | pos (HEX(_,p)) = p
        | pos BLANK = raise fejlUdtryk_ulovligtUdtryk
        | pos u = pos (#1 (par u))

      fun udregn _ (TAL(t,_)) = t
        | udregn _ (HEX(s,_)) = fkt.hexTilTal s
        | udregn var (STR(s,pos)) =
            ((let val vaerdi = liste.hent s var in
                case vaerdi of
                  I(t) => t
                | B _  => raise fejl(varBoolBrugesSomTal (s,pos))
                | S _  => raise fejl(varStrengBrugesSomTal (s,pos))
              end) handle liste.fejlListe_ikkeIListe =>
                raise fejl(varIkkeDef (s,pos)))
        | udregn _ BLANK = 0
        | udregn var u =
            let val (v,h) = par u
            in (ope u)(udregn var v, udregn var h) end
```

```
      fun udregnBool _ (TRUE _) = true
        | udregnBool _ (FALSE _) = false
        | udregnBool var (STRB(s,pos)) =
            ((let val vaerdi = liste.hent s var in
                case vaerdi of
                  B(boo) => boo
                | I _    => raise fejl(varTalBrugesSomBool (s,pos))
                | S _    => raise fejl(varStrengBrugesSomBool (s,pos))
              end) handle liste.fejlListe_ikkeIListe =>
                raise fejl(varIkkeDef (s,pos)))
        | udregnBool var ub =
            let val (v,h) = parBool ub
            in (opeBool ub)(udregn var v,udregn var h) end

      (* Bruges til at udregne udtryk fra inddatavariablene for komponenter.
       * Alle strenge laeses som udt(s). Boolske udtryk er ikke tilladt. *)
      fun udregnTalBoolS var (udtb(ub)) =
            (case ub of
               TRUE _  => B true
             | FALSE _ => B false
             | _       => raise fejlUdtryk)  (* kan ikke ske *)
        | udregnTalBoolS var (udt(STR(s,pos))) =
            ((liste.hent s var)  (* kan vaere bool, tal eller streng *)
             handle liste.fejlListe_ikkeIListe => raise fejl(varIkkeDef (s,pos)))
        | udregnTalBoolS var (udt(u)) = I(udregn var u)

    end
  end
```

### plan.sig

```
signature plan =
  sig

    type typ = (string * string) list * int * int  (* plan,indstoer,udstoer *)

    (* Planer bliver lavet om fra [[udN,indN],...,[ud0,ind0]] til [(ind0,ud0),...].
     * Samtidig kontrolleres, at de har samme stoerrelse inddata hhv.
     * uddata, at der ikke optraeder "0" i uddatastrenge, og at der ikke er angivet
     * for mange strenge i en linie i en plan.
     * Andet argument bruges kun ved fejlrejsning (i hvilken plan opstod fejlen). *)

    val lav : syntaks.streng list list -> syntaks.streng -> typ

  end
```

### plan.sml

```
structure plan :> plan =
  struct

    local open fejl
    in

      type typ = (string * string) list * int * int  (* plan,indstoer,udstoer *)

      fun ind lgd (s,pos) =
            if String.size s <> lgd then raise fejl(planInddataForskelligLgd pos)
            else s

      fun ud lgd (s,pos) =
            if String.size s <> lgd then raise fejl(planUddataForskelligLgd pos)
            else if fkt.delstreng "0" s then raise fejl(planUddataMed0 pos)
                 else s
```

```
      fun lav (plan as [(u,_),(i,_)]::_)) navn =
            let val indlgd = String.size i
              val udlgd = String.size u
            in
              ((List.foldl (fn (elm, liste) =>
                            case elm of
                              [u,i] => (ind indlgd i, ud udlgd u)::liste
                            | _     => raise fejl(planLinieSkalHaveToStrenge navn))
                 [] plan), indlgd, udlgd)
            end
        | lav [] navn = raise fejl(planErTom navn)
        | lav _ navn = raise fejl(planLinieSkalHaveToStrenge navn)

    end
  end
```

### bitvektor.sig

```
signature bitvektor =
  sig

    exception fejlBitvektor_ikkeSammensat

    (* Returnerer startposition for en bitvektor. *)
    val pos : syntaks.bitvektor -> syntaks.pos

    (* Returnerer hver del af en sammensat bitvektor. Rejser
     * fejlBitvektor_ikkeSammensat, hvis bitvektoren ikke er sammensat. *)
    val par : syntaks.bitvektor -> syntaks.bitvektor * syntaks.bitvektor

    (* Tager konstruktoeren i foerste  bitvektor og samler (v,h) med denne
     * konstruktoer. Rejser fejlBitvektor_ikkeSammensat, hvis foerste
     * bitvektor ikke er sammensat. *)
    val saml : syntaks.bitvektor -> syntaks.bitvektor * syntaks.bitvektor
            -> syntaks.bitvektor

  end
```

### bitvektor.sml

```
structure bitvektor :> bitvektor =
  struct

    local open syntaks
    in

      exception fejlBitvektor_ikkeSammensat

      fun par (FORBIND vh) = vh
        | par (SPLIT vh) = vh
        | par (SEKV vh) = vh
        | par _ = raise fejlBitvektor_ikkeSammensat

      fun pos (GENNEMLOEB(_,p)) = p
        | pos (AFLOEB(_,p)) = p
        | pos (KONST(_,p,_)) = p
        | pos (KOMP((_,p),_,_,_)) = p
        | pos (UD p) = p
        | pos (BITV(bv,_,_)) = pos bv
        | pos bv = pos (#1 (par bv))

      fun saml (FORBIND _) vh = FORBIND vh
        | saml (SPLIT _) vh = SPLIT vh
        | saml (SEKV _) vh = SEKV vh
```

```
        | saml _ _ = raise fejlBitvektor_ikkeSammensat

    end
  end
```

### liste.sig

```
signature liste =
  sig

    exception fejlListe_ikkeIListe
    exception fejlListe_alleredeIListe

    (* Indsaetter nyt element i liste. Rejser fejlListe_alleredeIListe,
     * hvis elementet allerede findes i listen. *)
    val indsaet : string -> 'a -> (string * 'a) list -> (string * 'a) list
    val indsaetPos : syntaks.pos -> 'a -> (syntaks.pos * 'a) list
                  -> (syntaks.pos * 'a) list

    (* Opdaterer element i liste. Rejser fejlListe_ikkeIListe, hvis elementet
     * ikke findes i listen. *)
    val opdater : string -> 'a -> (string * 'a) list -> (string * 'a) list
    val opdaterPos : syntaks.pos -> 'a -> (syntaks.pos * 'a) list
                  -> (syntaks.pos * 'a) list

    (* Returnerer true, hvis elementet findes i listen. *)
    val findes : string -> (string * 'a) list -> bool

    (* Henter data for element i listen. Rejser fejlListe_ikkeIListe, hvis
     * elementet ikke findes i listen. *)
    val hent : string -> (string * 'a) list -> 'a
    val hentPos : syntaks.pos -> (syntaks.pos * 'a) list -> 'a

  end
```

### liste.sml

```
structure liste :> liste =
  struct

    exception fejlListe_ikkeIListe
    exception fejlListe_alleredeIListe

    (* Globale funktioner *)

    fun indsaet s data [] = [(s,data)]
      | indsaet (s:string) data ((elm as (s',_))::liste) =
          if s < s' then (s,data)::elm::liste
          else if s = s' then raise fejlListe_alleredeIListe
               else elm::(indsaet s data liste)

    fun indsaetPos pos data [] = [(pos,data)]
      | indsaetPos (pos as (a,b)) data ((elm as (pos' as (c,d),_))::liste) =
          if (a < c orelse (a = c andalso b < d)) then (pos,data)::elm::liste
          else if pos = pos' then raise fejlListe_alleredeIListe
               else elm::(indsaetPos pos data liste)

    fun opdater _ _ [] = raise fejlListe_ikkeIListe
      | opdater s data ((elm as (s',_))::rest) =
          if (s = s') then (s,data)::rest
          else elm::(opdater s data rest)

    fun opdaterPos _ _ [] = raise fejlListe_ikkeIListe
      | opdaterPos pos data ((elm as (pos',_))::rest) =
          if (pos = pos') then (pos,data)::rest
```

```
          else elm::(opdaterPos pos data rest)

    fun findes s ((elm as (s':string,_))::liste) =
          (s = s') orelse (findes s liste)
      | findes s [] = false

    fun hent (_:string) [] = raise fejlListe_ikkeIListe
      | hent s ((s',data)::rest) =
          if s = s' then data else (hent s rest)

    fun hentPos pos [] = raise fejlListe_ikkeIListe
      | hentPos pos ((pos',data)::rest) =
          if pos = pos' then data else (hentPos pos rest)

  end
```

### fejl.sig

```
signature fejl =
  sig

    exception syntaks_strEjErklaeret of syntaks.pos
    exception syntaks_gennemloebUlovligt
    exception syntaks_internForbindelse of syntaks.pos

    datatype fejltype =
      intetHovedprogram
    | toHovedprogrammer
    | mainVaerdiIkkeTalBool
    | hovedprogramForkertAntalVar of int  (* int = korrekt # var *)
    | kompUdenStik of syntaks.streng
    | typAlleredeDef of syntaks.streng
    | typIkkeDef of syntaks.streng
    | navnMatcherIkkeTyp of syntaks.streng
    | navnMatcherIkkeSkel of syntaks.streng
    | skelFindesIkke of syntaks.streng
    | kompFindesIkke of syntaks.streng
    | skelUlovligBrug of syntaks.streng
    | skelMedAfled of syntaks.pos
    | varIkkeDef of syntaks.streng
    | bitvarDefSomKomp of syntaks.streng
    | bitvarDobbeltDef of syntaks.streng
    | kompDobbeltDef of syntaks.streng
    | planDobbeltDef of syntaks.streng
    | stikDefSomKomp of syntaks.streng
    | kompDefSomPlan of syntaks.streng
    | varTildelesIkkeTal of syntaks.streng
    | varTildelesIkkeBool of syntaks.streng
    | varTildelesIkkeStreng of syntaks.streng
    | varTalBrugesSomBool of syntaks.streng
    | varStrengBrugesSomBool of syntaks.streng
    | varBoolBrugesSomTal of syntaks.streng
    | varStrengBrugesSomTal of syntaks.streng
    | stoerrelseUdenStik of syntaks.pos
    | stikIngenStoerrelse of syntaks.streng
    | konstLaengereEndStoerrelse of syntaks.pos
    | rekursionIBitvarDef of syntaks.streng
    | planInddataForskelligLgd of syntaks.pos
    | planUddataForskelligLgd of syntaks.pos
    | planUddataMed0 of syntaks.pos
    | planLinieSkalHaveToStrenge of syntaks.streng
    | planErTom of syntaks.streng
    | flereVaerdierEndVar of syntaks.streng
    | flereVarEndVaerdier of syntaks.streng
    | bitvektorUdErTom of syntaks.pos
    | bitvektorIndErTom of syntaks.pos
```

```
    | bitvektorISplitTom of syntaks.pos
    | komponentRekursionUdenFald of string
    | afledBrugIkkeTomMedAndre of syntaks.pos
    | afledPaaTomBitvektor of syntaks.pos
    | afledIndMedKopier of syntaks.pos
    | afledUdenforBitvektor of syntaks.pos
    | feltinddelingUlovlig of syntaks.pos
    | udIndForskIForbind of syntaks.pos
    | indForskISplit of syntaks.pos
    | vektorUdenElm of syntaks.streng
    | vektorUdenAntal of syntaks.streng
    | stikStoerrelseLE0 of syntaks.pos
    | indbitOverlapper of syntaks.pos
    | indbitIkkeForbundet of string * syntaks.pos * int
    | udbitIkkeForbundet of string * syntaks.pos * int

    exception fejl of fejltype

    val oversaet : fejltype -> string

  end
```

### fejl.sml

```
structure fejl :> fejl =
  struct

    local open syntaks
    in

      exception syntaks_strEjErklaeret of pos
      exception syntaks_gennemloebUlovligt
      exception syntaks_internForbindelse of pos

      datatype fejltype =
        intetHovedprogram
      | toHovedprogrammer
      | mainVaerdiIkkeTalBool
      | hovedprogramForkertAntalVar of int  (* int = korrekt # var *)
      | kompUdenStik of streng
      | typAlleredeDef of streng
      | typIkkeDef of streng
      | navnMatcherIkkeTyp of streng
      | navnMatcherIkkeSkel of streng
      | skelFindesIkke of streng
      | kompFindesIkke of streng
      | skelUlovligBrug of streng
      | skelMedAfled of pos
      | varIkkeDef of streng
      | bitvarDefSomKomp of streng
      | bitvarDobbeltDef of streng
      | kompDobbeltDef of streng
      | planDobbeltDef of streng
      | stikDefSomKomp of streng
      | kompDefSomPlan of streng
      | varTildelesIkkeTal of streng
      | varTildelesIkkeBool of streng
      | varTildelesIkkeStreng of streng
      | varTalBrugesSomBool of streng
      | varStrengBrugesSomBool of streng
      | varBoolBrugesSomTal of streng
      | varStrengBrugesSomTal of streng
      | stoerrelseUdenStik of pos
      | stikIngenStoerrelse of streng
      | konstLaengereEndStoerrelse of pos
      | rekursionIBitvarDef of streng
```

```
| planInddataForskelligLgd of pos
| planUddataForskelligLgd of pos
| planUddataMed0 of pos
| planLinieSkalHaveToStrenge of streng
| planErTom of streng
| flereVaerdierEndVar of streng
| flereVarEndVaerdier of streng
| bitvektorUdErTom of pos
| bitvektorIndErTom of pos
| bitvektorISplitTom of pos
| komponentRekursionUdenFald of string
| afledBrugIkkeTomMedAndre of pos
| afledPaaTomBitvektor of pos
| afledIndMedKopier of pos
| afledUdenforBitvektor of pos
| feltinddelingUlovlig of pos
| udIndForskIForbind of pos
| indForskISplit of pos
| vektorUdenElm of streng
| vektorUdenAntal of streng
| stikStoerrelseLE0 of pos
| indbitOverlapper of syntaks.pos
| indbitIkkeForbundet of string * pos * int
| udbitIkkeForbundet of string * pos * int

exception fejl of fejltype

fun fejlPos (ln,sjl) =
  "Error at line "^makestring ln^", col "^makestring sjl^":\n"

fun oversaet intetHovedprogram = "main is missing"
  | oversaet toHovedprogrammer = "main defined twice"
  | oversaet mainVaerdiIkkeTalBool =
      "input to main has to be true, false or pos. integer"
  | oversaet (hovedprogramForkertAntalVar antal) =
      "main should have "^makestring antal^" arguments"
  | oversaet (kompUdenStik(s,pos)) =
      fejlPos pos^s^" have no channels"
  | oversaet (typAlleredeDef (s,pos)) =
      fejlPos pos^"a building block type with name "^s^" already exists"
  | oversaet (typIkkeDef(s,pos)) =
      fejlPos pos^"no building block type with name "^s^" exists"
  | oversaet (navnMatcherIkkeTyp(s,pos)) =
      fejlPos pos^"the building block name "^s^" matches no boulding block type"
  | oversaet (navnMatcherIkkeSkel(s,pos)) =
      fejlPos pos^"the boundary "^s^" matches no boundary type"
  | oversaet (skelFindesIkke(s,pos)) =
      fejlPos pos^"no boundary "^s^" exists"
  | oversaet (kompFindesIkke(s,pos)) =
      fejlPos pos^"no building block "^s^" exists"
  | oversaet (skelUlovligBrug(s,pos)) =
      fejlPos pos^"invalid use of boundary "^s
  | oversaet (skelMedAfled pos) =
      fejlPos pos^"derivation on boundary is illegal"
  | oversaet (varIkkeDef(s,pos)) =
      fejlPos pos^"the variable "^s^" is not defined"
  | oversaet (bitvarDefSomKomp(s,pos)) =
      fejlPos pos^s^" is already defined (as a bit variable)"
  | oversaet (bitvarDobbeltDef(s,pos)) =
      fejlPos pos^s^" is already defined"
  | oversaet (kompDobbeltDef(s,pos)) =
      fejlPos pos^s^" is already defined"
  | oversaet (planDobbeltDef(s,pos)) =
      fejlPos pos^s^" is already defined"
  | oversaet (stikDefSomKomp(s,pos)) =
      fejlPos pos^s^" is already defined (as a channel)"
```

```
| oversaet (kompDefSomPlan(s,pos)) =
    fejlPos posˆsˆ" is already defined (as a plane)"
| oversaet (varTildelesIkkeTal(s,pos)) =
    fejlPos posˆ" integer variable is assigned non-integer value in "ˆs
| oversaet (varTildelesIkkeBool(s,pos)) =
    fejlPos posˆ" bool variable is assigned non-bool value"ˆs
| oversaet (varTildelesIkkeStreng(s,pos)) =
    fejlPos posˆ" string variable is assigned non-string value"ˆs
| oversaet (varTalBrugesSomBool(s,pos)) =
    fejlPos posˆsˆ" has to be a boolean"
| oversaet (varStrengBrugesSomBool(s,pos)) =
    fejlPos posˆsˆ" has to be a boolean"
| oversaet (varBoolBrugesSomTal(s,pos)) =
    fejlPos posˆsˆ" has to be an integer"
| oversaet (varStrengBrugesSomTal(s,pos)) =
    fejlPos posˆsˆ" has to be an integer"
| oversaet (stoerrelseUdenStik pos) =
    fejlPos posˆ"size assigned to no channel"
| oversaet (stikIngenStoerrelse(s,pos)) =
    fejlPos posˆ"channel "ˆsˆ" is assigned no size"
| oversaet (konstLaengereEndStoerrelse pos) =
    fejlPos posˆ"constant too big for the given size"
| oversaet (rekursionIBitvarDef(s,pos)) =
    fejlPos posˆ"recursion in the bit variable "ˆs
| oversaet (planInddataForskelligLgd pos) =
    fejlPos posˆ"size of and planes differ"
| oversaet (planUddataForskelligLgd pos) =
    fejlPos posˆ"size of or planes differ"
| oversaet (planUddataMed0 pos) =
    fejlPos posˆ"0 not allowed in output from a plane"
| oversaet (planLinieSkalHaveToStrenge(s,pos)) =
    fejlPos posˆ"the plane "ˆsˆ" contains line with more or less than an"
    ˆ" input and an output"
| oversaet (planErTom(s,pos)) =
    fejlPos posˆ"the plane "ˆsˆ" is empty"
| oversaet (flereVaerdierEndVar(s,pos)) =
    fejlPos posˆ"the building block "ˆsˆ" is assigned to many values"
| oversaet (flereVarEndVaerdier(s,pos)) =
    fejlPos posˆ"the building block "ˆsˆ" is assigned to few values"
| oversaet (bitvektorUdErTom pos) =
    fejlPos posˆ"bit array in connection has empty output"
| oversaet (bitvektorIndErTom pos) =
    fejlPos posˆ"bit array in connection has empty input"
| oversaet (bitvektorISplitTom pos) =
    fejlPos posˆ"bit array in , has empty input"
| oversaet (komponentRekursionUdenFald s) =
    "Recursion with building block "ˆsˆ" where the size of the building block"
    ˆ" does not decrease"
| oversaet (afledBrugIkkeTomMedAndre pos) =
    fejlPos posˆ"empty derivation used amidst other derivations"
| oversaet (afledPaaTomBitvektor pos) =
    fejlPos posˆ"non-empty derivation used on empty bit array"
| oversaet (afledIndMedKopier pos) =
    fejlPos posˆ"* (copy) not allowed in in-derivations"
| oversaet (afledUdenforBitvektor pos) =
    fejlPos posˆ"derivation out of bounds (of bit array)"
| oversaet (feltinddelingUlovlig pos) =
    fejlPos posˆ"the size of the input does not match the size of the output"
    ˆ" in the connection"
| oversaet (udIndForskIForbind pos) =
    fejlPos posˆ"the size of the input does not match the size of the output"
    ˆ" in the connection"
| oversaet (indForskISplit pos) =
    fejlPos posˆ"the size of the input does not match the size of the other"
    ˆ" input in split (,)"
| oversaet (vektorUdenElm(s,pos)) =
```

```
            fejlPos posˆ"the array "ˆsˆ" should contain at least one element"
        | oversaet (vektorUdenAntal(s,pos)) =
            fejlPos posˆ"the array "ˆsˆ" has no specification of number of elements"
        | oversaet (stikStoerrelseLE0 pos) =
            fejlPos posˆ"channel size has to be positive"
        | oversaet (indbitOverlapper pos) =
            fejlPos posˆ"in-bit already connected"
        | oversaet (indbitIkkeForbundet(s,pos,bit)) =
            fejlPos posˆ"in-bit "ˆmakestring bitˆ" of "ˆsˆ" not connected"
        | oversaet (udbitIkkeForbundet(s,pos,bit)) =
            fejlPos posˆ"out-bit "ˆmakestring bitˆ" of "ˆsˆ" not connected"

    end
  end
```

### syntaks.sml

Denne fil har ikke en tilhørende `.sig`-fil.

```
structure syntaks =
  struct

exception testFejl (*???slet*)
val testref = ref false
fun test s = TextIO.output(TextIO.stdOut, sˆ"\n")
fun testif s = if (!testref) then TextIO.output(TextIO.stdOut, sˆ"\n") else ()

    (* syntaks for det abstrakte syntakstræ *)

    type pos = int * int       (* position in program (line, line pos) *)
    type streng = string * pos

    datatype udtryk =
      BLANK
    | STR of streng  (* indeholder pos *)
    | TAL of int * pos
    | HEX of string * pos
    | PLUS of udtryk * udtryk
    | MINUS of udtryk * udtryk
    | GANGE of udtryk * udtryk
    | DIV of udtryk * udtryk
    | MOD of udtryk * udtryk
    | HAT of udtryk * udtryk

    datatype udtrykBool =
      STRB of streng
    | TRUE of pos
    | FALSE of pos
    | LT of udtryk * udtryk
    | LEQ of udtryk * udtryk
    | LIG of udtryk * udtryk
    | GEQ of udtryk * udtryk
    | GT of udtryk * udtryk

    datatype udtrykmBool =
      udt of udtryk  (* indeholder alle strenge *)
    | udtb of udtrykBool

    datatype bitmoenster =
      BITM of (udtryk * udtryk) list  (* [(fra,til)] *)
    | FELTER of udtryk * (udtryk * udtryk) list list  (* (antal,[[(fra,til)]]) *)
    | KOPIER of udtryk

    datatype bitafled =
      TOM of pos
    | BITAF of bitmoenster list
```

```
    datatype bitvektor =
      FORBIND of bitvektor * bitvektor
    | SPLIT of bitvektor * bitvektor
    | SEKV of bitvektor * bitvektor
    | BITV of bitvektor * bitafled list * bitafled list

    | GENNEMLOEB of udtryk * pos          (* har brug for pos for gennemløb, *)
    | AFLOEB of udtryk * pos              (* afloeb og konstanter, så jeg kan *)
    | KONST of konst.typ * pos * udtryk   (* kende forskel på de forskellige *)
    | KOMP of streng * typen * bitafled list * bitafled list
    | SKEL of streng  (* indsaettes i afled.sml, saa findes ikke i parser *)
    | UD of pos  (* bruges kun i forbindelser i vektorer *)
  and
    typen =
      NIL
    | TYPE of udtrykmBool list * (bitvektor * bitvektor) list
    | SKELTYPE of streng * streng * udtryk  (* (type,fraKomp,bitNr) *)

    datatype saetning =
      VAR of streng * bitvektor
    | PLAN of streng * streng list list
    | IF of udtrykBool * saetning list * saetning list
    | SAETNING of bitvektor

    datatype programblok =
      MAIN of streng list * saetning list
    | KOMPDEF of streng * udtryk list * streng list * udtryk list * saetning list

  (* Diverse syntaks *)

    datatype intboolstreng = I of int | B of bool | S of string
    datatype vaerditype = taltype | booltype | ukendt | stoertype | strengtype

  end
```

## konst.sig

```
signature konst =
  sig

    exception fejlKonst_konstForKort

    (* Konstantens binaere vaerdi i liste af 0 og 1. *)
    type typ = int list

    val talTilKonst : int -> typ

    val hexTilKonst : string -> typ

    val stoerrelse : typ -> int

    val tilHex : typ -> string

    (* Udvider konstant til en bestemt stoerrelse. Det forventes, at
     * stoerrelsen er >= stoerrelsen af konstanten.
     * Rejser fejlKonst_konstForKort, hvis stoerrelsen af konstanten er
     * stoerre end den nye stoerrelse. *)
    val tilStoerrelse : typ -> int -> typ

    (* Spejler konstanten. Det antages, at denne har den korrekte stoerrelse. *)
    val spejles : typ -> typ

    (* Tager en del af konstanten ud. Det antages, at denne har den korrekte
     * stoerrelse, og at t<=t' angiver bit i konstanten (foerste bit er bit 0). *)
    val delKonst : typ -> int -> int -> typ
```

```
  end
```

### konst.sml

```
structure konst :> konst =
  struct

    exception fejlKonst_konstForKort
    exception fejlKonst

    (* Liste af 0 og 1 (bit). *)
    type typ = int list

    fun pad 0 konst = konst
      | pad t konst = pad (t-1) (0::konst)

    (* 0 bliver til []. *)
    fun talTilKonst int =
          let fun talTilKonst' 0 bin = bin
                | talTilKonst' int bin = talTilKonst' (int div 2) ((int mod 2)::bin)
          in if int=0 then [0] else (talTilKonst' int []) end

    fun hexTilKonst hex = talTilKonst (fkt.hexTilTal hex)

    fun stoerrelse konst = List.length konst

    fun tilHexS i =
          if i < 10 then Int.toString i
          else Char.toString(Char.chr(Char.ord #"A" + (i-10)))

    fun tilHex konst =
          let val padStr = (fn 1 => 3 | 3 => 1 | i => i) ((List.length konst) mod 4)
            fun tilHex' (b0::b1::b2::b3::rest) =
                    tilHexS(b0*8 + b1*4 + b2*2 + b3) ^ (tilHex' rest)
                | tilHex' [] =
                | tilHex' _ = raise fejlKonst
          in tilHex'(pad padStr konst) end

    fun tilStoerrelse konst str =
          let val str' = List.length konst in
            if str' > str then raise fejlKonst_konstForKort
            else pad (str-str') konst  (* foranstil 0'er *)
          end

    fun spejles konst = List.rev konst

    fun delKonst konst t t' = List.take(List.drop(konst,t),t'-t+1)

  end
```

### fkt.sig

```
signature fkt =
  sig

    val oploeft : int * int -> int

    (* Er foerste streng en delstreng af anden streng? *)
    val delstreng : string -> string -> bool

    (* Det antages, at string er med smaa bogstaver. *)
    val hexTilTal : string -> int

    (* Finder den streng i sliste, hvor string er foerste og stoerste
```

```
     * (i naevnte raekkefoelge) delstreng. Returnerer , hvis en
     * saadan streng ikke findes i listen. *)
    val dagDelstreng : string list -> string -> string

    val lessPos : (int * int) -> (int * int) -> bool

  end
```

### fkt.sml

```
structure fkt :> fkt =
  struct

    infixr 8 oploeft
    fun x oploeft 0 = 1
      | x oploeft y = x * x oploeft (y - 1)

    fun delstreng sub str = Regex.regexecBool (Regex.regcomp sub []) [] str

    fun hexTilCiffer c =
          if Char.isDigit c then Char.ord c - Char.ord #"0"
          else Char.ord c - Char.ord #"a" + 10

    fun hexTilTal s =
          List.foldl (fn (y,x) => x * 16 + y)
                     0 (List.map hexTilCiffer (String.explode s))

    fun lessPos (a,b) (c,d) = (a < c) orelse ((a = c) andalso (b < d))

local
    open Regex
    open Listsort

    fun concat [x] = "(" ^ x ^ ")"
      | concat (x::xs) = "(" ^ x ^ ")|" ^ concat xs
      | concat [] =

    fun compare (x,y) = String.compare(y,x)
in
    fun dagDelstreng ss =
        let
            val rx = regcomp (concat (sort compare ss)) [Extended, Icase]
        in
            fn str =>
                case regexec rx [] str of
                     NONE =>
                   | SOME v => Substring.string (Vector.sub (v, 0))
        end
end

(* The dagDelstreng function accepts a list of keywords, and returns a function
   match : string -> string. If the argument string for match contains
   one of the keywords, this keyword is returned; otherwise, the empty string
   is returned. The match is case-insensitive; it not the keyword, but the
   match that is returned, so
        (dagDelstreng ["foo"]) "FOO"
   will return "FOO", not "foo".

   The longest possible match is found, so
        (dagDelstreng ["foo", "foobar"]) "foobar"
   yields "foobar", not "foo".

   Restrictions:

    - The keywords cannot contain characters with special meaning to
      the regex module (primarily "().*" etc; alphanumeric (and Danish)
```

```
      characters work fine).

    - Contrary to what one might expect, matching the empty list of
      keywords fails regardless. Hence,
        (dagDelstreng []) "..."
      returns .

val match = dagDelstreng ["add", "add2", "mux", "flipflop"];
val rv = match "minAdd2";

*)


  end
```

# Bilag F

# Ind- og uddata fra afprøvningen

I dette bilag gives de programmer, der er brugt ved afprøvningen, og uddata ved oversættelse af disse. Uddata lægges i fortsættelse af de inddataprogrammer, de kommer af.

Først gives i afsnit F.1 alle de programmer, der bør give et korrekt og fungerende program. Uddata fra disse programmer består af indholdet af den fil, som konstrueres af Hadesoversætteren. Alle disse programmer er gennemset for fejl og er desuden også blevet oversat med C++-oversætteren (med SimSys-klassebiblioteket inkluderet).

Dernæst gives i afsnit F.2 de programmer, der bør rejse fejl samt den fejl, de enkelte programmer rejser.

## F.1 Fungerende programmer

### F.1.1

Nedenstående program kaldes med `2 false`.

```
main (bit addition)
  ff'bit 1'*2 >> plusMinus'bit addition' >> ff;
  flipflop'2*bit-1 2'*2 >> plusMinus2'2*bit-1 true' >> flipflop
end

plusMinus(inA inB bit, bit addition) >> (out bit)
  /* hvis addition saa lav addition ellers subtraktion */
  if addition then inB 0'1' >> [bit..]add            // inB,cIn
  else inB >> notarray'bit' 1'1' >> [bit..]add end;
  inA >> [inA]add'bit' >> out _'1'
end
```

hvorved følgende uddata opnås:

```
#include "simlib.h"
#include "interface.h"
#include "composite.h"


struct komp0 : public virtual AutoDelete {

INABSTRACTION& ina;
INABSTRACTION& inb;
OUTABSTRACTION& out;
komp0(INABSTRACTION& ina, INABSTRACTION& inb, OUTABSTRACTION& out)
  : ina(ina), inb(inb), out(out) {};
static komp0& mk(String home);
};
```

```
struct komp1 : public virtual AutoDelete {

INABSTRACTION& ina;
INABSTRACTION& inb;
OUTABSTRACTION& out;
komp1(INABSTRACTION& ina, INABSTRACTION& inb, OUTABSTRACTION& out)
  : ina(ina), inb(inb), out(out) {};
static komp1& mk(String home);
};

komp0& komp0::mk(String home) {

// PADs
INPAD& ina = INPAD::mk("ina", 2);
INPAD& inb = INPAD::mk("inb", 2);
OUTPAD& out = OUTPAD::mk("out", 2);

// Declarations

Adder& add = Adder::mk("add", 2);
NotGate& notarray0 = NotGate::mk("notarray0");
NotGate& notarray1 = NotGate::mk("notarray1");

// Connections

notarray0.out >> add.inB.subset(0,1);
notarray1.out >> add.inB.subset(1,1);
Const("1",1) >> add.cIn;
inb.subset(0,1) >> notarray0.in;
inb.subset(1,1) >> notarray1.in;
add.sum >> out.subset(0,2);
add.cOut >> 0;
ina.subset(0,2) >> add.inA;
  return *new komp0(ina.outside(),inb.outside(),out.outside());
};

komp1& komp1::mk(String home) {

// PADs
INPAD& ina = INPAD::mk("ina", 3);
INPAD& inb = INPAD::mk("inb", 3);
OUTPAD& out = OUTPAD::mk("out", 3);

// Declarations

Adder& add = Adder::mk("add", 3);

// Connections

inb.subset(0,3) >> add.inB;
Const("0",1) >> add.cIn;
add.sum >> out.subset(0,3);
add.cOut >> 0;
ina.subset(0,3) >> add.inA;
  return *new komp1(ina.outside(),inb.outside(),out.outside());
};

class MySimApp : public SimApp {
public:
  virtual void BuildModel();
  MySimApp() {}
};

SimApp* SimApp::mk(int argc, char *argv[], char *envp[])
{ return new MySimApp; };
```

```
void MySimApp::BuildModel() {

// Declarations

FlipFlop& ff = FlipFlop::mk("ff", 2, 1);
FlipFlop& flipflop = FlipFlop::mk("flipflop", 3, 2);
komp0& plusminus = komp0::mk("plusminus");
komp1& plusminus2 = komp1::mk("plusminus2");

// Connections

plusminus.out >> ff.in;
ff.out >> plusminus.ina;
ff.out >> plusminus.inb;
plusminus2.out >> flipflop.in;
flipflop.out >> plusminus2.ina;
flipflop.out >> plusminus2.inb;
};
```

## F.1.2

Nedenstående program

```
indkomp (in 3) >> ()
  in >> _’2’ obsolete’ff 1 0’ >> _’1’
end

main ()
  udkomp’3’ >> indkomp
end

udkomp (, bit) >> (out bit)
  (0x1’1’)*(bit/2) (0’1’)*(bit/2 + bit mod 2) >> out
end
```

giver følgende uddata

```
#include "simlib.h"
#include "interface.h"
#include "composite.h"


struct komp0 : public virtual AutoDelete {

INABSTRACTION& in;
komp0(INABSTRACTION& in)
  : in(in) {};
static komp0& mk(String home);
};
struct komp1 : public virtual AutoDelete {

OUTABSTRACTION& out;
komp1(OUTABSTRACTION& out)
  : out(out) {};
static komp1& mk(String home);
};

komp0& komp0::mk(String home) {

// PADs
INPAD& in = INPAD::mk("in", 3);

// Declarations

FlipFlop& obsolete = FlipFlop::mk("obsolete", 1, 0);
```

```
// Connections

obsolete.out >> 0;
in.subset(0,2) >> 0;
in.subset(2,1) >> obsolete.in;
  return *new komp0(in.outside());
};

komp1& komp1::mk(String home) {

// PADs
OUTPAD& out = OUTPAD::mk("out", 3);

// Connections

Const("1",1) >> out.subset(0,1);
Const("0",1) >> out.subset(1,1);
Const("0",1) >> out.subset(2,1);
  return *new komp1(out.outside());
};

class MySimApp : public SimApp {
public:
  virtual void BuildModel();
  MySimApp() {}
};

SimApp* SimApp::mk(int argc, char *argv[], char *envp[])
{ return new MySimApp; };

void MySimApp::BuildModel() {

// Declarations

komp0& indkomp = komp0::mk("indkomp");
komp1& udkomp = komp1::mk("udkomp");

// Connections

udkomp.out >> indkomp.in;
};
```

### F.1.3

Nedenstående program kaldes med 4.

```
main (bit)
  ff'2 2';
  if bit <= 2 then ff >> ff
  else if bit mod 2 = 0 then
    ff*(bit/2)/(bit/2) >> trae'bit'[1..0] >> ff
  else
    ff >> trae'bit' >> ff
  end end
end

trae (in bit, bit) >> (out 2)
  if bit < 2 then in*2 >> out
  else if bit > 2 then
    in >> 'bit-2' nor'2' >> trae'bit-1' >> out
  else in >> out
  end end
end
```

hvorved følgende uddata opnås:

```
#include "simlib.h"
#include "interface.h"
#include "composite.h"


struct komp0 : public virtual AutoDelete {

INABSTRACTION& in;
OUTABSTRACTION& out;
komp0(INABSTRACTION& in, OUTABSTRACTION& out)
  : in(in), out(out) {};
static komp0& mk(String home);
};
struct komp1 : public virtual AutoDelete {

INABSTRACTION& in;
OUTABSTRACTION& out;
komp1(INABSTRACTION& in, OUTABSTRACTION& out)
  : in(in), out(out) {};
static komp1& mk(String home);
};
struct komp2 : public virtual AutoDelete {

INABSTRACTION& in;
OUTABSTRACTION& out;
komp2(INABSTRACTION& in, OUTABSTRACTION& out)
  : in(in), out(out) {};
static komp2& mk(String home);
};

komp0& komp0::mk(String home) {

// PADs
INPAD& in = INPAD::mk("in", 4);
OUTPAD& out = OUTPAD::mk("out", 2);

// Declarations

NorGate& nor = NorGate::mk("nor", 2);
komp1& trae = komp1::mk("trae");

// Connections

trae.out >> out.subset(0,2);
nor.out >> trae.in.subset(2,1);
in.subset(2,2) >> nor.in;
in.subset(0,2) >> trae.in.subset(0,2);
  return *new komp0(in.outside(),out.outside());
};

komp1& komp1::mk(String home) {

// PADs
INPAD& in = INPAD::mk("in", 3);
OUTPAD& out = OUTPAD::mk("out", 2);

// Declarations

NorGate& nor = NorGate::mk("nor", 2);
komp2& trae = komp2::mk("trae");

// Connections

trae.out >> out.subset(0,2);
nor.out >> trae.in.subset(1,1);
in.subset(1,2) >> nor.in;
```

```
in.subset(0,1) >> trae.in.subset(0,1);
  return *new komp1(in.outside(),out.outside());
};

komp2& komp2::mk(String home) {

// PADs
INPAD& in = INPAD::mk("in", 2);
OUTPAD& out = OUTPAD::mk("out", 2);

// Connections

in.subset(0,2) >> out.subset(0,2);
  return *new komp2(in.outside(),out.outside());
};

class MySimApp : public SimApp {
public:
  virtual void BuildModel();
  MySimApp() {}
};

SimApp* SimApp::mk(int argc, char *argv[], char *envp[])
{ return new MySimApp; };

void MySimApp::BuildModel() {

// Declarations

FlipFlop& ff = FlipFlop::mk("ff", 2, 2);
komp0& trae = komp0::mk("trae");

// Connections

trae.out.subset(0,2).reverse() >> ff.in;
ff.out.subset(0,1) >> trae.in.subset(0,1);
ff.out.subset(0,1) >> trae.in.subset(1,1);
ff.out.subset(1,1) >> trae.in.subset(2,1);
ff.out.subset(1,1) >> trae.in.subset(3,1);
};
```

### F.1.4

Nedenstående program

```
main ()
  var = (’3’ , pla’plan’);
  plan = " --0 --1,-01 -1-,-10 -1-,10- 1--,011 1-- ";
  ff’4 2’ >> var [0..0]not >> add’3’[..1] []array’notarray 2’ >> ff;
  add[2..3] >> array
end
```

giver følgende uddata

```
#include "simlib.h"
#include "interface.h"
#include "composite.h"



class MySimApp : public SimApp {
public:
  virtual void BuildModel();
  MySimApp() {}
};
```

```
SimApp* SimApp::mk(int argc, char *argv[], char *envp[])
{ return new MySimApp; };

void MySimApp::BuildModel() {

// Declarations

char *plan[] = { --0 --1", -01 -1-", -10 -1-", "10- 1--", "011 1--" };
Adder& add = Adder::mk("add", 3);
NotGate& array0 = NotGate::mk("array0");
NotGate& array1 = NotGate::mk("array1");
FlipFlop& ff = FlipFlop::mk("ff", 4, 2);
NotGate& not = NotGate::mk("not");
PLA& pla = PLA::mk("pla", 3, 3, 5, plan);

// Connections

add.sum.subset(0,2) >> ff.in.subset(0,2);
array0.out >> ff.in.subset(2,1);
array1.out >> ff.in.subset(3,1);
pla.out >> add.inB;
not.out >> add.cIn;
ff.out.subset(0,3) >> pla.in;
ff.out.subset(3,1) >> not.in;
add.sum.subset(2,1) >> array0.in;
add.cOut >> array1.in;
ff.out.subset(0,3) >> add.inA;
};
```

### F.1.5

Nedenstående program

```
main ()
  ff'4 5' ff1'4 1';
  var0 = ff; X = var0; Y = ff1;
  if true then
    X Y 0'1' >> add'4' , ('4' notarray'4' _'1' 1'1' >> sub'add 4')
      >> X _'1' Y _'1'
  else X Y >> X Y end
end
```

giver følgende uddata

```
#include "simlib.h"
#include "interface.h"
#include "composite.h"



class MySimApp : public SimApp {
public:
  virtual void BuildModel();
  MySimApp() {}
};

SimApp* SimApp::mk(int argc, char *argv[], char *envp[])
{ return new MySimApp; };

void MySimApp::BuildModel() {

// Declarations

Adder& add = Adder::mk("add", 4);
FlipFlop& ff = FlipFlop::mk("ff", 4, 5);
FlipFlop& ff1 = FlipFlop::mk("ff1", 4, 1);
```

```
NotGate& notarray0 = NotGate::mk("notarray0");
NotGate& notarray1 = NotGate::mk("notarray1");
NotGate& notarray2 = NotGate::mk("notarray2");
NotGate& notarray3 = NotGate::mk("notarray3");
Adder& sub = Adder::mk("sub", 4);

// Connections

add.sum >> ff.in;
add.cOut >> 0;
sub.sum >> ff1.in;
sub.cOut >> 0;
ff.out >> add.inA;
ff1.out >> add.inB;
Const("0",1) >> add.cIn;
ff1.out.subset(0,1) >> notarray0.in;
ff1.out.subset(1,1) >> notarray1.in;
ff1.out.subset(2,1) >> notarray2.in;
ff1.out.subset(3,1) >> notarray3.in;
Const("0",1) >> 0;
notarray0.out >> sub.inB.subset(0,1);
notarray1.out >> sub.inB.subset(1,1);
notarray2.out >> sub.inB.subset(2,1);
notarray3.out >> sub.inB.subset(3,1);
Const("1",1) >> sub.cIn;
ff.out >> sub.inA;
};
```

### F.1.6

Nedenstående program

```
main ()
  if false then ff >> ff else if 2 >= 3 then ff >> ff
  else
    ff'16 0xf' >> zap'ff[0]' >> skel'pipestage ff[1]' >> ff
  end end
end
```

giver følgende uddata

```
#include "simlib.h"
#include "interface.h"
#include "composite.h"



class MySimApp : public SimApp {
public:
  virtual void BuildModel();
  MySimApp() {}
};

SimApp* SimApp::mk(int argc, char *argv[], char *envp[])
{ return new MySimApp; };

void MySimApp::BuildModel() {

// Declarations

FlipFlop& ff = FlipFlop::mk("ff", 16, 15);
PipeStageBoundary& skel = PipeStageBoundary::mk("skel", ff.out[1]);
ZapBoundary& zap = ZapBoundary::mk("zap", ff.out[0]);

// Connections
```

```
ff.out >> zap >> skel >> ff.in;
};
```

### F.1.7

Nedenstående program

```
main ()
  ff’4 2’/2[0]+[3,1] >> addends’2 2’ >> decoder’4’ >> encoder’4’
    >> fulladd (notarray’2’ >> xor’2’*2) >> ff
end
```

giver følgende uddata

```
#include "simlib.h"
#include "interface.h"
#include "composite.h"



class MySimApp : public SimApp {
public:
  virtual void BuildModel();
  MySimApp() {}
};

SimApp* SimApp::mk(int argc, char *argv[], char *envp[])
{ return new MySimApp; };

void MySimApp::BuildModel() {

// Declarations

Addends& addends = Addends::mk("addends", 2, 2);
Decoder& decoder = Decoder::mk("decoder", 4);
Encoder& encoder = Encoder::mk("encoder", 4);
FlipFlop& ff = FlipFlop::mk("ff", 4, 2);
FullAdder& fulladd = FullAdder::mk("fulladd");
NotGate& notarray0 = NotGate::mk("notarray0");
NotGate& notarray1 = NotGate::mk("notarray1");
XorGate& xor = XorGate::mk("xor", 2);

// Connections

fulladd.outA >> ff.in.subset(0,1);
fulladd.outB >> ff.in.subset(1,1);
xor.out >> ff.in.subset(2,1);
xor.out >> ff.in.subset(3,1);
encoder.out.subset(0,1) >> fulladd.inA;
encoder.out.subset(1,1) >> fulladd.inB;
encoder.out.subset(2,1) >> fulladd.inC;
encoder.out.subset(3,1) >> notarray0.in;
encoder.valid >> notarray1.in;
decoder.out >> encoder.in;
addends.addends[0] >> decoder.in.subset(0,2);
addends.addends[1] >> decoder.in.subset(2,2);
ff.out.subset(0,1) >> addends.multiplier.subset(0,1);
ff.out.subset(2,1) >> addends.multiplier.subset(1,1);
ff.out.subset(3,1) >> addends.multiplicand.subset(0,1);
ff.out.subset(1,1) >> addends.multiplicand.subset(1,1);
notarray0.out >> xor.in.subset(0,1);
notarray1.out >> xor.in.subset(1,1);
};
```

### F.1.8

Nedenstående program

```
main ()
  ff’8 3’/2[0;1] >> (andgate’4’ []ff[2,3]) >> prioritizer’3’;
  ff[6..] >> [0]+[1]ff2’2 1’ >> halfadd*(2^1) []prioritizer >> /4ff
end
```

giver følgende uddata

```
#include "simlib.h"
#include "interface.h"
#include "composite.h"



class MySimApp : public SimApp {
public:
  virtual void BuildModel();
  MySimApp() {}
};

SimApp* SimApp::mk(int argc, char *argv[], char *envp[])
{ return new MySimApp; };

void MySimApp::BuildModel() {

// Declarations

AndGate& andgate = AndGate::mk("andgate", 4);
FlipFlop& ff = FlipFlop::mk("ff", 8, 3);
FlipFlop& ff2 = FlipFlop::mk("ff2", 2, 1);
HalfAdder& halfadd = HalfAdder::mk("halfadd");
Prioritizer& prioritizer = Prioritizer::mk("prioritizer", 3);

// Connections

andgate.out >> prioritizer.in.subset(0,1);
ff.out.subset(2,1) >> prioritizer.in.subset(1,1);
ff.out.subset(3,1) >> prioritizer.in.subset(2,1);
ff.out.subset(0,1) >> andgate.in.subset(0,1);
ff.out.subset(4,1) >> andgate.in.subset(1,1);
ff.out.subset(1,1) >> andgate.in.subset(2,1);
ff.out.subset(5,1) >> andgate.in.subset(3,1);
halfadd.outA >> ff.in.subset(0,1);
halfadd.outB >> ff.in.subset(2,1);
halfadd.outA >> ff.in.subset(4,1);
halfadd.outB >> ff.in.subset(6,1);
prioritizer.out.subset(0,1) >> ff.in.subset(1,1);
prioritizer.out.subset(1,1) >> ff.in.subset(3,1);
prioritizer.out.subset(2,1) >> ff.in.subset(5,1);
prioritizer.valid >> ff.in.subset(7,1);
ff2.out.subset(0,1) >> halfadd.inA;
ff2.out.subset(1,1) >> halfadd.inB;
ff.out.subset(6,1) >> ff2.in.subset(0,1);
ff.out.subset(7,1) >> ff2.in.subset(1,1);
};
```

### F.1.9

Nedenstående program

```
main ()
  ff’2 3’ ff2’8 2’*2 >> /5[0;1]multiply’2 8’ orgate’8’[] >> [..7]mux’2 4’;
  ff*2 ff2 >> csadder’4’ >> [8..15]mux[0,1] >> ff;
  mux[..2] orgate*2 >> qualifiedDecoder’2’ [select]mux >> ff2
end
```

giver følgende uddata

```
#include "simlib.h"
#include "interface.h"
#include "composite.h"



class MySimApp : public SimApp {
public:
  virtual void BuildModel();
  MySimApp() {}
};

SimApp* SimApp::mk(int argc, char *argv[], char *envp[])
{ return new MySimApp; };

void MySimApp::BuildModel() {

// Declarations

CSAdder& csadder = CSAdder::mk("csadder", 4);
FlipFlop& ff = FlipFlop::mk("ff", 2, 3);
FlipFlop& ff2 = FlipFlop::mk("ff2", 8, 2);
Multiply& multiply = Multiply::mk("multiply", 2, 8);
Mux& mux = Mux::mk("mux", 2, 4);
OrGate& orgate = OrGate::mk("orgate", 8);
QualifiedDecoder& qualifieddecoder = QualifiedDecoder::mk("qualifieddecoder", 2);

// Connections

multiply.product.subset(0,4) >> mux.in[0];
multiply.product.subset(4,4) >> mux.in[1];
ff.out.subset(0,1) >> multiply.multiplier.subset(0,1);
ff.out.subset(1,1) >> multiply.multiplicand.subset(0,1);
ff2.out.subset(0,1) >> multiply.multiplicand.subset(2,1);
ff2.out.subset(1,1) >> multiply.multiplicand.subset(4,1);
ff2.out.subset(2,1) >> multiply.multiplicand.subset(6,1);
ff2.out.subset(3,1) >> multiply.multiplier.subset(1,1);
ff2.out.subset(4,1) >> multiply.multiplicand.subset(1,1);
ff2.out.subset(5,1) >> multiply.multiplicand.subset(3,1);
ff2.out.subset(6,1) >> multiply.multiplicand.subset(5,1);
ff2.out.subset(7,1) >> multiply.multiplicand.subset(7,1);
ff2.out >> orgate.in;
mux.out.subset(0,1) >> ff.in.subset(0,1);
mux.out.subset(1,1) >> ff.in.subset(1,1);
csadder.sumA >> mux.in[2];
csadder.sumB >> mux.in[3];
ff.out >> csadder.inA.subset(0,2);
ff.out >> csadder.inA.subset(2,2);
ff2.out.subset(0,4) >> csadder.inB;
ff2.out.subset(4,4) >> csadder.inC;
qualifieddecoder.out >> ff2.in.subset(0,4);
mux.out >> ff2.in.subset(4,4);
mux.out.subset(0,2) >> qualifieddecoder.in;
mux.out.subset(2,1) >> qualifieddecoder.qualifier;
orgate.out >> mux.select.subset(0,1);
orgate.out >> mux.select.subset(1,1);
};
```

### F.1.10

Nedenstående program

```
main ()
  reg’registerfile 1 1 1 2’ nand’2’ >> merger’1 1’ >> [0]reg;
```

```
  reg*3 nand*2 >> nand*2 fastmux’2 2’ >> [1..]reg;
  ff’2 2’*2 wrenff’1 0’ >> pgnode’2’ >> ff wrenff;
  prenc’priorityEncoder 2’*2 >> pgTree’2 20’ [cIn]serialAdd’2’
    >> ff2’6 3’*1+[..1] >> prenc [..3]serialAdd;
  5’8’ >> search’2 1’ qualifiedSearch’2 1’ >> validateWrite’1 1 2 0’
end
```

giver følgende uddata

```
#include "simlib.h"
#include "interface.h"
#include "composite.h"



class MySimApp : public SimApp {
public:
  virtual void BuildModel();
  MySimApp() {}
};

SimApp* SimApp::mk(int argc, char *argv[], char *envp[])
{ return new MySimApp; };

void MySimApp::BuildModel() {

// Declarations

FastMux& fastmux = FastMux::mk("fastmux", 2, 2);
FlipFlop& ff = FlipFlop::mk("ff", 2, 2);
FlipFlop& ff2 = FlipFlop::mk("ff2", 6, 3);
Merger& merger = Merger::mk("merger", 1, 1);
NandGate& nand = NandGate::mk("nand", 2);
PGNode& pgnode = PGNode::mk("pgnode", 2);
PGTree& pgtree = PGTree::mk("pgtree", 2, 20);
PriorityEncoder& prenc = PriorityEncoder::mk("prenc", 2);
QualifiedSearch& qualifiedsearch = QualifiedSearch::mk("qualifiedsearch", 2, 1);
RegisterFile& reg = RegisterFile::mk("reg", 1, 1, 1, 2);
Search& search = Search::mk("search", 2, 1);
SerialAdder& serialadd = SerialAdder::mk("serialadd", 2);
ValidateWrite& validatewrite = ValidateWrite::mk("validatewrite", 1, 1, 2, 0);
WrEnFlipFlop& wrenff = WrEnFlipFlop::mk("wrenff", 1, 0);

// Connections

merger.streamOut >> reg.wrSelect[0];
reg.rdData[0].subset(0,1) >> merger.streamIn;
reg.rdData[0].subset(1,1) >> merger.dataIn;
nand.out >> merger.mergeMask;
nand.out >> reg.wrEnable[0];
nand.out >> reg.wrData[0].subset(0,1);
fastmux.out.subset(0,1) >> reg.wrData[0].subset(1,1);
fastmux.out.subset(1,1) >> reg.rdSelect[0];
reg.rdData[0] >> nand.in;
reg.rdData[0] >> fastmux.in[0];
reg.rdData[0] >> fastmux.in[1];
nand.out >> fastmux.select.subset(0,1);
nand.out >> fastmux.select.subset(1,1);
pgnode.cOut >> ff.in;
pgnode.pOut >> wrenff.in;
pgnode.gOut >> wrenff.wrEnable;
ff.out >> pgnode.pIn;
ff.out >> pgnode.gIn;
wrenff.out >> pgnode.cIn;
ff2.out.subset(0,4) >> prenc.in;
ff2.out.subset(4,2) >> serialadd.inA;
ff2.out.subset(0,2) >> serialadd.inB;
```

```
pgtree.carries >> ff2.in.subset(0,2);
pgtree.cOut >> ff2.in.subset(2,1);
serialadd.sum >> ff2.in.subset(3,2);
serialadd.cOut >> ff2.in.subset(5,1);
prenc.out >> pgtree.pIn;
prenc.valid >> pgtree.gIn.subset(0,1);
prenc.out.subset(0,1) >> pgtree.gIn.subset(1,1);
prenc.out.subset(1,1) >> pgtree.cIn;
prenc.valid >> serialadd.cIn;
search.hits.subset(0,1) >> validatewrite.idxIn[0];
search.hits.subset(1,1) >> validatewrite.dataIn[0].subset(0,1);
qualifiedsearch.hits.subset(0,1) >> validatewrite.dataIn[0].subset(1,1);
qualifiedsearch.hits.subset(1,1) >> validatewrite.wrEnable[0];
Const("0",2) >> search.keys;
Const("0",1) >> search.key;
Const("0",2) >> qualifiedsearch.keys;
Const("1",1) >> qualifiedsearch.key;
Const("1",2) >> qualifiedsearch.valid;
};
```

### F.1.11

Nedenstående program

```
main ()
  sram'1 1 2 2'*3 >> sram
end
```

giver følgende uddata

```
#include "simlib.h"
#include "interface.h"
#include "composite.h"



class MySimApp : public SimApp {
public:
  virtual void BuildModel();
  MySimApp() {}
};

SimApp* SimApp::mk(int argc, char *argv[], char *envp[])
{ return new MySimApp; };

void MySimApp::BuildModel() {

// Declarations

SRam& sram = SRam::mk("sram", 1, 1, 2, 2);

// Connections

sram.dataOut[0] >> sram.wrEnable[0];
sram.dataOut[0] >> sram.dataIn[0];
sram.dataOut[0] >> sram.rdEnable[0];
};
```

Når dette forsøges oversat, gives følgende fejlmeddelelser:

```
main.cc: In method 'void MySimApp::BuildModel()':
main.cc:20: 'SRam' undeclared (first use this function)
main.cc:20: (Each undeclared identifier is reported only once
main.cc:20: for each function it appears in.)
main.cc:20: 'sram' undeclared (first use this function)
main.cc:20: parse error before '::'
```

## F.2 Fejlbehæftede programmer

### F.2.1

Heltalsvariabel ikke erklæret.

```
main ()
  flipflop’1 tal’ >> flipflop
end

Error at line 2, col 14:
the variable tal is not defined
```

### F.2.2

Heltalsvariabel med ulovlig type.

```
main (bit)
  if bit then flipflop’bit 0’ >> flipflop
  else flipflop’1 0’ >> not >> flipflop end
end

Error at line 2, col 15:
 integer variable is assigned non-integer value in flipflop
```

### F.2.3

Boolsk variabel ikke erklæret.

```
main ()
  if bit then flipflop’1 0’ >> flipflop
  else flipflop’1 0’ >> not >> flipflop end
end

Error at line 2, col 6:
the variable bit is not defined
```

### F.2.4

Boolsk variabel med ulovlig type.

```
main (bit)
  ff’bit 0’ >> not;
  if bit then not >> ff else not >> ff end
end

Error at line 3, col 6:
bit has to be a boolean
```

### F.2.5

Ulovlig erklæring af skel: skeltype findes ikke.

```
main ()
  ff’1 0’ >> skel’skel ff[0]’ >> ff
end

Error at line 2, col 19:
no boundary skel exists
```

### F.2.6

Ulovlig erklæring af skel: skeltype kan ikke udledes af navn.

```
main ()
  ff'1 0' >> skel'ff[0]' >> ff
end
```

```
Error at line 2, col 14:
the boundary skel matches no boundary type
```

### F.2.7

Ulovlig erklæring af skel: komponent findes ikke.

```
main ()
  ff'1 0' >> skel'zap not[0]' >> ff
end
```

```
Error at line 2, col 23:
no building block not exists
```

### F.2.8

Ulovlig erklæring af skel: udbit udenfor komponents bitvektor.

```
main ()
  ff'1 0' >> skel'zap ff[1]' >> ff
end
```

```
Error at line 2, col 26:
derivation out of bounds (of bit array)
0
```

### F.2.9

Ulovlig erklæring af type: typen kan ikke udledes.

```
main ()
  ff'1 0' >> nannd'1' >> ff
end
```

```
Error at line 2, col 14:
the building block name nannd matches no boulding block type
```

### F.2.10

Ulovlig erklæring af type: en parameter for meget.

```
main ()
  ff'1 0' >> xor'1 2' >> ff
end
```

```
Error at line 2, col 14:
the building block xor is assigned to many values
```

### F.2.11

Ulovlig erklæring af type: en parameter for lidt.

```
main ()
  ff'1 0' >> xor >> ff
end
```

```
Error at line 2, col 14:
the building block xor is assigned to few values
```

### F.2.12

Størrelsesvariabel = 0.

```
main ()
    ff’0 0’ >> not >> ff
end

Error at line 0, col 0:
channel size has to be positive
```

### F.2.13

Størrelsesvariabel $< 0$.

```
main ()
  ff’2-3 0’ >> not >> ff
end

Error at line 0, col 0:
channel size has to be positive
```

### F.2.14

Ulovlig erklæring af vektor: typen kan ikke udledes.

```
main ()
  ff’3 0’ >> nanndarray >> ff
end

Error at line 2, col 14:
the building block name nanndarray matches no boulding block type
```

### F.2.15

Ulovlig erklæring af vektor: en parameter for meget.

```
main ()
  ff’4 0’ >> orarray’2 2 1’ >> ff
end

Error at line 2, col 14:
the building block orarray0 is assigned to many values
```

### F.2.16

Ulovlig erklæring af vektor: en parameter for lidt.

```
main ()
  ff’4 0’ >> orarray’2’ >> ff
end

Error at line 2, col 14:
the building block orarray0 is assigned to few values
```

### F.2.17

Ulovlig erklæring af vektor: antal = 0.

```
main ()
  ff’4 0’ >> orarray’0 2’ >> ff
end

Error at line 2, col 14:
the array orarray should contain at least one element
```

### F.2.18

Ulovlig erklæring af vektor: antal $< 0$.

```
main ()
  ff’4 0’ >> orarray’2-3 2’ >> ff
end
```

```
Error at line 2, col 14:
the array orarray should contain at least one element
```

### F.2.19

Størrelsesvariabel = 0.

```
main ()
  ff’4 0’ >> orarray’2 0’ >> ff
end
```

```
Error at line 0, col 0:
channel size has to be positive
```

### F.2.20

Størrelsesvariabel $< 0$.

```
main ()
  ff’4 0’ >> orarray’2 2-3’ >> ff
end
```

```
Error at line 0, col 0:
channel size has to be positive
```

### F.2.21

2’0’.

```
main ()
  ff’1 0’ 2’0’ >> ff
end
```

```
Error at line 2, col 11:
constant too big for the given size
```

### F.2.22

_’0’.

```
main ()
  ff’1 0’ >> ff _’0’
end
```

```
11
Uncaught exception:
fejlSplit
```

### F.2.23

’0’.

```
main ()
  ff’1 0’ >> ’0’ not >> ff
end
```

```
Error at line 2, col 25:
in-bit already connected
```

### F.2.24

Ulovligt bitmønster: stik findes ikke.

```
main ()
  ff'2 0' >> or'2'[cout] >> ff
end

Error at line 2, col 20:
the variable cout is not defined
```

### F.2.25

Ulovligt bitmønster: stik på bitvektor, der ikke er komponent.

```
main ()
  ff'2 0' >> (not not1)[out] >> ff
end

Error at line 2, col 25:
the variable out is not defined
```

### F.2.26

Ulovligt bitmønster: bit $< 0$.

```
main ()
  ff'1 0' >> not >> ff;
  1'1' >> [(2-3)]not
end

Error at line 2, col 14:
in-bit 0 of not not connected
```

### F.2.27

Ulovligt bitmønster: bit $>$ sidste bit.

```
main ()
  ff'1 0' >> not >> ff;
  1'1' >> [1]not
end

Error at line 3, col 12:
derivation out of bounds (of bit array)
```

### F.2.28

Ulovligt bitmønster: bit $>$ sidste bit i felt.

```
main ()
  ff'4 0' >> /2[0;2]andarray'2 2'*2 >> ff
end

Error at line 2, col 19:
derivation out of bounds (of bit array)
```

### F.2.29

Ulovlig indafledning: del i 0 felter.

```
main ()
  ff’4 0’ >> /0andarray’2 2’*2 >> ff
end

Uncaught exception:
Div
```

### F.2.30

Ulovlig indafledning: del i $< 0$ felter.

```
main ()
  ff’4 0’ >> /(3-4)andarray’2 2’*2 >> ff
end

Uncaught exception:
Size
```

### F.2.31

Ulovlig indafledning: del i antal felter, der ikke går op i samlet antal bit.

```
main ()
  ff’4 0’ >> /3andarray’2 2’*2 >> ff
end

Error at line 2, col 15:
the size of the input does not match the size of the output in the connection
```

### F.2.32

Ulovligt udafledning: del i 0 felter.

```
main ()
  ff’4 0’ >> andarray’2 2’*2/0 >> ff
end

Uncaught exception:
Div
```

### F.2.33

Ulovligt udafledning: del i $< 0$ felter.

```
main ()
  ff’4 0’ >> andarray’2 2’*2/(3-4) >> ff
end

Uncaught exception:
Size
```

### F.2.34

Ulovligt udafledning: gang med 0.

```
main ()
  ff’4 0’ >> andarray’2 2’*0 >> ff
end

Error at line 2, col 33:
the size of the input does not match the size of the output in the connection
```

### F.2.35

Ulovligt udafledning: gang med $< 0$.

```
main ()
  ff’4 0’ >> andarray’2 2’*(3-4) >> ff
end

Error at line 2, col 37:
the size of the input does not match the size of the output in the connection
```

### F.2.36

Ulovligt udafledning: del i antal felter, der ikke går op i samlet antal udbit.

```
main ()
  ff’4 0’ >> andarray’2 2’/3[0]+[0] >> ff
end

Error at line 2, col 28:
the size of the input does not match the size of the output in the connection
```

### F.2.37

For stor konstant.

```
main ()
  ff’1 0’ >> _’1’; 2’1’ >> ff
end

Error at line 2, col 20:
constant too big for the given size
```

### F.2.38

Ulovlig brug af gennemløbsudtryk.

```
main ()
  ff’1 0’ >> _’1’; ’1’ >> ff
end

15
Uncaught exception:
fejlSplit
```

### F.2.39

Ulovlig brug af gennemløbsudtryk.

```
main ()
  ff’1 0’ >> ’1’; 0’1’ >> ff
end

15
Uncaught exception:
fejlSplit
```

### F.2.40

Definition af to komponenter med samme navn.

```
main ()
  ff’1 0’ >> ’1’ >> ff; ff’1 0’ >> ff
end

Error at line 2, col 25:
ff is already defined
```

### F.2.41

Definition af komponent med samme navn som plan.

```
main ()
  plan = " 0 1 "; plan’ff 1 0’ >> ’1’ >> plan
end

Error at line 2, col 19:
plan is already defined (as a plane)
```

### F.2.42

Definition af komponent med samme navn som bitvariabel.

```
main ()
  var = ff; var’not’ >> ff’1 0’ >> var
end

Error at line 2, col 13:
var is already defined (as a bit variable)
```

### F.2.43

Definition af komponent med samme navn som stik for programblokken.

```
main ()
  ff’1 0’ >> minkomp >> ff
end

minkomp (ind 1) >> (ud 1)
  ind >> ud’not’ >> ud
end

Error at line 6, col 10:
ud is already defined (as a channel)
```

### F.2.44

Definition af komponent og skel med samme navn.

```
main ()
  komp’ff 1 0’ >> komp’zap ff[0]’ >> komp
end

Error at line 2, col 19:
komp is already defined
```

### F.2.45

Definition af to skel med samme navn.

```
main ()
  ff'1 0' >> skel'zap ff[0]' >> skel'pipestage ff[0]' >> ff
end

Error at line 2, col 33:
skel is already defined
```

### F.2.46

Definition af skel med samme som plan.

```
main ()
  plan = " 0 1 "; ff'1 0' >> plan'zap ff[0]' >> ff
end

Error at line 2, col 30:
plan is already defined (as a plane)
```

### F.2.47

Definition af skel med samme navn som bitvariabel.

```
main ()
  var = ff; ff'1 0' >> var'zap ff[0]' >> var
end

Error at line 2, col 24:
var is already defined (as a bit variable)
```

### F.2.48

Definition af skel med samme navn som stik for programblokken.

```
main ()
  ff'1 0' >> minkomp >> ff
end

minkomp (ind 1) >> (ud 1)
  ind >> ud'zap ind[0]' >> ud
end

Error at line 6, col 10:
ud is already defined (as a channel)
```

### F.2.49

Ud- og indstørrelse forskellige i forbindelse af bitvektorer.

```
main ()
  ff'2 0' >> not >> ff
end

Error at line 2, col 14:
the size of the input does not match the size of the output in the connection
```

### F.2.50

Udstørrelse = indstørrelse = 0.

```
main ()
  ff’2 0’[] >> []not >> ff; ff >> not
end

Error at line 2, col 25:
the size of the input does not match the size of the output in the connection
```

### F.2.51

Ud- og indstørrelse forskellige i forbindelse af bitvektorer via skel.

```
main ()
  ff’2 0’ >> zap’ff[0]’ >> not >> ff
end

Error at line 2, col 28:
the size of the input does not match the size of the output in the connection
```

### F.2.52

Ulovlig brug af skel.

```
main ()
  zap’ff[0]’ >> ff’1 0’ >> ff
end

Error at line 2, col 3:
invalid use of boundary zap
```

### F.2.53

Ulovlig brug af skel.

```
main ()
  ff’1 0’ >> ff >> zap’ff[0]’
end

Error at line 2, col 20:
invalid use of boundary zap
```

### F.2.54

Ulovlig brug af skel.

```
main ()
  ff’1 0’ >> zap’ff[0]’ , not >> ff _’1’
end

Error at line 2, col 14:
invalid use of boundary zap
```

### F.2.55

Størrelserne af indbitvektorerne forskellige i bitvektor , bitvektor.

```
main ()
  ff’2 0’ >> xor’2’ , not >> ff
end

Error at line 2, col 23:
the size of the input does not match the size of the other input in split (,)
```

### F.2.56

Indstørrelser = 0 i bitvektor , bitvektor.

```
main ()
  ff’1 0’[] >> []not , []xor’1’[] >> ff;
  ff >> not[] , xor >> _’1’
end

Error at line 2, col 16:
the size of the input does not match the size of the output in the connection
```

### F.2.57

Ulovlig brug af skel.

```
main ()
  ff’2 0’ >> zap’ff[0]’ not >> ff
end

Error at line 2, col 14:
invalid use of boundary zap
```

### F.2.58

Ulovlig brug af skel.

```
main ()
  ff’2 0’ >> (zap’ff[0]’) >> ff
end

Error at line 2, col 15:
invalid use of boundary zap
```

### F.2.59

To planer med samme navn.

```
main ()
  plan = " 0 1 "; plan = " 0 1 "; ff’1 0’ >> pla’plan’ >> ff
end

Error at line 2, col 19:
plan is already defined
```

### F.2.60

Definition af plan med samme navn som stik for programblokken.

```
main ()
  ff’1 0’ >> minkomp >> ff
end

minkomp (ind 1) >> (ud 1)
  ind = " 0 1 "; ind >> pla’ind’ >> ud
end

Uncaught exception:
fejlTyper
```

### F.2.61

Ulovlig definition af plan: to andplaner med forskellig længde.

```
main ()
  plan = " 01 1,001 1 "; ff'1 0' >> pla'plan' >> ff
end

Error at line 2, col 12:
size of and planes differ
```

### F.2.62

Ulovlig definition af plan: to orplaner med forskellig længde.

```
main ()
  plan = " 01 1,11 11 "; ff'1 0' >> pla'plan' >> ff
end

Error at line 2, col 15:
size of or planes differ
```

### F.2.63

To bitvariable med samme navn.

```
main ()
  var = ff; var = ff; ff'1 0' >> var
end

Error at line 2, col 13:
var is already defined
```

### F.2.64

Rekursion i bitvariable: to bitvariable indgår i hinandens værdi.

```
main ()
  var0 = var1; var1 = var0; ff'1 0' >> var >> ff
end

Error at line 2, col 16:
recursion in the bit variable var1
```

### F.2.65

Rekursion i bitvariable: en bitvariabel indgår i sin egen værdi.

```
main ()
  var = ff var; ff'1 0' >> var
end

Error at line 2, col 3:
recursion in the bit variable var
```

### F.2.66

Bit i indbitvektor forbindes ikke.

```
main ()
  ff'2 0' >> not _'1' >> [0]ff
end

Error at line 2, col 3:
in-bit 1 of ff not connected
```

### F.2.67

Bit i udbitvektor forbindes ikke.

```
main ()
  ff’2 0’[0] >> not*2 >> ff
end

Error at line 2, col 3:
out-bit 1 of ff not connected
```

### F.2.68

Bit i indbitvektor forbindes to gange.

```
main ()
  ff’1 0’ >> not >> ff; ff >> not
end

Error at line 2, col 31:
in-bit already connected
```

### F.2.69

Krop med ulovlig brug af rekursion (værdisættet falder ikke).

```
main ()
  ff’1 0’ >> minkomp’1’ >> ff
end

minkomp (ind 1, tal) >> (ud 1)
  ind >> minkomp’tal’ >> ud
end

Recursion with building block minkomp where the size of the building block does not decrease
```

### F.2.70

Oversæt med en parameter for lidt. Programmet F.1.1 kaldes med `2` (det skal kaldes med heltal og bool). Uddata bliver:

```
main should have 2 arguments
```

### F.2.71

Oversæt med en parameter for meget. Programmet F.1.1 kaldes med `2 true 3` (det skal kaldes med heltal og bool). Uddata bliver:

```
main should have 2 arguments
```

### F.2.72

Manglende hovedprogram.

```
minkomp (ind 1, tal) >> (ud 1)
  ind >> ud
end

main is missing
```

### F.2.73

To hovedprogrammer.

```
main ()
  ff'1 0' >> ff
end

main ()
  ff'1 0' >> ff
end

main defined twice
```

### F.2.74

Stik med størrelse $0$.

```
main ()
  ff'1 0' >> minkomp >> ff
end

minkomp (ind 0) >> (ud 1)
  0'1' >> not >> ud
end

Error at line 5, col 14:
channel size has to be positive
```

### F.2.75

Stik med størrelse $< 0$.

```
main ()
  ff'1 0' >> minkomp >> ff
end

minkomp (ind 2-3) >> (ud 1)
  0'1' >> not >> ud
end

Error at line 5, col 14:
channel size has to be positive
```

### F.2.76

Komponent uden stik.

```
main ()
  minkomp; ff'1 0' >> ff
end

minkomp () >> ()
  0'1' >> _'1'
end

Error at line 5, col 1:
minkomp have no channels
```

### F.2.77

Komponent med samme navn som indbygget komponent.

```
main ()
  ff'1 0' >> notgate >> ff
end

notgate (in 1) >> (out 1)
  in >> out
end

Error at line 5, col 1:
a building block type with name notgate already exists
```

### F.2.78

Definition af to komponenter med samme navn.

```
main ()
  ff'1 0' >> minkomp >> ff
end

minkomp (ind 1) >> (ud 1)
  ind >> ud
end

minkomp (ind 1) >> (ud 1)
  ind >> ud
end

Error at line 9, col 1:
a building block type with name minkomp already exists
```

# Litteratur


[And01] Finn Schiermer Andersen. *Dat1E KB3: SimSys*. Datalogisk Institut, Københavns Universitet, 2001. Trykkeri: *HCØ Tryk*.

[Haw] http://www.cse.ogi.edu/PacSoft/projects/Hawk.

[Lav] http://www.cs.chalmers.se/ koen/Lava/.

[Sim] http://www.diku.dk/teaching/2001e/dat1e/SimSys/simintro.html.

[SR95] Robin Sharp and Ole Rasmussen. *An Introduction to Ruby*. Dept. of Computer Science, DTU, Oct 1995. Teaching notes.

[Ver] http://www.ee.ed.ac.uk/ gerard/Teach/Verilog/.