MASTER

Het ontwerp van een DMA controller met de ASA Silicon Compiler volgens de Top Down ontwerp filosofie

Crijns, J.H.H.J.

Award date:
1989

Technische Universiteit Eindhoven
FACULTEIT DER ELEKTROTECHNIEK
Vakgroep Digitale Systemen

Het ontwerp van een DMA controller met de ASA Silicon Compiler volgens de Top Down ontwerp filosofie.

Door: J.H.H.J. Crijns

Afstudeerverslag van: J.H.H.J. Crijns
idnr.: 180394
Lemmenhoek 8
6035 AK Ospel
tel. 04951-31897

Afstudeerhoogleraar: Prof. ir. M.P.J. Stevens
Begeleider TUE: Ir. F.P.M. Budzelaar
Begeleider Sagantec: Ing. J. Scheelen

Eindhoven, Augustus 1989.

SAMENVATTING

Er is een vierkanaals dma controller ontworpen als testcase voor de toepasbaarheid van de ASA Silicon Compiler als een tool bij de Top Down ontwerpfilosofie.

De Top Down filosofie komt in het kort op het volgende neer. Er wordt eerst een systeembeschrijving gemaakt die de specificatie vormt van het circuit. Deze beschrijving wordt steeds verder opgesplitst in functionele en later structurele blokken totdat we op gatesniveau zijn aangeland. Na elke decompositie dient de beschrijving simuleerbaar te zijn om de juistheid van het ontwerp te kunnen garanderen.

De ASA Silicon Compiler is een software pakket, dat een systeemontwerper, zonder specifieke IC ontwerpkennis, in staat stelt een IC op basis van standardcelllayout te ontwerpen. De benodigde IC ontwerpkennis is in feite in ASA ingebouwd als een soort artificiele intelligentie van de compiler.

Er is gebleken dat met ASA op elk functioneel niveau een simuleerbare beschrijving te maken is in de Sagantec Input Description (SID). Maar op het niveau van gates zijn we aangewezen op andere tools, die timing verificatie en timing analyse voor hun rekening kunnen nemen. Dit is een gevolg van het feit dat ASA alleen een functionele simulator bezit, die technologie onafhankelijk alle gates simuleert met een standaard delay van 1 time unit.

De ASA compiler houdt de hierarchie van het ontwerp aan tot op het niveau van layout. Dus de functionele decompositie op het hoogste niveau heeft invloed op het placement van de verschillende blokken op de chip. Dit kan oppervlakteverkwistende gevolgen hebben omdat een simpele functionele beschrijving van een deelblok helemaal niet hoeft te leiden tot een kleine oppervlakte op de chip.

De compiler is in staat om state machines en logische functies te vertalen naar gates of PLA's. Hij neemt dus een groot deel van de implementatiefase uit handen van de ontwerper. Tevens garandeert ASA correctness by construction.

De ASA Silicon Compiler is zeker te gebruiken als een handige tool bij het Top Down ontwerpen. Hij dient echter om het gehele ontwerptraject te kunnen bestrijken uitgebreid te worden met een kritisch pad analyser ten behoeve van de timing verificatie en een testvectorgenerator om, op een eenvoudigere manier dan handmatig, testvectoren te genereren.

INHOUDSOPGAVE

1 INLEIDING

Door de steeds groter wordende integratie van transistoren op een chip, is er een grote behoefte ontstaan aan gestructureerde ontwerpmethoden en aan tools ter ondersteuning van deze methoden. Een methode die veel wordt toegepast bij allerlei soorten ontwerpen is de Top Down Ontwerpmethode.

Een andere reden voor de steeds groter wordende behoefte aan goede tools en strategiën is de groeiende vraag naar ASIC's (Application Specific Integrated Circuits). Vroeger was het ontwerpen van ASIC's en IC's in het algemeen een aangelegenheid van de chip fabrikanten en design houses, want zij waren de enigen die IC ontwerpkennis in huis hadden. Door nu IC ontwerpkennis in te bouwen in tools voor het ontwerpen van IC's is het mogelijk geworden om systeemontwerpers met een beperkte IC technologiekennis, met behulp van deze tools en de top down strategie IC's te laten ontwerpen.

Een tool ter ondersteuning van de top down filosofie zou de door Sagantec Europe BV ontwikkelde ASA Silicon Compiler kunnen zijn. Om de toepasbaarheid van ASA als tool ter ondersteuning van de top down ontwerpstrategie te onderzoeken heb ik als testcase een eenvoudige dma-controller ontworpen.

De top down ontwerpmethode, zoals voorgesteld tijdens de "Structured VLSI Design Course" aan de Technische Universiteit Eindhoven, Faculteit der Electrotechniek, vakgroep Digitale Systemen (EB), zal worden beschreven. Vervolgens zullen de mogelijkheden van de ASA Silicon Compiler aan de orde komen en ook zal het ontwerp van de dma-controller worden beschreven.

2 TOP DOWN DESIGN

De specificatie van een digitaal systeem kan uitstekend worden beschreven in een pascal-achtige algoritmische vorm. Met daartoe geëigende CAD-hulpmiddelen (tools) kan zo het gedrag van een systeem al in de beginfase van het ontwerp worden gesimuleerd en geverifieerd.

Het systeem wordt gepartioneerd in deelsystemen, waarvan de specificaties eveneens in algoritmische vorm worden opgesteld (vgl. procedures bij software). Nu kunnen de onderdelen worden gesimuleerd maar evengoed het daaruit samengestelde systeem.

Stap voor stap wordt dit proces tot op het niveau van de layout voortgezet.

De 'top-down' ontwerpmethode komt in het kort op het volgende neer. Er wordt eerst een systeembeschrijving gemaakt die de specificatie vormt van het circuit. Dit wordt in het vervolg de algemene gedragsbeschrijving genoemd.

Die algemene gedragsbeschrijving wordt geverifieerd met behulp van een functionele simulator.

Vervolgens voeren we een functionele decompositie uit. Deze decompositie is een opsplitsing in processen, het is dus in feite de procesherkenning. We proberen de processen, functies, te onderscheiden waaruit het systeem is opgebouwd. Deze processen beschrijven we nu weer in pseudo pascal.

Beide bovengenoemde beschrijvingen kunnen worden weergegeven met een Hardware Description Language zoals bijvoorbeeld SID (Sagantec Input Description).

Deze HDL beschrijvingen worden na elke verdere decompositie functioneel gesimuleerd als apart blok en later als totale beschrijving.

De functionele decompositie wordt herhaald tot we op een niveau zijn aangeland van waaruit het triviaal is om een decompositie te doen naar een datapad en een controller. Hierbij bestaat het datapad uit een aantal logische en arithmetische operatoren. En wat

betreft de controller doen we op dit niveau een state synthese om aan te geven welke operaties in een bepaalde toestand worden uitgevoerd en in welke sequentie de handelingen worden uitgevoerd.

Nu hebben we een datapad dat bestuurd kan worden door een finite state machine. Dit geheel kan nu ook weer functioneel gesimuleerd worden.

Vervolgens wordt overgegaan tot de implementatie van het datapad en de controller. We kunnen hierbij gebruik maken van standaardcomponenten uit een bibliotheek waarna we een beschrijving hebben op het niveau van gates.

We kunnen nu nog een laatste functionele simulatie uitvoeren om ons ontwerp te verifiëren.

Nu er een beschrijving op gatesniveau is gaan we een kritisch pad analyse doen ten behoeve van de timing verificatie. Bij de timing verificatie wordt rekening gehouden met de delays van de gates en met de delays ten gevolge van de loading van de outputs van de gates.

Er dient nu ook testvectorgeneratie gedaan te worden om uiteindelijk de testbaarheid van het ontworpen IC te kunnen garanderen.

We kunnen nu tot slot een layout genereren, die we dan onderwerpen aan een timing analyse en aan een design rule check.

Het verschil tussen een timing verificatie en een timing analyse is hierin gelegen dat er bij een timing analyse ook nog rekening gehouden wordt met de capaciteiten en weerstanden van de bedradingen op de chip.

Bovenstaande beschrijving staat nog eens beknopt samengevat in figuur 2.1.

Door deze gestructureerde aanpak kunnen complexe systemen worden gerealiseerd zonder dat tijdens het ontwerpen het overzicht verloren gaat. Essentieel voor het volgen van de top-down ontwerpmethode is een grote vrijheid in de keuze van de bouwstenen op het laagste niveau van de hierarchie.

*** Algemene gedragsbeschrijving**

***Functionele decompositie**

-procesherkenning

-pseudopascal-beschrijving

***Hardware Description Language (Sagantec Input Description)**

***Functioneel Simuleren**

***Decompositie naar datapad en controller**

-logische en arithmetische operatoren

-state synthese

***Functioneel Simuleren**

***Implementatie**

-kritisch pad analyse

-timing verificatie

-testvectorgeneratie

***Layout**

-timing analyse

-Design Rule Check

Figuur 2.1 Top Down Ontwerpstrategie overzicht.

Top Down is een filosofie die op zeer grote ontwerpen zeker aan te bevelen is. Je doet eerst een functionele decompositieslag en de zodanig verkregen processen worden beschreven in een hardware description language zoals SID. Om op het allerhoogste niveau een beschrijving te maken in welke beschrijvingstaal ook is een misschien wel te grote hoeveelheid werk. Ten eerste omdat het een bijna onmogelijke opgave is om een alles omvattend algorithme te schrijven, wat overeenkomt met het ontwerpen van een grote finite state machine. Dat zo'n complexe fsm al snel een groot bord spagetti wordt valt niet moeilijk in te zien. Ten tweede omdat zo'n algorithme schrijven waarschijnlijk veel meer tijd zou kosten dan een aantal aanpassingen aan submodulen na een simulatie, m.a.w. de tijd benodigd voor het beschrijven van een ontwerp na functionele decompositie plus de simulatietijd zijn waarschijnlijk veel korter dan het schrijven van een allesomvattende beschrijving.

FUNCTIONEEL ONTWERPEN

Functioneel ontwerpen houdt in dat je een electronische bouwsteen volledig gaat beschrijven alsof het een stuk software betrof. Je gaat dus alle eigenschappen van de bouwsteen in een algoritme beschrijven.

Als we op dit punt een zeer intelligente silicon compiler tot onze beschikking hadden zouden we na de functionele beschrijving van een bouwsteen alleen nog maar hoeven te vertellen hoe snel deze bouwsteen moet zijn en dan zou de compiler een hardware beschrijving kunnen afleveren. Aangezien er bij mijn weten geen compiler bestaat die reeds zo ver naar de top van het top-down traject ontwikkeld is zullen we bepaalde functies en sequenties van elkaar opvolgende handelingen zelf nauwkeuriger moeten specificeren. Er bestaan compilers die reeds een vrij hoog niveau kunnen vertalen. Zij kunnen bijvoorbeeld al een groot stuk state synthese zelf uitvoeren en zijn ook mede hierdoor in staat arithmetische operatoren te vertalen, zie hiervoor bijvoorbeeld het artikel van Pangrle en Gajski [lit.8].

Bepaalde functies verder uitwerken wil zeggen dat we deze functies op een niveau gaan beschrijven dat door de compiler kan worden omgezet in gates.

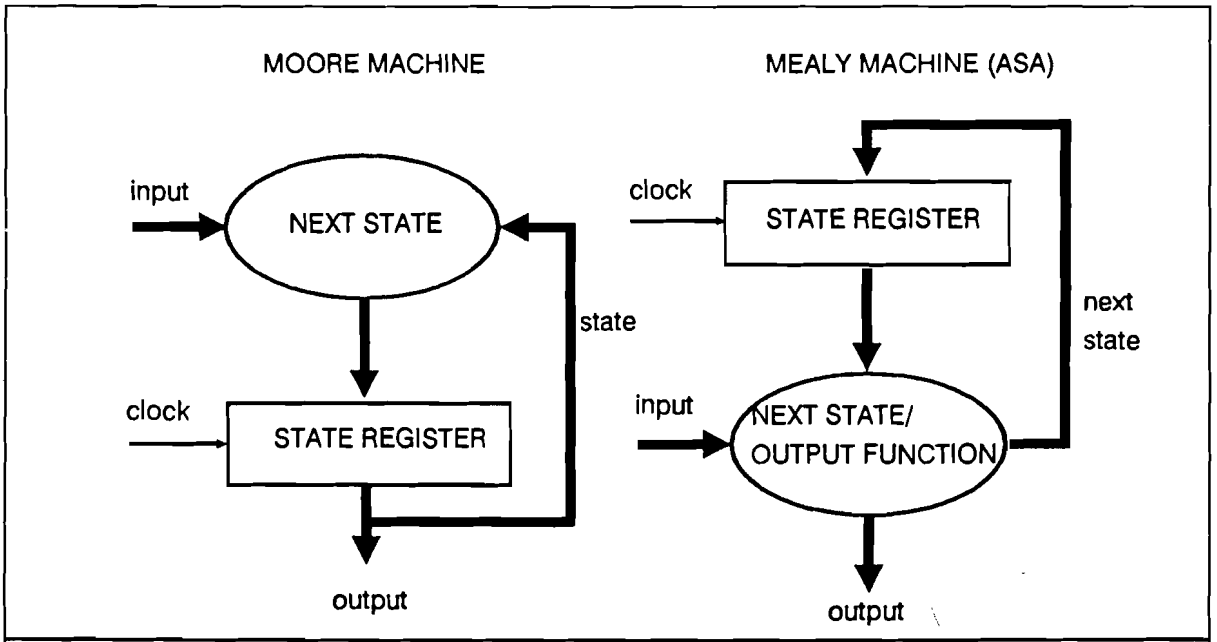
De sequentie van handelingen bepalen doen we door een finite state machine beschrijving te maken waarvan de in- en outputsignalen de controllijnen van en naar de functieblokken zijn. We beschrijven dus een datapad (de functieblokken) en een controller (de fsm beschrijving).

Het functionele ontwerp is dus de basisbeschrijving van een te realiseren bouwsteen. Deze beschrijving kan gesimuleerd worden en zodoende is het mogelijk met een klant of andere opdrachtgever na te gaan of dit wel precies is wat hij hebben wil.

SYNCHROON ONTWERPEN

Het synchroon ontwerpen houdt in dat alle informatie wordt overgenomen op de opgaande (in ons geval tenminste maar dit is optioneel) flank van de systeemklok. In ons geval zijn dan bij de volgende opgaande flank alle nieuwe toestanden weer stabiel. Synchroon ontwerpen zou echter met de mogelijkheid van Moore machines aanzienlijk worden beveiligd omdat dan de outputs zeker gedurende een groter deel van de state stabiel zouden zijn.

In figuur 2.2 zijn schematisch twee verschillende uitvoeringen van finite state machines weergegeven nl. de MEALY uitvoering en de MOORE zonder outputlogica uitvoering.



Figuur 2.2 Moore and Mealy finite state machines.

Hieruit dient vooral duidelijk te worden dat bij een Mealy machine de inputs rechtstreeks invloed kunnen hebben op de outputs en als we dus meerdere Mealy machines met elkaar doorverbinden kunnen er zeer lange kritische paden ontstaan.

Een voordeel van het gebruik van Moore machines is dat er geen onverwacht lange kritische paden kunnen ontstaan door het aan elkaar koppelen van machines, waardoor we ook bij het genereren van testvectoren in de problemen zouden kunnen komen omdat we geen gescheiden blokken meer hebben.

Gebruik van finite state machines -Mealy machines -Moore machines -Moore machines zonder outputlogica zou tot de keuzemogelijkheden van de ontwerper moeten behoren.

Het liefste zouden we een optie zien waarmee we machines konden ontwerpen zonder outputlogica en volgens het Moore principe. Dus machines waarvan de outputs de directe outputs van de geklokte flipflops (registers) zijn. We zouden dan nl. kunnen garanderen dat de outputs van de machine stabiel zijn gedurende de gehele klokcyclus.

3 SILICON COMPILERS

Silicon Compilation wordt voorgesteld als het proces om door een machine automatisch de layout van een IC te produceren. Die machine is een computer met specifieke programmatuur. Het programma dat de layout van een IC automatisch aanmaakt wordt Silicon Compiler genoemd.

De Silicon Compiler is in eerste instantie een ontwerpgereedschap voor de ontwerper van electronische systemen. Waarbij moet worden aangetekend dat de huidige Silicon Compilers zich beperken tot digitale systemen.

Het doel van de Silicon Compiler is de onwikkeling van IC's tijdens het ontwerptraject zoveel mogelijk te automatiseren. Aanvankelijk is de aandacht vooral gericht geweest op de laatste fase van het ontwerptraject, het maken van de layout. Nu gaat de aandacht vooral uit naar dat deel van het ontwerptraject waar het functionele ontwerp van het circuit tot stand komt.

Bij de definitie van de specificaties van het circuit en het creeeren en verifieren van het functionele ontwerp wordt de Silicon Compiler als intelligent ontwerpgereedschap gebruikt. De intelligentie is in de Silicon Compiler aanwezig in de vorm van programmatuur die ontwerpacties automatisch uitvoert en in de vorm van oproepbare functies in de database. Zowel de ontwerpacties als de functies kunnen worden beschouwd als IC-ontwerpkennis die in de Silicon Compiler is ingebouwd.

Omdat een Silicon Compiler het layoutontwerp voor zijn rekening neemt en daarbij geen beperkingen oplegt aan het gebruik van bepaalde bouwstenen, kan bij het functioneel ontwerpen de 'top-down' ontwerpmethode worden gevolgd.

Welbeschouwd is de Silicon Compiler een van de weinige mogelijkheden om de top-down methode te volgen. Bij ontwerpen die worden gerealiseerd met behulp van standaard IC's op PCB's is het top-down ontwerpen niet lang vol te houden. Immers, er zal altijd moeten worden gekozen uit componenten die verkrijgbaar zijn, en daarbij speelt vaak ook nog de prijs een belangrijke rol. Ook semi-custom oplossingen als Gate-array's

en standaardcel-IC's leggen de ontwerper beperkingen op, omdat al in een vroeg stadium rekening moet worden gehouden met de mogelijkheden van de gekozen semicustom-chip.

3.1 ASA SILICON COMPILER

De Sagantec Silicon Compiler (ASA) maakt het voor de ontwerper mogelijk om zonder gedetailleerde IC-ontwerpkennis, uitgaande van een hoog-niveau functionele beschrijving, op een technologie-onafhankelijke wijze IC's te ontwerpen tot op het niveau van layout.

De ASA Silicon Compiler is dus bestemd voor systeemontwerpers die niet over kennis van de IC-technologie beschikken. Deze compiler is ontwikkeld volgens de idee dat een systeemontwerper zich moet kunnen concentreren op het ontwerpen van het circuit en daarbij door de compiler wordt geholpen bij allerlei ontwerpactiviteiten. Alle activiteiten waarbij specifieke kennis van IC-technologie en layout nodig zou zijn, verlopen automatisch en onder controle van de compiler.

3.2 ASA ONDERDELEN

ASA is te zien als een bundeling van programma's en een invoertaal welke ons helpen bij het gestructureerd ontwerpen van een IC. Hieronder volgt een globale beschrijving van de voor ons van belang zijnde onderdelen van ASA.

SID

Sagantec Input Description, is een hardware description language. De beschrijving is hierarchisch opgebouwd uit **systems**, die met de buitenwereld in verbinding staan via **contacts**. Elk system kan gebruik maken van een aantal subsystems die in het blok **uses** worden aangeroepen. Deze subsystemen worden in het **connect** blok met elkaar verbonden. We kunnen voor een system ook een **function** blok beschrijven waarin het functionele gedrag van het blok wordt weergegeven.Hierin kunnen we gebruik maken van **sequential** en **parallel** statements, en tevens kunnen we delays opgeven via een **after** operator.

Functional Simulator

Dit is een simulator die een SID beschrijving kan simuleren. De stimuli worden ook als een system in SID beschreven dit system is dan te beschouwen als de buitenwereld welke we functioneel kunnen beschrijven.

De output van de simulator kan bekeken worden als hexadecimale codes en als timing diagrammen die lijken op de output van een logic analyser. De simulator biedt verder de mogelijkheid om aan de buitenzijde stimuli aan te leggen en vervolgens door de hierarchie te wandelen en op elk gewenst niveau kunnen dan probes aangelegd worden op contacten die we willen simuleren.

FSM

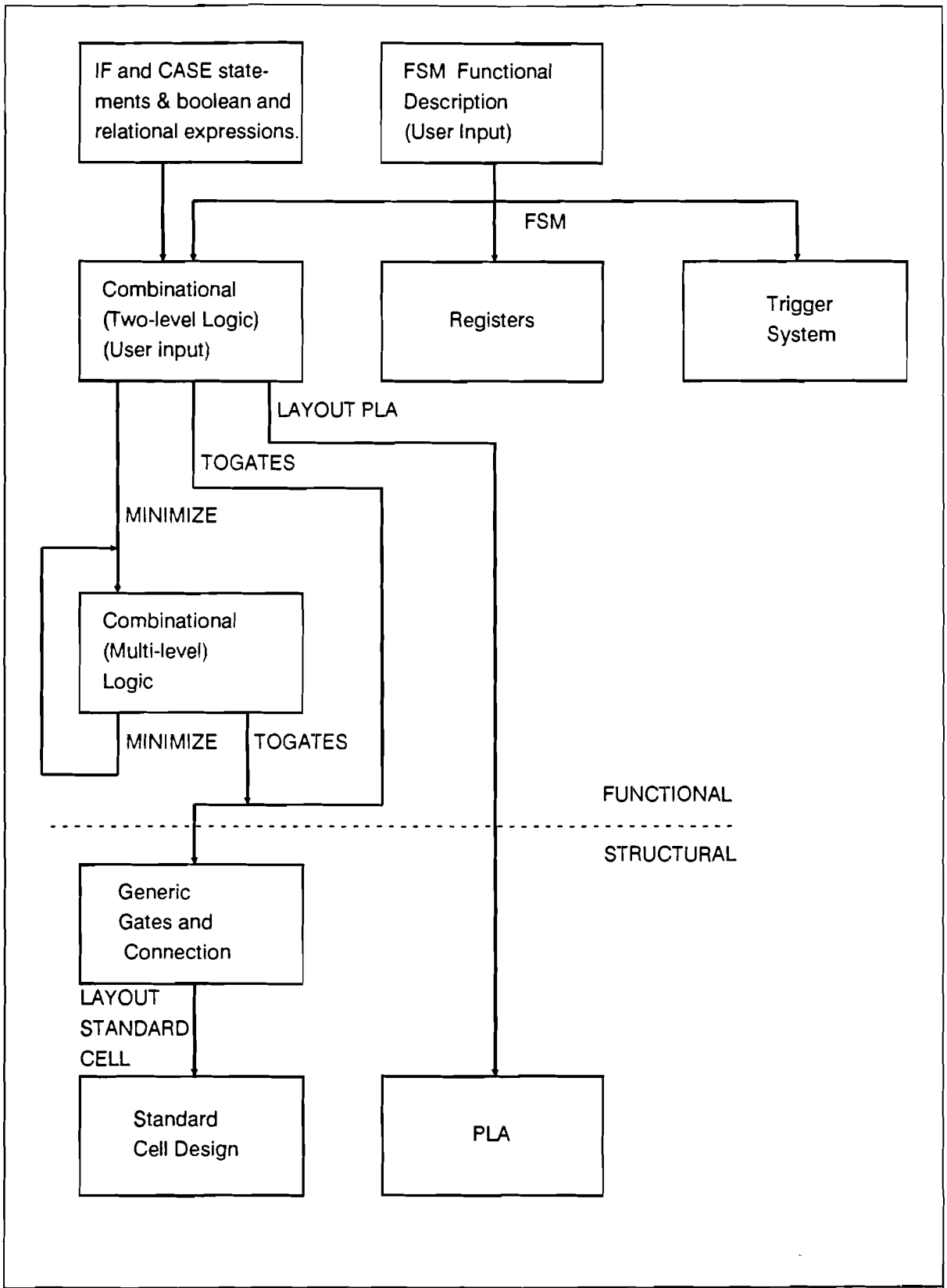
FSM is een programma dat functionele beschrijvingen kan vertalen naar logische functies. Niet alle functionele beschrijvingen zijn echter vertaalbaar. Vertaalbaar zijn: IF-THEN-ELSE en CASE statements. Verder kan FSM finite state machine beschrijvingen vertalen naar een registerdeel (state-registers) en een logicadeel (outputfunctions en next state decoder). FSM doet zelf state assignment en minimalisatie van de logische functies naar tweelaagslogica, die geimplementeerd kan worden als een PLA.

MINIMIZE

Minimalisatie. Kiezen we niet voor een PLA-implementatie van de logica dan kunnen met het minimize programma de logische functies geoptimaliseerd worden naar gates of naar pins. Optimalisatie naar pins houdt in dat we een minimaal aantal interne connecties krijgen bij de implementatie van de logica.

TOGATES

Hiermee vertalen we alle logische functies, die niet als PLA geïmplementeerd worden, naar een implementatie in gates.



Figuur 3.1 Flow chart van de ASA onderdelen tot layout

Na elke stap blijft de beschrijving een SID beschrijving, ASA voegt dus informatie toe aan onze initiele SID beschrijving maar het blijft SID en we zouden dus ook zelf op een lager niveau SID kunnen schrijven en zodoende enkele stappen van de compiler overslaan.

Tot op gatesniveau kunnen we dus compleet technologie onafhankelijk ontwerpen met de ASA Compiler.

Na ons gehele ontwerp vertaald te hebben naar gates of pla's hebben we met ASATOLASAR of ASATOSDL interfaces naar netlists waarop we timingverificatie kunnen uitvoeren met LASAR van Teradyne of HILO van Gen Rad.

Nu kunnen we door ASA een layout in een bepaalde technologie van een bepaalde foundry laten aanmaken. Dit gebeurt in vijf stappen.

Place:

Layoutgeneratie van leafmodules als standardcell, PLA of regelmatige structuur. Vervolgens doet het place algorithme de floorplanning van deze modules met gebruikmaking van de structurele informatie (hierarchie) die we in onze SID beschrijving hebben ingebouwd.

Resequence:

De in slices geplaatste modules worden herrangschikt om zodanig de verwachte totale wirelength te optimaliseren en de verwachte oppervlakte voor routing te minimaliseren.

Orient:

Roteert en spiegelt leaf modules om de totale chip oppervlakte en de totale bedradingslengte te reduceren.

Route:

Global router, door welk kanaal loopt welk net.

Assemble:

Channel router, uiteindelijke routing, hierna is de layout af.

MAKECIF

Dit is de laatste stap die met ASA wordt uitgevoerd. Hiermee wordt de layoutinformatie voor de maskers voor een foundry aangemaakt in Caltech Intermediate Form (CIF).

4 ASA ALS TOOL

De functionele simulator binnen ASA is op elk functioneel niveau bruikbaar, omdat de beschrijving altijd SID blijft.

De Top beschrijving (gedragsbeschrijving) is te beschrijven in SID als een serie statements waarbij wordt opgegeven of deze parallel dan wel sequentieel worden uitgevoerd. Tevens is het op dit niveau mogelijk om delays in te voeren met de after instructie. Binnen deze beschrijving mogen zowel logische als arithmetische operatoren gebruikt worden.

De "vertaalbare" SID beschrijving is simuleerbaar. We hebben op dit niveau logische functies, if-then-else en case statements en finite state machine beschrijvingen, als hoogste niveau beschrijvingen.

Tot slot is een beschrijving op gates niveau simuleerbaar.

Als we zoals bijvoorbeeld bij de ontworpen dma-controller vier functionele blokken onderscheiden na de eerste functionele decompositieslag, kunnen we elk blok in een verschillend stadium van ontwikkeling hebben en toch het geheel simuleren.

4.1 Het werk van de ontwerper

De functionele decompositie moet uitgevoerd worden door de ontwerper omdat hiervoor intelligentie vereist is, die niet door een computer kan worden overgenomen.

De opsplitsing in states waarin de acties achtereenvolgens dienen plaats te vinden is binnen ASA ook een taak voor de ontwerper evenals het ontwerpen van een datapad bestaande uit arithmetische en logische operatoren.

Vervolgens wordt een controller ontworpen die het datapad bestuurt. Waarbij de sequentie van handelingen wordt geregeld door een finite state machine.

Tot slot dient de ontwerper de arithmetische operatoren in het datapad zelf te ontwerpen.

4.2 Het werk van ASA

ASA vertaalt de vertaalbare SID beschrijving naar logische functies en state registers, die op hun beurt weer worden geimplementeerd als gates. Vervolgens kan ASA deze gates-beschrijving vertalen naar een layout met de kenmerken van de door de ontwerper gekozen technologie en foundry.

4.2.1 ASA ondersteunt

***Top down ontwerpen**

Elk niveau is simuleerbaar.

***Hierarchisch ontwerpen**

Men kan subsystemen ontwerpen.

***Synchroon ontwerpen**

Het is mogelijk controllers als finite state machines te beschrijven die alleen state transities uitvoeren op de opgaande flank van de klok.

***De beschrijving tot op gatesniveau is compleet technologieonafhankelijk.**

4.2.2 onderzoek naar layoutoptimalisatie

Er is tot slot gezocht naar een strategie om een beschrijving tot op gatesniveau te maken die een zo optimaal mogelijke layout aflevert met de ASA compiler.

We leveren dus een SID-tekst af die met onze voorkennis tot op dit niveau een zo optimaal mogelijke layout aflevert. Deze SID tekst is dan nl. nog compleet technologieonafhankelijk.

ASA ondersteunt drie manieren van implementatie van logica nl. de standardcell, pla en regelmatige structuren.

Vanuit het oogpunt van een systeemontwerper volgt hieronder een waardering voor deze ontwerpstijlen.

Standardcell

De standardcell is een ontwerpvorm die voor een systeemontwerper begrijpelijk moet zijn, hij kan bij een bepaald aantal basic cells voor een uitvoering als standardcell kiezen. Hij moet hierbij zorgen dat hij ongeveer even grote standardcellen krijgt door ongeveer evenveel basic cellen toe te laten in elke standardcell.

PLA

Een PLA is minder geschikt voor de systeemontwerper omdat hij niet weet bij hoeveel mintermen het qua oppervlaktewinst nuttig wordt om een stuk logica als PLA uit te voeren in plaats van als standardcell. Daarbij geeft ASA niet aan hoeveel mintermen het gebruikt voor de realisatie van een bepaald blok logica. Een oplossing zou kunnen zijn om ASA te laten melden wanneer het volgens hem nuttig is om een PLA te gebruiken om vervolgens de beslissing hierover aan de ontwerper te laten.

REGULAR (regelmatige structuren)

Voor het zelf maken van regelmatige structuren is IC ontwerpkennis nodig en is dus niet van toepassing voor een systeemontwerper. Er zijn echter een paar standaard regelmatige structuren zoals ROM en RAM die elke systeemontwerper hoort te kennen en die door de ASA bibliotheek worden meegeleverd. Een systeemontwerper zou echter wel met een logische beschrijving van een regelmatige structuur naar een IC-ontwerper kunnen gaan om hiervoor een regelmatige structuur te laten ontwerpen en deze vervolgens gebruiken.

5 ASA ONTWERPVOORBEELD: INCREMENTOR

Hier volgt een beschrijving van het ontwerptraject met ASA. Aan de hand van een incrementor, een simpele arithmetische operator, welke ik in het ontwerp van de dma-controller heb gebruikt in dma-channel blok zoals beschreven in hoofdstuk 9, zal ik laten zien hoe er ontworpen dient te worden tot op gatesniveau.

5.1 Gedragsbeschrijving

We beschrijven een system incrementor met als input A en als output Y. De breedte in bits van input en output geven we op met een parameter AB (AddressBits). Het functionele gedrag van een incrementor komt er op neer dat de input met 1 verhoogd wordt doorgegeven naar de output.

In SID komt bovenstaande beschrijving er dan als volgt uit te zien:

```
SYSTEM Incrementor;
PARAMETER
                AB;
CONTACT
A[0..AB-1]      :INPUT;
Y[0..AB-1]      :OUTPUT;
FUNCTION
Y = A + 1 AFTER 3;
```

Deze beschrijving kunnen we simuleren door een stimuli system in SID te beschrijven en deze twee systems in een hoger gelegen main system te connecteren en de parameterwaardes te definieren.

```
SYSTEM Stimuli;
PARAMETER
                AB;
CONTACT
A[0..AB-1]      :OUTPUT;
Y[0..AB-1]      :INPUT;
FUNCTION
A = 0 AFTER 0;
A = 3 AFTER 10;
WHEN Y->4 DO
BEGIN
A:=2 AFTER 20;
A:=3 AFTER 30;
END;
```

Voornoemde systems worden nu in Main gekoppeld zoals hieronder beschreven.

```
SYSTEM Main;
USE
IN              : Incrementor ( AB: = 4 );
STI             : Stimuli ( AB: = 4 );
CONNECT
IN.A = STI.A;
IN.Y = STI.Y;
```

Als we nu het bovenstaande system main simuleren met de functional simulator levert dat de resultaten op zoals weergegeven in figuur 5.1. We kunnen hierin zien dat ons uitgangssignaal Y telkens na een delay van 3 tijdseenheden reageert op een verandering van de input A. De uitgang wordt telkens een hoger als de input dus zoals we konden verwachten klopt het functionele gedrag.

5.2 Ontwerp van de arithmetische operator

Nu moeten we als ontwerper echter zelf deze arithmetische operator gaan ontwerpen omdat ASA niet in staat is dergelijke operatoren zelf te implementeren. We moeten dus nu een vertaalbare SID beschrijving zien te creeren.We hebben dit gedaan door gebruik te maken van het bit seriele principe om zodoende de parametriseerbaarheid op een simpele manier te handhaven. We hebben dus gekozen voor een zogenaamde ripple through incrementor.

Als we uitgaan van een eenbitsincrementor hebben we te maken met een Carry In en een Carry Out om koppeling van meerdere eenbitsincrementors mogelijk te maken.

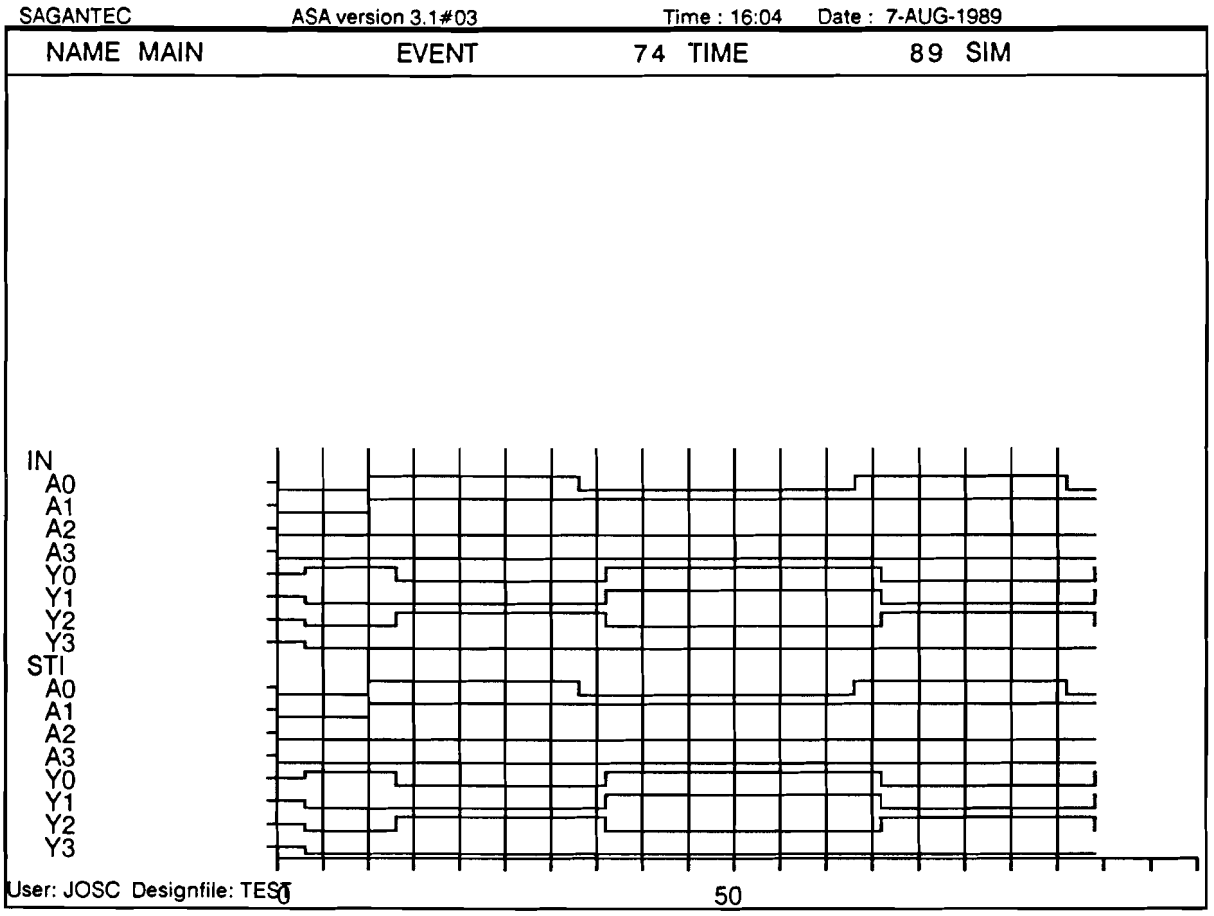
de functietabel van een bitslice van deze incrementor ziet er als volgt uit:

Inputs:		Outputs:	
A	CI	Y	CO
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

Dit resulteerd in de volgende functie voor Y en CO:

CO = A AND CI

Y = (NOT A AND CI) OR (A AND NOT CI)



Figuur 5.1 Resultaat van de functionele simulatie

We zouden de uitdrukkingen voor CO en Y ook als IF-THEN-ELSE of CASE statements kunnen beschrijven. En als derde optie zouden we nog relationele expressies kunnen gebruiken want die zijn ook vertaalbaar.

We hebben nu dus een system incrbit ontworpen:

```
SYSTEM Incrbit;
CONTACT
CI              :INPUT;
A               :INPUT;
CO              :OUTPUT;
Y               :OUTPUT;
FUNCTION
CO = A AND CI;
Y = (NOT A AND CI) OR (A AND NOT CI);
```

Het eerst bit van onze incrementor heeft echter geen CI dus schrijven we hiervoor een vereenvoudigd system Firstbit genaamd.

```
SYSTEM Firstbit;
CONTACT
A               :INPUT;
CO              :OUTPUT;
Y               :OUTPUT;
FUNCTION
CO = A;
Y = NOT A;
```

We kunnen nu voor dit ene bit weer een stimuli system schrijven en alle mogelijkheden uit de functietabel controleren maar dit is zo triviaal dat we nu direct overgaan tot het beschrijven van onze totale incrementor opgebouwd uit incrementorbits.

```
SYSTEM Incrementor;
PARAMETER
                AB;
CONTACT
A[0..AB-1]      :INPUT;
Y[0..AB-1]      :OUTPUT;
VAR
I[0..AB-1]      :MEMORY; {dit is een hulpvariable}
USE
INC[0..AB-1]    :Incrbit;
FB              :Firstbit;
CONNECT
A[0] = FB.A;
Y[0] = FB.Y;
INC[1].CI = FB.CO;
FOR I = 0 TO AB-2 DO
BEGIN
                A[I] = INC[I].A;
                Y[I] = INC[I].Y;
                INC[I + 1].CI = INC[I].CO;
END;
A[AB-1] = INC[AB-1].A;
Y[AB-1] = INC[AB-1].Y;
```

Hiermee hebben we een arithmetische operator ontworpen welke we opnieuw met dezelfde stimuli kunnen simuleren als welke we gebruikt hebben bij onze functionele beschrijving. ASA kan deze uitvoering vertalen naar een gates implementatie. Nu hebben we dus een incrementor, die parametriseerbaar is en functioneel gesimuleerd tot op het

niveau van gates. Voor elke verdere simulatie voor wat betreft timing gedrag zijn we afhankelijk van de te kiezen technologie en moeten we interfacen naar een ander pakket dan ASA.

6 ONTWERP VAN EEN DMA CONTROLLER MET DE ASA SILICON COMPILER

6.1 Beschrijving van de totale controller

De dma-controller, die ons voor ogen staat, is een component die gebruikt kan worden als een standaardcomponent op verschillende interfacechips. Omdat we de controller alleen wensen te gebruiken voor transfers van I/O naar MEMory of omgekeerd, is het een besturingsorgaan dat de verstuurde data niet intern opslaat.

Algemene beschrijving

De controller kan in een idle toestand of in een operational toestand verkeren. In de idle toestand is het mogelijk voor de CPU om de interne registers te lezen en te beschrijven. De controller gaat over in de operational toestand op het moment dat hij een ongemaskeerde aanvraag voor een dma transfer krijgt. Nu wordt er via een HOLD signaal naar de CPU de beschikking over de bussen aangevraagd. De registers kunnen dan nog gelezen en beschreven worden door de CPU totdat deze met een HoLDAcknowledge aangeeft dat hij de bussen afstaat aan de aanvrager. Vervolgens kan de eigenlijke transfer beginnen.

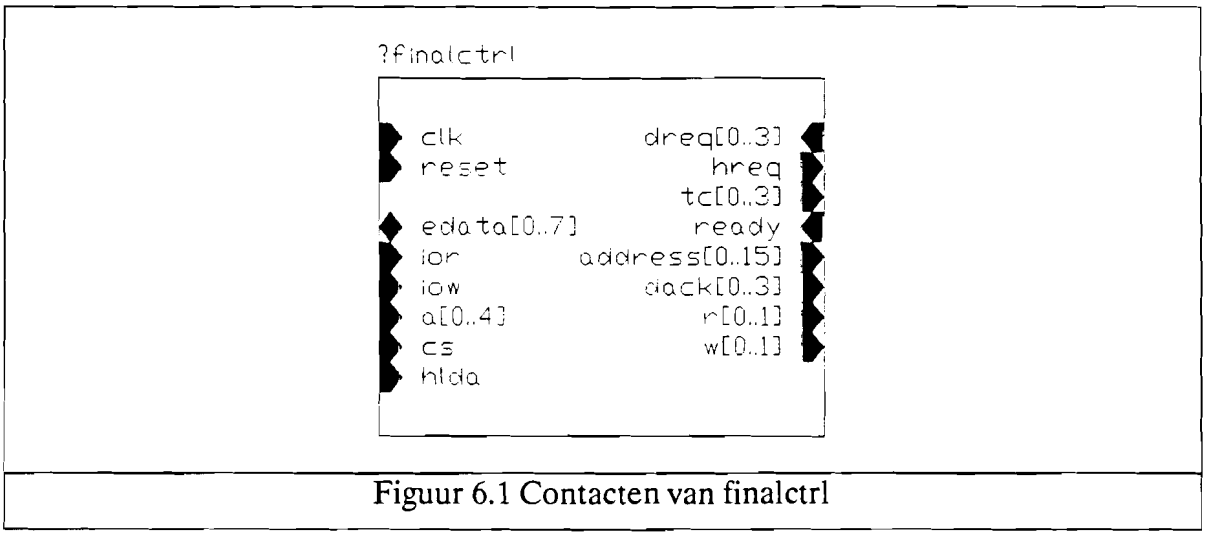
We gaan uit van een vierkanaals controller welke voor elk kanaal gebruik maakt van een adresregister. Het adresregister geeft het adres in een blok aan waarop de volgende transfer betrekking heeft.

Elk kanaal maakt tevens gebruik van een teller. De teller geeft aan hoeveel transfers er nog uitgevoerd moeten worden.

Verder bezitten de kanalen elk een mode register waarmee drie verschillende modes te weten Read, Write of Verify aangegeven kunnen worden. Tevens zijn er een maskbit en een statusbit ondergebracht in dit register. Hun functies zijn resp. het maskeren van een kanaal voor aanvragen voor dma door een randapparaat en het aangeven of een transferblok geheel afgehandeld is of niet.

Verder gaan we uit van een prioriteitencode, die aangeeft of we werken met rotating of met fixed priority en welk kanaal initieel de hoogste prioriteit heeft.

In de volgende paragraaf zullen alle externe signaallijnen van de controller beschreven worden.



Figuur 6.1 Contacten van finalctrl

Contactpennen

EDATA[0..DB-1] :BIDIRECTIONAL;
De externe databus voor het inlezen en uitlezen van de data van de registers in de controller.

IOR :INPUT;
Het read-signaal waarmee de CPU kan aangeven dat er data uitgelezen dient te worden.

IOW :INPUT;
Het write-signaal waarmee de CPU kan aangeven dat er data overgenomen dient te worden.

A[0..4] :INPUT;
De vijf adreslijnen waarmee de verschillende registers en evt. de hoge of lage byte geadresseerd kunnen worden. A[0] = 0 levert de lage byte, en moet ook altijd laag zijn bij registers die maar uit een byte bestaan. A[1..2] selecteren van een bepaalde registersoort het juiste kanaal. A[3..4] selecteren de registersoort Address,Counter,Mode of Priority, resp. voor A[3..4] is 00,01,10,11.

CS :INPUT;
Het Chip Select signaal, dat aangeeft dat de dma-controller geselecteerd is door de CPU om als slave, dus voor het lezen of schrijven van registers, gebruikt te worden.

HLDA :INPUT;
Het Holdacknowledge signaal, dat als reactie op een Holdrequest van de controller, aangeeft dat de controller nu de master is over de systeem- bussen.

DREQ[0..3] :INPUT;
De DmaREQuest lijnen waarop de aanvragen voor dma van de periferals binnen- komen, voor een bepaald kanaal.

HREQ :OUTPUT;
Het HoldREQuest signaal waarmee de controller de CPU verzoekt om de bussen af te staan voor dmatransfers.

TC[0..3] :OUTPUT;
Termination Count, hiermee geeft de controller aan de buitenwereld aan dat een blok data ter grootte van termination counter waarde is verwerkt en dat hij dus gestopt is om transfers via dit kanaal uit te voeren.

READY :INPUT;
Het ready signaal is een door de periferie verstuurd signaal om aan te geven dat een datatransfer kan worden uitgevoerd omdat de periferal er klaar voor is.

ADDRESS[0..AB-1] :OUTPUT;
Dit is de adresbus waarop de door de controller gegenereerde adressen naar buiten worden gestuurd, deze outputs zijn altijd in tristate behalve wanneer er dma transfers worden uitgevoerd.

DACK[0..3] :OUTPUT;
Het DmaACKnowledge signaal voor de verschillende kanalen, dit signaal geeft aan een periferal welke om dma gevraagd heeft met een DREQ door dat zijn aanvraag geserviced wordt.

R[0..1] :OUTPUT;
W[0..1] :OUTPUT;
W[0] en R[0] zijn de write en read controllijnen die in geval van write-cycle de dmatransfer regelen.(MEMW IOR)

W[1] en R[1] zijn de write en read controllijnen die in geval van write-cycle de dmatransfer regelen.(IOW MEMR)

6.2 Afspraken omtrent adressering en specificatie van registers

Hieronder volgt een overzicht van de gebruikte registers met de daarbij behorende specificaties ten behoeve van de adressering en hun functies binnen de controller.

-adressering

A[0]	0	Low byte selected.
	1	High byte selected.
A[1..2]	00	kanaal 0 selected.
	10	kanaal 1 selected.
	01	kanaal 2 selected.
	11	kanaal 3 selected.
A[3..4]	00	Addressregister.
	10	Termination counter register.
	01	Mode register.
	11	Priority code register.

-specificatie

Er zijn vier addressregisters voor elk kanaal welke het adres bevatten voor de volgende uit te voeren transfer van dat kanaal.

Er zijn vier termination counter registers welke het aantal nog uit te voeren transfers voor het betreffende kanaal aangeven.

Er zijn vier moderegisters met daarin opgeslagen de mode voor het betreffende kanaal, de status van het kanaal en het maskeerbit voor het kanaal.

	MODE[1..0]		MASK	STATUS
bit	3	2	1	0

Als het maskbit 1 is worden er door dat kanaal geen transfers uitgevoerd. Als het statusbit 1 is, is het termination count uitgangssignaal hoog, dit bit wordt 1 als de termination counter van dit kanaal na een transfer de waarde 0 bereikt.

MODE
= 0 of 3:
dan is dit kanaal in verify mode en worden er wel adressen en DACK signalen verzonden maar geen controlsignalen voor lezen of schrijven.
= 1:
dan is het kanaal in read mode en worden er dus memory to io transfers uitgevoerd MEMR, IOW.
= 2:
dan is het kanaal in write mode en worden er dus io to memory transfers uitgevoerd IOR, MEMW.

Verder hebben we een priority code register, waarin is opgeslagen welk kanaal bij de volgende transfer de hoogste prioriteit heeft en of we te maken hebben met rotating dan wel fixed priority.

ROTATING	SELECT[1..0]	
bit2	bit1	bit0

Als rotating hoog is wordt na elke uitgevoerde dmaslag aan select de waarde van het vorige geselecteerde kanaal + 1 gegeven. Als rotating echter 0 is blijft de waarde van select onveranderd.

6.3 Blokken na een functionele decompositieslag

We begonnen met het maken van een functionele beschrijving op een zo hoog mogelijk niveau. Dit heeft geleid tot een decompositie van de totale controller in vier functionele blokken, te weten:

- **Slave Businterface (SYSTEM SBI):**
  kijkt of de controller door de cpu wordt geselecteerd en selecteert interne registers voor lezen of schrijven.
- **Priority Control (SYSTEM PRIOR_ENC):**
  detecteert dma requests, vraagt bussen aan en start het juiste kanaal op.

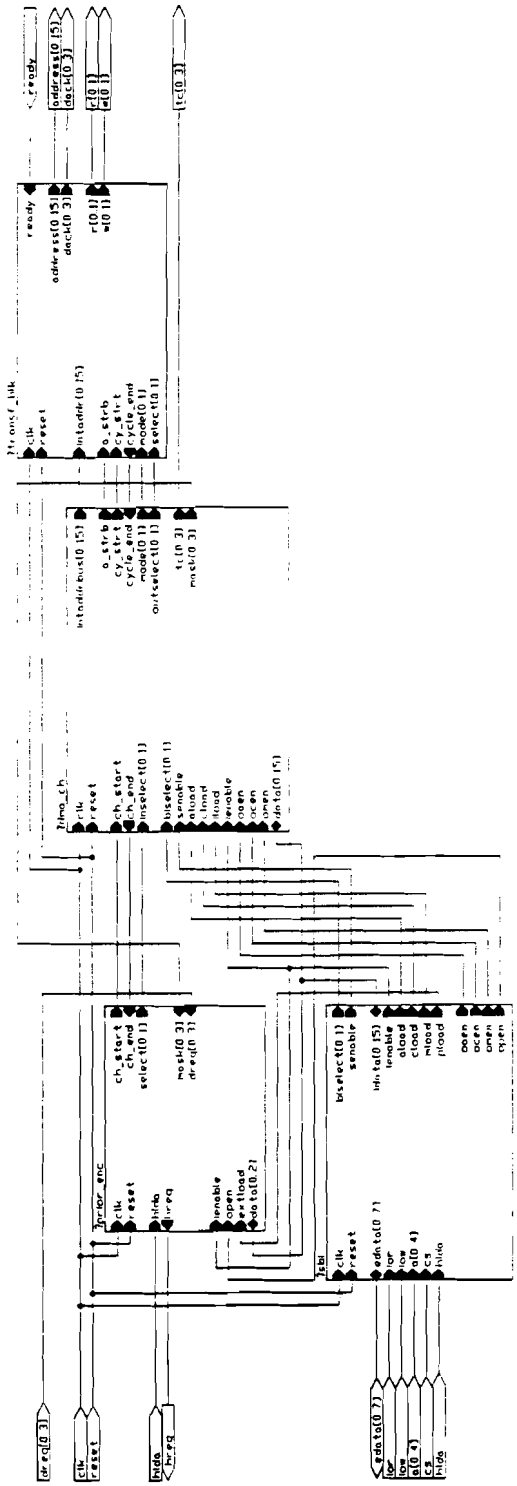
- **Dma Channel (SYSTEM DMA_CH):**
  bevat alle informatie van elk kanaal in een registerblok en houdt deze informatie up to date (incrementeert adresregister, decrementeert counter etc.).

- **Transfer Control (SYSTEM TRANSF_BLK):**
  regelt de gehele dma cyclus, een adres wordt op de bussen gezet en een peripheral wordt geacknowledged. Tevens worden de benodigde read- en writesignalen gegenereerd, en wordt de ready timing geregeld.

Deze blokken zijn vervolgens aan een volgende decompositie onderworpen, welke meer op het architectuurvlak gezien moet worden. Deze decompositie naar structuur was mogelijk omdat een verdere functionele opsplitsing ons weinig zinvol leek, de vier gekozen blokken waren met hun beschrijving reeds zo elementair dat hieruit best de architectuur te bepalen viel. Dit heeft geleid tot een opsplitsing van elk blok in een controller welke de sequentie van de operaties regelt en een of meer data(path)- blokken die de registers bevatten en de eigenlijke bewerkingen uitvoeren. Hieronder zien we de SID beschrijving van de interconnecties en uses van de controller (SYSTEM Finalctrl), waarvan op de volgende bladzijde, en uitklapbaar na hfst. 10 het schema is weergegeven.

```
SYSTEM Finalctrl;
PARAMETER
AB, DB;

CONTACT
CLK                 :INPUT;
RESET               :INPUT;

EDATA[0..DB-1]      :BIDIRECTIONAL;
IOR                 :INPUT;
IOW                 :INPUT;
A[0..4]             :INPUT;
CS                  :INPUT;

HLDA                :INPUT;
DREQ[0..3]          :INPUT;
HREQ                :OUTPUT;

TC[0..3]            :OUTPUT;

READY               :INPUT;
ADDRESS[0..AB-1]:OUTPUT;
DACK[0..3]          :OUTPUT;
R[0..1]             :OUTPUT;
W[0..1]             :OUTPUT;

USE
SBI                 :SBI(AB: = AB,DB: = DB);
PRI                 :PRIOR_ENC;
DMA                 :DMA_CH(AB: = AB);
TRA                 :TRANSF_BLK(AB: = AB);

CONNECT
CLK = SBI.CLK = PRI.CLK = DMA.CLK = TRA.CLK;
RESET = SBI.RESET = PRI.RESET = DMA.RESET
= TRA.RESET;

EDATA = SBI.EDATA;
IOR = SBI.IOR;
IOW = SBI.IOW;
A = SBI.A;
CS = SBI.CS;

HLDA = SBI.HLDA = PRI.HLDA;
DREQ = PRI.DREQ;
HREQ = PRI.HREQ;

TC = DMA.TC;

READY = TRA.READY;
ADDRESS = TRA.ADDRESS;
DACK = TRA.DACK;
R = TRA.R;
W = TRA.W;

SBI.IENABLE = PRI.IENABLE = DMA.IENABLE;
SBI.IDATA[0..2] = PRI.DATA[0..2];
SBI.IDATA = DMA.DATA;
SBI.ALOAD = DMA.ALOAD;
SBI.CLOAD = DMA.CLOAD;
SBI.MLOAD = DMA.ILOAD;
SBI.PLOAD = PRI.EXTLOAD;
SBI.OAEN = DMA.OAEN;
SBI.OCEN = DMA.OCEN;
SBI.OMEN = DMA.OMEN;
SBI.OPEN = PRI.OPEN;
SBI.BISELECT = DMA.BISELECT;
SBI.SENABLE = DMA.SENABLE;

PRI.MASK = DMA.MASK;
PRI.CH_START = DMA.CH_START;
PRI.CH_END = DMA.CH_END;
PRI.SELECT = DMA.INSELECT;

DMA.CYCLE_END = TRA.CYCLE_END;
DMA.INTADDRBUS = TRA.INTADDR;
DMA.CY_STRT = TRA.CY_STRT;
DMA.A_STRB = TRA.A_STRB;
DMA.MODE = TRA.MODE;
DMA.OUTSELECT = TRA.SELECT;
```

Listing 6.1 SID-beschrijving van Finalctrl



figuur 6.2 Finalctrl na eerste decompositie

6.4 Architectuurafspraken na functionele decompositie

Functionele grenzen

De grenzen van functionele blokken na de functionele decompositie kunnen in de structuur voor vervelende complicaties zorgen. Bv. afzonderlijke bussen vanuit elk datablok naar de businterface.

Daarom dienen er na het uitwerken van de functionele blokken structurele afspraken gemaakt te worden. De belangrijkste afweging is wel dat we rekening moeten houden met de oppervlakteverkwisting door het leggen van bussen tussen de hoofdblokken, omdat de logische componenten die elkaar opvolgen in verschillende blokken liggen. We hebben dus zo veel mogelijk een bundeling van logische componenten, die samen een architecturele eenheid vormen, nagestreefd.

Afspraken:

-We hebben voor een zo onafhankelijk mogelijke slave businterface gekozen. Onafhankelijk in de betekenis van gemakkelijk aan te passen voor verschillende besturingen van buitenaf.

In slave mode haalt de interface altijd de gehele registerinhoud van het geselecteerde register op, waardoor het mogelijk is de betreffende byte te overschrijven dan wel uit te lezen. In geval van overschrijven wordt het gehele register na afloop van de schrijfactie teruggeplaatst.

We hebben voor het transport van interne data van en naar de vier hoofdblokken voor een bidirectionele databus gekozen.

Layoutoverwegingen

Als je met ASA stijf vasthoudt aan functionele blokken krijg je "meestal" geen optimale layouts omdat er geoptimaliseerd wordt per SYSTEM (blok). M.a.w. de hierarchie van het ontwerp wordt aangehouden bij het maken van de layout, er wordt dus van het laagste naar het hoogste niveau gelayout zonder bij de "compactie" de randen van hoger gelegen slices weg te nemen.

7 SLAVE BUSINTERFACE

7.1 Functionele beschrijving

Dit blok wordt alleen gebruikt in de programming mode, dat wil zeggen dat de controller door de CPU geselecteerd wordt om te lezen of te schrijven van of naar de registers binnen de controller.

Dit blok wordt alleen geactiveerd als het geselecteerd wordt en HLDA is inactief. De extra toevoeging dat ook HLDA inactief moet zijn is nodig omdat anders misschien een door de controller verzonden adres de controller zelf zou kunnen adresseren.

Hieronder volgt een pseudo-pascal beschrijving van het uit te voeren proces en een beschrijving van de contacten zoals deze zijn weergegeven in de uitklapbare figuur aan het eind van hoofdstuk 10.

Procesbeschrijving in pseudo-pascal:

```
repeat
if (HLDA = 0 and dma controller selected)
then            select (register);
                {de CPU leest of programmeert nu, dus alle transfer control outputs moeten tristate zijn}
case IOW        :enable write buffer:
                write data to buffer;
case IOR        :enable read buffer;
                read data from register;
until forever;
```

Contactpennen:

IOR :INPUT;
Het read_signaal waarmee de CPU kan aangeven dat er data uitgelezen dient te worden.

IOW :INPUT;
Het write-signaal waarmee de CPU kan aangeven dat er data overgenomen dient te worden.

A[0..4] :INPUT;
De vijf adreslijnen waarmee de verschillende registers en evt. de hoge of lage byte geadresseerd kunnen worden. A[0] = 0 levert de lage byte, en moet ook altijd laag zijn bij registers die maar uit een byte bestaan. A[1..2] selecteren van een bepaalde registersoort het juiste kanaal. A[3..4] selecteren de registersoort Address, Counter, Mode of Priority, resp. voor A[3..4] is 00,01,10,11.

CS :INPUT;
Het Chip Select signaal, dat aangeeft dat de dma-controller geselecteerd is door de CPU om als slave, dus voor het lezen of schrijven van registers, gebruikt te worden.

HLDA :INPUT;
Het HoLDAcknowledge signaal, dat als reactie op een holdrequest van de controller, aangeeft dat de controller nu de master over de systeembussen is.

EDATA[0..DB-1] :BIDIRECTIONAL;
De externe databus voor het inlezen en uitlezen van data van de registers in de controller.

IDATA[0..AB-1] :BIDIRECTIONAL;
De interne databus voor het ophalen en beschrijven van de interne registers.

IENABLE :OUTPUT;
Het Input ENABLE signaal geeft aan dat er data op de interne databus wordt aangeboden ter overname.

ALOAD :OUTPUT;
Het Address LOAD signaal geeft aan dat er een adresregister geladen dient te worden met de data op de interne databus.

CLOAD :OUTPUT;
Idem voor een termination Counter register.

MLOAD :OUTPUT;
Idem voor een Mode register.

PLOAD :OUTPUT;
Idem voor een Priority code register.

OAEN :OUTPUT;
Het Output Address ENable signaal geeft aan dat de data van een intern adresregister naar de interne databus wordt doorgelaten.

OCEN :OUTPUT;
Idem voor een termination Counter register.

OMEN :OUTPUT;
Idem voor een Mode register.

OPEN :OUTPUT;
Idem voor een Priority code register.

BISELECT[0..1] :OUTPUT;
Het BusInteface SELECT signaal geeft aan welk van de vier kanalen er geselecteerd dient te worden.

SENABLE :OUTPUT;
Het SelectENABLE signaal geeft door dat de kanaalselectiesignalen van de slave businterface de de hoogste prioriteit hebben.

7.2 Decompositie naar datapad en controller

We hebben vervolgens een decompositie uitgevoerd naar een datapad en een controller om dit datapad te besturen. Beiden hebben we beschreven in vertaalbaar SID dus met gebruikmaking van parametriseerbare generische bouwstenen uit de ASA bibliotheek en de met FSM vertaalbare beschrijving van de finite state machine die de controller vormt.

7.2.1 datapadbeschrijving

Het system DATAREG bevat het datapad van de slave businterface. Er volgt nu een algemene beschrijving van dit datapad en een beschrijving van de externe contacten van dit blok.

Dit blok bevat een register ter breedte van het aantal gebruikte addressbits, in ons geval 16. Dit blok moet dus enerzijds 16 bits kunnen ontvangen en versturen over de interne databus, en anderzijds de High of de Low byte van dit register op of van de externe databus kunnen zetten resp. halen. Tevens moet dit blok aan de hand van twee adreslijnen het juiste kanaal selecteren, en aan de hand van twee andere adreslijnen het juiste type register selecteren. Er zijn vier typen registers aanwezig:

addressregisters, termination countregisters, moderegisters, prioritycoderegister.

Aan dit geselecteerde register moet dan een load of een output- enable signaal worden doorgegeven.

In figuur 7.1 zijn de contacten en interconnecties weergegeven van het datapad (datareg) en de controller (datactrl) waaruit de slave businterface is opgebouwd. De beschrijving van deze contacten en interconnecties in SID zijn te vinden in listing 7.1.

contactpennen:

EXTDATA[0..DB-1] :BIDIRECTIONAL;
Dit zijn de externe datalijnen.

INTDATA[0..AB-1] :BIDIRECTIONAL;
Dit zijn de interne datalijnen voor het transport van data van en naar de interne registers.

LEN :INPUT;
Het Low ENable signaal zorgt ervoor dat de data die in het Low REGister staat wordt doorgelaten naar de externe databus.

HEN :INPUT;
Het High ENable signaal zorgt ervoor dat de data die in het High REGister staat wordt doorgelaten naar de externe databus.

WEN :INPUT;
De Write ENable zorgt ervoor dat de op de externe databus aangeboden data wordt doorgelaten naar het hoge en het lage register.

LLD :INPUT;
Het Low LoaD signaal zorgt ervoor dat de aan het lage register aangeboden data wordt opgeslagen.

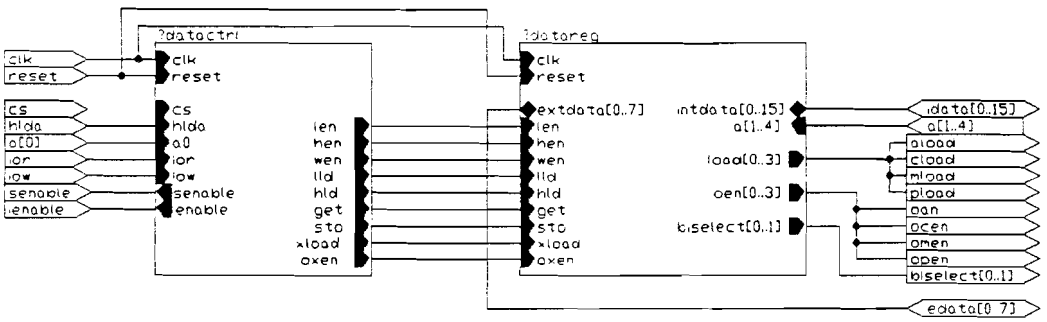
HLD :INPUT;
Het High LoaD signaal zorgt ervoor dat de aan het hoge register aangeboden data wordt opgeslagen.

GET :INPUT;
Het GET signaal zorgt ervoor dat de data die op de interne databus staat wordt doorgelaten naar de registers.

STO :INPUT;
Het STOre signaal zorgt ervoor dat de inhoud van de registers wordt doorgelaten naar de interne databus.

A[1..4] :INPUT;
A[1..2] zijn rechtstreeks te gebruiken als de biselectlijnen ter selectie van het juiste kanaal.

A[3..4] worden gebruikt om het juiste soort register te selecteren.



Figuur 7.1 sbi

```
SYSTEM Sbi;
PARAMETER
DB,AB;

CONTACT
CLK             :INPUT;
RESET           :INPUT;
EDATA[0..DB-1]  :BIDIRECTIONAL;
IDATA[0..AB-1]  :BIDIRECTIONAL;
IOR             :INPUT;
IOW             :INPUT;
A[0..4]         :INPUT;
CS              :INPUT;
HLDA            :INPUT;

IENABLE         :OUTPUT;
ALOAD           :OUTPUT;
CLOAD           :OUTPUT;
MLOAD           :OUTPUT;
PLOAD           :OUTPUT;
OAEN            :OUTPUT;
OCEN            :OUTPUT;
OMEN            :OUTPUT;
OPEN            :OUTPUT;
BISELECT[0..1]:OUTPUT;
SENABLE         :OUTPUT;

USE
DR              :DATAREG(DB: = DB,AB: = AB);
DC              :DATACTRL;

CONNECT
CLK = DR.CLK = DC.CLK;
RESET = DR.RESET = DC.RESET;
EDATA = DR.EXTDATA;
IDATA = DR.INTDATA;
IOR = DC.IOR;
IOW = DC.IOW;
A[0] = DC.A0;
A[1..4] = DR.A[1..4];
CS = DC.CS;
HLDA = DC.HLDA;

IENABLE = DC.IENABLE;
ALOAD = DR.LOAD[0];
CLOAD = DR.LOAD[1];
MLOAD = DR.LOAD[2];
PLOAD = DR.LOAD[3];
OAEN = DR.OEN[0];
OCEN = DR.OEN[1];
OMEN = DR.OEN[2];
OPEN = DR.OEN[3];
BISELECT = DR.BISELECT;
SENABLE = DC.SENABLE;

DR.LEN = DC.LEN;
DR.HEN = DC.HEN;
DR.WEN = DC.WEN;
DR.LLD = DC.LLD;
DR.HLD = DC.HLD;
DR.GET = DC.GET;
DR.STO = DC.STO;
DR.XLOAD = DC.XLOAD;
DR.OXEN = DC.OXEN;
```

Listing 7.1 SID-beschrijving van SBI

XLOAD :INPUT;
Het load signaal dat aangeeft dat er een intern register data dient over te nemen.

OXEN :INPUT;
Het output enable signaal dat aangeeft dat er een intern register data op de interne databus dient aan te bieden.

LOAD[0..3] :OUTPUT;
De load signalen die naar de interne registers gaan.

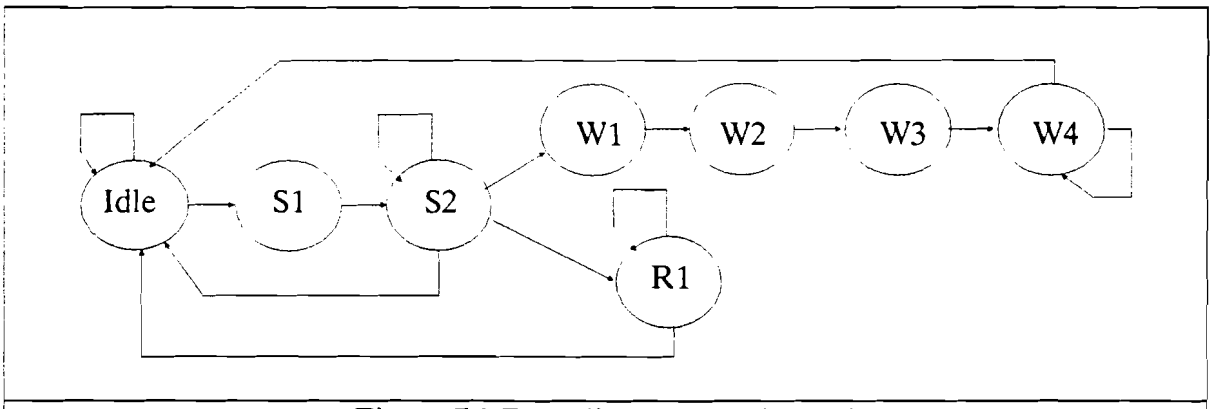
OEN[0..3] :OUTPUT;
De output enable signalen die de data van de juiste soort interne registers doorlaten naar de interne databus.

BISELECT[0..1] :OUTPUT;
De selectlijnen die het juiste kanaal selecteren.

7.2.2 finite state machinebeschrijving van de controller

De beschrijving van de controller gaat volgens onderstaand protocol:

States: NEXT STATE OVERGANGEN;
Beschrijving van de signalen en de uit te voeren acties binnen een bepaalde state.



Figuur 7.2 Flow diagram van datactrl

Idle: IF NOT CS AND NOT HLDA THEN GOTO S1;

Als er een CS (active low) aanwezig is en tevens geen HLDA (om te voorkomen dat de CS geactiveerd wordt door een door de controller verzonden adres) dan treedt de slave businterface in werking.

S1: GOTO S2;

Selecteer het kanaal (SENABLE); output enable het soort register (OXEN); zet de tristatepoorten voor het ophalen van interne data open (GET); laad beide bytes van het interne register met de data op de interne bus (HLD,LLD).

S2: IF NOT IOW THEN GOTO W1
ELSE IF NOT IOR THEN GOTO R1
ELSE IF CS THEN GOTO IDLE
ELSE GOTO S2;

IOR en IOW zijn de van de CPU afkomstige signalen en wij veronderstellen deze laag actief. Er wordt in deze state dus gewacht totdat er een lees of schrijf opdracht van de CPU komt en vervolgens wordt dan de lees of schrijf routine uitgevoerd.

W1: GOTO W2;

Als het writesignaal van de CPU komt is de in te lezen data aanwezig op de externe bus. Nu kan dus het hoge of lage byte van het interne registers overschreven worden. Write enable zet de externe tristatepoort open (WEN); geef afhankelijk A[0] HLD of LLD voor het laden van de betreffende byte.

W2: GOTO W3;

Dit is een dummy state die is tussengevoegd om te zorgen dat alle tristate buffers zeker tristated zijn voordat wordt doorgegaan.

W3: GOTO W4;

Enable de input van data vanuit de slave businterface voor alle andere subblokken (IENABLE). Zet de tristate output poort naar de interne databus open (STO). Geef het loadsignaal waardoor alle geupdate data worden teruggezet (XLOAD).

W4: IF NOT CS THEN GOTO W4
ELSE GOTO IDLE;

Zet alles uit en wacht op het afvallen van de CS. We wachten op het afvallen van de CS omdat we anders misschien twee keer hetzelfde uitvoeren, omdat alle signalen vanuit de CPU langer zouden kunnen duren.

R1: IF NOT CS THEN GOTO R1
ELSE GOTO IDLE;

Laat het lage of hoge byte van het opgehaalde register door naar buiten zolang de CS actief is en vervolgens naar de Idle toestand (LEN of HEN). LEN en HEN vallen direct met de CS af om zodoende buscontentie op de externe bussen te voorkomen.

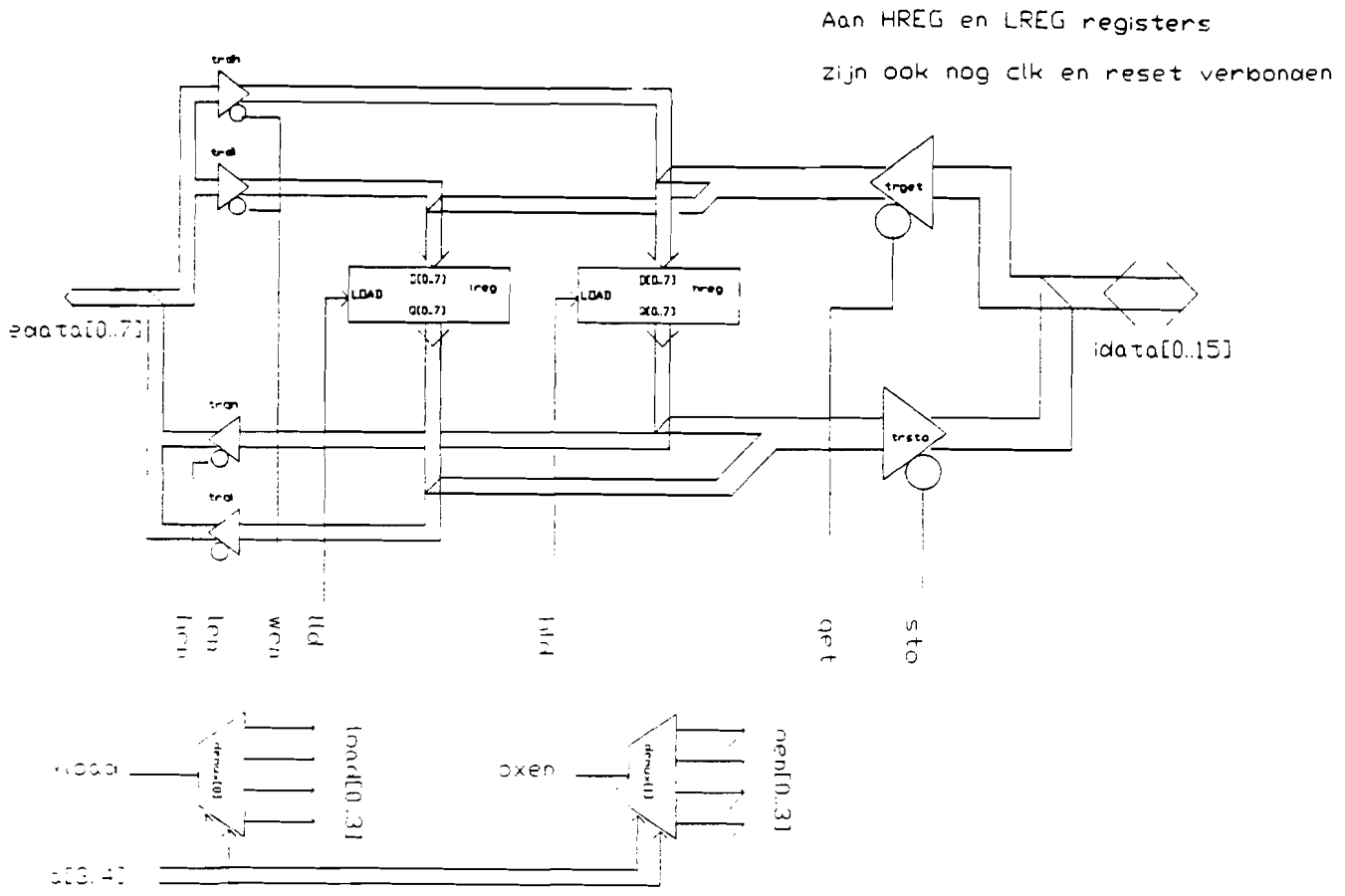
```
SYSTEM Datactrl;
CONTACT
CLK             :INPUT;
RESET           :INPUT;
CS              :INPUT;
HLDA            :INPUT;
A0              :INPUT;
IOR             :INPUT;
IOW             :INPUT;

SENABLE         :OUTPUT;
OXEN            :OUTPUT;
GET             :OUTPUT;
HLD             :OUTPUT;
LLD             :OUTPUT;
WEN             :OUTPUT;
IENABLE         :OUTPUT;
STO             :OUTPUT;
XLOAD           :OUTPUT;
HEN             :OUTPUT;
LEN             :OUTPUT;

FUNCTION
TRIGGER CLK-> 1

STATE IDLE:
SENABLE         = 0;
OXEN            = 0;
GET             = 0;
HLD             = 0;
LLD             = 0;
WEN             = 0;
IENABLE         = 0;
STO             = 0;
XLOAD           = 0;
HEN             = 0;
LEN             = 0;

IF NOT CS AND NOT HLDA
THEN GOTO S1;

STATE S1:
SENABLE         = 1;
OXEN            = 1;
GET             = 1;
HLD             = 1;
LLD             = 1;
WEN             = 0;
IENABLE         = 0;
STO             = 0;
XLOAD           = 0;
HEN             = 0;
LEN             = 0;

GOTO S2;

STATE S2:
SENABLE         = 1;
OXEN            = 0;
GET             = 0;
HLD             = 0;
LLD             = 0;
WEN             = 0;
IENABLE         = 0;
STO             = 0;
XLOAD           = 0;
HEN             = 0;
LEN             = 0;

IF NOT IOW THEN GOTO W1
ELSE
IF NOT IOR THEN GOTO R1
ELSE
IF CS THEN GOTO IDLE
ELSE GOTO S2;

STATE W1:
SENABLE         = 1;
OXEN            = 0;
GET             = 0;
HLD             = A0;
LLD             = NOT A0;
WEN             = 1;
IENABLE         = 0;
STO             = 0;
XLOAD           = 0;
HEN             = 0;
LEN             = 0;

GOTO W2;

STATE W2:
SENABLE         = 1;
OXEN            = 0;
GET             = 0;
HLD             = 0;
LLD             = 0;
WEN             = 0;
IENABLE         = 0;
STO             = 0;
XLOAD           = 0;
HEN             = 0;
LEN             = 0;

GOTO W3;

STATE W3:
SENABLE         = 1;
OXEN            = 0;
GET             = 0;
HLD             = 0;
LLD             = 0;
WEN             = 0;
IENABLE         = 1;
STO             = 1;
XLOAD           = 1;
HEN             = 0;
LEN             = 0;

GOTO W4;

STATE W4:
SENABLE         = 0;
OXEN            = 0;
GET             = 0;
HLD             = 0;
LLD             = 0;
WEN             = 0;
IENABLE         = 0;
STO             = 0;
XLOAD           = 0;
HEN             = 0;
LEN             = 0;

IF NOT CS THEN GOTO W4
ELSE GOTO IDLE;

STATE R1
SENABLE         = 0;
OXEN            = 0;
GET             = 0;
HLD             = 0;
LLD             = 0;
WEN             = 0;
IENABLE         = 0;
STO             = 0;
XLOAD           = 0;
HEN = A0 AND NOT CS;
LEN = NOT A0 AND NOT CS;

IF NOT CS THEN GOTO R1
ELSE GOTO IDLE;
```

Listing 7.2 SID-beschrijving van datactrl (FSM)

Het flow diagram is weergegeven in bovenstaande figuur 7.2, voor de complete SID-beschrijving van de controller zie listing 7.2. Hierin zijn precies alle outputs per state weergegeven.

7.3 Hierarchische beschrijving van het datapad

Het datapad is opgebouwd uit generische cellen welke allen terug te vinden zijn in de GENERIC CELL REFERENCE MANUAL [lit.14]. Deze generische cellen zijn op hun beurt weer opgebouwd uit de basic cellen waarvan er binnen ASA 11 in gebruik zijn.



Figuur 7.3 datareg

De generische cellen zijn parametriseerbaar en worden door ASA opgebouwd uit de basic cellen, wij hoeven dus alleen bouwstenen te beschrijven opgebouwd uit de generische cellen. Deze generische cellen zijn dus in feite onze basisbouwstenen.

Het datapad genaamd datareg, weergegeven in figuur 7.3, en de bijbehorende SID-listing, listing 7.3,vormen het laagste hierarchische beschrijvingsniveau. Met deze SID-beschrijving zijn we op het laagste beschrijvingsniveau aangeland en deze beschrijving kan dus door ASA rechtstreeks omgezet worden in een layout, waarbij ASA elke cell uit het use-blok dat als een generic beschreven is als een leaf module beschouwt. En als we dus geen standardcell attributes toevoegen op een hoger hierarchisch niveau maakt ASA van deze generische blokken standardcellen.

```
SYSTEM Datareg;
PARAMETER
AB,DB;

CONTACT
CLK                 :INPUT;
RESET               :INPUT;
EXTDATA[0..DB-1]:BIDIRECTIONAL;
INTDATA[0..AB-1]:BIDIRECTIONAL;
LEN                 :INPUT;
HEN                 :INPUT;
WEN                 :INPUT;
LLD                 :INPUT;
HLD                 :INPUT;
GET                 :INPUT;
STO                 :INPUT;
A[1..4]             :INPUT;
XLOAD               :INPUT;
OXEN                :INPUT;
LOAD[0..3]          :OUTPUT;
OEN[0..3]           :OUTPUT;
BISELECT[0..1]:OUTPUT;

USE
HREG,LREG       :GENERIC S$D_REG(
                                WORDLENGTH: = DB,
                                RESETABLE: = TRUE);
TRDL,TRDH,
TRQL,TRQH       :GENERIC S$TRBUSBUF(
                                WORDLENGTH: = DB,
                                DRIVE: = 1
                                ,INVERT: = FALSE,
                                NOTINVERT: = TRUE);

TRGET,TRSTO     :GENERIC S$TRBUSBUF(
                                WORDLENGTH: = AB,
                                DRIVE: = 1,
                                INVERT: = FALSE,
                                NOTINVERT: = TRUE);

DEMUX[0..1]     :GENERIC S$DEMULTI(WIDTH: = 4);

CONNECT
CLK = HREG.CK = LREG.CK;
RESET = HREG.RESET = LREG.RESET;
EXTDATA = TRDL.I = TRDH.I = TRQL.Y = TRQH.Y;
INTDATA = TRGET.I = TRSTO.Y;
HREG.D = TRDH.Y = TRGET.Y[DB..AB-1];
LREG.D = TRDL.Y = TRGET.Y[0..DB-1];
HREG.Q = TRQH.I = TRSTO.I[DB..AB-1];
LREG.Q = TRQL.I = TRSTO.I[0..DB-1];
HREG.LOAD = HLD;
LREG.LOAD = LLD;
TRDL.EN = TRDH.EN = WEN;
TRQL.EN = LEN;
TRQH.EN = HEN;
TRGET.EN = GET;
TRSTO.EN = STO;
BISELECT = A[1..2];
DEMUX[0].S = DEMUX[1].S = A[3..4];
DEMUX[0].I = XLOAD;
DEMUX[0].Y = LOAD[0..3];
DEMUX[1].I = OXEN;
DEMUX[1].Y = OEN[0..3];
```

Listing 7.3 SID-beschrijving datareg

8 PRIORITY ENCODER

8.1 Functionele beschrijving

Dit blok heeft de taak te kijken of er dma-requests zijn. En om na te gaan of deze geserviced kunnen/mogen worden. Dus kijken of het aangevraagde kanaal gemaskeerd is of niet. Dit blok vraagt de bussen aan bij de CPU en start het totale dma-proces op als de bussen zijn vrijgegeven.

De priority encoder zoekt in geval van gelijktijdige requests ook uit welke request de hoogste prioriteit heeft. Dit wordt gedaan aan de hand van het tijdens de initialisatie geladen priority code register.

De priority encoder blijft de bussen aangevraagd houden zolang er nog een ongemaskeerde request aanwezig is. Een dma request met een hogere prioriteit interumpeert elk actief kanaal met een lagere prioriteit na beeindiging van de dma-slag waarmee het bezig is.

Procesbeschrijving in pseudo-pascal:

```
HREQ: = 0; channel start: = 0;
repeat
                wait for ((not masked) and DREQ)  0);
                HREQ: = 1;
                wait for (HLDA = 1);
                        repeat
                                select channel;
                                if (selected = FALSE)
                                then HREQ: = 0;
                                else    set(select lines for i);
                                        channel start: = 1;
                                        update(priority code);
                                        wait for (channel end = 1);
                                        channel start: = 0;
                        until (HREQ = 0);
until forever;
```

Contactpennen:

HLDA :INPUT;
Het HoLDAcknowledge signaal, dat als reactie op een holdrequest, aangeeft dat de controller nu master over de systeembussen is.

HREQ :OUTPUT;
Het HoldREQuest signaal waarmee de controller de CPU verzoekt om de bussen af te staan voor dmatransfers.

DREQ[0..3] :INPUT;
De DmaREQuestlijnen waarop de aanvragen voor dma van de periferals binnenkomen, voor een bepaald kanaal.

MASK[0..3] :INPUT;
De MASKeersignalen afkomstig van het dma-channelblok waarmee wordt aangegeven of een kanaal al dan niet gemaskeerd is.

CH_START :OUTPUT;
Het CHannel START signaal geeft aan het dma-channelblok door dat er een dma- transfer gestart kan worden.

CH_END :INPUT;
Het CHannel END signaal is de terugmelding van het dma-channelblok dat een dma- transfer is afgehandeld.

SELECT[0..1] :OUTPUT;
Deze lijnen geven aan welk kanaal geSELECTeerd is voor het uitvoeren van een dma-transfer.

IENABLE :INPUT;
Input ENABLE signaal geeft aan dat er data op de interne databus wordt aangeboden ter overname.

OPEN :INPUT;
Het Output Prioritycode ENable signaal geeft aan dat de data van het interne priority code register naar de interne databus moet worden doorgelaten.

EXTLOAD :INPUT;
Het EXTernal LOAD signaal is het PLOAD signaal van de slave businterface. Het priority code load signaal geeft aan dat het priority register geladen dient te worden met de data op de interne databus.

DATA[0..2] :TRISTATE BIDIRECTIONAL;
De interne databus voor het laden en lezen van het priority code register.

8.2 Decompositie naar datapad en controller

8.2.1 datapadbeschrijving

Het system DATABLOCK bevat het datapad van de priority encoder (PRIOR_ENC). Er volgt nu een algemene beschrijving van dit datapad en een beschrijving van de externe contacten van dit blok. Dit datablock is weer onderverdeeld in drie subblokken welke de drie aanwezige subprocessen uitvoeren te weten:

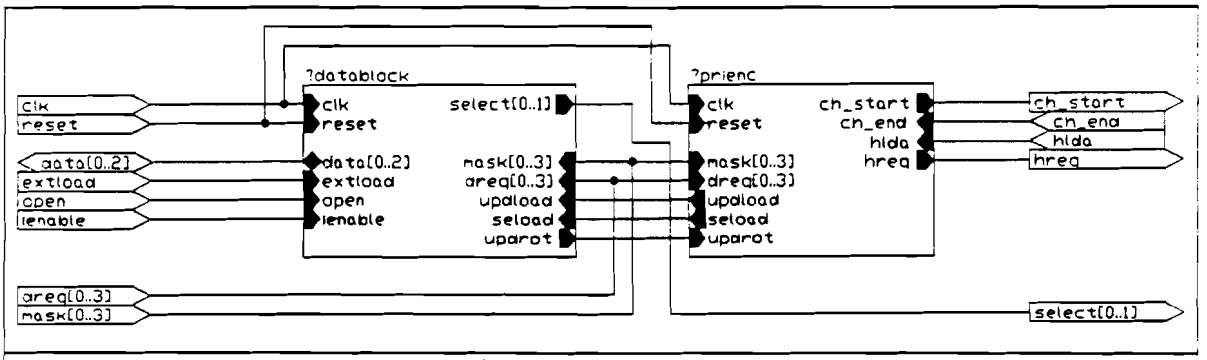
*codeblock: bevat het code register en de interface met de interne databus

*upd_l: bevat de update logica om de volgende initieel hoogste prioriteit te bepalen als er rotating priority is ingesteld.

*prio_l: bevat de logica die het juiste kanaal selecteert aan de hand van de DREQ's, MASK's en ingestelde priority van de kanalen

*srg: het selectregister waarin de selectie van het opgestarte kanaal wordt vastgehouden

In figuur 8.1 zijn de contacten en interconnecties weergegeven van het datapad (datablock) en de controller (prienc) waaruit de priority encoder is opgebouwd. De beschrijving van deze contacten en interconnecties in SID zijn te vinden in listing 8.1.



Figuur 8.1 prior_enc

```
SYSTEM PRIOR_ENC;

CONTACT
CLK             :INPUT;
RESET           :INPUT;
HLDA            :INPUT;
HREQ            :OUTPUT;
DREQ[0..3]      :INPUT;
MASK[0..3]      :INPUT;
CH_START        :OUTPUT;
CH_END          :INPUT;
SELECT[0..1]:OUTPUT;
IENABLE         :INPUT;
OPEN            :INPUT;
EXTLOAD         :INPUT;
DATA[0..2]      :TRISTATE BIDIRECTIONAL;

USE
PRI             :PRIENC;
DB              :DATABLOCK;

CONNECT
CLK=PRI.CLK=DB.CLK;
RESET=PRI.RESET=DB.RESET;
HLDA=PRI.HLDA;
HREQ=PRI.HREQ;
DREQ=PRI.DREQ=DB.DREQ;
MASK=PRI.MASK=DB.MASK;
CH_START=PRI.CH_START;
CH_END=PRI.CH_END;
SELECT=DB.SELECT;
IENABLE=DB.IENABLE;
OPEN=DB.OPEN;
EXTLOAD=DB.EXTLOAD;
DATA=DB.DATA;
DB.UPDROT=PRI.UPDROT;
DB.UPDLOAD=PRI.UPDLOAD;
DB.SELOAD=PRI.SELOAD;
```

Listing 8.1 SID-beschrijving van PRIOR_ENC

contactpennen:

SELECT[0..1] :OUTPUT;
Deze lijnen geven aan welk kanaal geSELECTeerd is voor het uitvoeren van een dma-transfer.

IENABLE :INPUT;
Input ENABLE signaal geeft aan dat er data op de interne databus wordt aangeboden ter overname.

OPEN :INPUT;
Het Output Prioritycode ENable signaal geeft aan dat de data van het interne priority code register naar de interne databus moet worden doorgelaten.

EXTLOAD :INPUT;
Het EXTernal LOAD signaal is het PLOAD signaal van de slave businterface. Het priority code load signaal geeft aan dat het priority register geladen dient te worden met de data op de interne databus.

DATA[0..2] :TRISTATE BIDIRECTIONAL;
De interne databus voor het laden en lezen van het priority code register.

DREQ[0..3] :INPUT;
De DmaREQuestlijnen waarop de aanvragen voor dma van de periferie binnenkomen, voor een bepaald kanaal.

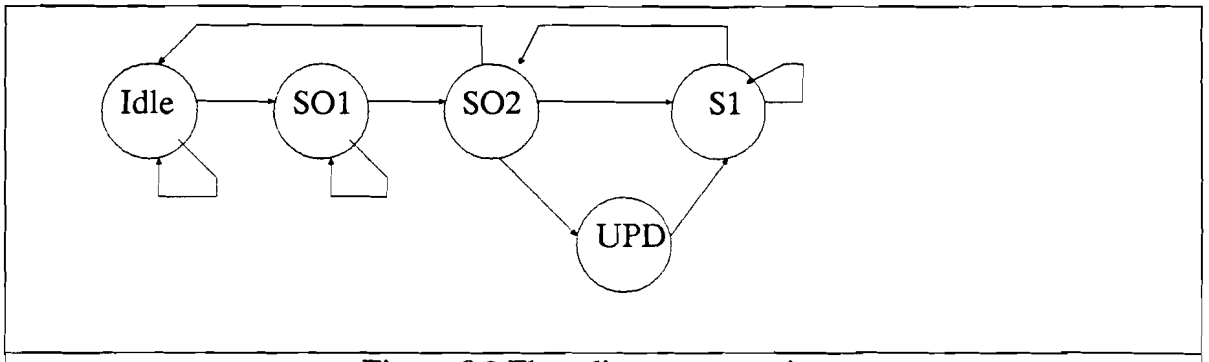
MASK[0..3] :INPUT;
De MASKeersignalen afkomstig van het dma-channelblok waarmee wordt aangegeven of een kanaal al dan niet gemaskeerd is.

UPDLOAD :INPUT;
Het LOAD signaal waarmee de initiele prioriteit in het priority code register wordt geupdate in het dat we te maken hebben met rotating priority.

SELOAD :INPUT;
Het SELect LOAD signaal zorgt ervoor dat het geselecteerde kanaal wordt vastgehouden in het selectieregister van waaruit we de SELECT lijnen naar buiten voeren.

UPDROT :OUTPUT;
UPDate bij ROTating priority geeft aan of er al dan niet sprake is van rotating priority

8.2.2 finite state machinebeschrijving van de controller



Figuur 8.2 Flow diagram van prienc

Idle: IF ((DREQ AND NOT MASK) < > 0) AND NOT HLDA THEN GOTO SO1;

Wacht tot er een kanaal ongemaskeerd is aangevraagd en de HLDA inactief is en blijf anders in de Idle toestand. De HLDA moet afgevallen zijn, we hebben nl. onze HREQ laag gemaakt dus kan het zijn dat de CPU hierop nog niet gereageerd heeft.

SO1: IF HLDA THEN GOTO SO2 ELSE GOTO SO1;

Stuur een HREQ naar buiten en wacht op een HLDA.

Eis: de CPU mag de HLDA niet wegnemen zolang HREQ aanwezig is.

SO2: IF ((DREQ AND NOT MASK) = 0) THEN GOTO IDLE ELSE IF UPDROT THEN GOTO UPD ELSE GOTO S1;

Neem het juiste selectiesignaal over in het output register waarmee dan het juiste kanaal geselecteerd wordt (SELOAD). Als er geen ongemaskeerde requests meer zijn ga dan naar idle anders ga, afhankelijk van updating of fixed priority instelling, naar resp. UPD of S1.

UPD: GOTO S1;

Laad de update waarde in het priority code register (UPDLOAD); stuur het channel startsignaal naar buiten (CH_START).

S1: IF CH_END THEN GOTO S02 ELSE GOTO S1;

Stuur het channel startsignaal naar buiten (CH_START). Wacht nu op het channel eindsignaal en ga daarna naar SO2 om eventueel een ander kanaal te servicen of de dma te beeindigen.

Het flow diagram is weergegeven in figuur 8.2, voor de complete SID beschrijving van de controller zie listing 8.2. Hierin zijn alle outputs per state weergegeven.

```
SYSTEM Prienc;

CONTACT

CLK             :INPUT;
RESET           :INPUT;
DREQ[0..3]      :INPUT;
MASK[0..3]      :INPUT;
CH_END          :INPUT;
HLDA            :INPUT;
UPDROT          :INPUT;
HREQ            :OUTPUT;
CH_START        :OUTPUT;
SELOAD          :OUTPUT;
UPDLOAD         :OUTPUT;

FUNCTION

TRIGGER CLK-1

STATE Idle      :

HREQ = 0;
CH_START = 0;
SELOAD = 0;
UPDLOAD = 0;

IF ((DREQ AND NOT MASK)0) AND NOT HLDA
THEN GOTO SO1;

STATE SO1       :

HREQ = 1;
CH_START = 0;
SELOAD = 0;
UPDLOAD = 0;

IF HLDA THEN GOTO SO2
ELSE GOTO SO1;

STATE SO2       :

HREQ = 1;
CH_START = 0;
SELOAD = 1;
UPDLOAD = 0;

IF ((DREQ AND NOT MASK) = 0)
THEN GOTO IDLE
ELSE IF UPDROT THEN GOTO UPD
ELSE GOTO S1;

STATE S1        :

HREQ = 1;
CH_START = 1;
SELOAD = 0;
UPDLOAD = 0;

IF CH_END THEN GOTO SO2
ELSE GOTO S1;

STATE UPD:
HREQ = 1;
CH_START = 1;
SELOAD = 0;
UPDLOAD = 1;

GOTO S1;
```

Listing 8.2 SID-becshrijving van prienc (FSM)

8.3 Hierarchische beschrijving van het datapad

Het datapad (datablock) is weer onderverdeeld in drie subblokken welke de drie aanwezige subprocessen uitvoeren te weten:

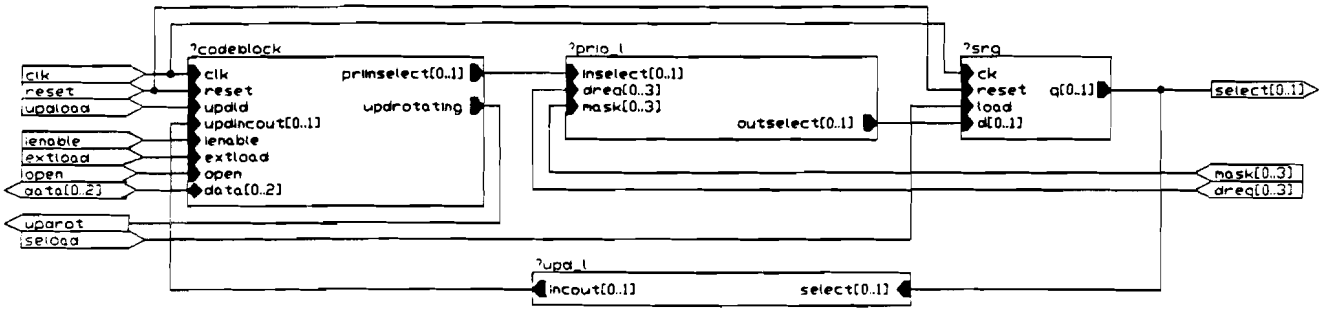
*codeblock: bevat het code register en de interface met de interne databus

*upd_l: bevat de update logica voor het geval er rotating priority is gevraagd

*prio_l: bevat de logica die het juiste kanaal selecteert aan de hand van de DREQ's, MASK's en ingestelde priority van de kanalen

*srg: het selectregister waarin de selectie van het opgestarte kanaal wordt vastgehouden. Dit register is reeds een generische cell uit de bibliotheek van ASA.

Bovengenoemde onderverdeling is weergegeven in figuur 8.3, en de SID beschrijving van de interconnecties in listing 8.3.



Figuur 8.3 datablock

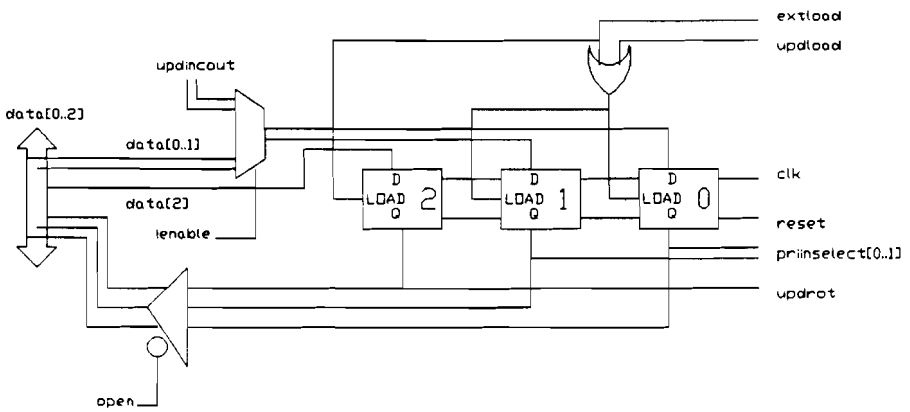
```
SYSTEM Datablock;

CONTACT

CLK             :INPUT;
RESET           :INPUT;
DATA[0..2]      :TRISTATE BIDIRECTIONAL;
EXTLOAD         :INPUT;
OPEN            :INPUT;
IENABLE         :INPUT;
SELECT[0..1]    :OUTPUT;
MASK[0..3]      :INPUT;
DREQ[0..3]      :INPUT;
UPDLOAD         :INPUT;
SELOAD          :INPUT;
UPDROT          :OUTPUT;

USE

CDB             :CODEBLOCK;
UPD             :UPD_L;
PRI             :PRIO_L;
SRG             :GENERIC S$D_REG(WORDLENGTH: = 2,
                                 RESETABLE: = TRUE);

CONNECT

CLK = CDB.CLK = SRG.CK;
RESET = CDB.RESET = SRG.RESET;
IENABLE = CDB.IENABLE;
EXTLOAD = CDB.EXTLOAD;
OPEN = CDB.OPEN;
DATA = CDB.DATA;
CDB.PRIINSELECT = PRI.INSELECT;
UPDROT = CDB.UPDROTATING;
UPDLOAD = CDB.UPDLD;
CDB.UPDINCOUT = UPD.INCOUT;
DREQ = PRI.DREQ;
MASK = PRI.MASK;
PRI.OUTSELECT = SRG.D;
SELECT = UPD.SELECT = SRG.Q;
SELOAD = SRG.LOAD;
```

Listing 8.3 SID-beschrijving van datablock

Het codeblock is opgebouwd uit generische cellen. Het schema hiervan is weergegeven in figuur 8.4 en de bijbehorende SID listing in listing 8.4.

Upd_l is in feite een tweebits incrementor en is beschreven in de vorm van logische functies, zie hiervoor listing 8.5.

Tot slot is het blok prio_1 opgebouwd uit een stuk logicabeschrijving in de vorm van logische funties, de encoder (deze had ook beschreven kunnen worden als een CASE of een geneste IF-THEN-ELSE statement), en enkele generische cellen, and's en inverters. Zie voor schema en SID-beschrijving figuur 8.5 en listing 8.6.



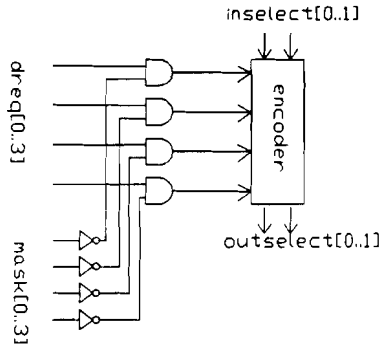
Figuur 8.4 codeblock

```
SYSTEM Codeblock;

CONTACT
CLK                 :INPUT;
RESET               :INPUT;
UPDLD               :INPUT;
UPDINCOUT[0..1]     :INPUT;
IENABLE             :INPUT;
EXTLOAD             :INPUT;
OPEN                :INPUT;
DATA[0..2]          :BIDIRECTIONAL;
UPDROTATING         :OUTPUT;
PRIINSELECT[0..1]           :OUTPUT;

USE

L_OR                :GENERIC S$ORN(N: = 2);
DFF[0..2]           :GENERIC S$D_RG1
                          (RESETABLE: = TRUE);
TROUT               :GENERIC S$TRBUSBUF
                          (WORDLENGTH: = 3,
                          DRIVE: = 1,INVERT: = FALSE,
                          NOTINVERT: = TRUE);
MUX[0..1]           :GENERIC S$MUX2;

CONNECT

RESET = DFF[0].RESET = DFF[1].RESET
 = DFF[2].RESET;
CLK = DFF[0].CK = DFF[1].CK = DFF[2].CK;
PRIINSELECT[0] = DFF[0].Q = TROUT.I[0];
DATA = TROUT.Y;
PRIINSELECT[1] = DFF[1].Q = TROUT.I[1];
UPDROTATING = DFF[2].Q = TROUT.I[2];
TROUT.EN = OPEN;
UPDLD = L_OR.I[0];
EXTLOAD = L_OR.I[1] = DFF[2].LOAD;
L_OR.Y = DFF[0].LOAD = DFF[1].LOAD;
UPDINCOUT = MUX[*].I[0];
DATA[0..1] = MUX[*].I[1];
DATA[2] = DFF[2].D;
MUX[*].Y = DFF[0..1].D;
MUX[0].S = MUX[1].S = IENABLE;
```

Listing 8.4 SID-beschrijving codeblock

```
SYSTEM Upd_l;
CONTACT
SELECT[0..1]      :INPUT;
INCOUT[0..1]      :OUTPUT;
FUNCTION
INCOUT[0] = NOT SELECT[0];
INCOUT[1] = (NOT SELECT[1] AND SELECT[0]) OR (SELECT[1] AND NOT SELECT[0]);
```

Listing 8.5 SID-beschrijving upd_l



Figuur 8.5 prio_l

```
SYSTEM ENCODER;

CONTACT

I0,I1,
R0,R1,R2,R3       :INPUT;
O0,O1             :OUTPUT;

FUNCTION

O1 = I1 AND NOT I0 AND R2 OR
I1 AND I0 AND R3 OR
NOT I1 AND I0 AND NOT R1 AND R2 OR
I1 AND NOT I0 AND NOT R2 AND R3 OR
NOT I1 AND NOT I0 AND NOT R0
AND NOT R1 AND R2 OR
NOT I1 AND I0 AND NOT R1 AND NOT R2 AND R3 OR
NOT I1 AND NOT I0 AND NOT R0
AND NOT R1 AND NOT R2 OR
I1 AND I0 AND NOT R0 AND NOT R1 AND NOT R3;

O0 = NOT I1 AND I0 AND R1 OR
I1 AND I0 AND R3 OR
NOT I1 AND NOT I0 AND NOT R0 AND R1 OR
I1 AND NOT I0 AND NOT R2 AND R3 OR
NOT I1 AND I0 AND NOT R1 AND NOT R2 AND R3 OR
I1 AND I0 AND NOT R0 AND R1 AND NOT R3 OR
NOT I1 AND NOT I0 AND NOT R0
AND NOT R1 AND NOT R2 OR
I1 AND NOT I0 AND NOT R0 AND NOT R2 AND NOT R3;
```

```
SYSTEM PRIO_L;

CONTACT

INSELECT[0..1]    :INPUT;
DREQ[0..3]        :INPUT;
MASK[0..3]        :INPUT;
OUTSELECT[0..1]:OUTPUT;

VAR
I[0..1]           :MEMORY;

USE

DM[0..3]          :GENERIC      S$ANDN(N: = 2);
INV[0..3]         :GENERIC      S$INV;
ENC               :ENCODER;

CONNECT

INSELECT[0] = ENC.I0;
INSELECT[1] = ENC.I1;
OUTSELECT[0] = ENC.O0;
OUTSELECT[1] = ENC.O1;
DM[0].Y = ENC.R0;
DM[1].Y = ENC.R1;
DM[2].Y = ENC.R2;
DM[3].Y = ENC.R3;
FOR I: = 0 TO 3 DO
BEGIN
DREQ[I] = DM[I].I[0];
INV[I].Y = DM[I].I[1];
MASK[I] = INV[I].I;
END;
```

Listing 8.6 SID-beschrijving prio_l en encoder

9 DMA CHANNEL

9.1 Functionele beschrijving

Er zijn vier kanalen die alleen geselecteerd kunnen worden indien ze ongemaskeerd zijn. Het dma-channel is een blok dat de informatie van alle kanalen bevat en deze via een selectie mechanisme (multiplexers) doorlaat naar het blok dat de eigenlijke transfer regelt, het transfer control blok.

Ook worden in het dma-channel alle bewerkingen op de data van het betreffende kanaal uitgevoerd om deze up to date te houden. Voor elk kanaal is de benodigde informatie opgeslagen in registers, te weten: addressregister, termination count register en een mode register, zoals beschreven in hoofdstuk 6, paragraaf 6.2.

Verder beschikt het kanaal over een incrementor en een decrementor want na elke start van een dma-transfer moet het adres met één verhoogt worden en de termination count met één verlaagt. We hebben maar één incrementor en één decrementor nodig omdat er nooit twee kanalen gelijktijdig actief kunnen zijn, en de update werkzaamheden kunnen worden uitgevoerd parallel met de eigenlijke transfer.

Procesbeschrijving in pseudo-pascal:

```
disable all registers;
channel end: = 0; cycle start: = 0;
repeat
                    wait for (channel start = 1);
                    enable (selected registers i);
                    set(select lines for i);
                    cycle start: = 1;

                    decrement(TERM.COUNT);
                    if (TERM.COUNT = 0)
                    then    TC[i]: = 1;
                            Maskbit[i]: = 1;
                    else    TC[i]: = 0;

                    increment(ADDRESSREG.);
                    wait for (cycle end = 1);
                    cycle start: = 0;
                    channel end: = 1;
                    wait for (channel start = 0);
                    channel end: = 0;
until forever;
```

Contactpennen:

CH_START :INPUT;
Het CHannel START signaal geeft aan dat een dma transfer gestart kan worden.

CY_STRT :OUTPUT;
Het CYcle STaRT signaal geeft aan het transfer control blok door dat er een dma transfer kan worden uitgevoerd met de aangeboden gegevens.

CYCLE_END :INPUT;
Het cycle end signaal is de terugmelding van het transfer control blok dat een dma transfer is afgerond.

INSELECT[0..1] :INPUT;
Deze lijnen geven aan welk kanaal er geserviced dient te worden in verband met een dma transfer.

BISELECT[0..1] :INPUT;
Deze selctlijnen zijn afkomstig van slave businterface en geven aan welk kanaal is geselecteerd voor lees of schrijf operaties van daaruit.

SENABLE :INPUT;
Het Select ENABLE signaal geeft aan welke selectlijnen gebruikt moeten worden voor de selectie van het kanaal. Als SENABLE hoog is worden de BISELECTlijnen doorgelaten en anders de INSELECT lijnen.

ALOAD :INPUT;
Laad het addressregister dat geselecteerd is.

CLOAD :INPUT;
Laad het termination count register dat geselecteerd is.

ILOAD :INPUT;
Laad het mode register dat geselecteerd is.

IENABLE :INPUT;
Input ENABLE geeft aan dat er data op de interne databus wordt aangeboden ter overname.

OAEN :INPUT;
Het Output Addressregister ENable signaal geeft aan dat de data die in het geselecteerde Adresregister staat moet worden aangeboden op de interne databus (DATA[0..AB-1]).

OCEN :INPUT;
Idem voor termination count register.

OMEN :INPUT;
Idem voor mode register.

DATA[0..AB-1] :TRISTATE BIDIRECTIONAL;
De interne databus welke de verbinding met de slave businterface onderhoudt.

INTADDRBUS[0..AB-1] :OUTPUT;
De interne databus welke naar het transfer control blok gaat met daarop het adres van de geheugenplaats die bij de volgende transfer moet worden geadresseerd.

A_STRB :OUTPUT;
Het Adres STRoBe signaal geeft aan dat de data op INTADDRBUS moet worden overgenomen omdat die op dat moment geldig is.

MASK[0..3] :OUTPUT;
Mask geeft aan de priority encoder door of een kanaal al dan niet gemaskeerd is.

TC[0..3] :OUTPUT;
De termination count signalen geven aan of een kanaal zijn blok transfers heeft afgehandeld.

MODE[0..1] :OUTPUT;
MODE geeft aan het transfer control blok door welke transfer mode moet worden gebruikt voor de uit te voeren transfer.

CH_END :OUTPUT;
Het CHannel END signaal is de terugmelding naar het priority encoder blok dat een dma-transfer is afgehandeld.

OUTSELECT[0..1] :OUTPUT;
Het OUTSELECT signaal wordt doorgegeven aan het transfer control blok, het geeft aan welke periferal moet worden geserviced.

9.2 Decompositie naar datapad en controller

9.2.1 datapadbeschrijving

Het datapad van het dma-channel is uitgevoerd als vier blokken met elk hun eigen taak, alle vier deze datablokken liggen op hetzelfde hierarchische niveau als de controller.

Deze vier datablokken en de controller zijn met hun contacten en interconnecties weergegeven in figuur 9.1 en listing 9.1. De vier datablokken worden hieronder summier besproken voor wat betreft hun inhouden en functies:

*DMA:

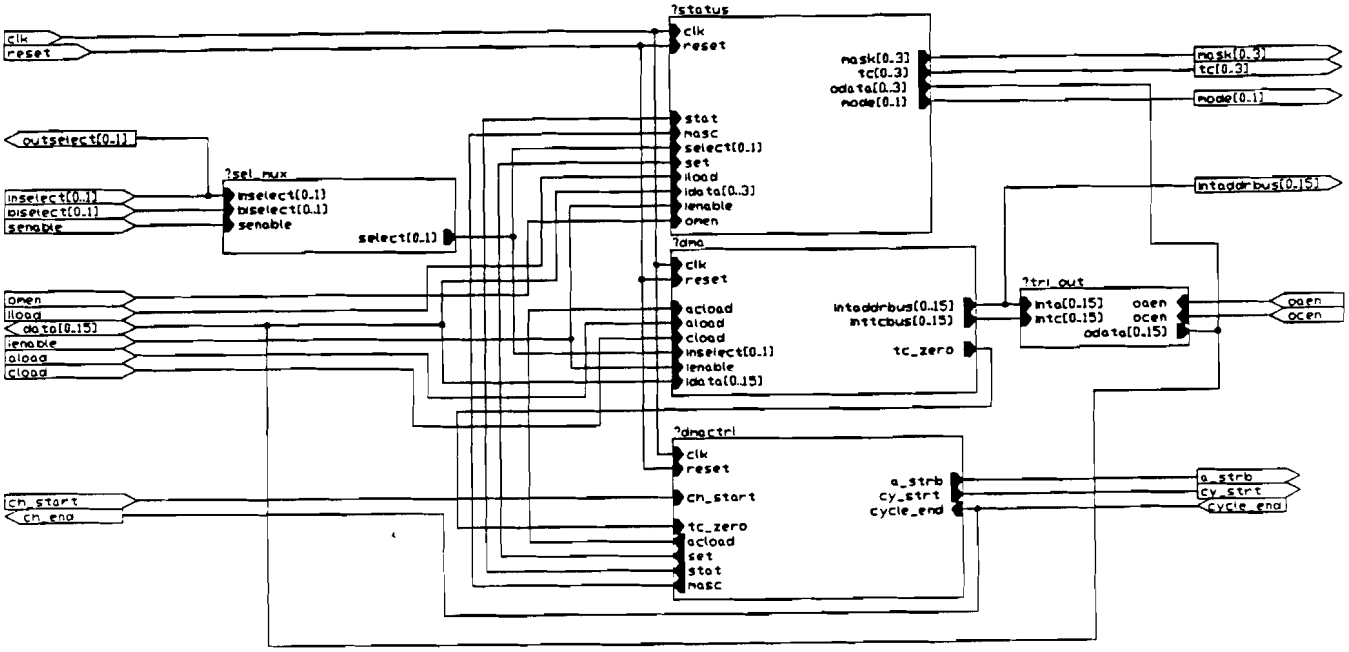
Dit blok bevat de addressregisters en de terminationcountregisters van alle kanalen. Ook bevat dit blok een incrementor en een decrementor om resp. het address en de termination count aan te passen. Verder is er selectielogica voor de kanalen en de aanwezig om het juiste kanaal en evt. het juiste type register te kunnen adresseren. Dit blok wordt uitgebreider besproken in paragraaf 9.3.

*STATUS:

Dit blok bevat de statusregisters (mode registers), logica om het kanaal en de juiste bits van elk register te kunnen adresseren en aan te passen. Zie figuur 9.3.

*TRI_OUT:

Dit blok bevat de logica om het voor de businterface mogelijk te maken om of het geselecteerde addressregister of het termination countregister uit te lezen.



Figuur 9.1 dma_ch

```
SYSTEM DMA_CH;

PARAMETER
AB;

CONTACT

CLK             :INPUT;
RESET           :INPUT;
CH_START        :INPUT;
CYCLE_END       :INPUT;
INSELECT[0..1]  :INPUT;
BISELECT[0..1]  :INPUT;
SENABLE         :INPUT;
ALOAD           :INPUT;
CLOAD           :INPUT;
ILOAD           :INPUT;
IENABLE         :INPUT;
OAEN            :INPUT;
OCEN            :INPUT;
OMEN            :INPUT;
DATA[0..AB-1]   :TRISTATE BIDIRECTIONAL;

INTADDRBUS[0..AB-1]:OUTPUT;
CY_STRT         :OUTPUT;
A_STRB          :OUTPUT;
MASK[0..3]      :OUTPUT;
TC[0..3]        :OUTPUT;
MODE[0..1]      :OUTPUT;
CH_END          :OUTPUT;
OUTSELECT[0..1]:OUTPUT;

USE

CTR             :DMACTRL;
DMA             :DMA(AB:=AB);
STA             :STATUS;
OTRI            :TRI_OUT(AB:=AB);
MUX             :SEL_MUX;

CONNECT

CLK=CTR.CLK=DMA.CLK=STA.CLK;
RESET=CTR.RESET=DMA.RESET=STA.RESET;
A_STRB=CTR.A_STRB;
CH_START=CTR.CH_START;
CYCLE_END=CTR.CYCLE_END=CH_END;
MUX.SELECT=DMA.INSELECT=STA.SELECT;
INSELECT=OUTSELECT=MUX.INSELECT;
BISELECT=MUX.BISELECT;
SENABLE=MUX.SENABLE;
ALOAD=DMA.ALOAD;
CLOAD=DMA.CLOAD;
ILOAD=STA.ILOAD;
IENABLE=DMA.IENABLE=STA.IENABLE;
DATA=DMA.IDATA;
DATA[0..3]=STA.IDATA;
INTADDRBUS=DMA.INTADDRBUS=OTRI.INTA;
OTRI.INTC=DMA.INTTCBUS;
CTR.TC_ZERO=DMA.TC_ZERO;
CY_STRT=CTR.CY_STRT;
MASK=STA.MASK;
TC=STA.TC;
MODE=STA.MODE;
STA.ODATA=DATA[0..3];
CTR.ACLOAD=DMA.ACLOAD;
CTR.SET=STA.SET;
CTR.STAT=STA.STAT;
CTR.MASC=STA.MASC;

DATA=OTRI.ODATA;
OCEN=OTRI.OCEN;
OAEN=OTRI.OAEN;
OMEN=STA.OMEN;
```

Listing 9.1 SID-beschrijving dma_ch

*SEL_MUX:

Hierin bevindt zich een multiplexer die of de selectsignalen van de businterface doorlaat of die van de priority encoder.

9.2.2 finite state machinebeschrijving van de controller

Idle: IF CH_START THEN GOTO S1;

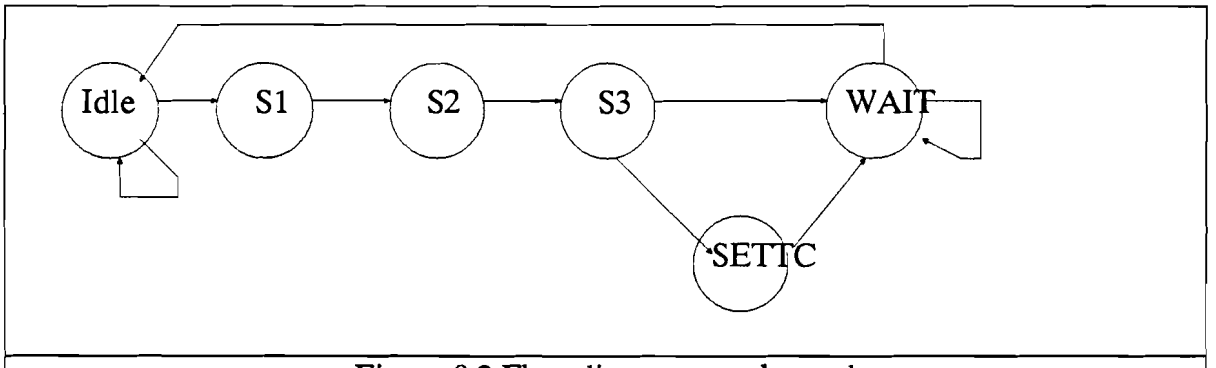
Wacht op het channel startsignaal.

S1: GOTO S2;

Maak cycle startsignaal actief (CY_STRT), geef een adresstrobe signaal om de geselecteerde kanaalinformatie door te geven aan het transfer control blok (A_STRB).

S2: GOTO S3;

Geef signaal dat de geupdate waardes uit de incrementor en de decrementor terugschrijft in de betreffende registers (ACLOAD).



Figuur 9.2 Flow diagram van dmactrl

S3: IF TC_ZERO THEN GOTO SETTC ELSE GOTO WAIT;

Test of de nieuwe termination counterwaarde nul geworden is, want is dat het geval dan moet het status en het maskbit worden geset.

SETTC: GOTO WAIT;

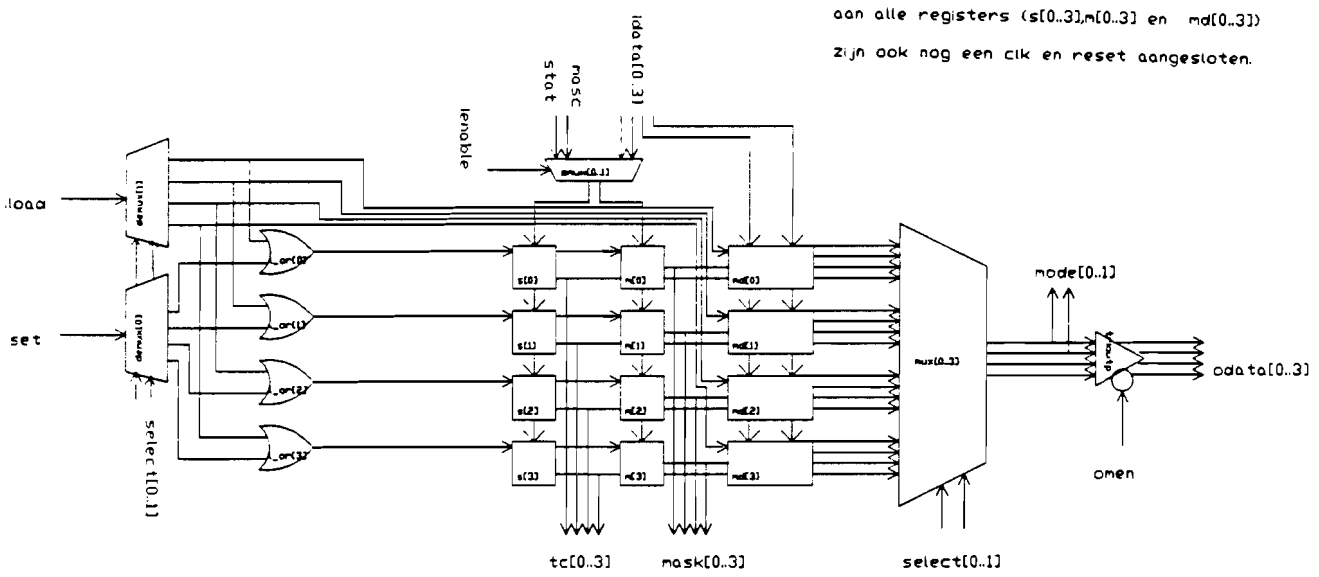
Set het status bit van het betreffende kanaal en maskeer dit kanaal zodat resp. het Term. Count signaal wordt geset en er geen transfers meer via dit kanaal mogelijk zijn zolang het maskbit niet gereset wordt via de CPU of system reset.

WAIT: IF CYCLE_END THEN GOTO IDLE ELSE GOTO WAIT;

Het CY_STRT signaal is nog altijd actief en wordt actief gehouden totdat een cycle end signaal wordt ontvangen tengevolge waarvan dan naar de Idle toestand wordt teruggekeerd.

```
SYSTEM Dmactrl;

CONTACT
CLK             :INPUT;
RESET           :INPUT;
CH_START        :INPUT;
CYCLE_END       :INPUT;
TC_ZERO         :INPUT;
CY_STRT         :OUTPUT;
A_STRB          :OUTPUT;
ACLOAD          :OUTPUT;
SET             :OUTPUT;
STAT            :OUTPUT;
MASC            :OUTPUT;

FUNCTION
TRIGGER CLK-1

STATE IDLE      :
CY_STRT = 0;
A_STRB = 0;
ACLOAD = 0;
SET = 0;
STAT = 1;
MASC = 1;

IF CH_START
THEN GOTO S1;

STATE S1        :
CY_STRT = 1;
A_STRB = 1;
ACLOAD = 0;
SET = 0;
STAT = 1;
MASC = 1;

GOTO S2;

STATE S2        :
CY_STRT = 1;
A_STRB = 0;
ACLOAD = 1;
SET = 0;
STAT = 1;
MASC = 1;

GOTO S3;

STATE S3        :
CY_STRT = 1;
A_STRB = 0;
ACLOAD = 0;
SET = 0;
STAT = 1;
MASC = 1;

IF TC_ZERO
THEN GOTO SETTC
ELSE GOTO WAIT;

STATE SETTC:
CY_STRT = 1;
A_STRB = 0;
ACLOAD = 0;
SET = 1;
STAT = 1;
MASC = 1;
GOTO WAIT;

STATE WAIT:
CY_STRT = 1;
A_STRB = 0;
ACLOAD = 0;
SET = 0;
STAT = 1;
MASC = 1;

IF CYCLE_END
THEN GOTO IDLE
ELSE GOTO WAIT;
```

Listing 9.2 SID-beschrijving dmactrl



Figuur 9.3 status

9.3 Hierarchische beschrijving van het datapadblok dma

De hierarchie van het totale ontwerp is te vinden op de tweede uitklapbare bladzijde na hoofdstuk 10.

Het blok dma, figuur 9.6, is opgebouwd uit een termination countregisterblok (tcreg), een addressregisterblok (addrreg) en een blok dat aangeeft of de termination count 0 is geworden (tcnor).

Addreg, figuur 9.4, en tcreg, figuur 9.5, zijn opgebouwd uit een voor beiden identiek datablk en resp. een incrementor en een decrementor. De incrementor is beschreven in hoofdstuk 5.

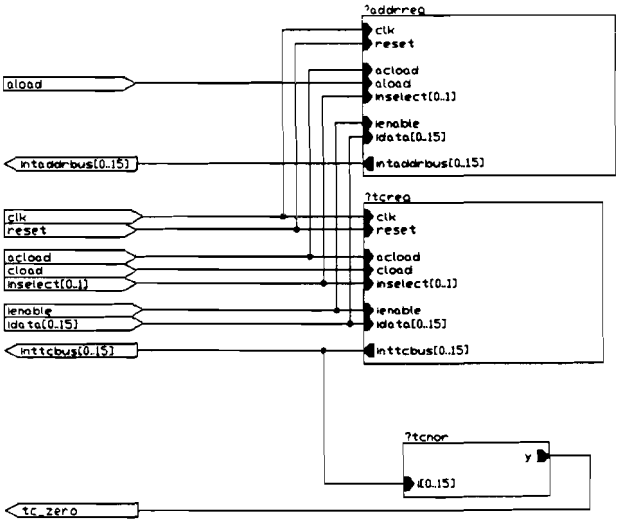
Het datablk is, zoals weergegeven in figuur 9.7, opgebouwd uit:

*loadblk: dit blok stuurt een eventueel load signaal naar het juiste register, zie figuur 9.8.

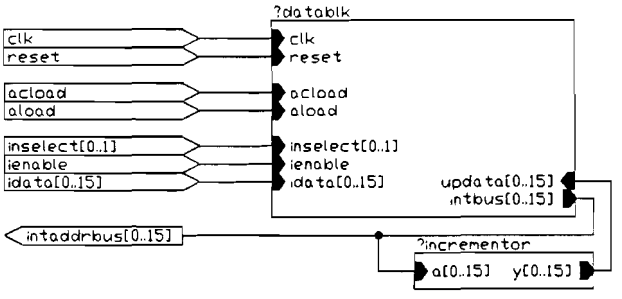
*registers: hierin zijn vier parallele registers ondergebracht, zie figuur 9.9.

*regselect: door dit blok wordt aan de hand van de inselectlijnen, de output van een van de registers doorgelaten. Zie figuur 9.10.

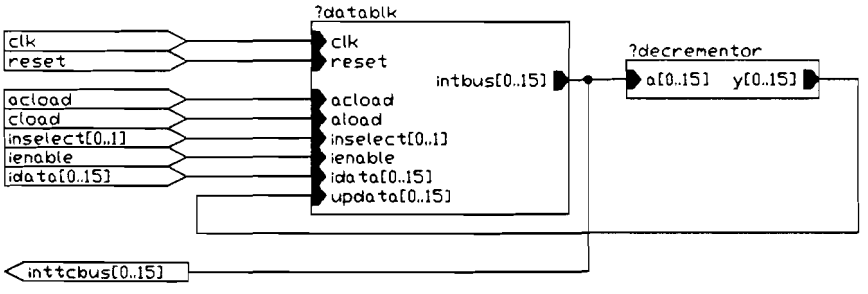
*inpmux: de input multiplexer zorgt ervoor dat of de data van de interne databus, of de geupdate data uit de incrementor/decrementor wordt doorgelaten naar de inputs van de registers. Zie figuur 9.11.



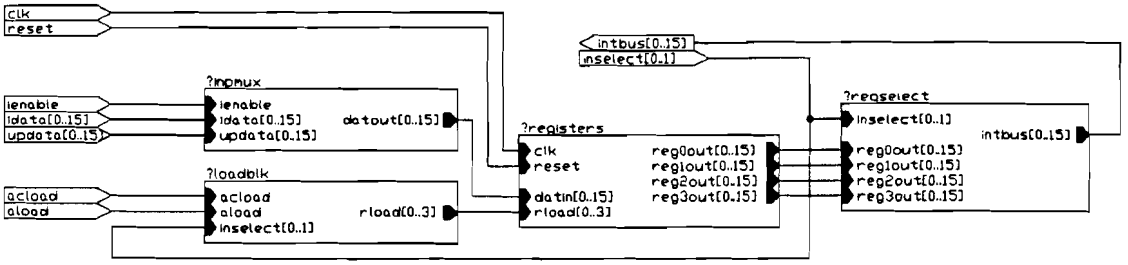
Figuur 9.6 dma



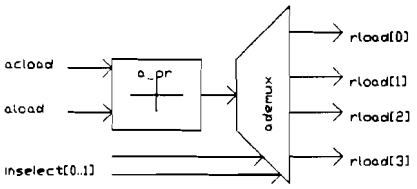
Figuur 9.4 addrreg



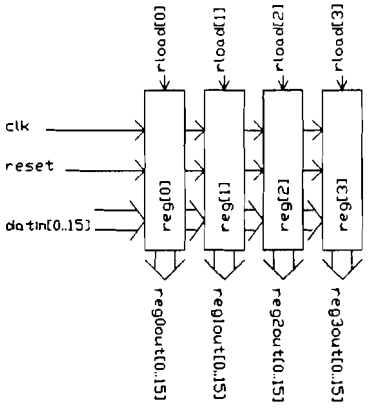
Figuur 9.5 tcreg



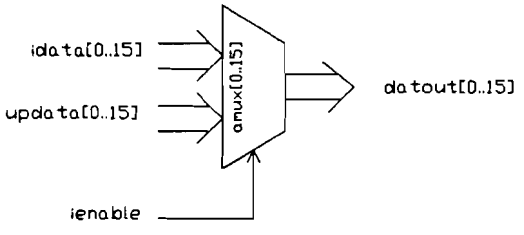
Figuur 9.7 datablk



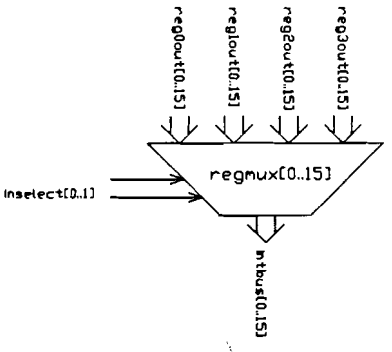
Figuur 9.8 loadblk



Figuur 9.9 registers



Figuur 9.11 inpmux



Figuur 9.10 regselect

10 TRANSFER CONTROL

10.1 Functionele beschrijving

Dit blok regelt de gehele datatransfer. Het selecteert de periferal via DmaACKnowledge signaal, en de geheugenplaats via de adresbus. Tevens genereert dit blok alle controlsignalen om een transfer te laten plaatsvinden.

Om dit alles te kunnen regelen moet dit blok informatie hebben over:

-welk kanaal moet worden geserviced i.v.m. DACK.
-welk adres moet verzonden worden.
-welk soort transfer is vereist.

Al deze informatie wordt uit het dma-channel blok opgehaald nadat dit blok het startsignaal voor een dma transfer (cycle) heeft gegeven.

Dit blok regelt ook het invoegen van eventuele wait states met behulp van de ready inputlijn. In de verify mode wordt er wel geselecteerd (adres en periferal) maar er worden geen controlsignalen verzonden. Dit kan nuttig zijn voor verificatiedoeleinden.

Procesbeschrijving in pseudo-pascal:

```
IOR: = 1;IOW: = 1;MEMR: = 1;MEMW: = 1;
cycle end: = 0;
repeat
                wait for (cycle start = 1);
                DACK[i]: = 1;
                addressbuffer: = ADDRESSREG.;
                modebuffer: = R_W_V;
                selectionbuffer: = select lines

                case

                Write :
                        IOR: = 0;
                        MEMW: = 0;
                        wait for (READY);
                        wait (clock pulse);
                        MEMW: = 1;
                        IOR: = 1;

                Read :
                        MEMR: = 0;
                        IOW: = 0;
                        wait for (READY);
                        wait (clock pulse);
                        IOW: = 1;
                        MEMR: = 1;

                Verify :
                        wait (dma cycle time);

                cycle end: = 1;
                DACK[i]: = 0;
                wait for (cycle start = 0);
                cycle end: = 0;
until forever;
```

Contactpennen:

INTADDR[0..AB-1] :INPUT;
Op deze bus wordt het door het dma-channel blok gegenereerde adres aangeboden waarmee de bij de transfer de geheugenplaats wordt geadresseerd.

MODE[0..1] :INPUT;
MODE geeft aan welk soort transfer er uitgevoerd moet worden.

SELECT[0..1] :INPUT;
Select geeft aan welke periferal geacknowledged moet worden.

CY_STRT :INPUT;
Cycle start geeft aan dat er een dma transfer kan worden gestart.

A_STRB :INPUT;
Met het adres strobe signaal wordt aangegeven dat het op intaddr aangeboden adres geldig is en in de adreslatch mag worden overgenomen.

READY :INPUT;
Het ready signaal is een door de periferie verstuurd signaal om aan te geven dat een datatransfer kan worden uitgevoerd omdat de periferal er klaar voor is.

ADDRESS[0..AB-1] :TRISTATE OUTPUT;
Dit is de adresbus waarop door de controller gegenereerde adressen naar buiten worden gestuurd, deze outputs zijn altijd in tristate behalve wanneer er dma transfers worden uitgevoerd.

DACK[0..3] :OUTPUT;
Het dma acknowledge signaal voor de verschillende kanalen, dit signaal geeft aan de periferal, welke om dma gevraagd heeft door dat zijn aanvraag geserviced wordt.

R[0..1] :TRISTATE OUTPUT;
W[0..1] :TRISTATE OUTPUT;
W[0] en R[0] zijn de write en read controllijnen die in geval van een write-cycle de dma transfer regelen.(MEMW,IOR).

W[1] en R[1] zijn de write en read controllijnen die in geval van een read-cycle de dma transfer regelen.(IOW,MEMR).

Deze signalen zijn laag actief.

CYCLE_END :OUTPUT;
Het cycle end signaal geeft aan het dma-channel blok door dat een transfer is afgerond.

10.2 Decompositie naar datapad en controller

10.2.1 datapadbeschrijving

Het datapad van het transfer control blok bestaat uit vijf blokken welke allen generische elementen uit de ASA bibliotheek zijn. Het betreft hier twee tristate buffers, twee registers en een multiplexer. Alle blokken zijn op hetzelfde hierarchische niveau als de controller beschreven.

Deze vijf blokken met hun contacten en interconnecties zijn als schema, figuur 10.1, en als SID-beschrijving, listing 10.1, weergegeven op de volgende bladzijden.

*TRIBUF:

een tristate output poort ter afsluiting van het latchregister op de systeembussen.

*LATCH:

het latchregister waarin het actieve adres wordt vastgehouden voor een dma-cyclus.

*DEMUX:

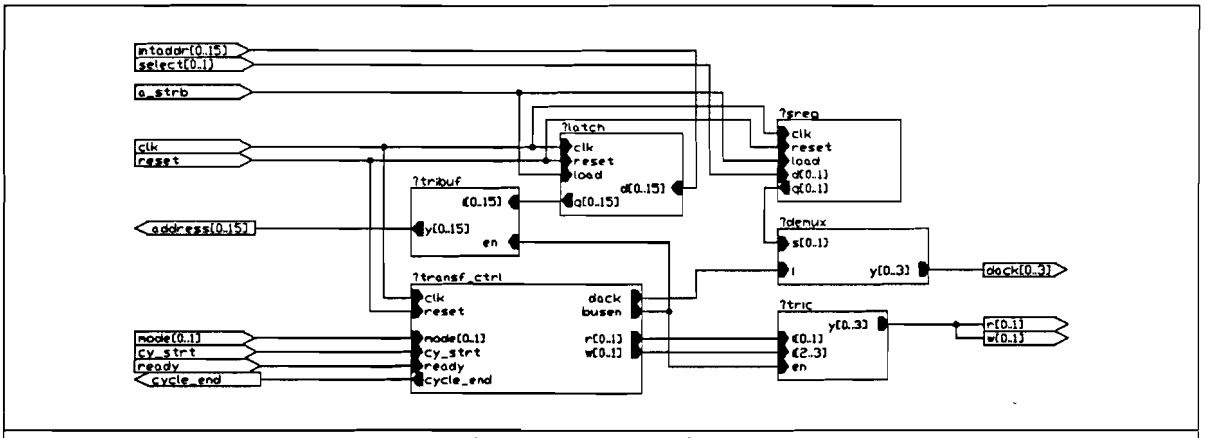
de multiplexerlogica die bepaalt aan welk kanaal een dma_acknowledge wordt doorgegeven met behulp van de select uit het dma-channel (dit is dezelfde die in de priority encoder wordt gegenereerd en vastgehouden).

*SREG:

het selectregister, dit neemt gelijktijdig met de adreslatch de nieuwe select over en houdt deze vast ten behoeve van de demux.selectlijnen.

*TRIC:

een tristate output poort ter afsluiting van de controllijnen die op de externe controlbus worden aangeboden.



Figuur 10.1 transf_blk

```
SYSTEM TRANSF_BLK;

PARAMETER
AB;

CONTACT

CLK                 :INPUT;
RESET               :INPUT;
INTADDR[0..AB-1]:INPUT;
MODE[0..1]          :INPUT;
SELECT[0..1]        :INPUT;
CY_STRT             :INPUT;
A_STRB              :INPUT;
READY               :INPUT;

ADDRESS[0..AB-1] :TRISTATE OUTPUT;
DACK[0..3]          :OUTPUT;
R[0..1]             :TRISTATE OUTPUT;
W[0..1]             :TRISTATE OUTPUT;
CYCLE_END           :OUTPUT;

USE

TRF                 :TRANSF_CTRL;
TRIBUF              :GENERIC S$TRBUSBUF(
                            WORDLENGTH: = AB,
                            DRIVE: = 1,
                            INVERT: = FALSE,
                            NOTINVERT: = TRUE);
TRIC                :GENERIC S$TRBUSBUF(
                            WORDLENGTH: = 4,
                            DRIVE: = 1,
                            INVERT: = FALSE,
                            NOTINVERT: = TRUE);
DEMUX               :GENERIC S$DEMULTI(WIDTH: = 4);
LATCH               :GENERIC S$D_REG
                            (WORDLENGTH: = AB,
                            RESETABLE: = TRUE);
SREG                :GENERIC S$D_REG
                            (WORDLENGTH: = 2,
                            RESETABLE: = TRUE);

CONNECT
RESET = TRF.RESET = SREG.RESET = LATCH.RESET;
CLK = TRF.CLK = SREG.CK = LATCH.CK;
A_STRB = LATCH.LOAD = SREG.LOAD;
LATCH.D = INTADDR;
TRIBUF.I = LATCH.Q;
TRIBUF.EN = TRIC.EN = TRF.BUSEN;
TRIBUF.Y = ADDRESS;
TRIC.I[0..1] = TRF.R;
TRIC.I[2..3] = TRF.W;
TRIC.Y[0..1] = R;
TRIC.Y[2..3] = W;
Cy_strt = TRF.Cy_strt;
Cycle_end = TRF.Cycle_end;
Ready = TRF.Ready;
MODE = TRF.MODE;
DEMUX.I = TRF.DACK;
SREG.D = SELECT;
DEMUX.S = SREG.Q;
DEMUX.Y = DACK;
```

Listing 10.1 SID-beschrijving transf_blk

10.2.2 finite state machinebeschrijving van de controller

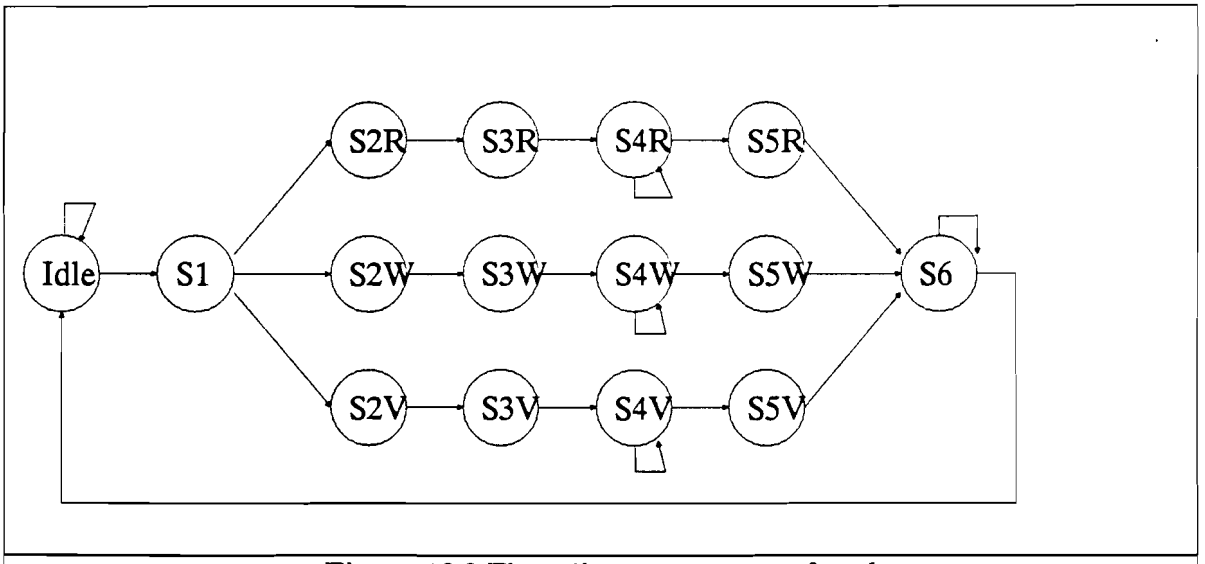
We zullen hier enkel de states en stateovergangen bespreken voor het geval van een read-cycle de write en verify cycle verlopen identiek met alleen andere waardes voor R en W.

Idle: IF CY_STRT THEN GOTO S1;

Wacht op CYcle STaRT signaal.

S1: IF MODE = 1 THEN GOTO S2R ELSE IF MODE = 2 THEN GOTO S2W ELSE GOTO S2V;

Enable de tristate buffers met het BUSENable signaal want er wordt begonnen met een dma transfercycle. Selecteer uitgaande van de MODE lijnen welke transfer er uitgevoerd moet worden. (In dit geval gaan we uit van MODE = 1)



Figuur 10.2 Flow diagram van transf_ctrl

S2R: GOTO S3R;

Maak de DmaACKnowledge actief om aan te geven dat er een periferal wordt geselecteerd. De demultiplexer zorgt er dan met gebruikmaking van de select informatie voor dat de DACK aan de juiste periferal wordt doorgegeven. R[1] wordt laag actief, hiermee is dan dus het memory read signaal geactiveerd.

S3R: GOTO S4R;

W[1], het IO Write signaal, wordt laag actief gemaakt.

S4R: IF READY THEN GOTO S5R ELSE GOTO S4R;

Wacht op een terugmelding van de periferal, die aangeeft dat hij klaar is voor de overname van de data, het READY signaal.

S5R: GOTO S6;

Laat het write signaal inactief (hoog) worden en geef aan het dma-channel blok door dat een dma CYCLE beEiNDigd is.

S6: IF NOT CY_STRT THEN GOTO Idle ELSE GOTO S6;

Maak nu ook het read signaal inactief (hoog). En wacht nu tot het CYcle STaRT signaal weggenomen wordt ter indicatie dat het dma channel op het cycle end signaal heeft gereageerd, en ga daarna naar de Idle toestand waar alle signalen weer inactief zijn.

Het flow diagram van deze controller is weergegeven in figuur 10.2, en de complete SID-beschrijving van de controller (transf_ctrl) is weergegeven in listing 10.2.

```
SYSTEM TRANSF_CTRL;

CONTACT
CLK             :INPUT;
RESET           :INPUT;
MODE[0..1]      :INPUT;
CY_STRT         :INPUT;
READY           :INPUT;

DACK            :OUTPUT;
BUSEN           :OUTPUT;
R[0..1]         :OUTPUT;
W[0..1]         :OUTPUT;
CYCLE_END       :OUTPUT;

FUNCTION
TRIGGER CLK-1

STATE Idle      :
R=3;
W=3;
DACK=0;
BUSEN=0;
CYCLE_END=0;

IF CY_STRT
THEN GOTO S1;

STATE S1:
R=3;
W=3;
DACK=0;
BUSEN=1;
CYCLE_END=0;

IF MODE=1 THEN GOTO S2R
ELSE
IF MODE=2 THEN GOTO S2W
ELSE GOTO S2V;

STATE S2V:
R=3;
W=3;
DACK=1;
BUSEN=1;
CYCLE_END=0;

GOTO S3V;

STATE S3V:
R=3;
W=3;
DACK=1;
BUSEN=1;
CYCLE_END=0;

GOTO S4V;

STATE S4V       :
R=3;
W=3;
DACK=1;
BUSEN=1;
CYCLE_END=0;
IF READY THEN GOTO S5V ELSE GOTO S4V;

STATE S5V       :
R=3;
W=3;
DACK=1;
BUSEN=1;
CYCLE_END=1;

GOTO S6;

STATE S2R:
R=1;
W=3;
DACK=1;
BUSEN=1;
CYCLE_END=0;

GOTO S3R;

STATE S3R:
R=1;
W=1;
DACK=1;
BUSEN=1;
CYCLE_END=0;

GOTO S4R;

STATE S4R       :
R=1;
W=1;
DACK=1;
BUSEN=1;
CYCLE_END=0;

IF READY THEN GOTO S5R
ELSE GOTO S4R;

STATE S5R       :
R=1;
W=3;
DACK=1;
BUSEN=1;
CYCLE_END=1;

GOTO S6;

STATE S2W:
R=2;
W=3;
DACK=1;
BUSEN=1;
CYCLE_END=0;

GOTO S3W;

STATE S3W:
R=2;
W=2;
DACK=1;
BUSEN=1;
CYCLE_END=0;

GOTO S4W;

STATE S4W:
R=2;
W=2;
DACK=1;
BUSEN=1;
CYCLE_END=0;

IF READY THEN GOTO S5W
ELSE GOTO S4W;

STATE S5W:
R=2;
W=3;
DACK=1;
BUSEN=1;
CYCLE_END=1;

GOTO S6;

STATE S6:
R=3;
W=3;
DACK=1;
BUSEN=1;
CYCLE_END=1;

IF NOT CY_STRT THEN GOTO Idle
ELSE GOTO S6;
```

Listing 10.2 SID-beschrijving van transf_ctrl

11 EVALUATIE VAN HET ONTWERP

Er is een dma controller ontworpen, die functioneel aan alle in de voorgaande hoofdstukken beschreven eisen voldoet. De controller is tot op gatesniveau functioneel gesimuleerd. Timing verificatie en testvectorgeneratie zijn achterwege gebleven omdat tools hiervoor niet voorhanden waren of nog niet bugfree waren.

Er is vervolgens wel een layout aangemaakt van het ontwerp om een idee te krijgen van de verbruikte oppervlaktes en om enkele layoutoptimalisatievuistregels te kunnen vaststellen voor een ontwerp op dit niveau met ASA. Deze vuistregels komen in het volgende hoofdstuk aan de orde.

Het ontwerpen van deze controller had achteraf gezien in een tijdsbestek van plus minus een half jaar gerealiseerd kunnen worden. Het was echter in de beginfase van mijn afstudeerwerk zo dat ASA nog niet bugfree was, en vooral de simulator is in de loop van de tijd verbeterd en uitgebreid met enkele features zoals het kunnen aanbrengen van probes op elk niveau in de hierarchie.

12 LAYOUTOPTIMALISATIE

De layout kan sterk verbeterd worden door vrij grote standardcellen te gebruiken. Dus door zelf standardcell attributes toe te voegen in de hierarchie op een hoger niveau dan door ASA zelf zou doen.

ASA voegt standardcell attributes toe op het niveau van de leaf modules in de hierarchische SID beschrijving. Dit zijn de eventueel geparametriseerde generische cellen en de door FSM gemaakte logica- en registerblokken.

Verder dient men het aantal bussen te beperken. Het is zelfs aan te bevelen na de functionele decompositie al rekening hiermee te houden, door bijvoorbeeld bidirectionele bussen te gebruiken tussen de verschillende blokken op het hoogste niveau. De oppervlakte die verspeeld wordt door het gebruik van tristate buffers in elk blok weegt niet op tegen de winst die geboekt wordt door het gebruik van maar half zo veel bussen. Je hebt dan nl. een input- en een outputbus vervangen door een bidirectionele bus.

Om dan de layout nog verder te optimaliseren zouden manual moves kunnen worden uitgevoerd op de verschillende layout onderdelen na het placement. Dit heeft echter als nadeel dat deze ontwerpstap niet automatisch reproduceerbaar is terwijl alles wat ASA zelf ontwerpt vanuit de SID-beschrijving wel automatisch reproduceerbaar is.

12.1 Optimalisatievuistregels.

Om met ASA een zo optimaal mogelijke layout te genereren moet met een aantal hieronder besproken vuistregels rekening gehouden worden.

1.

Gebruik maximaal 8 subsystems binnen een system. Hierdoor wordt voorkomen dat in het place algorithme een zogenaamde split procedure in werking treedt. Split verdeelt de subsystems nl. vrijwel random in meerdere blokken. De ontwerper kan beter zelf zijn hierarchie zodanig aanpassen dat subsystems met de grootste gebondenheid bij elkaar komen te staan. De ontwerper heeft meestal een beter inzicht in welke blokken bij elkaar horen dan welke "intelligente" compiler ook.

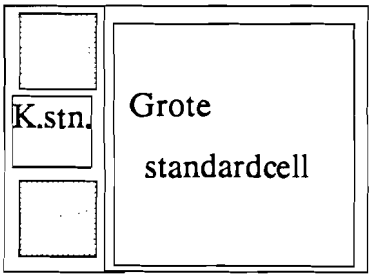
2.

Het gebruiken van standardcell attributes. Als er een standardcell attribute op een bepaald niveau in de hierarchie wordt toegevoegd worden alle onderliggende systemen in een standardcell uitgevoerd. Meestal is de oppervlakte van een standardcell kleiner dan de oppervlakte van hetzelfde system dat is opgebouwd als een combinatie van standardcellen, die lager in de hierarchie liggen.

Het zou dus uit oppervlakte oogpunt het gunstigste zijn om zo groot mogelijke standardcells te maken. Hieraan kleeft echter een nadeel: De verbruikte CPU tijd voor het aanmaken van standardcells stijgt meer dan lineair met het aantal basic cells. Daarom is er een grens gesteld aan het aantal te gebruiken basic cells per standardcell van circa 500 systems (basic cells).

3.

Probeer de layout op te bouwen uit gelijk grote standardcells. Als er op een gelijk hierarchisch niveau een grote en een kleine standardcell geplaatst moeten worden wordt er een groot deel van de oppervlakte verknoeit omdat de slices waarin beide standardcells worden geplaatst even hoog worden gemaakt. Zie figuur 12.1. Dit gebeurt omdat ASA de hierarchie van het ontwerp niet doorbreekt bij het plaatsen van de standardcells. De



We zien hier een kleine en een grote standardcell naast elkaar geplaatst in twee slices.

De grijze vlakken geven de oppervlakte verkwisting aan.

Figuur 12.1 Standardcellplaatsing per slice

standardcells worden in slices geplaatst van boven naar beneden in de hierarchie. Dit resulteert dus in een weinig gevulde slice voor de kleinste standardcel op hetzelfde niveau. Door het gericht toevoegen van standardcell attributes is er een grote oppervlaktewinst te boeken.

De door mij ontworpen dma-controller met de CMOS 3 micron technologie van de foundry MIETEC leverde de volgende resultaten op:

alleen standardcell attributes aan de leaf cellen (standaard door ASA)

oppervlakte 233mm^2

2 grote standardcellen binnen een van de vier functionele blokken

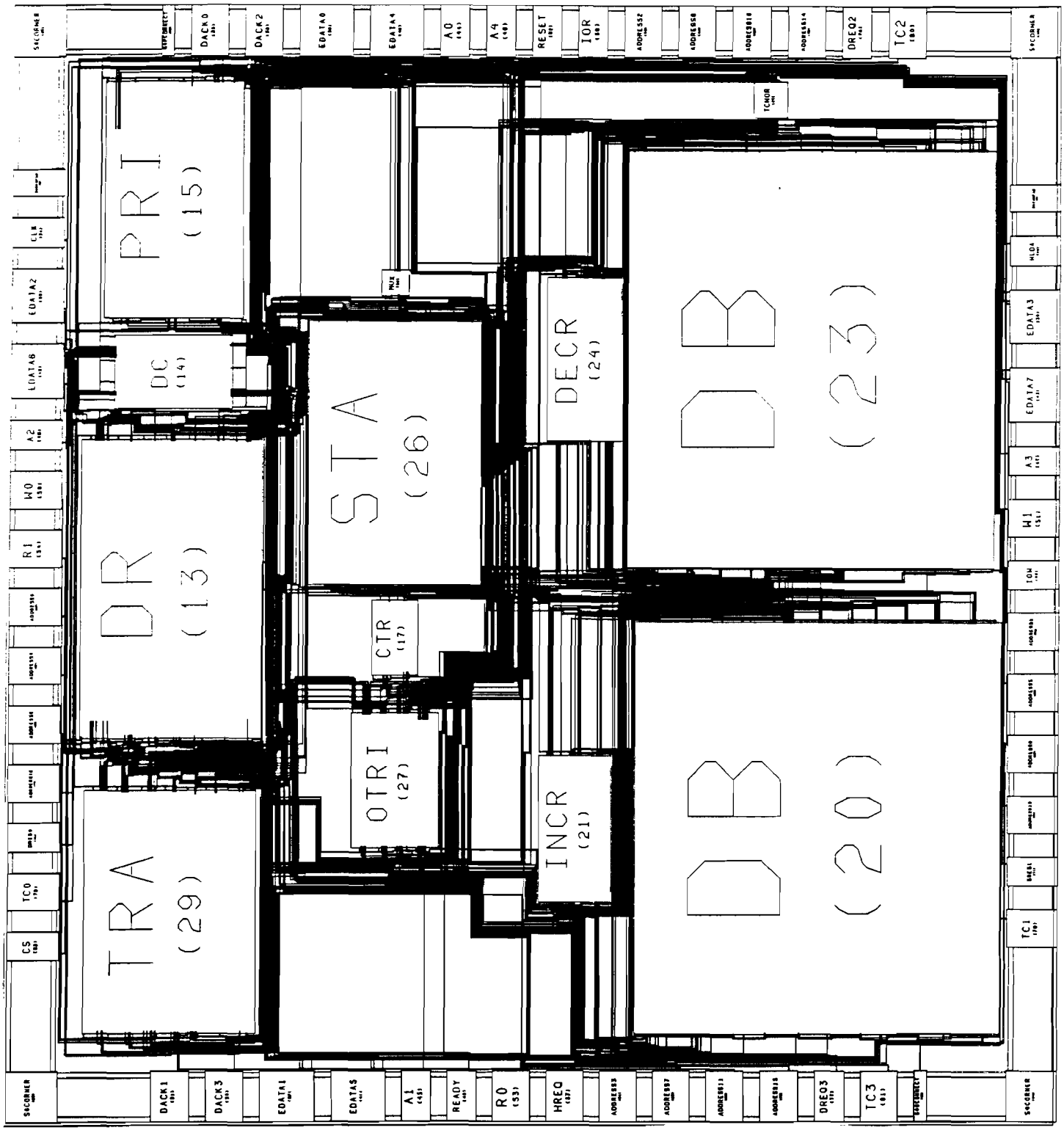
oppervlakte 120mm^2

standardcell attributes op een volgens mij optimale manier

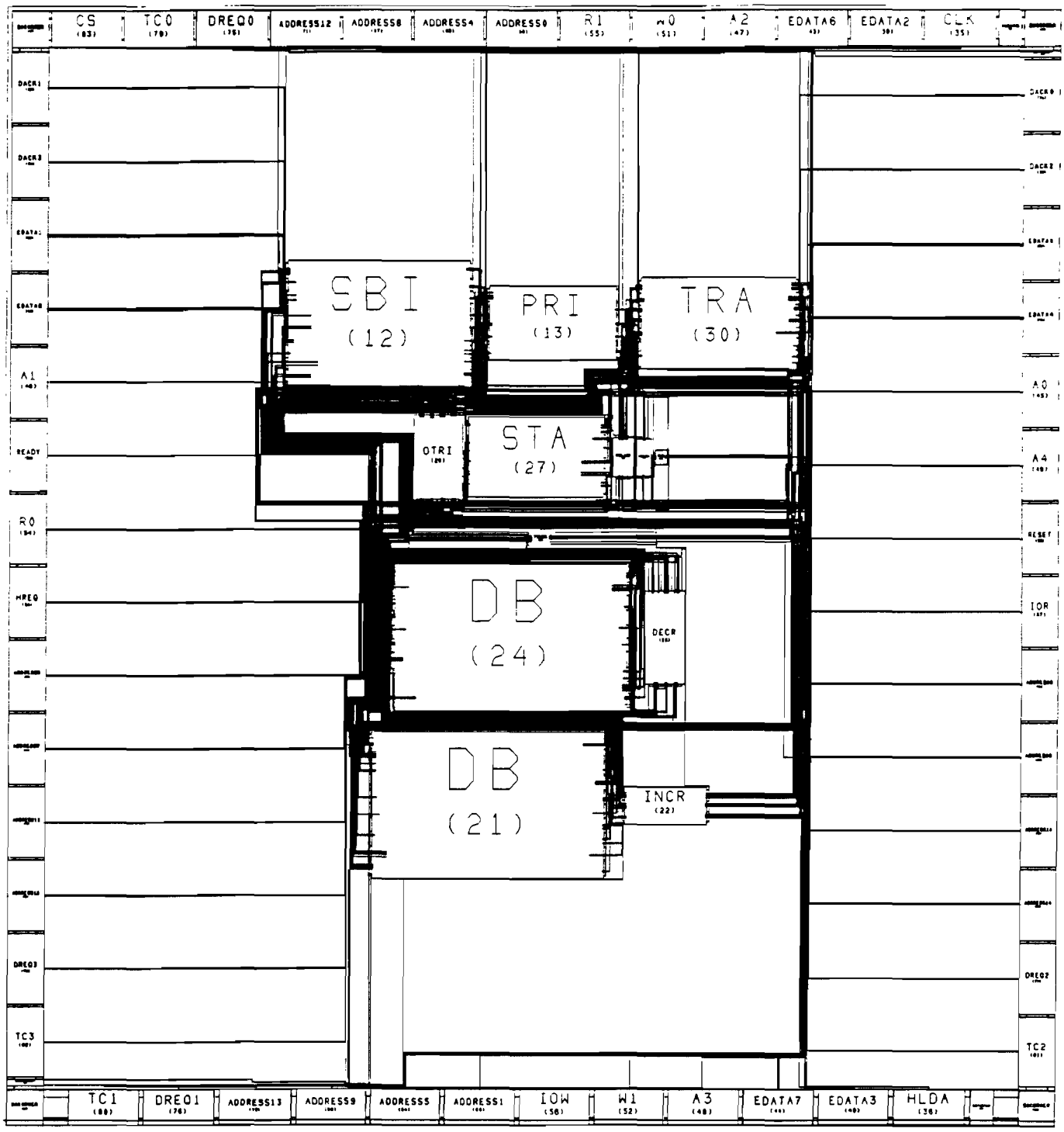
oppervlakte 57mm^2

De oppervlakte van de layout viel dus voor deze testcase met een factor 4 te verbeteren. Het resultaat is te zien in figuur 12.2, in deze figuur zijn de standardcellen weergegeven door een blok met daarin de naam van het system.

Dezelfde layout is ook nog een keer aangemaakt met de CMOS 1.5 micron technologie van de foundry ES2. Dit leverde als resultaat op dat we een pad-limited chip kregen met een oppervlakte van 60mm^2 waarvan slechts een oppervlak van 15mm^2 werd ingenomen door de dma-controller dus bij gebruikmaking van deze technologie is het mogelijk een dma-controller te gebruiken als een standardcomponent op bijvoorbeeld een interfacechip. Zie hiervoor figuur 12.3.



Figuur 12.2 Layout dma-controller, CMOS 3 micron



Figuur 12.3 Layout dma controller, CMOS 1.5 micron

13 ALGEMENE CONCLUSIES

Simuleren is op elk niveau van het grootste belang. Simulaties kunnen je op een vrij hoog niveau tonen of een ontwerp aan de gestelde eisen voldoet. Verder geloof ik echter niet dat het zinvol is om eerst de orginele SID beschrijving te simuleren vervolgens na FSM en dan nog eens na TOGATES Je blijft nl. met geïdealiseerde gevallen werken en de standaard ingestelde vertragingstijden bij de basic gates geven toch geen realistisch beeld van de werkelijkheid. De resultaten na FSM en TOGATES zijn alleen maar verschillend door de "onjuiste" vertragingstijden die zijn ingebouwd in de basic gates. Als je deze verschillen buiten beschouwing laat hebben de simulaties echter alleen maar tot doel om te testen of TOGATES een juiste implementatie heeft uitgevoerd en daarvan hoort de ontwerper in feite uit te kunnen gaan. Het is daarom aan te raden alleen de SID beschrijving na FSM te simuleren omdat dit zeer veel sneller gaat dan een simulatie op gate niveau. Omdat een simulatie op gate niveau toch niet volgens de realistische tijdvertragingen tot stand is gekomen verschaft deze simulatie geen extra informatie omtrent de juistheid van het ontwerp.

Je maakt dus een SID beschrijving op een niveau dat door de compiler vertaald kan worden en je simuleert deze beschrijving. Test vervolgens of deze beschrijving echt vertaalbaar is. Maak op het hoogste niveau een functionele beschrijving die misschien niet vertaalbaar maar wel simuleerbaar is. Deze beschrijving wordt aangepast tot we een vertaalbare beschrijving hebben die dezelfde simulatieresultaten oplevert als de functionele beschrijving die samen met de opdrachtgever is geverifieerd.

Bij grote ontwerpen is het mogelijk dat na een functionele decompositie verschillende onderdelen van het ontwerp door verschillende groepen mensen worden uitgewerkt tot vertaalbaar SID.

Functionele decompositie leidt zeker niet altijd tot een goede layout. Er moet na de functionele decompositie wel degelijk rekening gehouden worden met gevolgen voor de layout. Binnen de functionele beschrijving zie je bijvoorbeeld helemaal geen bussen en/of registerbreedtes. Deze worden pas zichtbaar als je met de architectuur bezig bent.

Een algemene conclusie ten opzichte van het top down ontwerpen is dat, het volgens mij, niet reeel is een functionele opsplitsing in de hierarchie te handhaven tot op layoutniveau, omdat er geen één op één mapping van functioneel op layout hoeft te bestaan. Bijvoorbeeld: een simpele functionele beschrijving resulteert helemaal niet per definitie in een kleine layoutoppervlakte.

14 CONCLUSIES OVER ASA

Top Down Strategie is toepasbaar met ASA.

ASA is handig als een tool, die een groot gedeelte van het top down traject zoals voorgesteld in de structured VLSI design course, bestrijkt. Enkele wenselijke aanvullingen zijn:

kritisch pad analyser
timing verificator
testvector generator

ASA's werking als silicon compiler begint in het topdown traject na de state synthese, een uitgewerkte finite state machine per functioneel blok, en de datapadbeschrijving, waarin de arithmetische operatoren zijn ontworpen.

De finite state machines worden geïmplementeerd als Mealy machines, we zouden echter graag ook een implementatiemogelijkheid als moore machines met en zonder outputlogica willen zien.

Het is mogelijk om vanaf een DAISY workstation schematic entry om te zetten naar SID en omgekeerd. Volgens mij is het echter gemakkelijker en minder foutgevoelig om in SID connecties te beschrijven dan via een schematic entry.

De omzetting van SID op elk hierarchisch niveau naar schematic entry kan echter wel handig zijn voor documentatiedoeleinden.

14.1 WENSEN MET BETREKKING TOT ASA

SID

Graag zou ik een hoger vertaalbaar niveau binnen ASA willen zien. Hierbij denk ik vooral aan de mogelijkheid om ASA zelf arithmetische operatoren te laten vertalen.

FSM

Binnen FSM zou ik graag een keuzemogelijkheid willen zien tussen verschillende soorten implementaties in plaats van alleen maar Mealy machines. Een gewenste uitbreiding is in elk geval Moore machines en misschien zelfs Moore zonder outputlogica.

SIMULATIE

Op gatesniveau zou het handig zijn als de gates tenminste typicalwaardes voor hun vertragingen hadden. Dit zou in elk geval een zinvoller beeld afleveren dan met een eenheidsvertraging voor elke basic cell. Deze typical waardes zijn echter wel technologieafhankelijk en dus in strijd met de ASA filosofie. Het is echter ook niet logisch om de delay van een inverter gelijk te stellen aan de delay van een D-flipflop. Verder zijn de delays sterk afhankelijk van de aangelegde load. Hierop zou de simulator na de togates vertaling moeten letten.

14.2 VOORDELEN VAN ASA

Het parametriseerbaar zijn van de logische bouwbloken in de generische bibliotheek bespaart het aan elkaar breien van lange ketens logische basisblokken.

Het is met ASA mogelijk om op een snelle manier een layout aan te maken opgebouwd uit standardcell, PLA en regelmatige structuur blokken. Hierdoor is het mogelijk om snel een layout aan te maken om te kijken of onze hierarchie en de toevoeging van de standardcell attributes goed is geweest. Dit gaat in elk geval veel sneller dan er met een groep ontwerpers dagenlang over speculeren wat misschien de beste resultaten zou opleveren.

Het aanmaken van de layout is foutvrij omdat ASA een correctness by construction garandeert.

Alle ontwerpacties gebeuren op een en dezelfde database wat een enorme tijdbesparing oplevert ten opzichte van het werken met losse tools die allen op een verschillende database werken, waarbij via interfaceprogramma's en misschien zelfs verschillende computersystemen van de ene tool naar de andere overgegaan moet worden om de volgende ontwerpstap uit te voeren.

LITERATUURLIJST

lit. [1] Crijns, J.H.H.J.
FUNCTIONEEL ONTWERP VAN EEN VIERKANAALS DMA-CONTROLLER.
Vakgroep Digitale Systemen, Faculteit der Elektrotechniek,
Technische Universiteit Eindhoven, 1988.
Stageverslag EB/88/165

lit. [2] Stevens, M.P.J.
STRUCTURED VLSI DESIGN COURSE.
Vakgroep Digitale Systemen, Faculteit der Electrotechniek,
Technische Universiteit Eindhoven, 1988.

lit. [3] Lewin, D.
DESIGN OF LOGIC SYSTEMS.
Berkshire, England: Van Nostrand Reinhold (UK) Co. Ltd., 1985.

lit. [4] Horbst, E and C. Muller-Schloer and H. Schwartsel
DESIGN OF VLSI CIRCUITS, Based on VENUS.
Heidelberg, Germany: Springer-Verlag, 1987.

lit. [5] Kemper, J.P. en M.P.J. Stevens
MICROCOMPUTER SYSTEEMARCHITECTUUR III. 5e druk (ongewijzigde herdruk).
Hoogezand: Uitgeverij Stuberg, 1986.

lit. [6] HIGH PERFORMANCE PROGRAMMABLE DMA CONTROLLER (8237A/8237A-4/8237A-5).
In: MICROCOMPONENTS HANDBOOK: Microprocessors Volume I.
Santa Clara, CA: Intel Literature Sales, 1986.
1986 handbooks, oreder number 230843.

lit. [7] Leibson, S.H.
DEVELOPING CAE TOOLS FOR INTELLIGENT SILICON COMPILATION.
EDN, vol.33, no.6, 1988, p.130-6

lit. [8] Pangrle, B.M. and D.D. Gajski
DESIGN TOOLS FOR INTELLIGENT SILICON COMPILATION.
IEEE Trans. Comput. -Aided Des. Integr. Circuits Syst.,
Vol. CAD-6, no.6, 1987, p.1098-112

lit. [9] Stenzel, W.J.
SILICON COMPILER TAME 10,000-GATE-PLUS ASICS, GATE ARRAYS.
EDN, vol.33, no.9, 1988, p.195-202

lit. [10] Eijnde, W.F.K. van den
NIEUWE SILICON COMPILER BEKORT ONTWERPTIJD DRASTISCH:
Chipontwerp in stroomversnelling.
I2 Electrotechniek Electronica, 5e jaargang, no. 1, 1989, p.36-7

lit. [11] Scheelen, J. and S.J. Klaver
ASA/USER GUIDE (Preliminary version)
Eindhoven, Sagantec Europe B.V., 1988.

lit. [12] Scheelen, J. and S.J. Klaver
ASA/SID-TUTORIAL
Eindhoven, Sagantec Europe B.V., 1988.

lit. [13] Scheelen, J. and S.J. Klaver
SID/REFERENCE MANUAL
Eindhoven, Sagantec Europe B.V., 1988.

lit. [14] Scheelen, J. and S.J. Klaver
GENERIC CELL REFERENCE MANUAL
Eindhoven, Sagantec Europe B.V., 1988.

1 BIJLAGE

SID-beschrijving van het totale ontwerp

```
{**********************SLAVE BUSINTERFACE BLOCK**************************}

SYSTEM Datareg;
PARAMETER
  AB,DB;

CONTACT
  CLK    :INPUT;
  RESET  :INPUT;
  EXTDATA[0..DB-1]:BIDIRECTIONAL;
  INTDATA[0..AB-1]:BIDIRECTIONAL;
  LEN    :INPUT;
  HEN    :INPUT;
  WEN    :INPUT;
  LLD    :INPUT;
  HLD    :INPUT;
  GET    :INPUT;
  STO    :INPUT;
  A[1..4]        :INPUT;
  XLOAD          :INPUT;
  OXEN           :INPUT;
  LOAD[0..3]     :OUTPUT;
  OEN[0..3]      :OUTPUT;
  BISELECT[0..1]:OUTPUT;

USE
  HREG,LREG : GENERIC S$D_REG(WORDLENGTH:=DB,
                              RESETABLE:=TRUE);
  TRDL,TRDH,
  TRQL,TRQH : GENERIC S$TRBUSBUF(WORDLENGTH:=DB,
                                 DRIVE:=1,INVERT:=FALSE,
                                 NOTINVERT:=TRUE);

  TRGET,TRSTO:GENERIC S$TRBUSBUF(WORDLENGTH:=AB,
                                 DRIVE:=1,INVERT:=FALSE,
                                 NOTINVERT:=TRUE);

  DEMUX[0..1]:GENERIC S$DEMULTI(WIDTH:=4);

CONNECT
  CLK=HREG.CK=LREG.CK;
  RESET=HREG.RESET=LREG.RESET;
  EXTDATA=TRDL.I=TRDH.I=TRQL.Y=TRQH.Y;
  INTDATA=TRGET.I=TRSTO.Y;
  HREG.D=TRDH.Y=TRGET.Y[DB..AB-1];
  LREG.D=TRDL.Y=TRGET.Y[0..DB-1];
  HREG.Q=TRQH.I=TRSTO.I[DB..AB-1];
  LREG.Q=TRQL.I=TRSTO.I[0..DB-1];
  HREG.LOAD=HLD;
  LREG.LOAD=LLD;
  TRDL.EN=TRDH.EN=WEN;
  TRQL.EN=LEN;
  TRQH.EN=HEN;
  TRGET.EN=GET;
  TRSTO.EN=STO;
  BISELECT=A[1..2];
  DEMUX[0].S=DEMUX[1].S=A[3..4];
  DEMUX[0].I=XLOAD;
  DEMUX[0].Y=LOAD[0..3];
  DEMUX[1].I=OXEN;
  DEMUX[1].Y=OEN[0..3];
```

```
SYSTEM Datactrl;
CONTACT
  CLK    :INPUT;
  RESET  :INPUT;
  CS     :INPUT;
  HLDA   :INPUT;
  A0     :INPUT;
  IOR    :INPUT;
  IOW    :INPUT;

  SENABLE:OUTPUT;
  OXEN   :OUTPUT;
  GET    :OUTPUT;
  HLD    :OUTPUT;
  LLD    :OUTPUT;
  WEN    :OUTPUT;
  IENABLE:OUTPUT;
  STO    :OUTPUT;
  XLOAD  :OUTPUT;
  HEN    :OUTPUT;
  LEN    :OUTPUT;

FUNCTION
 TRIGGER CLK->1

 STATE IDLE:
  SENABLE=0;
  OXEN   =0;
  GET    =0;
  HLD    =0;
  LLD    =0;
  WEN    =0;
  IENABLE=0;
  STO    =0;
  XLOAD  =0;
  HEN    =0;
  LEN    =0;

 IF NOT CS AND NOT HLDA THEN GOTO S1;

 STATE S1:
  SENABLE=1;
  OXEN   =1;
  GET    =1;
  HLD    =1;
  LLD    =1;
  WEN    =0;
  IENABLE=0;
  STO    =0;
  XLOAD  =0;
  HEN    =0;
  LEN    =0;

 GOTO S2;

 STATE S2:
  SENABLE=1;
  OXEN   =0;
  GET    =0;
  HLD    =0;
  LLD    =0;
  WEN    =0;
  IENABLE=0;
```

```
 STO     =0;
 XLOAD   =0;
 HEN     =0;
 LEN     =0;

IF NOT IOW THEN GOTO W1
           ELSE
            IF NOT IOR THEN GOTO R1
                       ELSE
                         IF CS THEN GOTO IDLE
                               ELSE GOTO S2;

STATE W1:
 SENABLE=1;
 OXEN    =0;
 GET     =0;
 HLD     =A0;
 LLD     =NOT A0;
 WEN     =1;
 IENABLE=0;
 STO     =0;
 XLOAD   =0;
 HEN     =0;
 LEN     =0;

GOTO W2;

STATE W2:
 SENABLE=1;
 OXEN    =0;
 GET     =0;
 HLD     =0;
 LLD     =0;
 WEN     =0;
 IENABLE=0;
 STO     =0;
 XLOAD   =0;
 HEN     =0;
 LEN     =0;

GOTO W3;

STATE W3:
 SENABLE=1;
 OXEN    =0;
 GET     =0;
 HLD     =0;
 LLD     =0;
 WEN     =0;
 IENABLE=1;
 STO     =1;
 XLOAD   =1;
 HEN     =0;
 LEN     =0;

GOTO W4;

STATE W4:
 SENABLE=0;
 OXEN    =0;
 GET     =0;
 HLD     =0;
 LLD     =0;
 WEN     =0;
```

```
 IENABLE=0;
 STO    =0;
 XLOAD  =0;
 HEN    =0;
 LEN    =0;

IF NOT CS THEN GOTO W4
          ELSE GOTO IDLE;

STATE R1:
 SENABLE=0;
 OXEN   =0;
 GET    =0;
 HLD    =0;
 LLD    =0;
 WEN    =0;
 IENABLE=0;
 STO    =0;
 XLOAD  =0;
 HEN    =A0 AND NOT CS;
 LEN    =NOT A0 AND NOT CS;

IF NOT CS THEN GOTO R1
          ELSE GOTO IDLE;
```

```
SYSTEM Sbi;
PARAMETER
 DB,AB;

CONTACT
 CLK             :INPUT;
 RESET           :INPUT;
 EDATA[0..DB-1]:BIDIRECTIONAL;
 IDATA[0..AB-1]:BIDIRECTIONAL;
 IOR             :INPUT;
 IOW             :INPUT;
 A[0..4]         :INPUT;
 CS              :INPUT;
 HLDA            :INPUT;

 IENABLE         :OUTPUT;
 ALOAD           :OUTPUT;
 CLOAD           :OUTPUT;
 MLOAD           :OUTPUT;
 PLOAD           :OUTPUT;
 OAEN            :OUTPUT;
 OCEN            :OUTPUT;
 OMEN            :OUTPUT;
 OPEN            :OUTPUT;
 BISELECT[0..1]:OUTPUT;
 SENABLE         :OUTPUT;

USE
  DR   : DATAREG(DB:=DB,AB:=AB);
  DC   : DATACTRL;

CONNECT
 CLK=DR.CLK=DC.CLK;
 RESET=DR.RESET=DC.RESET;
 EDATA=DR.EXTDATA;
 IDATA=DR.INTDATA;
 IOR=DC.IOR;
 IOW=DC.IOW;
 A[0]=DC.A0;
 A[1..4]=DR.A[1..4];
 CS=DC.CS;
 HLDA=DC.HLDA;

 IENABLE=DC.IENABLE;
 ALOAD=DR.LOAD[0];
 CLOAD=DR.LOAD[1];
 MLOAD=DR.LOAD[2];
 PLOAD=DR.LOAD[3];
 OAEN=DR.OEN[0];
 OCEN=DR.OEN[1];
 OMEN=DR.OEN[2];
 OPEN=DR.OEN[3];
 BISELECT=DR.BISELECT;
 SENABLE=DC.SENABLE;

 DR.LEN=DC.LEN;
 DR.HEN=DC.HEN;
 DR.WEN=DC.WEN;
 DR.LLD=DC.LLD;
 DR.HLD=DC.HLD;
 DR.GET=DC.GET;
 DR.STO=DC.STO;
 DR.XLOAD=DC.XLOAD;
```

```
{**********************PRIORITY ENCODER BLOCK*****************************}
SYSTEM Codeblock;

CONTACT
  CLK                :INPUT;
  RESET              :INPUT;
  UPDLD              :INPUT;
  UPDINCOUT[0..1]    :INPUT;
  IENABLE            :INPUT;
  EXTLOAD            :INPUT;
  OPEN               :INPUT;
  DATA[0..2]         :BIDIRECTIONAL;
  UPDROTATING        :OUTPUT;
  PRIINSELECT[0..1] :OUTPUT;

USE
 L_OR      :GENERIC S$ORN(N:=2);
 DFF[0..2]:GENERIC S$D_RG1(RESETABLE:=TRUE);
 TROUT     :GENERIC S$TRBUSBUF(WORDLENGTH:=3,
                     DRIVE:=1,INVERT:=FALSE,
                     NOTINVERT:=TRUE);
 MUX[0..1]:GENERIC S$MUX2;

CONNECT
 RESET=DFF[0].RESET=DFF[1].RESET=DFF[2].RESET;
 CLK=DFF[0].CK=DFF[1].CK=DFF[2].CK;
 PRIINSELECT[0]=DFF[0].Q=TROUT.I[0];
 DATA=TROUT.Y;
 PRIINSELECT[1]=DFF[1].Q=TROUT.I[1];
 UPDROTATING=DFF[2].Q=TROUT.I[2];
 TROUT.EN=OPEN;
 UPDLD=L_OR.I[0];
 EXTLOAD=L_OR.I[1]=DFF[2].LOAD;
 L_OR.Y=DFF[0].LOAD=DFF[1].LOAD;
 UPDINCOUT=MUX[*].I[0];
 DATA[0..1]=MUX[*].I[1];
 DATA[2]=DFF[2].D;
 MUX[*].Y=DFF[0..1].D;
 MUX[0].S=MUX[1].S=IENABLE;

SYSTEM ENCODER;
CONTACT
I0,I1,
R0,R1,R2,R3 :INPUT;
O0,O1       :OUTPUT;

FUNCTION
O1=I1 AND NOT I0 AND R2 OR
   I1 AND I0 AND R3 OR
   NOT I1 AND I0 AND NOT R1 AND R2 OR
   I1 AND NOT I0 AND NOT R2 AND R3 OR
   NOT I1 AND NOT I0 AND NOT R0 AND NOT R1 AND R2 OR
   NOT I1 AND I0 AND NOT R1 AND NOT R2 AND R3 OR
   NOT I1 AND NOT I0 AND NOT R0 AND NOT R1 AND NOT R2 OR
   I1 AND I0 AND NOT R0 AND NOT R1 AND NOT R3;

O0=NOT I1 AND I0 AND R1 OR
   I1 AND I0 AND R3 OR
   NOT I1 AND NOT I0 AND NOT R0 AND R1 OR
   I1 AND NOT I0 AND NOT R2 AND R3 OR
   NOT I1 AND I0 AND NOT R1 AND NOT R2 AND R3 OR
   I1 AND I0 AND NOT R0 AND R1 AND NOT R3 OR
   NOT I1 AND NOT I0 AND NOT R0 AND NOT R1 AND NOT R2 OR
```

```
   I1 AND NOT I0 AND NOT R0 AND NOT R2 AND NOT R3;

SYSTEM PRIO_L;
CONTACT
   INSELECT[0..1]:      INPUT;
   DREQ[0..3]:          INPUT;
   MASK[0..3]:          INPUT;
   OUTSELECT[0..1]:     OUTPUT;

VAR
   I[0..1]         :MEMORY;

USE
   DM[0..3] :GENERIC S$ANDN(N:=2);
   INV[0..3]:GENERIC S$INV;
   ENC      :ENCODER;

CONNECT
   INSELECT[0]=ENC.I0;
   INSELECT[1]=ENC.I1;
   OUTSELECT[0]=ENC.O0;
   OUTSELECT[1]=ENC.O1;
   DM[0].Y=ENC.R0;
   DM[1].Y=ENC.R1;
   DM[2].Y=ENC.R2;
   DM[3].Y=ENC.R3;
  FOR I:=0 TO 3 DO
   BEGIN
    DREQ[I]=DM[I].I[0];
    INV[I].Y=DM[I].I[1];
    MASK[I]=INV[I].I;
   END;



SYSTEM Upd_1;
CONTACT
SELECT[0..1] :INPUT;
INCOUT[0..1] :OUTPUT;
FUNCTION
 INCOUT[0] = NOT SELECT[0];
 INCOUT[1] = (NOT SELECT[1] AND SELECT[0]) OR (SELECT[1] AND NOT SELECT[0]);
```

```
SYSTEM Datablock;

CONTACT
 CLK           :INPUT;
 RESET         :INPUT;
 DATA[0..2]    :TRISTATE BIDIRECTIONAL;
 EXTLOAD       :INPUT;
 OPEN          :INPUT;
 IENABLE       :INPUT;
 SELECT[0..1]  :OUTPUT;
 MASK[0..3]    :INPUT;
 DREQ[0..3]    :INPUT;
 UPDLOAD       :INPUT;
 SELOAD        :INPUT;
 UPDROT        :OUTPUT;

USE
 CDB :CODEBLOCK;
 UPD :UPD_L;
 PRI :PRIO_L;
 SRG :GENERIC S$D_REG(WORDLENGTH:=2,
                      RESETABLE:=TRUE);

CONNECT
 CLK=CDB.CLK=SRG.CK;
 RESET=CDB.RESET=SRG.RESET;
 IENABLE=CDB.IENABLE;
 EXTLOAD=CDB.EXTLOAD;
 OPEN=CDB.OPEN;
 DATA=CDB.DATA;
 CDB.PRIINSELECT=PRI.INSELECT;
 UPDROT=CDB.UPDROTATING;
 UPDLOAD=CDB.UPDLD;
 CDB.UPDINCOUT=UPD.INCOUT;
 DREQ=PRI.DREQ;
 MASK=PRI.MASK;
 PRI.OUTSELECT=SRG.D;
 SELECT=UPD.SELECT=SRG.Q;
 SELOAD=SRG.LOAD;
```

```
SYSTEM Prienc;

CONTACT
 CLK            :INPUT;
 RESET          :INPUT;
 DREQ[0..3]     :INPUT;
 MASK[0..3]     :INPUT;
 CH_END         :INPUT;
 HLDA           :INPUT;
 UPDROT         :INPUT;
 HREQ           :OUTPUT;
 CH_START       :OUTPUT;
 SELOAD         :OUTPUT;
 UPDLOAD        :OUTPUT;

FUNCTION

 TRIGGER CLK->1

 STATE Idle:

  HREQ    =0;
  CH_START=0;
  SELOAD  =0;
  UPDLOAD =0;

  IF ((DREQ AND NOT MASK)<>0) AND NOT HLDA
  THEN GOTO SO1;

 STATE SO1:

  HREQ    =1;
  CH_START=0;
  SELOAD  =0;
  UPDLOAD =0;

  IF HLDA THEN GOTO SO2
          ELSE GOTO SO1;

 STATE SO2:

  HREQ    =1;
  CH_START=0;
  SELOAD  =1;
  UPDLOAD =0;

 IF ((DREQ AND NOT MASK)=0)
 THEN GOTO IDLE
 ELSE IF UPDROT THEN GOTO UPD
                ELSE GOTO S1;

 STATE S1:

  HREQ    =1;
  CH_START=1;
  SELOAD  =0;
  UPDLOAD =0;

 IF CH_END THEN GOTO SO2
           ELSE GOTO S1;

 STATE UPD:
  HREQ    =1;
```

```
SYSTEM PRIOR_ENC;

CONTACT
 CLK           :INPUT;
 RESET         :INPUT;
 HLDA          :INPUT;
 HREQ          :OUTPUT;
 DREQ[0..3]    :INPUT;
 MASK[0..3]    :INPUT;
 CH_START      :OUTPUT;
 CH_END        :INPUT;
 SELECT[0..1]:OUTPUT;
 IENABLE       :INPUT;
 OPEN          :INPUT;
 EXTLOAD       :INPUT;
 DATA[0..2]    :TRISTATE BIDIRECTIONAL;
USE
 PRI       :PRIENC;
 DB        :DATABLOCK;

CONNECT
 CLK=PRI.CLK=DB.CLK;
 RESET=PRI.RESET=DB.RESET;
 HLDA=PRI.HLDA;
 HREQ=PRI.HREQ;
 DREQ=PRI.DREQ=DB.DREQ;
 MASK=PRI.MASK=DB.MASK;
 CH_START=PRI.CH_START;
 CH_END=PRI.CH_END;
 SELECT=DB.SELECT;
 IENABLE=DB.IENABLE;
 OPEN=DB.OPEN;
 EXTLOAD=DB.EXTLOAD;
 DATA=DB.DATA;
 DB.UPDROT=PRI.UPDROT;
 DB.UPDLOAD=PRI.UPDLOAD;
 DB.SELOAD=PRI.SELOAD;
```

```
(**********************DMA_CHANNEL BLOCK*********************************)
SYSTEM Dmactrl;

CONTACT
  CLK                          :INPUT;
  RESET                        :INPUT;
  CH_START                     :INPUT;
  CYCLE_END                    :INPUT;
  TC_ZERO                      :INPUT;
  CY_STRT                      :OUTPUT;
  A_STRB                       :OUTPUT;
  ACLOAD                       :OUTPUT;
  SET                          :OUTPUT;
  STAT                         :OUTPUT;
  MASC                         :OUTPUT;

FUNCTION
  TRIGGER CLK->1

STATE IDLE:
  CY_STRT=0;
  A_STRB      =0;
  ACLOAD      =0;
  SET         =0;
  STAT        =1;
  MASC        =1;

  IF CH_START
  THEN GOTO S1;

STATE S1:
  CY_STRT=1;
  A_STRB      =1;
  ACLOAD      =0;
  SET         =0;
  STAT        =1;
  MASC        =1;

  GOTO S2;

STATE S2:
  CY_STRT=1;
  A_STRB      =0;
  ACLOAD      =1;
  SET         =0;
  STAT        =1;
  MASC        =1;

  GOTO S3;

STATE S3:
  CY_STRT=1;
  A_STRB      =0;
  ACLOAD      =0;
  SET         =0;
  STAT        =1;
  MASC        =1;

  IF TC_ZERO
   THEN GOTO SETTC
   ELSE GOTO WAIT;

STATE SETTC:
```

```
  CY_STRT=1;
  A_STRB      =0;
  ACLOAD      =0;
  SET         =1;
  STAT        =1;
  MASC        =1;

  GOTO WAIT;

STATE WAIT:
  CY_STRT=1;
  A_STRB      =0;
  ACLOAD      =0;
  SET         =0;
  STAT        =1;
  MASC        =1;

  IF CYCLE_END
   THEN GOTO IDLE
   ELSE GOTO WAIT;
```

```
SYSTEM Status;

CONTACT
  MASK[0..3]     :OUTPUT;
  TC[0..3]       :OUTPUT;
  ODATA[0..3]    :OUTPUT;
  MODE[0..1]     :OUTPUT;
  STAT           :INPUT;
  MASC           :INPUT;
  SELECT[0..1]   :INPUT;
  SET            :INPUT;
  ILOAD          :INPUT;
  IDATA[0..3]    :INPUT;
  IENABLE        :INPUT;
  CLK            :INPUT;
  RESET          :INPUT;
  OMEN           :INPUT;
LAYOUT
  STANDARDCELL;

VAR
  I[0..1]        :MEMORY;
USE
  S[0..3],
  M[0..3]        :GENERIC S$D_RG1(RESETABLE:=TRUE);
  MD[0..3]       :GENERIC S$D_REG(WORDLENGTH:=2,
                                  RESETABLE:=TRUE);
  MUX[0..3]      :GENERIC S$MUXN(WIDTH:=4);
  DEMUX[0..1]    :GENERIC S$DEMULTI(WIDTH:=4);
  L_OR[0..3]     :GENERIC S$ORN(N:=2);
  SMUX[0..1]     :GENERIC S$MUX2;
  TRIOUTP        :GENERIC S$TRBUSBUF(WORDLENGTH:=4,DRIVE:=1,
                                     INVERT:=FALSE,
                                     NOTINVERT:=TRUE);

CONNECT
  SET=DEMUX[0].I;
  ILOAD=DEMUX[1].I;
  SELECT=DEMUX[0].S=DEMUX[1].S;
  MUX[0].I=S[*].Q;
  MUX[1].I=M[*].Q;
  MUX[2].I=MD[*].Q[0];
  MUX[3].I=MD[*].Q[1];
  TC=S[*].Q;
  MASK=M[*].Q;
  MUX[*].Y=TRIOUTP.I;
  MODE[0]=MUX[2].Y;
  MODE[1]=MUX[3].Y;
  TRIOUTP.Y=ODATA;
  TRIOUTP.EN=OMEN;
  SMUX[0].I[0]=STAT;
  SMUX[1].I[0]=MASC;
  SMUX[0].S=SMUX[1].S=IENABLE;
  SMUX[0].I[1]=IDATA[0];
  SMUX[1].I[1]=IDATA[1];

  FOR I:=0 TO 3 DO
  BEGIN
   CLK=S[I].CK=M[I].CK=MD[I].CK;
   RESET=S[I].RESET=M[I].RESET=MD[I].RESET;
   DEMUX[0].Y[I]=L_OR[I].I[0];
   DEMUX[1].Y[I]=L_OR[I].I[1]=MD[I].LOAD;
```

```
    L_OR[I].Y=M[I].LOAD=S[I].LOAD;
    SELECT=MUX[I].S;
    S[I].D=SMUX[0].Y;
    M[I].D=SMUX[1].Y;
    MD[I].D=IDATA[2..3];
  END;

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

```
SYSTEM Decrbit;
CONTACT
  CI    :INPUT;
  A     :INPUT;
  CO    :OUTPUT;
  Y     :OUTPUT;
FUNCTION
  CO = A OR CI;
  Y  =(A AND CI) OR (NOT A AND NOT CI);

SYSTEM Incrbit;
CONTACT
  CI    :INPUT;
  A     :INPUT;
  CO    :OUTPUT;
  Y     :OUTPUT;
FUNCTION
  CO = A AND CI;
  Y  =(NOT A AND CI) OR (A AND NOT CI);

SYSTEM Firstbit;
CONTACT
  A     :INPUT;
  CO    :OUTPUT;
  Y     :OUTPUT;
FUNCTION
  CO = A;
  Y  = NOT A;

SYSTEM Incrementor;
PARAMETER
  AB;
CONTACT
  A[0..AB-1]:INPUT;
  Y[0..AB-1]:OUTPUT;
LAYOUT
  STANDARDCELL;
VAR
  I[1..AB-1]:MEMORY;
USE
  FB                    : Firstbit;
  INC[1..AB-1]: Incrbit;
CONNECT
  A[0]=FB.A;
  Y[0]=FB.Y;
  INC[1].CI=FB.CO;
  FOR I:=1 TO AB-2 DO
    BEGIN
      A[I]=INC[I].A;
      Y[I]=INC[I].Y;
      INC[I+1].CI=INC[I].CO;
    END;
  A[AB-1]=INC[AB-1].A;
  Y[AB-1]=INC[AB-1].Y;

SYSTEM Decrementor;
PARAMETER
  AB;
CONTACT
  A[0..AB-1]:INPUT;
  Y[0..AB-1]:OUTPUT;
LAYOUT
  STANDARDCELL;
```

```
VAR
  I[1..AB-1]:MEMORY;
USE
  FB                    : Firstbit;
  DEC[1..AB-1]: Decrbit;
CONNECT
  A[0]=FB.A;
  Y[0]=FB.Y;
  DEC[1].CI=FB.CO;
  FOR I:=1 TO AB-2 DO
    BEGIN
      A[I]=DEC[I].A;
      Y[I]=DEC[I].Y;
      DEC[I+1].CI=DEC[I].CO;
    END;
  A[AB-1]=DEC[AB-1].A;
  Y[AB-1]=DEC[AB-1].Y;
```

```
{*****************************    dma hierarchical design***************}
SYSTEM Loadblk;
CONTACT
 ACLOAD          : INPUT;
 ALOAD           : INPUT;
 INSELECT[0..1]  : INPUT;
 RLOAD[0..3]     : OUTPUT;
USE
 A_or   : GENERIC S$ORN(N:=2);
 Ademux : GENERIC S$DEMULTI(WIDTH:=4);
CONNECT
 ACLOAD=A_OR.I[0];
 ALOAD=A_OR.I[1];
 INSELECT=ADEMUX.S;
 RLOAD=ADEMUX.Y;
 A_OR.Y=ADEMUX.I;

SYSTEM Regselect;
PARAMETER AB;
CONTACT
 INSELECT[0..1]    : INPUT;
 INTBUS[0..AB-1]   : OUTPUT;
 REG0OUT[0..AB-1]  : INPUT;
 REG1OUT[0..AB-1]  : INPUT;
 REG2OUT[0..AB-1]  : INPUT;
 REG3OUT[0..AB-1]  : INPUT;
VAR
 I[0..(LOG AB)-1] :MEMORY;
USE
 Regmux[0..AB-1] : GENERIC S$MUXN(WIDTH:=4);
CONNECT
 FOR I:=0 TO AB-1 DO
  INSELECT=REGMUX[I].S;
 INTBUS=REGMUX[*].Y;
 REG0OUT=REGMUX[*].I[0];
 REG1OUT=REGMUX[*].I[1];
 REG2OUT=REGMUX[*].I[2];
 REG3OUT=REGMUX[*].I[3];

SYSTEM Registers;
PARAMETER AB;
CONTACT
 CLK               : INPUT;
 RESET             : INPUT;
 DATIN[0..AB-1]    : INPUT;
 RLOAD[0..3]       : INPUT;
 REG0OUT[0..AB-1]  : OUTPUT;
 REG1OUT[0..AB-1]  : OUTPUT;
 REG2OUT[0..AB-1]  : OUTPUT;
 REG3OUT[0..AB-1]  : OUTPUT;
VAR
 I[0..(LOG AB)-1] :MEMORY;
USE
 Reg[0..3] : GENERIC S$D_REG(WORDLENGTH:=AB,
                             RESETABLE:=TRUE);
CONNECT
 FOR I:=0 TO 3 DO
  BEGIN
   CLK=REG[I].CK;
   RESET=REG[I].RESET;
   DATIN=REG[I].D;
  END;
 RLOAD=REG[*].LOAD;
```

```
 REG0OUT=REG[0].Q;
 REG1OUT=REG[1].Q;
 REG2OUT=REG[2].Q;
 REG3OUT=REG[3].Q;

SYSTEM Inpmux;
PARAMETER AB;
CONTACT
 IENABLE          : INPUT;
 IDATA[0..AB-1]   : INPUT;
 UPDATA[0..AB-1]  : INPUT;
 DATOUT[0..AB-1]  : OUTPUT;
VAR
 I[0..(LOG AB)-1] :MEMORY;
USE
 Amux[0..AB-1] : GENERIC S$MUX2;
CONNECT
 FOR I:=0 TO AB-1 DO
  AMUX[I].S=IENABLE;
 DATOUT=AMUX[*].Y;
 IDATA=AMUX[*].I[1];
 UPDATA=AMUX[*].I[0];
```

```
SYSTEM Datablk;
PARAMETER AB;
CONTACT
 CLK              : INPUT;
 RESET            : INPUT;
 ACLOAD           : INPUT;
 ALOAD            : INPUT;
 INSELECT[0..1]   : INPUT;
 IENABLE          : INPUT;
 IDATA[0..AB-1]   : INPUT;
 UPDATA[0..AB-1]  : INPUT;
 INTBUS[0..AB-1]  : OUTPUT;
LAYOUT
  STANDARDCELL;
USE
 Ldb : Loadblk;
 Rgs : Regselect(ab:=ab);
 Reg : Registers(ab:=ab);
 Inp : Inpmux(ab:=ab);
CONNECT
 CLK=REG.CLK;
 RESET=REG.RESET;
 ACLOAD=LDB.ACLOAD;
 ALOAD=LDB.ALOAD;
 INSELECT=LDB.INSELECT=RGS.INSELECT;
 IENABLE=INP.IENABLE;
 IDATA=INP.IDATA;
 UPDATA=INP.UPDATA;
 INTBUS=RGS.INTBUS;
 LDB.RLOAD=REG.RLOAD;
 REG.DATIN=INP.DATOUT;
 REG.REG0OUT=RGS.REG0OUT;
 REG.REG1OUT=RGS.REG1OUT;
 REG.REG2OUT=RGS.REG2OUT;
 REG.REG3OUT=RGS.REG3OUT;
```

```
SYSTEM Addrreg;
PARAMETER AB;
CONTACT
 CLK              : INPUT;
 RESET            : INPUT;
 ACLOAD           : INPUT;
 ALOAD            : INPUT;
 INSELECT[0..1]   : INPUT;
 IENABLE          : INPUT;
 IDATA[0..AB-1]   : INPUT;
 INTADDRBUS[0..AB-1] : OUTPUT;
USE
 DB   : DATABLK(AB:=AB);
 INCR : INCREMENTOR(AB:=AB);
CONNECT
 CLK=DB.CLK;
 RESET=DB.RESET;
 ACLOAD=DB.ACLOAD;
 ALOAD=DB.ALOAD;
 INSELECT=DB.INSELECT;
 IENABLE=DB.IENABLE;
 IDATA=DB.IDATA;
 INCR.Y=DB.UPDATA;
 INTADDRBUS=DB.INTBUS=INCR.A;

SYSTEM Tcreg;
PARAMETER AB;
CONTACT
 CLK              : INPUT;
 RESET            : INPUT;
 ACLOAD           : INPUT;
 CLOAD            : INPUT;
 INSELECT[0..1]   : INPUT;
 IENABLE          : INPUT;
 IDATA[0..AB-1]   : INPUT;
 INTTCBUS[0..AB-1] : OUTPUT;
USE
 DB   : DATABLK(AB:=AB);
 DECR : DECREMENTOR(AB:=AB);
CONNECT
 CLK=DB.CLK;
 RESET=DB.RESET;
 ACLOAD=DB.ACLOAD;
 CLOAD=DB.ALOAD;
 INSELECT=DB.INSELECT;
 IENABLE=DB.IENABLE;
 IDATA=DB.IDATA;
 DECR.Y=DB.UPDATA;
 INTTCBUS=DB.INTBUS=DECR.A;
```

```
SYSTEM DMA;
PARAMETER AB;
CONTACT
 CLK                : INPUT;
 RESET              : INPUT;
 ACLOAD             : INPUT;
 ALOAD              : INPUT;
 CLOAD              : INPUT;
 INSELECT[0..1]     : INPUT;
 IENABLE            : INPUT;
 IDATA[0..AB-1]     : INPUT;
 INTADDRBUS[0..AB-1] : OUTPUT;
 INTTCBUS[0..AB-1]   : OUTPUT;
 TC_ZERO             : OUTPUT;
USE
 AD : ADDRREG(AB:=AB);
 TC : TCREG(AB:=AB);
 TCNOR : GENERIC S$NORN(N:=AB);
CONNECT
 CLK=AD.CLK=TC.CLK;
 RESET=AD.RESET=TC.RESET;
 ACLOAD=AD.ACLOAD=TC.ACLOAD;
 ALOAD=AD.ALOAD;
 CLOAD=TC.CLOAD;
 INSELECT=AD.INSELECT=TC.INSELECT;
 IENABLE=AD.IENABLE=TC.IENABLE;
 IDATA=AD.IDATA=TC.IDATA;
 INTADDRBUS=AD.INTADDRBUS;
 INTTCBUS=TC.INTTCBUS;
 TC_ZERO=TCNOR.Y;
 TCNOR.I=TC.INTTCBUS;

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

```
SYSTEM TRI_OUT;

PARAMETER
  AB;
CONTACT
  INTA[0..AB-1] :INPUT;
  INTC[0..AB-1] :INPUT;
  ODATA[0..AB-1]:OUTPUT;
  OAEN                    :INPUT;
  OCEN                    :INPUT;
USE
  TRTC,TRAD: GENERIC S$TRBUSBUF(WORDLENGTH:=AB,
                                DRIVE:=1,INVERT:=FALSE,
                                NOTINVERT:=TRUE);
CONNECT
  INTA=TRAD.I;
  INTC=TRTC.I;

  ODATA=TRTC.Y=TRAD.Y;
  OCEN=TRTC.EN;
  OAEN=TRAD.EN;

SYSTEM SEL_MUX;

CONTACT
  INSELECT[0..1] :INPUT;
  BISELECT[0..1] :INPUT;
  SENABLE        :INPUT;
  SELECT[0..1]   :OUTPUT;
USE
  MUX[0..1]:GENERIC S$MUX2;
CONNECT
  MUX[0].S=MUX[1].S=SENABLE;
  MUX[*].I[0]=INSELECT;
  MUX[*].I[1]=BISELECT;
  MUX[*].Y=SELECT;
```

```
SYSTEM DMA_CH;

PARAMETER
  AB;

CONTACT
  CLK                      :INPUT;
  RESET                    :INPUT;
  CH_START                 :INPUT;
  CYCLE_END                :INPUT;
  INSELECT[0..1]           :INPUT;
  BISELECT[0..1]           :INPUT;
  SENABLE                  :INPUT;
  ALOAD                    :INPUT;
  CLOAD                    :INPUT;
  ILOAD                    :INPUT;
  IENABLE                  :INPUT;
  OAEN                     :INPUT;
  OCEN                     :INPUT;
  OMEN                     :INPUT;
  DATA[0..AB-1] :TRISTATE BIDIRECTIONAL;

  INTADDRBUS[0..AB-1]:OUTPUT;
  CY_STRT                  :OUTPUT;
  A_STRB                       :OUTPUT;
  MASK[0..3]                   :OUTPUT;
  TC[0..3]                     :OUTPUT;
  MODE[0..1]                   :OUTPUT;
  CH_END                       :OUTPUT;
  OUTSELECT[0..1]              :OUTPUT;

USE
  CTR:       DMACTRL;
  DMA:       DMA(AB:=AB);
  STA:       STATUS;
  OTRI:      TRI_OUT(AB:=AB);
  MUX:       SEL_MUX;
CONNECT
  CLK=CTR.CLK=DMA.CLK=STA.CLK;
  RESET=CTR.RESET=DMA.RESET=STA.RESET;
  A_STRB=CTR.A_STRB;
  CH_START=CTR.CH_START;
  CYCLE_END=CTR.CYCLE_END=CH_END;
  MUX.SELECT=DMA.INSELECT=STA.SELECT;
  INSELECT=OUTSELECT=MUX.INSELECT;
  BISELECT=MUX.BISELECT;
  SENABLE=MUX.SENABLE;
  ALOAD=DMA.ALOAD;
  CLOAD=DMA.CLOAD;
  ILOAD=STA.ILOAD;
  IENABLE=DMA.IENABLE=STA.IENABLE;
  DATA=DMA.IDATA;
  DATA[0..3]=STA.IDATA;
  INTADDRBUS=DMA.INTADDRBUS=OTRI.INTA;
  OTRI.INTC=DMA.INTTCBUS;
  CTR.TC_ZERO=DMA.TC_ZERO;
  CY_STRT=CTR.CY_STRT;
  MASK=STA.MASK;
  TC=STA.TC;
  MODE=STA.MODE;
  STA.ODATA=DATA[0..3];
  CTR.ACLOAD=DMA.ACLOAD;
  CTR.SET=STA.SET;
```

```
{**********************TRANSFER CONTROL BLOCK*****************************}

SYSTEM TRANSF_CTRL;


CONTACT
CLK           :INPUT;
RESET         :INPUT;
MODE[0..1]    :INPUT;
CY_STRT   :INPUT;
READY         :INPUT;

DACK          :OUTPUT;
BUSEN         :OUTPUT;
R[0..1]       :OUTPUT;
W[0..1]       :OUTPUT;
CYCLE_END     :OUTPUT;

FUNCTION
  TRIGGER CLK->1

STATE Idle:
R=3;
W=3;
DACK=0;
BUSEN=0;
CYCLE_END=0;

IF CY_STRT
THEN GOTO S1;

STATE S1:
R=3;
W=3;
DACK=0;
BUSEN=1;
CYCLE_END=0;

 IF MODE=1 THEN GOTO S2R
           ELSE
           IF MODE=2 THEN GOTO S2W
                     ELSE GOTO S2V;

STATE S2V:
R=3;
W=3;
DACK=1;
BUSEN=1;
CYCLE_END=0;

GOTO S3V;

STATE S3V:
R=3;
W=3;
DACK=1;
BUSEN=1;
CYCLE_END=0;

GOTO S4V;

STATE S4V:
R=3;
```

```
W=3;
DACK=1;
BUSEN=1;
CYCLE_END=0;

IF READY THEN GOTO S5V
         ELSE GOTO S4V;

STATE S5V:
R=3;
W=3;
DACK=1;
BUSEN=1;
CYCLE_END=1;

GOTO S6;

STATE S2R:
R=1;
W=3;
DACK=1;
BUSEN=1;
CYCLE_END=0;

GOTO S3R;

STATE S3R:
R=1;
W=1;
DACK=1;
BUSEN=1;
CYCLE_END=0;

GOTO S4R;

STATE S4R:
R=1;
W=1;
DACK=1;
BUSEN=1;
CYCLE_END=0;

IF READY THEN GOTO S5R
         ELSE GOTO S4R;

STATE S5R:
R=1;
W=3;
DACK=1;
BUSEN=1;
CYCLE_END=1;

GOTO S6;

STATE S2W:
R=2;
W=3;
DACK=1;
BUSEN=1;
CYCLE_END=0;

GOTO S3W;

STATE S3W:
```

```
R=2;
W=2;
DACK=1;
BUSEN=1;
CYCLE_END=0;

GOTO S4W;

STATE S4W:
R=2;
W=2;
DACK=1;
BUSEN=1;
CYCLE_END=0;

IF READY THEN GOTO S5W
         ELSE GOTO S4W;

STATE S5W:
R=2;
W=3;
DACK=1;
BUSEN=1;
CYCLE_END=1;

GOTO S6;

STATE S6:
R=3;
W=3;
DACK=1;
BUSEN=1;
CYCLE_END=1;

IF NOT CY_STRT THEN GOTO Idle
                    ELSE GOTO S6;
```

```
SYSTEM TRANSF_BLK;
PARAMETER
  AB;

CONTACT
 CLK                          :INPUT;
 RESET                        :INPUT;
 INTADDR[0..AB-1]:INPUT;
 MODE[0..1]                   :INPUT;
 SELECT[0..1]                 :INPUT;
 CY_STRT                 :INPUT;
 A_STRB                       :INPUT;
 READY                        :INPUT;

 ADDRESS[0..AB-1]:TRISTATE OUTPUT;
 DACK[0..3]                   :OUTPUT;
 R[0..1]                      :TRISTATE OUTPUT;
 W[0..1]                      :TRISTATE OUTPUT;
 CYCLE_END                    :OUTPUT;

USE
   TRF    : TRANSF_CTRL;
   TRIBUF: GENERIC S$TRBUSBUF(
                                WORDLENGTH:=AB,
                                DRIVE:=1,
                                INVERT:=FALSE,
                                NOTINVERT:=TRUE);
   TRIC   : GENERIC S$TRBUSBUF(
                                WORDLENGTH:=4,
                                DRIVE:=1,
                                INVERT:=FALSE,
                                NOTINVERT:=TRUE);
   DEMUX : GENERIC S$DEMULTI(WIDTH:=4);
   LATCH : GENERIC S$D_REG(WORDLENGTH:=AB,
                           RESETABLE:=TRUE);
   SREG   : GENERIC S$D_REG(WORDLENGTH:=2,
                           RESETABLE:=TRUE);

CONNECT
   RESET=TRF.RESET=SREG.RESET=LATCH.RESET;
   CLK=TRF.CLK=SREG.CK=LATCH.CK;
   A_STRB=LATCH.LOAD=SREG.LOAD;
   LATCH.D=INTADDR;
   TRIBUF.I=LATCH.Q;
   TRIBUF.EN=TRIC.EN=TRF.BUSEN;
   TRIBUF.Y=ADDRESS;
   TRIC.I[0..1]=TRF.R;
   TRIC.I[2..3]=TRF.W;
   TRIC.Y[0..1]=R;
   TRIC.Y[2..3]=W;
   Cy_strt=TRF.Cy_strt;
   Cycle_end=TRF.Cycle_end;
   Ready=TRF.Ready;
   MODE=TRF.MODE;
   DEMUX.I=TRF.DACK;
   SREG.D=SELECT;
   DEMUX.S=SREG.Q;
   DEMUX.Y=DACK;
```

```
{**********************FINALDMACONTROLLER BLOCK**************************}

SYSTEM Finalctrl;
PARAMETER
 AB, DB;

CONTACT
 CLK              :INPUT;
 RESET            :INPUT;

 EDATA[0..DB-1]   :BIDIRECTIONAL;
 IOR              :INPUT;
 IOW              :INPUT;
 A[0..4]          :INPUT;
 CS               :INPUT;

 HLDA             :INPUT;
 DREQ[0..3]       :INPUT;
 HREQ             :OUTPUT;

 TC[0..3]         :OUTPUT;

 READY            :INPUT;
 ADDRESS[0..AB-1]:OUTPUT;
 DACK[0..3]       :OUTPUT;
 R[0..1]          :OUTPUT;
 W[0..1]          :OUTPUT;

USE
 SBI       : SBI(AB:=AB,DB:=DB);
 PRI       : PRIOR_ENC;
 DMA       : DMA_CH(AB:=AB);
 TRA       : TRANSF_BLK(AB:=AB);

CONNECT
 CLK=SBI.CLK=PRI.CLK=DMA.CLK=TRA.CLK;
 RESET=SBI.RESET=PRI.RESET=DMA.RESET=TRA.RESET;

 EDATA=SBI.EDATA;
 IOR=SBI.IOR;
 IOW=SBI.IOW;
 A=SBI.A;
 CS=SBI.CS;

 HLDA=SBI.HLDA=PRI.HLDA;
 DREQ=PRI.DREQ;
 HREQ=PRI.HREQ;

 TC=DMA.TC;

 READY=TRA.READY;
 ADDRESS=TRA.ADDRESS;
 DACK=TRA.DACK;
 R=TRA.R;
 W=TRA.W;

 SBI.IENABLE=PRI.IENABLE=DMA.IENABLE;
 SBI.IDATA[0..2]=PRI.DATA[0..2];
 SBI.IDATA=DMA.DATA;
 SBI.ALOAD=DMA.ALOAD;
 SBI.CLOAD=DMA.CLOAD;
 SBI.MLOAD=DMA.ILOAD;
 SBI.PLOAD=PRI.EXTLOAD;
```

```
SBI.OAEN=DMA.OAEN;
SBI.OCEN=DMA.OCEN;
SBI.OMEN=DMA.OMEN;
SBI.OPEN=PRI.OPEN;
SBI.BISELECT=DMA.BISELECT;
SBI.SENABLE=DMA.SENABLE;

PRI.MASK=DMA.MASK;
PRI.CH_START=DMA.CH_START;
PRI.CH_END=DMA.CH_END;
PRI.SELECT=DMA.INSELECT;

DMA.CYCLE_END=TRA.CYCLE_END;
DMA.INTADDRBUS=TRA.INTADDR;
DMA.CY_STRT=TRA.CY_STRT;
DMA.A_STRB=TRA.A_STRB;
DMA.MODE=TRA.MODE;
DMA.OUTSELECT=TRA.SELECT;
```

```
{******************MAIN BLOCK INCL. PAD'S EN PARAMETERS*****************}

SYSTEM MAIN;
USE
   FINALCTRL : FINALCTRL ( AB <- 16 , DB <- 8 );
   DACK[0..3]:      GENERIC S$OPAD;
   CLK:             GENERIC S$IPAD;
   HLDA:            GENERIC S$IPAD;
   EDATA[0..7]:     GENERIC S$BPAD;
   A[0..4]:         GENERIC S$IPAD;
   READY:           GENERIC S$IPAD;
   W[0..1]:         GENERIC S$OPAD;
   RESET:           GENERIC S$IPAD;
   R[0..1]:         GENERIC S$OPAD;
   IOW:             GENERIC S$IPAD;
   IOR:             GENERIC S$IPAD;
   HREQ:            GENERIC S$OPAD;
   ADDRESS[0..15]: GENERIC S$OPAD;
   DREQ[0..3]:      GENERIC S$IPAD;
   TC[0..3]:        GENERIC S$OPAD;
   CS:              GENERIC S$IPAD;

CONNECT
   DACK[*].I    =FINALCTRL.DACK;
   CLK.Y        =FINALCTRL.CLK;
   HLDA.Y       =FINALCTRL.HLDA;
   EDATA[*].Y   =EDATA[*].I=FINALCTRL.EDATA;
   EDATA[0].EN =EDATA[1].EN=EDATA[2].EN=EDATA[3].EN=IOR.Y;
   EDATA[4].EN =EDATA[5].EN=EDATA[6].EN=EDATA[7].EN=IOR.Y;
   A[*].Y       =FINALCTRL.A;
   READY.Y      =FINALCTRL.READY;
   W[*].I       =FINALCTRL.W;
   RESET.Y      =FINALCTRL.RESET;
   R[*].I       =FINALCTRL.R;
   IOW.Y        =FINALCTRL.IOW;
   IOR.Y        =FINALCTRL.IOR;
   HREQ.I       =FINALCTRL.HREQ;
   ADDRESS[*].I=FINALCTRL.ADDRESS;
   DREQ[*].Y    =FINALCTRL.DREQ;
   TC[*].I      =FINALCTRL.TC;
   CS.Y         =FINALCTRL.CS;
```