Een multifunctionele I/O-bouwsteen

Eindhoven
University of Technology
the Netherlands

Department of
Electrical Engineering

Een multifunctionele I/O-bouwsteen

door

M.P.J. Stevens
en
M.P.M. van Loon

EUT - Report 84-E-143
ISBN - 90-6144-143-9
ISSN - 0167-9708
Augustus 1984

Eindhoven University of Technology Research Reports
EINDHOVEN UNIVERSITY OF TECHNOLOGY

Department of Electrical Engineering
Eindhoven The Netherlands

EEN MULTIFUNCTIONELE I/O-BOUWSTEEN

door

M.P.J. Stevens

en

M.P.M. van Loon

EUT Report 84-E-143
ISBN 90-6144-143-9
ISSN 0167-9708
Coden: TEUEDE

Eindhoven
Augustus 1984

CIP-GEGEVENS KONINKLIJKE BIBLIOTHEEK, DEN HAAG

Stevens, M.P.J.

Een multifunctionele I/O-bouwsteen / door M.P.J. Stevens en M.P.M. van Loon. - Eindhoven: University of Technology. - Fig. - (Eindhoven University of Technology research report / Department of Electrical Engineering, ISSN 0167-9708; 84-E-143)
Met lit. opg., reg.
ISBN 90-6144-143-9
SISO 664.2 UDC 681.325.65.02 UGI 650
Trefw.: computers / hardware-ontwerp.

INHOUDSOPGAVE.

Abstract

In this report, the design of a multifunction I/O-controller is described. This device is used for the exchange of data between a microcomputer and a peripheral. Much attention is paid to the design strategies needed for the realization of (V)LSI-circuits. By partitioning of the controller into subsystems and carefully defining the interaction between these subsystems, the design can be implemented later in a number of technologies like custom designed NMOS or gate-arrays. The controller can be used with many Intel and Motorola processors and it has four programmable modes: direct I/O, handshake I/O, three wire I/O and bidirectional I/O.

Stevens, M.P.J. and M.P.M. van Loon

DESIGN OF A MULTIFUNCTION I/O-CONTROLLER. In Dutch.
Department of Electrical Engineering, Eindhoven University of Technology (Netherlands), 1984.
EUT Report 84-E-143

Address of the authors:

ir. M.P.J. Stevens,
Group of Digital Systems,
Department of Electrical Engineering,
Eindhoven University of Technology,
P.O. Box 513,
5600 MB EINDHOVEN,
The Netherlands

INLEIDING.

Dit rapport beschrijft een studie betreffende het ontwerp van een VLSI chip, de zogenaamde multifunction I/O-controller (MFIOC). Dit is een interfacebouwsteen, die parallelle data-uitwisseling tussen een microcomputer en een randapparaat moet verzorgen. Het doel van de studie is de ontwerpstrategie, die nodig is voor het realiseren van (V)LSI circuits te leren kennen en toegankelijk te maken voor de studenten van de T.H.Eindhoven. Het gaat hierbij niet om het maken van een zo klein of zo snel mogelijke bouwsteen. Het belangrijkste doel is de architectuur van de bouwsteen zo optimaal mogelijk te maken gezien vanuit de hoek van de systeemarchitectuur. Dit wil zeggen dat er een aantal functionele subsystemen gezocht moeten worden, die elk een logische functie realiseren. Door de onderlinge interacties tussen deze subfuncties goed op te zetten kan een bouwsteen gemaakt worden, die later via diverse technologieen geimplementeerd kan worden. Hierbij wordt gedacht aan bijvoorbeeld een custom designed NMOS realisatie, een TI gate array en een Philips gate array.
Het doel is om uitgaande van een aantal wensen en eisen te komen tot een "net" ontwerp dat bestaat uit een aantal afzonderlijke modules, die parallel werken. Bij het ontwerp zijn een aantal wensen als uitgangspunt genomen. Of deze wel of niet te realiseren zijn bij een gekozen technologie zal later bepalen of deze wensen al of niet over boord worden gezet.
De bouwsteen moet hierbij voor diverse I/O-protocollen geschikt zijn. De protocollen moeten voor zover dit relevant is zowel in polled mode als in een op interrupts gebaseerde mode toegepast kunnen worden.
De bouwsteen moet zo worden opgezet dat hij direct te koppelen is met een aantal veel gebruikte microprocessoren. Hierbij wordt gedacht aan de microprocessoren van Intel (8080, 8085, 8086 en 8088), Motorola (6800, 6802, 6809 en 68000). Hierbij komen verschillende bussystemen naar voren, die via het aanbrengen van een externe selectie gekozen moeten kunnen worden.

Uit de bovenstaande uitgangspunten zijn een aantal basismodulen in de bouwsteen te onderscheiden.
Allereerst is er een bidirectioneel datapad. Dit pad vormt de verbinding tussen de databus (koppeling met een processor) en de I/O-bus (koppeling met een randapparaat). De overdracht van data langs de I/O-bus vindt plaats onder besturing van de modulen input-control en output-control. Deze modulen kunnen een verscheidenheid van I/O-protocollen afhandelen. De gebruiker van de bouwsteen kan een protocol selecteren door programmering van het mode-control register. De informatie in dit register wordt door een mode-control decoder omgezet in besturingssignalen voor input- en output control. Indien gewenst kan de gebruiker via de sts-lijnen statusinformatie verkrijgen van de I/O-logica.

Tot slot is er een module die aangesloten wordt op de controlbus van de processor. Dit controlblok krijgt read/write-opdrachten met de bijbehorende chipselect signalen. Deze gegevens worden gebruikt voor de besturing van het:

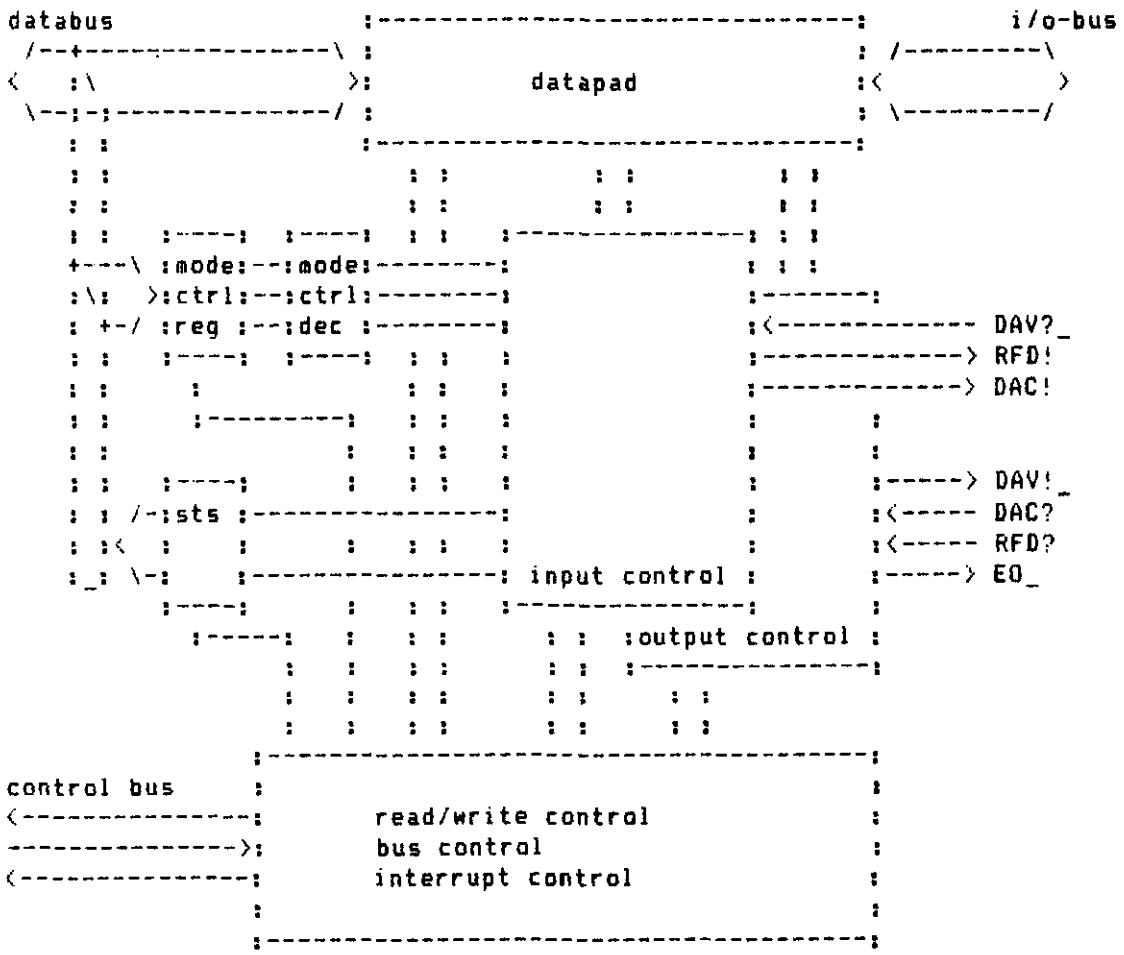
lezen van statusinformatie;

schrijven in mode-control register;

lezen en schrijven van I/O-data via het datapad.

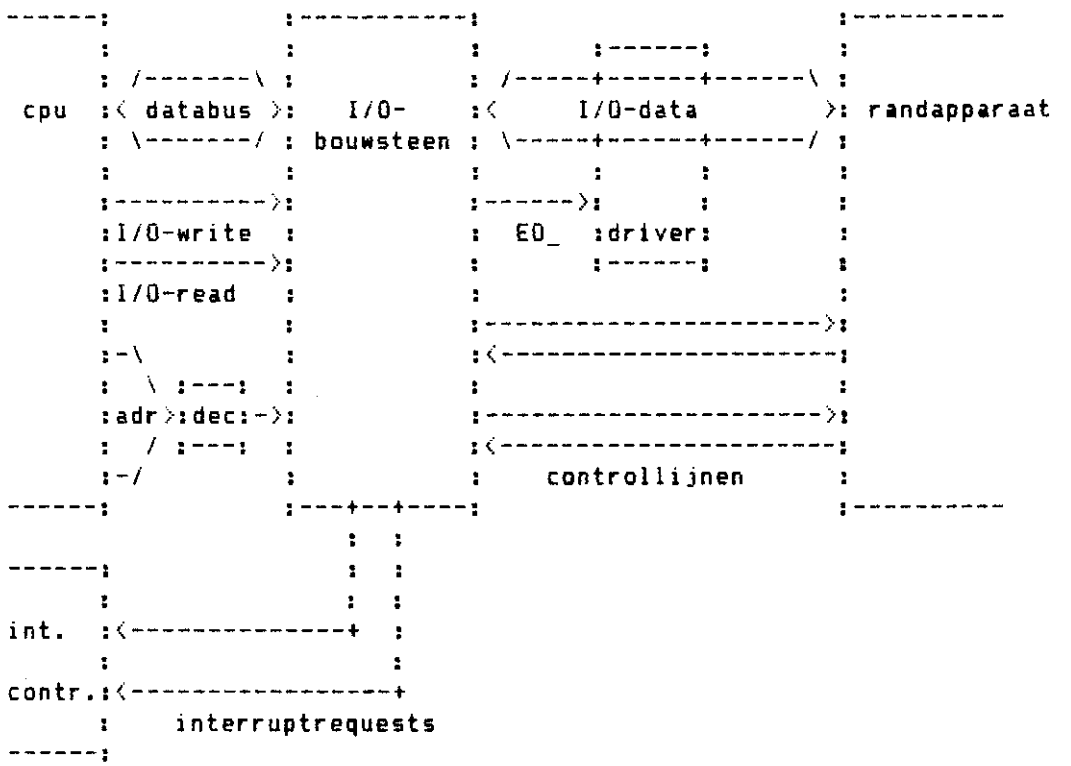
De lees- en schrijfoperaties die onder het laatste punt vallen, kunnen op basis van interrupts worden afgewikkeld. De interrupt requests worden door de I/O-control gegenereerd, en hierna door de controlmodule doorgegeven.

Onderstaande figuur geeft de functionele opbouw weer van de controller. Uitgaande signalen aan de zijde van de I/O-bus hebben een "!" meegekregen, ingaande een "?". Een 'underline' (_) geeft aan dat het signaal actief laag is.



De hardware van de controller komt uitvoerig aan de orde in hoofdstuk 3. De eerste twee hoofdstukken beschrijven de I/O-protocollen, die de bouwsteen moet kunnen verwerken.

De I/O-bouwsteen is aan de ene kant doorverbonden met de CPU en de interruptcontroller en aan de andere kant wordt een verbinding gemaakt met het te besturen I/O-device. Onderstaande afbeelding geeft de koppeling weer.


cpu
databus
I/O-bouwsteen
I/O-data
randapparaat
I/O-write
I/O-read
adr
dec
EO_
driver
controllijnen
int. contr.
interruptrequests

1. BESPREKING VAN DE I/O-PROTOCOLLEN.

Om de bouwstenen universeel inzetbaar te maken voor vele soorten I/O-interfaces moeten de volgende I/O-protocollen gerealiseerd kunnen worden:

Mode 1: directe output
2: directe input
3: fully interlocked handshake output
4: fully interlocked handshake input
5: three wire handshake output
6: three wire handshake input
7: bidirectionele I/O (master)
8: bidirectionele I/O (slave)

De keuze van de gewenste I/O-mode moet gebeuren door een besturingswoord naar het mode-controlregister te sturen. Initieel zal door de bouwsteen altijd de directe inputmode geselecteerd worden. Dit om te zorgen dat er extern geen ongewenste acties gestart kunnen worden in de randapparaten. Als vanuit de initiele mode overgegaan wordt op een andere mode, moet er voor gezorgd worden dat er geen signaalwisselingen optreden op de externe lijnen. Hieruit volgen dan enkele eisen voor de datalijnen. Dit komt bij de betreffende protocollen aan de orde.

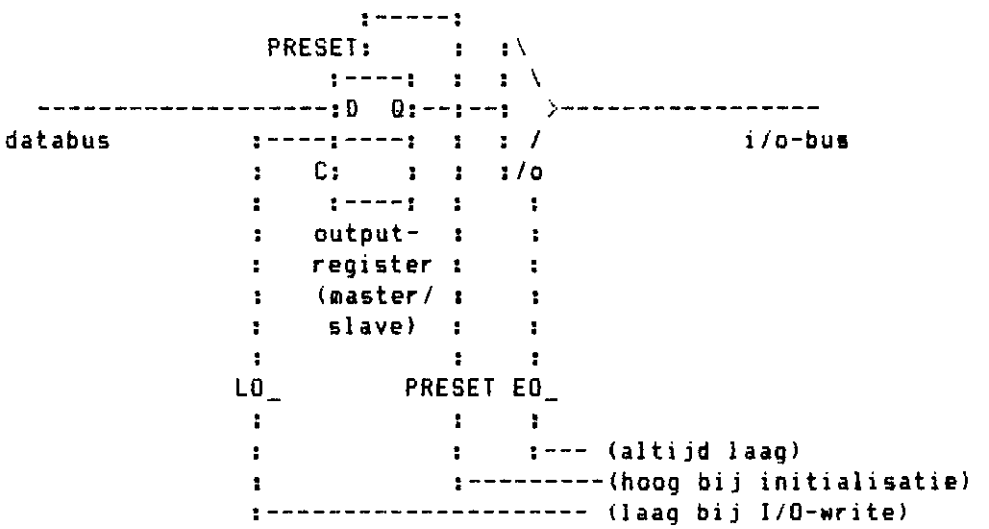
1.1. Directe outputmode.

De functie van de I/O-bouwsteen bij directe output is de data, die door de processor door middel van een outputinstructie naar buiten wordt gestuurd, in het outputregister te klokken en direct via de I/O-pennen door te laten naar een randapparaat of proces.

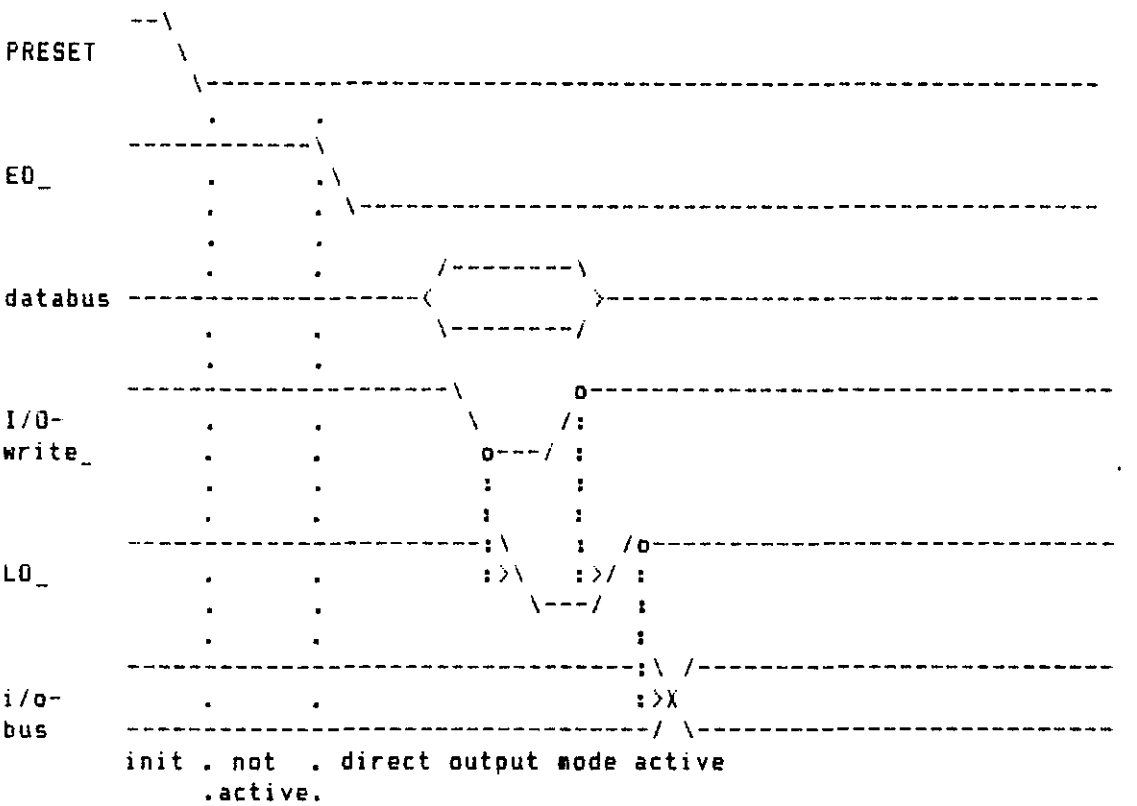
Als er een datawoord vanuit de CPU aangeboden wordt aan de I/O-bouwsteen is tegelijk met de data het I/O-adres (CE_ = chipenable) en het I/O-write signaal actief. De besturing in de I/O-bouwsteen moet uit deze signalen een intern signaal (LO_ = load outputregister) maken dat de data in het outputregister klokt. Om deze functie te kunnen realiseren is een outputregister nodig dat een klokingang heeft, waarmee de data in de flip-floppen van het register geklokt kan worden. Bovendien is het nodig dat het outputregister een presetingang heeft, zodat de uitgangen initieel op "1" kunnen worden geschakeld. Zo kunnen overgangsglitches voorkomen worden bij het omschakelen van de inputmode (waarbij de lijnen hoogohmig zijn) naar de outputmode. Hierdoor zullen externe TTL-schakelingen geen signaalverandering zien.
Omdat met deze bouwsteen zowel input- als outputfuncties moeten worden gerealiseerd, is het nodig om het outputregister af te kunnen schermen van de I/O-pennen. Tijdens input mag namelijk door de inhoud van het outputregister de inputdata niet verstoord worden. Om dit te kunnen realiseren moet een intern I/O-bussysteem gerealiseerd worden met tri-state drivers. Bij directe output moet de tri-state poort altijd open staan om de data "direct" door te laten naar de I/O-pennen. Dit wordt gerealiseerd door de tri-state poort te besturen met een signaal (EO_ = enable output) dat altijd actief is (d.w.z. EO_ = 0) wanneer de directe outputmode gekozen is.

De besturing in de I/O-bouwsteen zal bij keuze van de directe outputmode als eerste het EO_-signaal laag moeten maken zodat er outputdata naar de I/O-pennen kan gaan. Daarna moet het I/O-writesignaal (I/O-write plus het adres van de datapoort) bekeken worden door de interne besturing. Is I/O-write laag actief dan moet het LO_-signaal actief ("0") worden, zodat er data in het master gedeelte van de master/slave flip-floppen van het outputregister wordt gezet. Als I/O-write inactief hoog wordt, zal LO_ ook hoog moeten worden. D.w.z. bevriezen van de outputdata en deze doorlaten van de master naar de slave flip-flop. Dit proces zal zich herhalen zolang de directe outputmode actief is.

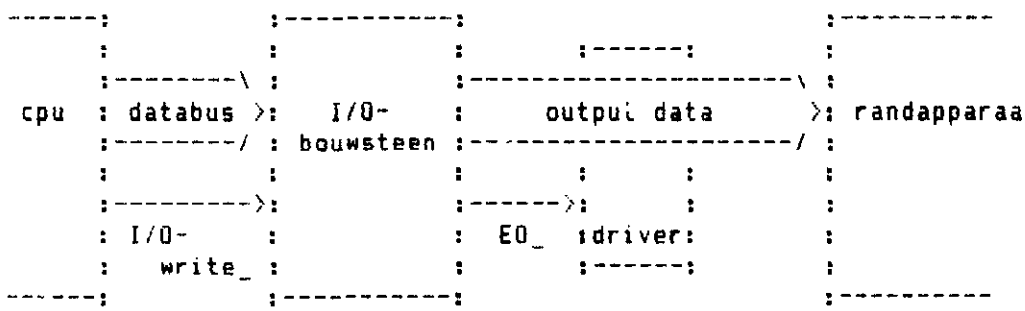
Het datapad voor directe output zal de structuur hebben, zoals in de onderstaande afbeelding is weergegeven.



Onderstaande afbeelding toont de timing voor directe outputmode. De interne besturing van de I/O-bouwsteen zal de signaalovergangen moeten realiseren. Eerst zien we de initialisatie (power on en reset) gevolgd door een inactieve periode (i.v.m. de automatische keuze van de directe inputmode), waarna de directe outputmode gekozen wordt



Onderstaande afbeelding geeft de koppeling weer tussen de I/O-bouwsteen en een randapparaat in geval van directe output. Er is geen enkel besturingssignaal tussen de bouwsteen en het randapparaat bij de directe outputmode. De interne besturingslijn EO_ is wel naar buiten uitgevoerd, omdat hiermee externe (power-)drivers kunnen worden geactiveerd. Deze lijn is laag actief.

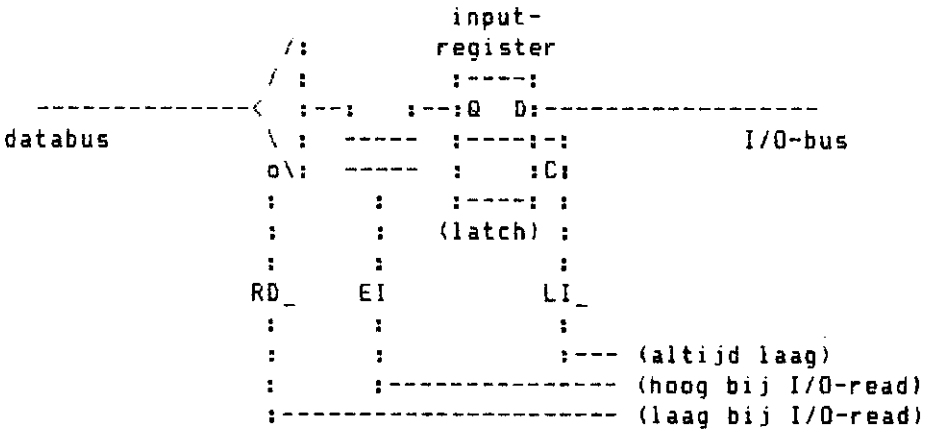


1.2. Directe inputmode.

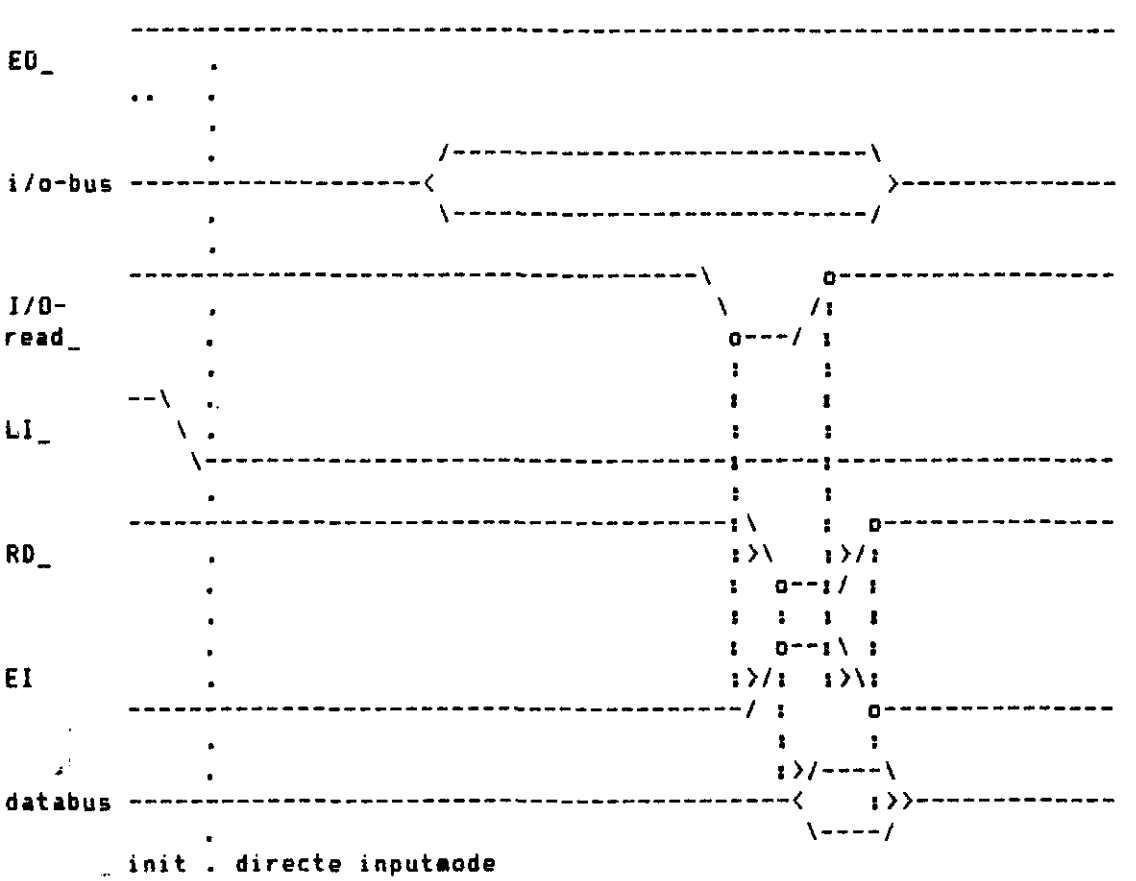
De functie van de I/O-bouwsteen bij directe input is de data, die op de I/O-pennen van de I/O-bouwsteen staat tijdens de uitvoering van een inputinstructie te bemonsteren en de gevonden waarde direct door te laten naar de databus van de processor.

Zodra er een inputinstructie door de CPU uitgevoerd wordt, moet er data naar de CPU doorgelaten worden. Hierbij zijn op de I/O-bouwsteen het I/O-adres en het I/O-read signaal actief. Door de besturing in de I/O-bouwsteen moet uit deze signalen een intern signaal (EI = enable input) gemaakt worden zodat de data, die op de I/O-pennen staat, doorgelaten wordt naar de databus van de processor. Om deze functie te realiseren gebruiken we een tri-state poort, die op het juiste ogenblik door het RD_-signaal opengezet kan worden. Deze tri-state poort krijgt data via een pass transistor, welke bestuurd wordt door het EI-signaal.
Omdat met deze bouwsteen zowel directe als gebufferde input moet kunnen worden uitgevoerd, zal er naast de tri-state poort ook nog een inputregister aanwezig zijn. Tijdens de directe inputmode zal dit register de data door moeten laten. Door voor het register latch flip-floppen te kiezen, is het mogelijk de data zowel door te laten als in te klokken. De latch flip-floppen besturen we door middel van het LI_-signaal (LI_= load inputregister). Voor de directe inputmode moeten we het LI_-signaal activeren zodat de flip-floppen open staan, d.w.z. LI_ = 0.
Bij het activeren van de directe inputmode moet door de besturing voor de output van de I/O-bouwsteen het signaal dat de tri-state poort voor output beheert (EO_) inactief ("1") maken. Daarnaast moeten de flip-floppen van het inputbuffer op doorlaten worden geschakeld (LI_ = 0). Nu is de initialisatie achter de rug en kan een lus uitgevoerd worden waarin het I/O-readsignaal (I/O-read plus adres van de datapoort) door de besturing bekeken wordt. Is I/O-read actief, dan moet data doorgelaten worden (EI = 1) en wordt de tri-state poort open gezet naar de databus. Als I/O-read inactief is moet EI laag zijn.

Uit de voorgaande overwegingen volgt het datapad voor directe input, dat hieronder weergegeven is.

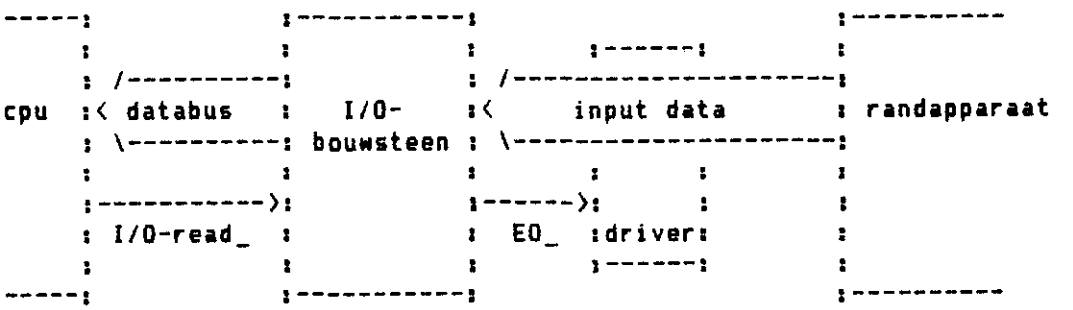


Onderstaande diagram geeft de timing voor directe inputmode weer. Eerst zien we de initialisatie met daarna de directe inputmode. Deze wordt immers automatisch gekozen.



Onderstaande schema geeft de koppeling van de i/o-bouwsteen met een randapparaat weer.

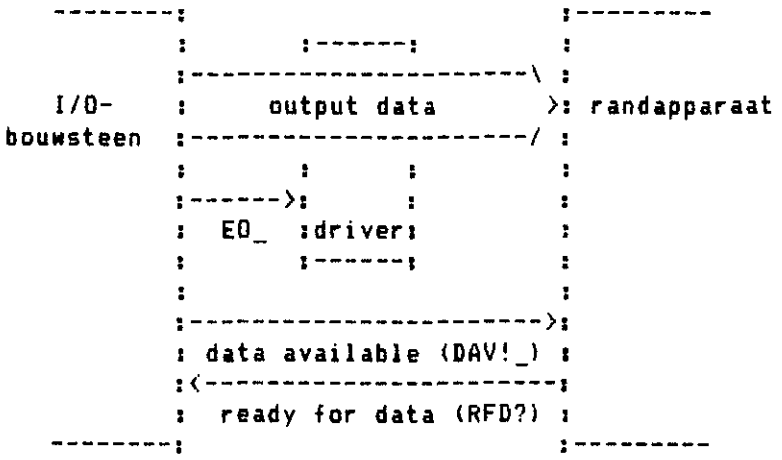
EO_ is de besturingslijn voor externe drivers.



1.3. Fully interlocked handshake output.

Bij handshake I/O wordt de data vanuit de I/O-bouwsteen naar een randapparaat of een andere I/O-bouwsteen overgedragen in een vraag en antwoordspel. Hierbij wordt via een aantal besturingslijnen een reeks signaalovergangen gegenereerd, die zorgen dat zowel bron (source) als bestemming (acceptor) stap voor stap de data vragen, opwekken, naar buiten sturen, overnemen en weer nieuwe data vragen. Om deze interacties mogelijk te maken, zijn voor de fully interlocked handshake outputmode naast het datapad twee signaallijnen nodig, namelijk RFD?(= ready for data) en DAV!_ (= data available). De lijn RFD? gaat vanuit de bestemming naar de bron en geeft achtereenvolgens aan wanneer om data wordt gevraagd en wanneer de geleverde data is overgenomen. RFD? laag wil zeggen dat er niet om data wordt gevraagd. Hoog is te beschouwen als een vraag om nieuwe data aan de bron. Vanuit de bron loopt de signaallijn DAV!_ naar de bestemming om aan te geven wanneer er geldige data aangeboden wordt en wanneer er om nieuwe data mag worden gevraagd. DAV!_ hoog wil zeggen dat er om data gevraagd mag worden. DAV!_ laag betekent dat er nu geldige data aangeboden wordt.

Het aansluitschema van de I/O-bouwsteen met een randapparaat ziet er als volgt uit:



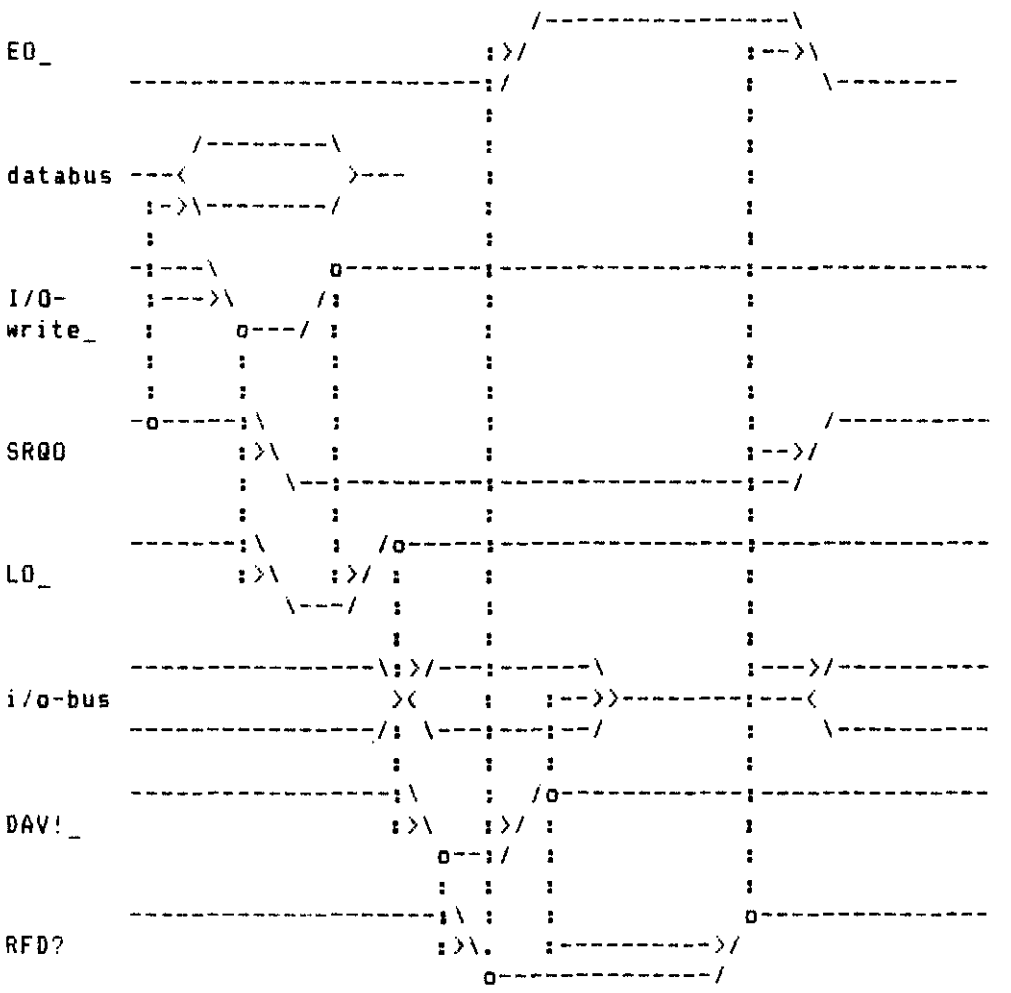
De hardware in het datapad van de I/O-bouwsteen is in de voorgaande paragrafen reeds besproken. Voor output hebben we te maken met een register van master/slave-flip-floppen gevolgd door tri-state poorten. De flip-floppen moeten om de reeds genoemde reden een presetingang hebben. Voor input hebben we latch-flip-floppen nodig, die transparant geschakeld kunnen worden.

De besturing van de I/O-bouwsteen zal via het DAV!_-signaal ("1") aan de bestemming vertellen wanneer de outputmode operationeel is. Dus bij het opstarten van de bouwsteen wordt automatisch DAV!_ gelijk aan "0" gemaakt om de buitenwereld te vertellen dat er nog niet om data mag worden gevraagd. Tevens is EO_=1, dus de tri-state poorten voor output zijn dicht. Is de bouwsteen in de fully interlocked handshake outputmode geschakeld, dan wordt als eerste actie DAV!_ hoog. Het outputregister is nu nog leeg. De bestemming mag hierop reageren door RFD?= 1 te maken. Dit wil zeggen: er mag data gezonden worden. De besturing reageert hierop met het openen van de tri-state poorten (EO_=0). Bovendien wordt aan de processor om data gevraagd, want er is geen geldige data in het outputregister aanwezig. Dit gebeurt door middel van

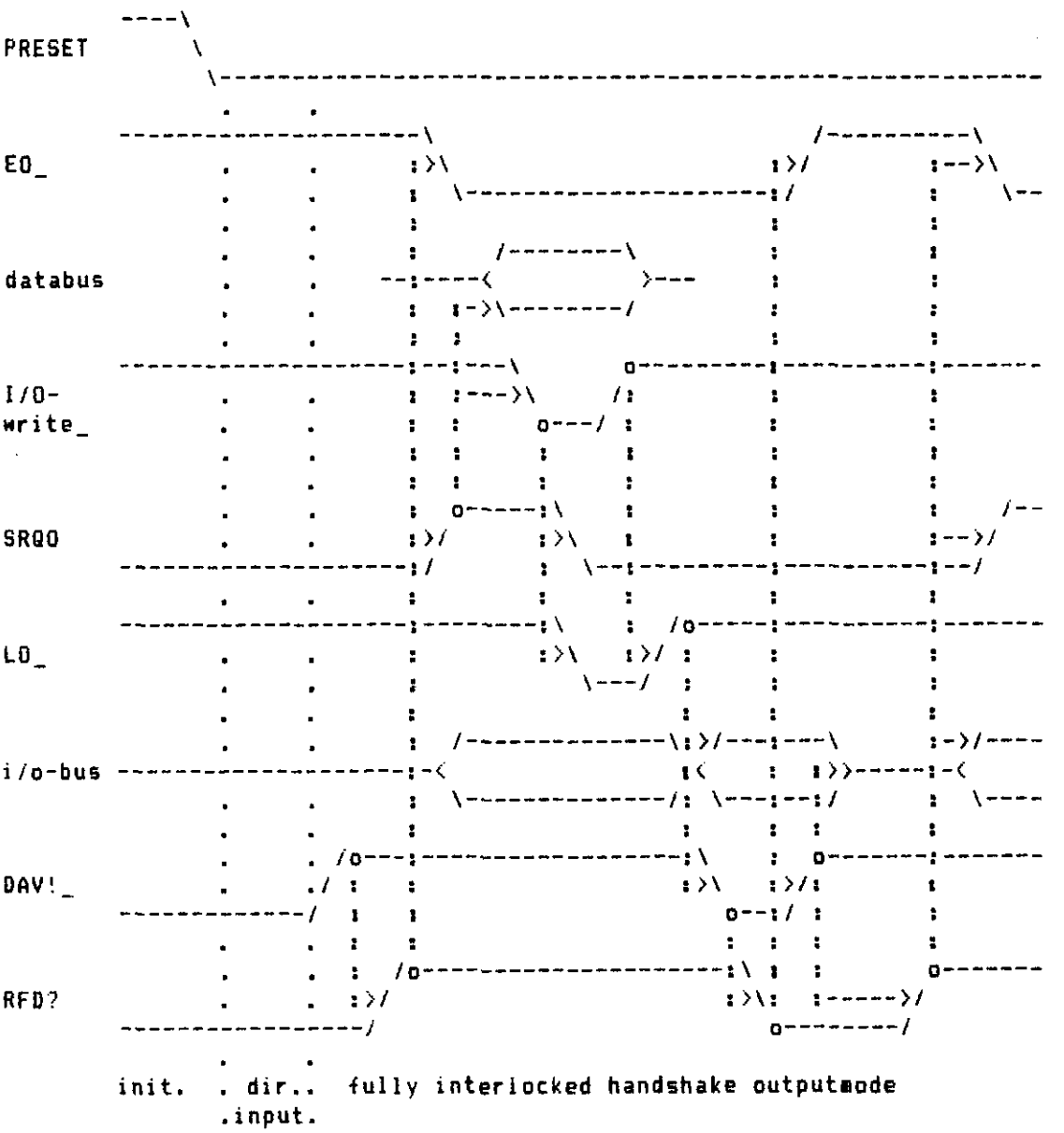
het zogenaamde output service request signaal (SRQO). Dit is een signaal dat aangesloten kan worden op een interruptingang van de processor. Een interrupt service programma zal, als reactie op de interrupt, data in het outputregister schrijven (write data = I/O-write plus adres van de data poort). De besturing in de I/O-bouwsteen moet hierbij naar het I/O-writesignaal van de processor kijken, en hieruit het LO_-signaal (load outputregister) afleiden. Zodra het I/O-writesignaal actief wordt, zal de besturing de service request (interrupt request) weg moeten nemen. Nadat het I/O-writesignaal weer hoog is geworden, zal de data van de masterflip-flop naar de slave worden geklokt door het hoog maken van LO_. De data komt nu extern beschikbaar. De stap, die hierop volgt, is het melden aan de bestemming dat er geldige data in het outputregister (en dan ook op de I/O-pennen) staat (DAV!_ = 0).
De bestemming kan nu de data van de I/O-pennen overnemen (copy) in zijn inputregister. Na het inklokken van de data wordt de voortgang gemeld door de vraag om data weg te nemen. Hierbij wordt RFD? laag gemaakt. De bestemming zal daarna intern de data verder processen. Dit is echter helemaal niet van belang voor de bron van de data. De bron heeft nu als taak te melden dat er om nieuwe data mag worden gevraagd. Dit wordt gedaan door het signaal DAV!_ hoog te maken. Hiermee is tevens de vorige data ongeldig geworden (verklaard). We bevinden ons nu weer in de uitgangssituatie.

We zien dat het fully interlocked handshake protocol zo steeds automatisch aan de CPU om de volgende byte zal vragen. Deze wordt door middel van het interrupt programma geleverd etc. Er komt nooit een einde aan. Ook niet als er geen data meer beschikbaar is. Vaak is het gewenst om deze zogenaamde blocktransfer te onderbreken. We gebruiken hiervoor een intern besturingssignaal (via het mode control register programmeerbaar) dat we de naam Continue/Non Wait Out (C_WOUT) geven. Als dit signaal hoog is, mag om de volgende byte worden gevraagd, die dan weer via een interrupt wordt geleverd. Willen we stoppen, dan kan dat door in de interrupt service routine net voor het naar buiten sturen van de laatste byte een WAIT-commando in het mode controlregister van de I/O-bouwsteen te zetten (C_WOUT = 0). Daarna schrijven we de laatste byte in het outputregister. De besturing zal deze byte via het normale outputprotocol naar buiten sturen. Om te voorkomen dat er weer om data wordt gevraagd, zal de besturing echter niet meer melden dat er om de volgende data mag worden gevraagd. De DAV!_-lijn wordt nu namelijk laag gehouden. Wanneer we wel verder willen gaan, kunnen we het mode controlregister van WAIT naar CONTINUE omschakelen (C_WOUT = 1) door er het juiste commandowoord naar toe te sturen. Daarna wordt DAV!_ weer hoog en de output verloopt weer normaal. Om te kunnen wachten, moeten we in de interruptroutines de nodige maatregelen nemen.

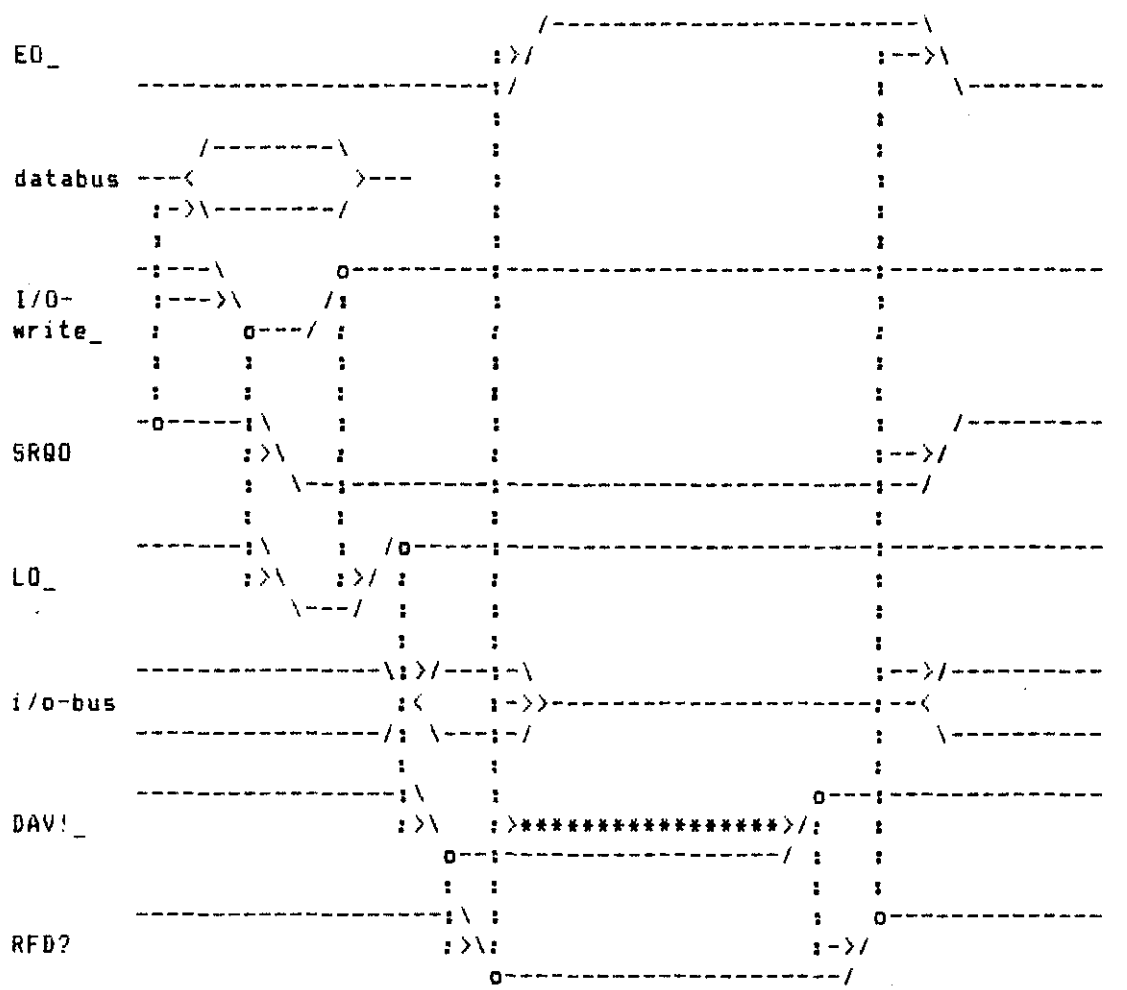
De timing voor een de fully interlocked handshake outputmode verloopt als volgt:



Bij de overgang van initialisatie via de directe inputmode naar de fully interlocked handshake outputmode krijgen we de volgende signaalovergangen:



Hier onder wordt een cyclus weergegeven waarin CONTINUE/WAIT geactiveerd wordt. Nadat een gehele cyclus afgemaakt is wordt het DAV_-signaal tijdelijk opgehouden. Hierdoor ontstaat een WAIT-state. Tijdens deze state is EO_ inactief.

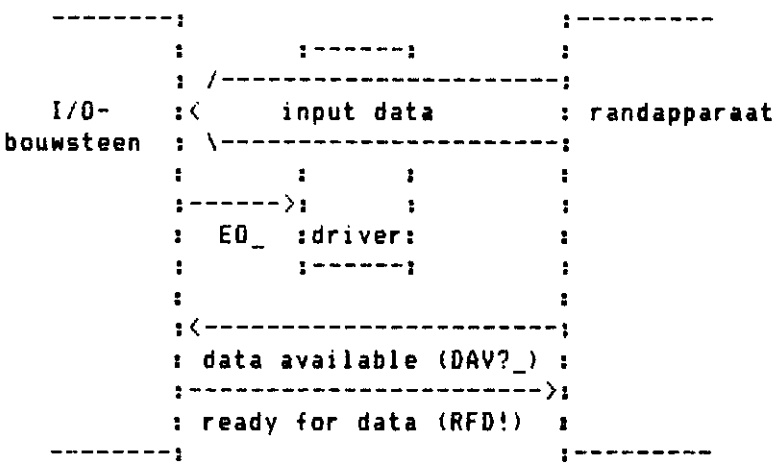


****
WAIT-state

1.4. Fully interlocked handshake input.

De tegenhanger van fully interlocked handshake output is fully interlocked input. We hebben bij handshake altijd met twee partners te maken. De ene realiseert de input en de ander de output. De besturingssignalen van de bron en van de bestemming zullen hierbij identiek moeten zijn.
Bij deze input-mode werken we met gebufferde input. Dit wil zeggen dat een sample van de data op de I/O-pennen niet meer direct doorgelaten wordt naar de databus. De inputdata wordt nu eerst in een inputregister geklokt. Hierna wordt een interrupt request signaal gegenereerd, waarna de processor in een service routine de data over zal nemen.

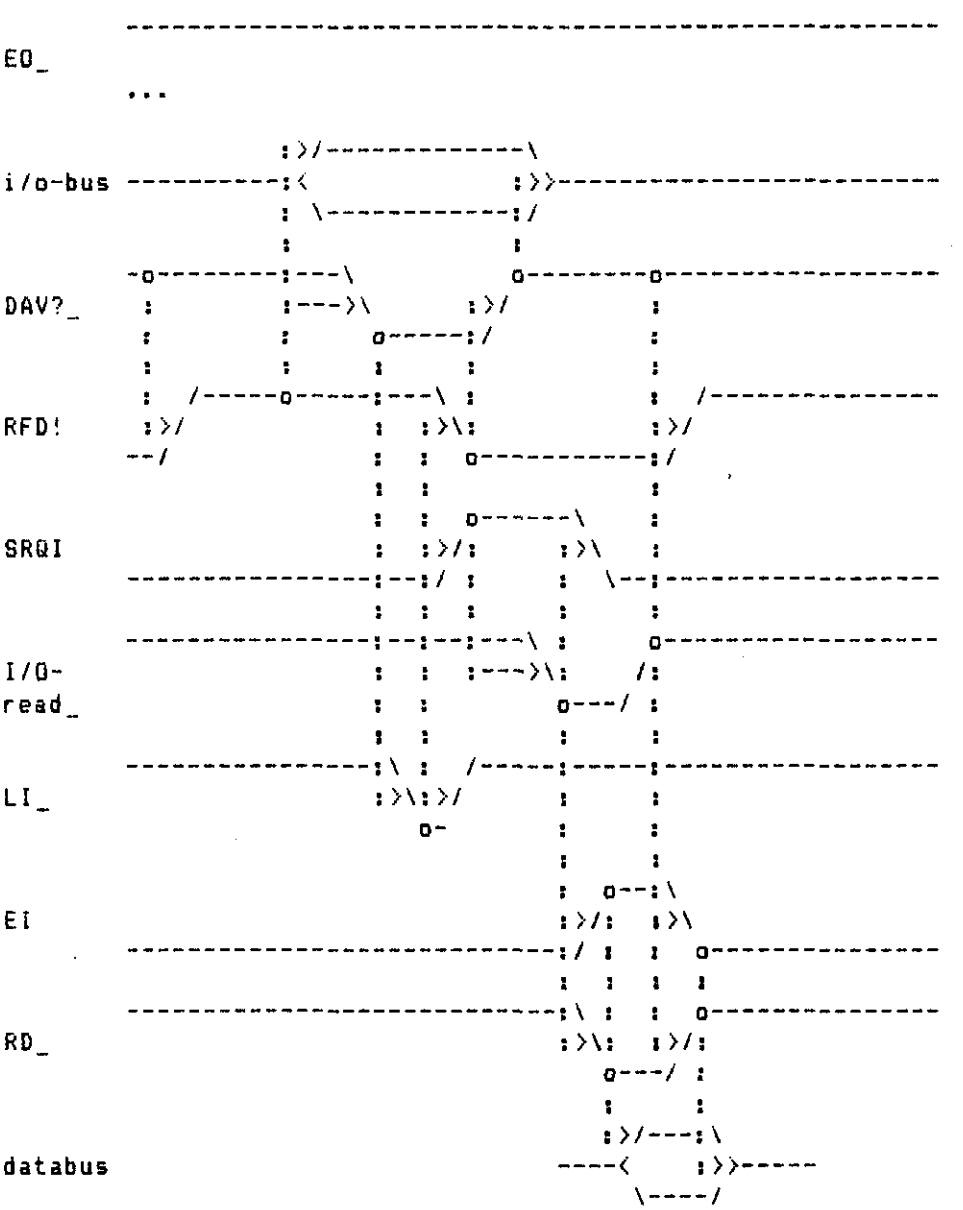
Onderstaande afbeelding toont de aansluiting van een randapparaat met een fully interlocked inputpoort.



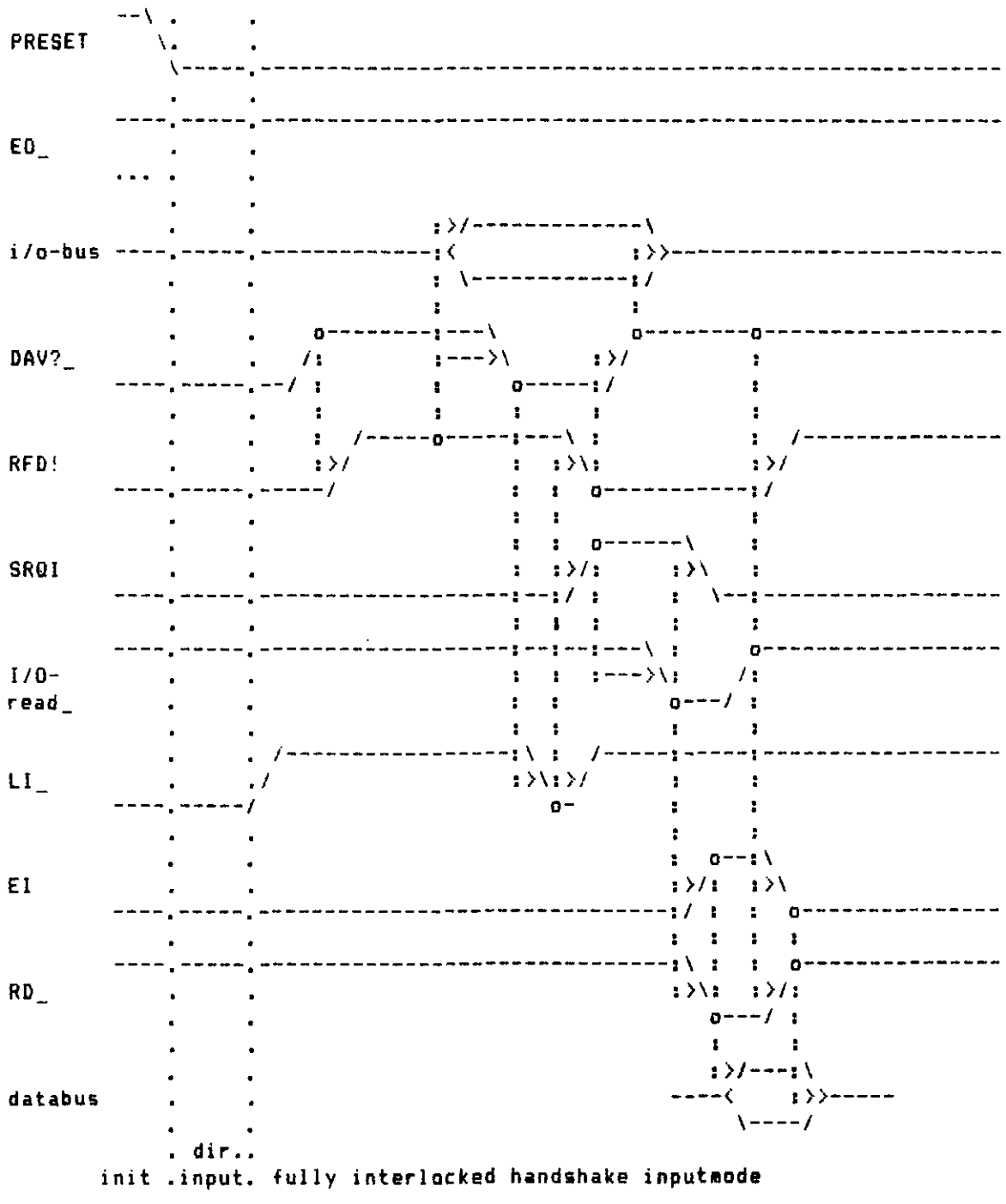
Als op fully interlocked input overgegaan wordt vanuit de directe inputmode blijft output via het datapad verboden. De besturing heeft hiervoor de tri-state poort dicht gezet door het EO_-signaal hoog te maken. De besturing moet wachten tot het betreffende outputapparaat klaar is voor actie. Als het outputapparaat klaar is om data te leveren, wordt dit aan de bestemming medegedeeld door het DAV?_-signaal hoog te maken. Het is nu de beurt aan de besturing van de inputpoort om aan de bron data te vragen. Dit gebeurt door het RFD!-signaal hoog te maken. De inputpoort moet nu wachten op de bron. Deze moet namelijk eerst de data ophalen en op de datalijnen zetten. Wanneer dit klaar is, ontvangt de inputbesturing de melding dat er geldige data op de datalijnen staat. Dit wordt vanuit de bron aangegeven door DAV?_ = 0. De volgende stappen zijn het inklokken van de data in het inputregister en het terugmelden dat de data overgenomen is, zodat die dan weggenomen mag worden door de bron (RFD!= 0). Het randapparaat reageert hierop door de data ongeldig te verklaren (DAV?_= 1). Tegelijk met RFD!= 0 wordt een interrupt request (SRQI = 1) gegenereerd aan de processorzijde. Deze zal een I/O-read (I/O-read plus adres van de data poort) tot gevolg hebben. Als I/O-read laag is, maakt de besturing het EI hoog en tevens RD_ en het interrupt request signaal laag. Omdat EI hoog en RD_ laag is, wordt nu de data doorgelaten naar de processor. Zodra het I/O-readsignaal wegvalt, wordt ook de tri-state poort naar de databus gesloten (RD_ hoog en EI laag). We zijn dan terug in de uitgangspositie. Ook nu kunnen we weer doorgaan met de volgende byte, maar we hebben ook weer de mogelijkheid om tijdelijk te wachten. Dit kan via een Continue/Non Wait In (C_WIN) signaal in het mode control register, dat de

inputcyclus tijdelijk op kan houden. Als C_WIN=0, dan zal na afloop van de protocolcyclus RFD! niet meer hoog worden. Hierdoor ontstaat een WAIT-state.

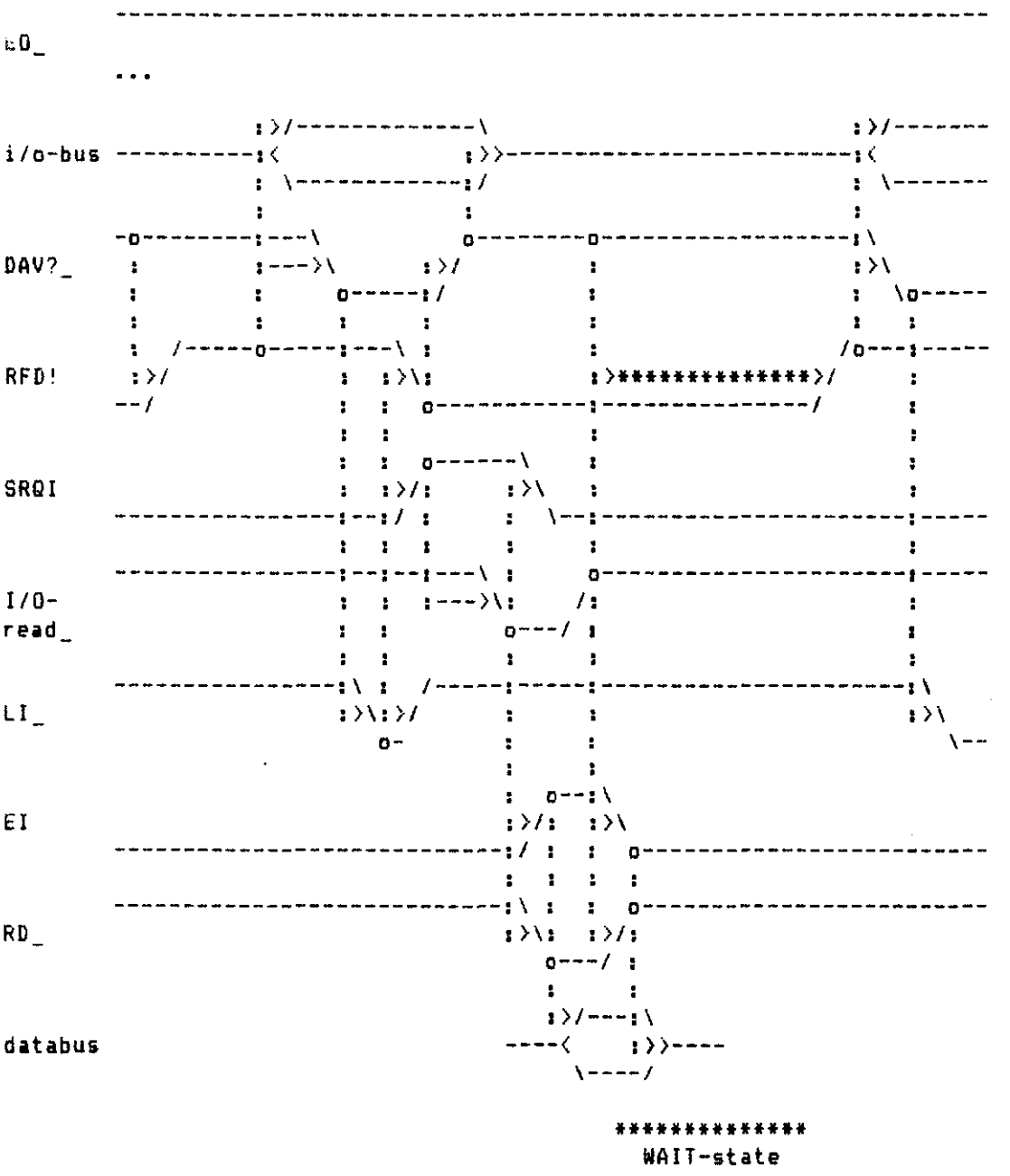
Onderstaande diagram toont de timing voor handshake inputmode.



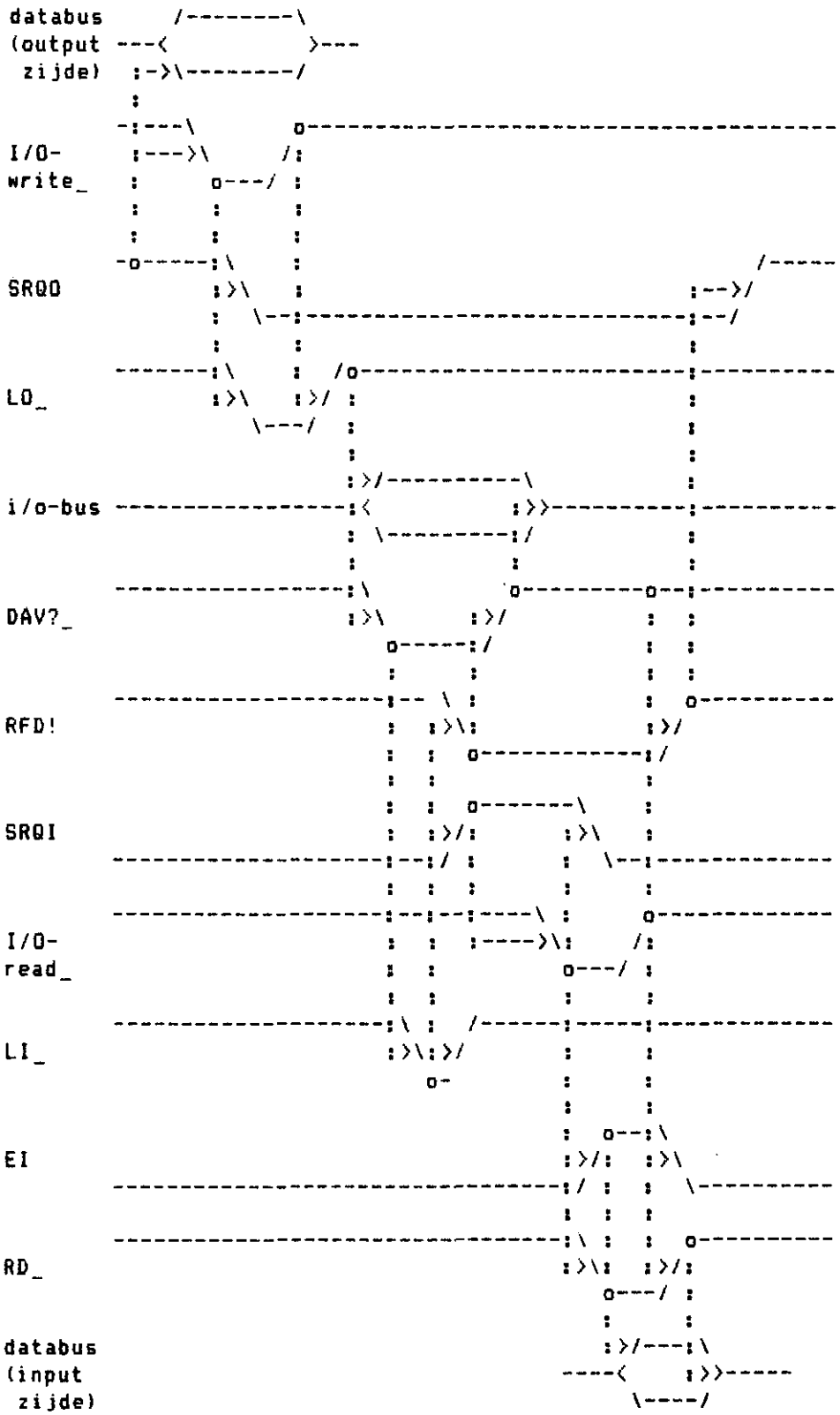
Bij overgang van initialisatie via direct input naar fully interlocked handshake input krijgen we het volgende beeld :



Als C_WIN=0, dan ontstaat een WAIT-state doordat RFD! niet meer hoog wordt na afloop van een protocolcyclus :



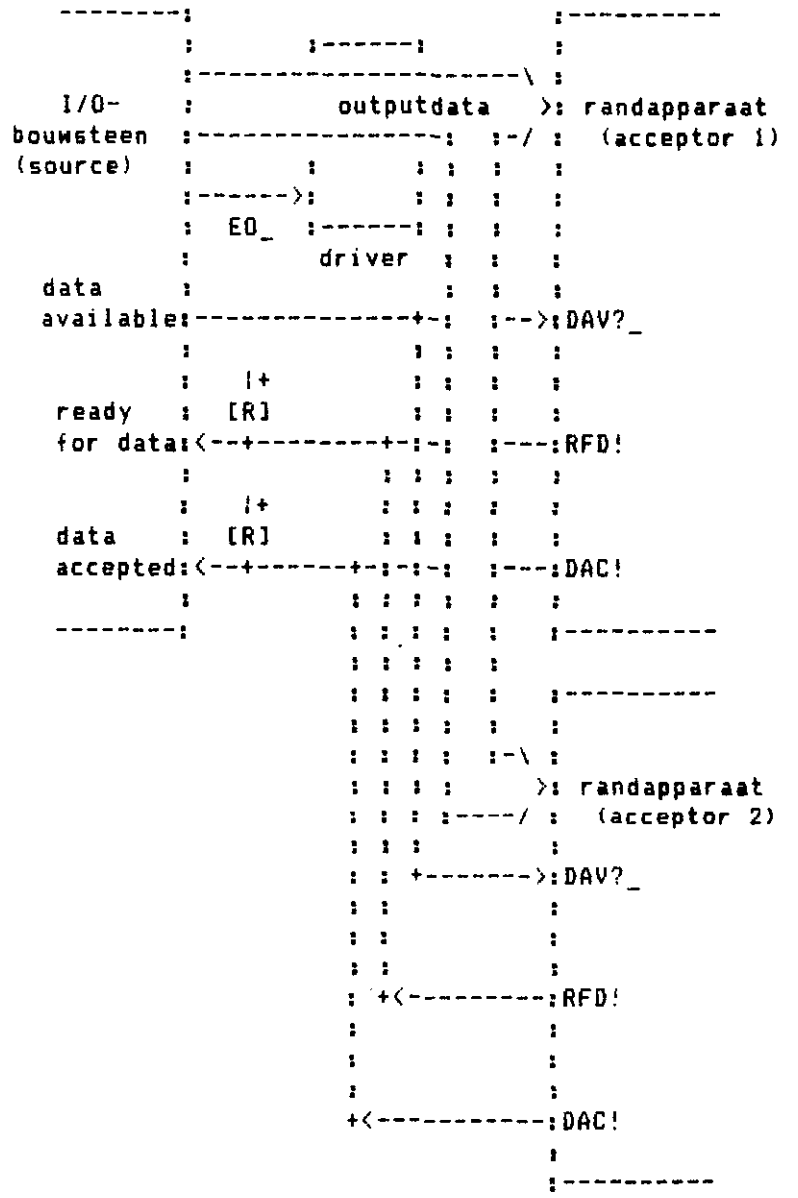
Onderstaande afbeelding geeft weer hoe de afwikkeling aan beide kanten werkt. Het toont hoe de input- en output via interrupts opgestart wordt en hoe daarna de terugmeldingen verlopen.



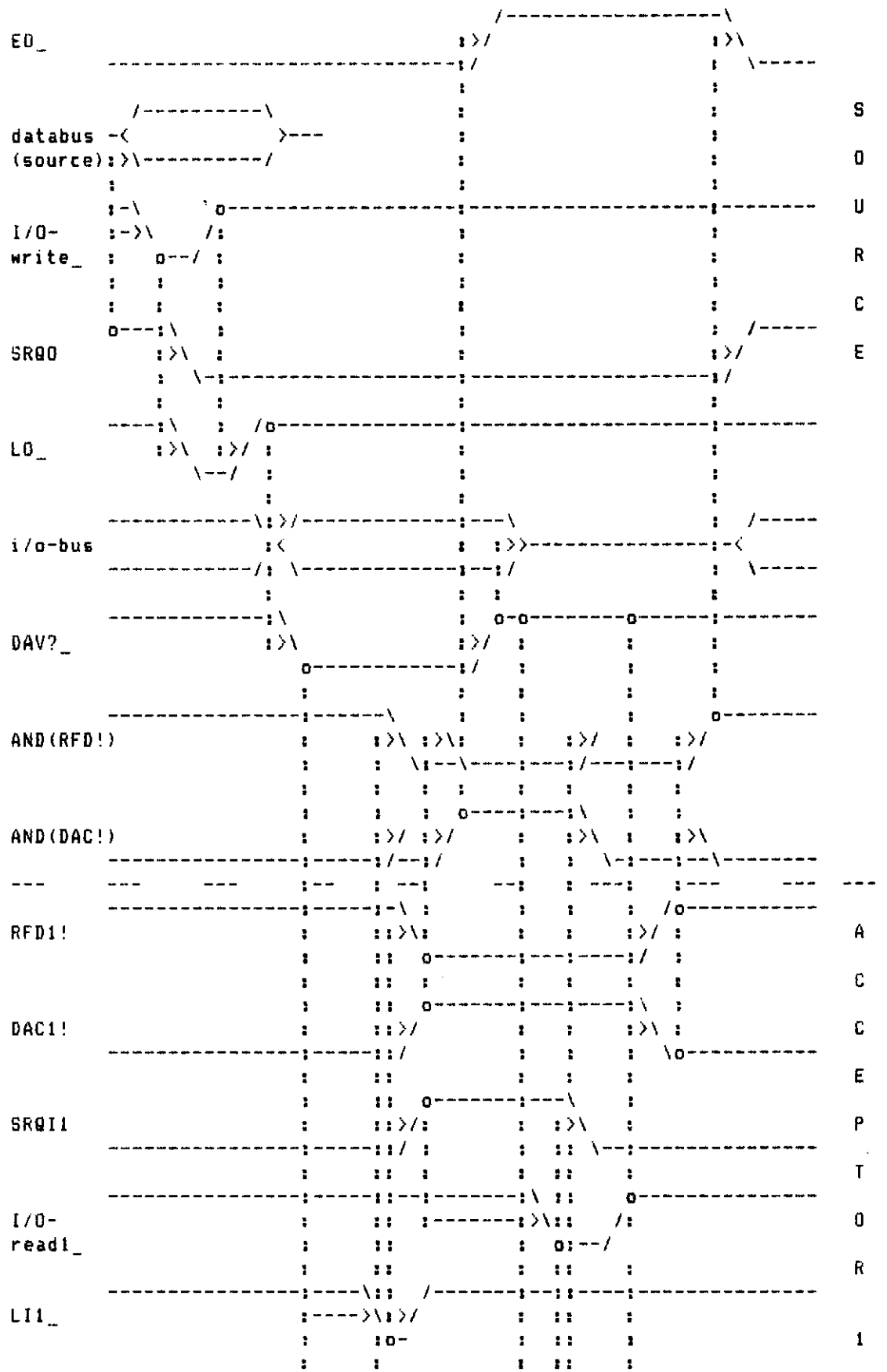
1.5. Three wire handshake input en output.

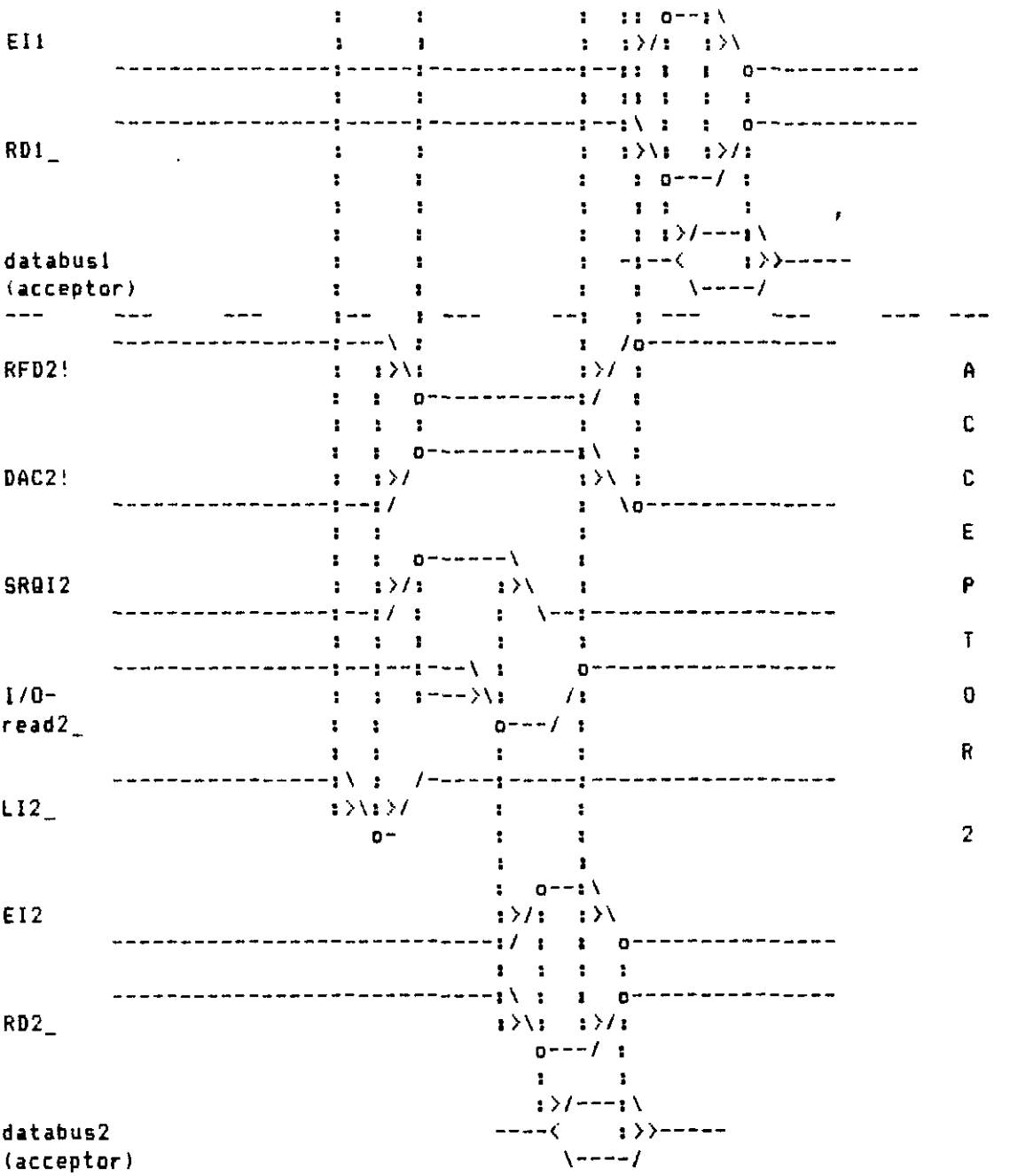
De functie van three wired handshake is het data uitwisselen vanuit een enkele bron (source) naar meerdere bestemmingen (acceptoren). We willen hierbij werken met hetzelfde protocol als bij normale handshake input en output. Dit zou betekenen dat elke acceptor een afzonderlijke terugmeldingslijn (RFD!) zou moeten hebben om aan te geven wanneer er data gewenst is en wanneer de aangeboden data overgenomen is. Dit zou een groot aantal RFD!-lijnen betekenen, met alle aansluitmoeilijkheden van dien. Een oplossing is het combineren van de RFD!-signalen door middel van een wired-AND schakeling. Hierbij worden alle RFD!-lijnen, die via een open drain naar buiten komen door middel van een weerstand naar de positieve voedingsspanning met elkaar doorverbonden. Als een van de RFD!-signalen laag is, zal de combinatie ook laag zijn. Er treden hier echter enkele problemen op. De source kan namelijk met acceptoren samen moeten werken, die elk een verschillende reactietijd hebben. Dit wil zeggen dat de gecombineerde RFD!-lijn pas weer hoog wordt als alle aangesloten lijnen hoog zijn. Dit is precies wat we willen. De acceptoren zullen ieder hun RFD!-lijn laag maken om aan te geven dat de data overgenomen is. Hier ontstaat het probleem dat de snelste acceptor zal bepalen wanneer de gecombineerde RFD!-lijn laag wordt. De source zal de data daarna wegnemen. Dit is echter niet gewenst, omdat de langzame acceptoren de data nog niet overgenomen hebben. We zouden in feite een gecombineerde RFD!-lijn willen hebben, die hoog wordt als alle acceptoren om data vragen en pas weer laag wordt als alle acceptoren de data overgenomen hebben. Deze functie is niet te realiseren met een enkele (wired- AND) lijn. De oplossing is vrij eenvoudig. We passen twee lijnen toe. De ene lijn is de normale RFD!-lijn, die een laag naar hoog overgang maakt als alle acceptoren klaar zijn om data te ontvangen. De andere lijn is de inverse van de RFD!-lijn. Deze heeft de naam DAC! (data accepted) gekregen. Deze lijn gaat van laag naar hoog als een acceptor de data overgenomen heeft. Door deze lijn ook weer via een wired-AND te combineren, zal bij de source de combinatie pas hoog worden als de langzaamste acceptor de data ingeklokt heeft. De source zal eerst steeds testen of alle acceptoren data willen hebben, d.w.z. testen op het hoog zijn van de gecombineerde RFD!-lijn. Daarna wordt data naar buiten gestuurd. De volgende stap is het wachten op het hoog worden van de gecombineerde DAC!-lijn. De rest van het protocol verloopt identiek aan de normale fully interlocked handshake. De I/O-bouwsteen kan zowel als source als acceptor worden geprogrammeerd.

Onderstaande afbeelding toont hoe twee acceptoren aangesloten worden op een source.



In onderstaande timing diagram is de signaal-uitwisseling tussen een source en twee acceptoren weergegeven. De eerste acceptor is hierbij trager dan de tweede acceptor. De gecombineerde signalen hebben bij de source voor hun naam de aanduiding AND gekregen om aan te geven dat we met een wired-AND functie van de signalen te maken hebben. Verder verloopt alles op gelijke wijze als fully interlocked handshake input en output. We kunnen daarbij ook de Wait-functie gebruiken.

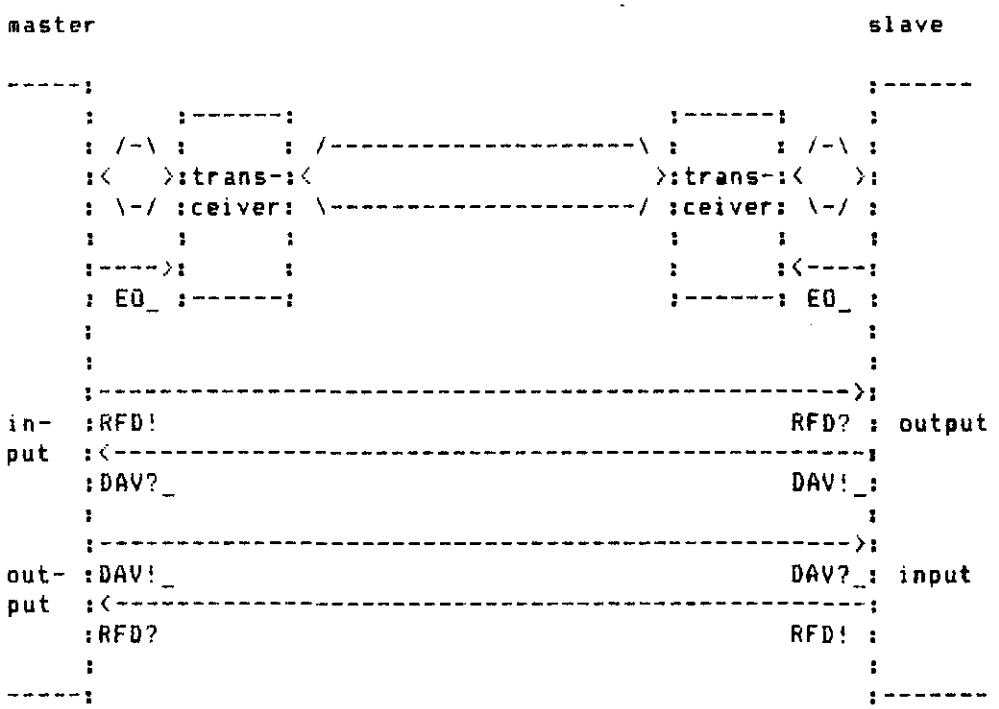

EO_
databus (source)
I/O-write_
SRQO
LO_
i/o-bus
DAV?_
AND(RFD!)
AND(DAC!)
RFD1!
DAC1!
SRQI1
I/O-read_
LI1_
SOURCE
1


EI1
RD1_
databus1
(acceptor)
RFD2!
DAC2!
SRQI2
I/O-
read2_
LI2_
EI2
RD2_
databus2
(acceptor)
A
C
C
E
P
T
O
R
2

1.6. Bidirectionele I/O.

Bij bidirectionele I/O willen we data uitwisselen in twee mogelijke richtingen. Dit wil zeggen dat een apparaat zowel de bron als de bestemming van de data kan zijn. Bij bidirectionele I/O werken we ook in de fully interlocked I/O-mode. Dit wil dus zeggen dat we bij output evenals bij input weer te maken hebben met een van de reeds besproken protocollen. De extra functie, die door deze I/O-mode wordt geboden is de mogelijkheid om tijdens bedrijf de richting van het datatransport te veranderen. De I/O-bouwsteen zal dus op het ene moment bron van de data zijn en even later bestemming. Het probleem is nu dat de data-overdracht toch geheel veilig moet verlopen. We moeten beide I/O-bouwstenen dus zeer strak synchroniseren bij het omschakelen van de richting van het datatransport. Om eventuele problemen uit de wereld te kunnen helpen zal er bij bidirectionele I/O een master moeten zijn en een slave. De besturingen in zowel de master als de slave moeten een volgorde doorlopen, die het mogelijk maakt om vanuit de master op een gegeven ogenblik vanuit de ene richting naar de andere richting om te schakelen. De slave moet dit omschakelen dan braaf volgen zonder dat er busconflicten ontstaan op de I/O-bus. De besturing van de master moet zo opgezet zijn dat het mogelijk wordt om na elke transportactie de richting om te kunnen schakelen. Voor dit protocol hebben we vier besturingslijnen nodig. Er zijn twee lijnen tussen de master en de slave voor de besturing van de input naar de master (d.w.z. output voor de slave). Daarnaast zijn er twee besturingslijnen tussen de master en de slave voor de output vanuit de master naar de slave.

Onderstaande afbeelding toont de aansluitingen bij bidirectionele I/O.



We zien dat er twee transceivers nodig zijn om de i/o-bus om te kunnen schakelen. Deze worden bestuurd door de EO_-lijnen van zowel de master en de slave. De besturingen moeten dus zorgen dat aan beide zijden de transceivers

op alle ogenblikken in de juiste richting geschakeld zijn. Om problemen te voorkomen, zorgen we dat alleen output mogelijk wordt gemaakt als dit duidelijk door de master is gevraagd. Output bij de master wil zeggen data vanuit de master naar de slave. Bij de slave betekent het output vanuit de slave naar de input van de master.
De master moet telkens aan de slave doorgeven in welke richting het datatransport plaats moet vinden. Het protocol moet zo ontworpen worden, dat er geen extra lijn nodig is tussen de twee betrokken I/O-bouwstenen. Het inputprotocol en het outputprotocol van de master moet steeds op een synchronisatiepunt terecht komen na elke transportactie. In dit punt moet het mogelijk worden om van de ene richting naar de andere om te schakelen. Door aan de kant van de master de uitgaande besturingslijnen niet te veranderen in het synchronisatiepunt, zal aan de kant van de slave gewacht worden omdat het protocol niet doorgestart wordt. Bijvoorbeeld zou bij een reeks outputacties de master de DAV!_-lijn laag kunnen houden om aan te geven dat er niet om een nieuw outputbyte mag worden gevraagd. De inputbesturing van de slave zal namelijk moeten wachten op het hoog worden van de DAV?_-lijn. Indien we hierbij zorgen dat bij de master de EO_-lijn in deze toestand hoog is (d.w.z. de tranceivers staan dan naar de master toe gericht) is een conditie geschapen waarbij we zonder meer de input naar de master op kunnen starten.
Bij het fully interlocked inputprotocol zal de inputcyclus opgestart worden door vanuit de bestemming (de master) de RFD!-lijn hoog de maken. Hierbij zal bij de master ook de EO_-lijn zo geschakeld moeten zijn dat er input mogelijk is. De outputbesturing van de slave wacht op het hoog worden van de RFD?-lijn vanuit de master. Is deze lijn hoog dan zal er aan de processor van de slave om een datawoord worden gevraagd (SRQ0). Na het naar buiten sturen van de data wordt DAV!_ laag gemaakt, zodat de bestemming de data over kan nemen. Dit wordt teruggemeld door de master aan de slave door RFD! laag te maken. Hierbij zal er ook aan de processor van de master om service worden gevraagd. We hebben nu weer het normale fully interlocked inputprotocol.
Indien de masterprocessor wil stoppen, zal er in de serviceroutine net voor het overnemen van de data een nieuw modecontrolwoord naar de I/O-bouwsteen moeten worden gestuurd, waarin de richtingsbit wordt omgezet van input naar output. Daarna wordt de data overgenomen. Het nog actieve inputprotocol wordt afgemaakt tot aan het punt waarin RFD! weer hoog zou worden. Dit punt is weer het synchronisatiepunt vanuit de master naar de slave. De outputbesturing van de slave wacht hier. In de besturing van de master kan nadat de inputactie afgemaakt is de inputbesturing opgehouden worden. De outputbesturing mag nu doorgaan met het hoog maken van DAV!_. De slave zal dit zien als een melding dat er om data mag worden gevraagd. Hierbij start het outputprotocol weer door. Dit zal net zo lang duren totdat er tijdens de service in de master besloten wordt voor het naar buiten sturen van de laatste beschikbare byte een nieuw mode control woord (opdracht tot input voor de master) naar buiten te sturen. We hebben hiermee een sluitend protocol verkregen waarmee we telkens na een volledige cyclus van transportrichting kunnen verwisselen. De master houdt hierbij dan steeds de lijnen op die het doorstarten van de handshake mogelijk maken. In geval van input is dit de laag-hoog overgang van RFD!. Bij output is dit de laag-hoog overgang van DAV!_.

Door op het synchronisatiepunt van de outputbesturing de EO_ naar input te schakelen komen we steeds in een situatie waarbij zonder meer om- of doorgeschakeld mag worden.

Bij het inschakelen van de I/O-bouwstenen zal door de software de ene bouwsteen bidirectionele master zijn en de andere bidirectionele slave. De master zal daarbij als input of als output worden geinitialiseerd. De slave is altijd op input geschakeld. De master zal na de initialisatie naar input RFD!

hoog maken, als tenminste zijn DAV?_ hoog is (slave ready). De slave herkent dit en zal een inputtransactie opstarten etc. Als de master bij initialisatie op output gezet was zou DAV!_ hoog geworden zijn en dus in de slave het inputprotocol actief zijn geworden. Na het zo opzetten van de richting van het datatransport kan het normale I/O-protocol afgehandeld worden. Hierbij zal steeds byte na byte door de bron naar de bestemming worden gebracht. Dit moet doorgaan totdat in de software van de master het commando wordt gegeven om van richting om te keren. Dit commando zal dan door de master naar de slave doorgegeven moeten worden. We moeten dit echter niet zomaar op een willekeurig ogenblik doen. Het resultaat zou kunnen zijn dat de handshake niet afgemaakt wordt of dat er een byte onderweg verloren gaat. We zullen daarom een verzoek tot omschakelen vanuit de master naar de slave geven op een ogenblik dat de slave een acknowledge verwacht op een vorige cyclus. Door eerst het omschakelcommando te geven, en daarna pas de acknowledge zal de slave de cyclus afmaken en dan niet een nieuwe actie opzetten, maar overschakelen naar het andere protocol.

De timing voor bidirectionele I/O is identiek aan de timing bij fully interlocked handshake.

2. FORMELE BESCHRIJVING I/O-PROTOCOLLEN.

2.1. Inleiding.

De I/O-protocollen van hoofdstuk 1 laten zich in twee groepen indelen :

1) direct I/O,
2) handshake I/O.

De protocollen van groep 2) zijn zodanig gedefinieerd dat ze grote mate van overeenkomst vertonen (vergelijk timingdiagrammen in hoofdstuk 1). Bij de beschrijving kan daarom worden volstaan met een beschrijving voor input en een voor output, die voor alle protocollen van deze groep voldoet.
Kenmerkende eigenschappen voor groep 2) zijn verder :

- elk protocol doorloopt een reeks acties in een vaste volgorde,
- de te ondernemen actie is afhankelijk van gegevens uit de buitenwereld (bv. een laag-hoog overgang van RFD door een randapparaat), en van de al uitgevoerde acties in het verleden.

Met andere woorden : een protocol doorloopt een sequentie van een eindig aantal toestanden, waarbij de nieuwe toestand een functie is van de huidige (oude) toestand en een reeks inputsignalen.
Voor de beschrijving van zo'n sequencer zijn een aantal hulpmiddelen beschikbaar, zoals toestandsdiagrammen, toestandstabellen en beschrijving met een programmeertaal als Pascal.

2.2.Toestandsdiagrammen.

2.2.1. Input.

Uitgaande van hoofdstuk 1 kunnen we voor de handshake protocollen een input toestandsdiagram samenstellen. Een toestandsdiagram is feitelijk een flowdiagram waarin iedere toestand door een rechthoek wordt gerepresenteerd. In de rechthoek wordt de naam van de toestand en de bijbehorende vector van uitgangssignalen aangegeven. Deze vector is opgebouwd uit : LI_, RFD!, SRQI. Hierin ontbreekt DAC! omdat geldt DACnon(RFD!).
De sequencer voor handshake input heeft twee ingangen, nl. DAV?_, en een read-signaal van de processor wat intern door de controller wordt gedecodeerd naar EI.
De handshake voor input doorloopt de volgende acties:

1) Wachten tot randapparaat klaar is om te communiceren (DAV?_=1).

2) Melden aan randapparaat dat data kan worden overgenomen (RFD/=1), en wachten tot randapparaat data op de I/O-bus zet (DAV?_=0).

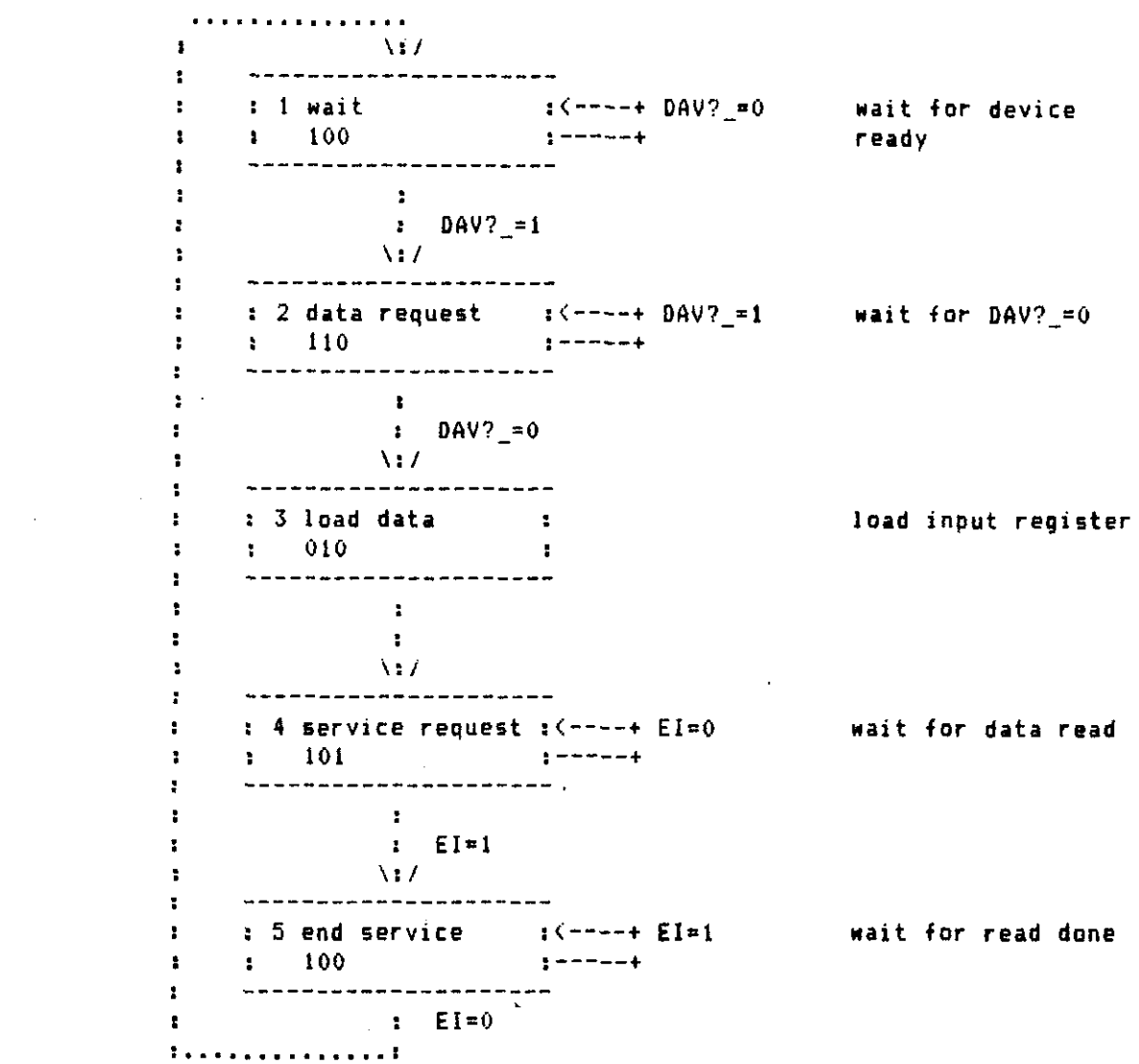
3) Data inklokken in inputregister (LI_=0).

4) Om service vragen bij processor (SRQI=1) en melden aan randapparaat dat data is overgenomen (RFD/=0).

5) Wachten tot processor data overneemt, en wegnemen service request.

6) Terug naar 1).

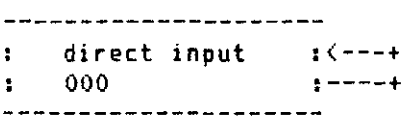
Het diagram wordt dan : (uitgangsvector : LI_, RFD!, SRQI)



We zien dat op een duidelijke wijze de handshake voor inputprotocollen kan worden weergegeven. Telkens wordt door de inputsequencer gewacht op een wijziging van een van de ingangssignalen, waarna dit vertaald wordt naar een nieuwe uitgangsvector in een nieuwe toestand. Hierop vormt toestand 3 een uitzondering. Toestand 3 wordt bereikt als DAV?_ actief is geworden, dus als er geldige inputdata op de I/O-bus staat. Deze data moet in het inputregister geklokt worden, dus moet een puls gegenereerd worden op de LI_ -lijn. We doen dit door in toestand 3 LI_ actief te maken, en de sequencer meteen (zonder tussenkomst van EI of DAV?_) naar toestand 4 te laten gaan waarin LI_=1. De lengte van de puls zal in het algemeen een klokperiode bedragen.
De data staat nu klaar om door de processor gelezen te worden. De sequencer zal daarom een interrupt request geven (SRQI=1) en wachten tot de data met EI=1 op de databus wordt gezet, enzovoort.

Direct input is een heel ander verhaal. Hier is geen sprake van een sequentie van acties, we blijven steeds in dezelfde toestand. Tijdens direct input zal altijd LI_ actief zijn, terwijl RFD en SRQI niet worden gebruikt. We kunnen

direct input daarom opvatten als een enkele toestand met uitgangsvector LI_, RFD/, SRQI = 0, 0, 0.



2.2.2. Output.

Op analoge wijze kan voor de outputzijde van de handshake protocollen een toestandsdiagram worden opgesteld. De uitgangsvector bestaat nu uit EO_, DAV/_, en SRQO. De ingangssignalen voor de sequencer zijn RFD?, DAC? en LO_. LO_ ontstaat uit I/O-write van de processor. Uit hoofdstuk 1 kunnen we de volgende sequentie afleiden :

1) Wachten tot randapparaat output kan accepteren (RFD? actief).

2) Vragen om geldige outputdata (SRQO actief), en wachten tot de processor deze in het outputregister schrijft (LO_ actief).

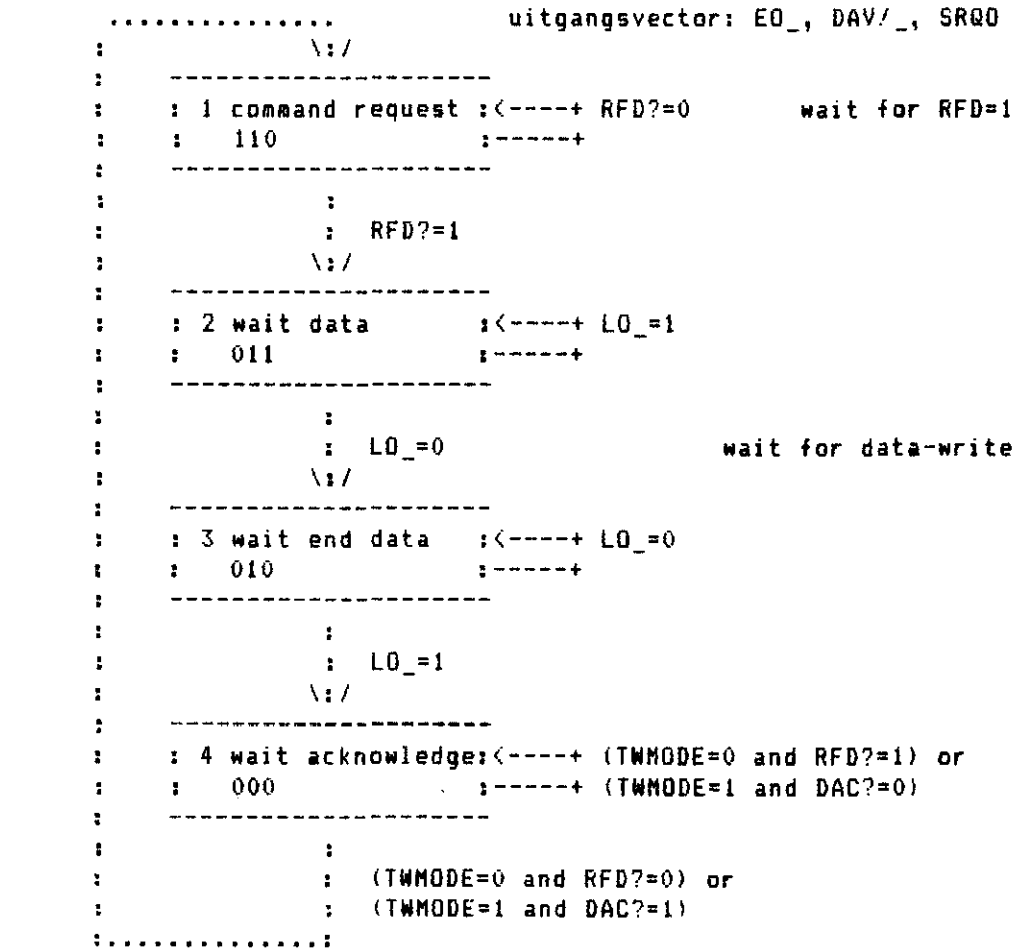
3) Wegnemen van SRQO en wachten tot LO_=1.

4) Melden aan randapparaat dat geldige data op de I/O-bus staat, en wachten op de terugmelding dat deze data is overgenomen.

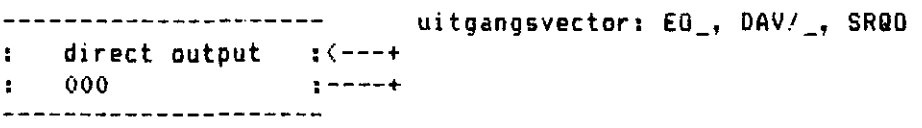
5) Terug naar 1).

Enige voorzichtigheid is geboden met betrekking tot de terugmelding van het randapparaat (onder punt 4)). Immers, er is hier verschil tussen fully interlocked handshake en 3-wire handshake. Bij de laatste weet de source dat alle acceptoren de data hebben overgenomen doordat DAC?=1 wordt. Bij fully interlocked handshake hebben we slechts met een bestemming te maken, zodat met RFD?=0 de overname gemeld kan worden.
Om dit onderscheid in het toestandsdiagram te kunnen verwerken, krijgt de sequencer het ingangssignaal TWMODE (three wire mode) erbij. Dit is een intern besturingssignaal dat uit het mode-control register kan worden afgeleid.

Het toestandsdiagram van de outputsequencer komt er na dit alles als volgt uit te zien :



De direct output mode kunnen we met een enkele toestand beschrijven :



2.2.3. Initialisatie en synchronisatie.

In de voorgaande paragrafen is voor alle I/O-protocollen een beschrijving met toestandsdiagrammen afgeleid. Om deze diagrammen in de praktijk bruikbaar te maken, moeten echter nog enige toevoegingen aangebracht worden.
Allereerst moet in de sequencers een correcte synchronisatie worden ingebouwd. Dit heeft betrekking op het (tijdelijk) onderbreken van handshake input of output, hetgeen zich op twee manieren voor kan doen :

1) Door de continue/wait bits van het mode-control register, C_WIN en C_WOUT,

2) In de bidirectionele mode : een master moet in staat worden gesteld om op het juiste moment om te schakelen tussen input en output (zie par. 1.6).

Beide mogelijkheden vertonen een grote mate van overeenkomst. Voor alle twee geldt dat verlies van I/O-bytes kan worden voorkomen door de handshake sequence pas op te houden na afloop van de protocol-cyclus. Dit kan worden gerealiseerd door voor de toestand met nummer 1 een extra toestand in te bouwen.

De inputsequencer komt dan in deze synchronisatietoestand na afloop van toestand 5, dus als EI inactief is geworden. De uitgangsvector wordt nu LI_, RFD!, SRQI=1, 0, 0, waarmee wordt bereikt dat het randapparaat geen actie meer zal ondernemen want RFD! blijft inactief. Zolang C_WIN=0 moet de sequencer de synchronisatietoestand handhaven. Met Continue actief volgt overgang naar toestand 1 .

De outputsequencer komt in zijn synchronisatiepunt na afloop van toestand 4. De uitgangsvector wordt in dit geval EO_, DAV/_, SRQO=1, 0, 0, dus DAV/_ wordt laag gehouden en EO_ inactief. Overgang naar toestand 1 volgt aan de hand van C_WOUT.

In de bidirectionele mode is er echter nog een probleem. Stel dat een handshake output wordt uitgevoerd en dat omschakeling naar input gewenst is. Dan zal, als de interrupt (SRQO) voor de laatste outputbyte bij de processor binnenkomt, net voordat de laatste outputbyte naar buiten gestuurd wordt in het mode-control register van de master naar input worden omgeschakeld (door middel van software). Dit zou tot gevolg hebben dat de inputsequencer van de master onmiddellijk een inputcyclus op gaat zetten terwijl zijn outputsequencer nog bezig is met de laatste outputbyte. Hierdoor kan contentie ontstaan op de I/O-bus.
Een soortgelijke situatie doet zich voor bij omschakeling naar output. Daar zal worden omgeschakeld nadat SRQI actief is geworden, en voordat de laatste inputdata gelezen wordt door de processor.
Deze problemen kunnen worden opgelost door uitbreiding van de uitgangsvector van input-, en outputsequencer met resp. IBUSY en OBUSY. Deze signalen zijn alleen laag als de sequencer zich in zijn synchronisatietoestand bevindt. De master weet nu wanneer de outputsequence is afgelopen zodat de inputsequence nog wat wordt opgehouden, en omgekeerd.

De overgang van de synchronisatietoestand naar toestand 1 is hierdoor wat gecomplic[illegible] gebruikt nu de interne besturingssignalen ISYNC en DSYNC.

Handshake input :

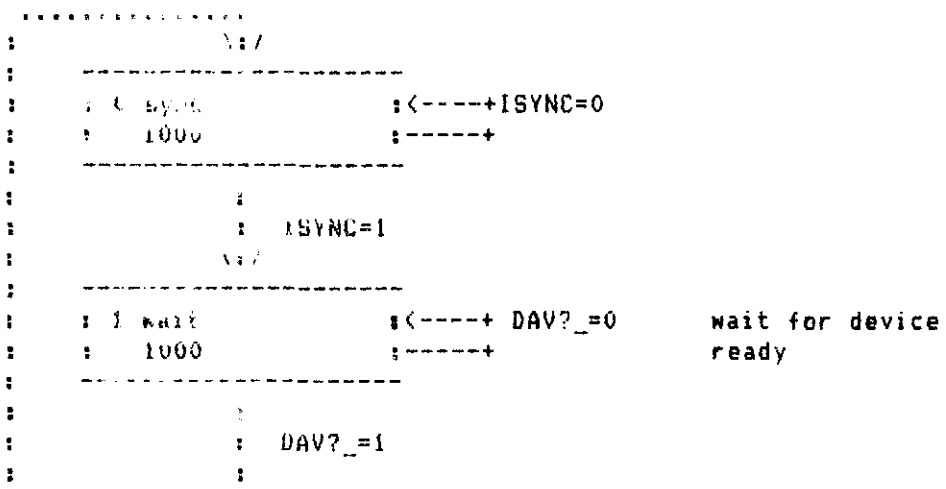
ISYNC = 1 :

(C mode and bidirectional slave or fully interl. input or three wire input))

or

(C mode and bidirectional master))

Toestandsdiagram : (uitg.vector LI_, IBUSY, RFD!, SRQI)



Handshake output:

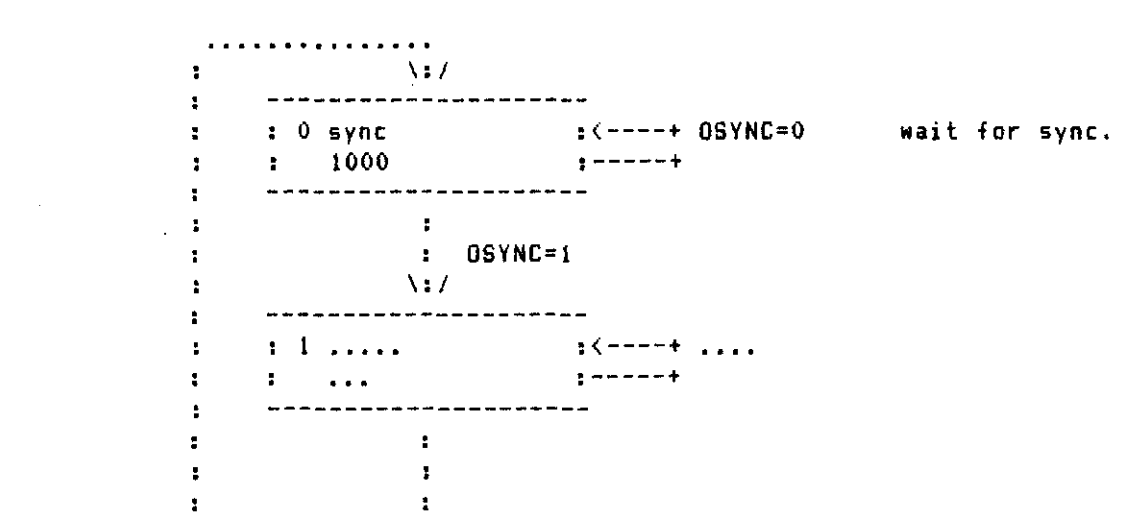
OSYNC = 1 if

(C_WOUT=1 and (bidirectional slave or fully interl. output or three wire output))

or

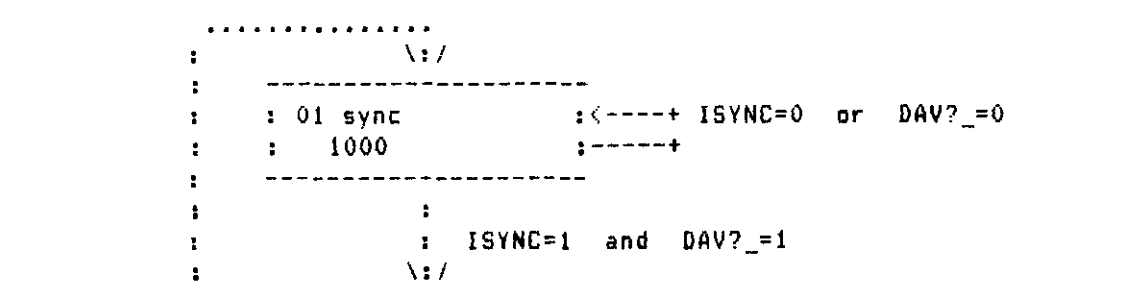
(C_WOUT=1 and IBUSY=0 and (bidirectional master))

Toestandsdiagram : (uitgangsvector EO_, OBUSY, DAV?_, SRQO)



Opmerking :

In de inputsequencer valt nu op dat de twee opeenvolgende toestanden 0 en 1 dezelfde uitgangsvector hebben. Hierdoor kunnen deze worden samengevoegd tot een nieuwe toestand met nummer 01 :



Resteert nog de initialisatie van de I/O-bouwsteen en het schakelen tussen de I/O-modes.
Bij power-on en bij een reset moet het interne PRESET signaal actief worden, waarna de directe input mode geselecteerd wordt. Vervolgens kan door programmering van het mode-control register een willekeurige andere mode gekozen worden. Bij deze mode-overgang mogen geen ongewenste signaal-wisselingen optreden op lijnen als EO_, DAV_, SRQI, enz.. Dankzij PRESET zullen op de I/O-bus geen wisselingen optreden.
Tot nu toe hebben we in de bouwsteen een reeks toestanden in de inputsequencer en in de outputsequencer, en twee 'losse' toestanden voor direct I/O. Elk van deze onderdelen wordt vanuit het mode-control register geselecteerd. Omdat er in totaal acht modes voorkomen, zijn hiervoor drie bits nodig. Hier komt nog een bit bij voor de keuze master/slave in de bidirectionele mode. Totaal dus vier mode-selectie bits.

Bij power-on en reset worden de mode-selectie bits automatisch op nul gezet. Dit kunnen we gebruiken om PRESET actief te maken. Verder zorgen we dat met PRESET=1 de directe input toestand gekozen wordt. De uitgangsvector hiervan is LI_, RFD!, SRQI=0, 0, 0 . Omdat RFD! = non(DAC/) geldt ook dat DAC/ inactief is.
Aan outputzijde willen we dat tijdens direct input EO_, DAV, SRQO=1, 0, 0. Dit bereiken we door met PRESET=1 de outputsequencer in zijn synchronisatie-toestand te drukken.
De initialisatie is nu correct verlopen. Als de gebruiker vervolgens naar een andere mode wil, zal hij de mode-selectie bits aanpassen. Dit zal tot gevolg hebben dat bij omschakeling naar

fully interlocked en 3-wire input :

> PRESET wordt inactief en uit de mode-selectie bits wordt het signaal HSMODE afgeleid. Door HSMODE=1 wordt de direct input toestand verlaten, en komt de inputsequencer in zijn synchronisatietoestand. Met ISYNC en DAV?_ kan de inputcyclus dan een aanvang nemen. Aan outputzijde moet de situatie ongewijzigd blijven.

fully interlocked en 3-wire output :

> PRESET wordt inactief. Omdat de outputsequencer al in toestand 0 verkeert, kan met OSYNC de afwikkeling beginnen. Door HSMODE=0 blijft de direct input toestand gehandhaafd. Hierdoor ontstaat voor de processor de mogelijkheid om het outputregister terug te lezen, want LI_ is steeds actief.

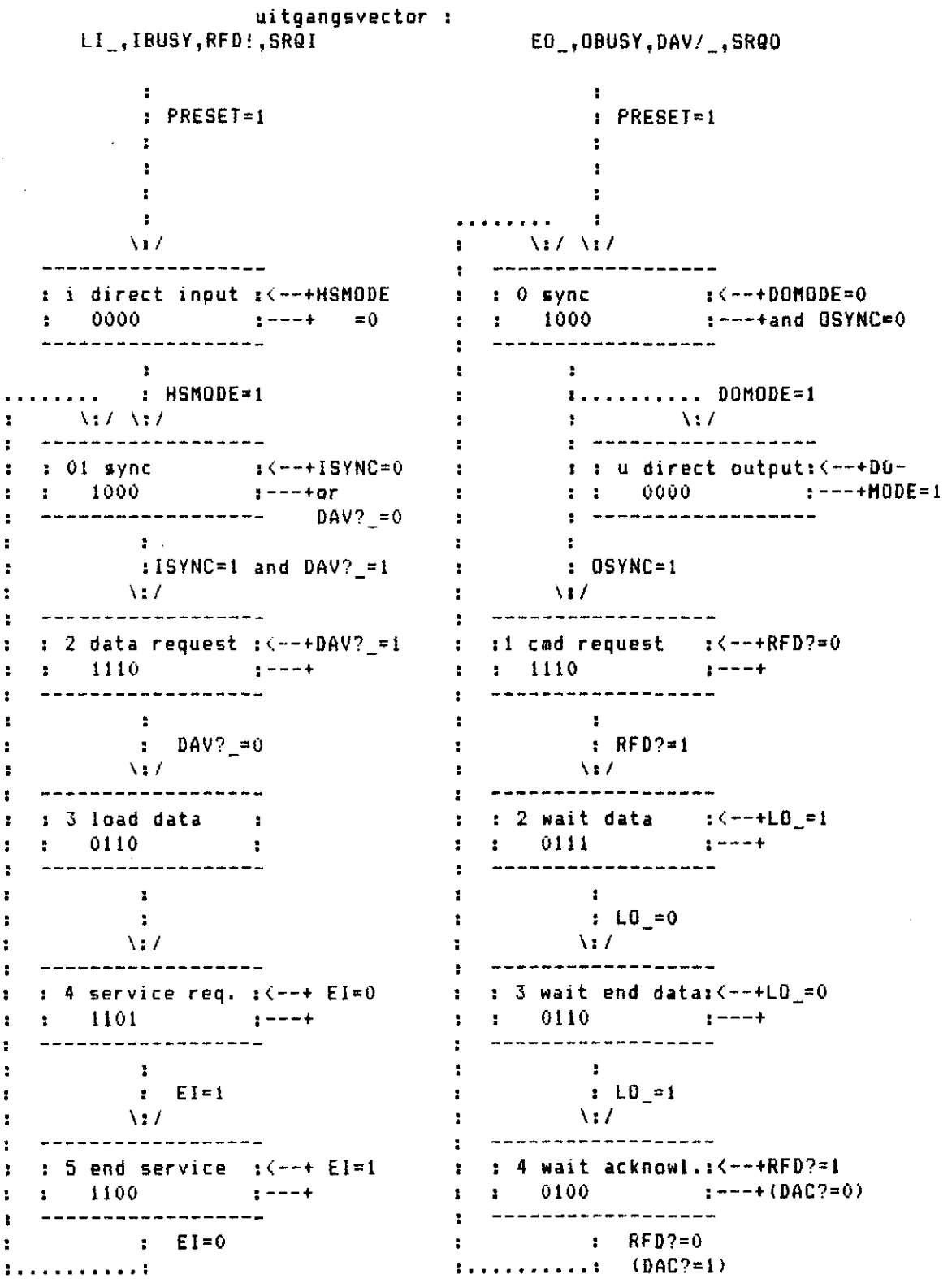
direct output :

> Met PRESET=0 en HSMODE=0 geldt aan inputzijde direct input. De outputsequencer kan met DOMODE (afgeleid uit mode-selectie) overgaan naar de direct output toestand.

bidirectional mode :

> Met HSMODE=1 komt de inputsequencer in toestand 01. De master kan nu met OSYNC en ISYNC het gewenste protocol afhandelen.

In een toestandsdiagram ziet het er dan uiteindelijk als volgt uit :



2.3. Toestandstabellen.

Naast de grafische representatie uit de vorige paragraaf, is ook een weergave in tabelvorm mogelijk. Zulke tabellen zien er voor input en output als volgt uit :
(H, L = hoog,laag ; - = don't care)

inputsequencer:

inputs							outputs					
PRESET	HSMOD EC	ISYNC	DAV?	EI	old state		new state	LIUSY_	IBUF!	RSDI	SRQI	
H	-	-	-	-	-		i	L	L	L	L	preset
L	L	-	-	-	i		i	L	L	L	L	direct input
L	H	-	-	-	i		01	H	L	L	L	
L	-	L	L	-	01		01	H	L	L	L	wait for
L	-	L	H	-	01		01	H	L	L	L	ISYNC
L	-	H	L	-	01		01	H	L	L	L	and
L	-	H	H	-	01		2	H	H	H	L	DAV?_
L	-	-	H	-	2		2	H	H	H	L	wait for
L	-	-	L	-	2		3	L	H	H	L	DAV?_=0
L	-	-	-	-	3		4	H	H	L	H	load input
L	-	-	H	-	4		4	H	H	L	H	wait for
L	-	-	H	-	4		5	H	H	L	L	EI=1
L	-	-	-	H	5		5	H	H	L	L	wait for
L	-	-	-	L	5		01	H	L	L	L	EI=0

outputsequencer:

inputs									outputs				
PRESET	DOMODE	OSYNC	TWOMODE	RFD?	LO_	DACK?	old state	new state	E0_	OBUSY	OAV!	SRQ0	
H	-	-	-	-	-	-	-	0	H	L	L	L	preset
L	L	L	-	-	-	-	0	0	H	L	L	L	wait for
L	H	-	-	-	-	-	0	u	L	L	L	L	OSYNC
L	-	H	-	-	-	-	0	1	H	H	H	L	or DOMODE
L	L	-	-	-	-	-	u	0	H	L	L	L	direct
L	H	-	-	-	-	-	u	u	L	L	L	L	output
L	-	-	-	L	-	-	1	1	H	H	H	L	wait for
L	-	-	-	H	-	-	1	2	L	H	H	H	RFD?=1
L	-	-	-	-	H	-	2	2	L	H	H	H	wait for
L	-	-	-	-	L	-	2	3	L	H	H	L	LO_=0
L	-	-	-	-	L	-	3	3	L	H	H	L	wait for
L	-	-	-	-	H	-	3	4	L	H	L	L	LO_=1
L	-	-	H	-	-	L	4	4	L	H	L	L	wait for
L	-	-	H	-	-	H	4	0	H	L	L	L	acknowledge
L	-	-	L	H	-	-	4	4	L	H	L	L	
L	-	-	L	L	-	-	4	0	H	L	L	L	

Tabellen als deze zijn een geschikt uitgangspunt voor optimalisatie en simulatie met computerprogramma's. Het nummer van iedere toestand moet dan echter wel eerst worden omgezet in een binaire code. Hiervoor zijn in dit geval meerdere mogelijkheden. Ten eerste zou elk nummer rechtstreeks naar binair kunnen worden omgezet. De tweede manier volgt door de uitgangsvectoren in de toestandsdiagrammen te bekijken. Dan blijkt dat iedere toestand eenduidig kan worden gerepresenteerd door zijn uitgangsvector. De tweede methode levert het minste aantal kolommen op in de toestandstabellen, zodat daarvoor is gekozen.

2.4. Representatie met een programmeertaal.

Als laatste beschrijvingswijze kunnen we een Pascal-achtige taal gebruiken. Op basis van voorgaande paragrafen leiden we nu een 'procedure outputcontrol' en een 'procedure inputcontrol' af.

```
Procedure outputcontrol;

repeat

   EO_:=1;                                  (* output disabled *)
   OBUSY:=0;                                (* output cycle not active *)
   DAV/_:=0;                                (* not ready for data request *)
   SRQO:=0;                                 (* no interrupt request *)

repeat until (DOMODE=1) or (OSYNC=1);       (* wait for sync. or dir. output *)

if (DOMODE=1) then repeat EO_:=0
                   until (PRESET=1) or (DOMODE=0);

              else (* OSYNC=1 *)            (* handshake output *)
    begin
            OBUSY:=1;                       (* output cycle active *)
            DAV/_:=1;                       (* ready for output command *)

            repeat until (RFD?=1);          (* wait for RFD? 0 --> 1*)

            EO_:=0;                         (* enable output *)
            SRQO:=1;                        (* interrupt request on *)

            repeat until LO_=0;             (* wait for data write from cpu *)

            SRQO:=0;                        (* reset interrupt request *)

            repeat until LO_=1;             (* wait for end of write *)

            DAV/_:=0;                       (* data valid message *)

            if (TWMODE=0)
            then repeat until (RFD?=1)      (* fully int. acknowledge *)
            else repeat until (DAC?=0);     (* 3-wire acknowledge *)

    end
until forever;
end.
```

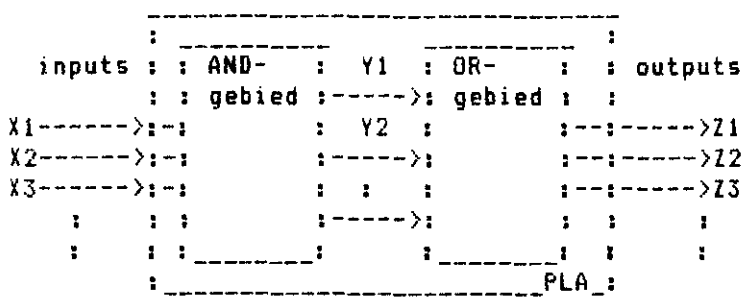
```
Procedure inputcontrol;

repeat

   LI_:=0;                              (* input latch open *)
   IBUSY:=0;                            (* input handshake ended *)
   RFD/:=0; DAC/:=1;                    (* not ready for data *)
   SRQI:=0;                             (* no interrupt request *)


   repeat until (HSMODE=1);             (* wait for HSMODE=1: handshake
                                           input mode or bidirectional mode *)

   repeat

  (* execute handshake input algorithm *)
  (* bidir, threewire, handshake *)

      LI_:=1;                           (* load signal inactive *)

      repeat until (ISYNC=1)            (* wait for sync and device ready *)
               and (DAV?_=1);

      IBUSY:=1;                         (* input cycle active *)
      RFD/:=1;                          (* send data request *)

      repeat until DAV?_=0;             (* wait for data available message *)

      LI_:=0;                           (* load data into inputregister *)

      (* wait one clock cycle *)

      LI_:=1;                           (* freeze data *)
      SRQI:=1;                          (* interrupt request on *)
      RFD/:=0; DAC/:=1;                 (* acknowledge input data *)

      repeat until (EI=1);              (* wait for read data from cpu *)

      SRQI:=0;                          (* interrupt request off *)

      repeat until (EI=0);              (* wait for end of data read *)

   until (HSMODE=0)
until forever
end.
```

3. I/O-CONTROLLER HARDWARE.

Dit hoofdstuk vormt een nadere uitwerking van de I/O-controller zoals deze in de inleiding globaal is geschetst. Begonnen wordt met twee wat algemenere paragrafen.

3.1. Programmable logic arrays.

Ter realisatie van combinatorische logica is in de I/O-controller een dankbaar gebruik gemaakt van PLA's (lit.[1]). De algemene structuur van een PLA is als volgt:



Een functie Z3 die hiermee geimplementeerd zou kunnen zijn :

Z3=OR(Y1, Y2)
met Y1=AND(X1_, X2, X3)
en Y2=AND(X1, X2_, X3_)

Aan de hand van soortgelijke uitdrukkingen kan een zgn. PLA-tabel worden opgesteld, waarbij de volgende afspraken worden gehanteerd :

* H,L,- : inputsignaal resp. hoog,laag of don't care,
* A, . : outputsignaal al dan niet geactiveerd door combinatie van inputs,
* van elk outputsignaal is de polariteit aangegeven, dat wil zeggen laag- of hoog-actief (L resp. H).

Bijvoorbeeld :

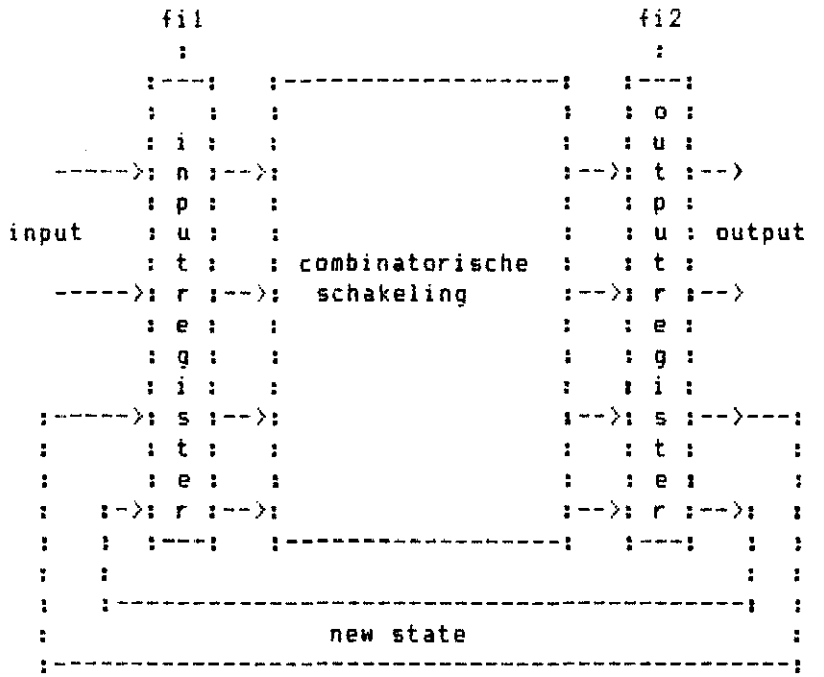
	inputs	outputs
	X X X X X	Z Z Z
	1 2 3	1 2 3
polarity:		L H H
	L H H - - -	. . A
	H L L - - -	. . A
	H - L - - -	A . .
	H L L - - -	. A .

De uitdrukking als boven gegeven voor Z3 laat zich (evenals voor de andere outputs) direct uit de tabel teruglezen. Het is dan ook vaak handiger meteen met een PLA-tabel te beginnen.
Op basis van de Quine-McCluskey-methode kan de PLA-tabel eventueel verder worden geoptimaliseerd.
Een PLA biedt een aantal voordelen :

- compacte en regelmatige lay-out bij integratie,
- een geprogrammeerde functie kan snel en gemakkelijk worden veranderd zonder ingrijpende wijzigingen in de lay-out.

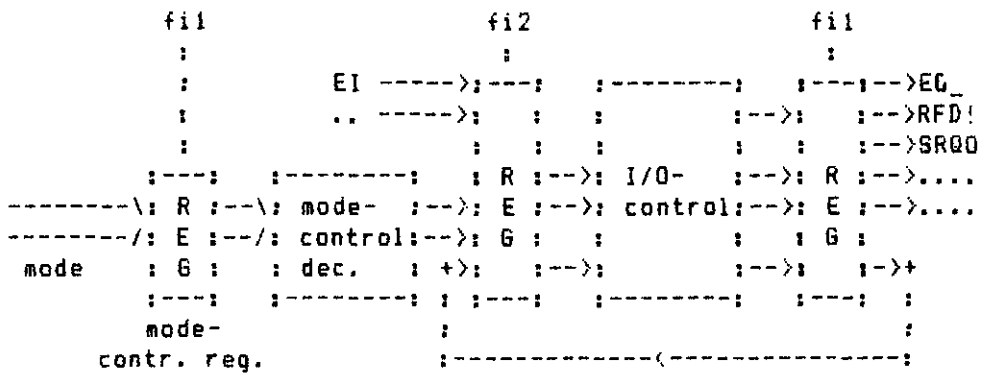
3.2. Finite State Machines en modulaire opbouw.

In hoofdstuk 2 is de I/O-afwikkeling beschreven door de protocollen op te delen in aantal toestanden. Er zijn een aantal mogelijkheden om dit in hardware te realiseren. Een daarvan is de 'finite state machine' van onderstaande figuur : (lit. [1])



De timing wordt gerealiseerd met een 2-fase klok. Tijdens fi1 worden samples genomen van de ingangssignalen. Deze worden samen met de huidige toestand van de combinatorische logica in het inputregister geklokt. De logica berekent hieruit een aantal uitgangssignalen en de nieuwe toestand. Deze uitkomsten worden tijdens fi2 in het outputregister gezet. De nieuwe toestand wordt teruggekoppeld voor gebruik bij de volgende slag.
Op deze wijze is een betrouwbare en overzichtelijke timing verkregen. Bovendien kan voor de logica gebruikt gemaakt worden van PLA's. Vergelijking met de toestandstabellen uit het vorige hoofdstuk leert dat deze rechtstreeks kunnen worden gebruikt als PLA-tabel.

De register - logica - register structuur leent zich verder bijzonder goed om tot een modulaire opbouw van de I/O-controller te komen. Als voorbeeld :

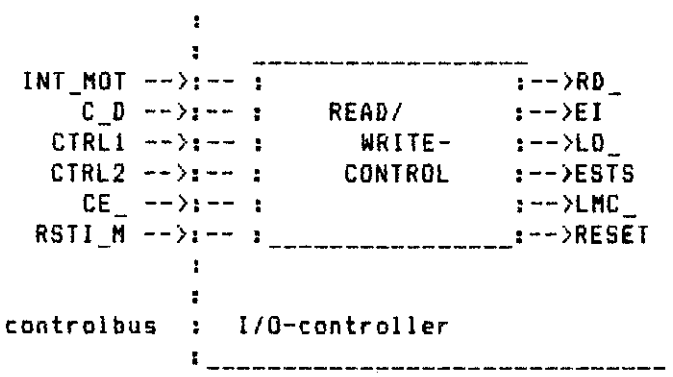


In een dergelijke opzet werken de afzonderlijke blokken logica parallel.

3.3. Interfacing met een processor.

3.3.1. Read/Write-control en Reset.

Het hart van de interface van de I/O-controller met een centrale processor wordt gevormd door een decoderings-PLA dat binnenkomende buscontrolsignalen omzet in interne besturingssignalen. Als bijzonderheid kan worden vermeld dat de I/O-controller zowel voor Motorola-, als voor Intel-systemen geschikt is. Hieronder is de logica meer gedetailleerd weergegeven :



Bussignalen :

INT_MOT	:	keuze Intel/Motorola
1	:	Intelbus
0	:	Motorolabus
C_D	:	Control/Data; selectie besturings-,of data-registers
1	:	schrijven naar mode-controlregister of lezen van statusregister
0	:	lezen/schrijven m.b.t. I/O-data
CTRL1,CTRL2:		chipselect-lijnen
Intel	:	CTRL1 = RD_ (processor-read)
	:	CTRL2 = WR_ (processor-write)
Motorola	:	CTRL1 = E (enable)
	:	CTRL2 = R/W_ (Read,non-Write)
CE_	:	chip enable
1	:	I/O-controller niet geadresseerd
0	:	wel geadresseerd
RSTI_M	:	I/O-controller reset; actief laag in geval van Motorolabus, actief hoog voor Intelsystemen

Interne controlsignalen :

RD_ : besturing tri-state drivers inputpad
EI : enable inputbyte (in combinatie met RD_)
LO_ : kloksignaal outputregister
RESET :intern resetsignaal voor mode-controlregister.
De controller komt hierdoor in de direct input mode.
ESTS :enable status;readsignaal voor statusinformatie,
zet statusinfo op de processordatabus.
LMC_ :load mode-control; writesignaal voor mode-controlregister.
Het mode-controlword wordt van de databus in het mode-controlregister geladen.

Om tot programmering van de Read/Write-control PLA te komen zouden als eerste stap de interne signalen als functie van de bussignalen kunnen worden opgeschreven, bv. :

```
RESET = (RSTI_M=1 and INT_MOT=1) or
        (RSTI_M=0 and INT_MOT=0)
```

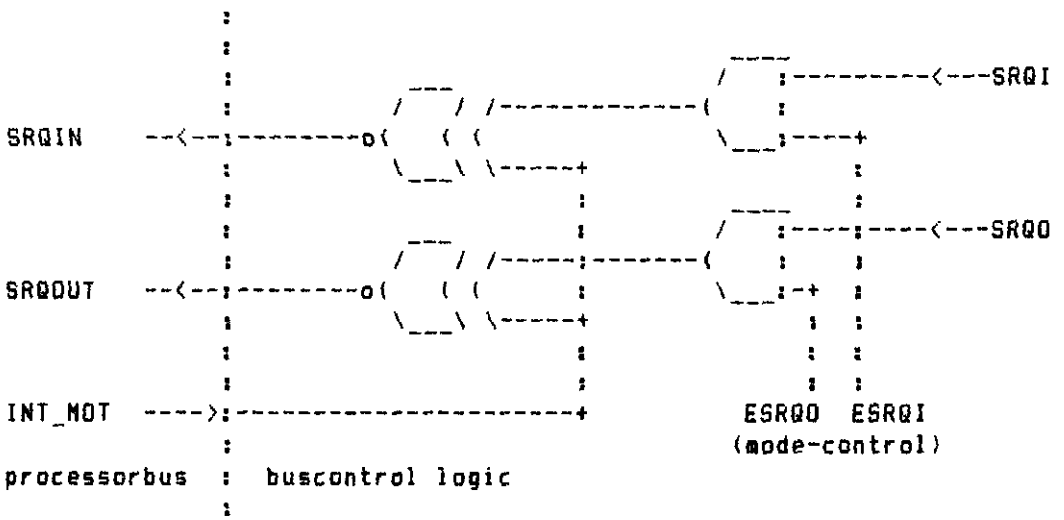
Aan de hand van deze uitdrukkingen kan de PLA-tabel worden opgesteld :

read/write-control:

	inputs							outputs						
	I			R										
	N			S									R	
	T		C	T						E			E	
	_		T	_						S		L	S	
	M	C	C	M	R	R	I	R	E	T	L	M	E	
	O	_	E	1	L	L	_	D	I	S	O	C	T	
	T	D	_	2	1	2	M	D	I	S	O	C	T	
polarity:								L	H	H	L	L	H	
	H	-	-	-	-	-	H	.	.	.	.	.	A	
	L	-	-	-	-	-	L	.	.	.	.	.	A	
	H	-	L	L	-	L		A	.	.	.	.	.	
	H	L	L	L	-	L		.	A	.	.	.	.	
	H	H	L	L	-	L		.	.	A	.	.	.	
	H	L	L	-	L	L		.	.	.	A	.	.	
	H	H	L	-	L	L		.	.	.	.	A	.	
	L	-	L	H	H	H		A	.	.	.	.	.	
	L	L	L	H	H	H		.	A	.	.	.	.	
	L	H	L	H	H	H		.	.	A	.	.	.	
	L	L	L	H	L	H		.	.	.	A	.	.	
	L	H	L	H	L	H		.	.	.	.	A	.	

3.3.2. Interrupt afhandeling.

Tijdens de afwikkeling van een I/O-protocol worden interrupt request signalen SRQI en SRQO gegenereerd. Beide zijn interne, actief hoge signalen. Naar de processor toe dienen de interrupts echter actief hoog (Intel) cq. actief laag (Motorola) te zijn. Bovendien willen we dat een processor in staat is het naar buiten komen van interrupts te blokkeren. Voor dit laatste is het mode-control register met twee bits uitgebreid, nl. de enable-SRQ bits (ESRQI/ESRQO).
Al deze functies kunnen weer met een PLA worden gerealiseerd. Een alternatief is implementatie met exclusive-nor's en and's :

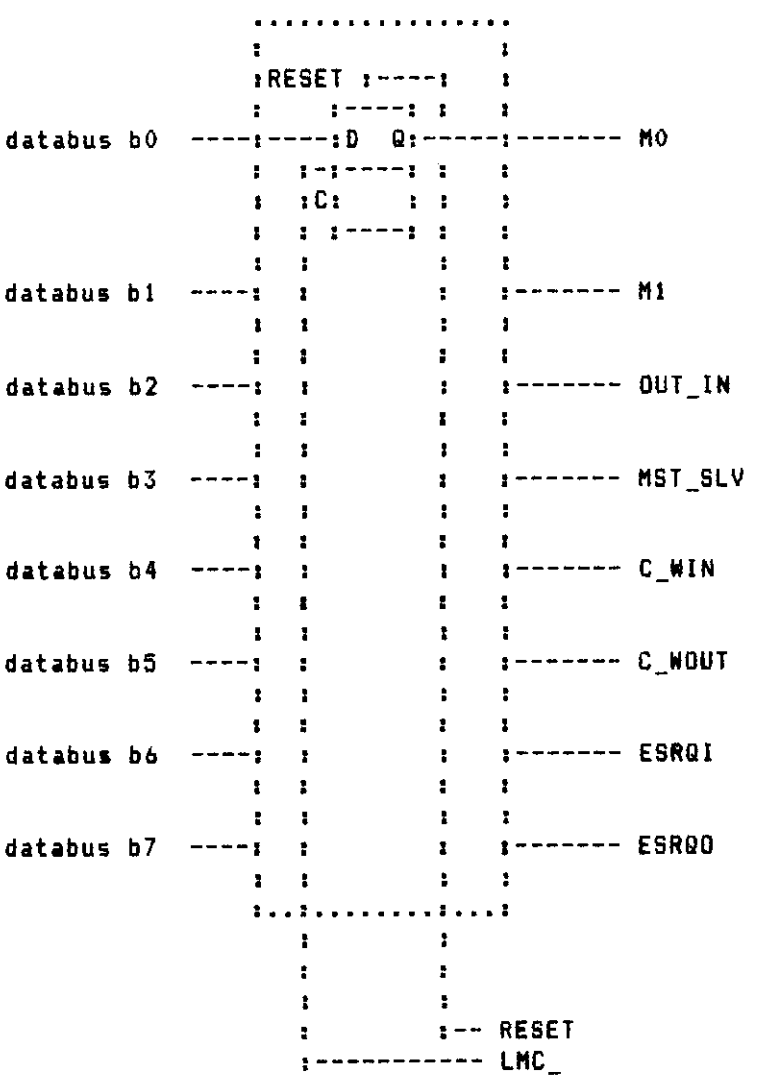


3.3.3. Statusinformatie.

Het lezen van de statusinfo door een processor komt neer op het bekijken van de SRQI- en SRQO-lijnen (zie vorige figuur). Deze informatie komt via de databus beschikbaar. De processor heeft hierdoor de mogelijkheid om te bekijken of de I/O-controller service behoeft, terwijl bv. de SRQIN uitgang van de controllerchip gedisabled is.

3.3.4. Mode-control.

De mode kan worden ingesteld via het mode-control register:



De afzonderlijke bits hebben als functie :

Mode bits:

M1 M0	: betekenis
0 0	: directe mode
0 1	: fully interlocked handshake mode
1 0	: three wire handshake mode
1 1	: bidirectional i/o-mode

Direction control:

OUT_IN	: betekenis
1	: data output
0	: data input

Master/slave control:

MST_SLV	: betekenis
0	: i/o-bouwsteen is slave
1	: i/o-bouwsteen is master

(MST_SLV heeft alleen betekenis bij bidirectionele I/O)

Continue/Wait voor input:

C_WIN	: betekenis
0	: stop input protocol tijdelijk
1	: input protocol stopt niet

Continue/Wait voor output:

C_WOUT	: betekenis
0	: stop output protocol tijdelijk
1	: output protocol stopt niet

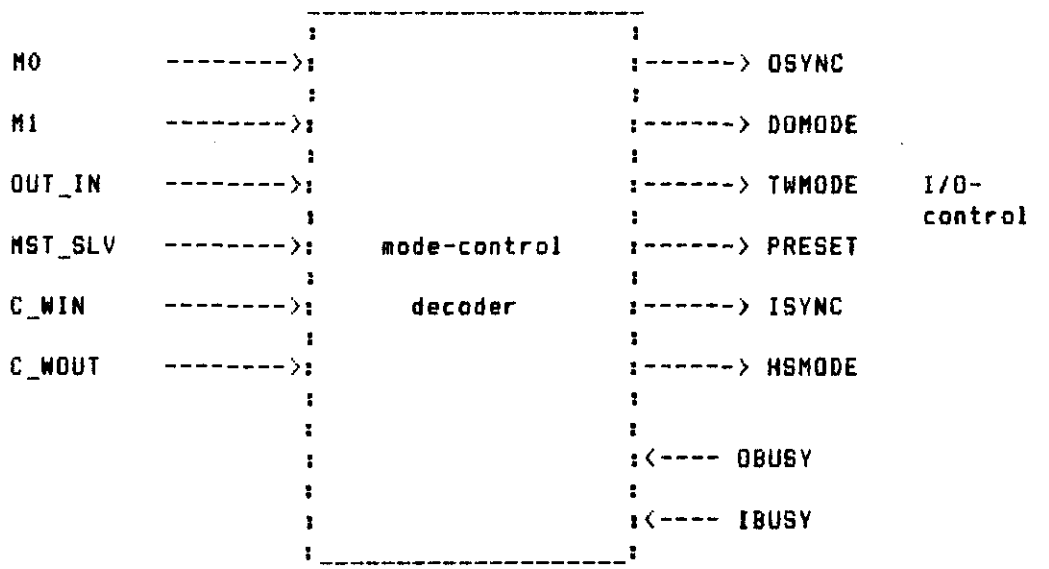
Enable input interrupt:

ESRQI	: betekenis
0	: disable input interrupt
1	: enable input interrupt

Enable output interrupt:

ESRQO	: betekenis
0	: disable output interrupt
1	: enable output interrupt

Evenals de buscontrol signalen, moet de mode informatie vertaald worden naar interne besturingssignalen :



Naast de al bekende ingangssignalen zien we bij de decoder nog de OBUSY en IBUSY ingangen. Deze twee worden benut in de bidirectionele mode met de controller geprogrammeerd als master. OBUSY en IBUSY zijn slechts dan laag (inactief) als het output-, resp. inputprotocol zich in het synchronisatiepunt bevindt, dus na afloop van een volledige protocolcyclus. Hieruit worden nu door de decoder synchronisatiesignalen afgeleid die een soepele omschakeling van input naar output en viceversa mogelijk maken.

De uitgangssignalen van de decoder hebben de volgende betekenis :

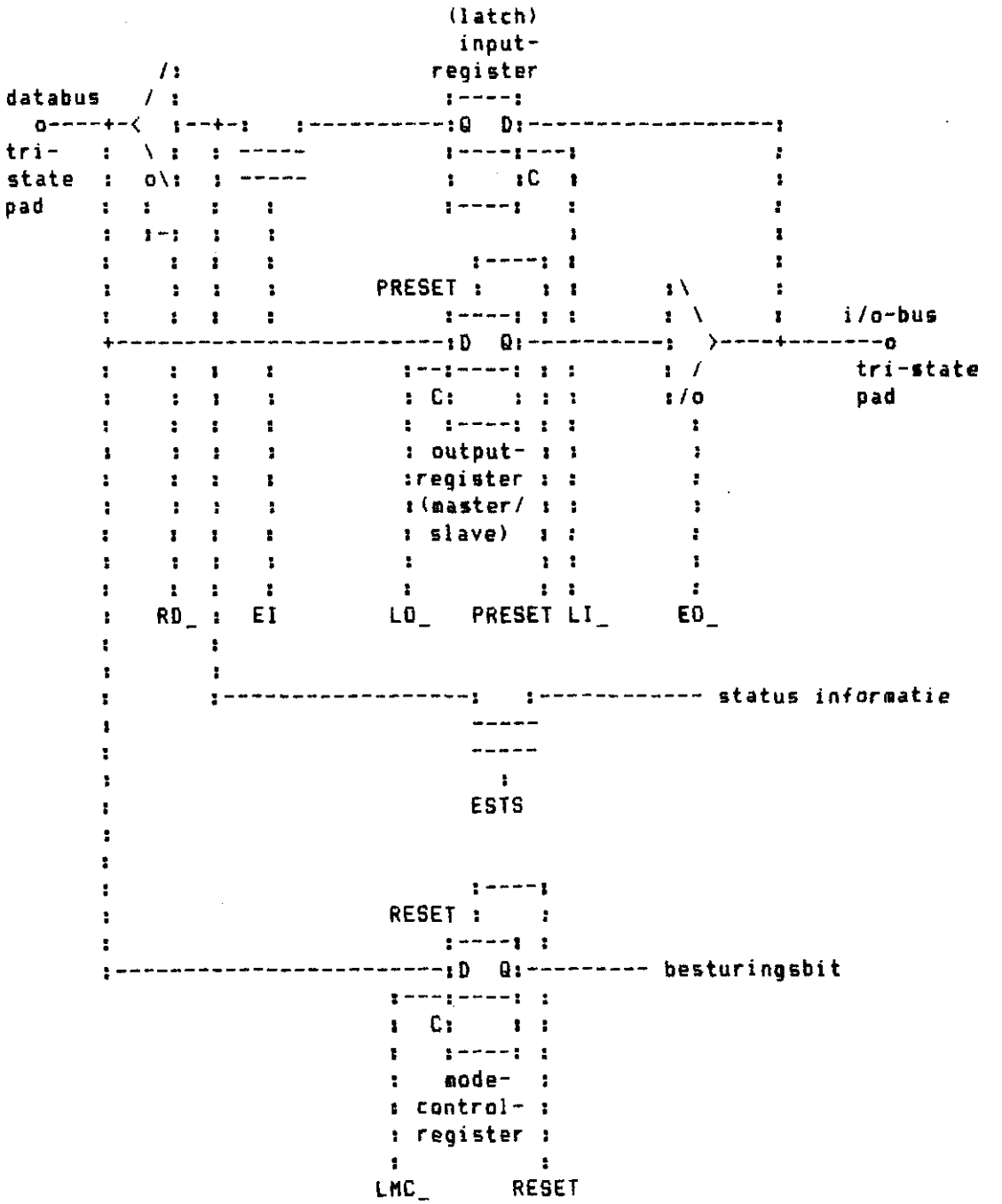
PRESET : reset voor de I/O-protocollen en selectie van direct input.
DOMODE : selectie van direct output.
TWMODE : selectie van three wire mode.
HSMODE : met preset inactief en hsmode actief worden de input handshake protocollen geselecteerd.
ISYNC :
OSYNC : synchronisatiesignalen

Ook nu kan een PLA tabel worden opgesteld voor de mode-control decoder. Hieruit zouden de volgende vergelijkingen tevoorschijn komen :

```
HSMODE =(M1M0=01 and OUT_IN=0) or          (*fully interl. input*)
        (M1M0=10 and OUT_IN=0) or          (*three wire input*)
        (M1M0=11)                          (*bidirectional*)

ISYNC  =(C_WIN=1) and                      (*input continue*)
        ((M1M0=11 and MST_SL=0) or         (*bidirectional slave*)
         (M1M0=11 and MST_SL=1 and         (*bidirectional master
          OUT_IN=0 and OBUSY=0) or           input mode while output
                                             cycle finished*)
         (M1M0=01 and OUT_IN=0) or         (*fully interlocked*)
         (M1M0=10 and OUT_IN=0))           (*three wire*)

DOMODE =(M1M0=00 and OUT_IN=1)             (*direct output*)

OSYNC  =(C_WOUT=1) and                     (*output continue*)
        ((M1M0=11 and MST_SL=0) or         (*bidirectional slave*)
         (M1M0=01 and OUT_IN=1) or         (*fully interl. output*)
          M1M0=10 and OUT_IN=1) or         (*three wire output*)
         (M1M0=11 and MST_SL=1 and         (*bidirectional master
          OUT_IN=1 and IBUSY=0))             output mode while input
                                             cycle finished*)

PRESET =(M1M0=00 and OUT_IN=0)             (*direct input*)

TWMODE =(M1M0=10)                          (*three wire mode*)
```

3.3.5. Datapad.

Al in hoofdstuk 1 is de structuur van het datapad uiteengezet. Hieronder volgt het nogmaals, met tevens de aansluitingen van statusinformatie en mode-control.



3.4. I/O-control.

Het beeld van de I/O-controller is nu compleet op de I/O-control na. Voor de implementatie maken we gebruik van de gegevens uit par. 3.2 en de toestandstabellen van hoofdstuk 2. Om zo klein mogelijke PLA's te krijgen (zo klein mogelijke chip-oppervlakte), zijn de toestandstabellen aan een optimalisatie onderworpen:

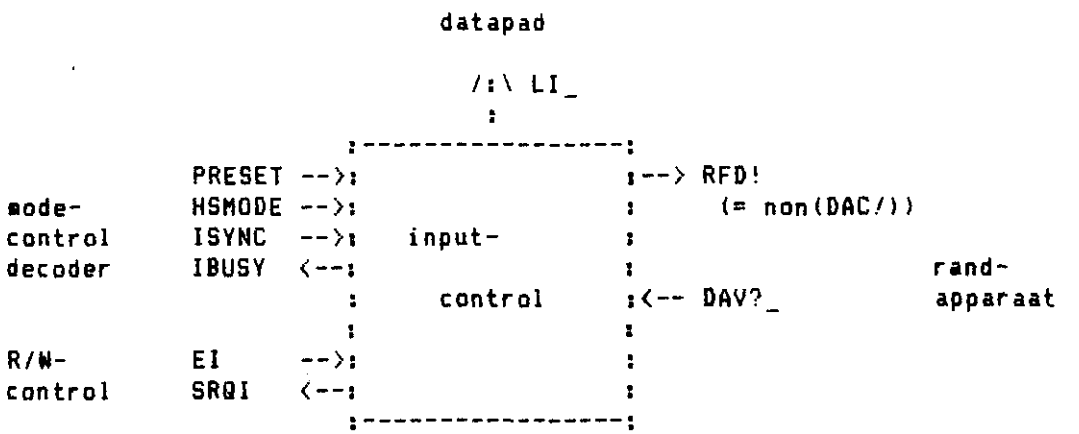
input-PLA :

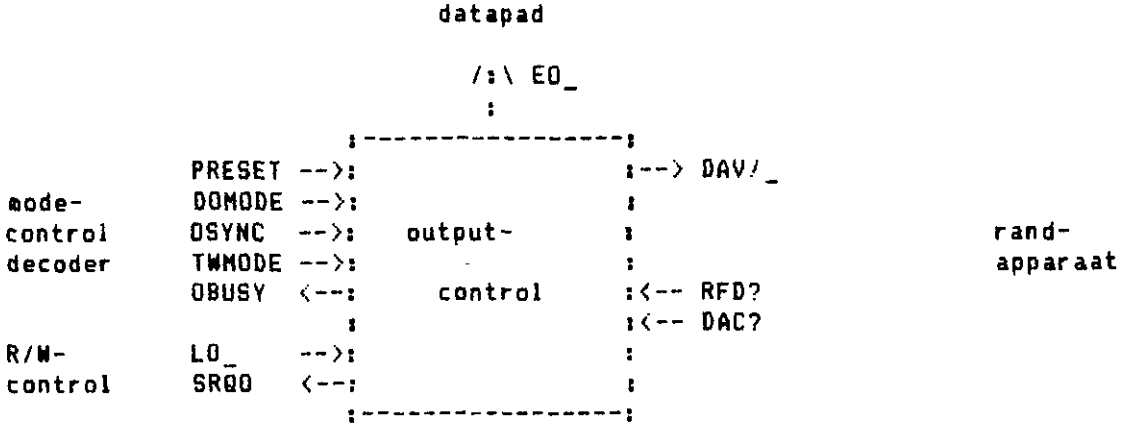
inputs											outputs			
					old state				new state					
P	H													
R	S	I	D		I				I					
E	M	S	A		B	R	S		B	R	S			
S	O	Y	V	L	U	F	R		L	U	F	R		
E	D	N	?	E	I	S	D	Q		I	S	D	Q	
T	E	C	_	I	_	Y	!	I		_	Y	!	I	
polarity										H	H	H	H	
L	-	H	H	-	H	L	-	-		.	A	A	.	
L	H	-	-	-	-	-	L	-		A	.	.	.	
L	-	-	-	L	-	-	-	H		.	A	.	A	
L	-	-	-	-	L	-	H	-		A	A	.	A	
L	-	-	-	-	H	-	H	-		.	A	A	.	
L	-	-	H	-	H	-	-	-		A	.	.	.	
L	-	-	-	H	H	-	-	-		.	A	.	.	

output-PLA :

inputs												outputs				
								old state				new state				
P	D		T													
R	O	O	W					O	D				O	D		
E	M	S	M	R		D		B	A	S			B	A	S	
S	O	Y	O	F	L	A	E	U	V	R		E	U	V	R	
E	D	N	D	D	O	C	O	S	!	Q		O	S	!	Q	
T	E	C	E	?	_	?	_	Y	_	O		_	Y	_	O	
polarity												L	H	H	H	
L	-	H	-	-	-	-	H	-	-	-		.	A	A	.	
L	-	-	-	-	H	-	-	-	-	H		.	.	A	A	
L	H	-	-	-	-	-	-	-	-	-		A	.	.	.	
L	-	-	-	H	-	-	-	H	H	-		A	A	.	.	
L	-	-	L	H	-	-	-	H	-	-		A	A	.	.	
L	-	-	H	-	-	L	-	H	L	-		A	A	.	.	
L	-	-	-	-	L	-	-	H	-	-		.	.	A	.	
L	-	-	-	H	-	-	H	H	-	-		.	.	.	A	

Bij de input-PLA willen we dat als PRESET=1 de uitgangsvector gelijk wordt aan 0,0,0,0 . Om dit te bereiken, is in de optimalisatie bij LI_ een H-polariteit gekozen. Bij de output-PLA is voor DAV/_ de H-polariteit aangenomen (uitgangsvector 1,0,0,0 bij PRESET=1).
De input-, en output-control hebben een register-logica-register structuur. Globaal weergegeven zien ze er als volgt uit :





3.5. Timing overwegingen.

Aan de processorzijde van de controller kunnen bij de handshake protocollen moeilijkheden optreden, die een correcte data-overdracht beletten. Deze problemen ontstaan steeds door de tijd die de hardware van de controller nodig heeft om een processorsignaal te verwerken.
Zo moet een I/O-write_ door de controller worden vertaald naar activering van LO_. Hiervoor is een bepaalde verwerkingstijd TAU nodig. Zodoende zal de laag - hoog overgang van LO_ pas TAU seconden volgen na het inactief worden van I/O-write_. Omdat de data met de opgaande flank van LO_ van de databus wordt overgenomen, moet worden geeist dat de data nog minstens gedurende TAU geldig blijft na het wegvallen van I/O-write_.
Een dergelijk probleem bestaat ook bij het lezen van data met I/O-read_. Na het wegvallen van I/O-read_ blijven RD_ en EI nog even actief. Dit kan aanleiding geven tot contentie op de databus.
Ook de interrupt afhandeling kan last hebben van verwerkingstijden. Een interrupt request zal door een I/O-read_ of I/O-write_ worden gevolgd. Daarom moet de request worden weggenomen tijdens de read- of write-puls. De pulsduur moet dus groter zijn dan de propagatietijd in de controller. Deze laatste bestaat uit twee delen:

decodering van read (write) naar RD_,EO (LO_),

verwerking van EI (LO_) in de I/O-control, zodat request wordt weggenomen

Opmerkingen:

Bij bovenstaande beschouwingen spelen naast het controller-ontwerp ook de gebruikte integratie technologie en de keuze van de processor een rol.

Als handshaking wordt gebruikt voor data-overdracht is de interactie tussen bron en bestemming intensiever. Hierdoor daalt de snelheid waarmee de informatie wordt uitgewisseld, maar stijgt de betrouwbaarheid.

SLOTWOORD.

Het basisontwerp is met het afsluiten van hoofdstuk 3 voltooid. Uitgangspunten waren steeds dat het ontwerp een flexibele en overzichtelijke structuur moest hebben die geschikt was voor integratie.

Om dit te realiseren is gekozen voor een modulaire opbouw, waarbij de modulen parallel werkzaam zijn en worden gekoppeld door registers. Op deze wijze kunnen gemakkelijk wijzigingen in het ontwerp worden aangebracht. Bovendien is uitbreiding met nieuwe modulen eenvoudig.
Voor de realisatie van de modulen is grotendeels gebruik gemaakt van PLA's.
De weg naar het uiteindelijke IC is echter nog lang. Allereerst zal met ontwerp worden getoetst met een simulatieprogramma. Als de resultaten naar wens zijn, zou kunnen worden begonnen met het ontwerpen van een IC-layout, bijvoorbeeld op basis van een NMOS-proces.
Een andere (snellere) methode is gebruik maken van gate-arrays. Dit principe is toegepast op de I/O-controller. Nadere informatie is te vinden in lit.[7].

Lijst van notaties.

*actief lage signalen worden met een underline (_) aangeduid.

notatie:	betekenis:	paragraaf:
C_D	control- non data	3.3.1
CE_	chip enable	3.3.1
CTRL1	chip selectie	3.3.1
CTRL2		
C_WIN	continue- non wait input	1.4
C_WOUT	continue- non wait output	1.3
DAC	data accepted	1.5
DAV_	data available	1.3
DOMODE	direct output mode	2.2.3
EI	enable input	1.2
EO_	enable output	1.1
ESRQI	enable interrupt request input	3.3.2
ESRQO	enable interrupt request output	3.3.2
ESTS	enable status	3.3.1
HSMODE	handshaske mode (input)	2.2.3
IBUSY	input protocol busy	2.2.3
INT_MOT	intel- non motorola	3.3.1
I/O-read	processor read	1.2
I/O-write	processor write	1.1
ISYNC	input handshake synchronisatie	2.2.3
LI_	load inputdata	1.2
LMC_	load mode control	3.3.1
LO_	load outputdata	1.1
M0	mode bit	3.3.4
M1		
MST_SLV	master- non slave	3.3.4
OBUSY	output protocol busy	2.2.3
OSYNC	output handshake synchronisatie	2.2.3
OUT_IN	output- non input	3.3.4
PRESET	reset I/O-protocollen	1.1, 2.2.3
RD_	read inputreg. of status	1.2
RESET	mode-control reset	3.3.1
RFD	ready for data	1.3
RSTI_M	reset intel- non motorola	3.3.1
SRQI	interrupt request (input)	1.4
SRQO	interrupt request (output)	1.3
TWMODE	three wire mode	2.2.2

Literatuur

(1) Mead, C.A. and L.A. Conway
INTRODUCTION TO VLSI SYSTEMS.
Reading, Mass.: Addison-Wesley, 1980.
Addison-Wesley series in computer science.

(2) Peatman, J.B.
DIGITAL HARDWARE DESIGN.
New York: McGraw-Hill, 1980.

(3) Rozendaal, L.T.
REALISATIE VAN EEN MULTIFUNCTIONELE I/O-CONTROLLER M.B.V. EEN GATE-ARRAY.
Stageverslag. Vakgroep Digitale Systemen, Afdeling der Elektrotechniek, Technische Hogeschool Eindhoven, aug. 1984.

EINDHOVEN UNIVERSITY OF TECHNOLOGY
THE NETHERLANDS
DEPARTMENT OF ELECTRICAL ENGINEERING Coden: TEUEDE

Eindhoven University of Technology Research Reports (ISSN 0167-9708)

(127) Damen, A.A.H., P.M.J. Van den Hof and A.K. Hajdasiński
THE PAGE MATRIX: An excellent tool for noise filtering of Markov parameters, order testing and realization.
EUT Report 82-E-127. 1982. ISBN 90-6144-127-7

(128) Nicola, V.F.
MARKOVIAN MODELS OF A TRANSACTIONAL SYSTEM SUPPORTED BY CHECKPOINTING AND RECOVERY STRATEGIES. Part 1: A model with state-dependent parameters.
EUT Report 82-E-128. 1982. ISBN 90-6144-128-5

(129) Nicola, V.F.
MARKOVIAN MODELS OF A TRANSACTIONAL SYSTEM SUPPORTED BY CHECKPOINTING AND RECOVERY STRATEGIES. Part 2: A model with a specified number of completed transactions between checkpoints.
EUT Report 82-E-129. 1982. ISBN 90-6144-129-3

(130) Lemmens, W.J.M.
THE PAP PREPROCESSOR: A precompiler for a language for concurrent processing on a multiprocessor system.
EUT Report 82-E-130. 1982. ISBN 90-6144-130-7

(131) Eijnden, P.M.C.M. van den, H.M.J.M. Dortmans, J.P. Kemper and M.P.J. Stevens
JOBHANDLING IN A NETWORK OF DISTRIBUTED PROCESSORS.
EUT Report 82-E-131. 1982. ISBN 90-6144-131-5

(132) Verlijsdonk, A.P.
ON THE APPLICATION OF BIPHASE CODING IN DATA COMMUNICATION SYSTEMS.
EUT Report 82-E-132. 1982. ISBN 90-6144-132-3

(133) Heijnen, C.J.H. en B.H. van Roy
METEN EN BEREKENEN VAN PARAMETERS BIJ HET SILOX-DIFFUSIEPROCES.
EUT Report 83-E-133. 1983. ISBN 90-6144-133-1

(134) Roer, Th.G. van de and S.C. van Someren Gréve
A METHOD FOR SOLVING BOLTZMANN'S EQUATION IN SEMICONDUCTORS BY EXPANSION IN LEGENDRE POLYNOMIALS.
EUT Report 83-E-134. 1983. ISBN 90-6144-134-X

(135) Ven, H.H. van de
TIME-OPTIMAL CONTROL OF A CRANE.
EUT Report 83-E-135. 1983. ISBN 90-6144-135-8

(136) Huber, C. and W.J. Bogers
THE SCHULER PRINCIPLE: A discussion of some facts and misconceptions.
EUT Report 83-E-136. 1983. ISBN 90-6144-136-6

(137) Daalder, J.E. and E.F. Schreurs
ARCING PHENOMENA IN HIGH VOLTAGE FUSES.
EUT Report 83-E-137. 1983. ISBN 90-6144-137-4

EINDHOVEN UNIVERSITY OF TECHNOLOGY
THE NETHERLANDS
DEPARTMENT OF ELECTRICAL ENGINEERING Coden: TEUEDE

Eindhoven University of Technology Research Reports (ISSN 0167-9708):

(138) Nicola, V.F.
A SINGLE SERVER QUEUE WITH MIXED TYPES OF INTERRUPTIONS: Application to the modelling of checkpointing and recovery in a transactional system.
EUT Report 83-E-138. 1983. ISBN 90-6144-138-2

(139) Arts, J.G.A. and W.F.H. Merck
TWO-DIMENSIONAL MHD BOUNDARY LAYERS IN ARGON-CESIUM PLASMAS.
EUT Report 83-E-139. 1983. ISBN 90-6144-139-0

(140) Willems, F.M.J.
COMPUTATION OF THE WYNER-ZIV RATE-DISTORTION FUNCTION.
EUT Report 83-E-140. 1983. ISBN 90-6144-140-4

(141) Heuvel, W.M.C. van den and J.E. Daalder, M.J.M. Boone, L.A.H. Wilmes
INTERRUPTION OF A DRY-TYPE TRANSFORMER IN NO-LOAD BY A VACUUM CIRCUIT-BREAKER.
EUT Report 83-E-141. 1983. ISBN 90-6144-141-2

(142) Fronczak, J.
DATA COMMUNICATIONS IN THE MOBILE RADIO CHANNEL.
EUT Report 83-E-142. 1983. ISBN 90-6144-142-0

(143) Stevens, M.P.J. en M.P.M. van Loon
EEN MULTIFUNCTIONELE I/O-BOUWSTEEN.
EUT Report 84-E-143. 1984. ISBN 90-6144-143-9