ONTWIKKELING VAN PML SIMULASIE-SAGTEWARE

deur

GYSBERT JOHANNES BOOYSEN

Skripsie voorgelê ter gedeeltelike voldoening aan die vereistes vir:

MEESTERSDIPLOMA IN TEGNOLOGIE

Elektriese Ingenieurswese (Swakstroom)

In die Fakulteit Ingenieurswese aan die Tecnikon O.V.S.

Datum van inhandiging: JUNIE 1994

Studieleier: Mnr. G.D. Jordaan

BEDANKINGS

Ek wil graag die volgende persone en instansies bedank vir hulle bydrae tot die voltooing van die projek:

My studieleier, Mnr. G.D. Jordaan, vir sy hulp en leiding gedurende die verloop van die projek.

Technikon O.V.S. vir die bereidwilligheid om die projek te hanteer.

Telkom, vir die geleenthede wat hulle aan my gebied het.

My ouers vir die opvoeding en geleenthede aan my verskaf om tot op hierdie vlak te vorder.

My vrou, Hannie, vir haar ondersteuning en aanvaarding vir die tyd wat ek van haar moes ontneem om die projek te voltooi.

UITTREKSEL

Programmeerbare logika bied 'n gunstige alternatief vir 'n wye verskeidenheid van logika toepassings en word steeds belangriker in digitale ontwerpe.

Die projek het die ontwikkeling van 'n volledige stel sagteware vir die implementering van verskeie programmeerbare matriks logika (PML) toestelle behels. Die sagteware behels die volgende programme:

- *PML-databasis sagteware:* Tydens die bepaling van 'n sekeringkaart, voorbereiding van 'n JEDEC lêer en die simulering van die verwagte werking van 'n PML word 'n groot hoeveelheid inligting van die besondere komponent benodig. Sodanige inligting word aan die stelsel beskikbaar gestel deur die uitvoering van hierdie program.

- S*ekeringkaart sagteware:* Die gebruiker van 'n PML spesifiseer die verlangde werking daarvan d.m.v. Boole uitdrukkings. Hierdie program maak die invoer en wysiging van sodanige uitdrukkings moontlik. Dit lewer ook 'n skematiese voorstelling van die geprogrammeerde PML vir evaluering deur die gebruiker. Kommersiële PML programmeerders benodig JEDEC lêers as inset. Sodanige lêers word ook deur hierdie sagteware geskep en gestoor.

• *Simulasie sagteware:* Die gebruiker van 'n PML kan die korrektheid van die Boole uitdrukkings verifieer deur die verwagte werking van 'n PML wat daarvolgens geprogrammeer sou word, te simuleer. Sodanige simulasie word aan die hand van die betrokke JEDEC lêer gedoen. Hiertydens word insetstimulante op die kring na willekeur geselekteer en die verwagte werking van die PML word in die vorm van 'n waarheidstabel gelewer.

Evaluering van sagteware: Die korrekte werking van die sagteware is bevestig deur die praktiese implementering van 'n verskeidenheid ontwerpe. Verskeie van hierdie kringe is in onverwante projekte gebruik en bevredigende resultate is daarmee behaal.

Besondere kennis aangaande onderstaande is ingewin:

• Eienskappe en programmering van PMLs.

• Formaat en samestelling van JEDEC lêers.

• Die beginsel en toepassing van simulasie van elektroniese kringe in die algemeen - en PMLs in die besonder.

SYNOPSIS

Programmable logic offers a favourable solution to a variety of logic applications and is of increasing importance in digital designs.

The work in this project concerns the development of a complete set of software for the implementation of a variety of Programmable array logic (PAL) devices. The software consists of the following programs:

- *PAL database software:* A large amount of information regarding the specific component is needed for the determination of the fuse map, preparation of the JEDEC file and simulation of the expected operation of a PAL. The necessary information is made available to the system by the execution of this program.

- *Fuse map software:* The user of a PAL specifies the desired operation thereof by means of Boolean expressions. The program which has been developed makes the input and modification of such expressions possible. It also produce a schematic representation of the programmed PAL for evaluation by the user. Commercial PAL programmers use JEDEC files as input. Such files can also be compiled and stored with this software.

• *Simulation software:* The user of a PAL can verify the correctness of a Boolean expression by simulating the expected operation of a PAL which has been programmed in this way. This simulation is performed by using the relevant JEDEC file. During simulation, input stimuli for the circuit are selected randomly by the operator, and the expected operation of the PAL is represented in the form of a truth table.

Evaluation of the software: Correct operation of the software was verified by the practical implementation of a variety of designs. A number of these circuits were used in unrelated projects and satisfactory results have been achieved.

Particular knowledge regarding the following has been obtained:

• Characteristics and programming of PALs.

• Format and construction of JEDEC files.

• Principles and applications of simulation of electronic circuits in general and PALs in particular.

INHOUD

LYS VAN FIGURE EN TABELLE

HOOFSTUK 2

HOOFSTUK 3

HOOFSTUK 4

HOOFSTUK 5

LYS VAN TABELLE

HOOFSTUK 2

HOOFSTUK 5

HOOFSTUK 1

INLEIDING

1.1 OORSIG

'n Programmeerbare logika toestel (programmable logic device) is 'n grootskaalse integrasie (LSI) toestel waarvan die kringstruktuur deur die ontwerper verander kan word vir 'n spesifieke toepassing. Daar word aanvaar dat programmeerbare logika elke dag belangriker word in digitale ontwerpe.

1.2 PROBLEEMSTELLING

Wanneer 'n programmeerbare matriks logika (programmable array logic) toestel geprogrammeer word met 'n programmeerbare matriks logika (PML) programmeerder (en die apparaat verifieer dat die toestel wel korrek geprogrammeer is) gebeur dit tog soms dat wanneer hierdie toestel in gebruik geneem word dit nie funksioneer soos verwag is nie.

Om hierdie probleem te oorkom is dit wenslik om oor 'n program te beskik wat in staat is om die werking van die PML ontwerp te simuleer. Dit sal dus toelaat dat 'n baie komplekse ontwerp in 'n vroeë stadium geevalueer kan word

en verhoed dat ontwerpfoute by die hardewarevlak gediagnoseer moet word.

1.3 DOELWIT VAN PROJEK

Die hoofdoel van die projek was om sagteware te ontwikkel wat die werking van 'n PML toestel kan simuleer. Dit het ook die ontwikkeling van sagteware behels om 'n kring as 'n sekeringkaart voor te stel en die sekeringkaart om te skakel na 'n JEDEC (Joint Electronic Device Engineering Council) lêer.

Die werking van die PML ontwerp word vanaf die JEDEC lêer gesimuleer. Die JEDEC lêer word vanaf die rekenaar na die PML programeerder d.m.v. 'n RS232 koppeling oorgedra ten einde die PML te programmeer.

1.4 HIPOTESE

Na die mening van die navorser kon sagteware ontwikkel word om die volgende funksies te verrig:

- 'n JEDEC lêer vir die programmering van 'n PML te genereer ter realisering van 'n stel Boole uitdrukkings.
- Die verwagte werking van die PML aan die hand van sodanige lêer te simuleer.

1.5 BELANGRIKHEID VAN PROJEK

Programmeerbare logika is 'n tegnologie wat tans 'n vlak van ontwikkeling en groei ondergaan wat dit stewig vestig as die mees betekenisvolle vooruitgang in die elektroniese industrie sedert die koms van die mikroverwerker (Levy,1986: 31).

Die ontwikkeling van programmeerbare logika het vinnig gegroei en teen 1990 was daar reeds sowat 3000 toestelle op die mark wat die meer as 300 afsonderlike argitekture en 'n verskeidenheid van name ondersteun (Levy, 1991: 56).

As gevolg van hierdie groeiende belang in programmerbare logika tegnieke is dit van die uiterste belang dat Suid-Afrikaanse ingenieurs en tegnici op hoogte bly van hierdie ontwikkeling. Die primêre doel van hierdie projek is om gespesialiseerde kennis in die verband op te doen.

1.6 METODE VAN NAVORSING

Die projek is in vier fases uitgevoer, nl:

- Ontwikkeling van PML-databasissagteware.
- Ontwikkeling van sekeringkaartsagteware.
- Ontwikkeling van simulasie-sagteware.
- Evaluering van die sagteware.

1.6.1 ONTWIKKELING VAN PML-DATABASISSAGTEWARE

'n Program is ontwikkel om 'n databasis daar te stel wat al die inligting omtrent elke PML wat met hierdie stelsel gebruik kan word te stoor. Hierdie data word benodig om die JEDEC lêer saam te stel sowel as vir die simulasieproses.

1.6.2 ONTWIKKELING VAN SEKERINGKAARTSAGTEWARE

'n PML toestel bevat 'n aantal sekerings wat in 'n bepaalde patroon voorkom. Deur hierdie sekerings in 'n vooraf bepaalde patroon te smelt kan 'n verlangde logiese werking verkry word. Daar word na hierdie patroon as die sekeringkaart verwys.

Sagteware is ontwikkel om so 'n sekeringskaart saam te stel en dan 'n JEDEC lêer te kompileer wat gebruik word om die PML te programmeer en om die werking daarvan te simuleer.

1.6.3 ONTWIKKELING VAN SIMULASIE-SAGTEWARE

'n Simulasieprogram is ontwikkel om die werking van 'n PML te simuleer. Hierdie program maak gebruik van die JEDEC lêer waarvolgens die PML geprogrammeer gaan word,

data in die PML-databasis en toetsvektore om die werking van die PML te simuleer.

Die toetsvektore word deur die operateur in die vorm van ene en nulle verskaf. Hierdie vektore en die resultate word dan in die vorm van 'n waarheidstabel op die skerm vertoon. 'n Uitdruk kan ook, indien verlang, verkry word.

1.6.4 EVALUERING VAN DIE SAGTEWARE

Na voltooiing van die sagteware was dit moontlik om die werking van verskeie PML kring ontwerpe te simuleer - wat wel met groot sukses gedoen is.

Die JEDEC lêers wat met die sagteware gekompileer is, is ook gebruik om 'n aantal PMLs te programmeer en die werking daarvan is getoets. Dit is dus 'n indirekte evaluering van die geldigheid van die JEDEC lêer sagteware.

HOOFSTUK 2

PML AS PROGRAMMEERBARE LOGIKA TOESTEL

2.1 INLEIDING

Programmeerbare logika toestelle (PLTe) is besig om in gebiede in te beweeg wat ontoeganklik was vir konvensionele logika. Alhoewel tradisionele logikafamilies waarskynlik nog vir baie jare sal bestaan word die ontwikkellende PLT tegnologie al hoe meer deur ontwerpers aangewend as 'n oplossing tot ontwerpprobleme (Frost, 1989b: 666).

PLTe is nie beperk tot enige spesiale toepassingsveld nie. Elke ontwerper van elektroniese kringe moet dit sien as 'n potensiële oplossing vir sy ontwerpprobleem (Bostock, 1987: 140).

PLTe sal waarskynlik mettertyd ontvou in kompleksiteit tot die vlak waarteen toepassing spesifieke geïntegreerde kringe (TSGK) tot onlangs aangebied is, en dit moontlik selfs verbysteek.

2.2 DIE AARD VAN PROGRAMMEERBARE LOGIKA TOESTELLE

Programmeerbare logika toestel is 'n algemene term vir alle toestelle wat deur die eindgebruiker geprogrammeer kan word (Frost, 1989a: 499). Die term programmeerbaar impliseer dat die funksie van die kring gespesifiseer kan word nadat die geïntegreerde kring vervaardig is (Prosser & Winkel, 1987: 143).

Volgens hierdie beginsel is mikrorekenaars ook programmeerbare logika toestelle, maar die definisie word beperk tot logika komponente met programmeerbare EN en/of OF matrikse. Die PLT laat die ontwerper dus toe om som-van-produkte funksies met 'n groot aantal veranderlikes te realiseer.

2.3 WAAROM PROGRAMMEERBARE LOGIKA?

Baie logika toepassings is te gekompliseerd vir mediumskaalse integrasie (MSI) toestelle en nie geskik vir die gebruik van 'n mikroverwerker nie. Programmeerbare logika toestelle bied dus die gunstigste oplossing tot 'n verskeidenheid van logika toepassings. PLTe kombineer programmeringsbuigsaamheid met hoë spoed en 'n omvattende keuse van koppelvlakopsies teen relatief lae koste (Lakshminarayanan, 1988: 4).

PLTe het dieselfde buigsame argitektuur as doelspesifieke geïntegreerde kringe (custom ICs), maar is meer geredelik beskikbaar en is goedkoper en makliker om te gebruik. Deur van PLTe gebruik te maak kan 'n ekwivalente logika ontwerp voltooi word teen 'n fraksie van die tyd benodig vir ander moontlike oplossings - wat weer die algehele ontwikkelingstyd van die produk (waarvan die PLT deel is) drasties kan verminder (Lakshminarayanan, 1988: 5).

Deur van 'n PLT in plaas van doelspesifieke geïntegreerde kringe gebruik te maak, kan kringe byna onmiddellik hervervaardig word deur eenvoudig 'n nuwe sekeringskaart saam te stel en dit te gebruik om die toestel te programmeer (Meyer, 1988b: 67).

Volgens Bolton (1990: 20) dra die volgende punte by tot die toenemende gebruik van programmeerbare logika:

- Stelsel ontwerp hoef nie op hekvlak uitgevoer te word nie. Funksionele beskrywings kan direk in 'n PLT beskrywing saamgestel word.
- Subsisteme met 'n kompleksiteit van 1000-2000 twee-insethekke kan in 'n enkele pakket geakkomodeer word.
- Met herprogrammeerbare PLTe, kan foute tydens die prototipe stadium opgespoor en herstel word sonder die verlies van 'n duur komponent.
- Moderne PLTe se spoed en kragvebruik is vergelykbaar met die van hoë spoed CMOS logika.
- 'n Klein kapitale belegging word verlang.
- Ontwerptyd is baie korter as met ander tipes van TSGK.

- Die funksie van 'n bestaande stelsel kan gewysig word deur slegs die PLT te vervang of te herprogrammeer.
- Gedrukte kringbord ontwerp kan vereenvoudig word, as gevolg van die vryheid van seintoewysings aan penne.

2.4 PLT KARAKTERISTIEKE

PLTe is 'n hele familie toestelle met 'n gemeenskaplike eienskap, nl. dat die basiese struktuur van almal bestaan uit 'n matriks van EN-hekke en OF-hekke soos geïllustreer in fig. 2.1.

Geïntegreerde kringe het aan die kringontwerper twee keuses gelaat vir die konstruksie van digitale kringe. Die een keuse is vaste funksie toestelle - soos die 7400 reeks - en die ander is doelspesifieke geïntegreerde kringe. Die ontwikkeling van PLTe bied nou aan die ontwerper 'n derde opsie, naamlik een tussen vaste funksie toestelle en doelspesifieke geïntegreerde kringe.

'n PLT kan gedefinieer word as 'n toestel met 'n ongebonde logikamatriks. Om hierdie rede beskou die meeste ontwerpers dit as 'n semi-unieke toestel, omdat die ontwerper dit kan programmeer na gelang van sy eie spesifikasies (Burton, 1990: 1). Voorbeelde van PLTe is Programmeerbare lees alleen geheues (PLAGs), Programmeerbare logika matrikse (PLMe) en Programmeerbare matriks logika (PML) toestelle.

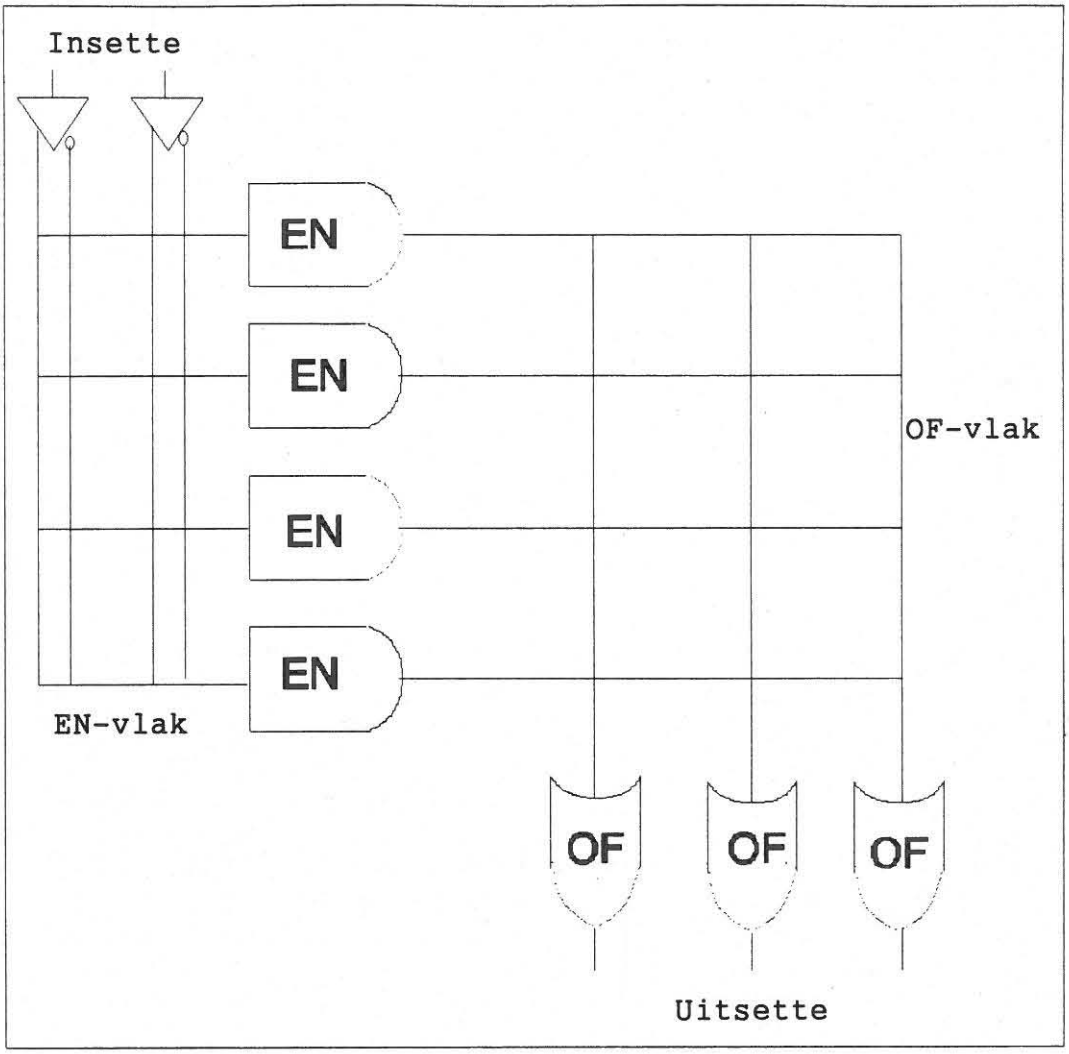


Fig. 2.1: Basiese struktuur van 'n PLT.

Om die begrip t.o.v PLTe te vergemaklik is 'n spesiale konvensie aangeneem. Figuur 2.2(a) is 'n voorbeeld van 'n drie inset EN-hek volgens die konvensionele notasie. Figure 2.2(b) tot 2.2(d) is 'n soortgelyke EN-hek, maar volgens die PLT notasie. Daar moet op gelet word dat die hek met slegs een insetlyn getoon word. Daar word na hierdie lyn verwys as die produklyn. Die loodregte lyne A,B en C word as die insette beskou.

Figuur 2.2(b) is 'n voorbeeld van 'n hek waar al die sekerings gesmelt is, terwyl figuur 2.2(c) 'n hek toon waar al die sekerings nog ongeskonde is. Figuur 2.2(d) wys 'n ongebruikte hek waarvan al die sekerings nog ongeskonde is. 'n Asterisk (*) verteenwoordig 'n ongeskonde sekering wat dan daardie inset deel maak van die produkterm. Die afwesigheid van 'n asterisk verteenwoordig 'n gesmelte sekering wat daardie inset elimineer van die produkterm.

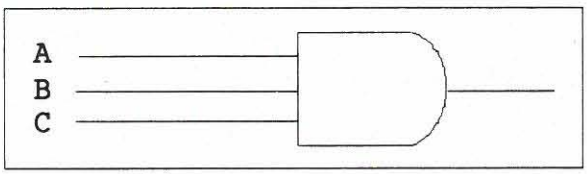


Fig. 2.2(a): EN-hek simbool

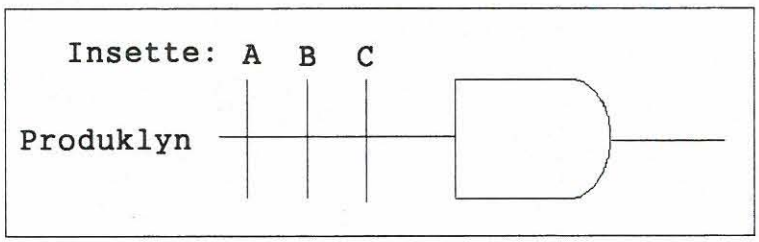


Fig. 2.2(b): PLT simbool waar al die sekerings gesmelt is.

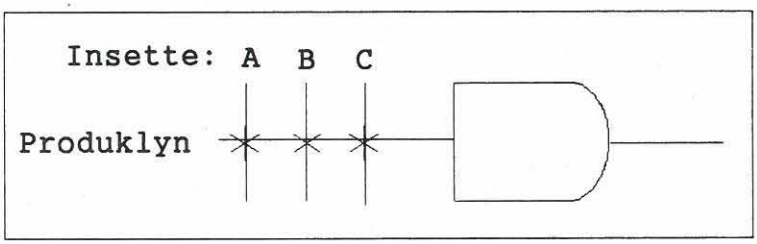


Fig. 2.2(c): PLT simbool waar al die sekerings nog heel is.

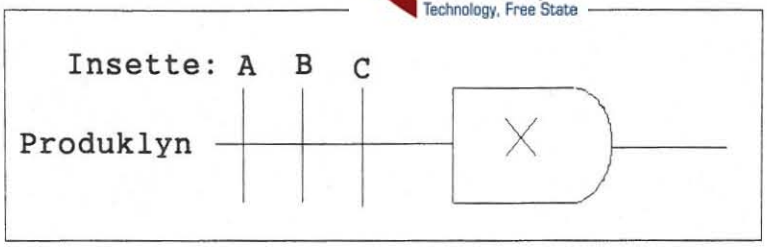


Fig. 2.2(d): PLT simbool wat aandui dat EN-hek nie gebruik word nie.

In figuur 2.3 word van basiese PLT simbole gebruik gemaak om 'n twee inset programmeerbare EN-matriks wat 'n OF-hek voer te illustreer. Die uitset van die OF-hek is volgens die funksie $F = A\bar{B} + \bar{A}B$ geprogrammeer.

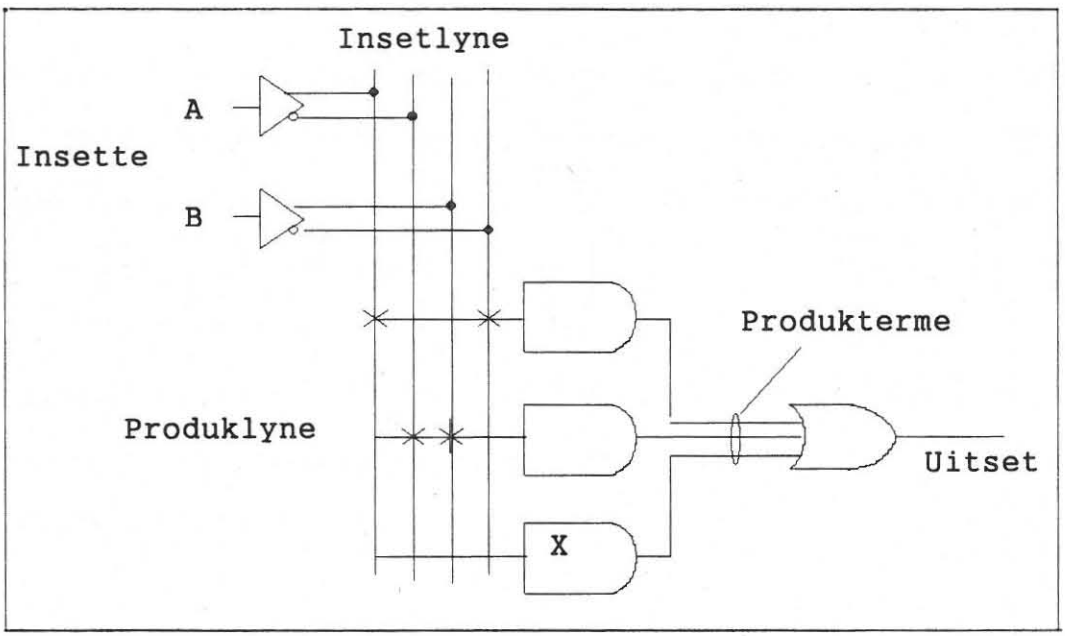


Fig. 2.3: Illustrasie van 'n basiese PLT.

2.5 PROGRAMMEERBARE MATRIKS LOGIKA (PML)

In 1978 het Monolithic Memories 'n programmeerbare matriks logika familie van omtrent vyftien toestelle

bekend gestel. Die doel was om dit moontlik te maak om die bestaande logika toestelle in 'n gegewe kring te verminder deur dit te vervang met PMLs. 'n Enkele PML kan 5 tot 10 logika toestelle in 'n kring vervang (Klingman, 1982: 96). Behalwe dat die PMLs deur die gebruiker geprogrammeer kan word, bied dit ook die volgende: tristaat uitsette, register uitsette en programmeerbare inset/uitset (dit beteken dat die uitsetpenne geprogrammeer kan word om as 'n inset of uitset te funksioneer). Hierdie karateristieke vergemaklik koppeling met mikroverwerkeenhede.

Die gebruik van hek matriks logika (gate arrays) was gewoonlik 'n groot uitdaging vir die onervare ontwerper. Selfs vir die ervare ontwerper was die ontwerpsiklus soms lank en duur terwyl die eindresultaat soms onbuigsaam en selfs foutief was. In teenstelling hiermee voorsien programmeerbare matriks logika 'n struktuurontwerp benadering wat eenvoudig en maklik is om te leer (Greiner & Schmitz, 1984: 243).

Die PML voorsien 'n sistematiese manier om komplekse logikafunksies te genereer. Programmeerbare matriks logika is 'n spesiale geval van programmeerbare logika matriks wat 'n vaste OF matriks het i.p.v. die programmeerbare OF matriks soos by gewone PLMs. Omdat die produkterme van 'n PML gebonde is tot 'n beperkte aantal vir elke uitset is die PML se gebruike meer beperk as die

van 'n PLM. PMLs is egter baie goedkoper en is makliker om te programmeer (Lam & O'Malley, 1988: 195).

Die programmeerbare lees alleen geheue (PLAG) voorsien 'n vooraf bepaalde stel produkterme en die ontwerper kan die gedrag van elke som-van-produkte spesifiseer. Hierdie struktuur is bruikbaar as 'n geheue en vir logika bewerkings wat voldekodering van insette verlang. Tog het die PLAG beperkte toepassings ten opsigte van algemene logika gebruike a.g.v. die beperkte aantal insetveranderlikes en die formaat van die uitdrukkings wat dit kan realiseer.

By die PML kan die ontwerper die gedrag van die produkterme spesifiseer, maar die Boolse sommering van produkterme vir realisering van 'n spesifieke uitdruking is nie programmeerbaar nie. Die programmeerbare produkmatriks en die nie-programmeerbare OF-matriks voldoen baie goed aan die vereistes van digitale ontwerpe (Prosser & Winkel, 1987: 147).

Die buigsaamheid van PMLs is tussen die van PLMe en PLAGs. In die praktyk is die PML egter by vêrre die mees bruikbare van die drie toestelle (Prosser & Winkel, 1987: 147). Eienskappe soos I/U penne, registers en veranderlike klok funksies maak PMLs ideaal vir implementering van logika funksies (Lakhminarayanan, 1988: 5).

2.5.1 PML BENAMING

Anders as by ander PLTe kan baie omtrent die stuktuur van die PML vanaf die toestel benaming afgelei word. PMLs word as volg genommer: PALmXn, waar die m die aantal insette na die EN-hek matriks voorstel, n is die aantal uitsette en X definieer die uitset struktuur.

Die algemene letters wat by PMLs gebruik word om die uitset struktuur te definieer is:

- H - aktief HOË uitset
- L - aktief LAE uitset
- R - uitset met wipkring
- P - programmeerbare polariteit
- S - gedeelde EN-hekke

'n Twintigpen PML het gewoonlik 18 logika verbindings. Indien m+n groter is as 18 beteken dit dat die PML Inset/Uitset (tweerigting) penne het. Die aantal toegewysde insetpenne kan dan bepaal word deur n af te trek vanaf 18. Dit kan makliker verduidelik word aan die hand van 'n voorbeeld.

Oorweeg die PAL16L8. Hier is m+n groter as 18 en dus is daar I/U penne teenwoordig. Dit lewer die volgende konstruksie:

- 10 insetpenne: 18-8= 10

- 6 penne is uitsette met terugvoer, of kan ook as insette gebruik word: 16-10=6
- 2 uitsetpenne (sonder terugvoer): 8-6=2

2.5.2 PML KONFIGURASIES

PMLs is beskikbaar in beide bipolêre en CMOS konstruksies. Eersgenoemde maak gebruik van smeltbare verbindings en kan slegs eenmaal geprogrammeer word. Laasgenoemde maak gebruik van geïsoleerde hek MOS en is UV of elektries uitveebaar (Horowitz & Hill, 1989: 502). 'n Ontwerper kan effektief 'n doelspesifieke geïntegreerde kring binne 'n paar uur ontwerp en maak deur gebruikmaking van 'n PML toestel (Uffenbeck, 1987: 311).

Verskeie PMLs is ontwikkel om die totale spektrum van logika funksies te dek. Dit laat die ontwerper toe om 'n PML te kies wat die beste by sy toepassing sal pas.

2.5.2.1 HEK MATRIKSE

'n PML implementeer som-van-produkte logika deur gebruik te maak van 'n programeerbare EN-matriks waarvan die produkterme 'n gebonde OF-matriks voer. Omdat die som-van-produkte vorm haas enige Boole oordragfunksie kan uitdruk, word die PMLs se

gebruike slegs beperk deur die aantal terme beskikbaar in die EN en OF matrikse (Coates, 1988: 668).

PML hek matrikse is beskikbaar in verskeie groottes met beide aktief hoë en aktief lae uitset konfigurasies. Figuur 2.4 illustreer so 'n hek matriks met 'n aktief hoë uitset.

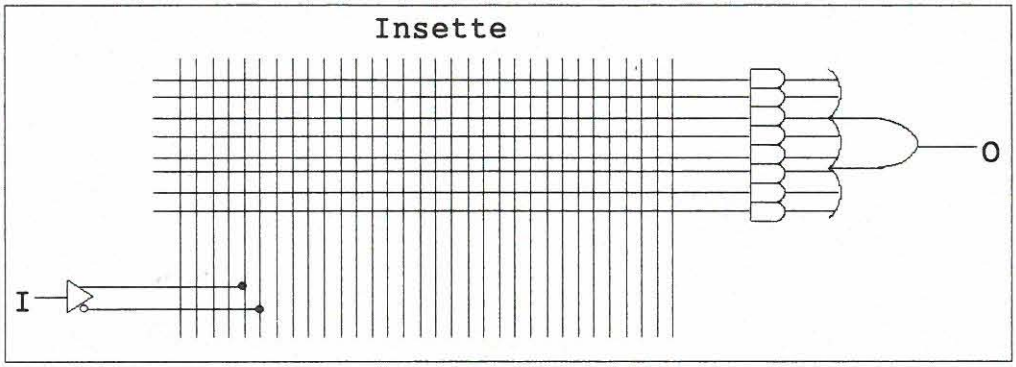


Fig 2.4: Hek matriks

2.5.2.2 PROGRAMMEERBARE I/U

Die maksimum moontlike aantal insette en uitsette by PMLs is nie altyd vas nie. Toestelle, soos die 16L8, is beskikbaar waarvan die uitsette ook as insette gebruik kan word soos bespreek in paragrawe 2.5 en 2.5.1. Figuur 2.5 is 'n voorbeeld hiervan. 'n Driestaat uitsetbuffer bepaal die rigting van datavloei wat die toestelle geskik maak vir

bewerkings soos skuif en rotering van serie data (Levy, 1986: 31).

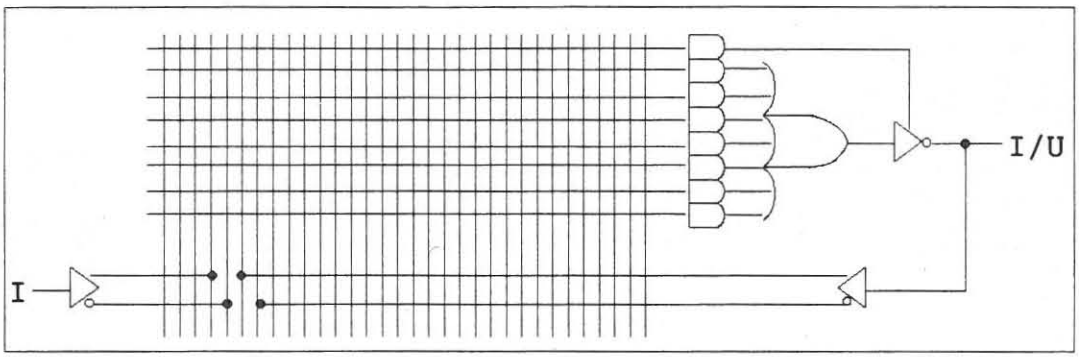


Fig 2.5 Programmeerbare I/U.

2.5.2.3 REGISTER UITSET MET TERUGVOERING

Van die PMLs (soos die 16R8) het register uitsette met terugvoering (figuur 2.6). Elke som-van-produkte uitset word gestoor in 'n D-tipe wipkring tydens die leirand van die klokpuls. Die Q uitset van die wipkring kan dan na die uitsetpen deurgelaat word deur die aktief lae tristaat buffer te aktiveer.

Die nie-Q uitset van die wipkring word teruggevoer na die PML matriks as 'n insetveranderlike. Die terugvoering stel die PML in staat om die vorige staat te "onthou" en sodoende sy funksie te wysig as 'n resultaat van daardie staat. Dit laat die ontwerper toe om 'n PML te programmeer om funksies

soos optelling en aftelling uit te voer (Birkner & Coli, 1983: 1.9).

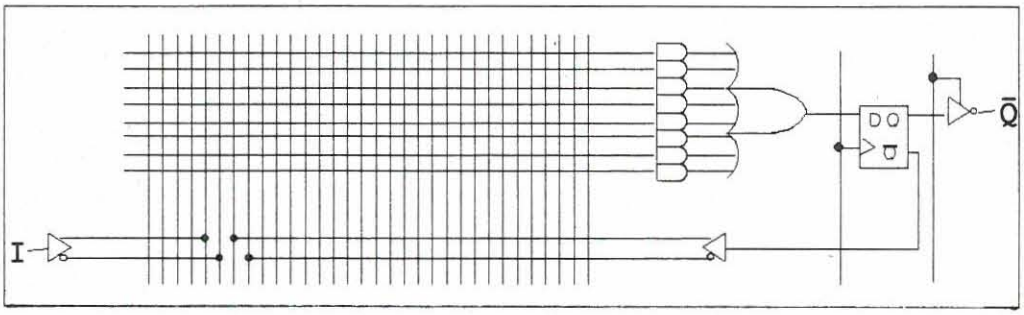


Fig 2.6 Register uitset met terugvoering.

2.5.3 PML TEGNOLOGIE

National Semiconductor PMLs word vervaardig deur van dieselfde hoë-volume tegnologie gebruik te maak as wat gebruik word tydens die vervaardiging van PLAGs. Die insette bestaan uit PNP transistors. Al die uitsette is standaard TTL drywers met interne aktiewe-optrektransisters (Progammable Logic Databook by National Semiconductor, 1983: 24.11).

Emitter-gekoppelde-Logika (EKL) het gelei tot belangrike deurbrake in die vermindering van voortplantingsvertraging. EKL bied meer spoed en verwante voordele as enige ander tipe logikakring. National Semiconductor bied 'n verskeidenheid van EKL PMLs aan (Burton, 1990: 95).

'n Alternatief tot PMLs is Harde matriks logika (HML). HML is soortgelyk aan PMLs, maar verskil in die opsig dat dit masker programmeerbaar is. Die HML is 'n koste-effektiewe oplossing indien groot getalle van dieselfde ontwerp verlang word. Dit is uniek in die sin dat dit 'n hekmatriks is met 'n programmeerbare prototipe (Birkner & Coli, 1983: 1.11). Dit wil sê dat 'n PML geprogrammeer word en indien dit korrek fuksioneer word dit as 'n prototipe gebruik om 'n klomp HMLs te vervaardig.

2.5.4 KEUSE VAN PML

Voordat 'n PML geselekteer word vir gebruik in 'n kring, is die ontwerper verplig om die volgende twee waarnemings omtrent die logikakring te maak:

- Die aantal insette en uitsette na en van die kring moet minder of gelyk wees aan die insette en uitsette van die PML.
- Die aantal produkterme en die verpreiding daarvan na die OF-hekke moet in aanmerking geneem word (Jay, 1986: 67).

Die finale keuse t.o.v. watter toestel om te gebruik sal van spoed, drywingsdissipasie en koste afhang.

2.5.5 PROGRAMMERING VAN PMLs

Die meeste moderne programmeerders word gebruik in samewerking met programmeringsagteware soos PALASM, ABEL en CUPL. Hierdie sagteware stel 'n sekeringskaart in JEDEC formaat saam wat dan d.m.v. 'n RS232 koppeling na 'n PML programmeerder oorgedra word (Jordaan, 1988: 15).

Die eenvoudigste sagteware laat die gebruiker toe om die sekerings wat gesmelt moet word te kies. Die gebruiker moet dus bepaal watter sekerings gesmelt moet word om die verlangde logika werking te verkry. Beter sagteware pakkette laat toe dat 'n Boole uitdrukking gespesifiseer word. Die uitdrukking word dan geminimaliseer en omskep in 'n sekeringskaart wat gebruik word om die PML te programmeer (Horowitz & Hill, 1989: 504).

Die volgende prosedure word gevolg tydens die programmering van 'n PML:

- 'n Logikafunksie word in die vorm van som-van-produkte geskryf.
- 'n PML word geselekteer.
- 'n Sekeringskaart word saamgestel en grafies vertoon (hierdie stap word soms uitgelaat).
- Die sekeringskaart word omgeskakel na 'n JEDEC formaat.

• JEDEC lêer word na die programmeerder oorgedra d.m.v 'n RS232 koppeling en die PML word geprogrammeer.

2.5.5.1 JEDEC FORMAAT

Die formaat wat gebruik word om sekeringskaart-inligting oor te dra na die programmeerder is die JEDEC Standaard No.3-A (Standaard data oordrag tussen die data voorbereidingsisteem en 'n programmeerbare logika programmeerder). 'n Tipiese JEDEC lêer word in Figuur 2.7 uitgebeeld.

```
•Ontwerp deur: G.J. Booysen
Datum ontwerp: 22-02-1993
Part Nommer: PAL16r8
*
QP20*
QF2048*
G0*
F0*
L0512 01111111111111111111111111111111*
L0554 11111111110111011111111111111111*
L0576 11111111110111011111111111111111*
L0768 01111111111111111111111111111111*
L0800 11111111111110111111111111111111*
C12DE*
V0001 XXXXXXXXXN1LLLLZZLLN*
V0002 C1XXXXXXXN0LLLLLLLLN*
V0003 C0XXXXXXXN0LLLLHLLLN*
♥0000
```

Fig. 2.7: JEDEC lêer

'n Jedec lêer bestaan uit die volgende:

- Begin-van-teks karakter (ASCII 02 hex).

- 'n Aantal velde wat met een of twee karakter-identifiseerders begin en eindig met 'n asterisk (*).
- Einde-van-teks karakter (ASCII 03).
- 'n Viersyfer heksadesimale transmissie kontrolegetal.

Tabel 2.1 is 'n samevatting van die JEDEC veld-identifiseerders.

Tabel: 2.1 JEDEC veldidentifiseerders

Identifiseerder	Beskrywing
<Geen>	Ontwerpspesifikasie
QP	Die aantal penne van toestel
QV	Die totale aantal toetsvektore
QF	Die totale aantal sekerings
G	Sekuriteitsekering
F	Aanvanklike sekeringswaarde
L	Sekeringslys
C	Sekering kontrolegetal
V	Toetsvektore

Die eerste veld na die begin-van-teks karakter is die ontwerpspesifikasies. Dit bevat informasie soos die ontwerper se naam, die datum, ens. Die veld eindig met 'n asterisk.

Die G veld spesifiseer of die sekuriteitsekerings van die toestel gesmelt moet word of nie.

Die F, L en C velde spesifiseer die sekeringskaart informasie as volg:

- Die F veld spesifiseer die sekeringswaarde d.w.s 'n een of 'n nul, wat gebruik word vir die sekeringsposisies wat nie gespesifiseer word in die sekeringslys nie (L veld).
- Een of meer L velde spesifiseer die sekeringskaart data. Die desimale getal na die L karakter is die aanvanklike sekeringswaarde vir daardie veld. Die desimale getal word gevolg deur 'n spasie en 'n lys 0 en 1 waardes. 'n Sekeringswaarde van 0 dui aan dat die sekering ongeskonde bly terwyl 'n 1 aandui dat die sekering geblaas moet word.
- Die C veld spesifiseer 'n viersyfer heksadesimale sekering kontrolegetal. Die kontrolegetal word bereken slegs vanaf die sekeringlys data in die L veld. Hierdie kontrolegetal verskil van die globale transmissiekontrolegetal wat na die einde-van-teks karakter verskyn.

Die volgende V veld is opsioneel en spesifiseer die toetsvektor data van die ontwerp. Die desimale getal na die V karakter is die toetsvektornommer. Die toetsvektore sal aangewend word volgens hul nommers en nie volgens die volgorde waarin hul in

die JEDEC lêer verskyn nie. Die vektornommer word gevolg deur 'n spasie en 'n lys toetsvektore.

2.6 OPSOMMING

Die gebruik van PLTe verminder die totale aantal komponente in 'n ontwerp en gee dus aanleiding tot kleiner gedrukte stroombane en verbeterde betroubaarheid (Meyer, 1988a: 59). Dus verhoog die vertroue in die sisteem terwyl die koste daarvan verlaag.

Elkeen van die subfamilies bv: LAGs, PLMs en PMLs het spesifieke voordele. LAGs is bruikbaar waar die spesifikasie 'n woordstruktuur formaat het of waar 'n groot aantal produkterme verlang word. PLMs is die mees buigsame tipe omdat alle produkte gedeel kan word. Die PML is egter dikwels die beste keuse van die drie omdat die PML meer ekonomies is om te gebruik met tipies die kortste voortplantingsvertraging (Bolton, 1990: 150).

HOOFSTUK 3

SIMULASIE VAN ELEKTRONIESE KRINGE

3.1 INLEIDING

Simulasie is 'n alledaagse verskynsel wat op verskeie terreine toegepas word. Deur simulasie word gepoog om die werklike situasie na te boots teen 'n relatief lae koste. Die resultaat van die simulasieproses sal nie noodwendig presies ooreenstem met wat in die praktyk ondervind word nie, maar dit gee 'n goeie aanduiding van wat verwag kan word. Simulasie word gedefinieer as 'n kwantitatiewe prosedure waardeur 'n werklike proses of situasie omskryf word deur 'n model van daardie proses of situasie op te stel en die werking van die proses oor die verloop van 'n bepaalde tydsduur na te boots ten einde die gedrag van die werklike proses te voorspel (Redelinghuis, et al, 1985: 389).

3.2 DIE AARD VAN SIMULASIE

Ten einde die verwagte werking van 'n kring of proses (algemeen na verwys as 'n stelsel of sisteem) te voorspel moet 'n aantal aannames gemaak word van hoe dit funksioneer. Hierdie aannames, wat gewoonlik die vorm van wiskundige of logiese verwantskappe aanneem, vorm 'n

model wat gebruik word om begrip vir die werking van so 'n sisteem te verkry. Indien die verwantskappe wat so 'n model uitmaak relatief eenvoudig is, is dit moontlik om van wiskundige metodes gebruik te maak om akkurate inligting te bekom op vrae van belang. Dit word die analitiese metode genoem.

Daar is egter sisteme wat te kompleks is om deur middel van realistiese modelle analities geëvalueer te word en hierdie modelle moet deur middel van simulasie ondersoek word. By simulasie word 'n rekenaar gebruik om 'n model numeries te evalueer en data word versamel om 'n skatting te maak van die korrekte karakteristieke wat verlang word van die model.

Simulasie van sisteme is 'n algemene gebruik in navorsing, ingenieurswese en die bestuurswetenskappe. 'n Nog wyer aanvaarding en benutting van simulasie word egter deur 'n aantal faktore verhoed. Modelle wat gebruik word om groot sisteme te ondersoek, is soms baie kompleks en om rekenaarprogramme te skryf om dit te simuleer is baie veeleisend. Die ontwikkeling van uitstekende sagtewareprodukte wat voorsien in baie van die eienskappe wat benodig word vir die kodering van so 'n simulasiemodel het egter hierdie taak in die onlangse verlede aansienlik vergemaklik.

Simulasie van komplekse sisteme neem baie rekenaartyd in beslag. Hierdie probleem word egter minder ernstig namate

die koste van rekenaars en rekenaartyd afneem. Laastens bestaan daar ongelukkig die indruk dat simulasie slegs 'n oefening - alhoewel 'n gekompliseerde een - in rekenaarprogrammering is (Law & Kelton, 1991: 1).

Simulasie van komplekse kringe vind baie stadiger plaas as die spoed waarteen die werklike kringe funksioneer. 'n Streng funksionele simulasie van 'n elektroniese kring, waar daar aangeneem word dat al die komponente zero vertraging het, is relatief vinnig, maar 'n baie meer realistiese simulasie, waar al die slegste-geval strek van vertragings vir alle seinpaaie bereken en oorweeg word, is baie stadiger (Wakerly, 1990: 662).

3.3 SIMULANTDATABASIS

Enige simulant maak gebruik van 'n databasis om sekere inligting oor alle komponente waarvan die werking gesimuleer kan word te stoor. By geïntegreerde kringe kan so 'n databasis byvoorbeeld 'n komponentmodel bevat wat die toestelle se logiese en elektriese werking beskryf. Die minimum wat so 'n komponentmodel van 'n geïntegreerde kring moet bevat is die identifisering van elke pen as uitset of inset. Die komponentdatabasis kan ook die slegste-geval vertragingswaarde vir elke inset-na-uitset, asook die opstel- en houtye vir toestelle wat van 'n klokpuls gebruik maak, voorsien.

Die data beskikbaar in die databasis word deur die simulant aangewend om die algehele werking van die kring vir 'n gegewe reeks insette te voorspel. Die ontwerper verskaf 'n reeks insette wat op die kring toegepas kan word en die simulantprogram bepaal hoe die kring sal funksioneer op sodanige insette. Op hierdie manier is dit moontlik om 'n hele kring te ontfout sonder enige kraaineskonstruksie, en om 'n gedrukte kring te monteer wat die eerste keer reg behoort te werk (Wakerly, 1990: 661).

3.4 SIMULASIE VAN ANALOOGKRINGE

'n Aantreklike manier om 'n analoogkring te analiseer is d.m.v. rekenaarsimulasie. 'n Rekenaar is in staat om van gedetailleerde modelle van verskeie komponente gebruik te maak tydens kringanalise. Dit kan fisiese faktore soos temperatuurafhanklikheid en komponent toleransies infaseer tydens 'n simulasieproses. Met die toenemende gebruik van persoonlike rekenaars en verwante sagteware het simulasie 'n populêre tegniek geword op baie gebiede van kring analisering en ontwerp (Savant, Roden & Carpenter, 1991: 27).

Kringsimulante gebruik elektriese elemente as basiese elemente. Die mees gewilde analoog kringsimulant is die Berkeley SPICE program wat haas enige diskrete komponent as basiese element gebruik. SPICE is 'n groot en kragtige

meerdoelige kringsimulasieprogram vir gelykstroom, wisselstroomtransiënte en Fourier-analises. Kringe kan bestaan uit weerstande, kapasitors, induktors, diodes, afhanklike en onafhanklike spanning- en stroombronne, sowel as ander halfgeleiertoestelle (Scott, 1987: 703).

Ander rekenaar simulasieprogramme soos PSPICE en MICRO-CAP III is beskikbaar vir simulasie van analoogkringe. 'n Groot verskil tussen die verskillende simulasieprogramme is die manier waarop die kringe geteken word. SPICE vereis dat die gebruiker die komponente spesifiseer in terme van die nodes waaraan hulle verbind is terwyl MICRO-CAP III die gebruiker toelaat om die kring direk op die rekenaarskerm te teken (Savant, et al, 1991: 27).

3.5 SIMULASIE VAN DIGITALE TOESTELLE

Tydens die ontwerp van digitale sisteme is die koste en tydsvertraging betrokke in die produksie van hardeware te hoog om ontwerp d.m.v. gissing en vergissing toe te laat. Tog moet elke poging aangewend word om ontwerpfoute op te spoor en te korrigeer. Rekenaarsimulasies is baie bruikbaar vir hierdie doel (Unger, 1989: 255).

Om 'n uitset te verskaf benodig 'n logikasimulant twee insette. Die eerste inset verskaf 'n logikabeskrywing van die kring onder evaluering, terwyl die tweede 'n

toetspatroon van logika ene en nulle bevat wat op die insette van die kring verlang word. Die simulant plaas die ene en nulle in die sagtewaremodel en verskaf 'n verwagte uitset in die vorm van logikavlakke (Meyer, 1988b: 65).

By logika analise is dikwels slegs die logiese funksie van die toestelle van belang en daar word aangeneem dat al die hekke dieselfde vertragingstyd het. Omdat die tydsvertraging hier nie in berekening gebring word nie is hierdie tipe simulasie net bruikbaar om die logiese korrektheid van 'n ontwerp te verifieer. Indien die logies korrekte ontwerp verkry is, kan 'n tydanalise uitgevoer word. Hierdie sagteware program bereken die node-tot-node vertraging op 'n soortgelyke wyse as wat die ontwerper dit met die hand kan bereken (Schilling, et al, 1989: 866).

3.5.1 SIMULANT TIPES

Die doel van simulasiesagteware is om die korrektheid - tot 'n meerdere of mindere mate en op verskillende vlakke van logikaverteenwoordiging - van 'n ontwerp te bevestig. 'n Aantal verskillende tipes simulante is oor die jare ontwikkel. Hierdie simulante verskil van mekaar t.o.v. hul doel en die basiese elemente wat hulle gebruik om die logika te verteenwoordig. Die simulante kan in die volgende tipes ingedeel word:

- Hekvlaksimulante
- Funksionele en gemengde-modus simulante

3.5.1.1 HEKVLAKSIMULANTE

Daar word van twee benaderings by die hekvlaksimulantsagteware gebruik gemaak. Enersyds word twee aparte programme gebruik om die logika-analise en tydanalise uit te voer. Alternatiewelik word 'n gekombineerde program gebruik om albei analises uit te voer.

Parameters soos logikafunksies, insetkapasitansie, voortplantingsvertragings, ens. is belangrike parameters vir die hekvlaksimulant en word in 'n databasis gestoor. Hierdie parameters word gebruik as insetdata vir die simulant.

Insette na die simulant, soos voorgestel in figuur 3.1, is die vergelyking wat deur die logikasagteware gegenereer word, die insetstimulus en die data in die databasis. Die simulant voer logika- sowel as tydanalise uit deur van die insette gebruik te maak.

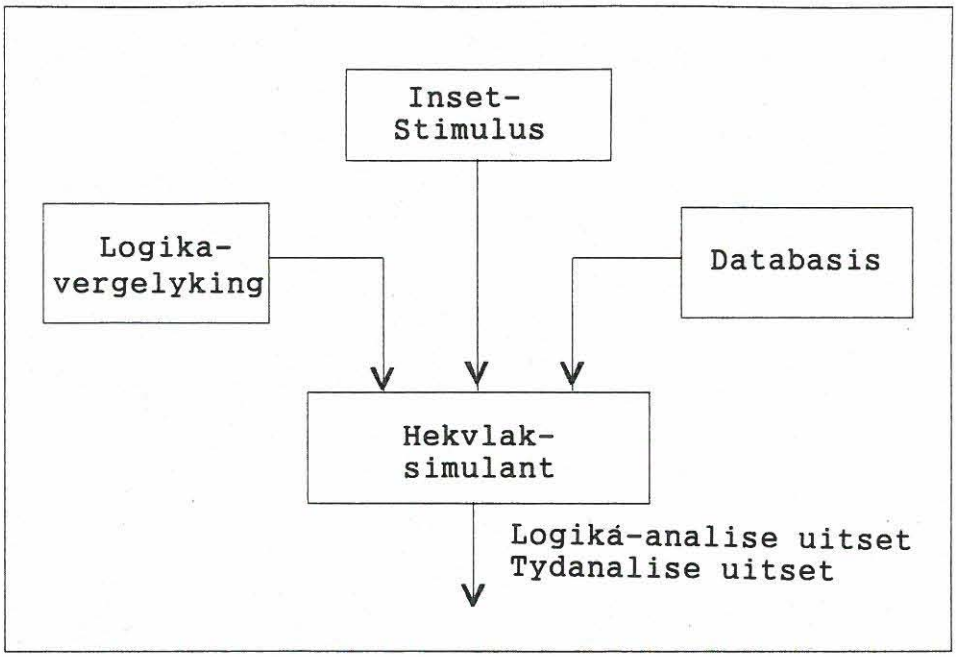


Fig. 3.1: Blokdiagram van 'n hekvlak simulant

3.5.1.2 FUNKSIONELE EN GEMENGDE-MODUS SIMULANTE

Groot ontwerpe maak gewoonlik van gemeenskaplike funksionele blokke soos ewe toeganklike geheues (ETG), lees alleen geheues (LAGs), rekenkundige en logiese eenhede (RLE), ens. gebruik. Dus kan die simulasietyd van die sentrale verwerkeenheid verminder word aangesien 'n blokbou benadering gevolg word. Sodra 'n groot blok ontwerp, en op kring- of hekvlak gesimuleer is, kan 'n funksionele model geskep word wat in die plek van 'n blok geplaas word vir totale kringsimulasie. Funksionele-simulante is simulante wat groot

blokdefinisies kan akkommodeer en geïntegreerde werking van 'n kring kan simuleer in terme van groot blok inset- en uitsetseine (dus 'n hiërargiese benadering).

Simulante wat 'n mengsel van hierdie groot blokke en elementêre hekke kan akkommodeer word gemengde-modus simulante genoem.

3.5.2 KOMPONENTMODELLE

Simulante maak van verskillende komponentmodelle, wat in die databasis gestoor word, gebruik. Funksies word deur die simulant opgestel en opgelos deur gebruik te maak van hierdie modelle en die kringstruktuur. Digitale simulante maak gewoonlik gebruik van een van die volgende modelle vir elke logika element:

3.5.2.1 GEDRAGSMODEL

'n EN hek lewer byvoorbeeld 'n "1", na die nodige vertraging, as uitset indien al die insette verander het na 'n "1". Hierdie tipe beginsel word deur die gedragsmodel uitgevoer en word tydens simulasie gebruik.

3.5.2.2 STRUKTUURMODEL

Groter elemente kan gemodelleer word as 'n versameling van kleiner elemente met gedefinieerde gedragsmodelle. 'n 74LS138 3-tot-8 dekodeerder kan byvoorbeeld gemodelleer word as 'n reeks hekke wat volgens die interne logikakring gekoppel is. Alternatiewelik kan die dekodeerder 'n gedragsmodel hê wat uit 'n waarheidstabel en tydinformasie bestaan.

Bogenoemde modelle word sagtewaremodelle genoem omdat die totale werking van die geïntegreerde kringe deur sagteware gesimuleer word.

3.5.2.3 FISIESE MODEL

By 'n fisiese model word die geïntegreerde kring self aan die rekenaar, wat die simulasie uitvoer, gekoppel. Die rekenaar verskaf insette aan die kring en bepaal die uitsette - wat dan as insette vir ander modelle in die simulasieproses gebruik word (Wakerly, 1990: 661).

3.5.3 SIMULASIE VAN PROGRAMMEERBARE LOGIKA TOESTELLE

'n Goeie logikasimulant kan 'n groot verskil maak in die algehele resultate wat verkry word met programmeerbare logika toestelle (PLTe). 'n Simulant kan ontwerpfoute opspoor voordat 'n toestel geprogrammeer word en sodoende die koste en tyd spaar wat dit sou neem om die toestel te programmeer. Indien die simulant vinnig en maklik is om te gebruik kan dit die ontwerper aanmoedig om met nuwe ontwerpe en tegnieke te eksperimenteer wat tot beter oplossings van ontwerpprobleme kan lei (Burton, 1990: 171).

Wanneer die verlangde funksie van 'n PLT gespesifiseer is, en die funksie is in 'n sekeringkaart saamgestel, is dit wenslik om te verseker dat die toestel sal funksioneer soos verwag indien geprogrammeer. In die afwesigheid van wiskundig-gebaseerde metodes om die werking van die kring te voorspel moet daar op simulasie vertrou word (Bolton, 1990: 323).

Simulasie kan nie effektief met die hand gedoen word nie - veral nie t.o.v volgorde logika ontwerpe nie. Aangesien dit moontlik is om 'n simulasie te voltooi wat nie werklik die kring se werking aanspreek nie, is dit noodsaaklik om nie net 'n baie goeie idee te hê van wat in die kring moet gebeur nie, maar 'n uitstekende idee van wat nie moet gebeur nie (Barwise, 1987: 25).

'n PLT toestel hoef nie noodwendig gesimuleer te word voor programmering nie. Die kanse is egter goed dat so 'n toestel nie soos beoog sal funksioneer nie. Vanaf die simulasieresultate kan foute, byvoorbeeld 'n foutiewe pendefinisie, maklik opgespoor word. Meer komplekse foute, soos onverlangde logika werking van 'n toestel, kan ook deur inspeksie van die resultate opgespoor word. Die simulasie verhoog nie alleen die vertrouensvlak rondom 'n nuwe ontwerp nie, maar dit gee ook vertroue in die programmeringsproses (Frost, 1989a: 502).

ABEL is 'n sagtewarepakket wat gebruik maak van 'n hardeware beskrywingstaal (hardware description language) vir die ontwerp en simulering van logika sisteme t.o.v PLTe. 'n Hardeware beskrywingstaal is 'n beskrywende nutsprogram wat van sagteware gebruik maak om die werking van die hardeware van 'n ontwerp te beskryf. ABEL laat twee tipes simulasies toe, naamlik die simulasie van die funksies - waar ontfouting kan geskied nog voordat 'n toestel gekies is - en simulasie van die JEDEC lêer wat die ontwerp simuleer met die gekose toestel. Simulasie-uitsette kan vertoon word in tabelvorm of as golfvorms (Anderson, 1991: 927).

3.5.3.1 SIMULASIE SOOS AANGESPREEK IN PROJEK

'n PML-sagtewarepakket soos PALASM2 maak gebruik van 'n teksredigeerder waarin die PML ontwerp beskryf word. Na die ontwerpbeskrywing kan die simulasieproses in dieselfde teksredigeerder beskryf word. Hierna word die beskrywing gekompileer na 'n JEDEC lêer (Greiner & Schmitz, 1984: 248). Ander sagteware pakkette soos ABEL, CUPEL en AMAZE maak ook gebruik van teksredigeerders en werk op 'n soortgelyke wyse as PALASM2.

Die sagteware wat tydens die projek ontwikkel is verskil radikaal van bogenoemde pakkette. Daar word van geen teksredigeerder gebruik gemaak om 'n JEDEC lêer te kompileer of om 'n simulasie uit te voer nie. Die sagteware wat die JEDEC lêer skep en die simulantsagteware staan heeltemal onafhanklik van mekaar.

Die ontwikkelde simulant maak gebruik van drie insette om 'n uitset te lewer. Die volgende is 'n kort beskrywing van die werking van die simulant.

Die eerste inset is die JEDEC lêer wat in die rekenaar se geheue ingelees word. Daar is tydens die projek sagteware ontwikkel wat so 'n JEDEC lêer kan saamstel vanaf die betrokke Boole uitdrukkings

wat in 'n PML gerealiseer moet word.

Die volgende inset na die simulant is die PML spesifikasies wat vanaf die PML-databasis verkry word. Sagteware is ook ontwikkel om 'n PML-databasis daar te stel.

Die laaste inset na die simulant is die simulasievektore wat deur die gebruiker self ingevoer word in die vorm van logika ene en nulle.

Hierdie insette na die simulant word dan deur die simulant verwerk en die resultaat word in die vorm van 'n waarheidstabel verskaf.

3.5.3.2 TOETSING

Die toetsing en simulasie van PLTe moet nie verwar word nie. Simulasie van 'n PLT moet plaasvind voordat die PLT geprogrammeer word, terwyl toetsing na programmering plaasvind.

'n Probleem bestaan om te bevestig dat daar geen foute as gevolg van kringdefekte is na programmering van 'n PLT nie. Die simulasievektore is dikwels onvoldoende vir hierdie taak en generering van die verlangde toetsvektore moet verkieslik deur 'n toetsvektorgenereringsprogram

gedoen word. 'n Hele aantal toetsvektorgenerator algoritmes is al bekend gestel waarvan die belangrikste die D-Algoritme, Podem (Path-Oriented Decision Making) en FAN is (Smit, 1992: 39).

Die kringdefekte waarna in die vorige paragraaf verwys word is defekte wat tydens die vervaardigingsproses van die PLT ontstaan het. 'n Sekering kan byvoorbeeld ontbreek waar daar wel een moes wees.

Na programmering van 'n PLT moet 'n reeks toetsvektore - wat bestaan uit 'n reeks insette en verwagte uitsette - gegenereer word. Toetsvektore kan gegenereer word deur die verskaffing van die insetstimulis na die simulant en die verkryging van die ooreenstemmende uitsetgedrag. Tipiese toetsvektore bevat so veel as 16000 stimuli en response om bevredigende toetsing vir komplekse ontwerpe te verseker.

Daar word van twee benaderings tot sagteware gebruik gemaak om 'n komplekse ontwerp te toets naamlik; node-skakeling analise en vasgeklem-by (stuck-at) analise (Schilling, et al, 1989: 874).

gedoen word. 'n Hele aantal toetsvektorgenerator algoritmes is al bekend gestel waarvan die belangrikste die D-Algoritme, Podem (Path-Oriented Decision Making) en FAN is (Smit, 1992: 39).

Die kringdefekte waarna in die vorige paragraaf verwys word is defekte wat tydens die vervaardigingsproses van die PLT ontstaan het. 'n Sekering kan byvoorbeeld ontbreek waar daar wel een moes wees.

Na programmering van 'n PLT moet 'n reeks toetsvektore - wat bestaan uit 'n reeks insette en verwagte uitsette - gegenereer word. Toetsvektore kan gegenereer word deur die verskaffing van die insetstimulis na die simulant en die verkryging van die ooreenstemmende uitsetgedrag. Tipiese toetsvektore bevat so veel as 16000 stimuli en response om bevredigende toetsing vir komplekse ontwerpe te verseker.

Daar word van twee benaderings tot sagteware gebruik gemaak om 'n komplekse ontwerp te toets naamlik; node-skakeling analise en vasgeklem-by (stuck-at) analise (Schilling, et al, 1989: 874).

3.6 OPSOMMING

In sekere gevalle is dit nie moontlik om 'n stelsel wiskundig te analiseer nie. Derhalwe word van simulasie gebruik gemaak om die verwagte werking van die stelsel te bepaal. 'n Voorbeeld hiervan is die simulering van programmeerbare logika toestelle waarvoor daar geen wiskundig-gebaseerde metodes vir analisering bestaan nie.

'n Probleem t.o.v. die gebruikmaking van programmeerbare logika toestelle is die duur toerusting wat benodig word. Dit kan moeilik geregverdig word vir 'n klein ontwikkelingslaboratorium of opvoedkundige instellings (Coates, 1988: 668).

Die sagteware wat tydens die projek ontwikkel is bied 'n oplossing tot bogenoemde probleem. Enige persoon met toegang tot 'n persoonlike rekenaar kan m.b.v. hierdie sagteware 'n JEDEC lêer saamstel en die ontwerp simuleer. Die JEDEC lêer word ook gebruik om die PML te programmeer. Indien 'n programmeerder nie beskikbaar is nie, kan die JEDEC lêer na die naaste PML handelaar wat oor 'n programmeerder beskik geneem word om die PML te programmeer.

Bogenoemde moet nie gesien word as 'n omvattende beskrywing van simulasie van programmeerbare logika toestelle nie. Hierdie beginsels is egter tydens die projek aangespreek t.o.v. programmeerbare matriks logika toestelle.

HOOFSTUK 4

SAGTEWARE

Al die sagteware is geskryf deur gebruik te maak van Turbo Pascal 6.0. Die sagteware wat tydens die projek ontwikkel is bestaan hoofsaaklik uit vier programme nl:

- Hoofprogram
- PML-databasisprogram
- Sekeringkaartprogram
- Simulasieprogram

4.1 HOOFPROGRAM

Hierdie program voorsien 'n menu vanwaar 'n keuse gemaak kan word tussen drie opsies. Die eerste opsie is om 'n PML se data by die PML-databasis te voeg. Met die tweede opsie kan 'n sekeringkaart saamgestel en in 'n JEDEC-lêer gekompileer word. Laastens is daar 'n keuse om die werking van 'n PML te simuleer vanaf inligting in 'n JEDEC lêer. 'n Uitdruk van die program kan in Bylaag A gesien word terwyl Figuur 4.1 die basiese uitleg van hierdie program toon.

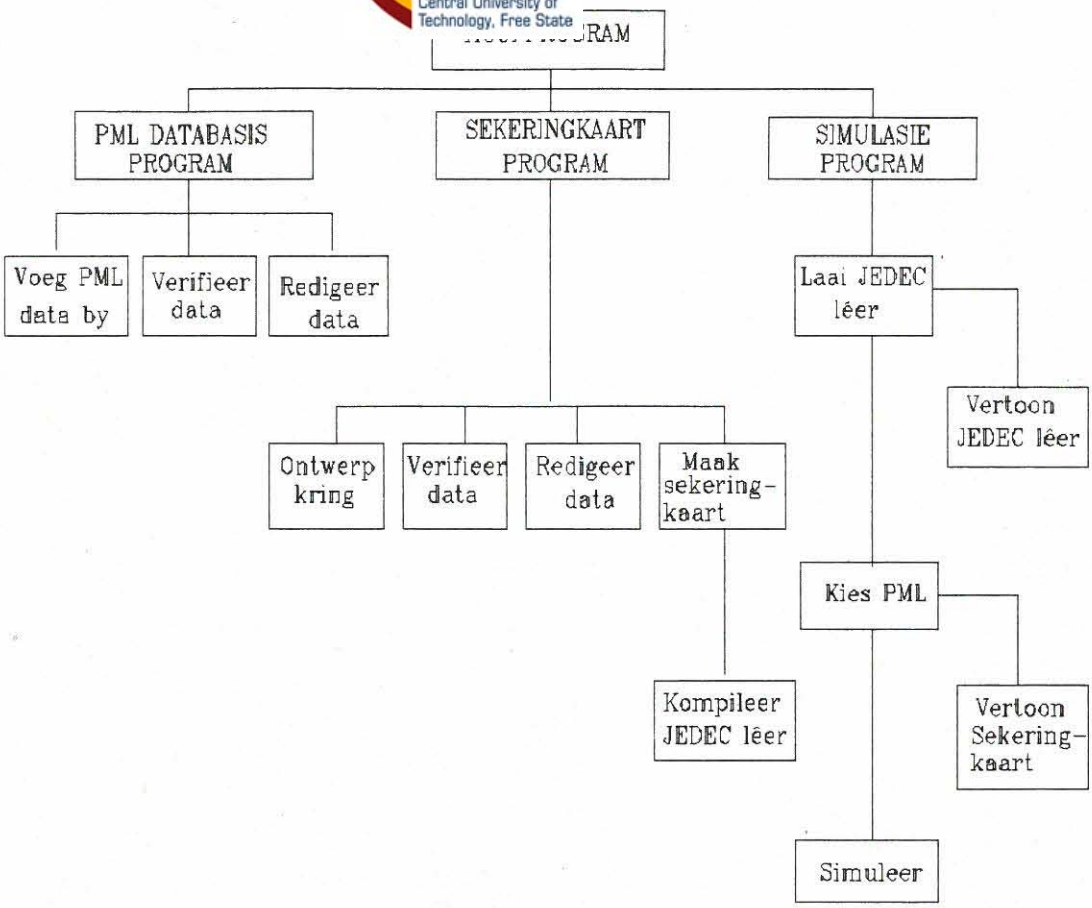


Fig. 4.1: Struktuurkaart van hoofprogram

4.2 PML-DATABASISPROGRAM

Die PML-databasisprogram word opgedeel in 'n aantal subprogramme en roetines. Bylaag B is 'n uitdruk van hierdie program. Die subprogramme is as volg:

- Databasissubprogram
- Verifikasie-subprogram
- Redigeringsubprogram

4.2.1 DATABASISSUBPROGRAM

Die doel van hierdie program is om 'n databasis daar te stel vir al die tipes PMLs wat deur die gebruiker van die sagteware gebruik kan word. Tydens simulasie en die ontwikkeling van die sekeringkaart, word toegang tot die PML-databasis verkry en kan een van die PMLs wat in die databasis voorkom geselekteer word. Op hierdie wyse was dit moontlik om 'n program vir die ontwikkeling van 'n sekeringkaart en simulasie te ontwikkel wat onafhanklik is van die tipe PML wat gebruik word.

Deur van hierdie konsep gebruik te maak word die moontlikhede van die simulant vergroot, omdat die simulant aanpasbaar is by verskeie tipe PMLs. Die konsep ondersteun ook die redigering van bestaande data in die databasis.

Die PML-databasis betaan uit 'n aantal rekords. 'n rekord bevat al die relevante data van 'n enkele PML. Die data word in dieselfde volgorde in al die rekords gestoor. Tydens die ontwikkeling van die sekeringkaart, sowel as tydens simulasie, moet die operateur aandui watter PML gebruik gaan word. Die spesifieke PML se data word dan uit die rekord in die databasis na die rekenaar se geheue oorgedra.

Omdat die data in al die rekords in dieselfde opeenvolgende wyse gestoor word, word 'n eenvormige metode verlang om die data vanaf die rekords te lees. Die metode het die voordeel dat die gedeelte van die program wat die inlees van die PML data hanteer baie vereenvoudig kon word.

Die data wat verskaf moet word tydens die opstel van die PML-databasis word vanaf geskikte databoeke verkry. Die PML-databasisprogram toon aan die gebruiker watter inligting omtrent die PML op elke stadium verlang word.

Die PML-databasis word as volg opgestel:

1. Verkry naam van PML.
2. Aantal penne
3. Vir i=0 tot aantal penne.
   3.1 Pen status (inset, uitset, klok, ens.)
   3.2 Insetpen?
      3.2.1. Ja - Aan water insetlyn verbind?
      3.2.2. Anders - Gaan na 3.3.
   3.3. Uitsetpen?
      3.3.1 Ja - Gaan na 3.3.3.
      3.3.2 Anders - Gaan na 3.1.
      3.3.3. PML met registers?
         3.3.3.1. Bevat betrokke uitset 'n register?
      3.3.4. Magtigingslyn?
      3.3.5. Identifiseer eerste produklyn.

3.3.6. Aantal produklyne.

3.3.7. Uitset met terugvoering?

3.3.7.1. Ja - Aan watter insetlyn verbind?

3.3.7.1.1 Gaan na 3.1.

3.3.7.2. Anders - Gaan na 3.1.

4. Einde van subprogram.

In gevalle waar die uitset 'n magtigingslyn het moet hierdie produklyn nie as die eerste produklyn van die uitset gesien word nie. Alhoewel die toevoeging van data tot die PML-databasis na 'n omslagtige en gekompliseerde proses lyk, moet daar in gedagte gehou word dat die byvoeging van 'n PML tot die databasis 'n eenmalige proses is. Nadat die PML by die databasis gevoeg is, kan die sekeringkaartprogram toegang tot die data verkry tydens die prosessering van 'n sekeringkaart.

4.2.2 VERIFIKASIE-SUBPROGRAM

Dit is baie belangrik dat die data in die PML-databasis korrek moet wees aangesien hierdie data gebruik word tydens die samestelling van 'n sekeringkaart. Indien hierdie data foutief sou wees is dit onmoontlik om 'n korrekte sekeringkaart te ontwikkel en sal die simulasie resultate ook foutief wees.

Die verifikasie-program is ontwikkel om die korrektheid van die PML-databasis te verifieer. Hierdie program lees die verlangde PML se data en vertoon dit op die skerm of verskaf 'n uitdruk afhangend van wat verlang word.

4.2.3 REDIGERINGSUBPROGRAM

Indien die data van enige PML foutief sou wees in die PML-databasis kan dit met hierdie program reggestel word. Hierdie program werk basies soos die databasis-subprogram wat hierbo bespreek is. Die verskil is egter dat hier gekies kan word om byvoorbeeld net die een pen se data te verander.

Daar moet egter opgelet word dat al die data in die databasis uitgewis kan word.

4.3 SEKERINGKAARTPROGRAM

Die hoofdoel van die program is om vanaf 'n stel Boole uitdrukkings 'n sekeringkaart saam te stel en te kompileer in 'n JEDEC lêer. Tydens die proses word daar toegang tot die PML-databasis, wat in punt 4.2.1. bespreek is, verkry. Die program stel 'n databasis saam van al die kringe wat met hierdie program ontwikkel word. Daar word na hierdie databasis as die kringdatabasis

verwys. Die volgende is 'n pseudokode lys van hierdie program.

1. Lees name van bestaande kringe na rekenaar.
2. Lees name van PMLs vanaf PML-databasis.
3. Vertoon menu
   3.1. Nuwe ontwerp?
      3.1.1. Ja - Gaan na kringdatabasissubprogram (sien paragraaf 4.3.1).
      3.1.2. Anders - Gaan na 3.2.
   3.2. Vertoon bestaande ontwerp?
      3.2.1. Ja - Gaan na kringdatasubprogram (sien paragraaf 4.3.2).
      3.2.2. Anders - gaan na 3.3.
   3.3. Redigeer bestaande ontwerp?
      3.3.1. Ja - Gaan na redigeringsubprogram (sien paragraaf 4.3.5).
      3.3.2. Anders - Gaan na 3.4.
   3.4. Maak sekeringkaart?
      3.4.1. Ja - Gaan na sekeringkaartsubprogram (sien paragraaf 4.3.3).
      3.4.2. Anders - Gaan na 3.5.
   3.5. Vertoon JEDEC lêer?
      3.5.1. Ja - Gaan na JEDEC subprogram (sien paragraaf 4.3.4).
      3.5.2. Anders - Gaan na 3.6.
   3.6. Stop?
      3.6.1. Ja - Einde van program.
      3.6.2. Anders - Gaan na 3.

4.3.1 KRINGDATABASISSUBPROGRAM

Die doel van hierdie subprogram is om 'n nuwe ontwerp tot die databasis te voeg. Hierdie data word deur die sekeringkaartsubprogram gebruik om die sekeringkaart en die JEDEC lêer te kompileer. Figuur 4.2 is 'n voorstelling van hierdie subprogram.

Getalle kan nie as veranderlikes aan die inset- en uitsetpenne, wat gebruik gaan word, toegeken word nie. Indien 'n pen nie gebruik word nie moet 'n spasiekarakter gebruik word om dit aan te dui. Die veranderlikes moet net een karakterlengte wees, behalwe waar die PML uitsette aktief laag is. In so 'n geval moet daar aangedui word dat die uitset aktief laag is, bv. (!Y = NIE Y).

Dit is dus ook duidelik dat die ! karakter nie as 'n veranderlike gebruik kan word nie. Indien die aktiewe lae uitsette nie op die manier aangedui word nie, sal die JEDEC lêer verkeerd gekompileer word.

'n Boole uitdrukking moet vir elke uitset waarvoor 'n veranderlike toegeken is verskaf word. Die formaat van die uitdrukking is as volg: Y = ABC + AC + BC.

In gevalle waar die uitset 'n magtigingslyn het moet 'n Boole uitdrukking (produkterm) ook vir hierdie

4.3.1 KRINGDATABASISSUBPROGRAM

Die doel van hierdie subprogram is om 'n nuwe ontwerp tot die databasis te voeg. Hierdie data word deur die sekeringkaartsubprogram gebruik om die sekeringkaart en die JEDEC lêer te kompileer. Figuur 4.2 is 'n voorstelling van hierdie subprogram.

Getalle kan nie as veranderlikes aan die inset- en uitsetpenne, wat gebruik gaan word, toegeken word nie. Indien 'n pen nie gebruik word nie moet 'n spasiekarakter gebruik word om dit aan te dui. Die veranderlikes moet net een karakterlengte wees, behalwe waar die PML uitsette aktief laag is. In so 'n geval moet daar aangedui word dat die uitset aktief laag is, bv. (!Y = NIE Y).

Dit is dus ook duidelik dat die ! karakter nie as 'n veranderlike gebruik kan word nie. Indien die aktiewe lae uitsette nie op die manier aangedui word nie, sal die JEDEC lêer verkeerd gekompileer word.

'n Boole uitdrukking moet vir elke uitset waarvoor 'n veranderlike toegeken is verskaf word. Die formaat van die uitdrukking is as volg: Y = ABC + AC + BC.

In gevalle waar die uitset 'n magtigingslyn het moet 'n Boole uitdrukking (produkterm) ook vir hierdie

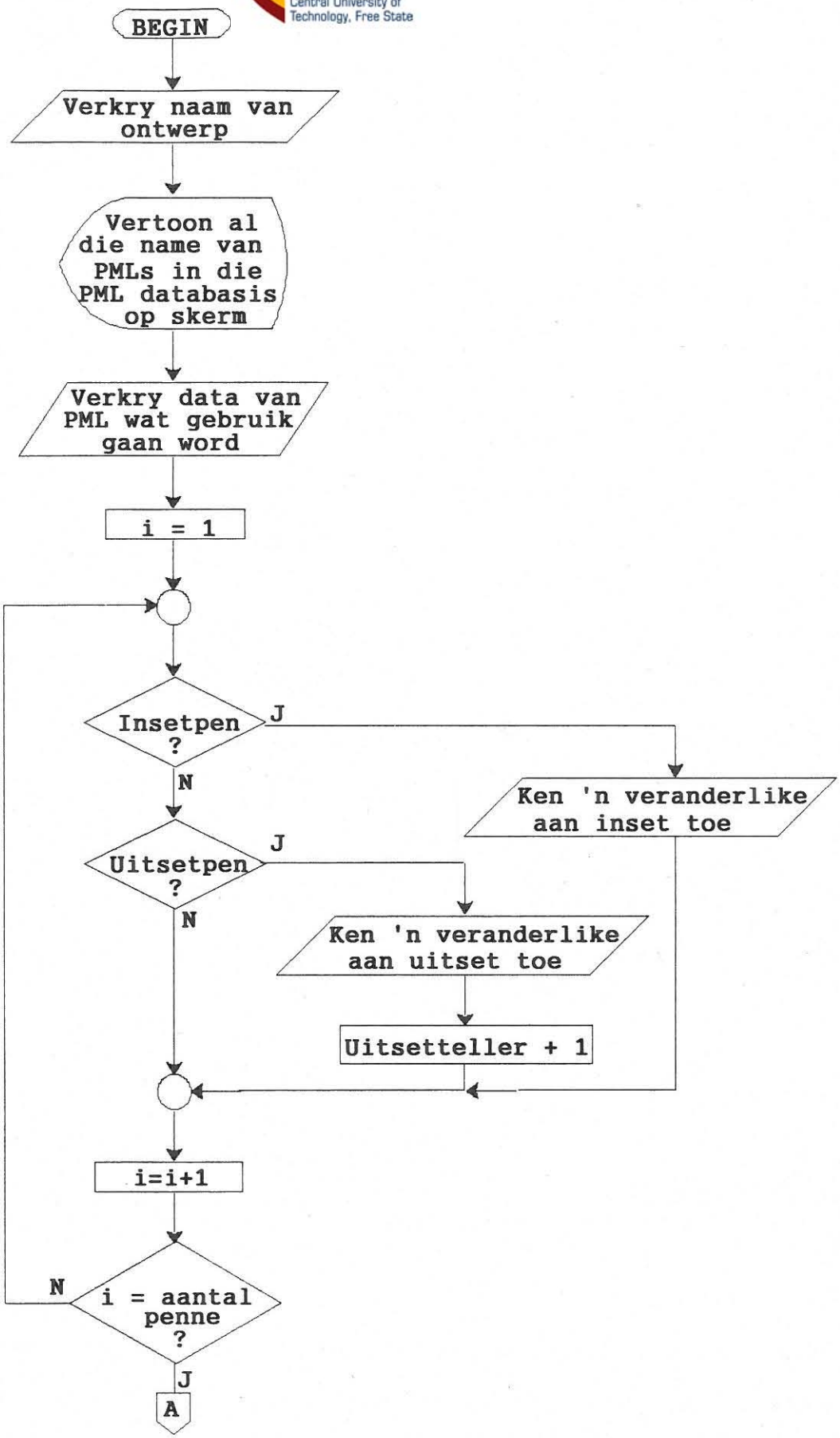


Fig. 4.2: Vloeikaart van kringdatabasissubprogram.

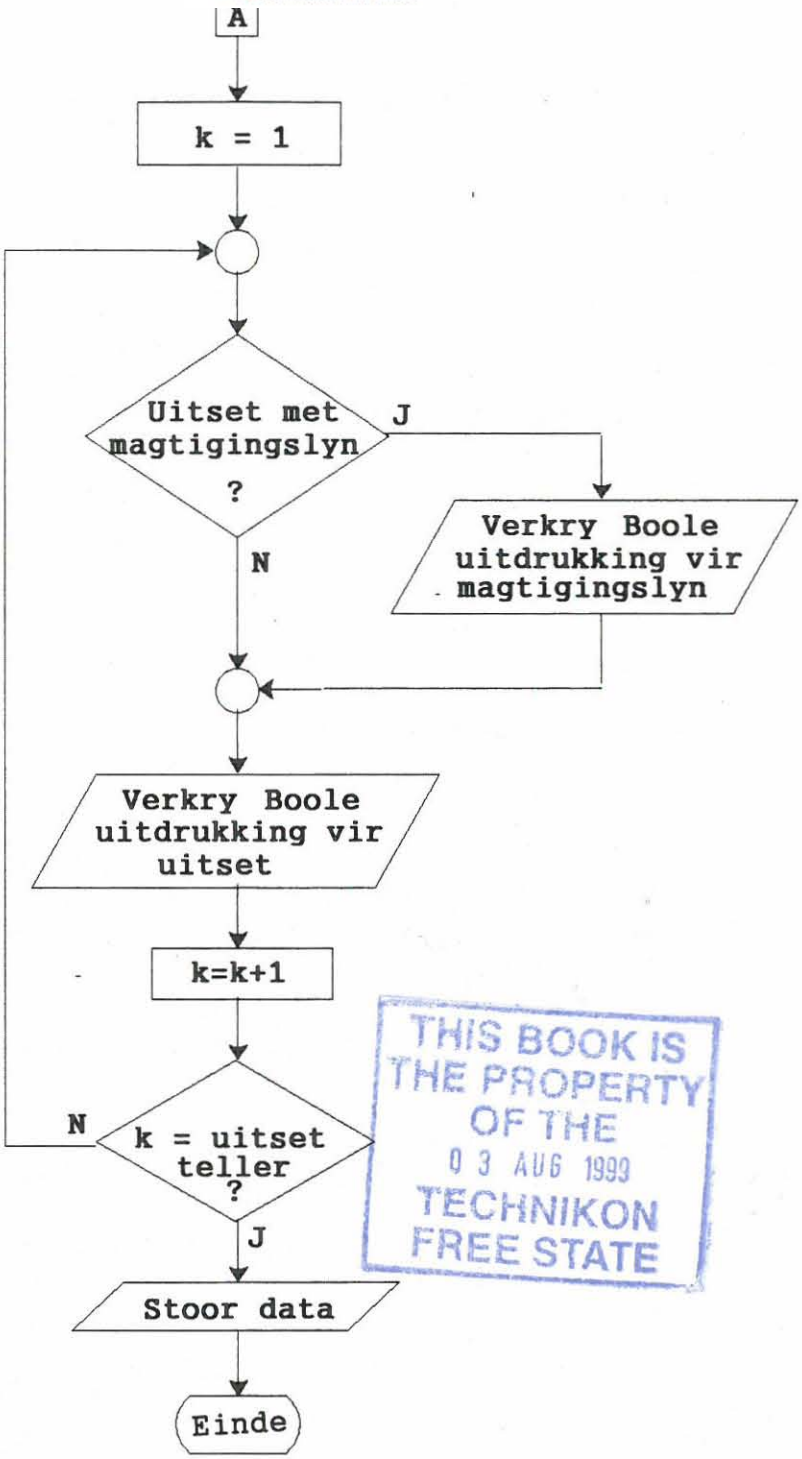


Fig. 4.2: (Vervolg) Vloeikaart van kringdatabasis-subprogram.

lyn verskaf word. 'n Voorbeeld van waar dit mag voorkom is by die 16L8 tipe PML. So 'n uitset, afhangend van die Boole uitdrukking van die

magtigingslyn, kan dus as 'n inset, uitset of 'n inset/uitset (dus beide rigtings) gebruik word.

Die magtigingslyn kan as volg voorgestel word:

- Y = 0, pen word as 'n inset gebruik en daar word dus geen Boole uitdrukking vir die uitset verskaf nie.
- Y = 1, pen word as uitset gebruik en 'n Boole uitdrukking word vir die uitset verskaf bv. Y = ABC + AB + CD
- Y = ABC, pen word as inset en 'n uitset gebruik en 'n Boole uitdrukking word ook vir die uitset verskaf bv. Y = XY + XZ + YZ. Dus wanneer ABC waar is word die pen as 'n uitset gebruik, andersins word dit as 'n inset gebruik.

4.3.2 KRINGDATASUBPROGRAM

Die korrektheid van data t.o.v enige kringontwerp in die kringdatabasis kan nagegaan word deur dit op skerm te vertoon of om 'n drukstuk daarvan te maak. Hierdie is 'n eenvoudige program en word in pseudokode voorgestel.

1. Vertoon name van kringe op skerm.
2. Watter kring?
   2.1. Vertoon op skerm?
      2.1.1. Ja - Vertoon data.
         2.1.1.1. Gaan na 3.

2.1.2. Anders - Gaan na 2.2.

2.2. Verlang drukstuk?

2.2.1. Ja - Skryf data na drukker.

2.2.1.1. Gaan na 3

2.2.2. Anders - Gaan na 3.

3. Stop?

3.1. Ja - Einde van program.

3.2. Anders - Gaan na 1.

4.3.3 SEKERINGKAARTSUBPROGRAM

Hierdie program maak gebruik van die kringdatabasis en die PML-databasis om 'n sekeringkaart van die betrokke kring saam te stel en op die skerm te vertoon of 'n drukstuk daarvan te maak. Indien die ontwerper tevrede is met die sekeringkaart, word hierdie inligting gebruik om 'n JEDEC lêer saam te stel. Die volgende pseudokode is 'n voorstelling van die program.

1. Vertoon name van kringe.
2. Selekteer kring.
3. Lees data van kring na rekenaar.
4. Vir i = 1 tot aantal penne

4.1. Uitset?

4.1.1. Ja - Word uitset gebruik in ontwerp ?

4.1.1.1. Ja - Verkry uitdrukking vir uitset.

4.1.1.1.1. Bepaal korrekte karakter vir al die insetlyne van elke produklyn betrokke by die uitdrukking bv. '-' of '*'.

4.1.1.1.2. Gaan na 4.1.3.

4.1.1.2. Anders - Skryf '+' karakter vir al die insetlyne van elke produklyn.

4.1.1.2.1. Gaan na 4.1.3.

4.1.2. Anders - Gaan na 4.1.3.

4.1.3. Volgende i.

5. Vertoon sekeringkaart op skerm.

6. Word drukstuk verlang?

6.1 Ja - Skryf sekeringkaart na drukker.

6.1.1. Gaan na 7.

6.2 Anders - Gaan na 7.

7. Word JEDEC lêer verlang?

7.1. Ja - Gaan na JEDEC roetine (sien paragraaf 4.3.3.1).

7.2. Anders - Einde van roetine.

4.3.3.1 JEDEC ROETINE

'n Lêer met die agtervoegsel .JED word geopen waarna inligting omtrent die kringontwerp na die lêer geskryf word. Anders as by die sekeringkaart, word net die inligting van produklyne wat betrokke is by die funksie, geprosesseer en na die JEDEC lêer geskryf.

Die lynnommer vir elke produklyn wat betrokke is word bepaal en die omskakeling van die funksie na 'n formaat van ene en nulle vind plaas. Hierdie formaat is te sien in die L-veld van fig. 2.7. Insetlyne wat nie in die betrokke tipe PML voorkom nie word ook bepaal en weggelaat. Nadat die L-veld van die JEDEC lêer voltooi is word die kontrolegetal bereken en na die JEDEC lêer geskryf.

Die kontrolegetal is van uiterste belang vir die programmering van die PML. Hierdie getal word deur die PML programmeerder gebruik om te bepaal of al die sekerings wat gesmelt moes word wel gesmelt is. Figuur 4.3 is 'n vloeidiagram van die JEDEC roetine.

4.3.4 VERTOON JEDEC SUBPROGRAM

Hierdie program lees slegs die data vanaf die skyf en vertoon dit op die skerm of verskaf 'n uitdruk.

4.3.5 REDIGERINGSUBPROGRAM

Hierdie program word gebruik om enige veranderings t.o.v. 'n sekere kringontwerp aan te bring. Dit behels dat die naam van die kring of 'n spesifieke pentoekenning verander kan word. Die Boole uitdrukking van een of meer uitsette kan ook verander word.

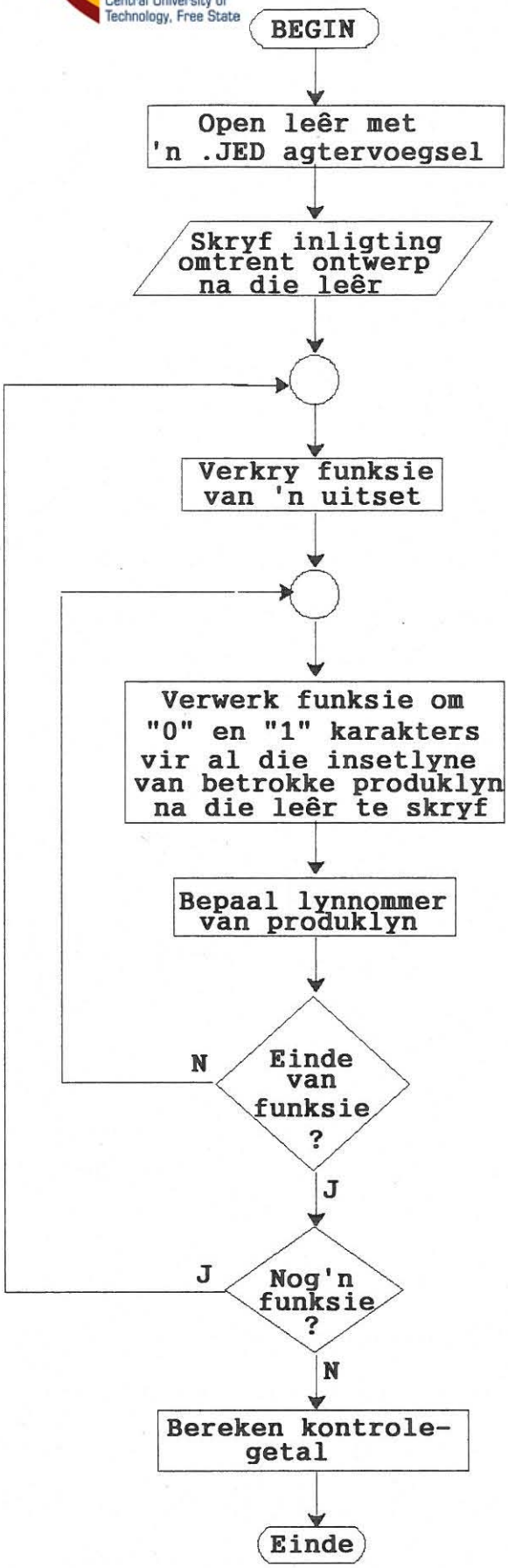


Fig. 4.3: Vloeikaart van JEDEC roetine

Die program laat ook toe dat 'n spesifieke kringontwerp uitgevee kan word of dat al die kringontwerpe tot op datum uitgevee kan word indien dit nie meer verlang word nie.

4.4 PML SIMULASIEPROGRAM

Die simulasieprogram maak gebruik van 'n JEDEC lêer en data in die PML-databasis om die werking van die spesifieke PML te simuleer. Die program bestaan uit die volgende subprogramme:

- Laaisubprogram
- Vertoon JEDEC subprogram
- PML subprogram
- Vertoon Sekeringkaartsubprogram
- Simulasiesubprogram

4.4.1 LAAI-SUBPROGRAM

Die doel van hierdie program is om 'n JEDEC lêer vanaf 'n harde- of sagteskyf in die geheue van die rekenaar in te lees. Die korrekte lêernaam, tesame met die agtervoegsel moet verskaf word. Indien nie, sal daar aan die gebruiker uitgewys word dat die verkeerde naam of agtervoegsel gebruik is.

4.4.2 PML SUBPROGRAM

Nadat die JEDEC lêer in die rekenaar se geheue gelaai is moet die korrekte PML gespesifiseer word wat in die kringontwerp gebruik is. Indien die gebruiker sou probeer om die PML te spesifiseer voordat die JEDEC lêer gelaai is sal die program aandui dat 'n JEDEC lêer eers gelaai moet word.

Indien die gebruiker nie weet watter PML vir die betrokke kring gebruik is nie, kan die JEDEC lêer eers na die rekenaar se skerm geskryf word. In die JEDEC lêer kan al hierdie inligting verkry word.

4.4.3 VERTOON SEKERINGKAARTSUBPROGRAM

Die roetine kompileer 'n sekeringkaart vanaf die inligting in die JEDEC lêer en vertoon hierdie sekeringkaart op die skerm. Die gebruiker kan hierdie program dus gebruik om die sekeringkaart te ontleed om te bepaal of die funksies korrek is al dan nie.

4.4.4 SIMULASIE SUBPROGRAM

Die program bepaal vanaf die JEDEC lêer watter uitsetpenne van die PML wel gebruik word (beginnende by pen 19 en eindigende by pen 11). Sodra 'n uitsetpen

gevind word wat wel gebruik word, word daar bepaal of hierdie uitset 'n magtigingsproduklyn het of nie. Indien wel word daar bepaal of die uitsetpen as 'n uitset, inset of tweerigting uitset gebruik word, soos bespreek in paragraaf 4.3.1. Wanneer hierdie uitset as 'n tweerigting uitset gebruik word, word bepaal watter insetlyne betrokke is by die aktivering van die uitset.

Hierna word die res van die uitsetpen se produklyne bepaal en word al die insetpenne wat by elk van hierdie produklyne betrokke is bepaal en in die rekenaar se geheue gestoor. Dit word bewerkstelig deur gebruik te maak van die data wat in die PML-databasis voorkom.

Wanneer al die uitsette wat gebruik word verkry is word hierdie uitsette op die skerm vertoon. Die gebruiker moet kies watter een van hierdie uitsette gesimuleer moet word. Al die uitsette kan ook gesamentlik gesimuleer word. Nadat die uitset gekies is, moet daar aangedui word hoeveel toetsvektore bepaal gaan word.

Indien daar aangedui word dat slegs een van die uitsette gesimuleer gaan word, word al die insetpenne wat by die uitset betrokke is op die skerm vertoon. Die gebruiker moet dan die logika kondisies wat 'n sekere toetsvektor uitmaak op die insetpenne aandui.

Nadat al die kondisies op die insette verkry is word daar bepaal of die uitset 'n magtigingslyn het al dan nie. Indien wel word bepaal of die uitset slegs as 'n uitset gebruik word of as 'n tweerigting uitset.

Indien die uitset as 'n tweerigting uitset gebruik word, word daar bepaal of die uitset gemagtig is met die gegewe insetkondisisies. Indien die uitset gemagtig is word die uitset bepaal wat die kondisies op die insette tot gevolg het en op die skerm vertoon. Indien die uitset egter nie gemagtig is nie word 'n hoë impedansie staat aangedui.

Wanneer die opsie gekies word om al die uitsette gelyktydig te simuleer, word daar eers bepaal of die betrokke PML registers bevat of nie. Indien wel word al die uitsette van die PML wat wel registers het, en wat gedefinieer is in die betrokke kringontwerp, op die skerm vertoon. Daar word dan van die gebruiker verlang om 'n aanvangskondisie op die omkeer-uitset van die registers aan te dui.

Hierna word al die insette en uitsette wat betrokke is op die skerm vertoon. Die eerste pen, wat die klok inset is, toon 'n "0" aan wat beteken dat die klokpuls laag is. Die verlangde logikakondisies moet dan vir al die insette verskaf word. Sodra die kondisies op die insette verskaf is word die inligting verwerk en die korrekte uitsetkondisies vertoon.

Die uitsette wat egter 'n register bevat sal nie van staat verander nie. Om die korrekte uitset hier te verkry moet die "enter" sleutel gedruk word sodat die klokpuls van 'n "0" na 'n "1" kan verander. Sodra pen 1 van 'n laag na 'n hoog verander neem die vertoonde register uitsette 'n waarde aan soos verwag na die toepassing van 'n klokpuls op die kring.

Nadat die verlangde aantal simulasievektore verskaf en die resultate vertoon is, word daar drie opsies verskaf om van te kies. Die eerste opsie is om al die simulasievektore en resultate weer te vertoon. Die tweede opsie laat die gebruiker die keuse om 'n drukstuk van die simulasievektore te verkry. Met die laaste opsie word daar teruggekeer na die PML simulasieprogam en die hele proses kan weer van vooraf plaasvind. Die uitdruk van die simulasieprogram kan in Bylaag D gesien word terwyl die volgende 'n pseudokode beskrywing van die simulasieproses is.

1. Lees JEDEC lêer vanaf skyf na rekenaargeheue.
2. Identifiseer PML gebruik.
3. Simuleer werking van PML?
   3.1. Ja - Een uitset?
      3.1.1. Ja - Verkry kondisies op insetpenne.
         3.1.1.1. Uitset met magtigingslyn?
            3.1.1.1.1. Ja - Is uitset gemagtig?
               3.1.1.1.1.1. Ja - Gaan na 3.1.1.1.2.

3.1.1.1.1.2. Anders - Uitset = Z.

3.1.1.1.2. Bepaal uitsetkondisie.

3.1.1.1.3. Gaan na 3.1.3.

3.1.2. Anders - PML met registers?

3.1.2.1. Ja - Verkry uitsetstaat van registers.

3.1.2.1.1. Verkry insetkondisies.

3.1.2.1.2. Bepaal uitsetkondisies van uitsette sonder registers en vertoon.

3.1.2.1.3. Wag vir klokpuls.

3.1.2.1.4. Bepaal en vertoon uitsetkondisie van register uitsette.

3.1.2.1.5. Gaan na 3.1.3.

3.1.2.2. Anders - Verkry insetkondisies.

3.1.2.2.1. Bepaal en vertoon uitsetkondisies.

3.1.3. Vertoon resultate weer?

3.1.3.1. Ja - Gaan na vertoon resultaat roetine.

3.1.3.1.1. Gaan na 3.1.3.

3.1.3.2. Anders - Word drukstuk verlang?

3.1.3.2.1. Ja - Skryf resultate na drukker.

3.1.3.2.1.1 Gaan na 3.

3.1.3.2.2. Anders - Gaan na 3.

3.2. Anders - Einde van program.

4.5 OPSOMMING

Daar is in hierdie navorsingsprojek so ver moontlik gepoog om die sagteware so gebruikersvriendelik en eenvoudig moontlik te hou. Die interaksie van die verskeie roetines, die 3 subprogramme en die hoofprogram dui op 'n goed gedefinieerde stelsel wat maklik evalueerbaar is.

HOOFSTUK 5

EVALUERING VAN SAGTEWARE

Programevaluering behels die toetsing van die sagteware deur data aan die program te verskaf en om te verseker dat dit korrek funksioneer. Dit is belangrik om te besef dat programtoetsing slegs die teenwoordigheid van foute aandui en nie die afwesigheid daarvan nie. Dus is dit moontlik dat foute nog steeds kan voorkom selfs nadat intensiewe toetse gedoen is.

Programtoetsing en programontfouting moet nie verwar word nie. Programtoetsing is 'n prosedure waarmee die moontlikheid van programfoute ondersoek word, terwyl progamontfouting 'n prosedure is om die spesifieke voorkoms van 'n fout aan te dui en dit te korrigeer (Sommerville, 1982: 152).

Tydens die ontwikkeling van die sagteware is daar van programtoetsing en -ontfouting gebruik gemaak om die korrekte werking van die program te verseker. Na voltooiing van die sagteware is die program ook baie goed getoets. Die belangrikste kriteria wat in ag geneem is, is as volg :

- Is die sekeringskaart 'n presiese weergawe van die Boole uitdrukkings wat verskaf is?

- Is die sekering kontrolegetal wat tydens die kompilering van die JEDEC lêer bereken is korrek?
- Is die resultate van die simulasieproses 'n ware weergawe van die kring wat gesimuleer is se waarheidstabel?

Die res van die hoofstuk handel oor twee kringe wat ontwerp en gesimuleer is (op 'n IBM aanpasbare rekenaar) met behulp van die sagteware wat ontwikkel is. Die eerste voorbeeld is 'n 7-segment-na-binêre enkodeerder waar daar van 'n 16L8 PML gebruik gemaak is. Die tweede is 'n teller wat van 0 tot 7 tel en waar daar van 'n herstellyn gebruik gemaak word om die teller te enige tyd na 0 te herstel en ook daar te hou. Hier is van 'n 16R4 PML gebruik gemaak.

Afdeling 5.1 behandel die ontwerp en simulering van die 7-segment-na-binêre enkodeerder terwyl afdeling 5.2 die tellerkring bespreek.

5.1 SEWE-SEGMENT-NA-BINÊRE ENKODEERDER

Daar is besluit om 'n enkodeerkring te ontwerp wat 'n omskakeling kan doen van 'n 7-segment na 'n aktief lae 4-bis binêre getal. Hierdie is 'n funksie wat nie beskikbaar is in 'n standaard geïntegreerde kring nie.

Die insette is die individuele segmentseine wat gewoonlik van a tot g gemerk is soos gesien in figuur 5.1. Figuur 5.2 toon aan hoe die syfers op 'n 7-segment-vertooneenheid verteenwoordig word.

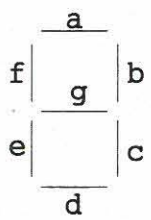


Fig. 5.1: Segmentseine

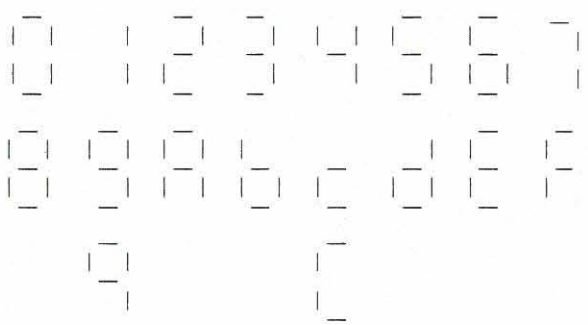


Fig. 5.2: 7-segment-vertoonkodes.

Daar moet gelet word dat 9 en C op twee verskillende maniere vertoon kan word.

Daar is op die volgende veranderlikkes besluit:

Pen 1 = a Inset, segment a

Pen 2 = b Inset, segment b

Pen 3 = c Inset, segment c

Pen 4 = d Inset, segment d

Pen 5 = e Inset, segment e

Pen 6 = f Inset, segment f

Pen 7 = g Inset, segment g

Pen 16 = !W Uitset, minsbeduidende bis van binêre kode

Pen 17 = !Y Uitset

Pen 18 = !X Uitset

Pen 19 = !Z Uitset, meesbeduidende bis van binêre kode

Uit voorafgaande kan die volgende definisies afgelei word:

- zero = a & b & c & d & e & f & !g
- een = !a & b & c & !d & !e & !f & !g
- twee = a & b & !c & d & e & !f & g
- drie = a & b & c & d & !e & !f & g
- vier = !a & b & c & !d & !e & f & g
- vyf = a & !b & c & d & !e & f & g
- ses = a & !b & c & d & e & f & g
- sewe = a & b & c & !d & !e & !f & !g
- agt = a & b & c & d & e & f & g
- nege = a & b & c & !d & !e & f & g

  of a & b & c & d & !e & f & g
- a = a & b & c & !d & e & f & g
- b = !a & !b & c & d & e & f & g
- c = !a & !b & !c & d & e & !f & g

  of a & !b & !c & d & e & f & !g
- d = !a & b & c & d & e & !f & g
- e = a & !b & !c & d & e & f & g
- f = a & !b & !c & !d & e & f & g

Die aktiewe bisse vir die verskillende heksadesimale getalle kan as volg opgesom word:

- !W = een, drie, vyf, sewe, nege, b, d en f
- !Y = twee, drie, ses, sewe, a, b, e en f
- !X = vier, vyf, ses, sewe, c, d, en f
- !Z = agt, nege, a, b, c, d, e en f

Die produkterme is d.m.v. die Quine-McCluskey metode as volg uitgebrei:

- !W = abcd!eg + abc!efg + acd!efg + bc!d!e!f!g + !b!c!defg + a!b!defg + !abcde!fg
- !Y = a!b!cefg + !bcdefg + abc!defg + ab!cde!fg + abcd!e!fg + abc!d!e!f!g
- !X = a!b!cdefg + a!bcdfg + a!b!cefg + !abc!d!efg + !abcde!fg + !a!b!cde!fg + abc!d!e!f!g
- !Z = abcfg + a!b!cdef + a!b!cefg + !a!bcdefg + !abcde!fg + !a!b!cde!fg

Figuur 5.3 is 'n uitdruk van die kringontwerp soos dit deur die sekeringkaartprogram saamgestel is. Hier kan die veranderlikes wat gedefinieer is vergelyk word en ook seker gemaak word of die Boole terme korrek is al dan nie.

```
Summary of 7toBIN making use of 16L8

Pin 1 : INPUT = a
Pin 2 : INPUT = b
Pin 3 : INPUT = c
Pin 4 : INPUT = d
Pin 5 : INPUT = e
Pin 6 : INPUT = f
Pin 7 : INPUT = g
Pin 8 : INPUT =
Pin 9 : INPUT =
Pin 10 : VCC
Pin 11 : INPUT =
Pin 12 : OUTPUT =
Pin 13 : OUTPUT =
Pin 14 : OUTPUT =
Pin 15 : OUTPUT =
Pin 16 : OUTPUT = !W
Pin 17 : OUTPUT = !Y
Pin 18 : OUTPUT = !X
Pin 19 : OUTPUT = !Z
Pin 20 : GND
Pin 16
Output enable : !W = 1
Expression : !W = abcd!eg + abc!efg + acd!efg +
bc!d!e!f!g + !b!c!defg + a!b!defg + !abcde!fg
Pin 17
Output enable : !Y = 1
Expression : !Y = a!b!cefg + !bcdefg + abc!defg +
ab!cde!fg + abcd!e!fg + abc!d!e!f!g
Pin 18
Output enable : !X = 1
Expression : !X = a!b!cdefg + a!bcdfg + a!b!cefg +
!abc!d!efg + !abcde!fg + !a!b!cde!fg + abc!d!e!f!g
Pin 19
Output enable : !Z = 1
Expression : !Z = abcfg + a!b!cdef + a!b!cefg + !a!bcdefg
+ !abcde!fg + !a!b!cde!fg
```

Fig. 5.3: Kringontwerp van die 7-segment-na-binêre enkodeerder.

Figuur 5.4 is 'n uitdruk van die sekeringkaart waar daar nagegaan kan word of die korrekte sekerings gesmelt sal word in ooreenstemming met die Boole uitdrukkings.

Die betekenis van karakters wat in die sekeringkaart voorkom is as volg:

- *, Sekering bly ongeskonde.
- -, Sekering word gesmelt.
- +, Sekerings word nie in ontwerp gebruik nie.
- Geen karakter, produklyne bestaan nie by PML nie.

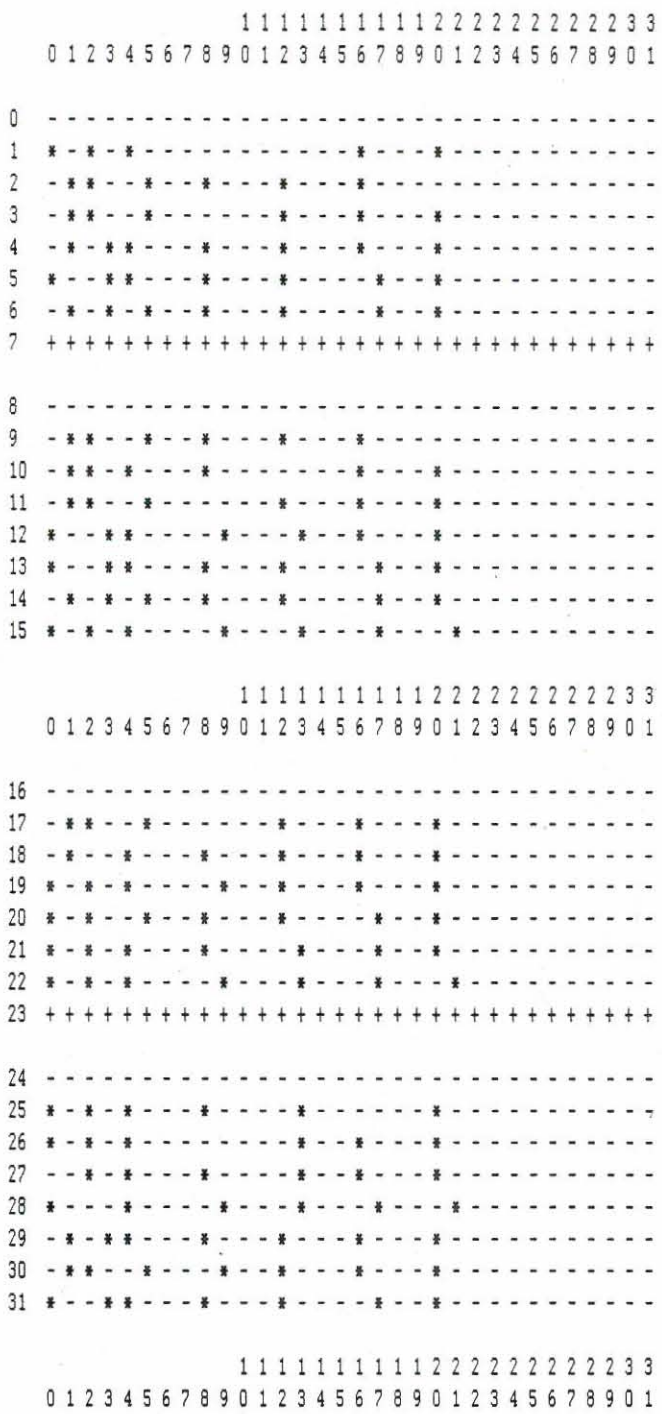


Fig. 5.4: Uitdruk van sekeringkaart van die 7-segment-na-binêre enkodeerder.

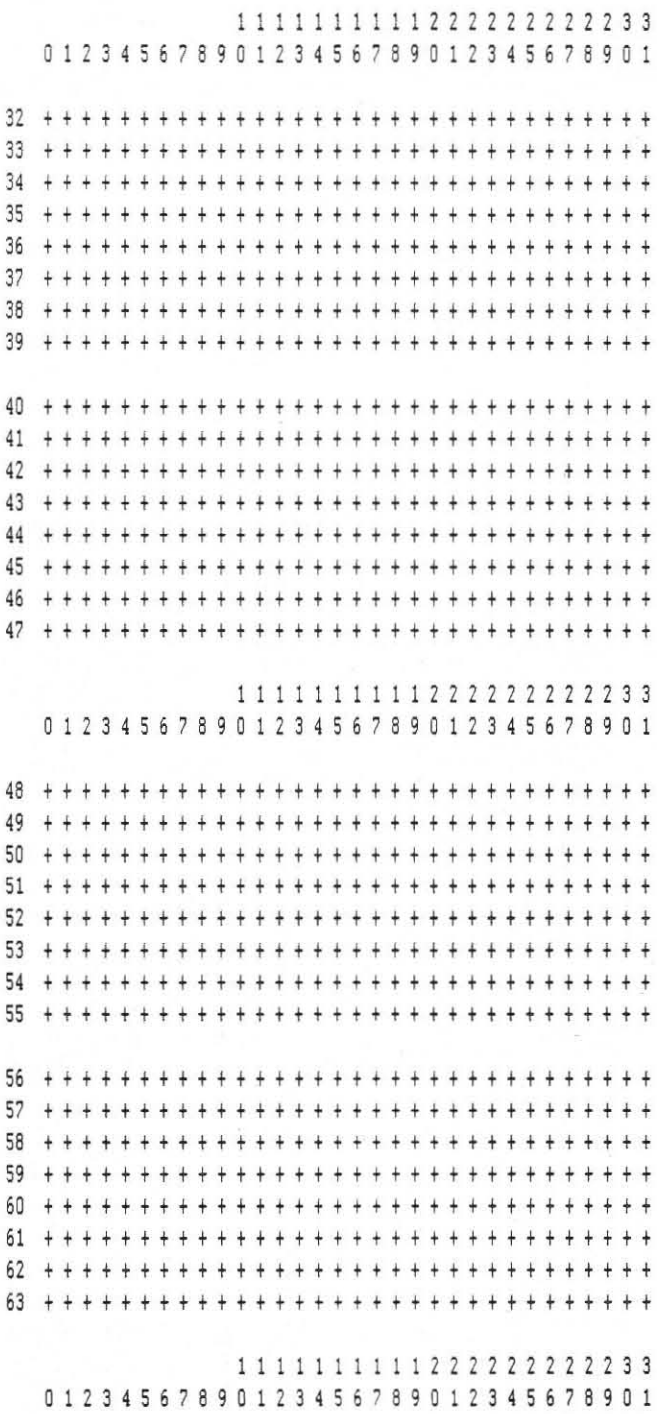


Fig. 5.4: (Vervolg) Uitdruk van sekeringkaart van die 7-segment-na-binêre enkodeerder.

Die JEDEC lêer soos saamgestel deur die sekeringkaart-program kan in figuur 5.5 gesien word. Hier kan nagegaan

word of die sekerings wat gesmelt moet word ooreenstem met die in die sekeringkaart. Daar kan ook vasgestel word of die sekeringkontrolegetal korrek is.

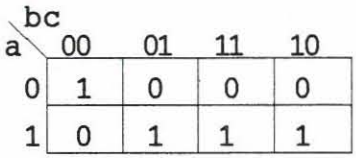
```
•PAL : 16L8
Project name : 7toBIN
Designed by : G.J. Booysen
*
QP20*
QV*
GO*
FO*
L000000 1111 1111 1111 1111 1111 1111 1111 1111*
L000032 0101 0111 1111 1111 0111 0111 1111 1111*
L000064 1001 1011 0111 0111 0111 1111 1111 1111*
L000096 1001 1011 1111 0111 0111 0111 1111 1111*
L000128 1010 0111 0111 0111 0111 0111 1111 1111*
L000160 0110 0111 0111 0111 1011 0111 1111 1111*
L000192 1010 1011 0111 0111 1011 0111 1111 1111*
L000256 1111 1111 1111 1111 1111 1111 1111 1111*
L000288 1001 1011 0111 0111 0111 1111 1111 1111*
L000320 1001 0111 0111 1111 0111 0111 1111 1111*
L000352 1001 1011 1111 0111 0111 0111 1111 1111*
L000384 0110 0111 1011 1011 0111 0111 1111 1111*
L000416 0110 0111 0111 0111 1011 0111 1111 1111*
L000448 1010 1011 0111 0111 1011 0111 1111 1111*
L000480 0101 0111 1011 1011 1011 1011 1111 1111*
L000512 1111 1111 1111 1111 1111 1111 1111 1111*
L000544 1001 1011 1111 0111 0111 0111 1111 1111*
L000576 1011 0111 0111 0111 0111 0111 1111 1111*
L000608 0101 0111 1011 0111 0111 0111 1111 1111*
L000640 0101 1011 0111 0111 1011 0111 1111 1111*
L000672 0101 0111 0111 1011 1011 0111 1111 1111*
L000704 0101 0111 1011 1011 1011 1011 1111 1111*
L000768 1111 1111 1111 1111 1111 1111 1111 1111*
L000800 0101 0111 0111 1011 1111 0111 1111 1111*
L000832 0101 0111 1111 1011 0111 0111 1111 1111*
L000864 1101 0111 0111 1011 0111 0111 1111 1111*
L000896 0111 0111 1011 1011 1011 1011 1111 1111*
L000928 1010 0111 0111 0111 0111 0111 1111 1111*
L000960 1001 1011 1011 0111 0111 0111 1111 1111*
L000992 0110 0111 0111 0111 1011 0111 1111 1111*
C70C5*❤
```

Fig. 5.5: JEDEC lêer van 7-segment-na-binêre enkodeerder.

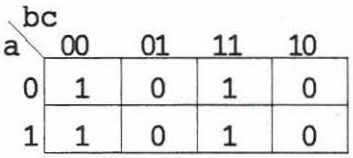
Figuur 5.6 is 'n uitdruk van die resultate van die simulasieproses en verteenwoordig 'n korrekte weergawe van die betrokke kring se waarheidstabel.

Tabel 5.1 Waarheidstabel van modulus-8 teller.

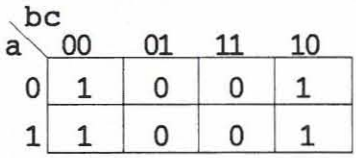
!a	!b	!c	a	b	c	Da	Db	Dc
0	0	0	1	1	1	1	1	0
0	0	1	1	1	0	1	0	1
0	1	0	1	0	1	1	0	0
0	1	1	1	0	0	0	1	1
1	0	0	0	1	1	0	1	0
1	0	1	0	1	0	0	0	1
1	1	0	0	0	1	0	0	0
1	1	1	0	0	0	1	1	1



!a = !a!b!c + ac + ab



!b = !b!c + bc



!c = !c

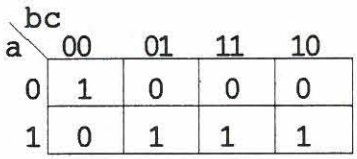
Fig. 5.7: Karnaugh kaarte van teller

Om die kring te herstel of by 0 te hou is daar 'n term by die uitdrukings gevoeg, om die volgende te verkry:

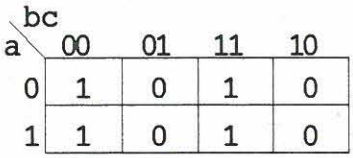
- !a = !a!b!c + ac + ab + r
- !b = !b!c + bc + r
- !c = !c + r

Tabel 5.1 Waarheidstabel van modulus-8 teller.

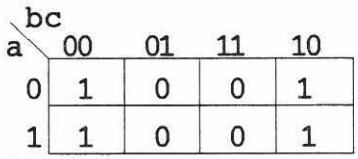
!a	!b	!c	a	b	c	Da	Db	Dc
0	0	0	1	1	1	1	1	0
0	0	1	1	1	0	1	0	1
0	1	0	1	0	1	1	0	0
0	1	1	1	0	0	0	1	1
1	0	0	0	1	1	0	1	0
1	0	1	0	1	0	0	0	1
1	1	0	0	0	1	0	0	0
1	1	1	0	0	0	1	1	1



!a = !a!b!c + ac + ab



!b = !b!c + bc



!c = !c

Fig. 5.7: Karnaugh kaarte van teller

Om die kring te herstel of by 0 te hou is daar 'n term by die uitdrukings gevoeg, om die volgende te verkry:

- !a = !a!b!c + ac + ab + r
- !b = !b!c + bc + r
- !c = !c + r

'n Uitdruk van die kringontwerp soos in voorafgaande bespreek kan in figuur 5.8 gesien word. Figuur 5.9 is 'n uitdruk van die sekeringkaart, terwyl die JEDEC lêer soos saamgestel deur die sekeringkaartprogram in figuur 5.10 gesien kan word. Figuur 5.11 is 'n uitdruk van die resultate verkry deur die simulasie van die teller.

```
Summary of counter making use of 16R4

Pin 1  : INPUT = CLK
Pin 2  : INPUT = r
Pin 3  : INPUT =
Pin 4  : INPUT =
Pin 5  : INPUT =
Pin 6  : INPUT =
Pin 7  : INPUT =
Pin 8  : INPUT =
Pin 9  : INPUT =
Pin 10 : VCC
Pin 11 : INPUT =
Pin 12 : OUTPUT = !a
Pin 13 : OUTPUT = !b
Pin 14 : OUTPUT = !c
Pin 15 : OUTPUT =
Pin 16 : OUTPUT =
Pin 17 : OUTPUT =
Pin 18 : OUTPUT =
Pin 19 : OUTPUT =
Pin 20 : GND

Pin 14
Expression     : !a = ab + !a!b!c + ac + r
Pin 15
Expression     : !b = bc + !b!c + r
Pin 16
Expression     : !c = !c + r
```

Fig. 5.8: Kringontwerp van modulus-8 teller

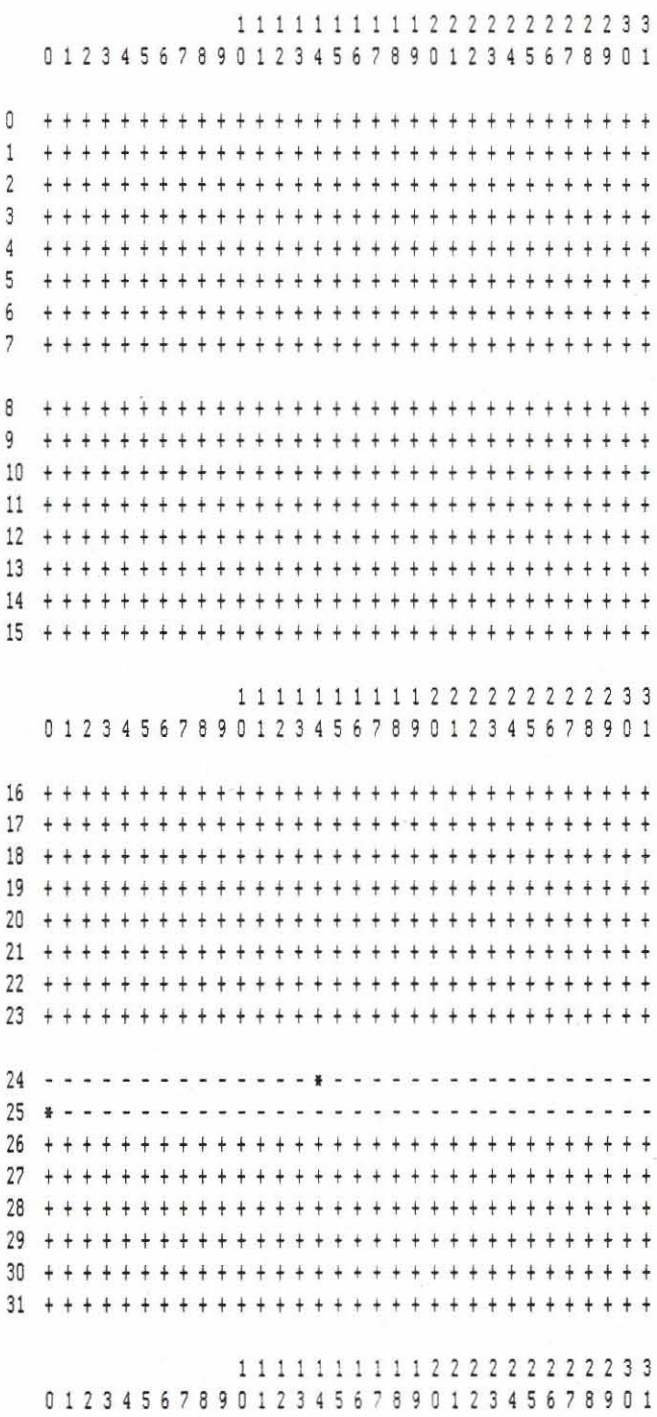


Fig. 5.9: Sekeringkaart van modulus-8 teller

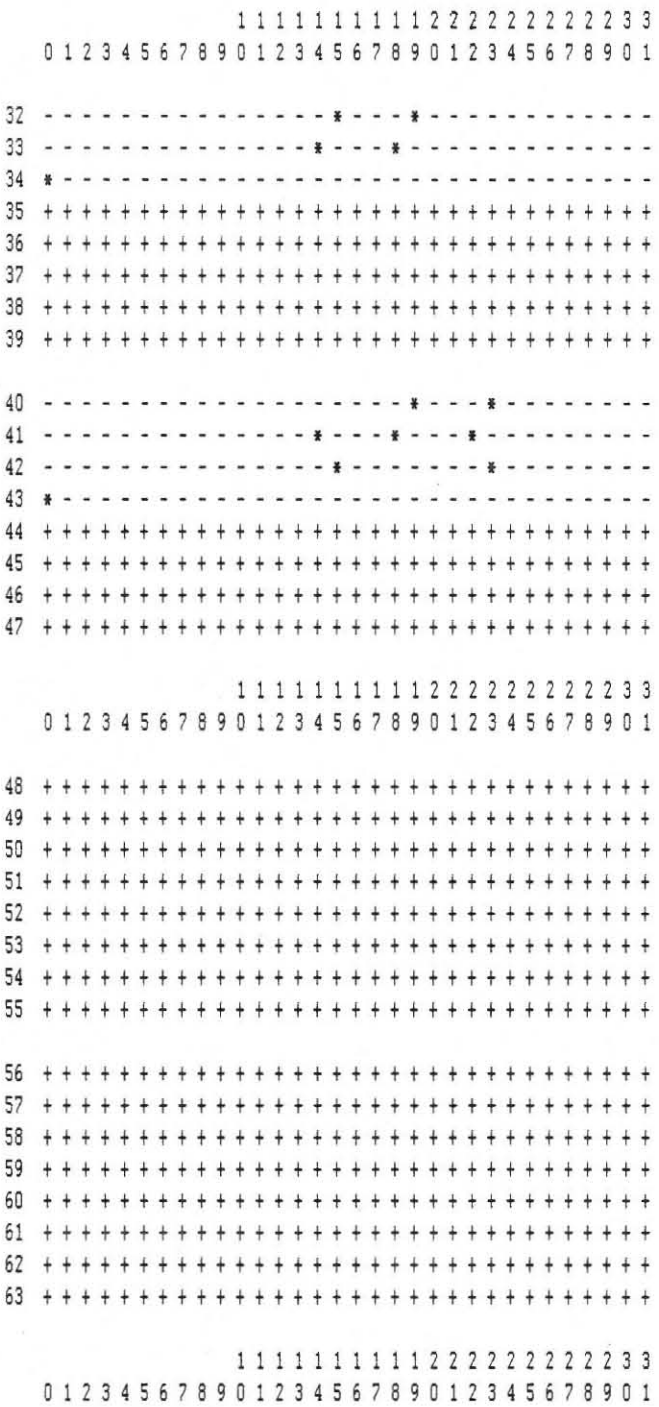


Fig. 5.9: (Vervolg) Sekeringkaart van modulus-8 teller

```
•PAL : 16R4
Project name : counter
Designed by : G.J. Booysen
*
QP20*
QV*
GO*
FO*
L000768 1111 1111 1111 1101 1111 1111 1111 1111*
L000800 0111 1111 1111 1111 1111 1111 1111 1111*
L001024 1111 1111 1111 1110 1110 1111 1111 1111*
L001056 1111 1111 1111 1101 1101 1111 1111 1111*
L001088 0111 1111 1111 1111 1111 1111 1111 1111*
L001280 1111 1111 1111 1111 1110 1110 1111 1111*
L001312 1111 1111 1111 1101 1101 1101 1111 1111*
L001344 1111 1111 1111 1110 1111 1110 1111 1111*
L001376 0111 1111 1111 1111 1111 1111 1111 1111*
C20C1*♥
```

Fig. 5.10: JEDEC lêer van teller

Testvectors for all the outputs of counter.jed :

Sta PIN	CLK 1	IN 2	OUT 14	OUT 15	OUT 16
Vec					
1	0	0	x	x	x
	1	0	0	0	0
2	0	0	0	0	0
	1	0	0	0	1
3	0	0	0	0	1
	1	0	0	1	0
4	0	0	0	1	0
	1	0	0	1	1
5	0	0	0	1	1
	1	0	1	0	0
6	0	0	1	0	0
	1	0	1	0	1
7	0	0	1	0	1
	1	0	1	1	0
8	0	0	1	1	0
	1	0	1	1	1
9	0	0	1	1	1
	1	0	0	0	0
10	0	0	0	0	0
	1	0	0	0	1
11	0	0	0	0	1
	1	0	0	1	0
12	0	0	0	1	0
	1	0	0	1	1
13	0	1	0	1	1
	1	1	0	0	0

Fig. 5.11: Gesimuleerde uitset van teller

14	0	0	0	0	0
	1	0	0	0	1
15	0	0	0	0	1
	1	0	0	1	0
16	0	0	0	1	0
	1	0	0	1	1
17	0	1	0	1	1
	1	1	0	0	0
18	0	1	0	0	0
	1	1	0	0	0
19	0	1	0	0	0
	1	1	0	0	0
20	0	0	0	0	0
	1	0	0	0	1

Fig 5.11: (Vervolg) Gesimuleerde uitset van teller

5.3 GEVOLGTREKKING

Die resultate behaal met die twee kringe wat bespreek is dui op die korrekte werking van die sagteware wat ontwikkel is, maar verteenwoordig nie 'n omvattende evalueringsproses nie. Daar is egter 'n wye verskeidenheid kringontwerpe met groot sukses getoets. Daar het wel verskeie foute tydens die implementeringsfase voorgekom, wat onmiddelik reggestel is.

Die sagteware is ook reeds met groot sukses deur studente aan die Technikon OVS gebruik, waar die JEDEC lêer wat gekompileer is ook gebruik is om die PMLs te programmeer.

'n Belangrike gevolgtrekking wat gemaak kan word is dat die JEDEC lêer die kern is waarom die implementering van PMLs draai. Daarom is die simulasie-sagteware so

ontwikkel dat die JEDEC lêer gebruik word om die werking van 'n PML te simuleer i.p.v. die Boole uitdrukings.

Die sagteware wat ontwikkel is om die JEDEC lêer te kompileer is volledig getoets en funksioneer korrek. Verder kan daar afgelei word dat die PML-databasis alle data omtrent die PMLs bevat wat benodig word om so 'n JEDEC lêer te kompileer. Hierdie aspek is ook gekontroleer deur die uitvoering van die PML-databasis verifikasie-subprogram, waartydens die inligting in die PML-databasis vergelyk is met die relevante data in PML databoeke. Die sagteware wat ontwikkel is om die PML-databasis saam te stel voldoen dus aan alle vereistes.

Die korrekte werking van die sagteware is bevestig deur die praktiese implementering van 'n verskeidenheid ontwerpe. Verskeie van hierdie kringe is in onverwante projekte gebruik en bevredigende resultate is daarmee behaal.

Die afhandeling van die projek het dus gelei tot die ontwikkeling van 'n volledige stel sagteware vir die totale implementering van 'n groot verskeidenheid PMLs. Die voltooiing daarvan het die navorser deur 'n studieproses gevoer waartydens besondere kennis aangaande onderstaande ingewin is:

- Eienskappe en programmering van PMLs.
- Formaat en samestelling van JEDEC lêers.

- Die beginsel en toepassing van simulasie van elektroniese kringe in die algemeen - en PMLs in die besonder.

HOOFSTUK 6

SAMEVATTING

Die hoofdoel van die projek was om die programmering en veral simulering van PMLs te bestudeer.

'n Teoretiese oorsig van programmeerbare logika toestelle is in Hoofstuk 2 gelewer. Die PML word dan ook hier as 'n PLT bespreek.

Die simulasie van elektroniese kringe word in Hoofstuk 3 bespreek. Hier word die simulasie van analoogkringe sowel as digitale toestelle bespreek.

Die sekering uitleg en die werking van PMLs is ondersoek deur die ontwikkeling van die sekeringskaart- en simulasiesagteware soos bespreek in Hoofstuk 4. Deur die simulasie en programmering van 'n aantal PMLs, was dit moontlik om die akkuraatheid van die beginsels soos verstaan gedurende die proses te bepaal.

Die ontwerp van 'n 7-segment-na-binêre enkodeerder en 'n modulus-8 teller soos geïmplementeer word in hoofstuk 5 beskryf. In albei gevalle is die JEDEC lêer foutloos geskep en kom die resultate van die simulasieproses ooreen met die waarheidstabelle van die kringe.

Dit dui dus daarop dat die data in die PML-databasis voldoende en korrek is, dat die JEDEC lêer korrek gekompileer is en dat die simulasieprogram korrek funksioneer.

BYLAAG A
MAINPROG.PAS

```
{*************************************************************************}
{This is the main program that display a menu from where a choice can be made}
{to run one of three programs viz. PAL_DATA.PAS, FUSEMAP.PAS or SIM.PAS      }
{*************************************************************************}

{$M 1024,0,8}                              {STACK SIZE = 10k, HEAP MIN = 0K, HEAP MAX = 8K
Program MainProgram;

Uses                                                                {Units to be linked to program}
  Crt,                                                              {Pascal CRT unit}
  Dos;                                                              {Pascal Dos unit}

{*****************************************}
{Procedure to show main menu and select topic}
{*****************************************}
Procedure DoMenu;

 Var
   Wait : Char;
   I    : Byte;

 Begin
   Repeat
     Clrscr;
     GotoXY(20,10);
     Write('+----------------------------------+');
     For I := 11 to 19 do
      Begin
        GotoXY(20,I);
        Write('¦');
        GotoXY(57,I);
        Write('¦');
      End;
     GotoXY(20,20);
     Write('+----------------------------------+');
     GotoXY(25,12);
     Write('1 - PAL DATABASE');
     GotoXY(25,13);
     Write('2 - FUNCTIONS');
     GotoXY(25,14);
     Write('3 - SIMULATOR');
     GotoXY(25,16);
     Write('Esc - Exit');
     GotoXY(25,19);
```

BYLAAG A
MAINPROG.PAS

```
    Write('Option : ');
     Repeat
       Wait := Readkey;
       Case Wait of
         '1' : Begin
                 Write(wait);
                 Exec('PAL_DATA.PAS','');
               End;
         '2' : Begin
                 Write(wait);
                 Exec('FUSEMAP.PAS','');
               End;
         '3' : Begin
                 Write(wait);
                 Exec('SIM.PAS','');
               End;
        End ; {Case}
     Until (Wait = '1') or (Wait = '2') or (Wait = '3') or (Wait = #27);
  Until (Wait = #27);                                          {Esc}
End;{Domenu}

Begin
  DoMenu;                                                      {Start program}
End.
```

BYLAAG B
PAL_DATA.PAS

```
{************************************************************}
{The purpose of this program is to create a PAL database to }
{view data in the database and to edit data in the database}
{************************************************************}
Program Pal_Data;

Uses Crt,Dos;

Const
     MaxNo = 20;                                    {Max number of pins and product lines}
     MaxPal = 30;                                   {Max number of Pal types }
     Printer = 'PRN';                               {Pointer to printer}
     screen = '';                                   {Pointer to screen}

TYPE
  PalRec =  Record
    Name         : String[5];                       {Pal name}
    State        : Char;                            {States of PAL outputs}
    Count        : Byte;                            {Number of pins}
    Status       : Array[1..MaxNo] OF String[6];    {Status of pins}
    Inputline    : ARRAY[1..MaxNo] OF Integer;      {Input pin connected to which input line}
    Proden       : array[1..MaxNo] Of Char;         {Output with a product enable line or not}
    Productline  : ARRAY[1..MaxNo] OF Integer;      {The number of the first product line of the output}
    Reg          : Array[1..Maxno] Of Char;         {Output with register}
    NoOf         : ARRAY[1..MaxNo] OF Integer;      {Number of Product lines connected to OR gate of output pin}
    Feedback     : ARRAY[1..MaxNo] OF Char;         {Output with feedback}
    Fback        : ARRAY[1..MaxNo] OF Integer;      {Feedback to which input line}
  End;{PalRec}

VAR
  PRec                : PalRec;                     {Variable to PAL data}
  PFile               : FILE OF  PalRec;            {Typed file for PAL data}  {
  Choice,Save,enable,
  wipe,over           : Char;
  i,j,n,              : Integer;                    {Misc variables for general use}
  RecNumber,          : Integer;                    {Number of PALs in database}
  digit,              : Integer;                    {Specific PAL to be used}
  pos                 : Integer;                    {Pin to be editted}
  Quits,Done,Quit     : Boolean;
```

BYLAAG B
PAL_DATA.PAS

```
  Name               : String[5];                          {PAL name}
  Xlate              : string[6];                          {String of 6 caracters}
  PalSet             : Set of Char;                        {PAL output conditions}
                                                           {Pals in database}
  PalTypes           : Array[0..MaxPal] Of String[5];

Procedure Get;Forward;

{**************************************}
{Procedure to save Pal data in database}
{**************************************}
PROCEDURE  WriteData;
 Begin
   Assign( PFile,'PalFile.PAL' );
   {$i-}
      Reset(PFile);
   {$i+}
   IF (IoResult <> 0) THEN
     Rewrite(PFile);
   If  over = 'Y' Then                                     {Write over existing data}
    Begin
      Seek(Pfile,digit);
      Over := 'N';
    End{if}
   Else
    Seek( PFile,FileSize(PFile));                          {New data at end of file}
   Write(PFile,PRec);
   Close(PFile)
 End;

{**********************************}
{Procedure to add data to database}
{**********************************}
PROCEDURE AddRecords;
 Begin
   For i := 0 to MaxNo do
    Begin
      Prec.reg[i] := 'n';
      prec.inputline[i] := 99;
```

BYLAAG B
PAL_DATA.PAS

```
    prec.productline[i] := 99;
  End;{for}
 PalSet := ['l','L','h','H','r','R'];
 Writeln('Add a new record. ');
 Writeln;
 Write('PAL name ? : ');
 Readln(Prec.Name);
 Over := 'N';
 For i := 0 to Recnumber do
  Begin
    If Paltypes[i] = Prec.name Then
     Begin
       Write('Pal already exist, do you  want to overwrite ? Y/N ');
       Read(over);
       Over := Upcase(over);
       Digit := i;
     End;{if}
  End;{for}
 If (Prec.name <> paltypes[i]) or (over = 'Y') Then
  Begin
    WITH PRec DO
     Begin
       Done := False;
       i := 1;
       Repeat
         If Name[i] in PalSet Then
          Begin
            State := Upcase(Name[i]);
            Done := true;
          End{If}
         Else
          Inc(i);
       Until Done;
       Write('Number of pins ? : ');
       Readln(Count);
       Writeln('Give pin status, Input, Output , GND , VCC ext.');
       Writeln;
       For i := 1 TO Count DO
        Begin
```

BYLAAG B
PAL_DATA.PAS

```
Write('Status of pin no ',i,' ? : ');
Readln(Status[i]);
Xlate := Status[i];
For j := 1 to length(xlate) do
 Begin
   Xlate[j] := Upcase(Xlate[j]);
 End;{for}
Status[i] := Xlate ;
If Status[i] = 'INPUT' Then
 Begin
   Productline[i] := 99;
   Write('Input ',i,' connect to which input line ? : ');
   Readln( Inputline[i] )
 End{if}
Else
 If Status[i] = 'OUTPUT' Then
  Begin
    Inputline[i] := 99;
    If State = 'R' Then
     Begin
       Done := False;
       Repeat
         Write('Is it a Register Output Y/N : ');
         Readln(Choice);
         Case Choice Of
           'Y','y','N','n' : Done := true;
         Else
          Write('Error : Try again. ');
         End;{case}
       Until Done;
       Reg[i] := Upcase(choice);
       If Reg[i] = 'Y' Then
        Proden[i] := 'N'
       Else
        Begin
          Write('Is there a product line to enable output  Y or N ? ');
          Readln(Proden[i]);
          Proden[i] := upcase(Proden[i]);
        End;{else}
```

BYLAAG B
PAL_DATA.PAS

```
            End{if}
           Else
            Begin
              Write('Is there a product line to enable output  Y or N ? ');
              Readln(Proden[i]);
              Proden[i] := upcase(Proden[i]);
            End;{else}
           Write('First product line connect to output ',i,' ? : ');
           Readln( Productline[i] );

           Write('Number of Product lines conected to OR gate of output ',i,' ? : ');
           Readln( NoOF[i] );
           Done := False;
           WHILE NOT Done DO
            Begin
              Write('Output with feedback Y or N :');
              Readln( FeedBack[i] );
              FeedBack[i] := Upcase(FeedBack[i]);
              IF FeedBack[i] = 'Y' THEN
               Begin
                 Done := True;
                 Write(' Feedback to which input line :');
                 Readln( Fback[i] );
               End{if}
              Else
               Begin
                 Done := True;
                 Fback[i] := 0;
               End;{else}
            End;{while}
         End;{if ,status}
       End;{For,count}
     End; {With}
    WriteData;
    Get;
  End;{if}
 Readln;
End;{addrecords}
```

BYLAAG B
PAL_DATA.PAS

```
PROCEDURE PrintRecord;                                    {To print data}
 Begin
   Assign(OUTPUT,Printer);
   Rewrite(OUTPUT);
 End;

PROCEDURE  ReadRecord;                                    {To view data}
 Begin
   Assign(OUTPUT,Screen);
   Rewrite(OUTPUT);
 End;


{*****************************************************************}
{Procedure to read PAL data from database and to view or print it}
{*****************************************************************}
PROCEDURE  ReadPALs;
 Begin
   Clrscr;
   Assign( PFile,'PalFile.PAL' );
  {$i-}
   Reset(PFile);
  {$i+}
   IF IoResult <> 0 THEN
    Writeln('No data of PALs yet')
   ELSE
    Begin
      Writeln('Pals in database');
      Writeln;
      j := 0;
      n := 3;
      For i := 0 to RecNumber Do
       Begin
         GotoXY(j*18,n);
         Write('_ ');
         Write(i + 1,'. ',PalTypes[i]);
         Inc(j);
         If j = 4 Then
          Begin
```

BYLAAG B
PAL_DATA.PAS

```
        Writeln;
        j := 0;
        Inc(n);
      End;{if}
   End;{For}
  Writeln;
  Writeln;
  quit := False;
  Repeat
    Write('Enter digit Of PAL to be read : ');
    Readln(digit);
    For i := 1 To Recnumber + 1 do
     Begin
       If i = digit Then
        Quit := True;
     End;
  Until Quit;
  Writeln;
  Quit := False;
  Repeat
    Write('Output to S.creen, P.rinter ? ');
    Readln(Choice);
    Case Choice Of
     'P','p' : Begin
                 Printrecord;
                 Quit := True;
               End;
     'S','s' : Begin
                 Readrecord;
                 Quit := True;
               End;
    End;{case}
  Until Quit;
  Clrscr;
  Seek(pfile,(digit - 1));
  Read(PFile,PRec);
  Writeln('Data of PAL : ',PRec.Name);
  If Prec.State = 'L' Then
```

BYLAAG B
PAL_DATA.PAS

```
      Writeln('Outputs is active low.');
     If Prec.State = 'H' Then
      Writeln('Outputs is active High.');
     If Prec.State = 'R' Then
      Writeln('Outputs is active low.');
     Writeln('Number of pins : ',PRec.Count);
     FOR i := 1 TO PRec.Count DO
      Begin
        Write('Pin ',i,' ', PRec.Status[i]);
        If PRec.Status[i] = 'INPUT' Then
         Begin
           Writeln(', connect to input line ',PRec.Input line[i]);
         End {If,input}
        Else
         Begin
           If PRec.Status[i] = 'OUTPUT' Then
            Begin
              Write(', 1st Pline ',PRec.Productline[i]);
              Write(', No.of Plines ',PRec.NoOf[i]);
              If Prec.Proden[i] = 'Y' Then
              Write(', Pline ',Prec.Productline[i] - 1,' = oe');
              If Prec.Reg[i] = 'Y' Then
               Writeln(', Register output, Fback to inLine',Prec.Fback[i])
              Else
               If Prec.Feedback[i] = 'Y' Then
                Writeln(', Fback to inLine ',PRec.Fback[i])
               Else
                Writeln(', No feedback')
            End{if,output}
           Else
            Writeln
         End;{else}
      End;{for}
     Close(Pfile);
     Writeln('(Inline = Input line, Pline = Product line, Fback = Feedback, oe = output enable)');
     Readrecord;
     Write('Press enter to go on. ');
     Readln;
   End;{else}
```

```
 End; {ReadPALs}

Procedure EraseRecords;                                    {To erase the PAL database}
 Begin
   Assign(Pfile,'PALfile.pal');
   Writeln('All data in database wil be lost !!!');
   Quit := False;
   Repeat
     Write('Do you want to proceed Y/N ? ');
     Read(wipe);
     wipe := Upcase(wipe);
     If  wipe = 'Y' Then
      Begin
        Erase(Pfile);
        Quit := true;
      End
     Else
      If wipe = 'N' Then
       Quit := true;
   Until Quit
 End;{EraseRecords}


{*********************************************************************}
{Procedure to edit certain data in the database or delete all data}
{*********************************************************************}
Procedure EditRec;
 Begin
   PalSet := ['1','L','h','H','r','R'];
   Clrscr;
   Assign( PFile,'PalFile.PAL' );
  {$i-}
    Reset(PFile);
  {$i+}
   IF IoResult <> 0 THEN
    Writeln('No data of PALs yet')
   ELSE
    Begin
      Quit := False;
```

BYLAAG B
PAL_DATA.PAS

```
Repeat
  Write('Edit One record /0, or Erase All records /A ? ');
  Readln(Wipe);
  wipe := upcase(wipe);
Until (wipe = '0') or (wipe = 'A');
If wipe = 'A' Then
 Eraserecords                                          {Delete all Data}
Else
 Begin
   Clrscr;
   Writeln('Pals in database');
   Writeln;
   j := 0;
   n := 3;
   For i := 0 to RecNumber Do
    Begin
      GotoXY(j*18,n);
      Write('_ ');
      Write(i + 1,'. ',PalTypes[i]);
      Inc(j);
      If j = 4 Then
       Begin
         Writeln;
         j := 0;
         Inc(n);
       End;{if}
    End;{For}
   Writeln;
   Writeln;
   quit := False;
   Repeat
     Write('Enter digit Of PAL to be Edit : ');
     Readln(digit);
     For i := 1 To Recnumber + 1 do
      Begin
        If i = digit Then
         Quit := True;
      End;{for}
   Until Quit;
```

BYLAAG B
PAL_DATA.PAS

```
Clrscr;
Seek(pfile,(digit - 1));
Read(PFile,PRec);
Writeln('Data of PAL : ',PRec.Name);
Writeln('Number of pins : ',PRec.Count);
FOR i := 1 TO PRec.Count DO
 Begin
   Write('Pin ',i,' ', PRec.Status[i]);
   If PRec.Status[i] = 'INPUT' Then
    Begin
      Writeln(', connect to input line ',PRec.Input line[i]);
    End {If,input}
   Else
    Begin
      If PRec.Status[i] = 'OUTPUT' Then
       Begin
         Write(', 1st Pline ',PRec.Productline[i]);
         Write(', No.of Plines ',PRec.NoOf[i]);
         If Prec.Proden[i] = 'Y' Then
          Write(', Pline ',Prec.Productline[i] - 1,' = oe');
          If Prec.Reg[i] = 'Y' Then
           Writeln(', Register output, Fback to inLine',Prec.Fback[i])
          Else
           If Prec.Feedback[i] = 'Y' Then
            Writeln(', Fback to inLine ',PRec.Fback[i])
           Else
            Writeln(', No feedback')
       End{if,output}
      Else
       Writeln
    End;{else}
 End;{for}
Writeln;
Repeat
  Write('Edit --> N.ame, P.ins ,D.elete ? ');
  Readln(choice);
  choice := upcase(choice);
Until (choice = 'N') or (choice = 'P') or (choice = 'D');
If choice = 'D' Then                                    {Delete data of one PAL from database}
```

```
        Begin
          Seek(pfile,recnumber);
          Read(PFile,PRec);
          dec(recnumber);
        End{if}
       Else
        Begin
          If choice = 'N' Then                          {Edit name of PAL in database}
           Begin
             Write('PAL name ? : ');
             Readln(Prec.Name);
             Done := False;
             i := 1;
             Repeat
               If Prec.Name[i] in PalSet Then
                Begin
                  Prec.State := Upcase(Prec.Name[i]);
                  Done := true;
                End{If}
               Else
                Inc(i);
             Until Done;
           End{if}
          Else                                          {Edit data of one pin of a specific PAL in the database}
           Begin
             Write('Enter the pin number to be edit. ');
             Readln(pos);
             With prec do
              Begin
                Writeln;
                Write('Give new status of pin ',pos,' : ');
                Readln(Status[pos]);
                Xlate := Status[pos];
                For j := 1 to length(xlate) do
                 Begin
                   Xlate[j] := Upcase(Xlate[j]);
                 End;{for}
                Status[pos] := Xlate ;
                If Status[pos] = 'INPUT' Then
```

BYLAAG B
PAL_DATA.PAS

```
 Begin
   Productline[pos] := 99;
   Write('Input ',pos,' connect to which input line? : ');
   Readln( Inputline[pos] )
 End{if}
Else
 If Status[pos] = 'OUTPUT' Then
  Begin
    Inputline[pos] := 99;
    If State = 'R' Then
     Begin
       Done := False;
       Repeat
         Write('Is it a Register Output Y/N : ');
         Readln(Choice);
         Case Choice Of
          'Y','y','N','n' : Done := true;
         Else
          Write('Error : Try again. ');
         End;{case}
       Until Done;
       Reg[pos] := Upcase(choice);
       If Reg[pos] = 'Y' Then
        Proden[pos] := 'N'
       Else
        Begin
          Write('Is there a product line to enable output  Y or N ? ');
          Readln(Proden[pos]);
          Proden[pos] := upcase(Proden[pos]);
        End;{else}
     End{if}
    Else
     Begin
       Write('Is there a product line to enable output  Y or N ? ');
       Readln(Proden[pos]);
       Proden[pos] := upcase(Proden[pos]);
     End;{else}
    Write('First product line connect to output ',pos,' ? : ');
    Readln( Productline[pos] );
```

BYLAAG B
PAL_DATA.PAS

```
                Write('Number of Product lines connected to OR gate of output ',pos,' ? : ');
                Readln( NoOF[pos] );
                Done := False;
                WHILE NOT Done DO
                 Begin
                   Write('Output with feedback Y or N :');
                   Readln( FeedBack[pos] );
                   FeedBack[pos] := Upcase(FeedBack[pos]);
                   IF FeedBack[pos] = 'Y' THEN
                    Begin
                      Done := True;
                      Write(' Feedback to which input line? :');
                      Readln( Fback[pos] );
                    End{if}
                   Else
                    Begin
                      Done := True;
                      Fback[pos] := 0;
                    End;{else}
                  End;{while}
               End;{if}
            End;{with}
          End;{else}
        End;{else}
       Seek(pfile,digit - 1);
       Write(pfile,prec);
       Close(Pfile);
     End;{else}
  End;{else}
 Get;
End;{EditRec}
```

BYLAAG B
PAL_DATA.PAS

```
{************************************************************}
{Procedure to get the number and names of PALs in the database}
{************************************************************}
Procedure Get;
 Begin
   Assign( PFile,'PalFile.PAL' );
  {$i-}
    Reset(PFile);
  {$i+}
    IF IoResult <> 0 THEN
     Writeln('No data of PALs yet.')
    ELSE
     Begin
       RecNumber := FileSize(PFile) - 1;
       For i := 0 to RecNumber do
        Begin
          Seek(pfile,i);
          Read(PFile,PRec);
          PalTypes[i] := PRec.Name;
        End;{For}
       Close(pFile);
     End;{Else}
 End;{GET}

BEGIN                                          {Beginning of main program}
  Get;
  Clrscr;
  Writeln('PAL database.');
  Quits := False;
  Repeat                                       {Menu to select a topic}
    Writeln;
    Write( 'A.dd, R.ead, E.dit, Q.uit ? ');
    Readln( Choice );
    Case Choice OF
     'A','a' : AddRecords;
     'R','r' : ReadPALs;
     'E','e' : Editrec;
     'Q','q' : Quits := True;
    ELSE Write('Error : Try again. ')
```

```
    End; {Case}
  UNTIL Quits ;
END.
```

BYLAAG C
FUSEMAP.PAS

```
{************************************************************}
{The purpose of this program is to create a circuit database, }
{compile a fusemap, compile a JEDEC-file, display data in the }
{database and to edit data in the database                    }
{************************************************************}
Program Fusemap_JEDEC;

Uses
   Crt,                                                  {Pascal CRT unit}
   Dos;                                                  {Pascal Dos unit}

Const
     MaxNo = 20;                                         {Max number of pins and product lines}
     MaxPal = 30;                                        {Max number of Pal types }
     Printerx = 'PRN';                                   {Assign to printer}
     screen = '';                                        {Assign to screen}

TYPE

  DataPtr = ^DataRecord;                                 {Data record memory pointer}

  DataRecord = record                                    {Data record structure}
     Dataln      : String[70];                           {String to keep info}
     Next        : Dataptr;                              {Pointer to next record for linked list}
  End;{datarec}

  FuseRec =  Record                                      {Circuit record}
     Name        : String[8];                            {Name of circuit}
     Dig         : Integer ;                             {Digit of PAL type in PAL record}
     PinName     : Array[1..MaxNo] Of char;              {Array to store the pin names}
     Notname     : Array[1..Maxno] Of char ;             {Array to store active low or high}
     Counter     : Byte;                                 {Number of  outputs that are used}
     Out         : Array[1..MaxNo] Of Integer;           {Store the pin number of each output that is used}
     Expres      : Array[1..MaxNo] Of String[80];        {Store the function of each output}
     ExEnable    : Array[1..MaxNo] Of String[80];        {Store the function of each output enable line}
  End;{FuseRec}

  PalRec =  Record                                       {Pal record}
     Name        : String[5];                            {PAL name}
     State       : Char;                                 {States of PAL outputs}
     Count       : Byte;                                 {Number of pins}
     Status      : Array[1..MaxNo] OF String[6];         {Status of pins}
     Inputline   : ARRAY[1..MaxNo] OF Integer;           {Input pin connected to which input line}
     Proden      : array[1..MaxNo] Of Char;              {Output with a product enable line or not}
```

BYLAAG C
FUSEMAP.PAS

```
    Productline : ARRAY[1..MaxNo] OF Integer;          {The number of the first product line of the output}
    Reg         : Array[1..Maxno] Of Char;             {Output with register}
    NoOf        : ARRAY[1..MaxNo] OF Integer;          {Number of Product lines conected to OR gate of output pin}
    Feedback    : ARRAY[1..MaxNo] OF Char;             {Output with feedback}
    Fback       : ARRAY[1..MaxNo] OF Integer;          {Feedback to which input line}
  End;{PalRec}

VAR

  Firstline,
  Prevline,
  Currentline    : DataPtr;                            {Linked list pointers}
  Heaptop        : ^word;
  FRec           : FuseRec ;                           {Variable to structure of circuit data}
  Ffile          : FILE OF  FuseRec ;                  {Typed file for circuit data}
  FXfile         : File of fuseRec;                    {Typed file to store circuit data teporary}
  PRec           : PalRec;                             {Variable to structure of PAL data}
  PFile          : FILE OF  PalRec;                    {Typed file for PAL data}
  JFile          : Text;                               {Text file for JEDEC file}
  PalTypes       : Array[0..MaxPal] of String[5];      {Existing PALs in PAL data base}
  Product        : Array[1..8] of String[32];          {Temporary storage of compiled data for one output}
  Valid          : Array[1..24] of integer;            {Non existing input used in expression}
  Templine       : Array[0..16] of integer;            {Used input lines}
  PLine          : Array[0..16] of integer;            {Phantom input lines}
  Circuit        : Array[0..10] of string[8];          {Circuit names in circuit database}
  Script         : string[80];                         {Output to screen}
  Dname          : string[20];                         {Name of designer to be saved in JEDEC file}
  Ydig           : string[2];
  Lineone        : string[70];                         {One product line from JEDEC to memory}
  Oneline        : String[48];                         {One product line to screen}
  FN,LFN         : string[7];                          {First fuse number of pruduct line}
  Ones           : String[1];                          {Change one hex number for checksum}
  Checksum       : String[4];                          {Checksum in HEX format for JEDEC file}
  Pname          : string[2];                          {Pin name}
  ochr           : string[2];                          {Line number in memory}
  Temp           : string[6];                          {Status of PAL pin}
  TempEx         : string[80];                         {Temporary storage of function for one output}
  TempProd       : String[32];                         {Temporary input lines for one product lines}
  Hardcop,       : Char;                               {Hardcopy of fusemap}
  EN             : Char;                               {ENable}
  Makejed        : Char;                               {Compile jedec file}
  NoX,First,wipe,
  Xdig,Choice    : Char;                               {Misc variables for general use}
  Okay,Quits,Done,
```

BYLAAG C
FUSEMAP.PAS

```
  Quit,Nvalid    : Boolean;
  PalSet         : Set of Char;                {Type of PAL output -> H,L or R}
  CharSet        : Set Of Char;                {Alphabet to ensure correct pin name}
  RecNumber      : Integer;                    {Number of PALs in PAL database}
  Hex,Checks,one,
  rem,res        : Integer;                    {Variables to determine Checksum}
  templ          : Integer;                    {Number of inputlines}
  phanline       : Integer;                    {Number of phantom input lines}
  phan           : Integer;                    {Number of phantom product lines at a specific output}
  Code           : Integer;                    {Variable to show if conversion is successful}
  cirnumber      : Integer;                    {Number  of circuits in circuit database}
  Fusenumber     : Integer;
  i,j,n,a,m,k,l,
  p,digit,pos,
  dig,i2,Dont,
  Terms,Xinsert,
  PCount,PNumber : Integer;                    {Misc variables for general use}
  csum           : Longint;                    {Checksum in decimal}


{**********************}
{Procedure to save data}
{**********************}
PROCEDURE WriteFData;
 Begin
   Assign(Ffile,'Cirfile.cir');
  {$i-}
   Reset(Ffile);
  {$i+}
   If (IoResult <> 0) Then
    Rewrite(Ffile) ;
   Seek(Ffile,fileSize(Ffile)) ;
   Write(Ffile,Frec) ;
   Close(Ffile) ;
 End;

Procedure check;forward;                       {Procedure will follow}
Procedure Getx;forward;                        {Procedure will follow}
```

BYLAAG C
FUSEMAP.PAS

```
{*********************************}
{Procedure to add data to database}
{*********************************}
PROCEDURE ExpresInfo;
 Begin
   Clrscr;                                          {Clear screen}
   Pname := '';
   CharSet := ['a'..'z','A'..'Z',' ','!'];
   Writeln('Generation of fuse map.');
   Writeln;
   Write('Enter the name of project or circuit : ');
   Readln(Frec.name);                               {Get circuit name}
   Clrscr;
   Writeln('Chose one of the folowing PALs ') ;
   Writeln;
   j := 0;
   n := 3;
   For i := 0 to RecNumber Do                       {Write names of PALs in PAL database to screen}
    Begin
      GotoXY(j*18,n);
      Write('_ ');
      Write(i + 1,'. ',PalTypes[i]);
      Inc(j);
      If j = 4 Then
       Begin
         Writeln;
         j := 0;    {j was :=1}
         Inc(n);
       End;{if}
    End;{For}
   Quit := False;
   Writeln;
   Writeln;
   Repeat
     Write('Enter digit of Pal to be used : ');
     Readln(digit);
     For i := 1 To Recnumber + 1 do                 {Select PAL}
      If i = digit Then
     Quit := True;
   Until Quit;
   Assign( Pfile,'PalFile.PAL' ) ;                  {Open PAL file}
  {$i-}
   Reset(Pfile) ;
  {$i+}
```

BYLAAG C
FUSEMAP.PAS

```
If IoResult <> 0 Then
 Writeln('No record of PAL  try again.')
Else
 Begin
   Seek(pfile,(digit - 1));
   Read(PFile,Prec) ;                                  {Get data of selected PAL}
   Close(Pfile);                                       {Close PAL file}
   Frec.Dig := (digit - 1);
   Writeln;
   Writeln('Name pins to be used, eg. pin 1 Input = x ') ;
   Writeln('Use alphabet or press space bar for no entry.');
   Writeln;
   k := 0;
  For i := 1 to  Prec.Count do                         {Get PAL pin names}
    Begin
      If (PRec.Status[i] = 'INPUT') Then
       Repeat
         Write('Pin ',i,' : ',PRec.Status[i],' = ') ;
         XPname := readkey;
         If XPname = '!' Then
          Begin
            Frec.notName[i] := '!';
            Writeln(xpname);
            Xpname := readkey;
            Frec.PinName[i] := xPname;
          End{if}
         Else
          Begin
            Writeln(xpname);
            FRec.PinName[i] := xPname;
            frec.notName[i] := '*';
          End;
         If Not (Frec.PinName[i] In CharSet) Then
          Begin
            Write('Sorry Try again ? ,');
            Writeln(' Use alphabet or press space bar for no entry.');
            Writeln;
          End;{if}
       Until Frec.PinName[i] In CharSet
      Else
       Begin
         If (PRec.Status[i] = 'OUTPUT') THEN
          Begin
            Repeat
```

BYLAAG C
FUSEMAP.PAS

```
              Write('Pin ',i,' : ',PRec.Status[i],' = ') ;
              xPname := readkey;
              If xPname = '!' Then
               Begin
                 Frec.notName[i] := '!';
                 write(xpname);
                 Xpname := readkey;
                 Frec.PinName[i] := xPname;
                 writeln(xpname);
               End{if}
              Else
               Begin
                 FRec.PinName[i] := xPname;
                 writeln(xpname);
                 Frec.NotName[i] := '*';
              End;{else}
              If Not (Frec.PinName[i] In CharSet) Then
               Begin
                 Write('Sorry Try again ? ,');
                 Writeln(' Use alphabeth or press space bar for no entry.');
                 Writeln;
               End;{if}
            Until Frec.PinName[i] In CharSet;
            If Not (FRec.PinName[i] = ' ') Then
             Begin
               Inc(k) ;
               Frec.Out[k] := i;
               Frec.Counter := k;
             End;{if}
          End{if}
         Else
          Writeln('Pin ',i,' : ',PRec.status[i]);
       End;{else}
      Delete(pname,1,2);
    End;{for}
 End;{else}
For k := 1 to Frec.Counter Do                                   {Get Boolean expression for PAL outputs}
 Begin
   For i := 1 to Prec.Count Do
    Begin
      If i = Frec.Out[k]  Then
       Begin
         if Prec.proden[i] = 'Y' then
          Begin
```

```
            Script := 'Enter Boolean expression for output enable of pin no ';
            EN := 'y';
            check;
            a := 0;
            Repeat
              Inc(a);
            until (TempEx[a] = '0') or (a = length(tempex));
            If tempex[a] <> '0' Then
             Begin
               Script := 'Enter Boolean expression for output pin no ';
               EN := 'n';
               check;
             End;{if}
            EN := 'n';
           End{if}
          Else
           Begin
             Script := 'Enter Boolean expression for output pin no ';
             EN := 'n';
             check;
           End;{else}
        End;{if}
      End;{for}
   End;{for}
  If Frec.counter = 0 Then
   Write('Error ? : No output defined, try again no info will be saved.')
  Else
   Begin
     Writefdata;
     Getx;
   End;{else}
End;{ExpresInfo}

{*****************************************}
{Procedure to Get expression of an output and}
{to check if data in expressions is valid    }
{*****************************************}
Procedure Check ;
 Begin
   Terms := 0;
   Done := false;
   n := 0;
   NValid := True;
   a := 0;
```

BYLAAG C
FUSEMAP.PAS

```
m := 0;
Repeat
  l := 0;
  Writeln(Script,i) ;
  Readln(TempEx);                                          {Get expression}
  Repeat
    Inc(a);
  Until (TempEx[a] = '=') or (a = length(tempex));
  If a = length(tempex) then
   Begin
     Writeln('No equal(=) sign in term, try again');
     NValid := False;
   End{if}
  Else
   Begin
     For j := 1 To (a - 1) Do
      Begin
        If (TempEx[j] <> ' ') Then
         Begin
           If (Frec.NotName[i] <> '!') Then
            Begin
              If (Tempex[j] = '!') Then
               Begin
                 If (tempex[j+1] <> ' ') Then
                  Begin
                    Writeln(TempEx[j],Tempex[j+1],' not defined at this output. Try again.');
                    NValid := False;
                  End{if}
                 Else
                  Begin
                    Writeln(TempEx[j],' not defined at this output. Try again.');
                    NValid := False;
                   End;{else}
               End{if}
              Else
               Begin
                 If (TempEx[j] <> Frec.PinName[i]) Then
                  Begin
                    Writeln(TempEx[j],' not defined at this output. Try again.');
                    NValid := False;
                  End;{If}
               End;{else}
            End{if}
           Else
```

BYLAAG C
FUSEMAP.PAS

```
            Begin
              Inc(l);
              Insert(Tempex[j],pname,l);
              Inc(l);
              Insert(Tempex[j+1],pname,l);
              If (Pname[1] <> Frec.NotName[i]) AND (Pname[2] <> Frec.pinName[i]) Then
               Begin
                 Writeln(Pname,' not defined at this output. Try again.');
                 NValid := False;
               End;{if}
            End{else};
         End;{if}
      End;{For}
   End;{else}
  If NValid = True Then
   Begin
     For j := (a + 1) To Length(TempEx) Do
      Begin
        If TempEx[j] = '+' Then
         Inc(Terms)
        Else
         Begin
           If (TempEx[j] <> ' ')  Then
            Begin
              If (TempEx[j] <> '!')  Then
               Begin
                 If EN = 'y' Then
                  Begin
                    If TempEx[j] <> '1' Then
                     Begin
                       If TempEx[j] <> '0' Then
                        Begin
                          m := 0;
                          Repeat
                            Inc(m);
                          Until (TempEx[j] = Frec.PinName[m])  or  (m >= Prec.Count);
                        End;{if}
                     End;{if}
                  End{if}
                 Else
                  Begin
                    m := 0;
                    Repeat
                      Inc(m);
```

```
                Until (TempEx[j] = Frec.PinName[m])  or  (m >= Prec.Count);
               End;{else}
              If m >= Prec.Count Then
               Begin
                 NValid := False;
                 inc(n);
                 Valid[n] := j;
                 m := 0;
               End;{if}
            End{if};
         End;{if}
      End;{else}
   End;{for}
a := 0;
If (Terms < Prec.NoOf[i]) and (NValid = true) Then
 Begin
   If EN = 'y' Then
    Begin
      If Terms > 0 Then
       Begin
         Terms := 0;
         Writeln('Product terms cant be used for output enable, try again. ');
       End{if}
      Else
       Begin
         Frec.ExEnable[i] := TempEx;
         Done := True;
       End;{else}
    End{if}
   Else
    Begin
      Frec.Expres[i] := TempEx;
      Done := True;
    End;{else}
 End{if}
Else
 Begin
   If (Terms >= Prec.NoOf[i]) Then
    Begin
      Writeln('To many product terms ! , Try again .') ;
      Terms := 0 ;
    End{if}
   Else
    Begin
```

```
            For m := 1 to n do
             Begin
               Writeln('Not valid no input for ',TempEx[valid[m]]);
             End;{for}
           End;{else}
         Nvalid := true;
         n := 0;
         Terms := 0;
       End;{else}
     End{if}
    Else
     Begin
       NValid := True;
       a := 0;
     End;{else}
  Until Done ;
End;{procedure}

PROCEDURE PrintRecord;                                    {To print data}
 Begin
   Assign(OUTPUT,Printerx);
   Rewrite(OUTPUT);
 End;

PROCEDURE  ReadRecord;                                    {To display data on screen}
 Begin
   Assign(OUTPUT,Screen);
   Rewrite(OUTPUT);
 End;

{*****************************************}
{Procedure to read data from circuit database}
{and PAL database and store in records       }
{*****************************************}
Procedure OpenRecs;
 Begin
   Assign( Ffile,'CirFile.Cir' ) ;
  {$i-}
   Reset(Ffile) ;                                         {Open File with circuit info}
  {$i+}
   If IoResult <> 0 Then
    Begin
      Writeln('No record of Fuse map  try again.');
      Okay := False;
```

```
   End
  Else
   Begin
     Okay := True;
     Writeln('Data of next circuits in database ');
     Writeln;
     j := 0;
     n := 3;
     For i := 0 to cirnumber do                          {Write names of all the existing circuits}
      Begin                                              {in the data base to the screen}
        GotoXY(j*18,n);
        Write('¦');
        Write(i + 1,'. ',circuit[i]);
        inc(j);
        If j = 4 Then
         Begin
           writeln;
           j := 0; {j was := 1}
           Inc(n);
         End;{if}
       End;{for}
      Writeln;
      Writeln;
      quit := False;
      Repeat
        Write('Enter the digit of circuit you want to work with : ');
        Readln(dig);                                     {Select a circuit}
        For i := 1 To cirnumber + 1 do
         If i = dig Then
          Quit := True;
      Until Quit;
      Seek(Ffile,(dig - 1));
      Read(Ffile,Frec);                                  {Read data of specific circuit to memory}
      Digit := Frec.Dig;                                 {Number of PAL in PAL data base}
      Assign( Pfile,'PalFile.Pal' ) ;
     {$i-}
       Reset(Pfile);                                     {Open PAL file}
     {$i+}
      Seek(pfile,digit);
      Read(Pfile,Prec) ;                                 {Read data of PAL to memory}
      Close(ffile);                                      {Close Circuit file}
      close(pfile);                                      {Close PAL file}
   End;{else}
End;{openrecs}
```

BYLAAG C
FUSEMAP.PAS

```
{*************************************************************************}
{Procedure to display data of circuit on screen or to make a printout thereof}
{*************************************************************************}
PROCEDURE ReadExpresInfo;
 Begin
   Clrscr;
   openrecs;
   Writeln;
   If okay then
    Begin
      Quit := False;
      Repeat
        Write('Output to S.creen, P.rinter ? ');
        Readln(Choice);
        Case Choice Of
          'P','p' : Begin
                      Printrecord;
                      Quit := True;
                    End;
          'S','s' : Begin
                      Readrecord;
                      Quit := True;
                    End;
        End;{case}
      Until Quit;
      Clrscr;
      Writeln('Summary of ',Frec.name,' making use of ',Prec.Name);
      Writeln ;
      j := 1 ;
      n := 3 ;
      For i := 1 to PRec.Count DO
       Begin
         If (PRec.Status[i] = 'INPUT') or (PRec.Status[i] = 'OUTPUT') THEN
          Okay := True
         Else
          Okay := False ;
         If Okay Then
          Begin
            If (choice = 'S') or (choice = 's') Then
             Begin
               If frec.Notname[i] = '!' Then
                Write('Pin ',i,' : ',PRec.Status[i],' = ',Frec.NotName[i],FRec.PinName[i])
               Else
                Write('Pin ',i,' : ',PRec.Status[i],' = ',FRec.PinName[i]);
```

BYLAAG C
FUSEMAP.PAS

```
          GotoXY(j*25,n) ;
           Write('_ ');
           j := j + 1;
         End{if}
        Else
         If frec.Notname[i] = '!' Then
          Writeln('Pin ',i,' : ',PRec.Status[i],' = ',Frec.NotName[i],FRec.PinName[i])
         Else
          Writeln('Pin ',i,' : ',PRec.Status[i],' = ',FRec.PinName[i])
      End {if}
     Else
      Begin
        If (choice = 'S') or (choice = 's') then
         Begin
           Write('Pin ',i,' : ',PRec.status[i]) ;
           GotoXY(j*25,n) ;
           Write('_ ');
           j := j + 1 ;
         End{if}
        Else
         Writeln('Pin ',i,' : ',PRec.status[i]) ;
      End;{Else}
     If j = 4 Then
      Begin
        Writeln;
        Inc(n);
        j := 1;
      End;{if}
  End;{for}
 Writeln ;
 Writeln ;
 For k := 1 to Frec.Counter Do
  Begin
    For i := 1 To Prec.Count Do
     Begin
       If i = Frec.Out[k] Then
        Begin
          Writeln('Pin ',i);
          If Prec.ProdEn[i] = 'Y' Then
           Begin
             Writeln('Output enable :',Frec.ExEnable[i]);
           End;{if}
          Writeln('Expression    : ',FRec.expres[i]);
        End;{if}
```

BYLAAG C
FUSEMAP.PAS

```
          End;{for}
        End;{for}
      End;{if}
    Readrecord;
    Write('Press enter to go on. ');
    Readln;
  End;{ReadExpresInfo}

  PROCEDURE Dontcare; Forward;                    {Procedures to folow}
  Procedure Phantom; Forward;
  Procedure Find; Forward;
  Procedure Booleanx; Forward;
  Procedure FindIn;Forward;
  Procedure FusesBlown;Forward;
  Procedure Jedec;Forward;
  Procedure Bin;Forward;
  Procedure Jedmake;Forward;

{*********************************}
{Procedure to store data in memory}
{*********************************}
  Procedure Point;
   Begin
     If Firstline = Nil Then
      Begin
        New(Currentline);
        Currentline^.Dataln := lineOne;
        Currentline^.Next := Nil;
        Firstline := Currentline;
      End{if}
     Else
      Begin
        Prevline := Currentline;
        New(Currentline);
        Currentline^.Dataln := lineOne;
        Prevline^.Next := Currentline;
        Currentline^.Next := Nil;
      End;{else}
   End;{point}
```

BYLAAG C
FUSEMAP.PAS

```
{*****************************************}
{Procedure to display data in memory on screen}
{*****************************************}
Procedure Readmap;

 Var
  Data : DataPtr;

 Begin
   j := 0;
   Data := Firstline;
   Writeln('                    1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2 3 3');
   Writeln('    0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1');
   Writeln;
   While Data <> Nil Do
    Begin
      With Data^ Do
      Writeln(Dataln);
      Data := Data^.Next;
      Inc(j);
      If j = 8 then
       Writeln;
      If j = 16 Then
       Begin
         Writeln;
         Writeln('                    1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2 3 3');
         Writeln('    0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1');
         If hardcop = 'N' Then
          Readln
         Else
          Writeln;
          j := 0;
       End;{if}
    End;{while}
 End;{readmap}

{************************}
{Procedure to make fusemap}
{************************}
PROCEDURE MakeMap ;
 Begin
   Clrscr;
   Openrecs;
   If Okay Then
```

BYLAAG C
FUSEMAP.PAS

```
Begin
  Clrscr;
  hardcop := 'N';
  Firstline := nil;
  mark(heaptop);
  j := 0;
  Temp := '';
  FindIn;
  Templ := 0;
  While j < Prec.Count Do                                  {Determine which input lines exist in pal}
   Begin
     If Temp = 'INPUT' Then
      Begin
        Templine[Templ] := Prec.Inputline[j];
        Inc(Templ);
      End{if}
     Else
      Begin
        If Prec.Feedback[j] = 'Y' Then
         Begin
           Templine[Templ] := Prec.Fback[j];
           Inc(Templ);
         End;{if}
      End;{else}
     FindIn;
   End;{while}
  Phanline := 0;
  j := 0;
  Repeat                                                   {Determine phantom input lines}
    i := 0;
    Done := false;
    Repeat
      If j = Templine[i] then
      Done := true
      Else
       Begin
         If i = (Templ - 1) Then
          Begin
            Pline[phanline] := j;
            Inc(Phanline);
            Done := true;
          End{if}
         Else
          Inc (i);
```

BYLAAG C
FUSEMAP.PAS

```
        End;{else}
      Until Done;
      j := j + 2;
    Until j = 32;
    i := (Prec.Count + 1) ;
    Find;
    If n > 7 Then
     Begin
       m := 0;
       Phantom;
     End;{if}
    While i > 1 do
     Begin
       If FRec.PinName[i] = ' ' Then
        Begin
          Writeln;
          Dontcare;
          Find;
        End{if}
       Else
        Begin
          If Prec.Proden[i] = 'Y' Then
           Begin
             k := 1;
             TempEx := Frec.ExEnable[i];
             Booleanx;
             a := 0;
             Repeat
               Inc(a);
             Until (TempEx[a] = '0') or (a = length(tempex));
             If tempex[a] <> '0' Then
              Begin
                Inc(k);
                TempEx := Frec.Expres[i];
                Booleanx;
              End;{if}
           End{if}
          Else
           Begin
             k := 1;
             TempEx := Frec. Expres[i];
             Booleanx;
           End;{else}
          FusesBlown;
```

BYLAAG C
FUSEMAP.PAS

```
          Find;
        End;{else}
      End;{while}
     Repeat
       Inc(i);
       Temp := prec.status[i];
     Until (Temp = 'OUTPUT') ;
     j := Prec.productline[i];
     If Prec.Proden[i] = 'Y' Then
      m := j + 7
     Else
      m := j + 8 ;
     If Prec.Count = 20 Then
      n := (64 - m)
     Else
      n := (80 - m);
     If n <> 0 Then
      Phantom;
      Release(heaptop);
      Readmap;
      Write('Do you want a hardcopy, Y/N ?');
      Readln(hardcop);
     If Upcase(hardcop) = 'Y' Then
      Begin
        Printrecord;
        Readmap;
        Readrecord;
      End;{if}
     Write('Make jedecfile, Y/N ? ');
     Readln(Makejed);
    If Upcase(Makejed) = 'Y' Then
     Begin
       Jedec;
       Write('Jedec File has been made.');
     End{if}
    Else
     Writeln('No jedec file was made.');
End;{MakeMap}
```

BYLAAG C
FUSEMAP.PAS

```
{***************************}
{Procedure to make JEDEC file}
{***************************}
Procedure Jedec;
 Begin
   Write('Enter your name. ');
   readln (Dname);
   First := 'Y';
   PNumber := 0;
   Pcount := 0;
   i := Prec.Count;
   Assign(Jfile,FRec.name+'.JED');
   Rewrite(Jfile);
   Writeln(jFile,Char($02));
   Writeln(jfile,'PAL used : ',Prec.Name);
   Writeln(jfile,'Projek name : ',Frec.name);
   Writeln(jfile,'Designed by : ',Dname);
   Writeln(Jfile,'*');
   Writeln(Jfile,'QP',prec.count,'*');
   Writeln(Jfile,'QV*');
   Writeln(Jfile,'G0*');
   Writeln(Jfile,'F0*');
   find;
   If Prec.Proden[i] = 'Y' Then
    PNumber := (n - 1)
   Else
    PNumber := n;
   Repeat
     If Frec.PinName[i] <> ' ' Then
      Begin
        If Prec.Proden[i] = 'Y' Then
         Begin
           k := 1;
           TempEx := Frec.ExEnable[i];
           Bin;
           Inc(k);
           a := 0;
           Repeat
             Inc(a);
           Until (TempEx[a] = '0') or (a = length(tempex));
           If tempex[a] <> '0' Then
            Begin
              TempEx := Frec.Expres[i];
              Bin;
```

BYLAAG C
FUSEMAP.PAS

```
            End;{if}
          End{if}
        Else
         Begin
           k := 1;
           TempEx := Frec. Expres[i];
           Bin;
         End;{else}
        Jedmake;
      End;{if}
     If Prec.Proden[i] = 'Y' Then
      Phan := (7 - Prec.No0f[i])
     Else
      Phan := (8 - Prec.No0f[i]);
     PNumber := (Pnumber + Phan);
     PCount := (PCount + ((PHanline*2)*Phan));
     find;
  Until i = 1;
  j := 4;
  Checksum := '0000';
  If csum >= 16 Then
   Begin
     Repeat
       Rem := csum Mod 16;
       If Rem > 9 Then
        ones := Char(rem + 55)
       Else
        ones := Char(rem + 48);
       Delete(checksum,j,1);
       Insert(ones,Checksum,j);
       Dec(j);
       csum := csum Div 16
     Until csum < 16;
   End;{if}
  If csum > 9 Then
   ones := Char(csum + 55)
  Else
   Ones := Char(csum + 48);
  Delete(Checksum,j,1);
  Insert(ones,Checksum,j);
  Write(Jfile,'C',Checksum,'*');
  Write(Jfile,char($03));
  Close(Jfile);
End;{Jedec}
```

BYLAAG C
FUSEMAP.PAS

```
{*********************************************************}
{Procedure to change expression over to '1' and '0' format}
{*********************************************************}
Procedure Bin;
 Begin
   j := 0 ;
   NoX := 'n';
   m := 0;
   Tempprod := '11111111111111111111111111111111';
   Repeat
     Inc(m);
   Until Tempex[m] = '=';
   For m := (m + 1)  To Length(TempEx) Do
    Begin
      If (TempEx[m] <> ' ') Then
       Begin
         If (TempEx[m] <> '+') Then
          Begin
            If TempEx[m] = '!' Then
             NoX := 'y'
            Else
             Begin
               j  := 0;
               Repeat
                 Temp := ' ' ;
                 FindIn;
               Until  ( TempEx[m] = Frec.PinName[j]) or (TempEx[m] = '1') or (TempEx[m] = '0');
               If (TempEx[m] = '1') or (TempEx[m] = '0') Then
                Begin
                  If (TempEx[m] = '1') Then
                   TempProd := '11111111111111111111111111111111';
                  Else
                   TempProd := '00000000000000000000000000000000';
                End{if}
               Else
                Begin
                  Tprod := 0;
                  If Temp = 'INPUT' Then
                   Tprod := Prec.Inputline[j]
                  Else
                   Tprod := Prec.Fback[j];
                  If frec.notname[j] = '*' then
                   Begin
                     If NoX = 'y' Then
```

BYLAAG C
FUSEMAP.PAS

```
                    Begin
                      Inc(Tprod);
                      NoX := 'n';
                    End;{If}
                   Inc(Tprod);
                   TempProd[Tprod] := '0' ;
                 End{if}
                Else
                 Begin
                   If nox = 'n' then
                    Inc(tprod);
                   Inc(tprod);
                   nox := 'n';
                   TempProd[Tprod] := '0' ;
                 End;{else}
              End;{else}
           End;{else}
         End{if}
        Else
         Begin
           Product[k] := TempProd;
           Inc(k);
           TempProd := '11111111111111111111111111111111';
         End;{else}
      End{if}
   End;{for}
  Product[k] := TempProd;
  TempProd := '11111111111111111111111111111111';
  m := k ;
End{Procedure Bin};

{*******************************}
{Procedure to finalise JEDEC file}
{*******************************}
Procedure Jedmake;
 Begin
   If Prec.Proden[i] = 'Y' Then
    Begin
      Dec(n);
      Fusenumber := ((n*32) - (PNumber*32)) - Pcount;
    End{if}
   Else
    FuseNumber := ((n*32) - (PNumber*32)) - Pcount;
   For k := 1 To m  Do
```

BYLAAG C
FUSEMAP.PAS

```
Begin
  LFN := 'L000000';
  TempProd := Product[k];
  For l := 0 to (Phanline - 1) do
   Begin
     a := (Pline[l] + 1) - (l*2);
     Delete(Tempprod,a,2);
   End;{for}
  If first = 'Y' Then
   Begin
     j := 0;
     First := 'N';
     Repeat
       Inc(j)
     Until (TemPprod[j] = '1') or (j = (length(tempProd) +1));
     If j = (length(tempprod) + 1) Then
      First := 'Y'
     Else
      Begin
        If j > 4 Then
         Rem := j Mod 4
        Else
         Rem := j;
        res := 4 - Rem;
        Hex := 1;
        For p := 1 to Rem do
         Begin
           Hex := Hex*2;
         End;{for}
        Hex := Hex div 2 ;
        Csum := Hex;
        For p := 1 to Res do
         Begin
           Hex := Hex*2;
           Inc(j);
           Ones := copy(tempprod,j,1);
           Val(ones,one,code);
           Checks := Hex*One;
           Csum := Csum + Checks;
         End;{for}
        Inc(j);
        Hex := Hex*2;
      End;{else}
   End{if}
```

BYLAAG C
FUSEMAP.PAS

```
      Else
       j := 1;
      If First = 'N' Then
       Begin
         For j := j To Length(Tempprod) do
          Begin
            Ones := copy(tempprod,j,1);
            Val(ones,one,code);
            Checks := Hex*One;
            Hex := Hex*2;
            Csum := Csum + Checks;
            If Hex = 256 Then
             Hex := 1;
          End;{for}
       End;{if}
      Str(FuseNumber,FN);
      Xinsert := 8 - Length(FN);
      Insert(FN,LFN,Xinsert);
      Write(Jfile,LFN);
      Write(Jfile,' ');
      For j := 1 to Length(TempProd) Do
       Begin
         Write(JFile,TempProd[j]);
         If (j = 4) or (j = 8) or (j = 12) or  (j = 16) or (j = 20) or (j = 24) or (j = 28) Then
          Write(Jfile,' ');
       End;{For}
      Writeln(Jfile,'*');
      Inc(n);
      a := (Phanline*2*k);
      FuseNumber := (((n*32) - (PNumber*32)) - Pcount) - a;
    End;{For}
 End;{Jedmake}

{*****************************************************************}
{Procedure to change the expression of a output to fusemap format}
{*****************************************************************}
Procedure Booleanx;
 Begin
   j := 0 ;
   NoX := 'n';
   m := 0;
   Tempprod := '--------------------------------';
   Repeat
```

BYLAAG C
FUSEMAP.PAS

```
  Inc(m);
Until Tempex[m] = '=';
For m := (m + 1)  To Length(TempEx) Do
 Begin
   If (TempEx[m] <> ' ') Then
    Begin
      If (TempEx[m] <> '+') Then
       Begin
         If TempEx[m] = '!' Then
          NoX := 'y'
        Else
         Begin
           j  := 0;
           Repeat
             Temp := ' ' ;
             FindIn;
           Until  ( TempEx[m] = Frec.PinName[j]) or (TempEx[m] = '1') or (TempEx[m] = '0');
           If (TempEx[m] = '1') or (TempEx[m] = '0') Then
            Begin
              If (TempEx[m] = '1') Then
               TempProd := '------------------------------'
              Else
               TempProd := '******************************';
            End{if}
           Else
            Begin
              Tprod := 0;
              If Temp = 'INPUT' Then
               Tprod := Prec.Inputline[j]
              Else
               Tprod := Prec.Fback[j];
              If frec.notname[j] = '*' then
               Begin
                 If NoX = 'y' Then
                  Begin
                    Inc(Tprod);
                    NoX := 'n';
                  End;{If}
                 Inc(Tprod);
                 TempProd[Tprod] := '*' ;
               End{if}
              Else
               Begin
                 If nox = 'n' then
```

BYLAAG C
FUSEMAP.PAS

```
                    Inc(tprod);
                   Inc(tprod}
                   nox := 'n';
                   Tempprod[tprod] := '*';
                 End;{else}
               End;{else}
             End;{else}
           End{if}
         Else
          Begin
            Product[k] := TempProd;
            Inc(k);
            TempProd := '--------------------------------';
          End;{else}
       End;{if}
    End;{for}
   Product[k] := TempProd;
   TempProd := '--------------------------------';
   m := k ;
 End{Booleanx};

{*********************************************************}
{Procedure to finalise fusemap format for all product lines}
{for a output and store it in memory                      }
{*********************************************************}
Procedure FusesBlown;
 Begin
   If Prec.Proden[i] = 'Y' Then
    Begin
      Dec(n);
      Dont := ((Prec.NoOf[i] + 1) - m);
    End{if}
   Else
    Dont := (Prec.NoOf[i] - m);
   For k := 1 To m  Do
    Begin
      lineone := Product[k];
      For i2 := 1 to phanline do
       Begin
         lineone[pline[i2-1]+1] := ' ';
         lineone[pline[i2-1]+2] := ' ';
       End;{for}
      Str(n,ochr);
      If n < 10 Then
```

BYLAAG C
FUSEMAP.PAS

```
      Insert(' ',ochr,2);
     For i2 := 1 to 31 do
      Insert(' ',lineone,i2*2);
     Insert('  ',lineone,1);
     Insert(ochr,lineOne,1);
     Point;
     Inc(n);
  End;{For}
 For j := 1 To Dont Do
  Begin
    LineOne := '++++++++++++++++++++++++++++++++';
    Str(n,ochr);
    For i2 := 1 to phanline do
     Begin
       lineone[pline[i2-1]+1] := ' ';
       lineone[pline[i2-1]+2] := ' ';
     End;{for}
    If n < 10 then
     Insert(' ',ochr,2);
    For i2 := 1 to 31 do
     Insert(' ',lineone,i2*2);
    Insert('  ',lineone,1);
    Insert(ochr,lineOne,1);
    Point;
    Inc(n);
  End;{for}
 If Prec.Proden[i] = 'Y' Then
  m := (7 - Prec.NoOf[i])
 Else
  m := (8 - Prec.NoOf[i]);
 For j := 1 To m Do
  Begin
    lineOne :=  '                                ';
    Str(n,ochr);
    For i2 := 1 to phanline do
     Begin
       lineone[pline[i2-1]+1] := ' ';
       lineone[pline[i2-1]+2] := ' ';
     End;{for}
    If n < 10  then
     Insert(' ',ochr,2);
    For i2 := 1 to 31 do
     Insert(' ',lineone,i2*2);
    Insert('  ',lineone,1);
```

```
      Insert(ochr,lineOne,1);
      Point;
      Inc(n);
    End;{for}
 End{FusesBlown};

{***********************************}
{Procedure to find inputs and outputs}
{***********************************}
Procedure FindIn;
 Begin
   Repeat
     Inc(j);
     Temp := Prec.Status[j];
   Until (Temp = 'INPUT') or (Temp = 'OUTPUT') or (j >= Prec.Count) ;
 End;{findin}

{*************************}
{Procedure to find outputs}
{*************************}
Procedure Find;
 Begin
   Repeat
     Dec(i);
     Temp := prec.status[i];
   Until (Temp = 'OUTPUT') or (i = 1);
   Temp := '      ';
   n := Prec.Productline[i];
 End;

{*********************************************************************}
{Procedure to put correct character in fusemap for all Phantom outputs}
{*********************************************************************}
Procedure Phantom;
 Begin
   For j := 0 to (n - 1) do
    Begin
      lineOne := '                                ';
      str(m,ochr);
      For i2 := 1 to phanline do
       Begin
         lineone[pline[i2-1]+1] := ' ';
         lineone[pline[i2-1]+2] := ' ';
       End;{for}
```

BYLAAG C
FUSEMAP.PAS

```
      If m < 10 Then
       Insert(' ',ochr,2);
      For i2 := 1 to 31 do
       Insert(' ',lineone,i2*2);
      Insert('  ',lineone,1);
      Insert(ochr,lineOne,1);
      Point;
      Inc(m);
    End;{for}
 End;

{*************************************************}
{Procedure to put corect character in fusemap for all}
{outputs that are not used in application          }
{*************************************************}
Procedure dontcare;
 Begin
   If Prec.proden[i] = 'Y' Then
    Begin
      dec(n);
      m := (7 - Prec.NoOf[i]);
      For j := 1 To (Prec.NoOf[i] + 1) Do
       Begin
         LineOne := '++++++++++++++++++++++++++++++++';
         str(n,ochr);
         For i2 := 1 to phanline do
          Begin
            lineone[pline[i2-1]+1] := ' ';
            lineone[pline[i2-1]+2] := ' ';
          End;{for}
         If n < 11 Then
          Insert(' ',ochr,2);
         For i2 := 1 to 31 do
          Insert(' ',lineone,i2*2);
         Insert('  ',lineone,1);
         Insert(ochr,lineOne,1);
         Point;
         Inc(n);
       End;
    End{if}
   Else
    Begin
      m := (8 - Prec.NoOf[i]) ;
      For j := 1 To Prec.NoOf[i] Do
```

BYLAAG C
FUSEMAP.PAS

```
      Begin
        LineOne := '++++++++++++++++++++++++++++++++';
        str(n,ochr);
        For i2 := 1 to phanline do
         Begin
           lineone[pline[i2-1]+1] := ' ';
           lineone[pline[i2-1]+2] := ' ';
         End;{for}
        If n < 10 Then
         Insert(' ',ochr,2);
        For i2 := 1 to 31 do
         Insert(' ',lineone,i2*2);
        Insert('  ',lineone,1);
        Insert(ochr,lineOne,1);
        Point;
        Inc(n);
      End;
    End;{else}
   For j := 1 to m Do
    Begin
      LineOne := '                                ';
      str(n,ochr);
      For i2 := 1 to phanline do
       Begin
         lineone[pline[i2-1]+1] := ' ';
         lineone[pline[i2-1]+2] := ' ';
       End;{for}
      If n < 11 Then
       Insert(' ',ochr,2);
      For i2 := 1 to 31 do
       Insert(' ',lineone,i2*2);
      Insert('  ',lineone,1);
      Insert(ochr,lineOne,1);
      Point;
      Inc(n)
    End;
 End;{dontcare}

{****************************************}
{Procedure to display JEDEC file on screen}
{****************************************}
Procedure readjedec;
 Begin
   Clrscr;
```

BYLAAG C
FUSEMAP.PAS

```
  openrecs;
  Writeln;
  If okay then
   Begin
     Quit := False;
     Repeat
       Write('Output to S.creen, P.rinter ? ');
       Readln(Choice);
       Case Choice Of
        'P','p' : Begin
                    Printrecord;
                    Quit := True;
                  End;
        'S','s' : Begin
                    Readrecord;
                    Quit := True;
                  End;
       End;{case}
     Until Quit;
     Assign(Jfile,Frec.name+'.jed');
    {$i-}
     Reset(Jfile);
    {$i+}
     If IoResult <> 0 Then
      Writeln('There is no jedec file of ',Frec.name)
     Else
      Begin
        While Not Eof(Jfile) Do
         Begin
           Readln(Jfile,oneline);
           Writeln(oneline);
         End;{While}
        Close(JFile);
      End;{else};
   End;{if}
  Readrecord;
  Writeln;
  Write('Press enter to go on. ');
  Readln;
End;{readjedec}
```

BYLAAG C
FUSEMAP.PAS

```
{*****************************************************************}
{Procedure to get the number and names of circuits in the database}
{*****************************************************************}
Procedure Getx;
 Begin
   Assign(Ffile,'Cirfile.CIR');
  {$i-}
    Reset(Ffile);
  {$i+}
   If IoResult <> 0 Then
    Writeln('No data circuits in database. ')
   Else
    Begin
      cirnumber := Filesize(Ffile) - 1;
      For i := 0 to Cirnumber do
       Begin
         Seek(Ffile,i);
         Read(Ffile,Frec);
         circuit[i] := Frec.Name;
       End;{for}
      Close(Ffile);
    End;{else}
 End;{getx}

{***********************************************}
{Procedure to erase all records from the database}
{***********************************************}
Procedure EraseRecords;
 Begin
   close(ffile);
   Writeln('All data in database wil be lost !!!');
   Quit := False;
   Repeat
     Write('Do you want to proceed Y/N ? ');
     Read(wipe);
     wipe := Upcase(wipe);
     If  wipe = 'Y' Then
      Begin
        Erase(ffile);
        Quit := true;
      End
     Else
      If wipe = 'N' Then
       Quit := true;
```

```
  Until Quit
 End;{erase}

{************************************}
{Procedure to change data in data base}
{************************************}

Procedure EditRec;
 Begin
   Clrscr;
   CharSet := ['a'..'z','A'..'Z',' '] ;
   Assign( Ffile,'CirFile.Cir' ) ;
  {$i-}
   Reset(Ffile) ;
  {$i+}
   If IoResult <> 0 Then
    Begin
      Writeln('No record of Fuse map  try again.');
      Okay := False;
    End
   Else
     Begin
       ydig := '';
       Quit := False;
       Repeat
         Write('Edit One record /E, Erase one record /0 or Erase all record./A ? ');
         Readln(Wipe);
         wipe := upcase(wipe);
       Until (wipe = '0') or (wipe = 'A') or (wipe = 'E');
        If wipe = 'A' Then
         Eraserecords
        Else
         Begin
           If wipe = '0' then
            Begin
              Clrscr;
              Okay := True;
              Quit := false;
              Repeat
                Writeln('Data of next circuits in database ');
                Writeln;
                j := 0;
                n := 3;
                For i := 0 to cirnumber do
```

```
  Begin
    GotoXY(j*18,n);
    Write('¦');
    Write(i + 1,'. ',circuit[i]);
    inc(j);
    If j = 4 Then
     Begin
       writeln;
       j := 0;
       Inc(n);
     End;{if}
  End;{for}
 Writeln;
 Writeln;
 Write('Enter the digit of circuit you want to delete or Esc to quit : ');
 Xdig := readkey;
 If Xdig = #27 then
  Quit := true
 Else
  Begin
    write(xdig);
    insert(xdig,ydig,1);
    xdig := readkey;
    If xdig <> #13 Then
     Insert(xdig,ydig,2);
    Val(ydig,dig,code);
    If dig < cirnumber + 2 then
     Quit := true
    Else
     Begin
       writeln;
       writeln('You must enter a digit smaller than ',cirnumber+2,' !!!');
     End;{else}
  End;{else}
Until Quit;
If Xdig = #27 then
 Begin
   Clrscr;
   Exit;
 End;
Clrscr;
Assign(FXfile,'cirfile.xxx');
Rewrite(FXfile);
For j := 0  to cirnumber do
```

BYLAAG C
FUSEMAP.PAS

```
        Begin
          If (j + 1) <> dig Then
           Begin
             Seek(Ffile,j);
             read(ffile,frec);
             Seek(FXfile,fileSize(FXfile));
             Write(FXfile,FRec);
           End{if};
         End{for};
        Rewrite(ffile);
        For j := 0  to (cirnumber - 1) do
         Begin
           Seek(FXfile,j);
           Read(fXfile,frec);
           Seek(Ffile,fileSize(Ffile));
           Write(Ffile,FRec);
         End{for};
        Close(Fxfile);
        Close(ffile);
        Erase(Fxfile);
      End{if}
     Else
      Begin
        Clrscr;
        Okay := True;
        Writeln('Data of next circuits in database ');
        Writeln;
        j := 0;
        n := 3;
        For i := 0 to cirnumber do
         Begin
           GotoXY(j*18,n);
           Write('¦');
           Write(i + 1,'. ',circuit[i]);
           inc(j);
           if j = 4 Then
            Begin
              writeln;
              j := 0;
              Inc(n);
            End;{if}
         End;{for}
        Writeln;
        Writeln;
```

```
quit := False;
Repeat
  Write('Enter the digit of circuit you want to work with or Esc to quit : ');
  Xdig := readkey;
  If Xdig = #27 then
   Begin
     Clrscr;
     Exit;
   End
  Else
   Begin
     Write(Xdig);
     insert(xdig,ydig,1);
     xdig := readkey;
     if xdig <> #13 Then
      insert(xdig,ydig,2);
     val(ydig,dig,code);
   End;{else}
  For i := 1 To cirnumber + 1 do
    If i = dig Then
     Quit := True;
Until Quit;
Seek(Ffile,(dig - 1));
Read(Ffile,Frec);
Digit := Frec.Dig;
Assign( Pfile,'PalFile.Pal' ) ;
{$i-}
Reset(Pfile);
{$i+}
Seek(pfile,digit);
Read(Pfile,Prec) ;
close(pfile);
Clrscr;
Writeln;
Repeat
  Write('Edit C.ircuit name, P.in name, eX.prestion ? ');
  Readln(choice);
  choice := upcase(choice);
Until (choice = 'C') or (choice = 'P') or (choice = 'X');
If choice = 'C' Then
 Begin
   Write('cuircuit name ? : ');
   Readln(Frec.Name);
End{if}
```

BYLAAG C
FUSEMAP.PAS

```
Else
 Begin
   clrscr;
   Writeln('Summary of ',Frec.name,' making use of ',Prec.Name);
   Writeln ;
   j := 1 ;
   n := 3 ;
   For i := 1 to PRec.Count DO
    Begin
      If (PRec.Status[i] = 'INPUT') or (PRec.Status[i] = 'OUTPUT') THEN
       Okay := True
      Else
       Okay := False ;
      If Okay Then
       Begin
         If frec.Notname[i] = '!' Then
          Write('Pin ',i,' : ',PRec.Status[i],' = ',Frec.NotName[i],FRec.PinName[i])
         Else
         Write('Pin ',i,' : ',PRec.Status[i],' = ',FRec.PinName[i]);
         GotoXY(j*25,n) ;
         Write('_ ');
         j := j + 1;
       End{if}
      Else
       Writeln('Pin ',i,' : ',PRec.status[i]) ;
      If j = 4 Then
       Begin
         Writeln;
         Inc(n);
         j := 1;
       End;{if}
    End;{for}
   Writeln ;
   Writeln ;
   For k := 1 to Frec.Counter Do
    Begin
      For i := 1 To Prec.Count Do
       Begin
         If i = Frec.Out[k] Then
          Begin
            Writeln('Pin ',i);
            If Prec.ProdEn[i] = 'Y' Then
             Begin
               Writeln('Output enable :',Frec.ExEnable[i]);
```

```
          End;{if}
         Writeln('Expression    : ',FRec.expres[i]);
       End;{if}
     End;{for}
  End;{for}
 If choice = 'P' Then
  Begin
    Write('Enter the pin number to be edit. ');
    Readln(pos);
    If (PRec.Status[pos] = 'INPUT') Then
     Begin
       Repeat
        Write('Pin ',pos,' : ',PRec.Status[pos],' = ') ;
        xPname := readkey;
        IF xPname = '!' Then
         Begin
           Write(xPname);
           Frec.notName[pos] := '!';
           xpname := readkey;
           Frec.Pinname[pos] := xPname;
           write(xPname);
         End{if}
        Else
         Begin
           FRec.PinName[pos] := xPname ;
           Frec.notName[pos] := '*';
           write(xpname);
         End;{else}
        If Not (Frec.PinName[pos] In CharSet) Then
         Begin
           Write('Sorry Try again ? ,');
           Writeln(' Use alphabet or press space bar for no entry.');
           Writeln;
         End;{if}
       Until Frec.PinName[pos] In CharSet
     End{if}
    Else
     Begin
       If (PRec.Status[pos] = 'OUTPUT') THEN
        Begin
          Repeat
           Write('Pin ',pos,' : ',PRec.Status[pos],' = ') ;
           xPname := readkey;
           If xPname = '!' Then
```

```
                Begin
                  Write(xpname);
                  Frec.notname[pos] := '!';
                  Xpname := readkey;
                  FRec.PinName[pos] := xPname;
                  Write(xPname);
                End{if}
               Else
                Begin
                  frec.pinName[pos] := xPname;
                  Frec.NotName[pos] := '*';
                  Write(xpname);
                End;{else}
               If Not (Frec.PinName[pos] In CharSet) Then
                Begin
                  Write('Sorry Try again ? ,');
                  Writeln(' Use alphabeth or press space bar for no entry.');
                  Writeln;
                End;{if}
              Until Frec.PinName[pos] In CharSet;
            End{if}
           Else
            Writeln('Error ???  / Pin ',pos,' : ',Prec.status[pos]);
         End;{else}
       End;{if}
     If choice = 'X' Then
      Begin
        Write('Enter the pin number to be edit. ');
        readln(pos);
        i := pos;
        If Prec.proden[pos] = 'Y' then
         Begin
           Script := 'Enter Boolean expression for output enable of pin no ';
           EN := 'y';
           check;
           Script := 'Enter Boolean expression for output pin no ';
           EN := 'n';
           check;
         End{if}
        Else
         Begin
           Script := 'Enter Boolean expression for output pin no ';
           EN := 'n';
           check;
```

BYLAAG C
FUSEMAP.PAS

```
                End;{else}
              End;{if}
            End;{else}
           Seek(ffile,dig - 1);
           Write(Ffile,Frec);
           Close(ffile);
          End;{else}
        End;{Else}
    End;{else}
  Getx;
 End;{edit}

BEGIN                                                    {Begin program}
  Getx;
  Clrscr;
  Assign( PFile,'PalFile.PAL' );
 {$i-}
   Reset(PFile);
 {$i+}
  If IoResult <> 0 THEN
   Writeln('No data of PALs ')
  Else
   Begin
     RecNumber := FileSize(PFile) - 1;
     For i := 0 to RecNumber do
      Begin
        Seek(pfile,i);
        Read(PFile,PRec);
        PalTypes[i] := PRec.Name;
      End;{For}
     Close(pfile);
   End;{Else}
  Clrscr;
  Writeln('Fuse Map data.');
  Quits := False;
  Repeat
    Writeln;
    Write( 'A.dd, R.ead, M.ake, J.edec, E.dit, Q.uit ? ');
    Readln(Choice);
     CASE Choice OF
      'A','a' : ExpresInfo;
      'R','r' : ReadExpresInfo;
      'M','m' : MakeMap ;
      'e','E' : editrec;
```

BYLAAG C
FUSEMAP.PAS

```
      'J','j' : Readjedec;
      'Q','q' : Quits := True;
      Else Write('Error : Try again. ')
     End; { CASE }
   Until Quits;
END.
```

BYLAAG D
SIM.PAS

```
{************************************************}
{This program simulate the operation of a PAL device}
{************************************************}
{Sm 16384,0,655360}                                  {STACK SIZE = 16k, HEAP MIN = 0, HEAP MAX = 64k}
Program Simulate;

Uses
 Crt,                                                {Pascal CRT uint}
 Dos;                                                {Pascal Dos unit}

Const
     MaxNo = 20;                                     {Max number of pins and product lines}
     MaxPal = 30;                                    {Max number of Pal types }
     Printerx  = 'PRN';
     screen = '';

Type
  DataPtr = ^DataRecord;                             {Data record memory pointer}

  DataRecord = record                                {Data record structure}
     Dataln : String[50];                            {String to keep info}
     Next   : Dataptr;                               {Pointer to next record for linked list}
  End;{datarec}

  EditPtr = ^EditRecord;                             {Edit record memory pointer}

  EditRecord = Record                                {Edit record structure}
     Editln : String[50];                            {String to keep info}
     Next   : EditPtr;                               {Pointer to next record for linked list}
  End;{editrec}

  PalRec =  Record
     Name        : String[5];                        {Pal name}
     State       : Char;                             {Status of pal outputs}
     Count       : Byte;                             {Number of pins}
     Status      : Array[1..MaxNo] OF String[6];     {Status of pins}
     Inputline   : ARRAY[1..MaxNo] OF Integer;       {Inputline conect to which input line}
     Proden      : array[1..MaxNo] Of Char;          {Got output a product enable line or not}
     Productline : ARRAY[1..MaxNo] OF Integer;       {The number of the first product line of the output}
     Reg         : Array[1..Maxno] Of Char;          {Output with register}
     NoOf        : ARRAY[1..MaxNo] OF Integer;       {Number of Product lines conected to OR gate of output pin}
     Feedback    : ARRAY[1..MaxNo] OF Char;          {Output with feedback}
     Fback       : ARRAY[1..MaxNo] OF Integer;       {Feedback to whith input line}
  End;{PalRec}
```

BYLAAG D
SIM.PAS

```
VAR
  Firstline,
  Prevline,
  Currentline    : DataPtr;                    {Linked list pointers for data record}
  Firstln,
  Prevln,
  Currentln      : EditPtr;                    {Linked list pointers for edit rocord}
  PFile          : FILE OF  PalRec;            {Typed file with PAL data}
  Jfile          : Text;                       {Text file for JEDEC file data}
  PRec           : PalRec;                     {Variable to structure of PAL data}
  look,look2,
  look3,look4,   : Char;                       {To check if JEDEC file exist before simulation}
  xchoice,Choice : Char;                       {Misc variables for general use}
  InCount        : Integer;                    {Total inputs used in JEDEC file}
  Templ          : Integer;                    {Number of input lines}
  Phanline       : Integer;                    {Number of phantom input lines}
  Phan           : Integer;                    {Phntom product lines at a spesific output}
  RecNumber      : Integer;                    {Number of PALs in PAL data base}
  Fn             : Integer;                    {First fuse number product line}
  Code           : Integer;                    {Variable to show if conversion is successful}
  Strcount       : Integer;                    {Couter for Onestr}
  Notinc         : Integer;                    {Counter vir Notin}
  PrCount        : Integer;                    {Number product lines not used}
  Prodcount      : Integer;                    {Number of product lines used}
  ZeroLine       : Integer;                    {First product line}
  Placen         : Integer;                    {Number of inputs for expression}
  Excount        : Integer;                    {Counter for Expresar}
  Countjed       : Integer;                    {Counter for Jedprod}
  Rescount       : Integer; {counter vir result}
  Nextnumber,Klinenumber,
  CountDel,Prevnumber,
  storeprod,
  Digit,Multiply,
  i,j,n,m,k,a,n2,
  z,w,m2,j2      : Integer;                    {Misc variables for general use}
  Quits,Done,
  Quit,donex     : Boolean;                    {Misc variables for general use}
  Name           : String[5];                  {PAL name}
  Filename       : String[30];                 {JEDEC file name}
  Oneline        : String[50];                 {Oneline of JEDEC file}
  Noneline       : String[42];                 {String to store if no product line exist}
  Uoneline       : String[42];                 {String to store if product line is not used}
  Ones           : String[2];
  Onestr         : String[3];                  {One input}
```

BYLAAG D
SIM.PAS

```
  LFN           : String[12];                                   {Linenumber in JEDEC file}
  Temp          : String[6];                                    {Temporrary  status of PAL pin}
  PalSet        : Set of Char;                                  {Type of PAL output -> H,L or R}
  Vecset        : Set of byte;
  Regset        : Set of byte;
  Templine      : Array[0..16] of Integer;                      {Input lines used}
  Pline         : Array[0..16] of Integer;                      {Phantom input lines}
  Pno           : Array[0..20] of Integer;                      {Input that corespond with Templine}
  PrLine        : Array[0..63] of Integer;                      {Phantom product lines}
  Prodlines     : Array[0..63] of Integer;                      {Product lines existing in PAL}
  Jedprod       : Array[0..63] of Integer;                      {Product lines used}
  Pinout        : Array[1..20] of Integer;                      {Output pins used}
  Place         : Array[0..20] of Integer;                      {Input pins in sequence}
  Totalin       : Array[1..20] of Integer;                      {Total inputs used in JEDEC}
  Notin         : Array[1..10] of Integer;                      {I/O  pins with Totalin}
  Expresar      : Array[1..80] of Integer;                      {Temporarily function for output}
  Enstatus      : array[1..20] of char;                         {Status of output enable}
  Result        : Array[1..200] of String[18];                  {Tesvectors + results}
  PalTypes      : Array[0..MaxPal] of String[5];                {Existing PALs in  PAL data base}
  Expres        : Array[1..20] of Array[0..80] of String    {Function for all outputs}
  ExPlace       : Array[1..20] Of Array[0..20] Of integer;  {Inputs used in order}
  EnPlace       : Array[1..20] Of Array[0..20] of integer;  {Functions for pin enables}
  y,x,Terms     : Real;
  Heaptop       : ^word;

Procedure Put_in_dont;forward;                                {Procedure will follow}

{***********************************************}
{Procedure to load JEDEC file from disk to memory}
{***********************************************}
Procedure Load;
 Procedure Point;                                             {Procedure to store one line of text file in memory}
  Begin
    If Firstline = Nil Then
     Begin
       New(Currentline);
       Currentline^.Dataln := Oneline;
       Currentline^.Next := Nil;
       Firstline := Currentline;
     End{if}
    Else
     Begin
       Prevline := Currentline;
       New(Currentline);
```

```
      Currentline^.Dataln := Oneline;
      Prevline^.Next := Currentline;
      Currentline^.Next := Nil;
    End;{else}
 End;{point}

Begin
  Writeln;
  Write('Enter Jedec Filename. : ');
  Readln(Filename);
  Assign(Jfile,Filename);
 {$i-}
   Reset(JFile);
 {$i+}
  IF IoResult <> 0 THEN
   Begin
     Writeln;
     Writeln('Error ??? (Wrong filename or extention.) ');
   End{if}
  Else
   Begin
     look := 'Y';
     Look2 := 'N';
     Look3 := 'Y';
     release(heaptop);
     mark(heaptop);
     firstline := nil;
     While not Eof(Jfile) Do
      Begin
        Readln(Jfile,Oneline);
        Point;
      End;{while}
     Close(Jfile);
   End;{else}
End;{Load}

{************************************************************************}
{Procedure to read data of JEDEC file from memory and display on screen}
{************************************************************************}
Procedure Readmem;

Var
 Data : DataPtr;
```

BYLAAG D
SIM.PAS

```
Begin
  If look3 <> 'Y' Then
   Writeln('First try load.')
  Else
   Begin
     Data := Firstline;
     While Data <> Nil Do
      Begin
        With Data^ Do
        Writeln(Dataln);
        Data := Data^.Next;
      End;{while}
     Readln;
   End;{Else}
End;{readmem}

{*****************************************************************************}
{Procedure to read data of converted JEDEC file from memory and display on screen}
{*****************************************************************************}
Procedure Read_Put_in;

Var
 EditData : EditPtr;

Begin
  If (Look3 <> 'Y') Or (look2 <> 'Y') Then
   Writeln('Error ? First try load or pal.')
  Else
   Begin
     j := 0;
     EditData := Firstln;
     Writeln('            11111111112222222233');
     Writeln('   01234567890123456789012345678901');
     Writeln;
     While EditData <> Nil Do
      Begin
        With EditData^ Do
        Writeln(Editln);
        EditData := EditData^.Next;
        Inc(j);
        If j = 16 Then
         Begin
           Readln;
```

```
          Writeln('            11111111112222222233');
          Writeln('   01234567890123456789012345678901');
          Writeln;
          j := 0;
        End;{if}
      End;{while}
     Readln;
   End;{else}
End;{Put_in}

{*****************************************************************************}
{Procedure to convert a JEDEC file to a type of fusemap. The simulation procedure uses}
{this  converted JEDEC file to simulate the operation of the PAL                }
{*****************************************************************************}
Procedure Put_in_dont;

 Var
  Data : DataPtr;

 {*********************************************************}
 {Procedure to change JEDEC line number to product line number}
 {*********************************************************}
 Procedure Lnumber;
  Begin
    Done := false;
    i := 0;
    Repeat
      If FN = i*Multiply Then
       Begin
         KLinenumber := Prodlines[i];
         Done := true;
       End{if}
      Else
       Inc(i);
    Until done;
  End;{Lnumber}

 {*****************************************************}
 {Procedure to store one line of converted data in memory}
 {*****************************************************}
 Procedure Pointx;
  Begin
    If Firstln = Nil Then
     Begin
```

```
      New(Currentln);
      Currentln^.Editln := Oneline;
      Currentln^.Next := Nil;
      Firstln := Currentln;
    End{if}
   Else
    Begin
      If (memavail < 100) then
       Begin
         Writeln('FATAL error: Not enough memory to perform opreation !!');
         Writeln('This is probably due to selecting the incorrect PAL!!');
         Halt(1);
       End{if}
      Else
       Begin
         Prevln := Currentln;
         New(Currentln);
         Currentln^.editln := Oneline;
         Prevln^.Next := Currentln;
         Currentln^.Next := Nil;
       End;{else}
    End;{else}
 End;{pointx}

{************************************************************}
{Procedure to determine product lines not used or not existing}
{************************************************************}
Procedure Notuse;
 Begin
   For i := Prevnumber + 1 to Klinenumber - 1 do
    Begin
      Oneline := ' ';
      Str(i,Ones);
      If i < 11 Then
       Insert(' ',ones,2);
      If i = prodlines[j] Then
       Begin
         Insert(Ones,Uoneline,1);
         Insert(Uoneline,oneline,1);
         Inc(j);
         Pointx;
         Delete(Uoneline,1,2);
       End{if}
      Else
```

```
      Begin
        Insert(Ones,Noneline,1);
        Insert(Noneline,oneline,1);
        Pointx;
        Delete(Noneline,1,2);
      End;{Else}
    End;{for}
 End;{Notuse}

{***********************************}
{Procedure to convert one product line}
{***********************************}
Procedure Do_line;
 Begin
   j := storeprod + 1;
   Notuse;
   storeprod := j;
   Oneline := '';
   Oneline := Data^.Dataln;
   Delete(oneline,1,CountDel);
   For i := 2 to length(oneline) do
    Begin
      Choice := Oneline[i];
      If (choice = ' ')  Then
       Delete(Oneline,i,1);
    End;{for}
   Delete(oneline,length(oneline),1);
   Choice := 'n';
   For i := 0 to phanline - 1 do
    Begin
      j := pline[i];
      Insert(choice,oneline,j + 2);
      Insert(choice,oneline,j + 3);
    End;{for}
   Str(Klinenumber,Ones);
   If Klinenumber < 11 Then
    Insert(' ',ones,2);
   Insert(Ones,Oneline,1);
   Pointx;
   prevnumber := klinenumber;
 End;{do_line}

Begin                                    {Start Putin_dont program}
  For i := 0 to 63 do
```

BYLAAG D
SIM.PAS

```
 jedprod[i] := 99;
countjed := 0;
For i := 1 to zeroline do
 Begin
   Oneline := ' ';
   NOneline := ' nnnnnnnnnnnnnnnnnnnnnnnnnnnnnnnnn';
   Str((i - 1),Ones);
   If i < 11 Then
    Begin
      Insert(' ',ones,2);
    End;{if}
   Insert(Ones,Noneline,1);
   Insert(Noneline,oneline,1);
   Pointx;
   Delete(Noneline,1,2);
 End;{for}
Data := Firstline;
Done := False;
Repeat
  With Data^ Do
  Choice := Dataln[1];
  If Choice = 'L' Then
   Done := true
  Else
   Data := Data^.Next;
Until done;
Oneline := Data^.Dataln;
i := 1;
LFN := '';
Repeat
  Inc(i);
  Choice := Oneline[i];
  If choice <> ' ' Then
   Insert(choice,LFN,i);
Until Choice = ' ';
CountDel := Length(LFN) + 1;
Val(LFN,FN,Code);
Multiply := (32 - 2*Phanline);
Lnumber;
Jedprod[countjed] := klinenumber;
Inc(countjed);
UOneline := ' 00000000000000000000000000000000';
   Choice := 'n';
For i := 0 to phanline - 1 do
```

BYLAAG D
SIM.PAS

```
 Begin
   j := pline[i];
   Insert(choice,uoneline,j + 2);
   Insert(choice,uoneline,j + 3);
 End;{for}
Delete(uoneline,(length(uoneline) - Phanline*2),Phanline*2);
j := 0;
Prevnumber := Zeroline - 1;
notuse;
StoreProd := j;
Oneline := ' ';
Oneline := Data^.Dataln;
Delete(oneline,1,CountDel);
For i := 2 to length(oneline) do
 Begin
   Choice := Oneline[i];
   If (choice = ' ')  Then
    Delete(Oneline,i,1);
 End;{for}
Delete(oneline,length(oneline),1);
Choice := 'n';
For i := 0 to phanline - 1 do
 Begin
   j := pline[i];
   Insert(choice,oneline,j + 2);
   Insert(choice,oneline,j + 3);
 End;{for}
Str(Klinenumber,Ones);
If Klinenumber < 11 Then
 Insert(' ',ones,2);
Insert(Ones,Oneline,1);
Pointx;
Data := Data^.Next;
Quit := False;
PrevNumber := Klinenumber;
Repeat
  With Data^ Do
  Choice := Dataln[1];
  If Choice = 'C' Then
   Quit := true
  Else
   Begin
     Oneline := Data^.Dataln;
     i := 1;
```

BYLAAG D
SIM.PAS

```
      LFN := '';
      Repeat
        Inc(i);
        Choice := Oneline[i];
        If choice <> ' ' Then
         Insert(choice,LFN,i);
      Until Choice = ' ';
      CountDel := Length(LFN) + 1;
      Val(LFN,FN,Code);
      Multiply := (32 - 2*Phanline);
      LNumber;
      jedprod[countjed] := klinenumber;
      Inc(countjed);
      Do_line;
      Data := Data^.Next;
    End;{else}
  Until Quit;
  Klinenumber := 64;
  j := storeprod + 1;
  notuse;
End;{put_in_dont}

{*****************************************}
{Procedure to determine Phantom Product lines}
{*****************************************}
Procedure Dont;
 Begin
  If Prec.Proden[i] = 'Y' Then
   Begin
     Inc(n);
     n := n + Prec.NoOf[i];
     m := 7 - Prec.NoOf[i];
   End{if}
  Else
   Begin
     n := n + Prec.NoOf[i];
     m := 8 - Prec.NoOf[i];
   End;{Else}
End;{dont}
```

BYLAAG D
SIM.PAS

```
{************************************}
{Procedure to find output or input pin}
{************************************}
Procedure FindIn;
 Begin
   Repeat
     Inc(j);
     Temp := Prec.Status[j];
   Until (Temp = 'INPUT') or (Temp = 'OUTPUT') or (j >= Prec.Count) ;
 End;{findin}

{************************}
{Procedure to find output}
{************************}
Procedure Find;
 Begin
   Repeat
     Dec(i);
     Temp := prec.status[i];
   Until (Temp = 'OUTPUT') or (i = 1);
   Temp := '      ';
   n := Prec.Productline[i];
 End;

{***********************************************}
{Procedure to get PAL data from the PAL database}
{***********************************************}
Procedure PALs;
 Begin
   Firstln := Nil;
   For i := 0 to 63 do
    Begin
      Prline[i] := 99;
      prodlines[i] := 99;
    End;
   If look = 'N' Then
    Writeln('First load a jedecfile.')
   Else
    Begin
      Clrscr;
      Assign( PFile,'PalFile.PAL' );
     {$i-}
      Reset(PFile);
     {$i+}
```

BYLAAG D
SIM.PAS

```
IF IoResult <> 0 THEN
 Writeln('No data of PALs yet')
ELSE
 Begin
   Look2 := 'Y';
   Writeln('Pals in database');
   Writeln;
   j := 0;
   n := 3;
   For i := 0 to RecNumber Do
    Begin
      GotoXY(j*18,n);
      Write('_ ');
      Write(i + 1,'. ',PalTypes[i]);
      Inc(j);
      If j = 4 Then
       Begin
         Writeln;
         j := 1;
         Inc(n);
       End;{if}
    End;{For}
   Writeln;
   Writeln;
   quit := False;
   Repeat
     Write('Enter digit Of PAL to be used : ');
     Readln(digit);
     For i := 1 To Recnumber + 1 do
      Begin
        If i = digit Then
         Quit := True;
       End;{for}
   Until Quit;
   Seek(pfile,(digit - 1));
   Read(PFile,PRec);
   Close(Pfile);
 End;{else}
j := 0;
Temp := '';
FindIn;
Templ := 0;
While j < Prec.Count Do
 Begin
```

```
      If Temp = 'INPUT' Then
       Begin
         Templine[Templ] := Prec.Inputline[j];
         Pno[Templ] := j;
         Inc(Templ);
       End{if}
      Else
       Begin
         If Prec.Feedback[j] = 'Y' Then
          Begin
            Templine[Templ] := Prec.Fback[j];
            Pno[Templ] := j;
            Inc(Templ);
          End;{if}
       End;{else}
      FindIn;
   End;{while}
  Phanline := 0;
  j := 0;
  Repeat
    i := 0;
    Done := false;
    Repeat
      If j = Templine[i] then
       Done := true
      Else
       Begin
         If i = (Templ - 1) Then
          Begin
            Pline[phanline] := j;
            Inc(Phanline);
            Done := true;
          End{if}
         Else
          Inc (i);
       End;{else}
    Until Done;
    j := j + 2;
  Until j = 32;
  i := (Prec.Count + 1) ;
  k := 0;
  Find;
  Prcount := 0;
  ZeroLine := 0;
```

BYLAAG D
SIM.PAS

```
If n > 7 Then
 Begin
   Zeroline := n;
   j := 0;
   For k := 0 to (n - 1) do
    Begin
      PrLine[k] := j;
      Inc(j);
      prcount := (j)
    End;{For}
 End;{if}
While i > 1 do
 Begin
   Dont;
   For k := (prcount) To ((prcount + m)- 1) Do
    Begin
      PrLine[k] := n;
      Inc(n);
      inc(prcount);
    End;{for}
   Find;
 End;{while}
Repeat
  Inc(i);
  Temp := prec.status[i];
Until (Temp = 'OUTPUT') ;
j := Prec.productline[i];
If Prec.Proden[i] = 'Y' Then
 n := j + 7
Else
 n := j + 8 ;
m := (64 - n);
For k := prcount To ((prcount + m)- 1) do
 Begin
   PrLine[k] := n;
   Inc(n);
   Inc(Prcount);
 End;{for}
Prodcount := 0;
For j := 0 to 63 do
 Begin
   i := 0;
   done := false;
   Repeat
```

```
          If j = prline[i] Then
           Done := True
          Else
           Begin
             If i = Prcount Then
              Begin
                Prodlines[Prodcount] := j;
                inc(prodcount);
                Done := true;
              End{if}
             Else
              Inc(i);
           End;{else}
        Until done;
      End;{for}
     Put_in_dont;
     look := 'N';
   End;{else}
 End;{Pals}

{*****************************************************************}
{Procedure to get the names of all the PALs in the database}
{*****************************************************************}
Procedure Get;
 Begin
  Assign( PFile,'PalFile.PAL' );
 {$i-}
   Reset(PFile);
 {$i+}
   IF IoResult <> 0 THEN
     Writeln('No data of PALs yet.')
   ELSE
    Begin
      RecNumber := FileSize(PFile) - 1;
      For i := 0 to RecNumber do
       Begin
         Seek(pfile,i);
         Read(PFile,PRec);
         PalTypes[i] := PRec.Name;
       End;{For}
      Close(pFile);
    End;{Else}
 End;{GET}
```

BYLAAG D
SIM.PAS

```
{*****************************************}
{Procedure to simulate the operation of the PAL}
{*****************************************}
Procedure simulation;

 Var
   EditData        : EditPtr;
   Binstr          : string[32];
   Clkstatus,testn,
   vecn,outcount,
   fn2,pin_en      : integer;
   pinstr          : string[4];
   Vec1,Digit2,Outs : char;
   Vecstr,vecstr2  : string[18];
   Vecar           : Array[1..10] Of String[18];
   Dovec           : array[1..10] of integer;          {To check a if condition exist on all inputs}
   vecok           : array[1..10] of integer;          {To check if all the outputs is tested}
   vecpin          : array[1..20] Of char;             {Condition on pins}
   Regpin          : array[1..20] Of char;             {Previous output conditions of registers}

{*************************************************************************}
{Function to raise to the power of. This is used to determine the checksum}
{*************************************************************************}
Function Raise(x : real) : real;
Begin
  if x = 0.0 then
   raise := 1.0
 Else
     Begin
     raise := Exp(2 * ln(x))
     end
End;{Raise}

{*************************************}
{Procedure to get one converted product line}
{*************************************}
Procedure getline;
 Begin
   Done := false;
   Repeat
     With EditData^ Do
     oneline := Editln;
       If i = jedprod[j] Then
         Begin
```

```
          Done := true;
          Inc(j);
         End{if}
        Else
         Begin
           Inc(i);
           EditData := EditData^.Next;
           If editdata = nil Then
            Done := true;
         End;
      Until done;
End;

{*********************************************}
{Procedure to determine all the input lines used}
{*********************************************}
Procedure lineIn;
 Begin
   n := 0;
   Done := false;
   Repeat
     If Templine[n] = templ Then
      Done := true
     Else
      inc(n);
   Until Done;
   a := 0;
   Quit := false;
    Repeat
      If (Place[a] = Pno[n]) or (a = placen) Then
       Quit := true
      Else
       Inc(a);
    Until Quit;
     If Place[a] <> Pno[n] Then
      Begin
        a := 1;
        Quit := False;
        Repeat
          If Pno[n] < Place[a] Then
           Inc(a)
          Else
           Begin
             Prevnumber := Place[a];
             Place[a] := Pno[n];
```

BYLAAG D
SIM.PAS

```
          Inc(placen);
          For a := (a + 1) to placen Do
           Begin
             NextNumber := Place[a];
             Place[a] := Prevnumber;
             Prevnumber := nextnumber;
           End;{for}
          Quit := true;
        End;{else};
      Until Quit
    End;{if}
  Str(pno[n],ones);
  Insert(ones,onestr,strcount);
  Strcount := 1;
End;{inline}

{*****************************************************************}
{Procedure to determine the Boolean expression from the JEDEC file}
{*****************************************************************}
Procedure GetExpres;
 Begin
  strcount := 1;
  For k := 4 To 35 do
   Begin
     Choice := oneline[k];
     If Choice = '0' Then
      Begin
        templ := k - 4;
        If odd(Templ) Then
         Begin
           Choice := '!';
           Insert(choice,onestr,strcount);
           Inc(strcount);
           Dec(templ);
           lineIn;
         End{if}
        Else
         lineIn;
        expresar[excount] := onestr;
        Inc(excount);
        Onestr := '';
      End{if}
   End;{for}
End;{GetExpres}
```

BYLAAG D
SIM.PAS

```
{************************************************************************** ****}
{Procedure to write the heading when viewing the simulation results of only one output}
{*****************************************************************************************}
Procedure heading;
 Begin
   Onestr := 'Sta';
   Write(onestr);
   Pinstr := '¦ IN';
   For j := explace[pinout[fn],0] Downto 1 do
    Write(pinstr);
   Pinstr := '¦OUT';
   Write(pinstr);
   pinstr := '¦    ';
   Writeln(pinstr);
   onestr := 'PIN';
   Write(onestr);
   For j := explace[pinout[fn],0] Downto 1 do
    Begin
      Pinstr := '';
      Str(explace[pinout[fn],j],pinstr);
      if explace[pinout[fn],j] > 9 Then
       Insert('¦ ',pinstr,1)
      Else
       Insert('¦  ',pinstr,1);
      Write(pinstr);
    End;{for}
   Pinstr := '';
   Str(pinout[fn],pinstr);
   If Pinout[fn] > 9 Then
    Insert('¦ ',pinstr,1)
   Else
    Insert('¦  ',pinstr,1);
   Write(pinstr);
   writeln('¦');
   onestr := '---';
   write(onestr);
   Pinstr := '+---';
   For j := 1 to explace[pinout[fn],0] + 1 Do
    Write(pinstr);
   Writeln('¦');
   onestr := 'Vec';
   Write(onestr);
   pinstr := '¦    ';
   For j := 1 to explace[pinout[fn],0] + 1 Do
```

```
    Write(pinstr);
   writeln('¦');
 End;{heading}

{*****************************************************************************************}
{Procedure to write the heading when viewing the simulation results of all the outputs again}
{*****************************************************************************************}
Procedure headingall;
 Begin
   Onestr := 'Sta';
   Write(onestr);
   If Prec.status[1] = 'CLK' Then
    Begin
      Pinstr := '¦CLK';
      Write(pinstr);
    End;{if}
   Pinstr := '¦ IN';
   For j := 1 to (incount - notinc) do
    Write(pinstr);
   Pinstr := '¦OUT';
    for j := 1 to outcount do
   Write(pinstr);
   pinstr := '¦    ';
   Writeln(pinstr);
   onestr := 'PIN';
   Write(onestr);
   If Prec.status[1] = 'CLK' Then
    Begin
      Pinstr := '¦  1';
      Write(pinstr);
    End;{if}
   For j := incount downto 1 do   {incount -> totale insetpenne vir pal}
    Begin
      a := 0;
      Repeat
        If notinc <> 0 then
         Inc(a);
      Until (totalin[j] = notin[a]) or (a = notinc);
      If notin[a] <> totalin[j] then
       Begin
         Pinstr := '';
         Str(totalin[j],pinstr);
         if totalin[j] > 9 Then
          Insert('¦ ',pinstr,1)
```

```
        Else
         Insert('¦  ',pinstr,1);
        Write(pinstr);
       End;{if}
    End;{for}
   For j := outcount downto 1 do
   Begin
     Pinstr := '';
     Str(pinout[j],pinstr);
     If Pinout[j] > 9 Then
      Insert('¦ ',pinstr,1)
     Else
      Insert('¦  ',pinstr,1);
     Write(pinstr);
   End;{for}
   pinstr := '¦    ';
   Writeln(pinstr);
   onestr := '---';
   Write(onestr);
   Pinstr := '+---';
   If Prec.status[1] = 'CLK' Then
    Write(pinstr);
   For j := 1 to (incount + outcount - notinc) Do
    Write(pinstr);
   Writeln('¦');
   onestr := 'Vec';
   Write(onestr);
   pinstr := '¦    ';
   If Prec.status[1] = 'CLK' Then
    Write(pinstr);
   For j := 1 to (incount + outcount - notinc) Do
    Write(pinstr);
   writeln('¦');
 End;

{*********************************************************}
{Procedure to print the results of the simulation prosess}
{*********************************************************}

Procedure printvec;
 Begin
   Assign(OUTPUT,Printerx);
   Rewrite(OUTPUT);
   If fn <= outcount then
```

BYLAAG D
SIM.PAS

```
  Begin
    writeln('Testvectors for output ',pinout[fn],' of ',filename,' :');
    writeln;
    heading;
    For n2 := 1 to testn do
     Begin
       vecstr := '';
       vecstr := result[n2];
       pinstr := '';
       Str(n2,pinstr);
       If n2 > 9 Then
        Insert(' ',pinstr,1)
       Else
        Insert('  ',pinstr,1);
       write(pinstr);
       For i := 1 to length(vecstr) do
        Begin
          pinstr := '';
          insert(vecstr[i],pinstr,1);
          insert('¦  ',pinstr,1);
          write(pinstr);
        End;{for}
       writeln('¦');
     End;{for}
  End{if}
 Else
  Begin
    writeln('Testvectors for all the outputs of ',filename,' :');
    writeln;
    headingall;
    If prec.state = 'R' then
     Begin
       m2 := 1;
       For n2 := 1 to testn do
        Begin
          vecstr := '';
          vecstr := result[m2];
          inc(m2);
          pinstr := '';
          Str(n2,pinstr);
          If n2 > 9 Then
           Insert(' ',pinstr,1)
          Else
           Insert('  ',pinstr,1);
```

BYLAAG D
SIM.PAS

```
  Begin
    writeln('Testvectors for output ',pinout[fn],' of ',filename,' :');
    writeln;
    heading;
    For n2 := 1 to testn do
     Begin
       vecstr := '';
       vecstr := result[n2];
       pinstr := '';
       Str(n2,pinstr);
       If n2 > 9 Then
        Insert(' ',pinstr,1)
       Else
        Insert('  ',pinstr,1);
       write(pinstr);
       For i := 1 to length(vecstr) do
        Begin
          pinstr := '';
          insert(vecstr[i],pinstr,1);
          insert('¦  ',pinstr,1);
          write(pinstr);
        End;{for}
       writeln('¦');
     End;{for}
  End{if}
 Else
  Begin
    writeln('Testvectors for all the outputs of ',filename,' :');
    writeln;
    headingall;
    If prec.state = 'R' then
     Begin
       m2 := 1;
       For n2 := 1 to testn do
        Begin
          vecstr := '';
          vecstr := result[m2];
          inc(m2);
          pinstr := '';
          Str(n2,pinstr);
          If n2 > 9 Then
           Insert(' ',pinstr,1)
          Else
           Insert('  ',pinstr,1);
```

BYLAAG D
SIM.PAS

```
          Write(pinstr);
          For i := 1 to length(vecstr) do
           Begin
             pinstr := '';
             insert(vecstr[i],pinstr,1);
             insert('¦  ',pinstr,1);
             write(pinstr);
           End;{for}
          writeln('¦');
          vecstr := '';
          vecstr := result[m2];
          inc(m2);
          write('   ');
          For i := 1 to length(vecstr) do
           Begin
             pinstr := '';
             insert(vecstr[i],pinstr,1);
             insert('¦  ',pinstr,1);
             write(pinstr);
           End;{for}
          writeln('¦');
        End;{for}
      End{if}
     Else
      Begin
        For n2 := 1 to testn do
         Begin
          vecstr := '';
          vecstr := result[n2];
          pinstr := '';
          Str(n2,pinstr);
          If n2 > 9 Then
           Insert(' ',pinstr,1)
          Else
           Insert('  ',pinstr,1);
          write(pinstr);
          For i := 1 to length(vecstr) do
           Begin
             pinstr := '';
             insert(vecstr[i],pinstr,1);
             insert('¦  ',pinstr,1);
             write(pinstr);
           End;{for}
          writeln('¦');
        End;{for}
```

BYLAAG D
SIM.PAS

```
      End;{else}
    End;{else}
   Assign(OUTPUT,Screen);
   Rewrite(OUTPUT);
  End;{printvec}

{*****************************************}
{Procedure to view the simulation result again}
{*****************************************}
Procedure viewvec;
 Begin
   clrscr;
   If fn <= outcount then
    Begin
      heading;
      For n2 := 1 to testn do
       Begin
         vecstr := '';
         vecstr := result[n2];
         pinstr := '';
         Str(n2,pinstr);
         If n2 > 9 Then
          Insert(' ',pinstr,1)
         Else
          Insert('  ',pinstr,1);
         write(pinstr);
         For i := 1 to length(vecstr) do
          Begin
            pinstr := '';
            insert(vecstr[i],pinstr,1);
            insert('¦  ',pinstr,1);
            write(pinstr);
          End;{for}
         writeln('¦');
         If n2 in vecset Then
          Begin
            GotoXY(1,24);
            Write('Pres enter to go on. ');
            Readln;
            If n2 <> Testn Then
             Begin
               clrscr;
               heading
             End;{if}
```

BYLAAG D
SIM.PAS

```
        End;{if}
       If n2 = testn Then
        Begin
          GotoXY(1,24);
          Write('Pres enter to go on. ');
          Readln;
       end;{if}
    End;{for}
 End{if}
Else
 Begin
   headingall;
   If prec.state = 'R' then
    Begin
      m2 := 1;
      For n2 := 1 to testn do
       Begin
         vecstr := '';
         vecstr := result[m2];
         inc(m2);
         pinstr := '';
         Str(n2,pinstr);
         If n2 > 9 Then
          Insert(' ',pinstr,1)
         Else
          Insert('  ',pinstr,1);
         write(pinstr);
         For i := 1 to length(vecstr) do
          Begin
            pinstr := '';
            insert(vecstr[i],pinstr,1);
            insert('¦  ',pinstr,1);
            write(pinstr);
          End;{for}
         writeln('¦');
         vecstr := '';
         vecstr := result[m2];
         inc(m2);
         write('   ');
         For i := 1 to length(vecstr) do
          Begin
            pinstr := '';
            insert(vecstr[i],pinstr,1);
            insert('¦  ',pinstr,1);
```

```
            Write(pinstr);
          End;{for}
         writeln('¦');
         If n2 in regset Then
          Begin
            GotoXY(1,24);
            Write('Pres enter to go on. ');
            Readln;
            If n2 <> Testn Then
             Begin
               clrscr;
               headingall;
             End;{if}
          End;{if}
         If n2 = testn Then
          Begin
            GotoXY(1,24);
            Write('Pres enter to go on. ');
            Readln;
         end;{if}
       End;{for}
    End{if}
   Else
    Begin
      For n2 := 1 to testn do
       Begin
         vecstr := '';
         vecstr := result[n2];
         pinstr := '';
         Str(n2,pinstr);
         If n2 > 9 Then
          Insert(' ',pinstr,1)
         Else
          Insert('  ',pinstr,1);
         write(pinstr);
         For i := 1 to length(vecstr) do
          Begin
            pinstr := '';
            insert(vecstr[i],pinstr,1);
            insert('¦  ',pinstr,1);
            write(pinstr);
          End;{for}
         writeln('¦');
         If n2 in vecset Then
```

```
          Begin
            GotoXY(1,24);
            Write('Pres enter to go on. ');
            Readln;
            If n2 <> Testn Then
             Begin
               clrscr;
               headingall;
             End;{if}
          End;{if}
         If n2 = testn Then
          Begin
            GotoXY(1,24);
            Write('Pres enter to go on. ');
            Readln;
          end;{if}
        End;{for}
      End;{else}
   End;{else}
 End;{viewvec}

{************************************************************************}
{Procedure to write the heading when simulating only one output of a PAL}
{************************************************************************}
Procedure Pins;
 Begin
   Writeln('Give testvectors for the next inputs. ');
   Writeln;
   Onestr := 'Sta';
   Write(onestr);
   Pinstr := '¦ IN';
   For j := explace[pinout[fn],0] Downto 1 do
   Write(pinstr);
   Pinstr := '¦OUT';
   Write(pinstr);
   pinstr := '¦    ';
   Writeln(pinstr);
   onestr := 'PIN';
   Write(onestr);
   For j := explace[pinout[fn],0] Downto 1 do
    Begin
      Pinstr := '';
      Str(explace[pinout[fn],j],pinstr);
      if explace[pinout[fn],j] > 9 Then
```

BYLAAG D
SIM.PAS

```
      Insert('¦ ',pinstr,1)
     Else
      Insert('¦  ',pinstr,1);
     Write(pinstr);
   End;{for}
  Pinstr := '';
  Str(pinout[fn],pinstr);
  If Pinout[fn] > 9 Then
   Insert('¦ ',pinstr,1)
  Else
   Insert('¦  ',pinstr,1);
  Write(pinstr);
  pinstr := '¦    ';
  Writeln(pinstr);
  onestr := '---';
  Write(onestr);
  Pinstr := '+---';
  For j := 1 to explace[pinout[fn],0] + 1 Do
   Write(pinstr);
  Writeln('¦');
  onestr := 'Vec';
  Write(onestr);
  pinstr := '¦    ';
  For j := 1 to explace[pinout[fn],0] + 2 Do
   Write(pinstr);
End;{pins}

{*************************************************************}
{Procedure to load the PAL registers with the initial conditions}
{*************************************************************}
Procedure loadreg;
 Begin
  For j := 1 to 20 do
   Begin
     regpin[j] := 'x';
   End;{for}
  regpin[1] := '0';
  Writeln('Preload the next registers ');
  Writeln;
  write('OUTPUTPIN');
  For j := 1 to outcount do
   Begin
     If prec.reg[pinout[j]] = 'Y' then
      Begin
```

BYLAAG D
SIM.PAS

```
        Pinstr := '';
        Str(pinout[j],pinstr);
        If Pinout[j] > 9 Then
         Insert('¦ ',pinstr,1)
        Else
         Insert('¦  ',pinstr,1);
        Write(pinstr);
      End;{if}
  End;{for}
pinstr := '¦    ';
Writeln(pinstr);
Write('---------');
Pinstr := '+---';
For j := 1 to (outcount) Do
 Write(pinstr);
Writeln('¦');
write('PRE VALUE');
pinstr := '¦    ';
For j := 1 to (outcount + 1) Do
 Write(pinstr);
Vecstr := '';
Vecn := 1;
z := 0;
a := 13;
k := 5;
For j := 1 to outcount do
 Begin
   if prec.reg[pinout[j]] = 'Y' then
   Begin
     gotoXY(a,k);
      Repeat
        vec1 := readkey;
      Until (vec1 = '0') or (vec1 = '1') or (vec1 = #27);
      If vec1 <> #27 then
       Begin
         Write(vec1);
         regpin[pinout[j]] := vec1;
         a := a + 4;
       End{if}
      Else
       Begin
         clrscr;
         exit;
       End;
```

BYLAAG D
SIM.PAS

```
      End;{if}
    End;{for}
 End;{loadreg}

{************************************************************************}
{Procedure to write the heading when simulating all the outputs of a PAL}
{************************************************************************}
Procedure PinsAll;
 Begin
   Notinc := 0;
   For j := 1 to incount do
    Begin
      If Prec.status[totalin[j]] = 'OUTPUT' Then
       Begin
         If (Enstatus[totalin[j]] <> 'I') or (prec.reg[totalin[j]] = 'Y') Then
          Begin
            Inc(notinc);
            Notin[notinc] := totalin[j];
          End;{if}
       End;{if}
    End;{for}
   Writeln('Give testvectors for the next inputs. ');
   Writeln;
   Onestr := 'Sta';
   Write(onestr);
   If Prec.status[1] = 'CLK' Then
    Begin
      Pinstr := '¦CLK';
      Write(pinstr);
    End;{if}
   Pinstr := '¦ IN';
   For j := 1 to (incount - notinc) do
    Write(pinstr);
   Pinstr := '¦OUT';
    for j := 1 to outcount do
   Write(pinstr);
   pinstr := '¦    ';
   Writeln(pinstr);
   onestr := 'PIN';
   Write(onestr);
   If Prec.status[1] = 'CLK' Then
    Begin
      Pinstr := '¦  1';
```

BYLAAG D
SIM.PAS

```
    Write(pinstr);
  End;{if}
For j := incount downto 1 do   {incount -> totale insetpenne vir pal}
 Begin
   a := 0;
   Repeat
     If notinc <> 0 then
      Inc(a);
   Until (totalin[j] = notin[a]) or (a = notinc);
   If notin[a] <> totalin[j] then
    Begin
      Pinstr := '';
      Str(totalin[j],pinstr);
      if totalin[j] > 9 Then
       Insert('¦ ',pinstr,1)
      Else
       Insert('¦  ',pinstr,1);
      Write(pinstr);
    End;{if}
 End;{for}
For j := outcount downto 1 do
Begin
  Pinstr := '';
  Str(pinout[j],pinstr);
  If Pinout[j] > 9 Then
   Insert('¦ ',pinstr,1)
  Else
   Insert('¦  ',pinstr,1);
  Write(pinstr);
End;{for}
pinstr := '¦    ';
Writeln(pinstr);
onestr := '---';
Write(onestr);
Pinstr := '+---';
If Prec.status[1] = 'CLK' Then
 Write(pinstr);
For j := 1 to (incount + outcount - notinc) Do
 Write(pinstr);
Writeln('¦');
onestr := 'Vec';
Write(onestr);
pinstr := '¦    ';
If Prec.status[1] = 'CLK' Then
```

BYLAAG D
SIM.PAS

```
    Write(pinstr);
   For j := 1 to (incount + outcount + 1 - notinc) Do
    Write(pinstr);
 End;{pinsall}

Procedure testvecn;forward;

{*****************************************************************************}
{Procedure to determine the output condition for a given set of vectors for one output}
{*****************************************************************************}
Procedure Testvectors;
 Begin
   outs := '1';
   Vecstr := '';
   Vecn := 1;
   z := 0;
   a := 7;
   k := 7;
   For m2 := (m2 +1) to testn do
    Begin
      For j := 1 to explace[pinout[fn],0]  do
       Begin
         gotoXY(a,k);
         Repeat
           vec1 := readkey;
         Until (vec1 = '0') or (vec1 = '1') or (vec1 = #27);
         if vec1 <> #27 then
          Begin
            write(vec1);
            Insert(vec1,vecstr,vecn);
            Inc(vecn);
            a := a + 4;
          end{if}
         Else
          begin
            clrscr;
            exit;
         end;
      End;{for}
      If prec.proden[pinout[fn]] = 'Y' then
         Begin
           If Enstatus[pinout[fn]] = 'B' then
            Begin
              z := 0;
```

BYLAAG D
SIM.PAS

```
            m := 1;
            EditData := Firstln;
            Repeat
              EditData := Editdata^.Next;
              Inc(z);
            Until z = Prec.productline[pinout[fn]] - 1;
            Oneline := Editdata^.Editln;
            For n := Enplace[pinout[fn],0] DownTo 1 Do
             Begin
               If Prec.Status[Enplace[pinout[fn],n]] = 'INPUT' Then
                w := Prec.Inputline[Enplace[pinout[fn],n]]
               Else
                w := Prec.Fback[Enplace[pinout[fn],n]];
               If Oneline[w + 4] = '0' Then
                Begin
                  Oneline[w + 4] := Vecstr[m];
                  Inc(m);
                End{if}
               Else
                Begin
                  If Oneline[w + 5] = '0' Then
                   Begin
                     If vecstr[m] = '1' Then
                      oneline[w + 5] := '0'
                     Else
                      Oneline[w + 5] := '1';
                   End;{if}
                  Inc(m);
                 End;{else}
             End;{for}
            Outs := '1';
            For n := 4 To 35 Do
             Begin
               If Oneline[n] = '0' Then
                Outs := '0';
             End;{for}
            If outs = '0' Then
             Outs := 'Z';
          End;{if}
       End;{if}
    If Outs <> 'Z' Then
     Begin
       Done := false;
       z := 0;
```

BYLAAG D
SIM.PAS

```
m := 1;
EditData := Firstln;
If z <> Prec.productline[pinout[fn]] Then
 Begin
   Repeat
     EditData := Editdata^.Next;
     Inc(z);
   Until z = Prec.productline[pinout[fn]];
 End;{if}
Repeat
  Oneline := Editdata^.Editln;
  For n := Explace[pinout[fn],0] DownTo 1 Do
   Begin
     If Prec.Status[Explace[pinout[fn],n]] = 'INPUT' Then
      w := Prec.Inputline[Explace[pinout[fn],n]]
     Else
      w := Prec.Fback[Explace[pinout[fn],n]];
     If Oneline[w + 4] = '0' Then
      Begin
        Oneline[w + 4] := Vecstr[m];
        Inc(m);
      End{if}
     Else
      Begin
        If Oneline[w + 5] = '0' Then
         Begin
           If vecstr[m] = '1' Then
            oneline[w + 5] := '0'
           Else
            Oneline[w + 5] := '1';
         End;{if}
        Inc(m);
      End;{else}
   End;{for}
  Outs := '1';
  For n := 4 To 35 Do
   Begin
     If Oneline[n] = '0' Then
      Outs := '0';
   End;{for}
  If Outs = '1' Then
   Done := True;
  m := 1;
  Inc(z);
```

```
          If z > (Prec.Productline[pinout[fn]] + Prec.NoOf[pinout[fn]] - 1) Then
           Done := true;
          EditData := EditData^.Next;
         Until done;
         If (Prec.State = 'L') or (Prec.State = 'R') Then
          Begin
            If Outs = '1' Then
             Outs := '0'
            Else
             Outs := '1';
          End;{If}
       End;{if}
      GotoXY(a,k);
      Write(outs);
      Insert(outs,vecstr,vecn);
      result[m2] := vecstr;
      Vecstr := '';
      inc(k);
      a := 7;
      If m2 = testn then
       Writeln
      Else
       Begin
         If m2 in vecset Then
          Begin
            a := 1;
            inc(k);
            GotoXY(a,k);
            Write('Pres enter to go on. ');
            Readln;
            If n2 <> Testn Then
             Begin
               clrscr;
               Pins;
               Testvecn;
             End;{if}
          End;{if}
       End;{else}
    End;{for}
 End;{Testvectors}

Procedure vecnumber;forward;
```

BYLAAG D
SIM.PAS

```
{*********************************************************}
{Procedure to determine when output is enabled or disabled}
{*********************************************************}
Procedure Tvec_en;
 Begin
   EditData := Firstln;
      Repeat
        EditData := Editdata^.Next;
        Inc(z);
      Until z = (Prec.productline[pinout[j]] - 1);
      Oneline := Editdata^.Editln;
      For n := Enplace[pinout[j],0] DownTo 1 Do
       Begin
         If Prec.Status[Enplace[pinout[j],n]] = 'INPUT' Then
          w := Prec.Inputline[Enplace[pinout[j],n]]
         Else
          w := Prec.Fback[Enplace[pinout[j],n]];
         If Oneline[w + 4] = '0' Then
          Begin
            Oneline[w + 4] := Vecstr2[m];
            Inc(m);
          End{if}
         Else
          Begin
            If Oneline[w + 5] = '0' Then
             Begin
               If vecstr2[m] = '1' Then
                oneline[w + 5] := '0'
               Else
                Oneline[w + 5] := '1';
             End;{if}
            Inc(m);
          End;{else}
       End;{for}
      Outs := '1';
      For n := 4 To 35 Do
       Begin
         If Oneline[n] = '0' Then
          Outs := '0';
       End;{for}
 End;{Tvec_en}
```

BYLAAG D
SIM.PAS

```
{*********************************************************************}
{Procedure to determine the output condition of one output for a given}
{set of vectors when simulation of more than one output took place    }
{*********************************************************************}
Procedure Tvec;
 Begin
   EditData := Firstln;
   If z <> Prec.productline[pinout[fn2]] Then
    Begin
      Repeat
        EditData := Editdata^.Next;
        Inc(z);
      Until z = Prec.productline[pinout[fn2]];
    End;{if}
      Repeat
        Oneline := Editdata^.Editln;
        For n := Explace[pinout[fn2],0] DownTo 1 Do
         Begin
           If Prec.Status[Explace[pinout[fn2],n]] = 'INPUT' Then
            w := Prec.Inputline[Explace[pinout[fn2],n]]
           Else
             w := Prec.Fback[Explace[pinout[fn2],n]];
           If Oneline[w + 4] = '0' Then
            Begin
              Oneline[w + 4] := Vecstr2[m];
              Inc(m);
            End{if}
           Else
            Begin
              If Oneline[w + 5] = '0' Then
               Begin
                 If vecstr2[m] = '1' Then
                  oneline[w + 5] := '0'
                 Else
                  Oneline[w + 5] := '1';
               End;{if}
              Inc(m);
            End;{else}
         End;{for}
        Outs := '1';
        For n := 4 To 35 Do
         Begin
           If Oneline[n] = '0' Then
            Outs := '0';
```

```
        End;{for}
       If Outs = '1' Then
        Done := True;
       m := 1;
       Inc(z);
       If z > (Prec.Productline[pinout[fn2]] + Prec.NoOf[pinout[fn2]] - 1) Then
        Done := true;
       EditData := EditData^.Next;
     Until done;
     If (Prec.State = 'L') or (Prec.State = 'R') Then
      Begin
        If Outs = '1' Then
         Outs := '0'
        Else
         Outs := '1';
      End;{If}
End;{Tvec}


{*****************************************************************************}
{Procedure to determine the output conditions of all the outputs when simulation}
{of more than one output took place                                           }
{This procedure is used when simulation of PALs with no registers took place  }
{*****************************************************************************}
Procedure Tvectors;
 Begin
   Vecstr := '';
   Vecn := 1;
   z := 0;
   a := 7;
   k := 7;
   For m2 := (m2 +1) to testn do
    Begin
      For j:= 1 to 20 do
       vecpin[j] := 'x';
      For j := 1 to (incount - notinc) do
       Begin
         gotoXY(a,k);
         Repeat
           vec1 := readkey;
         Until (vec1 = '0') or (vec1 = '1') or (vec1 = #27);
         If vec1 <> #27 then
          Begin
            write(vec1);
            Insert(vec1,vecstr,vecn);
            Inc(vecn);
            a := a + 4;
```

```
      End{if}
     Else
      begin
        clrscr;
        exit;
      end;
  End;{for}
m := 0;
For j := incount downto (notinc + 1)do
 Begin
   Inc(m);
   vecpin[totalin[j]] := vecstr[m];
 End;{for}
vecstr2 := '';
For j := 1 to 20 do
 vecar[j] := vecstr2;
For j := 1 to outcount do
 Begin
   If prec.proden[pinout[j]] = 'Y' then
    Begin
      If Enstatus[pinout[j]] = 'B' then
       Begin
         vecpin[pinout[j]] := 'z'
       End;{if}
    End;{if}
 End;{for}
z := 0;
For j := 1 to outcount do
 Begin
   For m := explace[pinout[j],0] downto 1 do
    Begin
      inc(z);
      insert(vecpin[explace[pinout[j],m]],vecstr2,z)
    End;{for}
   vecar[j] := vecstr2;
   vecstr2 := '';
 End;{for}
For j := 1 to 10 do
 Begin
   dovec[j] := 1;
   vecOk[j] := 0;
 End;{for}
vecstr2 := '';
Quit := false;
```

BYLAAG D
SIM.PAS

```
Repeat
  For j := 1 to outcount do
   Begin
     If Vecpin[pinout[j]] <> 'z' then
      Begin
        If vecok[j] = 0 then
         Begin
           vecstr2 :=  vecar[j];
           For i := 1 to explace[pinout[j],0] do
            Begin
              If (vecstr2[i] = 'x') or (vecstr2[i] = 'z') then
               dovec[j] := 0;
            End;{for}
         End{if}
        Else
         dovec[j] := 0;
        If dovec[j] = 1 Then
         Begin
           Done := false;
           z := 0;
           m := 1;
           fn2 := j;
           Tvec;
           vecstr2 := '';
           Vecpin[pinout[fn2]] := outs;
           For i := 1 To outcount Do
            Begin
              z := 0;
              vecstr2 := vecar[i];
              For j2 := Explace[pinout[i],0] Downto 1 do
               Begin
                 Inc(z);
                 If explace[pinout[i],j2] =  pinout[fn2] Then
                  vecstr2[z] := outs;
               End;{for}
              vecar[i] := vecstr2;
              vecstr2 := '';
            End;{for}
           Vecok[j] := 1;
         End;{if}
        Vecstr2 := '';
      End{if}
     Else
      Begin
```

BYLAAG D
SIM.PAS

```
        vecstr2 := '';
         z := 0;
         pin_en := 1;
         For m := enplace[pinout[j],0] downto 1 do
          Begin
            inc(z);
            insert(vecpin[enplace[pinout[j],m]],vecstr2,z)
          End;{for}
         For m := 1 to enplace[pinout[j],0] do
          Begin
            If (vecstr2[m] = 'x') or (vecstr2[m] = 'z') then
             pin_en := 0;
          End;{for}
         If pin_en = 1 then
          Begin
            Done := false;
            z := 0;
            m := 1;
            tvec_en;
            If outs = '1' then
             vecpin[pinout[j]] := 'x'
            Else
             Begin
               Repeat
                 gotoXY(1,24);
                 write('Enter condition for I/O pin.' ,pinout[j]);
                 gotoXY(a - 4 + (outcount*4 - (j - 1)*4),k);
                 vec1 := readkey;
               Until (vec1 = '0') or (vec1 = '1') or (vec1 = #27);
               If vec1 <> #27 then
                Begin
                  write(vec1);
                  vecok[j] := 1;
                  vecpin[pinout[j]] := vec1;
                End{if}
               Else
                Begin
                  clrscr;
                  exit;
                End;
             End;{else}
          End;{if}
       End;{else}
    End;{for}
```

```
      j := 0;
      For i := 1 to outcount do
       j := j + vecok[i];
      If j = outcount then
       Quit := true
      Else
       for j := 1 to 10 do
        Begin
          dovec[j] := 1;
        End;{for}
    Until Quit;
    For j := outcount downto 1 do
     Begin
       GotoXY(a,k);
       Write(vecpin[pinout[j]]);
       a := a + 4;
     End;{for}
    vecstr := '';
    For j := 20 downto 1 do
     Begin
       If vecpin[j] <> 'x' then
        insert(vecpin[j],vecstr,1);
     End;{for}
     result[m2] := vecstr;
     Vecstr := '';
     inc(k);
     a := 7;
     If m2 = testn then
      Writeln
     Else
      Begin
        If m2 in vecset Then
         Begin
           a := 1;
           inc(k);
           GotoXY(a,k);
           Write('Pres enter to go on. ');
           Readln;
           If n2 <> Testn Then
            Begin
              clrscr;
              PinsAll;
              vecnumber;
            End;{if}
```

```
        End;{if}
      End;{else}
   End;{for}
End; {Tvectors}

{*****************************************************************************}
{Procedure to determine the output conditions of all the outputs when simulation}
{of more than one output took place                                           }
{This procedure is use when simulation of PALs with registers took place      }
{*****************************************************************************}
Procedure RTvectors;
 Begin
   Vecstr := '';
   Vecn := 1;
   z := 0;
   a := 11;
   k := 7;
   For m2 := (m2 +1) to testn do
    Begin
      For j:= 2 to 20 do
       vecpin[j] := 'x';
      gotoXY(7,k);
      write(clkstatus);
      For j := 1 to (incount - notinc) do
       Begin
         gotoXY(a,k);
         Repeat
           vec1 := readkey;
         Until (vec1 = '0') or (vec1 = '1') or (vec1 = #27);
         If vec1 <> #27 then
          Begin
            write(vec1);
            Insert(vec1,vecstr,vecn);
            Inc(vecn);
            a := a + 4;
          End{if}
         Else
          Begin
            clrscr;
            exit;
          End;
       End;{for}
      m := 0;
      For j := incount downto (notinc + 1) do
```

BYLAAG D
SIM.PAS

```
 Begin
   Inc(m);
   vecpin[totalin[j]] := vecstr[m];
 End;{for}
If m2 <> 1 Then
 Begin
   For j := 1 to outcount do
    if prec.reg[pinout[j]] = 'Y' then
     vecpin[pinout[j]] := regpin[pinout[j]];
 End;{if}
vecstr2 := '';
For j := 1 to 20 do
 vecar[j] := vecstr2;
For j := 1 to outcount do
 Begin
   If prec.proden[pinout[j]] = 'Y' then
    Begin
      If Enstatus[pinout[j]] = 'B' then
       Begin
         vecpin[pinout[j]] := 'z'
       End;{if}
    End;{if}
 End;{for}
z := 0;
For j := 1 to outcount do
 Begin
   For m := explace[pinout[j],0] downto 1 do
    Begin
      inc(z);
      insert(vecpin[explace[pinout[j],m]],vecstr2,z)
    End;{for}
   vecar[j] := vecstr2;
   vecstr2 := '';
   z := 0;
 end;{for}
For j := 1 to 10 do
 Begin
   dovec[j] := 1;
   vecOk[j] := 0;
 End;{for}
vecstr2 := '';
Quit := false;
Repeat
  For j := 1 to outcount do
```

BYLAAG D
SIM.PAS

```
Begin
  If Vecpin[pinout[j]] <> 'z' then
   Begin
     If vecok[j] = 0 then
      Begin
        vecstr2 :=  vecar[j];
        For i := 1 to explace[pinout[j],0] do
         Begin
           If (vecstr2[i] = 'x') or (vecstr2[i] = 'z') then
            Begin
              dovec[j] := 0;
              For m := 1 to explace[pinout[j],0] do
               Begin
                 If prec.reg[explace[pinout[j],m]] = 'Y' Then
                  begin
                    if vecpin[explace[pinout[j],m]] = 'x' then
                     vecok[j] := 1;
                  end{if}
               End;{for}
            End;{if}
         End;{for}
      End{if}
     Else
      dovec[j] := 0;
     If dovec[j] = 1 Then
      Begin
        Done := false;
        z := 0;
        m := 1;
        fn2 := j;
        Tvec;
        vecstr2 := '';
        If prec.reg[pinout[fn2]] = 'Y' then
         Regpin[pinout[fn2]] := outs
        Else
         Vecpin[pinout[fn2]] := outs;
        If prec.reg[pinout[fn2]] <> 'Y' then
        Begin
        For i := 1 To outcount Do
         Begin
           z := 0;
           vecstr2 := vecar[i];
           For j2 := Explace[pinout[i],0] Downto 1 do
            Begin
```

BYLAAG D
SIM.PAS

```
            Inc(z);
            If explace[pinout[i],j2] =  pinout[fn2] Then
             vecstr2[z] := outs;
          End;{for}
         vecar[i] := vecstr2;
         vecstr2 := '';
       End;{for}
      End;{if}
      Vecok[j] := 1;
    End;{if}
   Vecstr2 := '';
 End{if}
Else
 Begin
   vecstr2 := '';
    z := 0;
    pin_en := 1;
    For m := enplace[pinout[j],0] downto 1 do
     Begin
       inc(z);
       insert(vecpin[enplace[pinout[j],m]],vecstr2,z)
     End;{for}
    For m := 1 to enplace[pinout[j],0] do
     begin
       If (vecstr2[m] = 'x') or (vecstr2[m] = 'z') then
        pin_en := 0;
     end;{for}
    If pin_en = 1 then
     Begin
       Done := false;
       z := 0;
       m := 1;
       tvec_en;
       If outs = '1' then
        vecpin[pinout[j]] := 'x'
       else
        Begin
          Repeat
            gotoXY(1,24);
            write('Enter condition for I/O pin.' ,pinout[j]);
            gotoXY((a - 4) + ((outcount*4) div j),k);
            vec1 := readkey;
          Until (vec1 = '0') or (vec1 = '1') or (vec1 = #27);
          if vec1 <> #27 then
```

BYLAAG D
SIM.PAS

```
                Begin
                  write(vec1);
                  vecok[j] := 1;
                  vecpin[pinout[j]] := vec1;
                End{if}
               Else
                begin
                  clrscr;
                  exit;
                end;
              end;{else}
            End;{if}
        End;{else}
    End;{for}
    j := 0;
    For i := 1 to outcount do
     j := j + vecok[i];
    If j = outcount then
     Quit := true
    Else
     for j := 1 to 10 do
      Begin
        dovec[j] := 1;
      End;{for}
Until Quit;
For j := outcount downto 1 do
 Begin
   GotoXY(a,k);
   Write(vecpin[pinout[j]]);
   a := a + 4;
 End;{for}
vecstr := '';
For j := 20 downto 1 do
 Begin
   If m2 = 1 then
    Begin
      If vecpin[j] = 'x' then
       Begin
         For i := 1 to outcount do
          Begin
            If pinout[i] = j then
             insert(vecpin[j],vecstr,1);
          End;{for}
       End;{if}
```

BYLAAG D
SIM.PAS

```
        End;{if}
       If vecpin[j] <> 'x' then
        insert(vecpin[j],vecstr,1);
     End;{for}
    result[rescount] := vecstr;
    Inc(rescount);
    Vecstr := '';
    inc(k);
    a := 11;
    GotoXY(1,24);
    write('Press Enter to change Clock pulse.');
    Readln;
    clkstatus := 1;
    vecpin[1] := '1';
    For j := 1 to outcount do
     Begin
       If Prec.Reg[pinout[j]] = 'Y' Then
        Vecpin[pinout[j]] := regpin[pinout[j]];
     End;{for}
    vecstr2 := '';
    For j := 1 to 20 do
     vecar[j] := vecstr2;
    For j := 1 to outcount do
     Begin
       If prec.proden[pinout[j]] = 'Y' then
        Begin
          If Enstatus[pinout[j]] = 'B' then
           Begin
             vecpin[pinout[j]] := 'z'
           End;{if}
        End;{if}
     End;{for}
    z := 0;
    For j := 1 to outcount do
     Begin
       If Prec.Reg[pinout[j]] <> 'Y' then
        Begin
          For m := explace[pinout[j],0] downto 1 do
           Begin
             inc(z);
             insert(vecpin[explace[pinout[j],m]],vecstr2,z)
           End;{for}
          vecar[j] := vecstr2;
          vecstr2 := '';
        End;{if}
```

BYLAAG D
SIM.PAS

```
 End;{for}
For j := 1 to 10 do
 Begin
   dovec[j] := 1;
   vecOk[j] := 0;
 End;{for}
vecstr2 := '';
Quit := false;
Repeat
  For j := 1 to outcount do
   Begin
     If prec.reg[pinout[j]] = 'Y' Then
      vecok[j] := 1
     Else
      Begin
        If Vecpin[pinout[j]] <> 'z' then
         Begin
           If vecok[j] = 0 then
            Begin
              vecstr2 :=  vecar[j];
              For i := 1 to explace[pinout[j],0] do
               Begin
                 If (vecstr2[i] = 'x') or (vecstr2[i] = 'z') then
                  Begin
                    dovec[j] := 0;
                  End;{if}
               End;{for}
            End{if}
           Else
            dovec[j] := 0;
            If dovec[j] = 1 Then
             Begin
               Done := false;
               z := 0;
               m := 1;
               fn2 := j;
               Tvec;
               vecstr2 := '';
               Vecpin[pinout[fn2]] := outs;
               For i := 1 To outcount Do
                Begin
                  z := 0;
                  vecstr2 := vecar[i];
                  For j2 := Explace[pinout[i],0] Downto 1 do
```

```
            Begin
              Inc(z);
              If explace[pinout[i],j2] =  pinout[fn2] Then
               vecstr2[z] := outs;
            End;{for}
           vecar[i] := vecstr2;
           vecstr2 := '';
         End;{for}
        Vecok[j] := 1;
      End;{if}
     Vecstr2 := '';
  End{if}
 Else
  Begin
    vecstr2 := '';
    z := 0;
    pin_en := 1;
    For m := enplace[pinout[j],0] downto 1 do
     Begin
       inc(z);
       insert(vecpin[enplace[pinout[j],m]],vecstr2,z)
     End;{for}
    For m := 1 to enplace[pinout[j],0] do
     begin
       If (vecstr2[m] = 'x') or (vecstr2[m] = 'z') then
        pin_en := 0;
     end;{for}
    If pin_en = 1 then
     Begin
       Done := false;
       z := 0;
       m := 1;
       tvec_en;
       If outs = '1' then
        vecpin[pinout[j]] := 'x'
       Else
        Begin
          Repeat
            gotoXY(1,24);
            write('Enter condition for I/O pin.' ,pinout[j]);
            gotoXY((a - 4) + ((outcount*4) div j),k);
            vec1 := readkey;
          Until (vec1 = '0') or (vec1 = '1') or (vec1 = #27);
          If vec1 <> #27 then
```

BYLAAG D
SIM.PAS

```
                    Begin
                      write(vec1);
                      vecok[j] := 1;
                      vecpin[pinout[j]] := vec1;
                    End{if}
                   else
                    begin
                      clrscr;
                      exit;
                    end;
                 end;{else}
              End;{if}
           End;{else}
        End;{else}
     End;{for}
     j := 0;
     For i := 1 to outcount do
      j := j + vecok[i];
     If j = outcount then
      Quit := true
     Else
      for j := 1 to 10 do
       Begin
         dovec[j] := 1;
       End;{for}
  Until Quit;
  GotoXY(7,k);
  write(clkstatus);
  Done := false;
  For j := 1 to 20 do    { (outcount + incount - notinc) downto 1 do}
   Begin
     z := incount;
      Repeat
        If j = totalin[z] then
         Begin
           GotoXY(a,k);
           Write(vecpin[j]);
           a := a + 4;
           done := true;
         End{if}
        else
         dec(z);
      Until done Or (z = 0);
     If z = 0 then
```

BYLAAG D
SIM.PAS

```
        Begin
          z := outcount;
           Repeat
             If j = pinout[z] then
              Begin
                GotoXY(a,k);
                Write(vecpin[j]);
                done := true;
                a := a + 4;
              End{if}
             Else
              Dec(z);
           Until done Or (z = 0);
        End;{if}
        Done := false;
     End;{for}
    vecstr := '';
    For j := 20 downto 1 do
     Begin
       If m2 = 1 then
        Begin
          If vecpin[j] = 'x' then
           Begin
             For i := 1 to outcount do
              Begin
                If pinout[i] = j then
                 insert(vecpin[j],vecstr,1);
              End;{for}
           End;{if}
        End;{if}
       If vecpin[j] <> 'x' then
        insert(vecpin[j],vecstr,1);
     End;{for}
     result[rescount] := vecstr;
     inc(rescount);
     vecstr := '';
     vecn := 1;
     inc(k);
     GotoXY(1,24);
     Write('Press Enter to change Clock pulse');
     readln;
     a := 11;
     clkstatus := 0;
     vecpin[1] := '0';
```

```
      If m2 = testn then
       Writeln
      Else
       Begin
        If m2 in regset Then
         Begin
           a := 1;
           inc(k);
           GotoXY(a,k);
           Write('Pres enter to go on. ');
           Readln;
           If n2 <> Testn Then
            Begin
              clrscr;
              PinsAll;
              vecnumber;
            End;{if}
         End;{if}
       End;{else}
    End;{for}
End;{rtvectors}

{***********************************************}
{Procedure to complete vector table for one output}
{***********************************************}
Procedure Testvecn;
 Begin
   For n2 := (n2 + 1) to testn do
    Begin
      Writeln;
      Str(n2,Onestr);
      If n2 > 9 Then
       Insert(' ',onestr,1)
      Else
       Insert('  ',onestr,1);
       Write(onestr);
       For j := 1 to explace[pinout[fn],0] + 2 Do
        Write(pinstr);
       If (n2 in vecset) or (n2 = testn) Then
       Testvectors;
    End;{for}
 End;
```

BYLAAG D
SIM.PAS

```
{*******************************************************}
{Procedure to complete vector table for all outputs of PAL}
{*******************************************************}
Procedure vecnumber;
 Begin
   For n2 := (n2 + 1) to testn do
    Begin
      Writeln;
      Str(n2,Onestr);
      If n2 > 9 Then
       Insert(' ',onestr,1)
      Else
       Insert('  ',onestr,1);
      Write(onestr);
      If Prec.status[1] = 'CLK' Then
       Write(pinstr);
      For j := 1 to (incount + outcount + 1 - notinc) Do
       Write(pinstr);
      If Prec.status[1] = 'CLK' Then
       Begin
         Writeln;
         Write('   ');
         For j := 1 to (incount + outcount + 2 - notinc) Do
          Write(pinstr);
         If (n2 in regset) or (n2 = testn) Then
          RTvectors;
       End{if}
      Else
       Begin
         If (n2 in vecset) or (n2 = testn) Then
          Tvectors;
       End;{else}
    End;{for}
 End;

{***********************}
{Procedure to sort inputs}
{***********************}
Procedure switch;
 Begin
   For i := 1 to incount do
       Begin
         For a := incount downto i do
          Begin
```

BYLAAG D
SIM.PAS

```
          If totalin[a+1] > totalin[a] then
           Begin
             m2 := totalin[a];
             totalin[a] := totalin[a+1];
             totalin[a+1] := m2;
           End;{if}
         End;{for}
       End; {for}
End;{switch}

Begin                                                    {Start of simulation procedure}
  Writeln;
    If (look = 'Y') or (look3 = 'N') Then
     Begin
       If look = 'Y' Then
        Writeln('Error ? First select a pal by pressing (P).')
       Else
        Writeln('Error ? First load a jedecfile by pressing (L).')
     End{if}
    Else
     Begin
       For j := 1 to 20 do
        Begin
          totalin[j] := 0;
          place[j] := 0;
        End;{for}
       For j := 1 to 10 do
        vecar[j] := '';
       Incount := 0;
       j := 0;
       m := 0;
       placen := 1;
       Place[0] := 0;
       onestr := '';
       For i := 1 to 80 do
        Expresar[i] := onestr;
       i := 0;
       Excount := 1;
       done := false;
       EditData := firstln;
       Getline;
       k := 0;
       Repeat
         Inc(k);
```

BYLAAG D
SIM.PAS

```
Until (Prec.Productline[k] = i) or (Prec.Productline[k] = i + 1);
Inc(m);
Pinout[m] := k;
If Prec.Productline[k] = i + 1 Then
 Begin
   noneline := '';
   Insert(oneline,noneline,1);
   delete(noneline,1,3);
   If noneline = '00000000000000000000000000000000' then
    Begin
      Enstatus[pinout[m]] := 'I';
      Dec(m);
    End{if}
   Else
    Begin
      If noneline = '11111111111111111111111111111111' then
       Begin
         Enstatus[pinout[m]] := '0';
       End{if}
      Else
       Begin
         Enstatus[pinout[m]] := 'B';
         GetExpres;
         Str(excount - 1,onestr);
         ExEnable[pinout[m],0] := onestr;
         For k := 1 to 80 do
          Begin
            ExEnable[pinout[m],k] := expresar[k];
          End;{for}
         Enplace[pinout[m],0] := placen - 1;
         For k := 1 to 20 do
          Begin
            Enplace[pinout[m],k] := place[k];
            m2 := 1;
            Donex := False;
            Repeat
              If TotalIn[m2] = Place[k] Then
               Begin
                 donex := true;
               End{if}
              Else
               Begin
                 If (m2 - 1) = Incount Then
                  Begin
```

BYLAAG D
SIM.PAS

```
                    inc(incount);
                    totalin[incount] := place[k];
                    donex := true;
                   End;{if}
                  Inc(m2);
                 End;{else}
               Until donex
             End;{for}
         End;{else}
        Excount := 1;
        placen := 1;
        For k := 1 to 20 do
         place[k] := 0;
        Place[0] := 0;
        onestr := '';
        for k := 1 to 80 do
         expresar[k] := onestr;
      End;{else}
     Getline;
     GetExpres;
   End{if}
  Else
   Begin
     GetExpres;
   End;{else}
  Repeat
    Getline;
    If i < (prec.productline[Pinout[m]] + Prec.NoOf[Pinout[m]]) Then
     Begin
       Expresar[excount] := '+';
       Inc(Excount);
     End{if}
    Else
     Begin
       Str(excount - 1,onestr);
       Expres[pinout[m],0] := onestr;
       For k := 1 to 80 do
        Begin
          Expres[pinout[m],k] := expresar[k];
        End;{for}
       Explace[pinout[m],0] := placen - 1;
       For k := 1 to 20 do                          {of ->Placen - 1 do}
        Begin
          Explace[pinout[m],k] := place[k];
```

BYLAAG D
SIM.PAS

```
    m2 := 1;
    Donex := False;
      Repeat
        If TotalIn[m2] = Place[k] Then
         Begin
           donex := true;
         End{if}
        Else
         Begin
           If (m2 - 1) = Incount Then
            Begin
              inc(incount);
              totalin[incount] := place[k];
              donex := true;
            End;{if}

          Inc(m2);
        End;{else}
      Until donex
  End;{for}
Excount := 1;
placen := 1;
For k := 1 to 20 do
 place[k] := 0;
Place[0] := 0;
onestr := '';
for k := 1 to 80 do
 expresar[k] := onestr;
k := 0;
If i    < 64 Then
 Begin
   Repeat
    Inc(k);
   Until (Prec.Productline[k] = i) or (Prec.Productline[k] = i + 1);
   Inc(m);
   Pinout[m] := k;
   If Prec.Productline[k] = i + 1 Then
    Begin
      noneline := '';
      insert(oneline,noneline,1);
      delete(noneline,1,3);
      If noneline = '00000000000000000000000000000000' then
       Begin
         Enstatus[pinout[m]] := 'I';
```

BYLAAG D
SIM.PAS

```
      Dec(m);
    End{if}
  Else
   Begin
    If noneline = '11111111111111111111111111111111' then
     Begin
       Enstatus[pinout[m]] := '0';
     End{if}
    Else
     Begin
       Enstatus[pinout[m]] := 'B';
       GetExpres;
       Str(excount - 1,onestr);
       ExEnable[pinout[m],0] := onestr;
       For k := 1 to 80 do
        Begin
          ExEnable[pinout[m],k] := expresar[k];
        End;{for}
       Enplace[pinout[m],0] := placen - 1;
       For k := 1 to 20 do
        Begin
          Enplace[pinout[m],k] := place[k];
          m2 := 1;
          Donex := False;
          Repeat
            If TotalIn[m2] = Place[k] Then
             Begin
               donex := true;
             End{if}
            Else
             Begin
               If (m2 - 1) = Incount Then
                Begin
                  inc(incount);
                  totalin[incount] := place[k];
                  donex := true;
                End;{if}
               Inc(m2);
             End;{else}
          Until donex
        End;{for}
       Excount := 1;
       placen := 1;
       For k := 1 to 20 do
```

BYLAAG D
SIM.PAS

```
                place[k] := 0;
                Place[0] := 0;
                onestr := '';
                for k := 1 to 80 do
                 expresar[k] := onestr;
               End;{else}
             End;{else}
           Getline;
         End;{if}
       End;{if}
    End;{else}
   If i < 64 Then
    Getexpres;
Until (EditData = Nil) ;
switch;
Clrscr;
Writeln('The next outputs are used.');
Writeln;
For i := 1 To m Do
 Writeln(i,'. ',pinout[i]);
Writeln(i+1,'. ','all outputs');
Repeat
  Write('Enter digit to select the output to be simulated, or Esc to quit.');
  Digit2 := Readkey;
  If digit2 = #27 Then
   Exit
  Else
   Val(Digit2,Fn,Code);
  If fn > (i + 1) Then
   Begin
     Writeln;
     Writeln('Error ??? (Enter Digit or Esc) ');
   End;{if}
Until (Fn <= (i + 1)) or (digit2 = #27);
Writeln;
Writeln;
If prec.state = 'R' then
 Write('How many clock pulses ? ')
Else
 Write('How many testvectors ? ');
Readln(testn);
Clrscr;
vecstr := '';
for j := 1 to 200 do
```

```
    result[j] := vecstr;
    firstline := nil;
    outcount := m;
    rescount := 1;
    m2 := 0;
    n2 := 0;
    If fn <= i Then
     Begin
       Pins;
       Testvecn;
     End{if}
    Else
     Begin
       If prec.state = 'R' then
        Begin
          clkstatus := 0;
          vecpin[1] := '0';
          loadreg;
          Clrscr;
        End;{if}
       pinsall;
       Vecnumber;
     End;
    Writeln;
  End;{else}
gotoXY(1,24);
Write('Press enter to go on.');
Readln;
Done := false;
Repeat
  clrscr;
  Writeln('1. Do you want to view testvectors again ? ');
  Writeln('2. Do you want a hardcopy of testvectors ? ');
  Writeln('3. Go to menu.');
  Writeln;
  Write('Press digit to select your choice :  ');
  Readln(xchoice);
  Case xChoice Of
    '1' : viewvec;
    '2' : printvec;
    '3' : done := true;
    Else
     Write('Error : try again. ');
  End;{ case}
```

BYLAAG D
SIM.PAS

```
  Until Done;
End;{simulation}

Begin                                                    {Begin simulation program}
  Get;
  vecset := [15,30,45,60,75,90,105,120,135,150,165,180,195];
  regset := [7,14,21,28,35,42,49,56,63,70,77,84,91];
  Look := 'N';
  Look2 := 'N';
  Look3 := 'N';
  Mark(heaptop);
  Firstline := Nil;
  Firstln := Nil;
  For i := 0 to 63 do
   Begin
     Prline[i] := 99;
     prodlines[i] := 99;
   End;
  Clrscr;
  Quits := False;
  Repeat
    Writeln;
    Write('L.oad, J.edec(read), P.al, R.ead(Map), S.imulate, Q.uit ? ');
    Readln(Choice);
     Case Choice Of
       'L','l' : Load;
       'J','j' : ReadMem;
       'P','p' : PALs;
       'Q','q' : Quits := True;
       'R','r' : Read_Put_in;
       'S','s' : simulation;
      Else
       Write('Error : try again. ');
     End;{case}
  Until Quits;
End.
```

LITERATUURLYS

ANDERSON,J. 1991: PLD package says bye bye to breadboard In: *Electronics world + Wireless World*, Vol 97: 925-928. November 1991.

BARWISE,M. 1987: Hardware design concepts In: *Electronics Today International*, Vol. 16, No. 8: 24-26. August 1987.

BIRKNER,J.M. en COLI,V.J. 1983: *Programmable Array Logic Handbook*, 3rd Ed. Santa Clara, Monolithic Memories Inc.

BOLTON,M. 1990: *Digital Systems Design with Programmable logic*, Wokingham, Addison-Wesley Puplishing Company.

BOSTOCK,G. 1987: *Programmable Logic Devices - Technology & Applications*, New-York, Mcgraw-Hill.

BURTON, Von L. 1990: *The Programmable Logic Device Handbook*, Blue Ridge Summit, Tab Books, Inc.

COATES,B. 1988: Upgrading from 68000 to 68020/68881 In: *Electronics & Wireless World*, Vol 94 No 1629: 665-671. July 1988

FROST,B.J. 1989a: DIY PLD In: *Electronics & Wireless World*: 499-502. May 1989.

FROST,B.J. 1989b: DIY PLD In: *Electronics & Wireless World*: 666-669. July 1989.

GREINER,J. en SCHMITZ,N. 1984: Software Aids in PAL Circuit Design, Simulation, and Verification In: *Electronic Design*: 243-250. May 1984.

HOROWITZ,P. en HILL,W. 1989: *The Art of Electronics*, 2nd Ed. Cambridge, Cambridge University Press.

JAY,C. 1986: Programmable logic design In: *Electronics & Wireless World*, Vol. 92 No. 1609: 65-69. November 1986.

JORDAAN,G.D. 1988: *Development of a Programmable Array Logic Programmer Using a Home Computer.* Ongepubliseerde MDip TECH verhandeling. Technikon O.V.S. Bloemfontein.

KLINGMAN,E.E. 1982: *Microprocessor System Design*, Vol. 2, Englewood Cliffs, New Jersey, Prentice-Hall, Inc.

LAKSHMINARAYANAN,V. 1988: Programming p.l.ds In: *Electronics & Wireless World*: 4-7. January 1988.

LAM,H. en O'MALLEY,J. 1988: *Fundamentals of Computer Engineering - Logic Design and Microprocessors*, New York,John Wiley & Sons.

LAW,A.M. en KELTON,D.W. 1991: *Simulation Modelling & Analysis*, 2nd Ed. Singapore, McGraw-Hill Inc.

LEVY,A. 1986: Programmable Logic - The Silent Revolution In: *Pulse* : 31-33. April, 1986.

LEVY,D.C. 1991: Teaching Modern Logic Design using Workstations In: *Proc. 2nd Biennial Conf*: 56-58. February 1991.

MEYER,E. 1988a: Programmable Logic Devices In: *Radio-Electronics*, Vol.59, No.2: 59-64. February 1988.

MEYER,E. 1988b: Programmable Logic Devices, Part 2 In: *Radio-Electronics*, Vol.59, No.3: 63-67. March 1988.

PROGRAMMABLE LOGIC DATABOOK, 1983: Santa Clara, National Semiconductor Corporation.

PROSSER,F.P. en WINKEL,D.E. 1987: *The Art of Digital Design An Introduction to Top Down Design*, 2nd Ed. Englewood Cliffs, New Jersey, Prentice-Hall International, Inc.

REDELINGHUIS,A, JULYAN,F.W, STEYN,B.L, BENADE,F.J.C 1985: *Kwantitatiewe Metodes vir Bestuursbesluitneming*, Pretoria, Butterworth.

SAVANT,C.J., RODEN,M.S., CARPENTER,G.L. 1991: *Electronic Design - Circuits and Systems*, 3rd Ed. Redwood City, California, Benjamin/Cummings Publishing Company, Inc.

SCHILLING,D.L., BELOVE,C., APELLEWICZ,T., SACCARDI,R.J, 1989: *Electronic Circuits - Discrete and Integrated*, New York, McGraw-Hill.

SCOTT,D.E. 1987: *An Introduction to Circuit Analysis - A Systems Approach*, Singapore, McGraw-Hill.

SMIT,J.J. 1992: *Ontwikkeling van 'n Outomatiese-toetsstelsel vir die Evaluering van Nie-programmeerbare Digitale Kringe*, Ongepubliseerde MDip TECH verhandeling. Technikon O.V.S. Bloemfontein.

SOMMERVILLE,I. 1982: *Software Engineering*, London, Addison-Wesley Publishing Company.

UFFENBECK,J. 1987: *The 8086/8088 Family: Design, Programming, and Interfacing*, Englewood Cliffs, New Jersey, Prentice-Hall International, Inc.

UNGER,S.H. 1989: *The Essence of Logic Circuits*, Englewood Cliffs, New Jersey, Prentice-Hall International, Inc.

WAKERLY,J.F. 1990: *Digital Design Principles and Practices*, Englewood Cliffs, New Jersey, Prentice-Hall, Inc.

ADDISIONELE BRONNE
(GERAADPLEEG, MAAR NIE NA VERWYS NIE)

BARWISE,M. 1987: Hardware design concepts In: *Electronics Today International*, Vol. 16, No. 1: 21-24. February 1987.

CHAKRAVARTY,D. & GOTTLIEB,E. 1986: Programmable Logic Devices In: *Pulse*: 9-11. February 1986.

CROMIE,J. 1989: EPLD Programmer Design In: *Electronics & Wireless World*: 157-160. February 1989.

FROST,B.J. 1989: DIY PLD In: *Electronics & Wireless World*: 578-581. June 1989.

HALL,D.V. 1983: *Microprocessors and Digital Logic*, 2nd Ed. Singapore, McGraw-Hill International Book Co.

HANNINGTON,S. 1990: Choosing and using Programmable Logic In: *Electronics World + Wireless World*: 619-623. July 1990.

HERGERT,D. 1991: *Turbo Pascal 6 Programming For The PC*, USA, SAMS.

IC MASTER, Vol. 1 1985: Edited by Howell,D. Garden City, New Jersey, Hearst Business Communications, Inc.

IC MASTER, Vol. 2 1985: Edited by Howell,D. Garden City, New Jersey, Hearst Business Communications, Inc.

LAKSHMINARAYANAN,V. 1988: Programming p.l.ds In: *Electronics & Wireless World*: 132-133. February 1988.

O'BRIEN,S.K. 1991: *Turbo Pascal 6 - The Complete Reference*, Berkeley, California, Osborne/McGraw-Hill.

PORAT,D.I. & BARNA,A 1979: *Introduction to Digital Techniques*, Chinchester Brisbane, John Wiley & Sons, Inc.

RILEY,J.H. 1987: *Programming using Turbo Pascal*, Boston, PWS-Kent Publishing Company.

SAVANT,C.J, CARPENTER,G.L, RODEN,M.S. 1987: *Electronic Circuit Design: An Engineering Approach,* Menlo Park, California, Benjamin/Cummings Publishing Company, Inc.

SWAN,T. 1991: *Mastering Turbo Pascal 6*, 4th Ed. Carmel, Indiana, Hayden Books.

TRIEBEL,W.A. en CHU,A.E. 1982: *Handbook of Semicondutor and Bubble Memories*, Englewood Cliffs, New Jersey, Prentice-Hall International, Inc.

WILKINS,B.R. 1986: *Testing Digital Circuits: An Introduction*, Wokingham, Van Nostrand Reinhold.