

## **Digitale systemen 2**

---

### **Programmeerbare logica met ISP componenten**

Luc Friant

Academiejaar 2004-2005

Editie (01/2004)



**Inhoud:**

|  |           |
|--|-----------|
| <b>1. Inleiding programmeerbare logica</b>                       | <b>4</b>  |
| <b>2. Voordelen bij het gebruik van PLD's</b>                    | <b>7</b>  |
| <b>3. Opbouw van programmeerbare logische bouwstenen</b>         | <b>8</b>  |
| 3.1 Bipolaire fuse-based verbindingen                            |           |
| 3.2 Floating gate verbindingen                                   |           |
| 3.3 Static-RAM verbindingen                                      |           |
| <b>4. PAL-architectuur</b>                                       | <b>10</b> |
| <b>5. De macrocellen van een PAL</b>                             | <b>12</b> |
| 5.1 De macrocellen van de 22CV10A                                |           |
| 5.2 De Security Cell   |           |
| 5.3 De elektronische handtekening                                |           |
| <b>6. MACH 4 CPLD FAMILY (Complex Programmable Logic Device)</b> | <b>19</b> |
| 6.1 Product-term array   |           |
| 6.2 Logic Allocator  |           |
| 6.3 Macrocell  |           |
| 6.4 Output Switch Matrix   |           |
| <b>7. ISP-technologie</b>  | <b>25</b> |
| 7.1 De JTAG-interface en het Boundary Scan-proces                |           |
| 7.2 Interfaces bij isp-componenten                               |           |
| 7.4 5-Wire ispJTAG Interface volgens IEEE 1149.1                 |           |
| 7.5 ISP-design en implementatie                                  |           |
| 7.6 Design flow  |           |
| 7.7 Dataformaat, JEDEC-bestand                                   |           |
| 7.8 Oefeningen   |           |
| <b>8. ABEL</b>   | <b>35</b> |
| 8.1 Algemene begrippen van ABEL                                  |           |
| 8.2 Basisstructuur   |           |
| 8.2.1 De Header  |           |
| 8.2.2 Declaraties  |           |
| 8.2.3 Beschrijving van de logica                                 |           |
| 1. beschrijving van de functie                                   |           |
| 2. beschrijving met een waarheidstabel                           |           |
| 3. beschrijving van een sequentieel systeem                      |           |
| 4. beschrijving met een toestandsdiagram                         |           |
| 8.3 Een hiërarchisch ontwerp                                     |           |
| 8.4 Oefeningen   |           |

## 1. Inleiding programmeerbare logica

In de digitale elektronica beschikt men over een brede keuze aan componenten om logische schakelingen te ontwikkelen.

De 'conventionele' IC's uit de standaard logica, ook wel discrete logica genoemd (TTL, CMOS) voeren een vaste functie uit, gedefinieerd door de fabrikant van de component. De gebruiker dient een aantal verschillende componenten met elkaar te verbinden om de gewenste schakeling op te bouwen. Deze werkwijze heeft het voordeel van zijn populariteit.

Wanneer men enkel de componenten in rekening brengt, kan ze goedkoop worden genoemd. Een nadeel is dat men, zelfs voor een relatief eenvoudige schakeling, snel een grote hoeveelheid componenten en de overeenkomstige ruimte op de gedrukte schakeling (PCB) nodig heeft.

Elke wijziging van het ontwerp resulteert in een aanpassing van de PCB layout.

De ontwikkelingstijd voor dergelijke schakelingen is vrij lang met een vanzelfsprekende invloed op de kostprijs.

Een **ASIC (Application Specific Integrated Circuit)** is een IC waarvan de functie eveneens specifiek is voor een toepassing, maar bepaald wordt door de gebruiker. ASIC's zijn opgebouwd uit een groot aantal logische cellen: poorten, flip-flop's, en zelfs meer complexe functie's zoals controllers, RAM enz.. De realisatie van een ASIC vergt echter productietechnieken die enkel door een gespecialiseerde fabrikant kunnen worden uitgevoerd.

Ondanks de hoge integratiegraad kunnen ook hier de ontwikkelingstijden vrij lang zijn, zeker indien tijdens de ontwerpfase fouten worden gemaakt en de IC's opnieuw geproduceerd moeten worden.

Om rendabel te zijn dienen deze bouwstenen in grote aantallen te worden aangemaakt.

Een programmeerbare logische bouwsteen of **PLD (Programmable Logic Device)** is een IC dat door de gebruiker zelf kan 'geconfigureerd' worden om een bepaalde logische functie uit te voeren. PLD's vervangen meestal een hele reeks discrete componenten.

Zij bestaan uit cellen samengesteld uit poorten en (eventueel) flip-flop's die via een bepaalde procedure intern aan elkaar worden gekoppeld tot een schakeling met de door de gebruiker gewenste functionaliteit. Dit proces is dikwijls omkeerbaar.

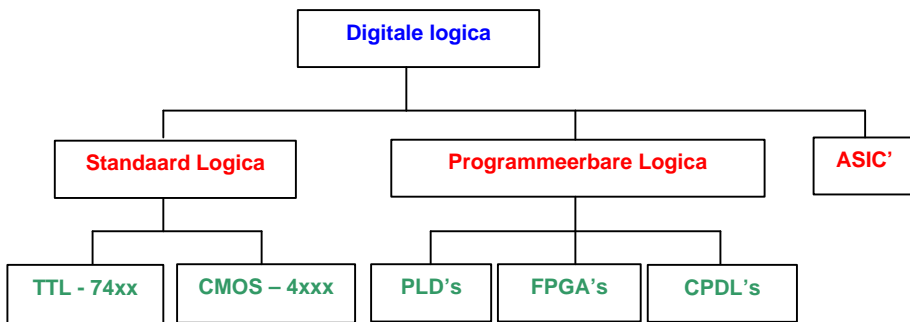
Een geconfigureerde (geprogrammeerde) PLD kan dan in zijn oorspronkelijke toestand hersteld worden. Dit betekent dat de verbindingen tussen de logische cellen 'gewist' worden en door andere kunnen vervangen worden, zodat een IC-bouwsteen met een nieuwe functie ontstaat. Dit is vooral belangrijk tijdens de ontwerpfase, zodat fouten snel kunnen gecorrigeerd worden.

Een PLD koppelt de voordelen van discrete componenten aan de voordelen van ASIC's:

- Eenvoud in ontwerp en productie, zonder tussenkomst van een gespecialiseerd fabrikant.
- De grote verscheidenheid van de op de markt zijnde componenten voor programmeerbare logica betreffende werkingssnelheid en verbruik van energie laat de gebruiker toe een gepaste keuze te maken in functie van de te realiseren toepassing(en).
- Eenvoudige gedrukte schakeling. De PCB is sterk vereenvoudigd omdat het aantal bouwstenen vermindert. Bovendien bieden PLD's een vrij grote flexibiliteit in het plaatsen van in- en uitgangspinnen. Daar de meeste functies van de schakeling intern worden gerealiseerd, kan men beginnen met het ontwikkelen van de PCB lay-out op het ogenblik dat de in- en uitgangen gekend zijn. De werkelijke details van het inwendige van een PLD kunnen dan onafhankelijk van het eindontwerp worden uitgewerkt. Noodzakelijke aanpassingen kunnen uitgevoerd worden binnen de PLD en hebben geen invloed meer op de PCB lay-out.
- Snelheid is één van de hoofdredenen waarom ontwerpers PLD-bouwstenen gebruiken. De PLD-elementen kunnen dikwijls betere prestaties leveren dan de snelste discrete logica. De looptijd van de signalen is kort vanwege de zeer korte verbindingen binnen de bouwsteen.
- Daar PLD's worden gebruikt om meerdere discrete schakelingen te vervangen, zal het energieverbruik van een ontwerp zeker lager zijn dan dat van de gecombineerde discrete componenten samen.
- De betrouwbaarheid is een gebied waar meer en meer zorg aan besteed wordt. De ontwerpen worden steeds groter en complexer en dit heeft meerdere chips voor gevolg. Meerdere chips betekenen een kleinere betrouwbaarheid van het ontwerp, er zijn "meer dingen die verkeerd kunnen gaan". Een oplossing om het aantal chips in een systeem te reduceren zal bijdragen tot een grotere betrouwbaarheid. Een benadering met programmeerbare logica kan een betrouwbaarder oplossing bieden, daar het ontwerp een kleiner aantal elementen vereist. Door een vermindering van het aantal chips kan men kleinere PCB's gebruiken. Dit vermindert overspraak en andere potentiële stoorbronnen waardoor het hele ontwerp mooier en betrouwbaarder wordt.

- Voor ieder praktisch ontwerp moeten de kosten binnen de perken blijven. De kost is bijna altijd een factor bij het overwegen van een nieuw ontwerp of bij een ontwerpverandering. Maar, een berekening van de totale ontwerpkost kan misleidend zijn als men niet alle aspecten ervan beschouwt. Veel van de kosten zijn moeilijk in te schatten. Het is bijvoorbeeld moeilijk het marktverlies van het product, te wijten aan een te late introductie, te beoordelen. De grootste voordelen tegenover een discreet ontwerp bekomt men door het feit dat een enkele PLD verschillende discrete chips kan vervangen. De ingenomen ruimte op een gedrukte schakeling wordt gemiddeld met 25% verminderd als men met PLD's werkt.
- De manier van ontwerpen wordt sterk vereenvoudigd door de ontwerpmiddelen die nu op de markt zijn. Ontwerpsoftware en programmeerbare componenten laten toe ontwerpen te implementeren met een minimum aan tijdverlies. Simulatie laat ons toe het ontwerp functioneel te testen vooraleer het element geprogrammeerd wordt.

Om al deze redenen zal de programmeerbare logica meer en meer de voorkeur krijgen boven de klassieke oplossingen met discrete componenten. Wat de verdere indeling van de programmeerbare logische bouwstenen in PLD's, **FPGA's** (Field Programmable Gate Arrays) en **CPLD's** (Complex Programmable Logic Devices ) betreft onthouden we enkel dat dit te maken heeft met de interne structuur, de oplopende inwendige complexiteit en het steeds groter wordende aantal logische cellen per IC in de vermelde volgorde van PLD, over FPGA tot CPLD. Deze indeling kan nog meer worden verfijnd en verschilt soms per fabrikant.



## 2. Voordelen bij het gebruik van PLD's

### 2.1 PLD's zijn (her)programmeerbaar



We kunnen logica ontwerpen aan de hand van een programmeertaal zoals ABEL-HDL, AHDL, VHDL, ... waardoor het ontwerp veel sneller vordert. Laten we hierbij eens een steekje vallen, dan kan de component meestal opnieuw geprogrammeerd worden. Programmeren duurt enkele seconden... Opnieuw te programmeren types vinden we terug in de window-versie (voorzien van een venstertje) of als elektrisch te wissen PAL. De types met een venstertje worden met UV-licht gewist en deze laten dan enkele minuten op zich wachten.

### 2.2 Een PLD bespaart ruimte op de printplaat



Omdat PAL technologie verschillende soorten poorten en allerhande andere logica in zich draagt, kan een volledige schakeling in één PAL worden ondergebracht.

### 2.3 Een PLD is verenigbaar met discrete componenten



Een PAL is meestal van huize uit een CMOS-component. Hij is voorzien van de nodige stuurlogica om de gekende andere technologieën te kunnen vervangen. We moeten echter wel rekening houden met zijn fan-in en fan-out.

### 2.4 We kunnen de inhoud beveiligen tegen spiekers



Een PAL biedt de mogelijkheid om een protect bit of security bit aan- of uit te zetten. Het wordt hierdoor eenvoudiger om een digitaal ontwerp te beschermen tegen ongewenst kopiëren.

### 2.5 Een PLD schakelt relatief snel



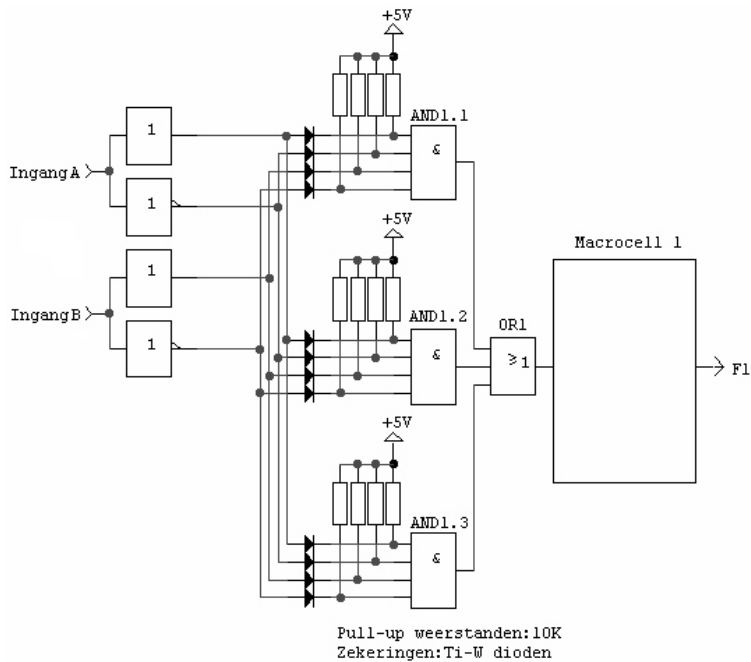
Normalerweise bouwen we met een PAL in feite enkele twee-lagen structuur op, waardoor de totale doorlooptijd van het signaal beperkt blijft. Uiteraard zal deze tijd beduidend verhogen wanneer we uitgangssignalen gaan teruglussen omdat bijvoorbeeld de gewenste functie te complex is voor verwerking in één stap. Hierdoor gaat dan ook de twee-lagen structuur verloren.

### 3. Opbouw van programmeerbare logische bouwstenen

Men kan de programmeerbare logische bouwstenen volgens allerlei criteria indelen. We beperken ons hier tot een indeling volgens de technologie waarmee de interne logische cellen met elkaar worden verbonden tot een schakeling. Bij PLD's worden de logische functies gerealiseerd door de "op elektrische wijze" doorverbinden (al dan niet omkeerbaar) van de cellen die in de chip aanwezig zijn. Dit kan echter ook gebeuren door het verbreken van bestaande, niet gewenste verbindingen. Men onderscheidt daarbij een aantal verschillende verbindingvormen.

#### 3.1 Bipolaire fuse-based verbindingen

Bij deze techniek worden de functies geprogrammeerd door alle niet gewenste verbindingen op elektrische wijze "door te smelten" zoals een zekering. De eventueel te verbreken verbindingen zijn gerealiseerd als weerstandselement of als PN-overgang (diode, transistor). Dit onomkeerbare principe vindt men onder andere terug bij PROM's en PAL's (Programmable Array Logic).



Basic PAL Architecture

### 3.2 Floating gate verbindingen

In dit geval gebruikt men *erasable* (wisbare) cellen (zoals bij bepaalde geheugen-IC's) om de interne verbindingen te realiseren. Correctie van fouten of het maken van een nieuwe toepassing met dezelfde chip is hier eenvoudig.

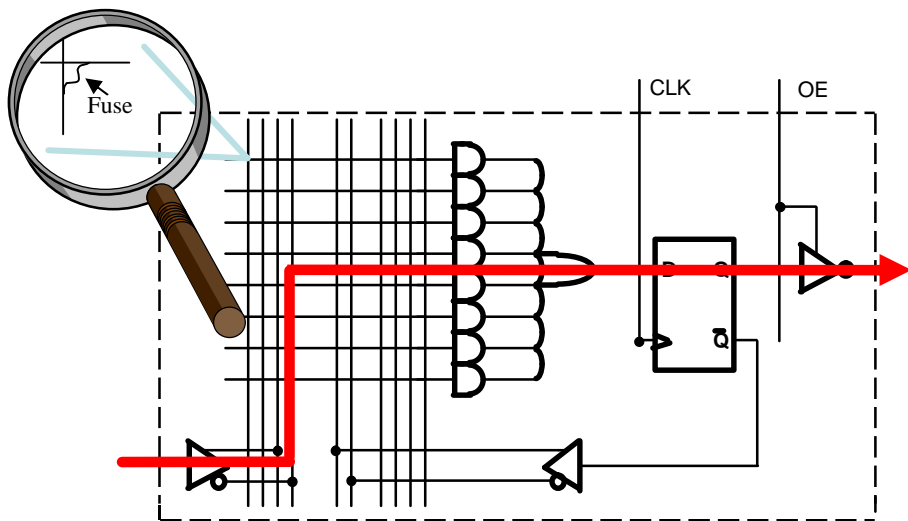
Bij de **EPLD's** (**E**rasable **P**rogrammable **L**ogic **D**evice) wordt de oorspronkelijke toestand hersteld door belichting met UV-licht (zoals bij EPROM geheugens).

Bij de **EEPLD's** (**E**lectrically **E**rasable **P**LD's) gebeurt dit langs elektrische weg (zoals bij EEPROM geheugens) en kan het (her)configureren of (her)programmeren eventueel zelfs gebeuren terwijl het IC in de (niet actieve) schakeling is opgenomen (in-circuit programming).

### 3.3 Static-RAM verbindingen

Bij sommige PLD's worden de verbindingen gerealiseerd door de toestanden (0 of 1) van statische RAM-geheugencellen. Dit geeft geheel nieuwe mogelijkheden. Door dergelijke PLD's samen met een microcontroller of microprocessor in een schakeling op te nemen, is het mogelijk deze schakeling dynamisch aan te passen aan de toepassing. De logica kan zeer snel van aard veranderen, zonder dat er ook maar één enkele wijziging aan de bedrading gebeurt. De logicaschakeling wordt geladen vanuit een (eventueel extern) processorgeheugen en zal onmiddellijk de nieuwe functie realiseren.

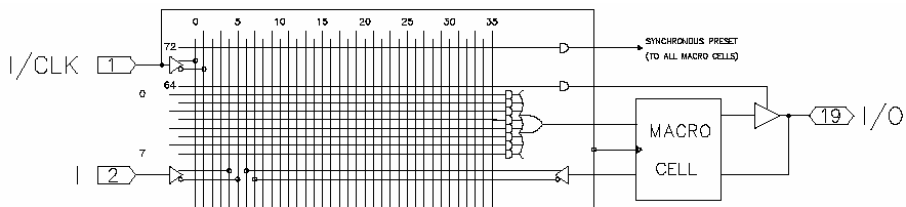
Basic PAL Architecture



#### 4. PAL-architectuur (Programmable Array Logic)

Bij nadere studie van het grote aanbod PLD's, die vaak slechts op enkele details verschillen, blijkt dat het in feite gaat om een beperkt aantal concepten en varianten daarvan. De databoeken van PLD-leveranciers tonen de interne opbouw van de componenten in de vorm van een principeschema. De kunst bestaat er in om, in deze steeds complexere schema's, een basisconcept en de daarop aangebrachte extra voorzieningen te herkennen. We beperken ons tot het veel gebruikte PAL-concept.

In onderstaande figuur lijken de EN-poorten slechts één ingangslijn te hebben.



PAL 22VC10

Deze lijn refereert echter naar de productlijn. De lijnen die loodrecht op de productlijnen van de EN-poorten staan vormen de mogelijke ingangen. Deze figuur toont een PAL-cel met 8 mogelijke producttermen met 36 mogelijke ingangen, afkomstig van 18 ingangspinnen. Er wordt hier, symbolisch, slechts één ingang van de cel voorgesteld.

Een PLD bevat steeds meerdere van dergelijke cellen, ook wel macrocellen genoemd, met gemeenschappelijke ingangslijnen.

De PAL-architectuur wordt veel toegepast, onder andere in EPLD's en EEPLD's. Bij deze architectuur is de EN-matrix (AND-array) programmeerbaar. Op elke EN-poort worden alle ingangsvariabelen aangeboden. Met behulp van de te maken of te verbreken verbindingen kan, door programmering, worden bepaald welke signalen door een EN-poort gebruikt worden. De uitgang van elke EN-poort komt op een OF-poort terecht (som van producttermen = sum of products = SOP).

De producttermen zijn dus gegroepeerd als vaste ingangen van een OF-poort. Het aantal ingangen van de OF-poort beperkt het maximale aantal te sommeren producttermen.

In een PLD met PAL-architectuur worden dus door de programmering de gewenste producttermen vastgelegd. Alle producttermen die dan aanwezig zijn worden naar buiten gevoerd.

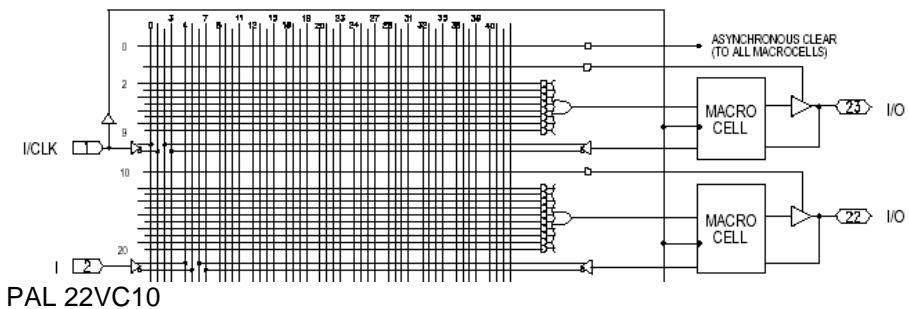
Het inverteren van ingangsvariabelen gebeurt dadelijk na het bufferen van de ingang. We hebben dus alle logische elementen in huis om een volwaardige combinatorische schakeling op te bouwen. Dikwijls is een PAL nog voorzien van instelbare macrocellen, vlak achter de of-poort, om een **uitgangstype** te kiezen of om **sequentiele circuits** te kunnen blazen.

Merk op dat de en-poorten welke we in hun geheel niet nodig hebben, de schakeling niet zullen beïnvloeden.  
Wanneer we immers alle ingangen van deze poorten intact laten, dan geldt:

$$A \cdot /A = 0$$

De desbetreffende en-poort zal hierbij dus nooit een "1" produceren op de ingang van de of-poort waarmee ze verbonden is.

We merken even op dat het aantal en-poorten per of-poort bij het type 18CV8 vast ligt op 8 en deze symmetrisch opgebouwd zijn.  
Bij het type 22CV10A is dit aantal variabel naargelang welke uitgang men kiest en de en-poorten zijn asymmetrisch opgebouwd.  
Sommige uitgangen bevatten of-poorten waar vrij veel en-poorten mee gekoppeld zijn, andere bezitten heel wat minder en-poorten.  
Wanneer de PAL asymmetrisch is opgebouwd houdt men best rekening met de complexiteit van de te realiseren functie.  
Is deze vrij groot, dan kiest men best een uitgang met veel en-poorten aan de of-poort. Is de te realiseren functie vrij eenvoudig, dan kiest men best een uitgang welke minder en-poorten bezit aan de of-poort.  
Hierdoor wordt de bezetting van de PAL efficiënter en zul je hem zo volledig mogelijk kunnen benutten.  
De ontwikkeltool ispLEVER van Lattice gebeurt dit automatisch en worden de producttermen automatisch optimaal gekozen, zodat er complexere schakeling kunnen ontworpen worden in een vrij kleine PLD die de 22CV10 is. Zie onderstaande figuur.



## 5. De macrocellen van een PAL

Meestal bevat een PAL tegenwoordig “macrocellen”. Een macrocel kan men in feite beschouwen als een digitale schakeling vlak achter de uitgang van de of-poort voor het realiseren van een uitgang. Deze macrocel bevat één of meerdere multiplexers, waarvan de adreslijnen worden bepaald door speciale programmeerbare **configuratiebits**.

De configuratiebits worden gezet of terug gezet d.m.v. zogeheten “**attributen**”. Deze vinden we in elke geschikte HDL-taal terug onder de **keywords** of **identifiers** die eigen zijn aan de HDL-taal.

Door dus deze attributen te declareren in de broncode, worden de configuratiebits juist gezet, welke de adreslijnen van de multiplexers besturen. Hierdoor creëren we in feite de gewenste signaalweg vanaf de uitgang van een of-poort tot aan de uitgangspin van onze PAL.

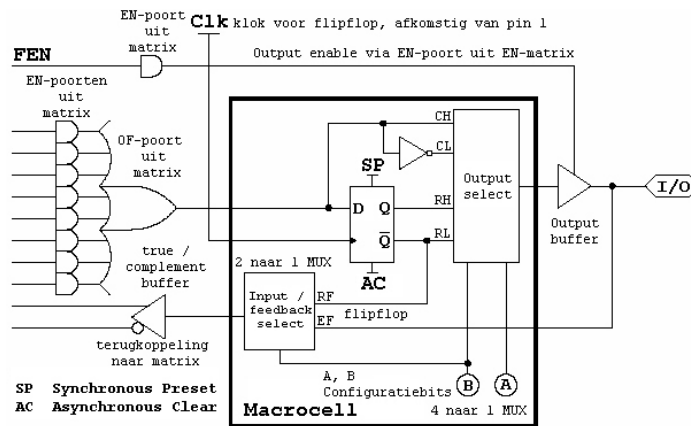
Bij het doorlopen van de gekozen signaalweg kunnen we onze functie nog verder laten beïnvloeden. Hiervoor zijn inverters, flip-flops, set, reset, terugkoppelingen naar de en-matrix, enz... voorzien in de macrocel vooraleer we onze PAL echt via een uitgangspin verlaten.

Uiteraard moeten de digitale componenten die het signaal moeten beïnvloeden wel hiervoor aanwezig zijn in onze macrocel. We dienen hiermee rekening te houden bij de keuze van een PAL.

### 5.1 De macrocellen van de 22CV10A

#### Situering van de macrocel

De eigenlijke macrocel bevindt zich in het kader met dikke zwarte rand. Links hiervan vinden we de of-poort waarvoor de macrocel een functie moet routeren. Per functie hebben we een of-poort met bijhorende macrocel. Aan de verschillende ingangen van onze of-poort vinden we de uitgangen terug van de verschillende en-poorten die met onze of-poort zijn verbonden. Het is eigenlijk vlak vóór de poorten dat we de diodematrix terugvinden die de programmeerbare zekeringen bevat.



Macrocell PAL 22VC10

### Signalen die de macrocel binnenkomen

We zien dat de uitgang van de of-poort de macrocel binnegaat en vindt ook een "output enable"-stuursignaal om de uitgangsbuffer op ingang of uitgang te configureren. De en-poort voor dit "output enable (**FEN**)" signaal kunnen we aanspreken vanuit onze broncode m.b.v. een sleutelwoord. Dit is ook zo voor "Synchronous Preset (SP)" voor het zetten van de flip-flop en de "Asynchronous Clear (AC)" voor het terugzetten van de flip-flop. Het kloksignaal (Clk), noodzakelijk voor de werking van de flip-flop, kunnen we hardwarematig aanleggen op pin 1 van de pal. Merk ook op dat een pal steeds werkt met D-flip-flops welke niet afzonderlijk te klokken zijn. Het laatsteingangssignaal dat mogelijk is, komt van onverwachte zijde, de I/O-pin. Wanneer we immers de output buffer uitschakelen met "output enable", dan kunnen we via de bijhorende I/O-pin een signaal de macrocel binnen sturen naar onze en-matrix, om daar mee te spelen bij het bepalen van uitgangstoestanden. We spreken hier van "External Feedback (EF)". Merk op dat hiervoor de ingangss-selector juist moet staan. Dit signaal komt op de en-matrix binnen via een "true / complement buffer". Hierdoor wordt elk stuursignaal naar de pal gebufferd om de signaalbron niet te belasten. De ingangss-selector laat hier enkel nog toe de gevormde functie uit de matrix rechtstreeks terug de matrix in te sturen via "registered feedback (RF)". Bij het type 18CV8, is ook nog rechtstreekse combinatorial feedback mogelijk.

### Signalen die de macrocel verlaten

Wanneer we de uitgangsbuffer niet actief zetten door "output enable" uit te zetten, dan kan er geen enkel signaal de macrocel nog uit. De uitgang van de macrocel heeft dan een zeer hoge impedantie, die het mogelijk maakt om een pal te gebruiken op een gemeenschappelijke bus zoals een adresbus, controlebus,... e.d.

Activeren we de uitgangsbuffer wél met een "output enable" signaal, dan zijn er verschillende signaalwegen mogelijk. We kunnen een combinatorisch gevormde functie door de macrocel leiden via de bovenste signaalweg van de uitgangselector. Er zal op deze wijze niets aan de functie wijzigen. Hierdoor komt het signaal naar buiten zoals het gevormd was aan de uitgang van de of-poort. De functie blijft combinatorisch en zal ook niet van teken veranderen, ze komt actief hoog naar buiten "Combinatorial High, CH". Uiteraard mag aan de nu gebruikte I/O-pin géén ingangssignaal liggen, omdat de uitgangsbuffer signalen vanuit de pal naar buiten zal sturen!

Wanneer we het combinatorisch gevormde signaal één ingang lager in de uitgangselector sturen, wordt onze functie omgekeerd nadat ze door de of-poort gevormd werd. Het signaal blijft wel combinatorisch "Combinatorial Low, CL".

We kunnen onze combinatorische functie vanuit de of-poort door een flip-flop sturen, waardoor ze één klokperiode later op de uitgangen van de flip-flop verschijnt. Kiezen we m.b.v. de uitgangselector voor de normale uitgang van de flip-flop, dan verschijnt het signaal "Registered High (RH)" aan de

uitgang van de macrocel. Kiezen we via de uitgangslector voor de inverse uitgang van de flip-flop, dan verschijnt het signaal "Registered Low (RL)" aan de uitgang van de macrocel. Merk op dat het **latchen** van de data gebeurt **op de stijgende flank** van de klok.

### Besturen van de macrocel

De macrocel kan op verschillende wijzen bestuurd of geconfigureerd worden. Om de flip-flop te gebruiken leggen we op **pin 1** van de pal een **klok-signaal** op TTL- niveau, net zoals de voedingsspanning van de pal. De frequentie van dit kloksignaal mag oplopen tot ongeveer 33 MHz. Uiteraard hangt het precieze cijfer af van het type pal dat men kiest. Merk op dat bij een pal de het **kloksignaal** meestal **gemeenschappelijk is voor alle aanwezige macrocellen** wat bij een (E)PLD dikwijls niet zo is. De data-ingang van de D-flip-flop hangt rechtstreeks aan de of-poort die de macrocel vooraf gaat. Het signaal op de D-ingang is dus via de en-matrix te programmeren. De "**synchronous preset**" en de "**asynchronous clear**" zijn te programmeren via hiervoor gereserveerde **sleutelwoorden in de HDL-taal** die men gebruikt. Het asynchroon zetten van de flip-flop kan op elk willekeurig tijdstip gebeuren. Het terugzetten gebeurt normalerwijze synchroon, dus op de volgende stijgende flank van de klok. Willen we het karakter van deze besturingslijnen wijzigen, dan moeten we dit expliciet zo programmeren.

Wanneer we de "**output enable**" willen programmeren en hierdoor dus de uitgangsbuffer willen besturen, dan kan dit ook via hiervoor **gereserveerde sleutelwoorden** in de HDL-taal. Deze "output enable" kan dynamisch gebruikt zeer handig zijn om de pal bruikbaar te maken op een gemeenschappelijke bus of bestuurbaar te maken door een busmanager. We kunnen deze controlelijn eveneens statisch gebruiken. Wanneer we immers de I/O-pin achter de macrocel als uitgang wensen te gebruiken, dan zetten we "output enable" actief. Uiteraard mag er nooit eeningangssignaal op de I/O-pin staan als "output enable" actief is! Moet de I/O-pin dienen als ingang, dan zetten we "output enable" non-actief.

Verder zijn er nog twee **configuratiebits** (A en B) beschikbaar die we eveneens kunnen programmeren vanuit onze HDL-taal. We maken hiervoor gebruik van zogenaamde **attributen**, welke **pal-afhankelijk** zijn. In de sourcecode of broncode wordt immers steeds aangegeven over welk type pal het gaat. Hierdoor kent de HDL-compiler de eigenschappen van de gebruikte pal en zal hij dus ook de beschikbare attributen kennen van de gebruikte component. Door gebruik van deze attributen worden de configuratiebits op éénduidige wijze ingesteld. Hierdoor zullen de adreslijnen van de ingangsselector en de uitgangsselector bepaald worden, waardoor beide multiplexers in een bepaalde stand worden geprogrammeerd. Daarmee is de uiteindelijke **signaalweg vanaf de of-poort tot aan de I/O-pin bepaald**. Merk op dat "output enable", "synchronous preset" en "asynchronous clear" dynamisch programmeerbaar zijn, terwijl de configuratiebits d.m.v. de attributen enkel

statisch worden gebruikt. Volgende tabel laat zien hoe het gesteld is met de twee configuratiebits van de 22CV10A:

| Configuration |   |   | Input/Feedback Select | Output Select |             |
|---------------|---|---|-----------------------|---------------|-------------|
| #             | A | B |                       | Register      | Active Low  |
| 1             | 0 | 0 | Register Feedback     |               | Active High |
| 2             | 1 | 0 |                       | Active Low    |             |
| 3             | 0 | 1 | Bi-Directional I/O    | Active Low    |             |
| 4             | 1 | 1 |                       | Active High   |             |

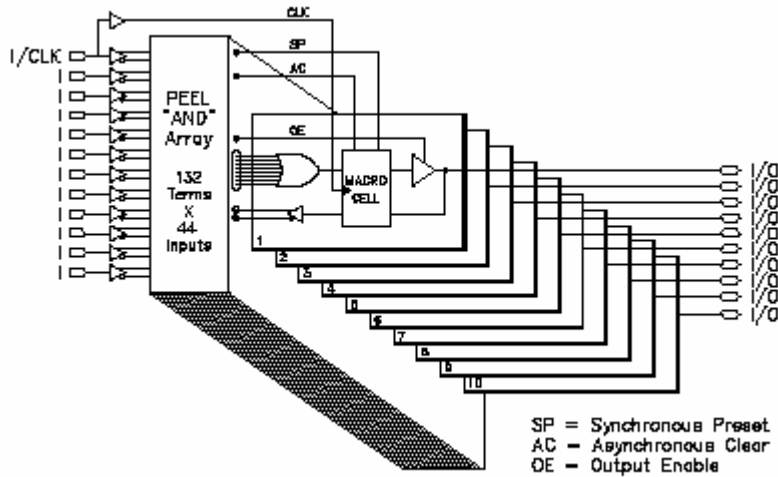
Merk op dat we beschikken over 2 bits, dus  $2^2 = 4$  mogelijke configuraties, welke hier volledig voorzien zijn. Bit "B" wordt echter voor beide multiplexers gebruikt, waardoor deze adreslijn niet volledig onafhankelijk meer werkt. We zien dat we voor de ingangselector maar één bit ("B") over houden. Hierdoor is er maar keuze tussen twee manieren van terugkoppelen: "RF of EF". Bij het type 18CV8 hadden we meer terugkoppelmogelijkheden. De uitgang blijft echter even soepel instelbaar.

### Soorten Programmable Array Logic:

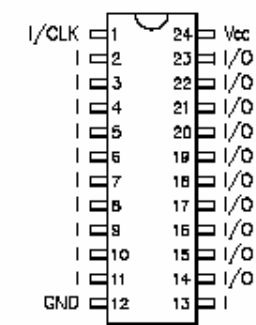
Hierover kunnen we in principe vrij kort zijn. Normalerweise geven de eerste letters van het type een identificatie van de fabrikant. De cijfers vooraan in het typenummer geeft het totaal aantal mogelijke ingangen weer. In het midden van het type krijgen we een identificatie van de inwendige technologie en de aard van de pal. De cijfers die hierop volgen geven het totaal aantal mogelijke uitgangen weer. Meestal volgt hierop een streepje, gevolgd door een cijfer. Dit laatste cijfer is meestal een kenteken voor de te verwachten propagation delay time en vormt dus onrechtstreeks een indicatie van de snelheid. Enkele voorbeelden:

- 10H8** Een ouder type met max. **10** ingangen en max. **8** uitgangen, welke actief **Hoog** zijn, nog in TTL-technologie. De "C" voor CMOS technologie ontbreekt.
- 16L8** Een ouder type met max. **16** ingangen en max. **8** uitgangen, welke actief **Laag** zijn, nog in TTL-technologie.
- 20R6** Een ouder type met max. **20** ingangen en max. **6** uitgangen, voorafgegaan door een flip-flop. **R** betekent: "Registered".
- PEEL 18CV8** Een modern type van fabrikant **ICT** met max. **18** ingangen en max. **8** uitgangen, voorzien van macrocellen (**V**) in (**C**)MOS technologie.
- PEEL 22V10-25** Een modern type van **ICT** met max. **22** ingangen, max. **10** uitgangen, voorzien van macrocellen (**V**), toch nog in TTL technologie met propagation delay van max. **25** ns.
- isp22CV10A-5** Een zelfde type, deze keer van Lattice, voorzien van **isp**-technologie, gebouwd in CMOS met **5** ns propagation delay.

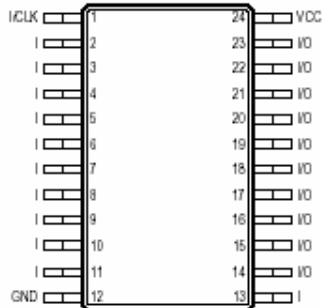
## Block diagram van de PAL 22CV10



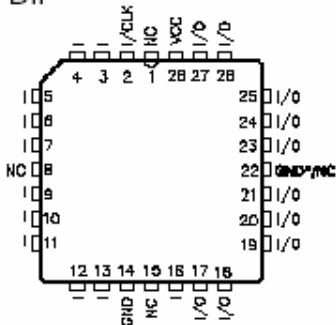
## Pin Configuration



DIP

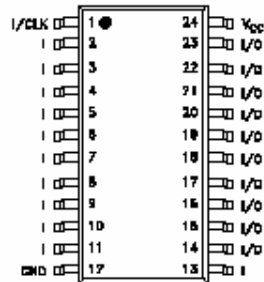


TSSOP

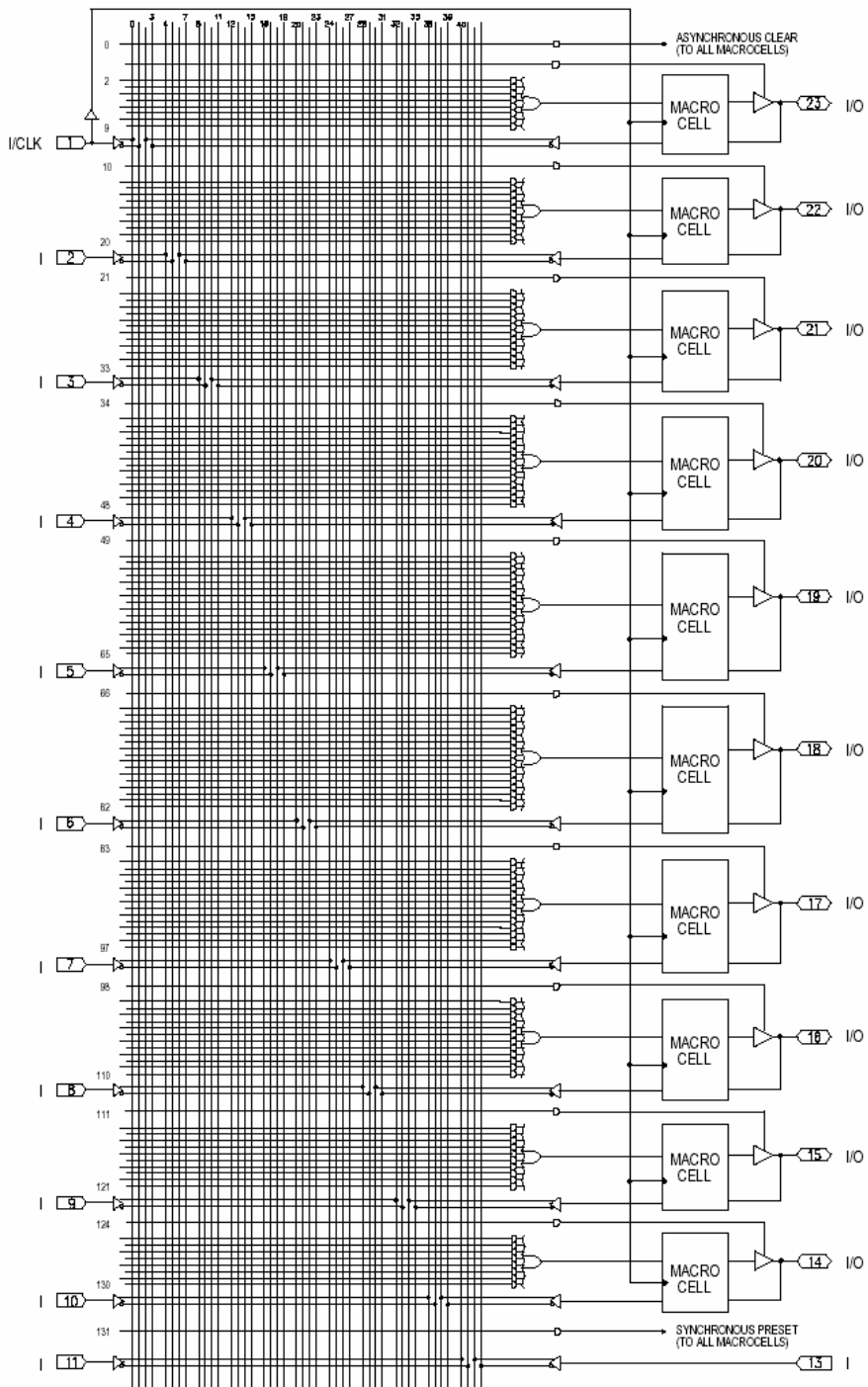


\*Optional extra ground pin for -5 speed grade.

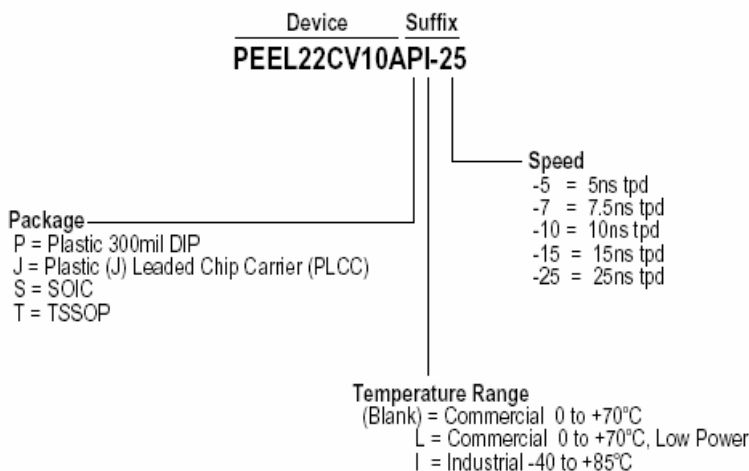
PLCC



SOIC



PAL22CV10A Logic Array Diagram



Spec's van de PEEL22CV10

## 5.2 De Security Cell

Om te voorkomen dat een uitgeleverde (dus in de applicatie gemonteerde) ispGAL wordt uitgelezen en gekopieerd, kan de zogeheten Security Cell gezet worden. Als dat eenmaal is gebeurd, kan niemand de inhoud van de ispGAL uitlezen. Deze mogelijkheid kan van pas komen wanneer het IC in een dongle als kopieerbeveiliging voor (dure) software wordt toegepast. Om die reden beveelt Lattice overigens aan om bij het programmeren van een isp-IC via een printerpoort aangesloten download-kabel een eventueel al aanwezige dongle te verwijderen – om te voorkomen dat die per ongeluk wordt gewist. Een ispGAL-IC kan, ook als de Security Cell gezet is, opnieuw worden geprogrammeerd – waarbij de Security Cell al dan niet opnieuw kan worden gezet. Door dat herprogrammeren wordt de oorspronkelijke inhoud onherroepelijk gewist: het is dus absoluut onmogelijk de inhoud van een beveiligd IC te 'kraken'.

## 5.3 De elektronische handtekening

In een ispGAL is een geheugenbereik van 64 bits gereserveerd voor de opslag van 'willekeurige' (applicatiespecifieke) gegevens – bijvoorbeeld versienummer, aanmaakdatum, klantnummer enz.. Deze 'elektronische handtekening' kan te allen tijde worden uitgelezen, ook wanneer de Security Cell gezet is. Ook kan dit geheugenbereik op elk moment opnieuw worden geprogrammeerd.

Wanneer de gehele ispGAL opnieuw wordt geprogrammeerd, dan wordt daarbij eveneens de elektronische (signature) gewist – om te voorkomen dat na het programmeren nog gegevens staan die bij een andere versie horen. Zoiets kan immers tot tragische misverstanden leiden!

## 6. MACH 4 CPLD FAMILY (Complex Programmable Logic Device)

Features:

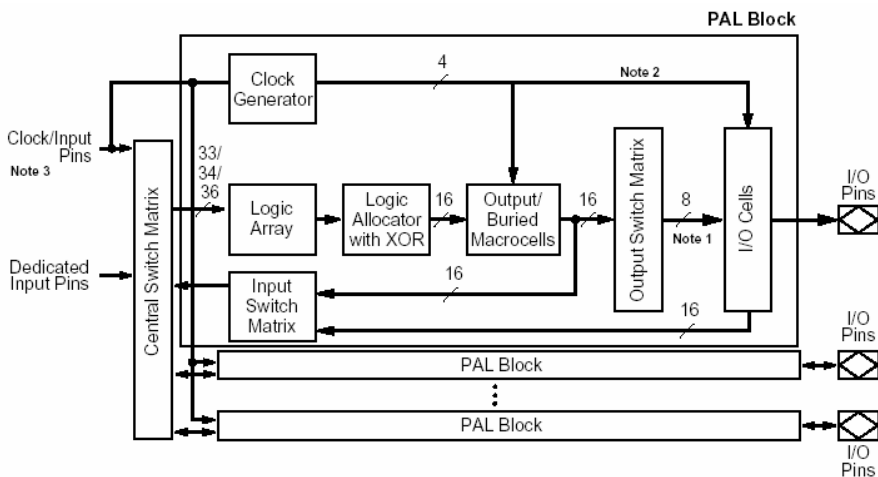
- E2CMOS 3.-V & 5-V CPLD
- Flexible architecture
- High speed
- 32 tot 256 macrocells; 32 tot 384 registers
- 44 tot 256 pin in PLCC, TQFP en BGA behuizingen
- Programmeerbaar met ispLEVER van Lattice

**M4-64/32 bevat:**

- 64 Macrocells
- 32 I/O Pins

### Functionele beschrijving:

De MACH4 family bestaat uit verschillende PAL-blocks verbonden met een Centrale Switch Matrix.



Een PAL-block bevat:

- Product-term array
- Logic allocator
- Macrocells
- Output switch matrix
- I/O cells
- Input switch matrix
- Clock generator

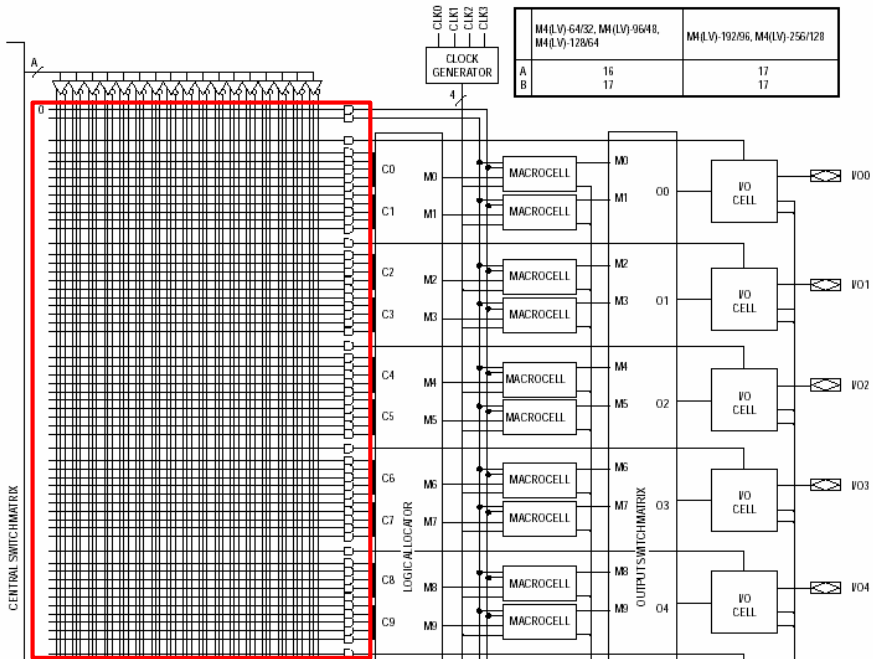
## 6.1 Product-term array:

De product-term array bevat de basislogica.

De inputs naar de AND gates komen van de central switch matrix (Table5).

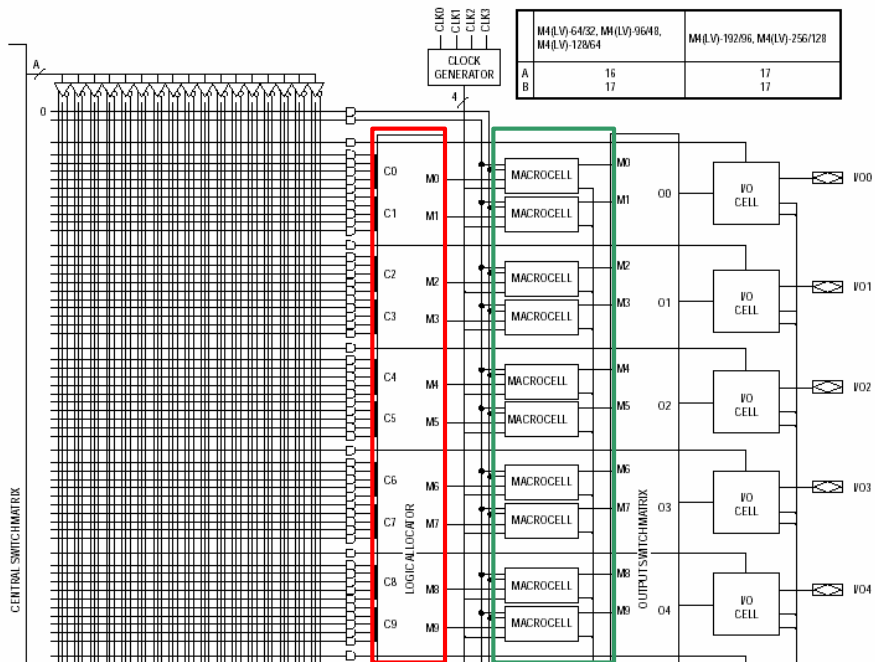
Table 5. PAL Block Inputs

| Device                      | Number of Inputs to PAL Block |
|-----------------------------|-------------------------------|
| M4-32/32 and M4LV-32/32     | 33                            |
| M4-64/32 and M4LV-64/32     | 33                            |
| M4-96/48 and M4LV-96/48     | 33                            |
| M4-128/64 and M4LV-128/64   | 33                            |
| M4-128N/64 and M4LV-128N/64 | 33                            |
| M4-192/96 and M4LV-192/96   | 34                            |
| M4-256/128 and M4LV-256/128 | 34                            |



## 6.2 Logic Allocator:

De Logic Allocator verdeelt de producttermen naar de verschillende macro-cells in product term clusters.



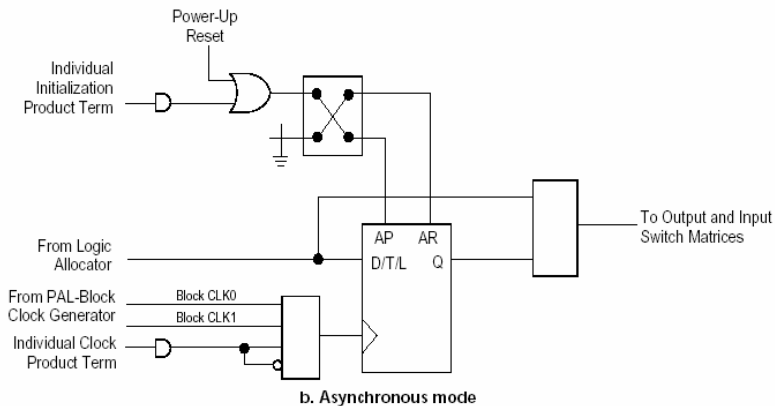
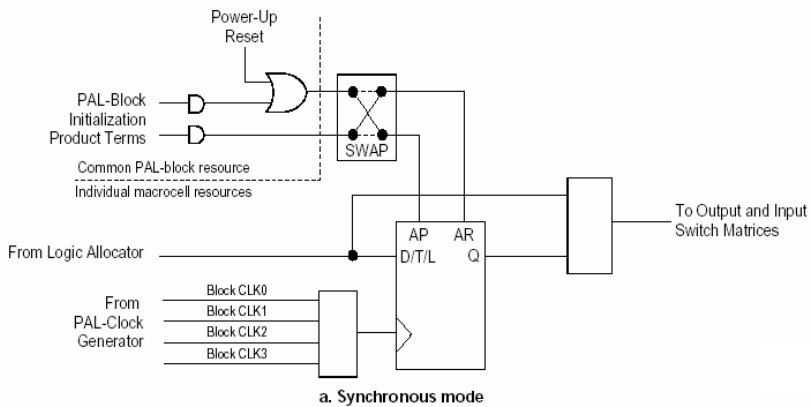
## 6.3 Macrocell:

De macrocel bestaat uit een:

- geheugencel
- routing resources
- een clock multiplexer
- initialisatie control

De macrocell werkt in twee modes, synchroon & asynchroon

## De macrocell in de synchrone- en asynchrone mode

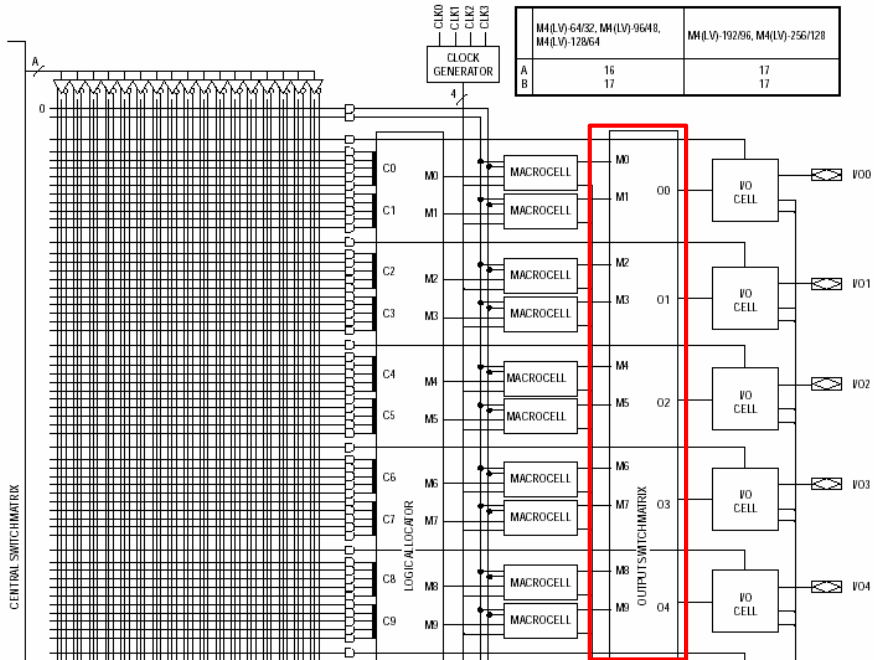


De flipflop kan geconfigureerd worden in: D-type, T-type latch, S-R register, J-K register.

De prioriteitsingangen van de flip-flops (RESET & Preset) zijn afhankelijk van de mode, synchrone- of asynchrone mode.

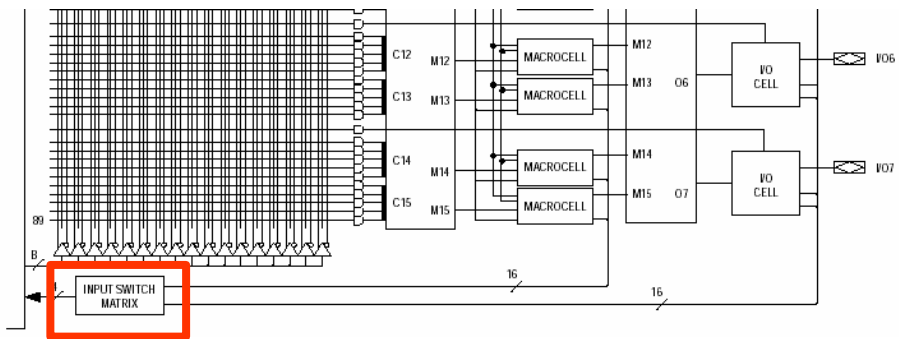
## 6.4 Output Switch Matrix:

De Output Switch Matrix stuurt de uitgang van de Macrocell naar de I/O-pins.

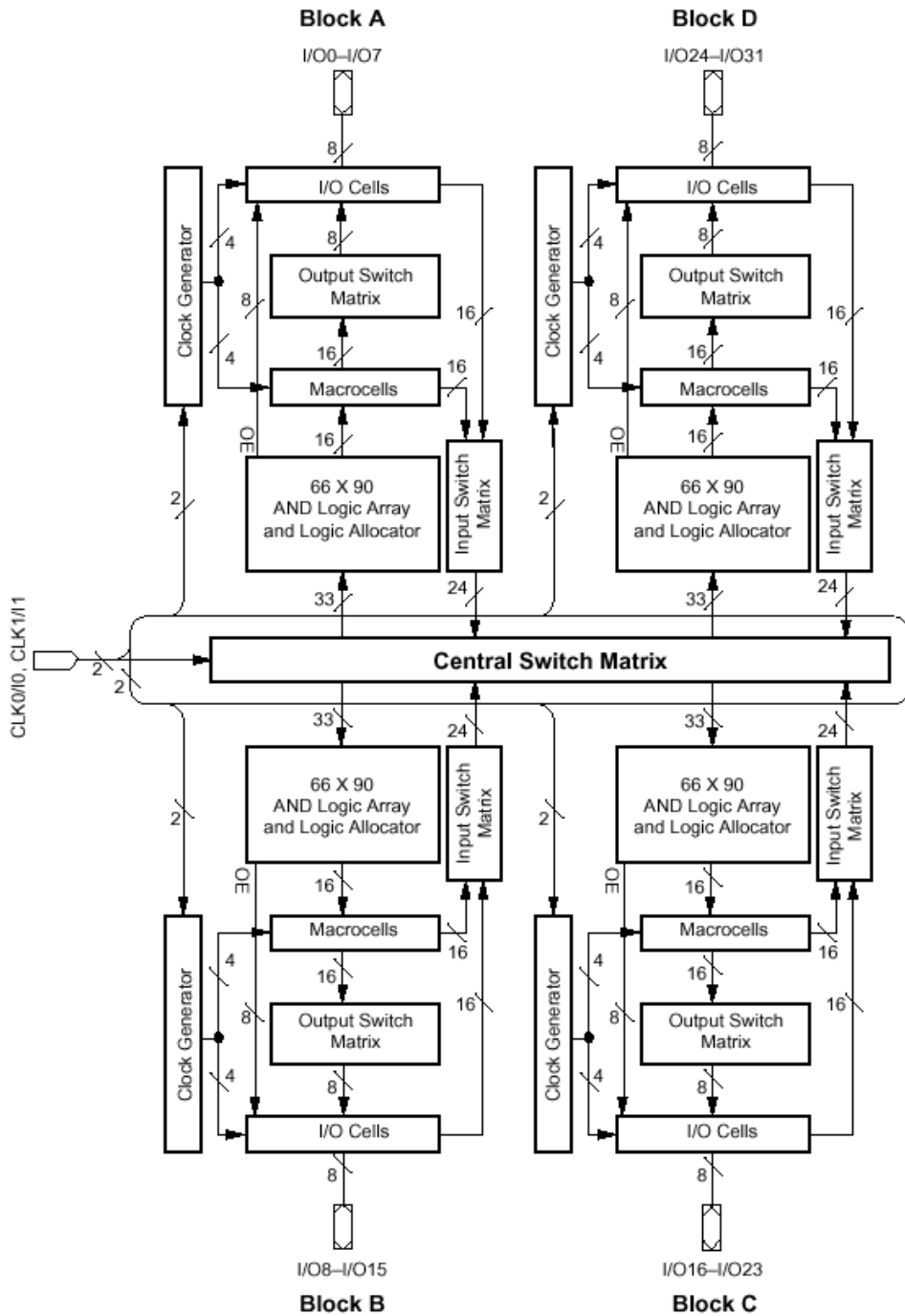


## 6.5 Input Switch Matrix:

De Input Switch Matrix optimaliseert de routing van de inputs doorheen de central switch matrix.



## Block Diagram M4-64/32



## 7. isp-technologie

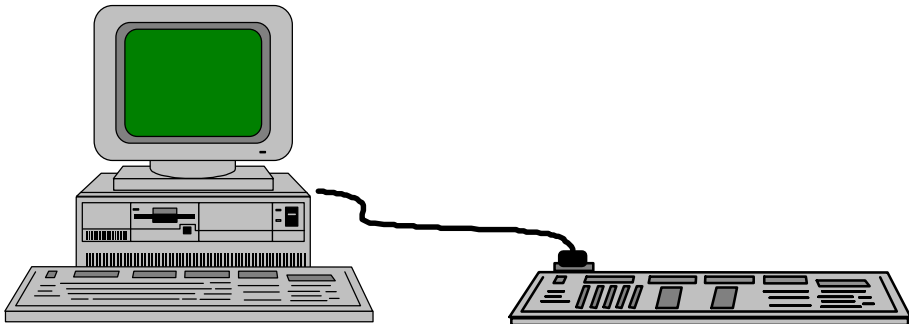
“**In System Programmability**”, de mogelijkheid om de logica van een component te voorzien of te wijzigen voor, tijdens of na aanmaak in zijn definitieve werkomgeving.

“Isp” is stilaan uitgegroeid tot een standaard in programmeerbare hardware op systeem- en componentniveau. Deze componenten zijn dus zeer snel en flexibel aan te passen, wat zeer handig is bij hardwarefouten en upgraden via floppy of per telefoonlijn. Hierdoor zijn ook op vrij eenvoudige wijze multifunctionele toestellen te ontwerpen, waar men steeds vertrekt van een zelfde hardware basisconfiguratie. De componenten worden immers meestal rechtstreeks op de printplaat bevestigd. Geen kromme of afgebroken pootjes meer, inpluggen en uittrekken van componenten uit een IC voet.

De isp-componenten zijn vervaardigd uit UltraMOS E<sup>2</sup>CMOS. Wissen en herprogrammeren gebeurt in enkele luttele seconden.

Er zijn geen speciale protocollen of toestellen nodig om deze componenten te programmeren. Een voedingsspanning van +5V is meestal wel voorhanden in b.v. een PC. Er hoeft enkel nog een 5-aderige interface voorzien te zijn om isp-componenten te programmeren op de printplaat waar ze hun normale taak zullen verrichten. Men kan zelfs de stimuli (test-vectoren) laten opwekken om de rest van de printplaat te testen.

### 7.1 De JTAG-interface en het Boundary Scan-proces



“**Boundary Scan**” volgens IEEE Standaard 1149.1 van Philips is eveneens mogelijk. Hierbij kan de test verlopen d.m.v. een “TAP” of Test Access Port. Deze bevat een seriële verbinding en enkele controlelijnen.

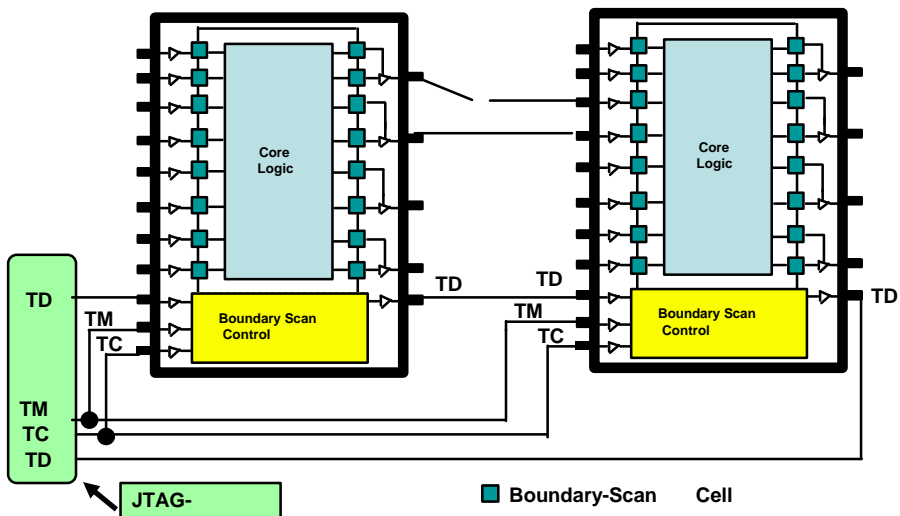
Over het algemeen zijn de componenten bedoeld om te werken in een temperatuurbereik van 0°C tot 70°C zoals de meeste commerciële componenten. Ze zouden minstens 10.000 maal te programmeren en te wissen zijn en hebben een verwachte levensduur van minstens 20 jaar. UltraMOS E<sup>2</sup>CMOS is niet vluchtig geheugen, dus geen back-up batterijen of geheugenverlies na power-down.

Er is eveneens een “**security cell**” voorzien om de inhoud te beschermen tegen kopiëren. Men kan echter zelfs bij een beveiligde component nog wel de functiecodes en de identificatie uitlezen.

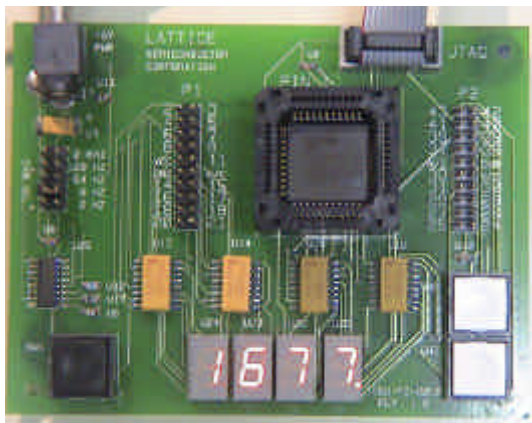
Het programmeren kan gebeuren via PC, programmeertoestel, embedded controller, ...

Bij het programmeren wordt de fuse map uit een **JEDEC-bestand** geconverteerd naar een dataformaat. Dit wordt serieel de component binnengeschoven, samen met de bijhorende adressen en commando's (b.v. "isp-STREAM")

Het programmeren gebeurt onder controle van een fabrikant afhankelijke interne tri-state statusmachine in de isp-component. Soms kan men hiervoor ook kiezen voor een IEEE Std 1149.1 1990 Boundary Scan Test Access Port interface. De isp-instructies zoals "PROGRAM", "BULK ERASE", "ADDRESS SHIFT" en dergelijke worden uitgevoerd via de instructieregisters van de component en de interface.



JTAG-interface en het Boundary Scan-proces

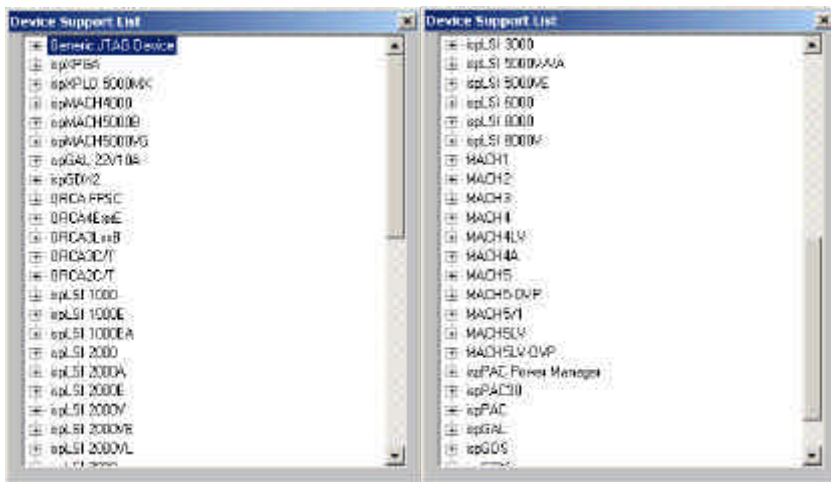


Print met JTAG-interface ISP Starter Kit

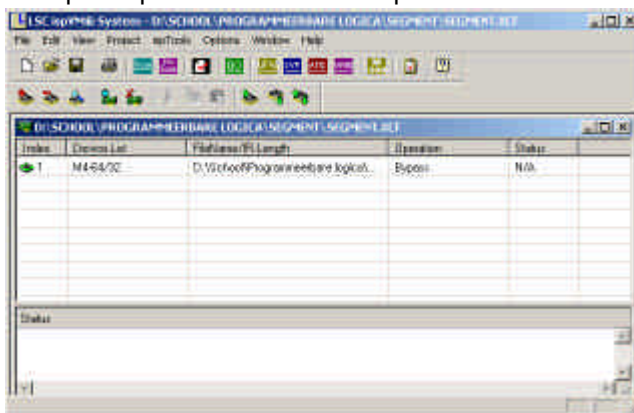
Over het algemeen worden de conventionele componenten geprogrammeerd met specifieke programmeertoestellen, omdat de programmeerspanningen meestal liggen tussen +12 tot +24V.

Bij isp-componenten zijn spanningen tussen +3.3V en +5V gangbaar. De programmeersignalen kunnen betrokken worden uit de normale TTL-poorten. Voor het programmeren van isp-componenten bestaan eenvoudige hulpmiddelen, denken we maar aan b.v. "ispVM System" van fabrikant Lattice Semiconductor Corporation. Deze gebruikt dan de parallelle poort van de PC.

Onderstaande figuur geeft een gedeeltelijk overzicht van de isp-componentenreeks van isp-fabrikant Lattice:



De isp-componentenreeks van isp-fabrikant Lattice



De interface van "ispVM System" van fabrikant Lattice met keuze van de 'Cable and I/O Port Setup'

## 7.2 Interfaces bij isp-componenten

<http://www.latticesemi.com/lit/docs/manuals/dcable.pdf>

Deze interfaces zijn 4- of 5-aderig uitgevoerd naargelang de gebruikte component. Via de seriële interface kunnen instructies als “PROGRAM”, “VERIFY” en “ERASE” worden meegegeven.

De meest voorkomende signalen zijn:

“Serial Data In” (SDI),  
“MODE select” (MODE),  
“Serial Data Out” (SDO) en  
“Serial Clock”. (SCLK)

Bij sommige uitvoeringen vinden we bovendien een laag actieve “ispEN”, om nog 4 andere controlesignalen mogelijk te maken. Is deze pin niet actief, dan worden de extra 4 pinnen als “dedicated” (vaste) ingangen beschouwd.

## 7.3 De LSC Lattice interface

Deze interface maakt gebruik van interne een tri-state statusmachine van Lattice:

|        |                 |
|--------|-----------------|
| MODE : | MODE CONTROL    |
| SDI :  | SERIAL DATA IN  |
| SDO :  | SERIAL DATA OUT |
| SCLK : | SERIAL CLOCK    |

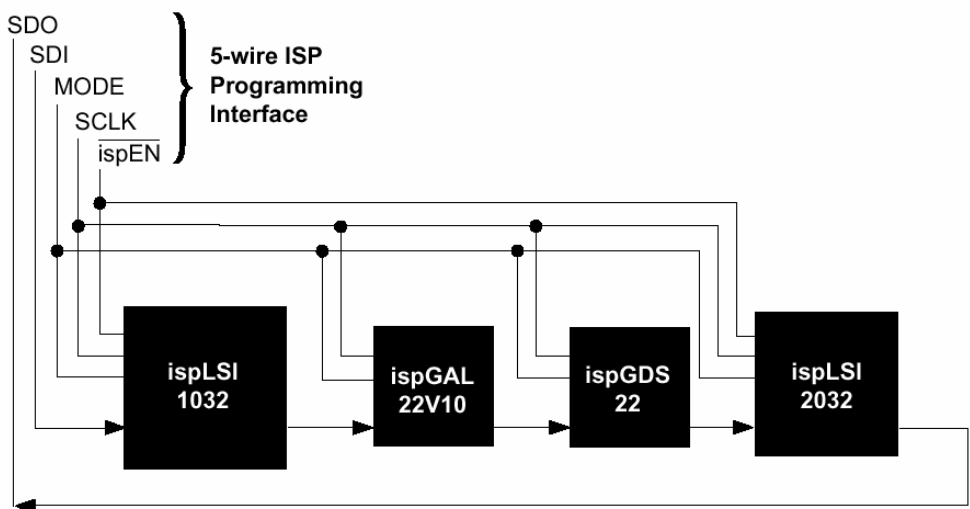
“MODE” en “SDI” zijn controle signalen.

“SDI” en “SDO” zorgen voor het dataverkeer naar en van het schuifregister.

“SCLK” levert het kloksignaal.

De grotere ispLSI-componenten bezitten ook nog een actief lage “ispEN” lijn, welke laag staat als de component in programmeermode staat.

Een hoge “ispEN” laat “user defined” functies toe tijdens de normale werking



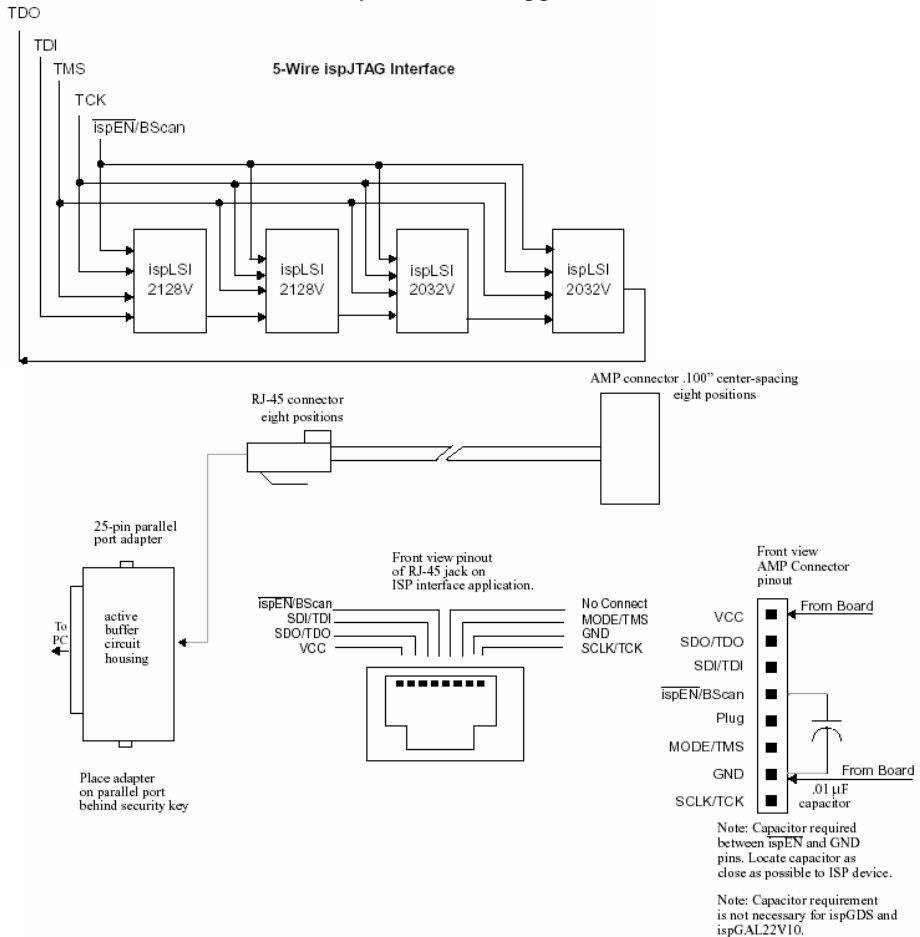
## 7.4 5-Wire ispJTAG Interface volgens IEEE 1149.1 Boundary Scan TAP (Test Access Point)

|     |   |                  |                      |
|-----|---|------------------|----------------------|
| TMS | : | Test Mode Select | (vergelijk met MODE) |
| TDI | : | Test Data In     | (vergelijk met SDI)  |
| TDO | : | Test data Out    | (vergelijk met SDO)  |
| TCK | : | Test Clock       | (vergelijk met SCLK) |

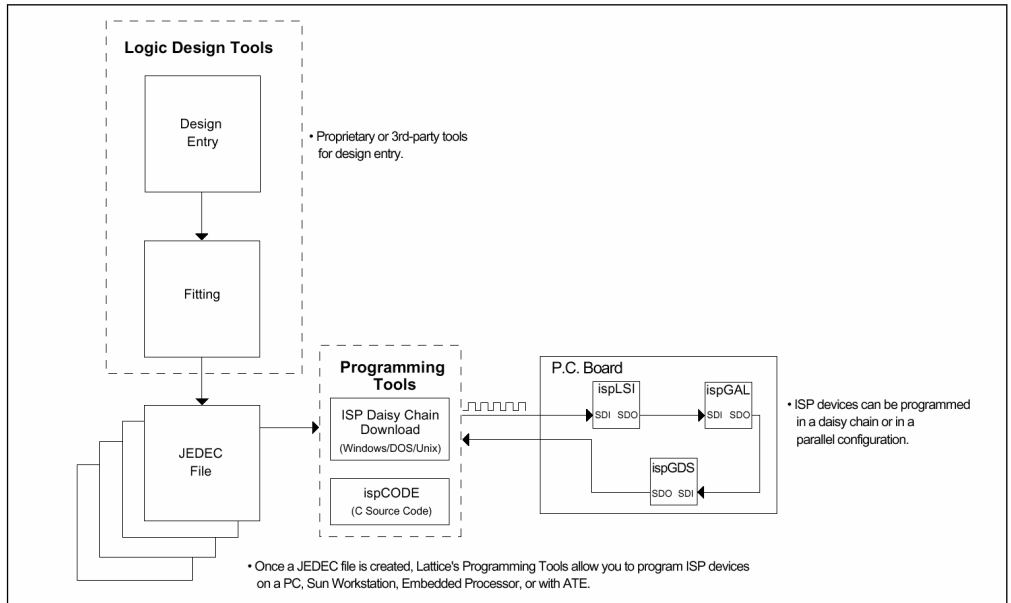
Bij de TAP wordt de statusmachine echter enkel bestuurd door "TMS".  
"TDI" wordt hier dus enkel gebruikt voor het binnenschuiven van adressen of instructies.

Ook hier bezitten de complexere ispLSI componenten een actief lage "ispEN" lijn, welke laag staat in programmeermode. Wanneer deze lijn hoog staat, kunnen "user defined" functies worden uitgevoerd.

Er kan bovendien nog een optionele actief lage "TRST" lijn aanwezig zijn, waarmee de TAP controller asynchroon teruggezet kan worden.



## 7.5 isp-design en implementatie



In bovenstaande figuur zien we hoe een ontwerp uiteindelijk op een PCB terecht komt.

Om een ontwerp in te geven, werken we met "logic design tools".

Hierin vinden we onder andere een softwarepakket om een ontwerp in te geven. Het ingeven van een schema, een gewenste golfvorm of de broncode van een ontwerp, noemt men in het Engels "design entry".

Onder de "logic design tools" hoort meestal ook een "fitter".

Dit is een software pakket dat ons ontwerp minimaliseert en eventueel opsplijst, wanneer het ontwerp binnen een gekozen component niet meer zou passen.

Deze "fitter" kan men eventueel ook de keuze van een component toevertrouwen. Meestal bezit deze "fitter" ook de software voor compilatie.

Bij compilatie wordt het schema, de golfvorm of de broncode verwerkt en omgezet naar de eigenlijke code die in de component moet worden binnengeschoven.

Het uiteindelijke resultaat hiervan vinden we meestal terug in een "JEDEC" bestand. Dit JEDEC bestand bevat alle gegevens die in de component moeten worden geprogrammeerd. Een JEDEC kan je beschouwen als normale leesbare tekens en je kunt deze dus als "tekst" naar een programmeertoe-stel doorsturen.

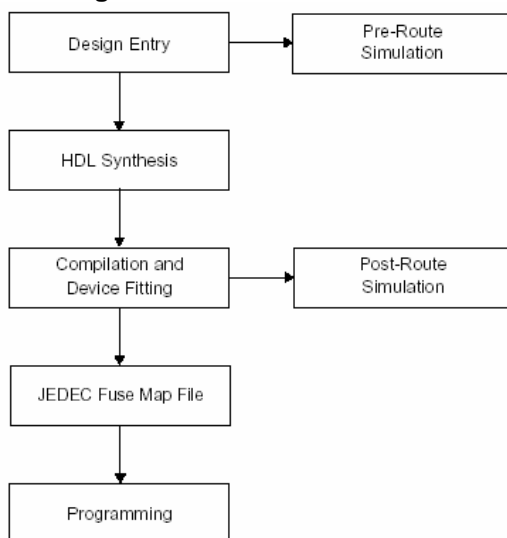
Zit het ontwerp gedefinieerd in het JEDEC-bestand, dan gaan we de "programming tools" gebruiken om de eigenlijk component te programmeren.

Bovenstaande figuur toont hiervoor twee pakketten, namelijk "ispDaisyChainDownload" en "ispCode".

**“ISPDaisyChainDownload”** is een softwarepakket dat met behulp van de parallelle poort van een PC het programmeren van isp-componenten verzorgt, welke op een PCB zijn aangebracht. Desondanks de parallelle poort gebruikt wordt, gaat het tóch om een seriële verbinding, met een volledig eigen protocol dat gebruik maakt van een “ketting”. Alle isp-componenten die voorkomen in deze “ketting” kunnen serieel worden aangesproken en geprogrammeerd of getest worden. Het pakket werkt hierbij dus op een vrij ongebruikelijke wijze met de parallelle poort van de PC. De isp-componenten kunnen echter ook op parallelle wijze worden geprogrammeerd.

**“ispCode”** is een softwarepakket dat kan gebruikt worden om de broncode van de isp-componenten te schrijven in “C”. Na compilatie zal de werkwijze verder verlopen zoals hierboven beschreven. Het “P.C.Board” stelt enkel het PCB voor dat we ontwerpen, met daarop de isp-componenten waarmee we het PCB wensen te voorzien.

## 7.6 Design flow



“Design entry” gebeurt d.m.v. “CAD software”. Na het eigenlijke ingeven van het ontwerp, gebeurt de compilatie, eveneens door de CAD software. De fitting controleert het ontwerp en probeert dit te laten passen binnen de gekozen component. Als alles in orde is wordt een JEDEC bestand (“fuse map”) aangemaakt. Het ingeven van een ontwerp kan door combinatie van ABEL of VHDL, schema, logische vergelijkingen, statusmachines of waarheidstabellen. Hierin verschilt ispLEVER van Lattice in niets van de traditionele ontwerpstechnieken. Na het aanmaken van een JEDEC bestand, kan men m.b.v. programmeersoftware de isp-componenten gaan programmeren, denken we maar aan b.v. “ispVM System” van fabrikant Lattice Semiconductor Corporation.

## 7.7 Dataformaat, JEDEC-bestand

In een programmeerbare IC kunnen elektronische verbindingen worden gemaakt of verbroken. Bij de eerste programmeerbare componenten werden deze verbindingen, die fuse ('zekeringen') werden genoemd, door middel van kortdurende stroompulsen doorgebrand. De informatie over welke fuses wel en welke niet doorgebrand moesten worden, stond toen en staat nog steeds in een zogenaamde JEDEC-bestand.

Dat bestand wordt door de programmeersoftware gegenereerd conform de schakeling van het te programmeren IC.

Bij het feitelijke programmeren van het IC wordt het JEDEC-bestand door de programmeersoftware ingelezen en doorgegeven aan de programmeerhardware die er vervolgens zorg voor draagt dat de juiste verbindingen in het programmeerbare IC worden gemaakt of verbroken.

JEDEC is een acroniem voor *Join Electronic Devices Engineering Council*; dit is een standaardiseringorganisatie binnen de EIA (Electronic Industries Association- de club van elektronica producenten in de Verenigde Staten). Door de JEDEC is het *Standaard Data Transfer Format between Data Preparation System an Programmable Logic Device Programmer* gedefinieerd, dus het data-overdrachtsprotocol tussen de programmeersoftware en de programmeerhardware. Deze standaard heet JESD3-C en kan via het Internet gratis worden gedownload van [www.jedec.org](http://www.jedec.org).

In de JEDEC-standaard 3-C is het te gebruiken dataformaat exact voorgescreven. De voorschriften betreffen een programmeerdataveld en een stesdataveld met de bijbehorende variabelen.

Aan de hand van het eenvoudige voorbeeld simple in paragraaf 3 wordt een JEDEC-bestand weergegeven.

- eerste teken STC (Start of Text)
- dataveld ASCII-karakters
- laatste teken ETX (End of Text)

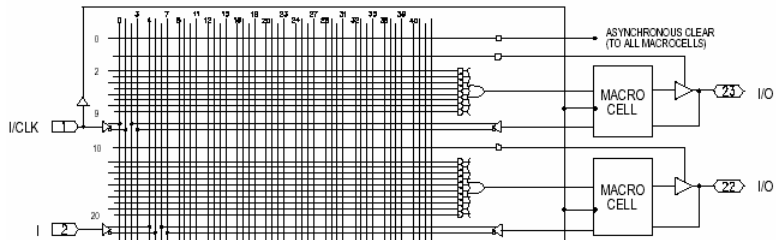
Elk volgende veld begint steeds met een bepaalde letter (identifier), gevolgd door een reeks data en afgesloten met een asterisk. Er zijn verschillende identifiers gedefinieerd. Hierna noemen we enkele belangrijke identifiers

- C** Fuse Checksum – controlegetal voor het gehele L-veld  
Dit getal controleert of de gehele dataoverdracht correct is verlopen
- L** Fuse List – bevat de te programmeren data; eerst wordt in decimale vorm het beginadres van het te programmeren geheugenbereik opgegeven ( het nummer van de eerste fuse), gevolgd door een uit '0' en '1' bestaande datareeks.
- QV** Maximum Numbers of Test Vectors
- QF** Number of Fuses in Device – aantal fuses in de de component
- QP** Number of Device Package Pins – werkelijk aantal pennen van het IC; ook ongebruikte pennen worden meegeteld
- G** Security Fuse

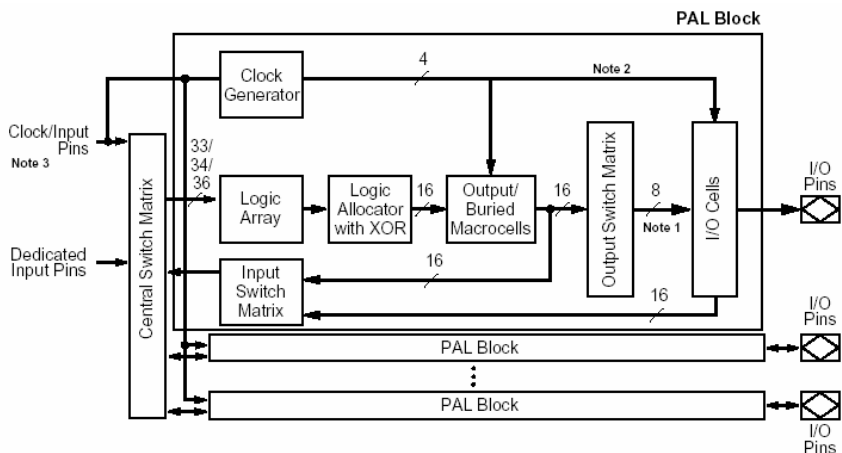


## 7.8 Oefeningen

1. Is het aantal producttermen per macrocell van programmeerbare componenten altijd constant? Geef een verklaring.
2. Tel het aantal input lijnen van onderstaande **AND/OR Logic Array**.



3. Wat is een **JTAG-interface** en wat is **Boundary Scan**?
4. Met welk dataformaat (file) worden programmeerbare componenten geprogrammeerd?
5. Wat is het verschil tussen een **PAL** en een **PLA**?
6. Wat is het verschil tussen een **PAL** en een **CPLD**?
7. Wat is het verschil tussen de **security Cell** en de **elektronische handtekening**?
8. Noem de vier ontwerpvormen in **ABEL-HDL**?
9. Waarvoor dienen **Test-Vectors** in de ABEL-HDL taal?
10. Wat kan je vertellen over onderstaand blokschema, en waarvoor dient de **Logic Allocator**



## 8. ABEL

ABEL is een taal die toelaat het gedrag, de functie, van een schakeling te beschrijven.

Men zal zeer snel kunnen inzien dat het gebruik van dit soort tools toelaat veel sneller een gewenste schakeling te realiseren. Dit onder andere omdat de minimalisatie en implementatie wordt overgenomen door software. Het enige wat de ontwerper nog moet doen is het gedrag van de schakeling te omschrijven. Dat is in wezen de echte taak van de ontwerper: creatief zijn. De repetitieve handelingen laat men best over aan machines.

Zo snel zal men kunnen zien dat het ontwerp van een sequentiële schakeling eindigt bij het toestandsdiagram, dit is na de eerste stap. Daarna kan men dit vrij eenvoudig vertalen naar een ABEL programma, waarna de implementatie automatisch verloopt. Ook voor eenvoudige schakelingen is het toepassen van Karnaugh minder belangrijk, aangezien ABEL zelf een minimalisatie zal doorvoeren.

ABEL is een taal die al lang bestaat. Ze leent zich uitstekend voor de programmatie van PAL's en nog meer geïntegreerde componenten.

Voor de lezer dit hoofdstuk verder leest, is het absoluut noodzakelijk noties te hebben van hoe een PAL er inwendig uitziet en te weten wat ongeveer de varianten zijn. Bij wijze van voorbeeld kan men bijvoorbeeld de 22V10 bestuderen. Deze component is zeer flexibel en laat heel veel PAL toepassingen toe. (zie hoofdstuk 4)

In dit hoofdstuk worden eerst enkele algemene begrippen aangehaald, zonder deze al echt te gebruiken. Daarna worden de verschillende mogelijkheden van de syntax van ABEL besproken. Merk op dat het ook in dit hoofdstuk niet de bedoeling is ABEL in al zijn mogelijkheden te beschrijven. Daarvoor bestaat er gespecialiseerde lectuur.

Waar het mogelijk is, zullen we ons beperken tot een C-achtige syntax. Andere mogelijkheden zullen slechts sporadisch aangehaald worden.

### 8.1. Algemene begrippen van ABEL

1. Er mogen tot 150 lettertekens op 1 lijn staan.
2. Identifiers, zoals namen van signalen, mogen zowel in hoofdletters als in kleine letters genoteerd worden, maar ze zijn wel 'case sensitieve': d.w.z. de signalen Klok en klok zijn twee verschillende signalen.
3. De gangbare beperkingen zoals in andere programmeertalen zijn geldig: zo mag er bijvoorbeeld geen blanco staan in de naam van een identifier. Gebruik eventueel een underscore '\_'.  
Gebruik eventueel een underscore '\_'.  
Gebruik eventueel een underscore '\_'.
4. De eigen ABEL identifiers mogen niet gebruikt worden. Deze zijn hieronder opgesomd.

|                   |           |                |
|-------------------|-----------|----------------|
| async_reset       | fuses     | state          |
| case              | goto      | state diagram  |
| declarations      | if        | state register |
| device            | in        | sync reset     |
| else              | interface | test vectors   |
| enable (obsolete) | istype    | then           |
| end               | library   | title          |
| endcase           | macro     | trace          |
| endwith           | module    | truth table    |
| equations         | node      | when           |
| external          | options   | with           |
| flag (obsolete)   | pin       |                |
| functional_block  | property  |                |

5. ABEL kent een aantal zeer belangrijke constanten, die onder andere gebruikt worden voor de bepaling van de test\_vectoren. Het is namelijk mogelijk in ABEL een verificatie van de schakeling te doen door de ingaven van enkele ingangen. ABEL zal dan de respectievelijke uitgangen berekenen.

Deze speciale constanten zijn:

- .C.: klokingang: laag – hoog – laag
- .D.: klok omlaag ingang: hoog – laag
- .F.: vlottende ingang of uitgang
- .K.: geklokte ingang: hoog – laag – hoog
- .P.: register preload
- .SVn: met  $1 < n < 10$ : ingang op een super potentiaal 2 t.e.m. 9 zetten
- .U.: klok omhoog ingang: laag – hoog
- .X.: dont'care
- .Z.: tri state waarde: hoogimpedant

Een zeer belangrijke hierbij is: .C.: deze laat de flipflops opnieuw data binnenklokken.

6. Net zoals is C is het toegelaten elk bevel te vervangen door meerdere bevelen. Het enige dat men daarvoor moet doen is deze bevelen tussen {en} plaatsen. Deze constructie noemt men een blok. Het is eveneens toegelaten blocks te nesten, dit wil zeggen een block binnen een ander block te gebruiken.

7. Elk statement wordt afgesloten door een; teken.

8. Commentaar kan men toevoegen door deze tussen dubbele aanhalings-tekens ("") te plaatsen. Een te prefereren manier is het commentaar te beginnen door twee / tekens: de rest van die lijn is dan commentaar. In C++ is dit ook het geval. Deze methode is zeer nuttig om (tijdelijk) een bevel weg te commentariëren. Een voorbeeld:

```
//Dit is commentaar tot op het einde van de lijn.
```

9. In elektronische schakelingen is het interessant om met getallen te kunnen werken die niet in het decimale talstelsel genoteerd worden. Denken we hier in het bijzonder aan het binaire en het decimale talstelsel. Men schrijft dit soort getallen als volgt:

|            |                      | voorbeeld      | decimale waarde |
|------------|----------------------|----------------|-----------------|
| $\wedge b$ | binair getal         | $\wedge b1000$ | 8               |
| $\wedge h$ | hexadecimaal getal   | $\wedge hA$    | 10              |
| $\wedge d$ | decimaal (standaard) | $\wedge d4$    | 4               |
| $\wedge$   | octaal getal         | $\wedge o10$   | 8               |

10. Een tekst staat steeds tussen enkelvoudige quote's ' '. Men gebruikt dit voorbeeld in de titel sectie of voor de naam van de module.

11. In ABEL zijn een groot aantal operatoren gedefinieerd.

#### Logische operatoren

Dit zijn de operatoren die de typische elektronisch poortfuncties toelaten.

Ze zijn:

|     |                          |
|-----|--------------------------|
| !   | invertor of not operator |
| &   | and functie              |
| #   | or functie               |
| \$  | exor functie             |
| !\$ | exnor functie            |

#### Rekenkundige operatoren

|    |           |                                      |
|----|-----------|--------------------------------------|
| -  | -A of A-B | twee complement of aftrekking        |
| +  | A+B       | optelling                            |
| *  | A*B       | vermenigvuldiging                    |
| /  | A/B       | unsigned deling                      |
| %  | A%B       | rest van een deling                  |
| << | A<<B      | schuif A over B posities naar links  |
| >> | A>>B      | schuif A over B posities naar rechts |

#### Relationele operatoren

|    |                           |
|----|---------------------------|
| == | gelijk aan                |
| != | verschillend van          |
| <  | kleiner dan               |
| >  | groter dan                |
| <= | kleiner dan of gelijk aan |
| >= | groter dan of gelijk aan  |

#### Toekenningsoperatoren

|    |   |
|----|---|
| =  | combinatorische toekenning d.w.z. zonder flipflop           |
| := | registered toekenning: gebruik voor uitgangen met flipflops |

#### Prioriteiten

Net zoals in een andere programmeertaal hebben in ABEL de operatoren ook een prioriteit. In volgende tabel wordt deze weergegeven. Merk op dat 1 de hoogste prioriteit is!

|    |                      |
|----|----------------------|
| 1: | -, !                 |
| 2: | &, <<, >>, *, /, %   |
| 3: | +, -, #, \$, !\$     |
| 4: | ==, !=, <, <=, >, >= |

12. Set. Een set is een verzameling van bij elkaar horende signalen. Men kan bijvoorbeeld denken aan de selectie of de data ingangen van een MUX. Man kan deze als volgt definiëren:

```
dataset = [data3..data0]
selectieset = [sel1..sel0]
```

Op sets kan men een groot aantal operatoren doorvoeren. Bijvoorbeeld:

```
set1 = set2 + set3 //optellen van twee sets
set4 = signal1 & signal2 & !signal3 //cocatenatie van signalen in een set
```

Let steeds goed op de dimensie van de gebruikte sets: tel alleen maar twee sets op die hetzelfde aantal elementen hebben. Er zijn uitzonderingen, maar die gaan buiten het bestek van de cursus.

## 8.2. Basisstructuur

In een ABEL-file komen steeds een aantal entiteiten weer.

Deze zijn:

| Logic        | Header  |
|--------------|---|
| Test         | <pre>Module source3 Title 'Example of a Source File'  Declarations   in1,in2,in3, clk PIN ;   all, none, other PIN ISTYPE 'reg';   in = [in1..in3] ;   out = [all,none,other] ;   C = .C.  Equations   out.clk = clk ;   all := in1 &amp; in2 &amp; in3 ;   none := !in1 &amp; !in2 &amp; !in3 ;   other := (!in1 # !in2 # !in3) &amp;            ( in1 # in2 # in3)  Test_Vectors   ([in,clk] -&gt; [out]   ([ 7, c ] -&gt; 4 ;   ([ 3, C ] -&gt; 1 ;  End source3</pre> |
| Vectors      |   |
| Description  |   |
| Declarations |   |
| End          |   |

**module**  
**title**  
**declarations**  
**equations** (functie van de PLD), met:  
 - equations  
 - state diagrams  
 - truth table  
**test\_vectors**  
**end**

Men begint met het module statement. Hierbij geeft men een vrij te kiezen naam op, in onderstaand voorbeeld source3. Het laatste statement is het end statement, namelijk source3. Bij de declaraties staan de verschillende in- en uitgangen. Van de uitgangen gaat men bijna

steeds vermelden of ze combinatorisch of registered zijn.

Bij de declaraties staan ook de definities van de sets (hier in en out) en van de constanten (hier de constante C).

Bij de declaraties staat de logica van de component. Merk op dat de klok van de registered uitgang gedefinieerd wordt (out.clk = clk).

De testvectoren vormen de input voor een simulatie.

### 8.2.1. De header

#### Module

Zoals al gemeld mag men voor de modulenaam zelf vrij een naam kiezen. Deze naam zal verder gebruikt worden bij het end statement.

#### Title

Het titel veld is optioneel, d.w.z. dat men het niet moet gebruiken. Men kan het nuttig gebruiken om de functie van de component te beschrijven. Men plaatst het steeds tussen gewone quotes.

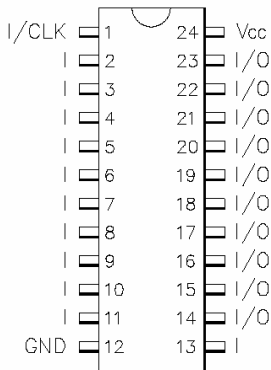
#### Device

De device is ook optioneel en wordt gebruikt om al in de ABEL file aan te geven in welke PLD men de logica wil realiseren.

### 8.2.2. Declaraties

#### Declaraties van signalen

Veronderstellen we hier bijvoorbeeld dat we van plan zijn van een 22V10 te gebruiken. De pinlayout voor de DIL (dual in line) verpakking is:



Men kan inzien dat de pennen 1 tot en met 11 en pen 13 zuiver ingangspennen zijn. De pennen 14 tot en met 23 kunnen zowel voor input als voor output gebruikt worden.

Signalen kunnen als pin of als mode gedefinieerd worden. Een node zal men gebruiken voor een intern signaal. Een toepassing hiervan kan bijvoorbeeld een minterm zijn die men meermaals kan gebruiken voor meerder uitgangen. Vanzelfsprekend is dit niet van toepassing voor een PAL, want daar is geen minterm 'sharing' mogelijk.

#### DIP

De syntax is als volgt:

Signaal1, signaal2,... PIN pin1, pin2,... ISTYPE attribuut;

Een voorbeeld: wanneer men een functie wil programmeren die drie ingangen (a, b, en clock) heeft en twee uitgangen (x en y, waarbij x een combinatorische functie is en y de uitgang van een flipflop), dan zou men kunnen schrijven:

|           |            |               |
|-----------|------------|---------------|
| a,b,clock | PIN 2,3,1; |               |
| x         | PIN 14     | ISTYPE 'com'; |
| y         | PIN15      | ISTYPE 'reg'; |

Bij de 22V10 is de gemeenschappelijke klokingang pen1. Voor de andere signalen werden willekeurige pennen gekozen. Meestal laat men de ontwerpsoftware de nummers van de pinnen kiezen.

Wanneer de PAL asymmetrisch is opgebouwd houdt men best rekening met de complexiteit van de te realiseren functie.

Is deze vrij groot, dan kiest men best een uitgang met veel en-poorten aan de of-poort. Is de te realiseren functie vrij eenvoudig, dan kiest men best een uitgang welke minder en-poorten bezit aan de of-poort.

Hierdoor wordt de bezetting van de PAL efficiënter en zul je hem zo volledig mogelijk kunnen benutten.

De ontwikkeltool ispLEVER van Lattice gebeurt dit automatisch en worden de producttermen automatisch optimaal gekozen, zodat er complexere schakeling kunnen ontworpen worden in een vrij kleine PLD die de 22CV10 is. Volgende tabel geeft de attributen die op alle programmeerbare componenten bruikbaar zijn.

|      | Attribuut | omschrijving  |
|------|-----------|---|
|      | 'com'     | combinatorische uitgang   |
|      | 'reg'     | geklokte uitgang  |
|      | 'dcc'     | elke niet gespecificeerde uitgang is don't                      |
| care |           |   |
|      | 'neg'     | elke niet gespecificeerde uitgang is 0                          |
|      |           | minimalisatie naar maxtermen                                    |
|      | 'pos'     | elke niet gespecificeerde uitgang is 1                          |
|      |           | minimalisatie naar mintermen                                    |
|      | 'retain'  | deze functie wordt niet geminimaliseerd (spikes en redundantie) |

Volgende tabel geeft de attributen die niet op alle programmeerbare componenten bruikbaar zijn.

| Attribuut | omschrijving                                  |
|-----------|---|
| 'buffer'  | geen invertor na het register (flipflop)      |
| 'invert'  | invertor na het register                      |
| 'xor'     | xor poort in programmeerbare component        |
| 'reg_d'   | D flipflop geklokt geheugenelement            |
| 'reg_g'   | D flipflop geklokt via een poort (g van gate) |
| 'reg_jk'  | JK flipflop geklokt geheugenelement           |
| 'reg_sr'  | SR flipflop geklokt geheugenelement           |
| 'reg_t'   | T flipflop geklokt geheugenelement            |

### Declaratie van constanten

Constanten zijn zinvolle namen die men gaat gebruiken voor parameters van een omschrijving.

Syntax:

Naam = uitdrukking

Vb:

T = 1;

F = 0;

C = .C.

### 8.2.3. Beschrijving van de logica

Binnen ABEL kan men de functie van een schakeling op drie verschillende manieren beschrijven. Van elk van deze manieren zullen we één of meer voorbeelden bespreken.

De mogelijkheden zijn:

1. beschrijving van de functie (combinatorisch)
2. beschrijving met een waarheidstabel (truth\_table)
3. beschrijving van een sequentieel systeem
4. beschrijving met een toestandsdiagram (using counter states)

#### 1. Beschrijving van de functie

Door gebruik te maken van de al hoger besproken logische operatoren kan men een functie volledig beschrijven. Het is hierbij niet nodig manueel te minimaliseren, aangezien ABEL Quine-Mc Cluskey kan gebruiken om dit voor de ontwerper te doen.

Een eenvoudig voorbeeld:

```
MODULE simple

TITLE 'eenvoudig ABEL voorbeeld'
//inputs
A1,A2,A3      pin 1,2,3;
N1,N2,N3      pin 4,5,6;
//outputs
AND, NAND     pin 25,26;

equations
AND           = A1 & A2 & A3;
!NAND        = N1 & N2 & N3;

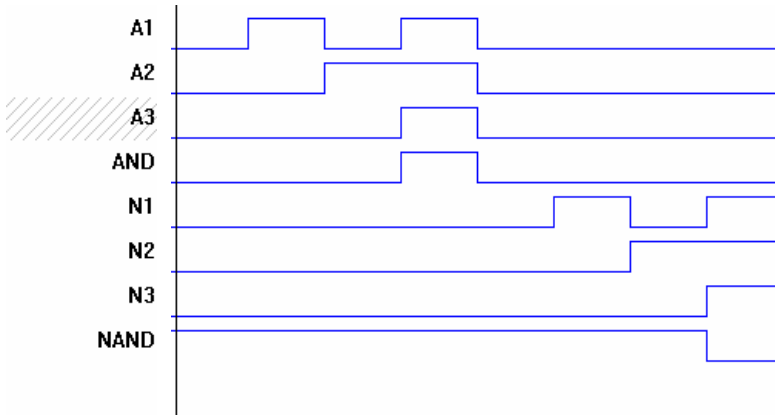
test_vectors 'test and gate'
  ([A1,A2,A3] -> AND)
  [0,0,0] -> 0;
  [1,0,0] -> 0;
  [0,1,0] -> 0;
  [1,1,1] -> 1;

test_vectors 'test nand gate'
  ([N1,N2,N3] -> NAND)
  [0,0,0] -> 1;
  [1,0,0] -> 1;
  [0,1,0] -> 1;
  [1,1,1] -> 0;

END simple
```

Een eigenaardigheid hierin is het ! teken voor het signaal !NAND. Nochtans is dit dé methode om actief lage signalen te gebruiken.

Als resultaat van de simulatie krijgt men onderstaande timing



De minimalisatie wordt hieronder weergegeven:

Title: eenvoudig ABEL voorbeeld

P-Terms Fan-in Fan-out Type Name (attributes)

| P-Terms | Fan-in | Fan-out | Type | Name (attributes) |
|---------|--------|---------|------|-------------------|
| 1/3     | 3      | 1       | Pin  | AND               |
| 3/1     | 3      | 1       | Pin- | NAND              |

=====  
 4/4 Best P-Term Total: 2  
 Total Pins: 8  
 Total Nodes: 0  
 Average P-Term/Output: 1

Equations:

$$\text{AND} = (\text{A1} \ \& \ \text{A2} \ \& \ \text{A3});$$

$$\text{NAND} = (!\text{N1} \ \# \ !\text{N2} \ \# \ !\text{N3});$$

Reverse-Polarity Equations:

$$!\text{AND} = (!\text{A1} \ \# \ !\text{A2} \ \# \ !\text{A3});$$

$$!\text{NAND} = (\text{N1} \ \& \ \text{N2} \ \& \ \text{N3});$$

Men moet eveneens in staat zijn bijvoorbeeld de klok van een flipflop te omschrijven. Daarvoor zijn er dot extensions gedefinieerd.

Onderstaande tabel geeft een overzicht van dot extensions, maar ook hier weer beperken we ons tot de architectuur onafhankelijke mogelijkheden:

| Dot extension | omschrijving             |
|---------------|--------------------------|
| .ACLR         | asynchrone clear         |
| .ASET         | asynchrone set           |
| .CLR          | synchrone clear          |
| .SET          | synchrone set            |
| .CLK          | klok                     |
| .COM          | combinatorische feedback |
| .FB           | register feedback        |
| .PIN          | pin feedback             |
| .OE           | outputs enable           |

De eerste vier mogelijkheden hebben te maken met de reset en set mogelijkheden van een flipflop. De .clk dot extensie geeft het kloksignaal van de flipflop. Daarna komen de verschillende vormen van terugkoppeling. Het is namelijk zo dat bij de programmeerbare logica bijna steeds de uitgangen ook als ingang van de schakeling kan gebruikt worden.

Men kan soms zelfs kiezen welk stuk van de uitgang als input moet gebruikt worden:

- Indien de terugkoppeling niet gespecificeerd wordt, komt deze van de pin indien de architectuur van de programmeerbare component dit toelaat. Is dit niet mogelijk, komt de feedback van een register.
- Indien men .fb specificeert, komt de terugkoppeling van de geïnverteerde uitgang van het register indien de architectuur het mogelijk maakt, anders komt deze van een pin.
- Indien men .com specificeert, komt de terugkoppeling van de uitgang van de or poort. Indien dit weer niet kan volgens de architectuur, komt de terugkoppeling van de pin.
- Indien men .q specificeert, komt de terugkoppeling van een register-uitgang indien de architectuur het mogelijk maakt, anders komt deze van een pin.
- Indien men .d specificeert, komt de terugkoppeling van de ingang van het register. Merk op dat dit zelden mogelijk is!
- De .oe extensie is ook zeer belangrijk: men kan er een uitgang mee hoogimpedant maken.

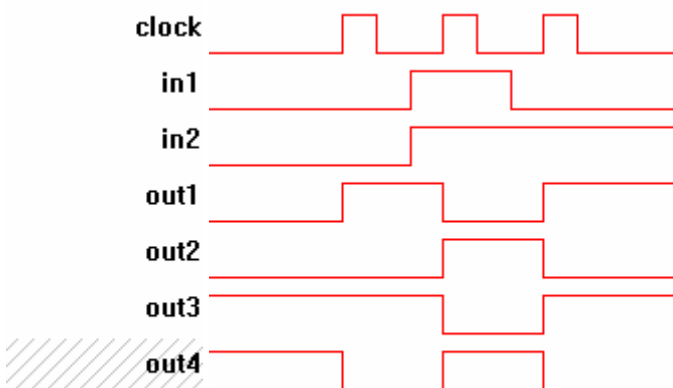
### Actief lage en hoge signalen.

Een actief laag signaal wordt gedefinieerd door een '!' voor de naam van het signaal bij de declaratie van dat signaal.

Voorbeeld:

```
MODULE actlow
TITLE 'voorbeeld actief lage signalen'
//constants assignments
c, x, H, L = .c., .x., 1, 0;
//inputs
clock pin 1;
in1 pin 2;
in2 pin 3;
//outputs
!out1 pin 23 ISTYPE 'reg,buffer'; //actief laag signaal
out2 pin 22 ISTYPE 'reg,buffer'; //actief hoog signaal
!out3 pin 21 ISTYPE 'reg,invert'; //actief laag signaal
out4 pin 20 ISTYPE 'reg,invert'; //actief hoog signaal
//set declaration
outputs = [out1, out2, out3, out4];
equations
out1 := in1 & in2;
out2 := in1 & in2;
out3 := in1 & in2;
out4 := in1 & in2;
outputs.clk = clock; // klok outputregister
test_vectors
([clock, in1, in2] -> [out1, out2, out3, out4])
[0, x, x] -> [x, x, x, x];
//deze vector wordt gebruikt om de flipflops te resetten
//de uitgangen out3 en out4 zijn 1 omdat er een invertor na het
//register staat
[c, 0, 0] -> [L, L, L, L];
[c, 1, 1] -> [H, H, H, H];
[c, 0, 1] -> [L, L, L, L];
END actlow
```

De resultaten van de simulatie van actlow:



## 2. Beschrijving van de logica met een waarheidstabel. (truth\_table)

```

MODULE toep4
TITLE '7 segment display decoder';
"
"      a
"      ---      BCD-to-seven-segment decoder: zoals de 74LS49
"      f|g|b
"      ---      segment identificatie
"      e|d|c
"      ---
jedec device 'GAL22V10B-10LJ';
d3,d2,d1,d0 PIN 4,5,6,7;
a,b,c,d,e,f,g PIN 17,18,19,20,21,23,24 ISTYPE 'COM,NEG';

BCD      = [d3,d2,d1,d0];
LED      = [a,b,c,d,e,f,g];
on,off,x = 0,1,.x;          "common anode LED's

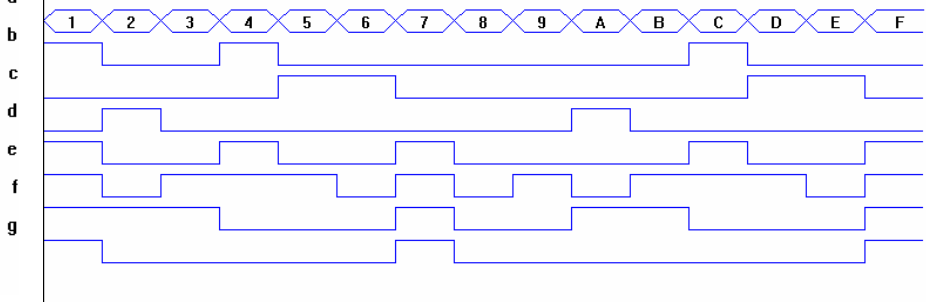
@DCSET                                "don't care set
TRUTH_TABLE (BCD -> [ a , b , c , d , e , f , g ])
0 -> [ on, on, on, on, on, on, off ];
1 -> [ off, on, on, off, off, off, off ];
2 -> [ on, on, off, on, on, off, on ];
3 -> [ on, on, on, on, off, off, on ];
4 -> [ off, on, on, off, off, off, on ];
5 -> [ on, off, on, on, on, off, on ];
6 -> [ on, off, on, on, on, on, on ];
7 -> [ on, on, on, off, off, off, off ];
8 -> [ on, on, on, on, on, on, on ];
9 -> [ on, on, on, on, off, on, on ];
TEST_VECTORS (BCD -> [ a , b , c , d , e , f , g ])
1 -> [ off, on, on, off, off, off, off ];
2 -> [ on, on, off, on, on, off, on ];
3 -> [ on, on, on, on, off, off, on ];
4 -> [ off, on, on, off, off, off, on ];
5 -> [ on, off, on, on, off, on, on ];
6 -> [ on, off, on, on, on, on, on ];
7 -> [ on, on, on, off, off, off, off ];
8 -> [ on, on, on, on, on, on, on ];
9 -> [ on, on, on, on, off, on, on ];
10 -> [ x, x, x, x, x, x, x ];
11 -> [ x, x, x, x, x, x, x ];
12 -> [ x, x, x, x, x, x, x ];
13 -> [ x, x, x, x, x, x, x ];
14 -> [ x, x, x, x, x, x, x ];
15 -> [ x, x, x, x, x, x, x ];

END

```

BCD

a De resultaten van de simulatie van 7 segment display decoder:



De minimalisatie van het 7 segment display decoder wordt hieronder weer-  
gegeven:

Title: 7 segment display decoder

P-Terms Fan-in Fan-out Type Name (attributes)

| P-Terms | Fan-in | Fan-out | Type | Name (attributes) |
|---------|--------|---------|------|-------------------|
| 2/4     | 4      | 1       | Pin  | a                 |
| 2/3     | 3      | 1       | Pin  | b                 |
| 1/3     | 3      | 1       | Pin  | c                 |
| 3/5     | 4      | 1       | Pin  | d                 |
| 2/2     | 3      | 1       | Pin  | e                 |
| 3/4     | 4      | 1       | Pin  | f                 |
| 2/4     | 4      | 1       | Pin  | g                 |

=====  
15/25 Best P-Term Total: 15  
Total Pins: 11  
Total Nodes: 0  
Average P-Term/Output: 2

#### Equations:

$$a = (!d0 \& !d1 \& d2 \# d0 \& !d1 \& !d2 \& !d3);$$

$$b = (!d0 \& d1 \& d2 \# d0 \& !d1 \& d2);$$

$$c = (!d0 \& d1 \& !d2);$$

$$d = (d0 \& d1 \& d2 \# !d0 \& !d1 \& d2 \# d0 \& !d1 \& !d2 \& !d3);$$

$$e = (d0 \# !d1 \& d2);$$

$$f = (d0 \& d1 \# d1 \& !d2 \# d0 \& !d2 \& !d3);$$

$$g = (d0 \& d1 \& d2 \# !d1 \& !d2 \& !d3);$$

#### Reverse-Polarity Equations:

$$!a = (d1 \# d3 \# d0 \& d2 \# !d0 \& !d2);$$

$$!b = (!d2 \# d0 \& d1 \# !d0 \& !d1);$$

$$!c = (d0 \# !d1 \# d2);$$

$$!d = (d3 \# !d0 \& d1 \# !d0 \& !d2 \# d1 \& !d2 \# d0 \& !d1 \& d2);$$

$$!e = (!d0 \& d1 \# !d0 \& !d2);$$

$$!f = (d3 \# !d0 \& !d1 \# !d0 \& d2 \# !d1 \& d2);$$

$$!g = (d3 \# !d0 \& d2 \# !d1 \& d2 \# d1 \& !d2);$$

Men kan ook combinatorische schakelingen ontwerpen met de:  
**When ... then ... else** structuur.

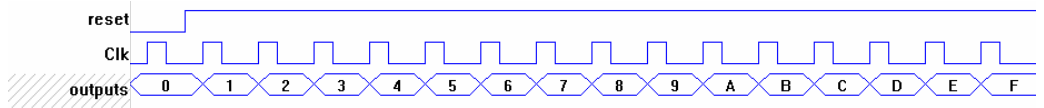
Voorbeeld:

```
WHEN (A=='0') THEN B=C;  
ELSE WHEN (A=='1') THEN B=D;  
ELSE B='0';
```

### 3. Beschrijving van een sequentieel systeem

**Ontwerp van een 4 bit binaire counter met een asynchrone clear.**

```
MODULE teller  
TITLE 'teller'  
"Binaire 4-bit binaire teller  
"Constants  
    C = .c.;  
"inputs  
    Clk, reset pin;  
"outputs  
    Q3..Q0 pin_istype 'reg,buffer';  
    outputs=[Q3..Q0];  
Equations  
    outputs.Clk = Clk;  
    outputs:=outputs+1;  
    outputs.aclr = !reset;//asynchrone reset  
Test_Vectors ([Clk,reset]->[Q3,Q2,Q1,Q0])  
    [C, 0 ] -> [0, 0, 0, 0]; //reset  
    [C, 1 ] -> [0, 0, 0, 1]; //count  
    [C, 1 ] -> [0, 0, 1, 0];  
    [C, 1 ] -> [0, 0, 1, 1];  
    [C, 1 ] -> [0, 1, 0, 0];  
    [C, 1 ] -> [0, 1, 0, 1];  
    [C, 1 ] -> [0, 1, 1, 0];  
    [C, 1 ] -> [0, 1, 1, 1];  
    [C, 1 ] -> [1, 0, 0, 0];  
    [C, 1 ] -> [1, 0, 0, 1];  
    [C, 1 ] -> [1, 0, 1, 0];  
    [C, 1 ] -> [1, 0, 1, 1];  
    [C, 1 ] -> [1, 1, 0, 0];  
    [C, 1 ] -> [1, 1, 0, 1];  
    [C, 1 ] -> [1, 1, 1, 0];  
    [C, 1 ] -> [1, 1, 1, 1];  
END teller
```



Merk op dat er wordt gebruik gemaakt van een dot extension .aclr.  
De software kiest zelf de pinnen op de component, bv de PAL 22V10. De  
pinnen worden zodanig gekozen dat de PAL optimaal wordt gebruikt.



#### **4. Beschrijving van de logica met een toestandsdiagram**

Bij het ontwerpen van sequentiële schakelingen is het de gewoonte om gebruik te maken van een toestandsdiagram.

De volgende syntaxen kunnen gebruikt worden:

##### “IF – THEN – ELSE in state diagrams”

```
STATE S0:  
  IF (A=='0') THEN S1;  
  ELSE IF (A=='1') THEN S0;  
  ELSE S9;
```

##### “IF – THEN – WITH - ELSE in state diagrams”

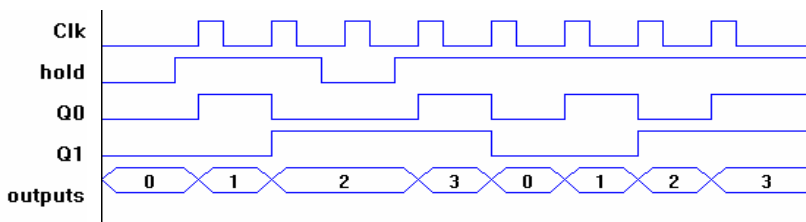
```
STATE S0:  
  IF (A=='0') THEN S1 WITH B=1;  
  ELSE IF (A=='1') THEN S0 WITH B=0 ;  
  ELSE B='0' WITH B=C;
```

##### “CASE in state diagrams”

```
STATE S0:  
  CASE ( A==0) :S1;  
    ( A==1) :S0;  
  ENDCASE;
```

## Ontwerp van een 2bit bin. teller met hold-input door gebruikt te maken van counter-states

```
MODULE cnt2
TITLE 'cnt2'
"2bit-counter met hold-input
"Constants
    C,x,H,L=.c...x..1,0;
"inputs
    Clk, hold      pin;
"outputs
    Q1,Q0         pin  istype 'reg';
"counter States
    S0 = ^b00;"binaire nul
    S1 = ^b01;"binaire één
    S2 = ^b10;"binaire twee
    S3 = ^b11;"binaire drie
"set
    OUTPUTS = [Q1,Q0];
@DCSTATE //gebruik deze directive in statediagrams
         //alle niet gebruikte states zijn dan
         //don't care
equations
    OUTPUTS.clk = Clk;
state_diagram OUTPUTS
state S0: if hold then S1 else S0;
state S1: if hold then S2 else S1;
state S2: if hold then S3 else S2;
state S3: if hold then S0 else S3;
test_vectors ([Clk,hold]->[Q1,Q0])
             [0,x]->[0,1];
             [C,1]->[0,1];
             [C,1]->[1,0];
             [C,0]->[1,0];
             [C,1]->[1,1];
             [C,1]->[0,0];
             [C,1]->[0,1];
             [C,1]->[1,0];
             [C,1]->[1,1];
end cnt2
```

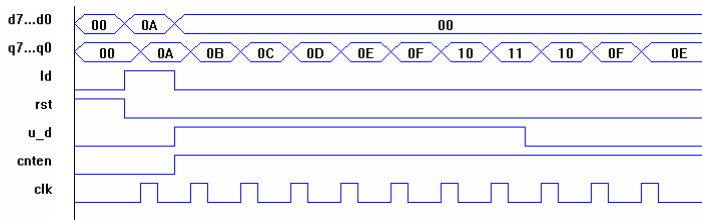


## Ontwerp van een 8 bit binaire up- down counter door gebruik te maken van: **when then else**

```

MODULE uni8Bcnt
interface (d7..d0, clk, rst, cnten, ld, u_d -> q7..q0);
TITLE '8bit universal counter with parallel load'
"constants
H,L,X,C,Z = 1, 0, .X., .C., .Z.;
"inputs
d7..d0 pin; "data inputs, 8 bits wide
clk pin; "clock input
rst pin; "asynchronous reset
cnten pin; "count enable
ld pin; "load counter with input data value
u_d pin; "up/down selector: high selects up
"outputs
q7..q0 pin istype 'reg,buffer'; "counter outputs
"sets
data = [d7..d0]; "data set
count = [q7..q0]; "counter set
"mode equations
MODE = [cnten,ld,u_d]; "mode set composed of control pins
LOAD = (MODE == [ X , 1 , X ]); "Various modes are defined by
HOLD = (MODE == [ 0 , 0 , X ]); "values applied to control pins.
UP = (MODE == [ 1 , 0 , 1 ]); "Symbolic name may be defined as
DOWN = (MODE == [ 1 , 0 , 0 ]); "a set equated to a value.
equations
when LOAD then count := data "load counter with data
else when UP then count := count + 1 "count up
else when DOWN then count := count - 1 "count down
else when HOLD then count := count; "hold count
count.clk = clk; "counter clock input
count.ar = rst; "counter reset input
test_vectors
([clk,rst,cnten,ld,u_d,data]->[count])
[0 , 1 , X , X , X , X ]->[0] ; "reset
[C , 0 , X , 1 , X , ^hA ]->[^hA] ; "preset
[C , 0 , 1 , 0 , 1 , X ]->[^hB] ; "count up
[C , 0 , 1 , 0 , 1 , X ]->[^hC] ; "count up
[C , 0 , 1 , 0 , 1 , X ]->[^hD] ; "count up
[C , 0 , 1 , 0 , 1 , X ]->[^hE] ; "count up
[C , 0 , 1 , 0 , 1 , X ]->[^hF] ; "count up
[C , 0 , 1 , 0 , 1 , X ]->[^h10] ; "count up
[C , 0 , 1 , 0 , 1 , X ]->[^h11] ; "count up
[C , 0 , 1 , 0 , 0 , X ]->[^h10] ; "count down
[C , 0 , 1 , 0 , 0 , X ]->[^hF] ; "count down
[C , 0 , 1 , 0 , 0 , X ]->[^hE] ; "count down
END

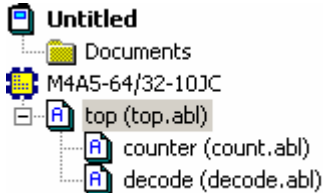
```



Het optioneel **interface** statement wordt gebruikt om aan te geven welke signalen de ingangen en de uitgangen zijn van de ABEL beschrijving voor het geval waarbij men vanuit een andere ABEL beschrijving de huidige als een soort functie zou willen oproepen.

### 8.3 Een hiërarchisch ontwerp

Men kan een ontwerp hiërarchisch uitvoeren zoals in onderstaande project-navigator. Het is een ontwerp in een MACH 4 CPLD.



Module top (top.abl) spreekt de module counter (count.abl) en decode (decode.abl) aan, zoals in bovenstaande figuur.

```
MODULE top
rst, clock      pin 9,11;  "rst is actually sw1 on the demo board
!sw2,!sw3       pin 21,31; "sw2 is going to be the up counter and sw3 is down

!led4a,!led4b,!led4c,!led4d,!led4e,!led4f,!led4g pin 2,3,4,5,6,7,8  istype'com';
!led3a,!led3b,!led3c,!led3d,!led3e,!led3f,!led3g pin 14,15,16,17,18,19,20 istype'com';
!led2a,!led2b,!led2c,!led2d,!led2e,!led2f,!led2g pin 24,25,26,27,28,29,30 istype'com';
!led1a,!led1b,!led1c,!led1d,!led1e,!led1f,!led1g pin 36,37,38,39,40,41,42 istype'com';
!leddp         pin 43  istype'com';

count15..count0          node istype'com';
up,dn                   node istype'com';

counter interface(clock,rst,up -> count15..count0);
my_count functional_block counter;

decode interface(in3..in0 -> out6..out0);

decode0 functional_block decode;
decode1 functional_block decode;
decode2 functional_block decode;
decode3 functional_block decode;
```

U kunt een **'functional\_block'** declareren in het hoogste niveau van een ABEL-bestand.

'functional\_block' Wijst een MODULE op een lager niveau aan.

VB: counter en decode (zie bovenstaand voorbeeld)

Men moet wel eerst met de 'interface' declaratie de module concretiseren.

VB: decode **interface**(in3..in0 -> out6..out0); (zie bovenstaand voorbeeld)

Het **'interface'** statement geeft aan welke signalen de ingangen en de uitgangen zijn van de ABEL beschrijving voor het geval waarbij men vanuit een andere ABEL beschrijving de huidige als een soort functie zou willen oproepen.

De logische vergelijkingen van het ontwerp (4-bit binary counter) worden weergegeven in onderstaande ABEL-bestand top.abl. Het gaat om een 16-bit counter waarvan de vier nibbles worden uitgedecodeerd naar een vier maal 7 segment display.

#### EQUATIONS

```

up = rst # sw2 # !dn;    "up is active when reset is active, when sw2 is active,
                        "or when DN is not active
dn = !up # sw3;         "dn is active when sw3 is active or up is not active.
leddp = up;             " the decimal on the LSB will light up when counting up.

my_count.clock = clock;
my_count.rst = rst;
my_count.up = up;
[count15..count0] = my_count.[count15..count0];

decode0.[in3..in0] = [count3,count2,count1,count0];
[led1g,led1f,led1e,led1d,led1c,led1b,led1a] = decode0.[out6..out0];

decode1.[in3..in0] = [count7,count6,count5,count4];
[led2g,led2f,led2e,led2d,led2c,led2b,led2a] = decode1.[out6..out0];

decode2.[in3..in0] = [count11,count10,count9,count8];
[led3g,led3f,led3e,led3d,led3c,led3b,led3a] = decode2.[out6..out0];

decode3.[in3..in0] = [count15,count14,count13,count12];
[led4g,led4f,led4e,led4d,led4c,led4b,led4a] = decode3.[out6..out0];
END

```

We maken hiervoor gebruik van een module op lager niveau 'decode'. Zie onderstaand ABEL-bestand decode.abl

```

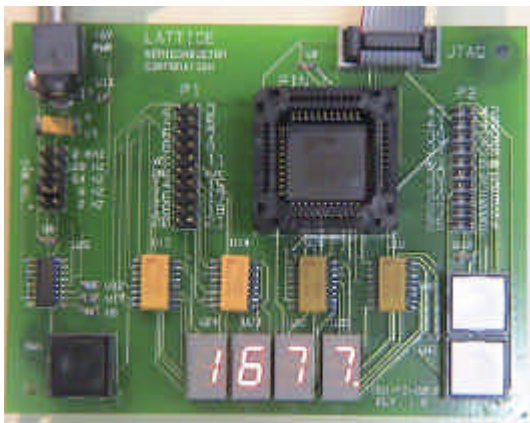
MODULE decode
in3..in0      pin;
out6..out0    pin_istype'com';
ins = [in3..in0];
outs = [out6..out0];
EQUATIONS
WHEN (ins == 0) THEN
  outs = [0,1,1,1,1,1,1];
ELSE WHEN (ins == 1) THEN
  outs = [0,0,0,0,1,1,0];
ELSE WHEN (ins == 2) THEN
  outs = [1,0,1,1,0,1,1];
ELSE WHEN (ins == 3) THEN
  outs = [1,0,0,1,1,1,1];
ELSE WHEN (ins == 4) THEN
  outs = [1,1,0,0,1,1,0];
ELSE WHEN (ins == 5) THEN
  outs = [1,1,0,1,1,0,1];
ELSE WHEN (ins == 6) THEN
  outs = [1,1,1,1,1,0,1];
ELSE WHEN (ins == 7) THEN
  outs = [0,0,0,0,1,1,1];
ELSE WHEN (ins == 8) THEN
  outs = [1,1,1,1,1,1,1];
ELSE WHEN (ins == 9) THEN
  outs = [1,1,0,0,1,1,1];
ELSE WHEN (ins == 10) THEN
  outs = [1,1,1,0,1,1,1];
ELSE WHEN (ins == 11) THEN
  outs = [1,1,1,1,1,0,0];
ELSE WHEN (ins == 12) THEN
  outs = [0,1,1,1,0,0,1];
ELSE WHEN (ins == 13) THEN
  outs = [1,0,1,1,1,1,0];
ELSE WHEN (ins == 14) THEN
  outs = [1,1,1,1,0,0,1];
ELSE WHEN (ins == 15) THEN
  outs = [1,1,1,0,0,0,1];
ELSE outs = [0,0,0,0,0,0,0];
END

```

Module 'counter' (count.abl) beschrijft de 16-bit counter.

```
MODULE counter
clock,rst,up    pin;
count15..count0 pin istype'reg';
count = [count15..count0];
EQUATIONS
count.clk = clock;
count.ar = rst;
WHEN (up) THEN
    count := count + 1;
ELSE
    count := count - 1;
END
```

Het gerealiseerde JEDEC-bestand kan worden gedownload met de "ispVM System" van Lattice naar onderstaande print via de JTAG-interface.



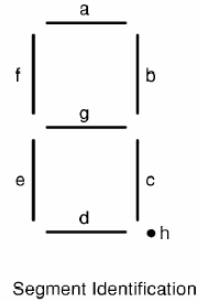
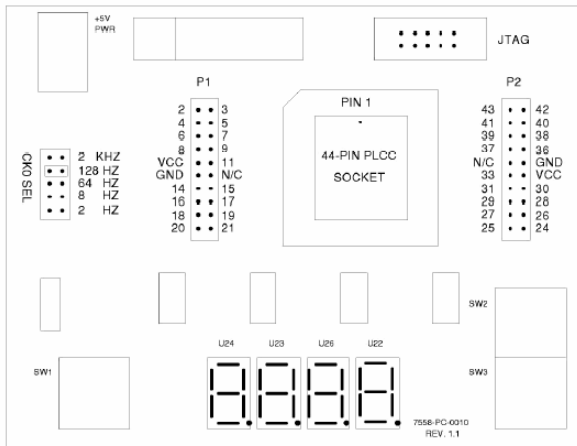
**Print met JTAG-interface en vier 7-segment displays**

<http://www.latticesemi.com/products/devtools/software/ispLEVER-starter-kit/index.cfm>

De pinning van het ontwerp ligt vast, zie lijst op volgende bladzijde.

## Pinning van het ISP Starter Kit

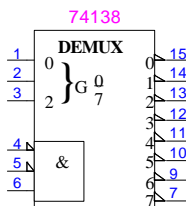
| Device Pin | Pin Definition | MACH Inputs | LED        | Comment            |
|------------|----------------|-------------|------------|--------------------|
| 1          | GND            |             |            |                    |
| 2          | I/O 0          |             | U24-A      |                    |
| 3          | I/O 1          |             | U24-B      |                    |
| 4          | I/O 2          |             | U24-C      |                    |
| 5          | I/O 3          |             | U24-D      |                    |
| 6          | I/O 4          |             | U24-E      |                    |
| 7          | I/O 5          |             | U24-F      |                    |
| 8          | I/O 6          |             | U24-G      |                    |
| 9          | I/O 7          | SW1         | U24-H (DP) | input only         |
| 10         | TDI            |             |            |                    |
| 11         | CLK 0 / I 0    | CK0 Clock   |            | Select with jumper |
| 12         | GND            |             |            |                    |
| 13         | TCK            |             |            |                    |
| 14         | I/O 8          |             | U23-A      |                    |
| 15         | I/O 9          |             | U23-B      |                    |
| 16         | I/O 10         |             | U23-C      |                    |
| 17         | I/O 11         |             | U23-D      |                    |
| 18         | I/O 12         |             | U23-E      |                    |
| 19         | I/O 13         |             | U23-F      |                    |
| 20         | I/O 14         |             | U23-G      |                    |
| 21         | I/O 15         | SW2         | U23-H (DP) | input only         |
| 22         | VCC            |             |            |                    |
| 23         | GND            |             |            |                    |
| 24         | I/O 16         |             | U26-A      |                    |
| 25         | I/O 17         |             | U26-B      |                    |
| 26         | I/O 18         |             | U26-C      |                    |
| 27         | I/O 19         |             | U26-D      |                    |
| 28         | I/O 20         |             | U26-E      |                    |
| 29         | I/O 21         |             | U26-F      |                    |
| 30         | I/O 22         |             | U26-G      |                    |
| 31         | I/O 23         | SW3         | U26-H (DP) | input only         |
| 32         | TMS            |             |            |                    |
| 33         | CLK 1 / I 1    | CK1 Clock   |            | 4 Hz Clock signal  |
| 34         | GND            |             |            |                    |
| 35         | TDO            |             |            |                    |
| 36         | I/O 24         |             | U22-A      |                    |
| 37         | I/O 25         |             | U22-B      |                    |
| 38         | I/O 26         |             | U22-C      |                    |
| 39         | I/O 27         |             | U22-D      |                    |
| 40         | I/O 28         |             | U22-E      |                    |
| 41         | I/O 29         |             | U22-F      |                    |
| 42         | I/O 30         |             | U22-G      |                    |
| 43         | I/O 31         |             | U22-H (DP) |                    |
| 44         | VCC            |             |            |                    |



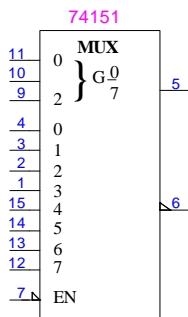
## Print layout van het ISP Starter Kit

### 8.4 Oefeningen

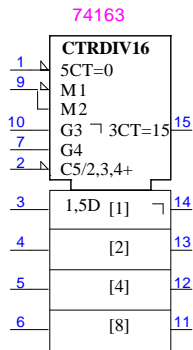
1. Ontwerp in de **PAL22V10** een **demux. 74xx138** de keuze van de pinnummers is onbelangrijk. Beschrijf de logica combinatorisch en daarna met een waarheidstabel (truth\_table).



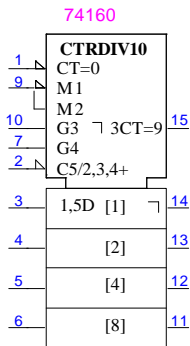
2. Ontwerp in de **PAL22V10** een **mux. 74xx151** de keuze van de pinnummers is onbelangrijk. Beschrijf de logica combinatorisch en daarna met een waarheidstabel (truth\_table).



3. Ontwerp in de **PAL22V10 de 74xx163**, de keuze van de pinnummers is onbelangrijk.



4. Ontwerp in de **PAL22V10 de 74xx160**, de keuze van de pinnummers is onbelangrijk.



5. Ontwerp in een **CPLD MACH4A5 64/32** een **BCD-teller** met als asynchrone ingangen:

RST, = sw1 op het ISP Starter Kit.

PST, = sw2 op het ISP Starter Kit.

HOLD, = sw3 op het ISP Starter Kit.

De uitgangen van de teller staan op de segmenten G van de 7-segment display's;

U24-G, U23-G, U26-G, en U22-G.

De clock wordt via de oscillator op het ISP Starter Kit gegenereerd.

Onderstaande gegevens geeft de nummering van de pinnen die kunnen gebruikt worden.

*"input*

*!RST, Clk* *pin 9,11;*

*!HOLD, !PST* *pin 21,31;*

*"output*

*Q3,Q2,Q1,Q0 pin 8,20,30,42;*

Realiseer tevens de verschillende testvectoren.

Programmeer de component via ispVM System Software van Lattice.

6. Ontwerp in een **CPLD MACH4A5 64/32** een **BCD U/D-counter** met als asynchrone Reset.

RST, = sw1 op het ISP Starter Kit.

UP, = sw2 op het ISP Starter Kit.

Down, = sw3 op het ISP Starter Kit.

De uitgangen van de teller staan op de segmenten G van de 7-segment display's;

U24-G, U23-G, U26-G, en U22-G.

De clock wordt via de oscillator op het ISP Starter Kit gegenereerd.

Onderstaande gegevens geeft de nummering van de pinnen die kunnen gebruikt worden.

*"input*

*!RST, Clk pin 9,11;*  
*!UP, !Down pin 21,31;*

*"output*

*Q3,Q2,Q1,Q0 pin 8,20,30,42;*

Realiseer tevens de verschillende testvectoren.

Programmeer de component via ispVM System Software van Lattice.

7. Ontwerp in een **CPLD MACH4A5 64/32** programmeerbaar **looplicht** van vier ledjes.

Met SW1 (RST) kunnen we al de ledjes doven.

Met SW2 (2LS) kiezen we tussen een dovend ledje, dat loopt of een brandend ledje, dat loopt.

RST, = sw1 op het ISP Starter Kit.

2LS, = 0 "looplicht van dovend ledje"

2LS, = 1 "looplicht van brandend ledje"

De uitgangen van het looplicht staan op de segmenten G van de 7-segment display's;

U24-G, U23-G, U26-G, en U22-G.

De clock wordt via de oscillator op het ISP Starter Kit gegenereerd.

Onderstaande gegevens geeft de nummering van de pinnen die kunnen gebruikt worden.

*"input*

*!RST, Clk, !2LS pin 9,11;21;*

*"output*

*Q3,Q2,Q1,Q0 pin 8,20,30,42;*

Realiseer tevens de verschillende testvectoren.

Programmeer de component via ispVM System Software van Lattice.