Technische beschrijving van de Pocketstem

Instituut voor Perceptie Onderzoek
Postbus 513 - 5600 MB Eindhoven

RW/rw 87/04
10.09.1987

Rapport no. 606

Technische beschrijving van de Pocketstem

R.P. Waterham
M.W.C. Verhoeven

1 Samenvatting.

Dit rapport beschrijft de ontwikkeling en de specificatie van de Pocketstem, een prototype van een hulpmiddel voor spraakgehandicapten. De Pocketstem is de opvolger van het proefmodel, de Compacte Spraakhulp I. De Compacte Spraakhulp I heeft een eerste evaluatie ondergaan en de resultaten van deze evaluatie hebben geleid tot de ontwerpeisen van de Pocketstem. De Pocketstem is naar aanleiding van de evaluatie van de Compacte Spraakhulp I compacter geworden en eenvoudiger te bedienen dan zijn voorganger.

Uit de evaluatie van de Compacte Spraakhulp I is naar voren gekomen, dat duidelijke symbolen op de toetsen van het hulpmiddel de bediening aanzienlijk vergemakkelijken. Dit heeft geleid tot het ontwerpen van een symbolenset voor de Pocketstem.

Ten behoeve van de evaluatie van de Pocketstem zijn 5 exemplaren gerealiseerd, die gedurende perioden van twee maanden tot een half jaar bij verschillende categorieën spraakgehandicapten zijn uitgeprobeerd. Aangezien de Pocketstemmen regelmatig van andere zinnen moeten worden voorzien, is ondersteunings programmatuur ontworpen en op een personal computer geïnstalleerd. Met dit systeem is het mogelijk een bestand op te bouwen en nieuwe zinnen aan het bestand toe te voegen. Voor het vullen van een Pocketstem kan op eenvoudige en snelle wijze een keus gemaakt worden uit het zinnenbestand. Vervolgens kan het geheugen van de Pocketstem automatisch van de nodige data worden voorzien.

De hoofdconclusie van de evaluatie is, dat de Pocketstem bij bepaalde doelgroepen goed voldoet. Toch blijven er enkele belangrijke wensen over. Met name bestaat er bij bepaalde gebruikersgroepen de wens om nog meer zinnen aan te kunnen roepen. Het probleem hierbij is niet de opslagcapaciteit van de Pocketstem, maar hoe selectie van de zinnen moet plaatsvinden.

De ontworpen symbolenset blijkt in praktijk goed te voldoen. Dit geldt ook voor de ondersteuningsprogrammatuur. Ten aanzien van het technisch functioneren van de Pocketstem kan opgemerkt worden dat dit geen echte problemen heeft opgeleverd, behalve dan dat de staart van het membraan keyboard erg slijtage-gevoelig was.

Inhoud

2 Inleiding.

In het kader van het in augustus 1984 gestarte project "Ergonomische communicatie apparatuur ten behoeve van spraakgehandicapten", is in februari 1985 gestart met de ontwikkeling en de realisatie van een proefmodel van een communicatie hulpmiddel voor vocaal gehandicapten. Het project is onderdeel van de activiteiten van de interfaculteitswerkgroep "Communicatie hulpmiddelen voor gehandicapten" (een samenwerkingsverband tussen het Instituut voor Perceptie Onderzoek IPO en de vakgroep Medische Elektrotechniek EME van de Technische Universiteit Eindhoven). Het proefmodel wordt de Compacte Spraakhulp I (CSH I) genoemd. Dit proefmodel is in praktijk (in samenwerking met het Instituut voor Revalidatie Vraagstukken IRV) uitgeprobeerd met als doel om, met de CSH I, een indruk te krijgen van de gebruiksmogelijkheden en de acceptatie van synthetische spraak bij vocaal gehandicapten. De resultaten van de ontwikkeling van de CSH I en de veldevaluatie zijn beschreven in [8] en [13]. Aan de hand van deze resultaten zijn, eind 1985/begin 1986, de ontwerpeisen voor een prototype (de CSH II) opgesteld [14]. Ook is in dezelfde periode een inventaris opgemaakt van bruikbare toetsenbordjes voor het bedoelde prototype [15]. Aan de hand van de ontwerpeisen zijn door M. Verhoeven en R. Waterham 5 prototypes ontworpen en gerealiseerd. Behalve de uitvoering is ook de naam van het prototype gewijzigd. Het prototype heet nu Pocketstem.

In hoofdstuk 2 wordt de lezer ingeleid en wordt gemeld wat er aan de ontwikkeling van de Pocketstem vooraf is gegaan. In hoofdstuk 3 worden de ontwerpeisen voor de Pocketstem op een rij gezet. Bij de eisen is de nadruk gelegd op compactheid en eenvoud van bediening. In hoofdstuk 4 wordt een beschrijving gegeven van de behuizing van de Pocketstem. In hoofdstuk 5 wordt de hardware van de Pocketstem beschreven en wordt de werking ervan uitgelegd. De hardware is opgesplitst in functionele deelschakelingen, die elk afzonderlijk zullen worden beschreven. In hoofdstuk 6 wordt de bij de hardware behorende software beschreven. De beschrijving handelt over de structuur van het μ-processor besturingsprogramma en er wordt, met behulp van flow-charts, getracht aan te geven hoe het programma werkt. Hoofdstuk 6 geeft ook informatie over de geheugen-indeling en de gebruikte registers.

Gelijktijdig met het ontwerpen van de prototypes is aandacht besteedt aan een tweetal andere ontwikkelingen.

Bij de evaluatie van de CSH I is duidelijk geworden dat de symbolen op

de toetsen van essentieel belang zijn. Daarom is besloten voor de Pocketstem een symbolenset te ontwikkelen. Dit is gedaan door twee studenten, M. Vroemen en H. van Bloemendaal, van de universiteit van Utrecht, studierichting Ergonomische Psychologie, [12] en [2]. In hoofdstuk 4 wordt een beschrijving gegeven van het ontwerp en het gebruik van deze symbolenset.

Voorts bleek de zinnenset, die in de CSH I is gebruikt, te beperkt. Er werd een grotere zinnenkeuze verlangd. Dit is de reden waarom er een zinnenbestand met bijbehorende beheer-software is ontwikkeld op een Apple IIe computer. Deze software is ontwikkeld door G. Tan [10] en G. Legdeur [6]. Met deze software wordt het zinnenbestand beheerd en kan een Pocketstem eenvoudig van nieuwe zinnen worden voorzien. In hoofdstuk 7 wordt deze ondersteunings-software ten behoeve van het zinnenbestand voor de Pocketstem besproken.

In dit rapport worden ook enige resultaten met betrekking tot de veldevaluatie besproken. Deze veldevaluatie van de Pocketstem is, in samenwerking met I. Speth-Lemmens van het IRV, in juli 1986 gestart en in april 1987 is het evaluatierapport verschenen [9]. Dit rapport bevat de evaluatieresultaten voor wat betreft de acceptatie en de gebruikswaarde van de Pocketstem als communicatiehulpmiddel. Alle technische aspecten met betrekking tot het functioneren van de Pocketstem zullen in hoofdstuk 8 besproken worden.

Tot slot van dit rapport worden in hoofdstuk 9 nog enige conclusies besproken, die betrekking hebben op het ontwerp en het functioneren van de Pocketstem.

In de bijlagen zijn de schema's, de printopstelling, printlay-out, aanzichttekeningen, de gebruiksaanwijzing en de assembler-listing van het Pocketstem besturingsprogramma opgenomen.

3 Ontwerpeisen en de realisatie ervan.

Het eerste proefmodel, de CSH I, was ontworpen om te onderzoeken welke eisen spraakgehandicapten stellen aan spraakvervangende apparatuur. [13] Naar aanleiding van de evaluatie van dit eerste proefmodel [8] en de beschikbaarheid van nieuwe electronische componenten, zijn enkele veranderingen in de bediening en het uiterlijk van het nieuwe ontwerp, de Pocketstem, voorgesteld.[14]
In dit hoofdstuk worden deze eisen op een rijtje gezet en daarna volgen de veranderingen aan de Pocketstem ten opzichte van de CSH I ten gevolge van die eisen.

3.1 Ontwerpeisen.

De technische en functionele eisen en de eisen aan het uiterlijk kunnen als volgt worden samengevat:

***Eisen aan de behuizing* :

- Het apparaat moet zo klein mogelijk worden.
- Er is geen display nodig (hetgeen wel aanwezig is bij de CSH I).
- De Pocketstem dient voorzien te zijn van een aan/uit-schakelaar.
- Er is een batterij-leeg indicator gewenst.
- Er moeten bevestigingsmogelijkheden aan het apparaat worden aangebracht om het te kunnen dragen of om het aan een rolstoel te kunnen bevestigen.
- Er moeten maatregelen worden genomen om wegslippen (b.v. van de tafel) te voorkomen.

***Eisen aan de bediening* :

- De bediening moet eenvoudiger.
- Het geluidsvolume moet groter worden en tevens moet het geluidsvolume in diverse niveau's instelbaar zijn.
- Er is een keyboard gewenst, dat minder ruimte beslag neemt.

- De toetsen van het keyboard dienen voorzien te worden van symbolen, die zoveel mogelijk overeenkomen met de in de ergonomie toegepaste richtlijnen.
- Er moet een afdekplaat voor het keyboard bijgeleverd worden voor patienten met motorische handicaps.

** *Eisen aan de spraakuitvoer – mogelijkheden* :

- Er moet een grotere keuze aan zinnen komen.
- De spraakkwaliteit moet, zover mogelijk, verbeterd worden.
- De gebruiker moet kunnen kiezen uit een mannen- of een vrouwenstem.
- Eventueel worden de zinnen met een verschillend intonatie-patroon opgeslagen, zodat er een variatie in de uitspraak van de zin mogelijk wordt.

3.2 Verschillen tussen de behuizingen van de Pocketstem en van de CSH I.

- De behuizing (zie hoofdstuk 4 voor de beschrijving), waarin de Pocketstem nu is ondergebracht, is veel kleiner dan die van de CSH I.
  Deze verbetering was niet alleen mogelijk doordat de dikte van het toetsenbord (zie 3.3) nu slechts ongeveer 1 mm bedraagt, maar ook omdat er geen display meer aanwezig is en omdat de luidspreker nu schuin is geplaatst.(Zie hfst 4)
  Door de schuine stand van de luidspreker en het ontbreken van een display, kan nu de hele bovenkant voor het toetsenbord worden gebruikt. Ook het feit, dat er minder electronische componenten nodig waren, past erg goed in het streven om het kastje zo klein mogelijk te maken.
- Uit de evaluatie is gebleken, dat het display (om te controleren of de geselecteerde zin wel de bedoelde was, voordat die via een druk op de "spreek"-toets kon worden uitgesproken) nauwelijks werd gebruikt. Vandaar, dat dit met een gerust hart mocht worden weggelaten.
  Binnen de behuizing is er nog wel een aansluitmogelijkheid voor een display. Deze is bedoeld voor de evaluatie van de Pocketstem.
  In het RAM-geheugen (Random Access Memory) wordt n.l. opgeslagen hoe vaak iedere zin wordt geselecteerd. De resultaten hiervan kunnen later via het display zichtbaar worden gemaakt.

- Om tijdens het transport niet per ongeluk een zin te laten uitspreken, als er toevallig iets tegen een toets aandrukt, is de Pocketstem ook voorzien van de gewenste aan/uit schakelaar. Hiermee wordt het toetsenbord geblokkeerd en de processor komt in de power-down mode. In deze mode is het stroom-verbruik minimaal, echter de data in het RAM-geheugen blijft bewaard.

- Tevens is de Pocketstem nu ook voorzien van een batterij-leeg indicator. Deze indicator geeft een waarschuwing, zodra de Ni-Cd accu te ver leegraakt. Nu hoeft er geen verwarring meer te bestaan tussen het te ver leeg zijn van de accu of het optreden van een eventueel defect aan dit tweede prototype.
  Als de accu te ver leeg is geraakt, dan wordt er niets meer uitgesproken, enkel het waarschuwingslampje brandt dan ongeveer 4 sec., als er een zin wordt geselecteerd. Daarna is het toetsenbord geblokkeerd. De accu zorgt er nog wel voor, dat de inhoud van de RAM intact blijft. Deze blokkade wordt opgeheven door een RESET (d.w.z. met de aan/uit-schakelaar het apparaat even uit- en dan weer aanzetten). Er wordt pas weer spraak geproduceerd, nadat de accu voldoende is opgeladen.

- In de ontwerpeisen is gesteld, dat het apparaat voorzien dient te zijn van bevestigingsmogelijkheden om het te kunnen dragen of om het aan een rolstoel te kunnen bevestigen.
  Bij de werkbesprekingen is gebleken, dat hiervoor verschillende ad hoc oplossingen toe te passen zijn. Er is daarom besloten om de instrumentatiegroepen van de diverse instituten deze voorzieningen, indien gewenst, zelf te laten aanbrengen.

- Om wegslippen te voorkomen, voor het geval de Pocketstem op een glad en/of hellend oppervlak staat, zijn er rubberdopjes aan de onderkant aangebracht. Door deze dopjes treedt er ook minder luidheidsverlies op als het kastje op zo'n oppervlak staat. Tussen het oppervlak en de gedeeltelijk naar de onderkant uitstralende luidspreker blijft nu wat ruimte vrij. Hierdoor wordt het geluid minder gedempt, dan wanneer deze dopjes er niet zouden zijn.

3.3 Veranderingen in de bediening.

- Voor gebruikers met motorische handicaps bleek de bediening van de CSH I te gecompliceerd. De bediening van de Pocketstem is daarom

eenvoudiger gemaakt dan die van de CSH I.
Bij de CSH I moeten er, voor de detectie en het uitspreken van een zin, minimaal twee toetsen worden ingedrukt: Allereerst de toets voor de gekozen zin en daarna de "spreek"-toets voor het uitspreken van die zin op het middelste geluidsvolume.
(Er zijn drie niveau's mogelijk: Wil je de uitspraak hard of zacht hebben, dan moet je, voordat de "spreek"-toets wordt ingedrukt, nog eens extra de "hard"- of "zacht"-toets indrukken.)
Als je langer wacht dan 8 seconden, voordat de volgende toets wordt ingedrukt, dan komt de CSH I terug in de 'stand-by'-toestand en moet de selectieprocedure van een zin weer van vooraf aan beginnen.
Deze 8 seconden bleken soms te kort en de methode te omslachtig.
Bij de Pocketstem hoef je nog slechts één toets in te drukken om een zin te selecteren en te laten uitspreken.

- Het geluidsvolume kan vooraf in diverse niveau's worden ingesteld met een aparte volumeschakelaar, die aan de onderkant van de behuizing is aangebracht.
  Door het overbodig worden van de extra "spreek"-toets en volume-toetsen zijn nu alle toetsen beschikbaar voor het selecteren en het uitspreken van de zinnen.
  De allereerste keer moet je natuurlijk wel de aan/uit-schakelaar in de goede stand zetten. (De Pocketstem wordt aangezet door de aan/uit-schakelaar in de richting van de batterij-leeg indicator te verschuiven. Op de behuizing is dit nog niet aangegeven.)

- Als toetsenbord voor de Pocketstem is een membraan-keyboard gekozen. Dit kunststof toetsenbord heeft slechts een geringe dikte. Daarnaast is het vocht- en vuilbestendig en het is gemakkelijk schoon te maken.
  Bovendien is de prijs ervan, na relatief hoge aanmaakkosten, beduidend lager dan die van sommige andere, in de overwegingen meegenomen, keyboards. [15] Als extra pluspunt geldt daarnaast nog, dat de oppervlakte van de toetsen erg vlak is, zodat er gemakkelijk stickers met symbolen kunnen worden opgeplakt. (Zie volgende punt)

- Voor de ontwikkeling en bruikbaarheid van standaard symbolen op de toetsen van het toetsenbord is er een samengewerking tot stand gekomen met de faculteit Ergonomische Psychologie van de Rijksuniversiteit Utrecht.

Dit resulteerde in een tweetal stages over dit onderwerp. In hoofdstuk 4 wordt hier nader op ingegaan.

- Er was tevens in de ontwerpeisen gesteld, dat er bij de Pocketstem een afdekplaat geleverd diende te worden, voor het geval, dat een patient moeilijkheden met de bediening had.
  Door de opbouw van het keyboard is enigszins tegemoet gekomen aan deze eis. Een toets reageert nl. alleen als er in het midden op wordt gedrukt.
  De toetsen zijn gescheiden door een opstaand randje en langs deze randjes kun je vrij gemakkelijk de gewenste toets opzoeken.
  Mocht dit voor bepaalde patienten nog onvoldoende blijken, dan stellen wij voor, dat ook hier de realisatie van de afdekplaat weer door de diverse instrumentatie-groepen wordt uitgevoerd.
  Is dit niet mogelijk, dan kan die afdekplaat eventueel op het IPO worden gemaakt. Als ook dit laatste niet het gewenste resultaat oplevert, dan gaat de Pocketstem over naar een andere gebruiker.

3.4 Veranderingen met betrekking tot de spraakuitvoer.

- Door het grotere aantal prototypes van de Pocketstem t.o.v. de CSH I (daarvan was er slechts één), komen er voor de evaluatie meer gebruikers in aanmerking en wordt er meer van gebruiker gewisseld.
  Er moest dus een grotere zinnenset komen, waaruit de gebruikers konden kiezen. Bovendien moest deze database gemakkelijk zijn uit te breiden.

  De verzameling zinnen, waaruit de gebruiker thans een keuze kan maken, is uitgebreid tot ruim 190 verschillende zinnen. Dit bestand is ondergebracht op floppy disks van een Apple IIe personal-computer en van hieruit kan, m.b.v. een EPROM-programmer, een EPROM-geheugen worden geprogrammeerd met de persoonlijke keuze van de gebruiker.
  Mede door de transporteerbaarheid van de Apple IIe en de EPROM-programmer is een flexibel systeem ontstaan, waarbij eventueel bij de gebruiker thuis de gewenste zinnenset kan worden gekozen uit het aanwezige bestand en ter plaatse in de EPROM kan worden geprogrammeerd.
  De EPROM's zijn door een technicus vrij gemakkelijk uitwisselbaar,

daar ze op de aanwezige IC-voet in de Pocketstem kunnen worden geplaatst.
(Voor meer informatie over het "Apple"-systeem: Zie hoofdstuk 7)

- De spraakkwaliteit kan worden verbeterd door de toepassing van een nieuwe spraak-synthesizerchip (PCF8200).

- Met deze nieuwe chip is het ook mogelijk om de Pocketstem met een mannen- of met een vrouwenstem te laten spreken.
  Er zijn reeds enkele laboratoriumversies van deze nieuwe chip in ons bezit en uitgetest. Het ligt in de bedoeling om in de toekomst (als de PCF8200 in produktie genomen is) alle Pocketstemmen met deze chip uit te rusten.

- De mogelijkheid om zinnen met een tweede intonatie-patroon op te slaan is gerealiseerd bij die apparaten, die met de oude spraakchip (MEA8000) zijn uitgerust. Als je daarbij twee keer achter elkaar op dezelfde toets drukt, dan krijg je de tweede keer de zin iets anders geformuleerd of met een andere intonatie te horen.
  De nieuwe spraakchip (PCF8200) heeft voor zijn betere spraakkwaliteit meer geheugenruimte nodig. Daardoor kan het voorkomen, dat de thans toegepaste EPROM-geheugens te weinig ruimte bevatten om die tweede intonatie op te slaan.
  Daarom bevatten die Pocketstemmen, die met een PCF8200 zijn uitgerust, nu nog slechts zinnen zonder die variatiemogelijkheid.
  Mocht uit de evaluatie blijken, dat een tweede intonatie gewenst is, dan moeten er voor de toekomstige versies andere oplossingen worden gezocht. B.v. grotere geheugens toepassen of het zuiniger opslaan van de gecodeerde spraak.
  Bij een PCF8200 is het mogelijk om de spreeksnelheid te variëren of om de toonhoogte te manipuleren. Dan heb je slechts één gecodeerde versie spraak in het EPROM-geheugen van de Pocketstem, die je op verschillende manieren ten gehore kunt brengen.

4 Gebruikte symboliek, behuizing en constructie-aspecten.

Zoals reeds in hoofdstuk 3 is vermeld zijn er aan de Pocketstem een aantal ontwerpeisen gesteld. Enkele van deze eisen hebben betrekking op de behuizing en de bediening.
De veranderingen t.o.v. de CSH I aan de behuizing en de bediening t.g.v. die eisen zijn tevens in dat hoofdstuk behandeld.
In dit hoofdstuk volgt de beschrijving van de Pocketstem.
Er wordt begonnen met een uiteenzetting over de symbolen, die aangebracht kunnen worden op de toetsen van het toetsenbord en de overwegingen, die tot het ontwerp van de huidige verzameling symbolen hebben geleid.
(Uitgebreide informatie over de gebruikte symbolen is te vinden in de rapporten van Vroemen [12] en van v.Bloemendaal [2].)
Vervolgens volgt de beschrijving van de behuizing: Het uiterlijk en hoe het er van binnen uitziet. Hierbij wordt ook ingegaan op de beschrijving van de printplaten (lay-out van de printplaten en de plaatsing van de componenten daarop).

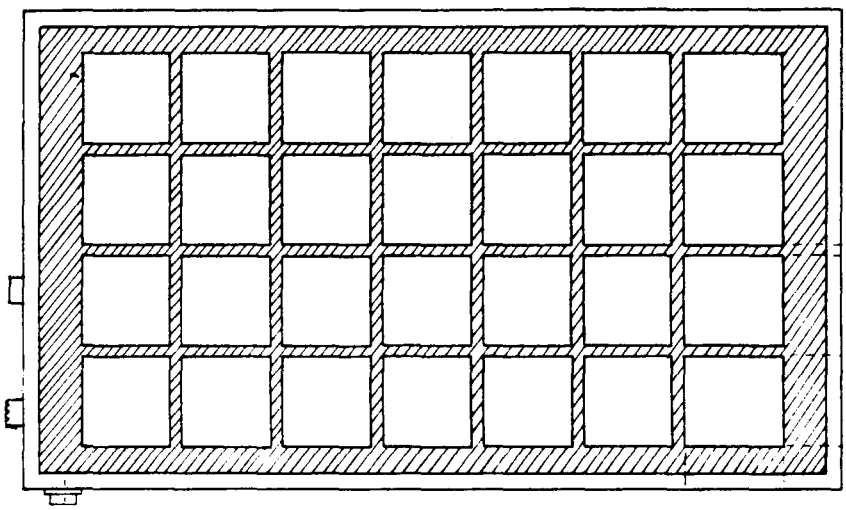
4.1 Gebruikte symboliek.

4.1.1 Algemeen.

Om een goede keuze uit de 28 toetsen van het toetsenbord (zie figuur 4.1) te kunnen maken zijn deze voorzien van symbolen. Deze symbolen geven een indicatie welke zin een gebruiker onder een toets kan verwachten.
Er is gekozen voor symbolen en niet voor tekst, omdat symbolen de informatie vaak sneller kunnen overdragen dan informatie in de vorm van tekst. (Zie bijvoorbeeld de symbolen op verkeersborden.) Onder slechte omstandigheden zijn symbolen meestal ook beter te zien dan tekst.

Algemeen kan gesteld worden, dat symbolen gezien kunnen worden als een middel om informatie over te dragen en te communiceren: De symbolen hebben een bepaalde vorm, die visuele informatie oplevert.
De manieren om die informatie visueel weer te geven verschillen onderling vooral door het niveau van abstractie. Modellen b.v. geven een veel realistischere visuele representatie van de werkelijkheid dan abstracte tekens.
Hoewel een direct en vrij realistisch symbool (b.v. vuur als waarschuwing voor brandgevaar of iets dat heet is) meer betekenis draagt dan een abstract symbool (b.v. een uitroepteken of een bliksemflits als algemeen waarschu-

wingsteken), kan gesteld worden dat geen enkel symbool, op zichzelf staand, een betekenis heeft. Deze betekenis is ontstaan uit afspraken tussen de gebruikers.



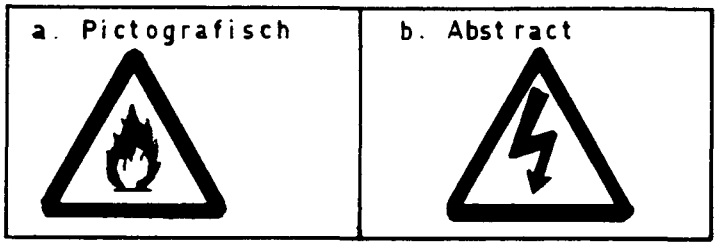
Figuur 4.1: Bovenaanzicht Pocketstem.

Voor symboolgebruik zijn drie elementen van belang:

- Het symbool : Een symbool moet goed waarneembaar zijn en goed begrepen worden.
- De referent : De betekenis kan gemakkelijker worden aangeleerd als symbolen zo concreet mogelijk refereren aan de werkelijkheid.
- De gebruiker : Door opleiding en/of ervaring moet de gebruiker enigszins zijn voorbereid op het gebruik van symbolische informatie. (Symbolen moeten dus zijn afgestemd op de gebruikersgroep.)

Vaak moet er een compromis gesloten worden tussen waarneembaarheid en begrijpelijkheid.
Een pictogram bevat meer details dan een abstract symbool. (B.v. een vlammetje t.o.v. een uitroepteken.) Het pictogram verwijst directer en re-

alistischer naar de referent (=het vlammetje) dan het abstracte symbool. Het abstracte symbool heeft nog wel kenmerken van de referent, maar is duidelijk vereenvoudigd.
Een pictogram is hierdoor begrijpelijker, echter het abstracte symbool is eenvoudiger en daardoor dus beter waar te nemen. (Zie figuur 4.2)



Figuur 4.2: Verschillen tussen symbolen.

4.1.2 Ontwerp-overwegingen.

Voor het ontwerpen van de symbolenset moet allereerst rekening worden gehouden met de toekomstige gebruikersgroep van de Pocketstem. Welke gebruikers ervoor in aanmerking komen, zal moeten blijken uit de evaluatie. Vast staat in ieder geval wel, dat de evaluatiegroep erg gedifferentieerd is. Tijdelijk en permanent spraak-gehandicapten, mensen met en zonder motorische gebreken, mensen met en zonder psychische storingen, jongeren, ouderen en zowel mannen als vrouwen komen er in voor.
De symbolen zullen niet te abstract mogen zijn, omdat ze dan door sommigen niet begrepen worden. Ze mogen ook niet teveel details bevatten, omdat ze dan voor anderen te slecht waarneembaar worden. (Vooral als je bedenkt, dat de toetsgrootte slechts 17x17 mm^2 bedraagt.)

Door de grote verscheidenheid in de evaluatiegroep bleek er geen standaard zinnenset te kunnen worden samengesteld, die voor iedereen de uitdrukkingen van de algemene, dagelijkse levensbehoeften en wensen omvat.
Op grond van bovenstaande beschouwing is er bij het ontwerpen van de symboliek uitgegaan van symbolen met algemene aanduidingen voor bepaalde behoeften, wensen en bezigheden en niet van symbolen voor specifieke zinnen. Een algemene referent, b.v. 'toiletbenodigdheden', beslaat alle specifieke zinnen, die hierop betrekking hebben. (Zoals wassen, douchen, haarkammen enz.)

Dit biedt de mogelijkheid om het aantal symbolen beperkt te houden. Er zijn nu nog slechts 52 categorieën zinnen met bijpassende symbolen overgebleven.
De gebruiker kan nu zelf het voor hem/haar meest geschikte symbool zoeken bij een bepaalde zin, die men op de Pocketstem beschikbaar wenst te hebben. Doordat men zelf het bijpassende symbool uitzoekt, zal men waarschijnlijk sneller en beter de betekenis ervan onthouden.
Doordat er slechts 52 categorieën zinnen bestaan, is de kans groot, dat bepaalde persoonlijke onderwerpen helemaal niet voorkomen. Hiervoor zijn blanco stickers aanwezig, waarop men zelf een passend symbool (tekst of getallen mag uiteraard ook) kan aanbrengen.

In de evaluatiefase moet ook onderzocht worden of de gebruikers de symbolen kunnen waarnemen (niet teveel details). Tevens moet er onderzocht worden of de symbolen op een betekenisvolle manier worden gerelateerd aan de desbetreffende referenten. Is dit niet het geval, dan moet bekeken worden of training (niet teveel, want het symbool moet snel en gemakkelijk onthouden kunnen worden) een oplossing biedt. Blijkt dan nog niet, dat het symbool bruikbaar is, dan moet het worden gewijzigd c.q. vervangen.
De bruikbaarheid van de symbolen is reeds beperkt uitgetest op een groep van 26 afasie-patienten [2]. Hieruit is gebleken, dat enkele symbolen gewijzigd dienden te worden en dat andere niet geschikt waren, zodat er voor die categorieën nieuwe symbolen ontworpen moesten worden. Ook bleek, dat bij wijzigingen en vervangingen rekening moest worden gehouden met de vorm van afasie, waaraan de gebruikers leden. Bij de ene vorm (afasie van Broca) zal de gebruiker meer moeite hebben met het uitdrukken van zijn gedachten, terwijl bij een andere vorm (de afasie van Wernicke) de gebruiker meer moeite heeft met het begrijpen van de taal.

In de toekomst kan mogelijk nog onderzocht worden of sommige symbolen slechts voor een bepaalde groep gebruikers geschikt is en dat andere symbolen meer algemeen kunnen worden benut.

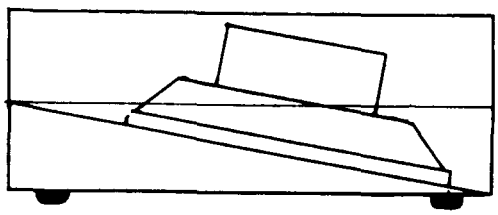
4.2 Beschrijving van de behuizing.

Evenals de CSH I is de Pocketstem ondergebracht in een kunststof behuizing. Het merk van de Pocketstem-behuizing is OKW Schalengehäuse en de bestelnummers zijn resp. 9409111 (zonder batterijvak) en 9409115 (met batterijvak).

Door de realisatie van de, in hoofdstuk 3 genoemde, ontwerpeisen heeft de Pocketstem een handzamer formaat gekregen (in bijlage III zijn de diverse aanzicht-tekeningen van de Pocketstem weergegeven) en een gemakkelijkere bediening.
Dit handzamere formaat kan goed geïllustreerd worden door de afmetingen en het gewicht van beide te vergelijken.
Voor de CSH I bedragen de afmetingen $l * b * h = 188 * 136 * 45\ mm^3$ en voor de Pocketstem blijkt, dat alle benodigde componenten kunnen worden ondergebracht in een behuizing van $l * b * h = 155 * 92 * 33\ mm^3$.
Daarnaast is de Pocketstem ook aanzienlijk lichter dan de CSH I: De Pocketstem weegt slechts 450 gr, terwijl de CSH I daar een gewicht van 850 gr tegenover stelt.

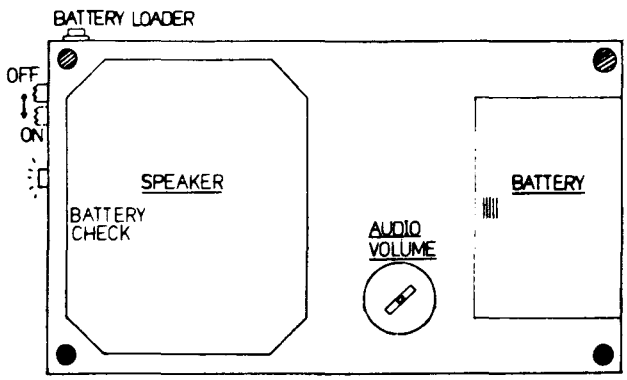
Doordat de "spreek"-toets en de geluidsvolume-toetsen niet meer nodig zijn voor de Pocketstem en door de andere plaatsing van de luidspreker kan nu het hele bovenoppervlak worden ingenomen door het toetsenbord (zie figuur 4.1).
Dit toetsenbord is een membraan-keyboard met een dikte van ongeveer 1 mm. Op dit toetsenbord is ruimte voor 28 toetsen, die elk een oppervlakte van $17 * 17\ mm^2$ beslaan.
Via de staart van het membraan-keyboard is het toetsenbord verbonden met een connector, die in de behuizing is ondergebracht.
Door middel van keyboard scanning is van deze 7x4 matrix de ingedrukte toets te bepalen (zie hoofdstuk 5).
In het bovendeksel is een gleuf gefreesd, waardoor de staart (zonder af te knellen) gemakkelijk aan de connector kan worden bevestigd. Tenslotte kan het toetsenbord simpelweg op het bovendeksel worden vastgeplakt.

De luidspreker (Philips AD2071/Z8), die een muziekvermogen van 1,5 watt heeft, geeft door de schuine plaatsing (zie fig 4.3) in 2 verschillende richtingen het geluid weer: Naar de onderzijde en naar de voorzijde van de behuizing (zie bijlage A.3). Hiertoe zijn aan de voorzijde gleufjes gefreesd en is de onderzijde voorzien van een gaatjes-patroon.



Figuur 4.3: Plaatsing luidspreker.

Als de Pocketstem in de hand wordt gehouden kan de onderzijde in de richting van de luisteraar worden gericht. Er is voor deze schuine plaatsing van de luidspreker gekozen, om het geluidsverlies te beperken als de Pocketstem op een vlak oppervlak wordt geplaatst. Nu wordt het geluid ook nog via de voorzijde weergegeven.
Mede door de ongeveer 5 mm hoge rubberen anti-slip dopjes aan de onderzijde, blijft het geluidsverlies via de onderzijde toch nog redelijk beperkt.



Figuur 4.4: Onderaanzicht behuizing.

De geluidsvolume-schakelaar (Zie figuur 4.4) steekt, aan de onderzijde, ongeveer 2 mm buiten de behuizing. De bediening van deze schakelaar is mogelijk d.m.v. een (stevige) nagel of een muntstuk. Aangezien sommige gebruikers niet in staat zijn zelfstandig het volume in te stellen, wordt op deze manier

voorkomen dat de instelling per ongeluk veranderd.

Met deze schakelaar kan het geluidsvolume van de brugversterker (met een sinusvermogen van 2,6 watt: Zie hoofdstuk 5) op 5 verschillende niveau's worden ingesteld.
De Pocketstem wordt gevoed door een Ni-Cd accu (9V-100mAh).
Via de bijgeleverde acculader kan een geheel lege accu in 14 uur worden opgeladen.
De acculader, FRIWO FW4001, heeft de volgende specificaties:

Netspanning	:	220 V $\sim \pm$ 10%/50-60 Hz
Oplaadspanning	:	10,15 V-
Oplaadstroom	:	10 mA
Oplaadtijd	:	14 uur

De batterij-leeg controle schakeling detecteert een te lage accuspanning reeds voordat de accu helemaal leeg is, zodat een nacht opladen in het algemeen voldoende is.

Als extra veiligheidsmaatregel is het dekseltje van het batterijvakje met een schroefje verzegeld. Dit is gedaan om te voorkomen, dat het batterijvakje spontaan openschiet. Omdat er gebruik wordt gemaakt van een oplaadbare accu en niet van een batterij, die soms moet worden verwisseld, hoeft dit vakje toch zelden te worden geopend.

4.3 Eisen aan de layout van de printplaten en realisatie ervan.

Binnen de behuizing van de Pocketstem zijn alle electronische componenten op twee printplaten gemonteerd: De topprint (de grootste van de twee) en de bottomprint.
De vorm en de grootte van deze printplaatjes zijn zodanig, dat ze precies passen in de ruimte binnen de behuizing, die het batterijvakje en het luidsprekergedeelte vrij laten. (Zie bijlage III)
Op de topprint, die (zoals de naam al zegt) onder de bovenzijde van de behuizing is bevestigd, zijn de "power-control", de spraaksynthesizer, de microprocessor, de EPROM met de bijbehorende latch en tenslotte de keyboard scanning gemonteerd. (Zie bijlage III: Op de topprint beginnen alle nummers van de daarop voorkomende componenten met '0' b.v. R001, D010, C005, IC01)

Om de lengten van de data-, adres- en besturingsbussen zo kort mogelijk te houden zijn de IC's, die hierop betrekking hebben, zo dicht mogelijk bij elkaar geplaatst.
Er bestaan van deze topprint 2 versies: De versie met een MEA8000 spraakchip en die met een PCF8200. De layouts zijn verschillend, daar de pinconfiguratie van beide verschillend is. Bovendien is de PCF8200 in een IC-voet geplaatst en is de MEA8000 rechtstreeks op de printplaat gesoldeerd.
Op de bottomprint (boven de onderplaat van de behuizing dus) zijn de componenten voor het audiogedeelte gesoldeerd. (Zie bijlage III: De nummers van de componenten op deze bottomprint beginnen met '1', b.v. IC11 en R105.) De werking van deze schakelingen wordt in hoofdstuk 5 uitvoerig behandeld.
Hier wordt nog vermeld, dat er 5 verbindingsdraden lopen tussen de top- en de bottomprint. Deze zorgen voor:

- De voedingsspanning van het audiogedeelte: V_{audio} (= 9 volt).
- De voedingsspanning van de relais-schakeling: U_{sch} (= 5 volt).
  (Het relais is noodzakelijk om geen in- en uitschakelklikken te krijgen via de luidspreker.)
- De sturing van de relais-schakeling : $\overline{Relais - on}$.
- Het analoge uitgangssignaal van de spraaksynthesizer:
  $Analoog_{out}$ van de MEA8000 of de PCF8200.
- Aarde.

Om verschillende redenen zijn enkele IC's niet meteen op de printplaat gesoldeerd, maar worden ze in IC-voetjes geplaatst: De EPROM, omdat de verschillende zinnensets uitwisselbaar moeten kunnen zijn (zie hoofdstuk 3), de processor en de spraaksynthesizer PCF8200, omdat de kans op beschadiging(en) bij vastsolderen groter is, dan bij het plaatsen in IC-voetjes.
Deze kans op beschadigingen geldt uiteraard ook voor de andere IC's, maar van de PCF8200 zijn pas enkele proef-exemplaren beschikbaar en de microprocessor is een dure chip in vergelijking met de andere. Voldoende reden om extra zuinig te zijn op deze IC's.
De ruimte aan de buitenzijde, tussen de printplaat en de binnenkant van de behuizing, is niet groot genoeg om daar de IC's-op-voetjes te kunnen plaatsen. Deze IC's moeten dus komen in de ruimte tussen de topprint en de bottomprint. (Deze ruimte wordt in het vervolg aangeduid als de tussenruimte.)

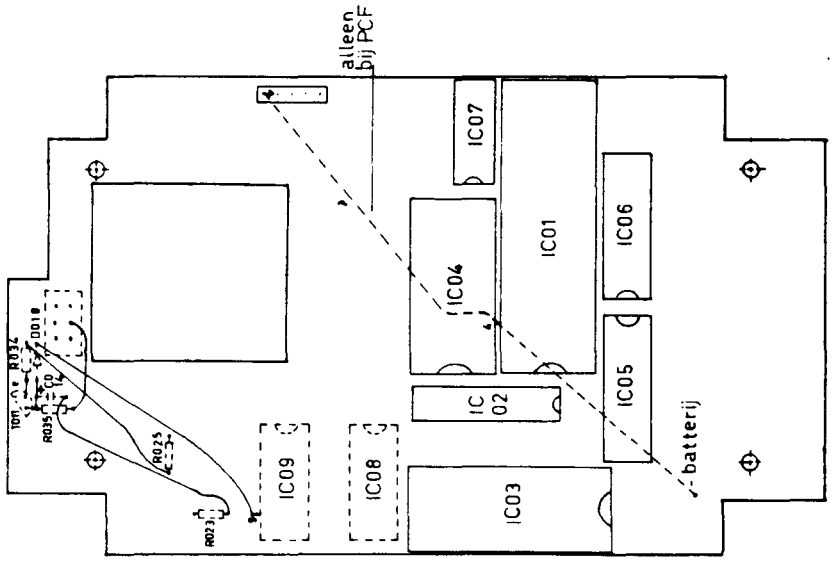
Doordat er voor de MEA8000 wel voldoende ruimte is aan de buitenzijde, omdat deze niet (zoals de PCF8200) op IC-voetjes is gemonteerd, is de MEA-print-layout zo ontworpen, dat de MEA8000 aan de buitenzijde moet worden gesoldeerd.

Bij het ontwerpen van de layout van de beide printplaten moet er rekening mee worden gehouden, dat er voldoende ruimte overblijft voor die IC's, die in IC-voetjes worden geplaatst. Er moeten daarom op de bottomprint zo weinig mogelijk componenten aan de kant van de tussenruimte komen, omdat anders de hoogte van de tussenruimte ontoereikend is. Dit levert voor deze print nauwelijks ontwerpproblemen, omdat er royaal voldoende ruimte is voor de benodigde audio-componenten op de bottomprint aan de buitenzijde.
Alleen het reed-relais (op de bottomprint) is te hoog om het aan de buitenzijde te gesoldeerd te worden. Bij de plaatsindeling van de componenten op de topprint moet er dus voor gezorgd worden, dat er voldoende ruimte over is voor dit relais in de tussenruimte.
Hetzelfde geldt ongeveer voor potmeter R033. Deze potmeter, voor de afregeling van de brugversterker, past wel aan de buitenzijde, maar hij moet (indien mogelijk) kunnen worden ingesteld vanaf de tussenruimte, zodat je niet de printplaat hoeft los te schroeven van de behuizing.
Tenslotte moet er op de topprint, aan de kant van de tussenruimte, voldoende plaats worden gereserveerd voor de connector van het LCD-display. Dit display is nodig bij de evaluatie, om uit het RAM-geheugen te kunnen lezen, hoe vaak een bepaalde zin is geselecteerd.

Bij het ontwerpen van de layout van de printplaten is gebruik gemaakt van het programma SMARTWORK. Dit is een programma, dat kan worden gebruikt op IBM-compatible personal computers. [16]
De steekgrootte (de afstand tussen de gaatjes in de printplaat) is hiermee in te stellen op de standaardgrootte en aan beide zijden van de printplaat kunnen printbanen worden getekend. Dit tekenen kan handmatig gebeuren, maar bij eenvoudige gevallen bestaat er autorouting: Het programma zoekt een route, die dan uiteraard geen andere route snijdt.
Bij het creëren van bevestigings-gaatjes, waarin de componenten gestoken moeten worden, volgt er een waarschuwing als er reeds een route aan de andere kant van de printplaat bestond. (Nu kun je de route verleggen of de component elders plaatsen.) Op deze manier zijn de dubbelzijdige printplaten ontworpen, rekening houdend met het ruimtegebrek in tussenruimte.

Het resulterende ontwerp kan via een plotter of een goede printer worden afgedrukt.
Na uitvoerige controle van het resultaat kan van deze afdrukken een fotonegatief worden gemaakt, als er geen ontwerpfouten meer worden aangetroffen. Hiermee kan een dubbelzijdige printplaat worden vervaardigd.
Je hoeft nu dus niet meer de printbanen zelf op te plakken en precies alle steekafstanden tussen de gaatjes uit te meten, voordat dat resultaat verder kan worden verwerkt door de fotograaf en de printenmakerij.
Dit levert niet alleen een enorme tijdsbesparing op, maar het resulterende ontwerp is nu ook netter geworden. Dit nettere eindprodukt vermindert uiteraard ook de kans op ontwerpfouten.
Een groot nadeel van de nu beschikbare printjes is, dat ze wel dubbelzijdig zijn, maar dat ze niet zijn doorgemetaliseerd. Hiervoor zijn geen faciliteiten aanwezig op de printenmakerij van de TUE en uitbesteding bleek door omstandigheden geen haalbare kaart.
De gewenste doormetaliseringen moet je er nu dus met de hand in solderen. Ook moet je componenten aan beide zijden vastsolderen, als er vanuit beide zijden printbanen lopen.
(Je moet wel oppassen, dat eerst de doormetaliseringen zijn aangebracht, voordat je daaroverheen een IC plaatst !)

In het RESET-netwerk zijn enige veranderingen aangebracht, die niet in de print lay-out zijn opgenomen. Er zijn 11 extra gaatjes geboord, waarin de volgende onderdelen moeten worden geplaatst: R034, R035, T011, D018 en C014. (C014 zat al wel in schakeling, maar D018 is nu tussen pin 9 van IC09 geplaatst. Daardoor moest C014 verplaatst worden.) In figuur 4.5 wordt aangegeven welke extra verbindingen er nu moeten worden gelegd met bedrading.



Figuur 4.5: Extra bedrading t.b.v. het RESET-netwerk.

De diode bij de oplaadplug van de NiCd-accu (D017) is niet op de print gemonteerd, maar is met de anode verbonden aan de oplaadplug en de kathode is verbonden aan de schakelaar (V_{bat}). De batterij-leeg detectie LED (D006) is met 2 draadjes verbonden met de printplaat. De schakelaar SW01 is ook met draadjes verbonden met de printplaat. SW01 bevat, evenals bij de aansluitingen op de printplaat, doorverbindingen van de diagonale kontakten (zie bijlage III). Dit bespaart 2 draadjes. De tussenliggende contacten zijn weer verbonden met de aarde en de batterijspanning.

5 Beschrijving van de hardware.

Dit hoofdstuk bevat de beschrijving van het hardware-ontwerp. Allereerst wordt ingegaan op de eisen die aan de hardware gesteld worden. Vervolgens zal de systeemconfiguratie besproken worden, waarna onderdelen van het ontwerp wat nader zullen worden behandeld. Indien de systeemonderdelen essentieel afwijken van het ontwerp van de CSH I [13] zullen ze uitgebreid worden besproken, anders zullen de verschillen met het vorige ontwerp worden aangegeven en zal voor nadere informatie doorverwezen worden.

5.1 Uitgangspunten bij het ontwerp.

Overwegingen die golden voor het ontwerp van de CSH I [13], zijn grotendeels ook geldig voor het ontwerp van de Pocketstem. Echter door de beschikbaarheid van geavanceerde μ-processoren, geheugens en een nieuwe spraaksynthesizer (de Philips PCF8200) is het mogelijk geworden het ontwerp efficiënter, kleiner van formaat en beter van kwaliteit te maken. Uiteraard zijn ook de uit de evaluatie naar voren gekomen tekortkomingen in het nieuwe ontwerp verwerkt.

De algemene eisen, gesteld aan het ontwerp, zijn identiek gebleven aan die van de CSH I. Aangezien het gaat om het realiseren van een compact, draagbaar hulpmiddel is bij het uitwerken steeds rekening gehouden met:

a - Het gebruik van kleine complete elementen.

b - Het zorgdragen voor voldoende componenten om de gewenste functies te kunnen uitvoeren.

c - Het ontwerpen van schakelingen, die met weinig componenten gerealiseerd kunnen worden.

d - Het gebruik van CMOS-componenten, teneinde het stroomverbruik binnen aanvaardbare grenzen te houden, en het ontwerpen van schakelingen, die weinig stroom verbruiken.

e - Het ontwerpen van een power-control schakeling, die zorgt voor een economisch stroomverbruik.

Zoals al aangegeven zijn, door de veranderende techniek, enkele eisen (met name de eisen a,c en d) beter realiseerbaar. De verbeteringen zullen hier even vernoemd worden en in de rest van dit hoofdstuk in detail beschreven worden. Het systeem is opgebouwd rond een μ-processor uit de

MCS-51 serie van Intel [5] met power-down fasciliteiten (eisen a + b + d), een gecombineerd program/data-memory (c), een eenvoudige keyboard-scanning met stand-by mogelijkheid (b + c), een nieuwe spraaksynthesizer [1] (de PCF8200) (d) en een powercontrol-circuit (e). Van het ontwerp is een printed-circuitboard gemaakt daar er, ten behoeve van de evaluatie, meerdere exemplaren van de Pocketstem gerealiseerd moesten worden. De algemene kenmerken van het ontwerp zijn:

- De Pocketstem (althans het keyboard ervan) kan uitgeschakeld worden i.v.m. transport.
- Een LED geeft aan dat de accu leegraakt (zie Gebruiksaanwijzing in bijlage B).
- Een brug-geschakelde audio-eindversterker zorgt voor voldoende uitgangsvermogen (2,6 Watt).
- Eén EPROM (32k×8 bit) bevat het programma, een dubbele (2×28) set zinnen (spraakdata) en de teksten van deze zinnen.
- Een aansluitmogelijkheid voor een display is aanwezig om gebruiksresultaten uit te kunnen lezen.
- Het audio-volume is in vijf niveau's instelbaar.
- Geen extern RAM is nodig daar het interne RAM van de μ-processor voldoende groot is en de processor in power-down mode nauwelijks stroom verbruikt. Het RAM wordt gebruikt om, ten behoeve van de evaluatie, bij te houden hoe vaak een zin gebruikt is.

5.2 Systeemconfiguratie.

Het ontwerp is opgebouwd rond de 80C31, een CMOS μ-processor uit de Intel MCS-51 serie. Deze processor heeft on-board:

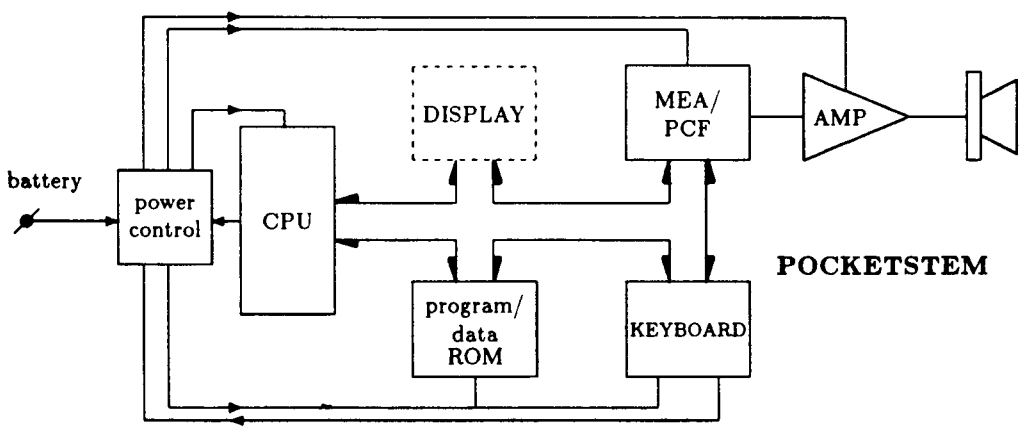
- Een 128-byte RAM.
- Verscheidene Interrupt mogelijkheden (2 externe, 2 timer en 1 seriële Interrupt met priority levels).
- Oscillator en clock-circuitry.

[1] Aangezien deze ten tijde van het gebruik van de Pocketstem nog niet leverbaar was, is het ontwerp ook gemaakt voor de MEA8000.

- Een 64k programma- en een 64k data-adres bereik.
- Etc. (zie Microcontroller Handbook [5]).

De 80C31 heeft geen on-board ROM. Deze versie is gekozen om twee redenen, te weten:

- In de ontwerp-fase moet een EPROM als programmageheugen gebruikt worden (i.v.m. software ontwikkeling) en er zijn nog geen betaalbare CMOS μ-processoren met on-board EPROM op de markt.
- De Pocketstem heeft naast zijn programmageheugen (max. 2kbyte) altijd een veelvoud daarvan aan datageheugen nodig (max. 30kbyte), hetgeen in dit ontwerp gecombineerd is tot een geheugen van 32kbyte (dus extern programmageheugen levert geen ruimteverlies op).



Figuur 5.1: De systeemconfiguratie van de Pocketstem.

Rondom de μ-procesor zijn de volgende circuit-onderdelen (zie Figuur 5.1) te ontdekken:
1. Een keyboard-scan circuit. 2. Een 32kbyte EPROM (zie memory-map: Figuur 6.1). 3. Een power-control circuit. 4. Een spraaksynthesizer

(MEA8000 of PCF8200) met audio-versterker. 5. Een connector om een display aan te sluiten.

Tabel 5.1: Indeling van de I/O-poorten van de μ-processor.

P0.7		P1.7	$\overline{LED}$; LED stuurlijn
P0.6		P1.6	bit2; power-controllijn
P0.5		P1.5	bit1; power-controllijn
P0.4	data/	P1.4	$\overline{relais-on}$; relaisstuurlijn
P0.3	adresbus	P1.3	$\overline{ce(keyboard)}$ keyboard-enable lijn
P0.2		P1.2	$\overline{ce(mea)}$; chip-enable voor MEA/PCF
P0.1		P1.1	A0; commandlijn voor MEA/LCD
P0.0		P1.0	$\overline{LCD}$; display-enablelijn
P2.7	n.c.	P3.7	$\overline{rd}$
P2.6		P3.6	$\overline{wr}$
P2.5		P3.5	n.c. wisram[1]; inputlijn
P2.4	hoge orde	P3.4	switch; inputlijn voor schakelaarstand
P2.3	adresbus	P3.3	$int1$; inputlijn voor batterij-controle
P2.2		P3.2	$int0$; interrupt van MEA/PCF
P2.1		P3.1	txd; zendlijn van serieel buffer
P2.0		P3.0	rxd; ontvangstlijn van serieel buffer

De μ-processor heeft 4 I/O-poorten van 8 lijnen met user-instelbare functies. Het gebruik van een extern geheugen houdt in dat Poort 0 (de gemultiplexte data/adres-bus) en Poort 2 (hoge orde adres-bus) niet als I/O-poort kunnen worden gebruikt. Met deze 16 adreslijnen kan men maximaal 128k-byte geheugen adresseren (64k-byte programma-geheugen aangestuurd d.m.v. de $\overline{PSEN}$-puls en 64k-byte data-geheugen aangestuurd door de read-puls). In de Pocketstem is 32kbyte programma-geheugen (via de $\overline{PSEN}$-puls gestuurd; zie micro-controller handbook [5]) gebruikt voor zowel het programma- als het spraakdata-geheugen. Poort 3 is voornamelijk voor andere zaken in gebruik (2$\times$interrupt, $\overline{RD}$, $\overline{WR}$ en de 2 seriële lijnen). Alleen P3.4 en P3.5 kunnen als testlijnen worden gebruikt. Poort 1 is gebruikt als control-bus. De indeling van de poorten is opgenomen in Tabel 5.1.

[1]Indien aan massa gelegd, reset het programma het RAM geheugen.

5.3 De μ-processor en de via de data-bus aangesloten onderdelen.

Informatie omtrent de timing van de μ-processor is te vinden in het microcontroller handbook [5] en informatie omtrent de aangesloten systeemonderdelen is grotendeels te vinden in het rapport van de CSH I [13]. Een aantal verschillen en nieuwe zaken zullen hier uitvoerig besproken worden. Op de μ-processor zijn een geheugen, een keyboardscan-circuit en een spraaksynthesizer aangesloten en kan een display worden aangesloten. In de bespreking van de onderdelen wordt telkens verwezen naar de in bijlage A.1 opgenomen schema's van de schakeling.

5.3.1 Het geheugen.

Zoals al eerder vermeld is, is het geheugen van de Pocketstem iets anders dan dat van de CSH I. De verschillen zijn:
Het externe RAM is niet meer nodig, daar het interne RAM van de μ-processor groot genoeg is (128 byte) en omdat de μ-processor permanent op spanning gehouden wordt (power-down mode). Program- en Data-ROM zijn gecombineerd, hetgeen mogelijk werd door het grotere adresseerbereik van de MCS-51 processoren en het verkrijgbaar worden van 27C256 (32kbyte$\times$8) EPROM's.

De aansturing van het geheugen gebeurt door middel van het Program-Store-ENable signaal (PSEN) [5] van de processor (timing-structuur van het uitlezen van program-memory). Het LO-adres wordt via de databus naar de latch gestuurd en met het Adress Latch Enable signaal ingeclockt. Het HO-adres staat gedurende de instructie-leescyclus op Poort 2. De PSEN-puls is de output enable voor het EPROM. Daar dit geheugen het programma bevat, moet het gedurende de werking van de μ-processor permanent geselecteerd zijn. In power-down toestand is het echter noodzakelijk om de chip-select lijn van het EPROM inactief te maken daar dit een forse besparing in het stroomverbruik oplevert (I_{cc}= 30mA (actief, tijdens het lezen), I_{cc}= 1mA (stand-by, wel geselecteerd) en I_{cc}= 1μA (typical stand-by, niet geselecteerd)). De geheugenindeling (memory-map) is te vinden in hoofdstuk 6.2.

5.3.2 De spraaksynthesizer.

De Pocketstem is ontworpen voor het gebruik met een nieuwe spraaksynthesizer van Philips (de PCF8200 [4]). Omdat echter tijdens het ontwerpen al vrij snel duidelijk werd dat de PCF8200 niet op tijd in productie genomen zou worden, is het ontwerp ook gemaakt voor gebruik met een MEA8000. De aansluiting en aansturing van de MEA8000 (inclusief het spanningsloos maken als hij niet gebruikt wordt) is identiek aan die beschreven in het rapport van de CSH I [13] en zal hier niet verder beschreven worden. Het gebruik van de PCF8200 echter levert enkele verschillen op die hier (hardware verschillen) en in hoofdstuk 6.4 (software verschillen) zullen worden besproken. De hardware veranderingen zijn:

- De PCF8200 kan permanent onder spanning gehouden worden (stand-by current typical $<20\mu$A).
- Er is een gescheiden voedingsspanning-aansluiting voor het digitale en voor het analoge deel van het IC ter voorkoming van hoogfrequente storing in het audio-signaal (V_{aa} kan via een LC- of een RC-schakeling worden afgeleid van V_{dd}).
- De frequentie waarop de PCF8200 werkt is 6 MHz. (hierdoor is ook voor de systeemfrequentie 6 MHz. gekozen).
- Er is veel minder externe filtering van het analog-out signaal nodig.
- Er is geen aparte command-line meer, de PCF8200 ontvangt zijn commando's via protocollen.
- De status van de PCF8200 wordt software-matig uitgelezen. Hierdoor zal wel de $\overline{RD}$-lijn op de PCF8200 moeten worden aangesloten.
- De pull-up weerstand van de request-lijn (MEA8000, open-drain output NMOS) is niet meer nodig, daar de PCF8200 een CMOS device is.

Er is ook nog een verandering t.o.v. de Application notes [4] van Philips, te weten: De referentiespanning die de PCF8200 nodig heeft, wordt via een hoogohmigere spanningsdeler verkregen (ong. $10\times$ hoogohmiger). Dit heeft zowel een voordeel als een nadeel. Het voordeel is dat er niet nodeloos veel stroom verbruikt wordt voor een instelling (5 t.o.v. 50 μA). Het nadeel is

dat de referentiespanning bij het opkomen van de voedingsspanning pas na 2 seconden op niveau is (RC-tijd). Aangezien de Pocketstem in zijn geheel in stand-by ong. 30 μA verbruikt en, behalve bij het wisselen van de batterij, altijd op spanning gehouden wordt, is de hoogohmigere spanningsdeler hier prima toepasbaar.

5.3.3 De keyboardscanning.

De keyboardscanning van de Pocketstem is nagenoeg identiek aan die van de CSH I [13], alleen bij de Pocketstem wordt het scan-adres niet met het ALE-signaal maar met het $\overline{WR}$-signaal ingelezen. Dit houdt in dat er nu twee instructies nodig zijn om een keyboard-adres uit te lezen i.p.v. een. Maar het feit dat het $\overline{WR}$-signaal makkelijker met het keyboard-enablesignaal te combineren is, het feit dat de scan-snelheid voornamelijk door toetsdender (ong. 10 ms.) wordt bepaald en het feit dat op deze manier de instructie die FF(hex) naar het keyboard schrijft niet meer op een vaste plaats in het geheugen behoeft te staan hebben ertoe geleid, dat voor de huidige schakeling is gekozen. Het keyboard (zie hoofdstuk 4.2) heeft 28 toetsen met een 7×4 matrix. Gekozen is voor een uitlezing met 7 stuurlijnen (scan-adressen) en met 4 retourlijnen, omdat in de retourlijnen pull-down weerstanden en dioden (t.b.v. Power-up keyboard signaal) moeten worden opgenomen.

5.3.4 LCD-display.

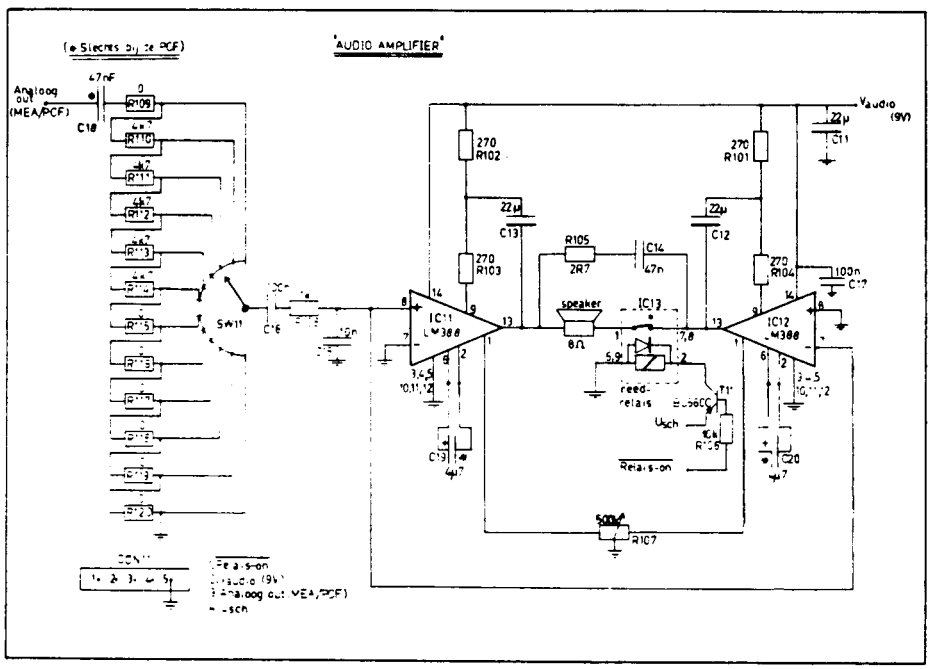
Zoals uit het systeemoverzicht (figuur 5.1) naar voren komt en ook uit de algemene beschrijving (hoofdstuk 5.2) blijkt, kan er wel een display op de Pocketstem worden aangesloten, al is deze niet ingebouwd. Uit de evaluatie van de CSH I bleek dat een display geen functie had bij het gebruik van het hulpmiddel. Een display is dan ook overbodig voor het gebruik van de Pocketstem. Maar omdat we gedurende de evaluatie wel een inzicht willen hebben in de intensiviteit van het gebruik van de Pocketstem, is er wel een display-aansluiting uitgevoerd. Hierop kan een display (Epson EA-X16027AR) worden aangesloten. Het display moet van een eigen voeding voorzien zijn, moet zelf de 4MHz. (6MHz. bij de PCF8200) clock delen en ook de contrast-regeling is niet op de print van de Pocketstem gerealiseerd. De aansturing is identiek aan die van het display van de CSH I [13].

5.4 De audioversterker en het power-controlcircuit.

In dit gedeelte van het verslag zullen de twee nog niet besproken systeemonderdelen worden behandeld.

5.4.1 De audioversterker.

De beschrijving van deze audioversterker (zie figuur 5.2) is voor het eerst te vinden in het rapport van de Tiepstem [3]. De audioversterker is opgebouwd rond twee LM388 audio-opamps. Met deze opamps is een brugschakeling gerealiseerd conform de 'Typical Applications' [7] op een paar details na.



Figuur 5.2: De audioversterker.

Ten eerste is hier evenals bij de CSH I [13] een inputfilter toegepast (R108,C15 zie figuur 5.2) met als doel oscillatie te voorkomen en hoogfrekwente storing in het audiosignaal weg te filteren. Ten tweede is de speaker (Philips AD2071/Z8) via een relais geschakeld. Dit is gedaan om in- en uitschakelklikken (de audioversterker gaat alleen aan als er iets moet worden uitgesproken) te voorkomen. Het relais is hier echter iets anders gebruikt

dan bij de CSH I. Werd bij de CSH I de speaker in de signaallijn geschakeld na het inschakelen van de spanning en uit de signaallijn voor het uitschakelen van de spanning, bij de Pocketstem wordt de speaker alleen tijdens het schakelen even uit de signaallijn gehaald. Bij de CSH I verbruikte het relais dus stroom (20 mA) tijdens het spreken, bij de Pocketstem verbruikt het relais slechts gedurende 2 $\times$ 20 ms. stroom per keer spreken.

Het uitgangsvermogen van deze audioversterker is theoretisch:
$P_{audio} = (V_{eff})^2/R_{bel.} = (0.5\sqrt{2} \times 7.6)^2/8 = 3.6$ Watt
Het maximale uitgangsvermogen (met 10% T.H.D.) is gemeten door R. Deliege [3] en bedraagt 2.6 Watt. De geluidssterkte van de Pocketstem kan door de gebruiker worden ingesteld met behulp van een meerstanden schakelaar. Het aantal niveau's is beperkt tot 5.

Opgemerkt dient nog te worden dat de PCF8200 een iets lagere audiospanning levert dan de MEA8000, hetgeen opgevangen kan worden door de versterking van de LM388's op te voeren van 20 (bij de MEA8000) tot 200 (nodig voor de PCF8200). De versterking van een LM388 kan 10 maal verhoogd worden door een condensator van 4.7μF aan te sluiten tussen de pootjes 2 en 6 van het IC.

5.4.2 Het power-controlcircuit.

De voeding van de Pocketstem wordt geleverd door een 9 Volt/100mAh Ni-Cd oplaadbare accu (Tr 7/8, no.5022 IEC(6F22)). Uit deze voeding moeten een aantal verschillende spanningen en signalen gerealiseerd worden. Zo moet een gedeelte van de schakeling permanent met 5 Volt, een gedeelte alleen tijdens het spreken met 5 Volt en een gedeelte tijdens het spreken met 9 Volt gevoed worden. Tevens zorgt het power-controlcircuit voor het resetten van de processor, het geven van een chip-enable aan het programmageheugen en het detecteren van een te lage voedingsspanning. Aangezien het power-controlcircuit wezenlijk verschilt van dat van de CSH I, zullen in dit gedeelte van het hoofdstuk de functies van het power-controlcircuit nader besproken worden. Voor het schema wordt weer verwezen naar bijlage A.1.

De Pocketstem kent een drietal modes waarin hij zich kan bevinden. We zullen nu deze modes en de signalen, die nodig zijn bij het wisselen van de modes, beschrijven. De drie modes zijn:

- Stand-by (de processor in power-down mode en alle componenten op minimaal stroomverbruik).
- Active (de processor werkt, maar de Pocketstem spreekt niet).
- Speaking (de meest stroomverbruikende mode, de Pocketstem spreekt een zin uit).

De overgangen tussen de modes zijn als volgt:
Stand-by $\Longleftrightarrow$ Active $\Longleftrightarrow$ Speaking

In Stand-by moet het circuit, met uitzondering van de MEA8000 en de audioversterker, voorzien worden van 5 Volt, maar met zo min mogelijk stroomverbruik (Back-up spanning). Het stroomverbruik van de gehele schakeling is dan ong. 30 μA.

Bij de overgang naar Active (veroorzaakt door de aan/uit schakelaar of door een toets van het keyboard) moet het power-controlcircuit een reset genereren, een enable-signaal geven aan het programmageheugen en het gehele circuit voeden met 5 Volt (het stroomverbruik van het power-controlcircuit loopt nu op tot enkele mA's).

In Active moet het power-controlcircuit de stand van de schakelaar doorgeven aan de processor, de batterijspanning controleren en doorgeven aan de processor en de 5 Volt voeding verzorgen. Het stroomverbruik van de schakeling is nu ong. 15 mA.

Bij de overgang naar Speaking (gestuurd door de processor) moet het power-controlcircuit de MEA8000 en de audioversterker van voeding gaan voorzien.

In Speaking controleert het power-controlcircuit de batterijspanning en verzorgt het het gehele stroomverbruik. Het stroomverbruik is nu minimaal 80 mA., maar is natuurlijk sterk afhankelijk van het gevraagde audiovermogen.

De overgangen van Speaking naar Active en van Active naar Stand-by worden, evenals de overgang van Active naar Speaking, gestuurd door de processor. We zullen deze sturing eens nader beschouwen. Om drie verschillende modes te kunnen besturen zijn minimaal 2 stuurlijnen nodig. Deze stuurlijnen zijn bit1 en bit2 genoemd (zie Tabel 5.2). We zullen nu de verschillende modes definiëren aan de hand van de waarden van de stuurlijnen. Aangezien de processor bij reset (geïnitieerd door het power-controlcircuit) alle poortlijnen hoog maakt en het de bedoeling is dat bij reset de schakeling

Active wordt, is de Active mode gedefinieerd met: bit1=1 en bit2=1. De andere modes kunnen nu met een willekeurige andere combinatie van bit1 en bit2 beschreven worden. We kiezen natuurlijk die combinaties die het eenvoudigst zijn bij de realisatie van de besturing.

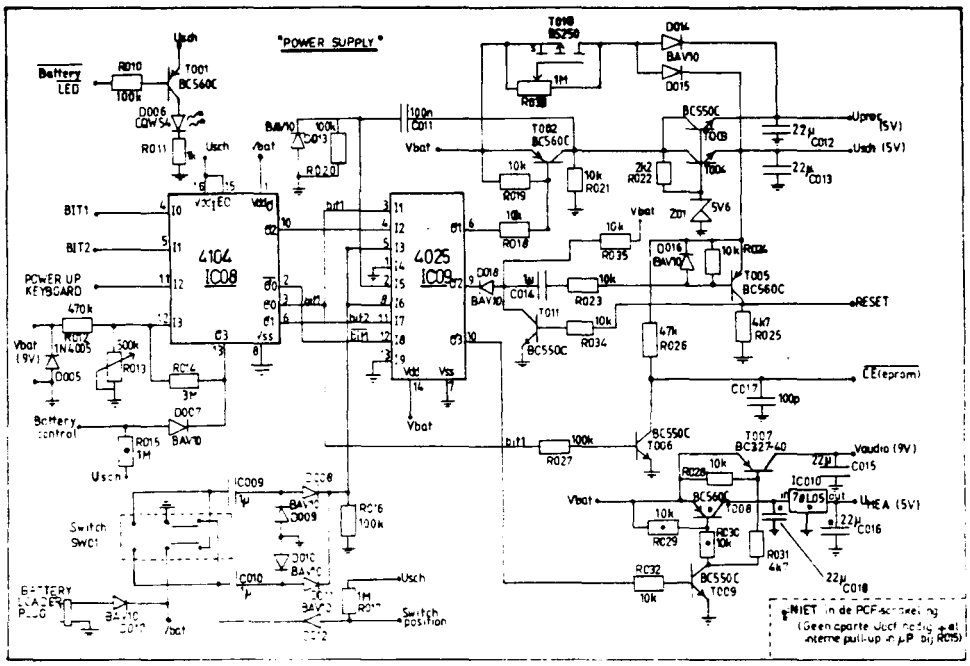
Tabel 5.2: De waarden van de stuurlijnen en de bijbehorende modes.

bit1	bit2	Mode
1	1	Active
1	0	Speaking
0	1	Stand-by
0	0	Not in use

In feite valt uit Tabel 5.2 af te lezen dat bit1 bepaalt of de processor actief is of niet en dat bit2 bepaalt of het audio-gedeelte (met de MEA8000) van spanning moet worden voorzien. Het spreekt dan ook voor zich dat de combinatie bit1=0 en bit2=0 (processor power-down en audio-gedeelte aan) niet voor mag komen.

We zullen aan de hand van figuur 5.3 de werking van het power-controlcircuit eens nader beschouwen. De twee stuurlijnen worden samen met de terugmelding van het keyboard (Power-up keyboard) door IC08 (level-converter) van 5 Volt logisch niveau naar 9 Volt omgezet. Dit is nodig daar de triple three-input NOR (IC09), die de praktisch alle schakelfunkties realiseert, rechtstreeks op de batterij is aangesloten, teneinde 9 Volt te kunnen schakelen. Op IC09 is ook via een differentiërend netwerkje de schakelaar (SW01) aangesloten. Aan de hand van eerder vermelde modes en hun overgangen zal het realiseren van de schakelfunkties worden besproken.

In Stand-by wordt de schakeling gevoed door het back-up circuit (hetgeen later besproken zal worden). In deze situatie zijn de ingangen I1, I2 en I3 (IC09) allen laag (bit1=0, geen keyboard-input en geen verandering in de stand van de schakelaar), zodat output O1 hoog is en T002 dus spert. Zo is ook O2 hoog (dus reset is niet actief). Echter O3 is laag want $\overline{bit1}$ en bit2 zijn hoog (dus het audio-gedeelte is spanningsloos).



Figuur 5.3: Het power-controlcircuit.

Bij de overgang van Stand-by naar Active, door een (positieve) puls op power-up keyboard of het veranderen van de stand van de schakelaar, zal I2 resp. I3 (van IC09) even hoog worden en dus O1 laag. Als O1 laag is zal de voedingsspanning (5 Volt), via de spanningsstabilisator rond T003 en T004, in staat zijn de schakeling van max. 100mA te voorzien. Tevens zal via C011 een positieve puls op I5 ontstaan hetgeen via het circuit achter O2 een resetpuls oplevert. Bij een reset op de processor gaan alle poortlijnen hoog, dus ook bit1, zodat deze via I1-O1 (IC08) de spanningsverzorging op peil blijft houden en tevens via T006 een enable-signaal naar het programmageheugen stuurt. (Het Program Store ENable signaal kunnen we niet als chip-enable signaal gebruiken, daar PSEN in power-down van de processor laag is, hetgeen het geheugen juist zou selecteren.) De reden waarom de schakelaar via het differentiërend netwerkje ook met I6 is verbonden, is dat de processor altijd een reset moet krijgen als de schakelaar van stand verandert, ook al is de processor op dat moment actief (merk op dat de processor geen resetpuls

krijgt via C011 als de spanning via T002 al aanwezig is). Een laatste wijziging in het schema is de toevoeging van de one-shot (T005, T011), daar bij praktijkgebruik is gebleken dat korte stoorpulsen tot te korte resetpulsen leidden, waardoor het interne geheugen van de processor werd verminkt. De one-shot zorgt ervoor dat de resetpuls een gedefinieerde lengte van $\approx$ 10 ms. (R023 $\times$ C014) heeft.

Eenmaal Active hoeft het power-controlcircuit alleen maar stroom te leveren en via de rond IC08 gesitueerde voedingsspanningsdetector de processor te melden of de spanning nog voldoende hoog is.

De overgang van Active naar Speaking vereist verder geen aktie van het circuit. Alleen moet nu de voedingsspanning voor de MEA8000 en het audiogedeelte ingeschakeld worden (via T009). Deze situatie blijft gehandhaafd gedurende de tijd dat de schakeling in de mode Speaking is.

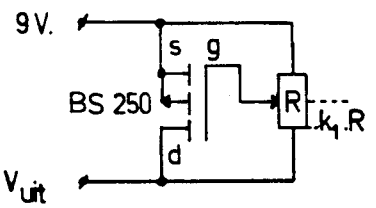
De overgangen terug van Speaking via Active naar Stand-by worden gerealiseerd door bit2 hoog resp. bit1 laag te maken. Dan worden de spanning voor het audio-gedeelte resp. de spanning voor de schakeling afgeschakeld. Bij het laag maken van bit1 wordt ook het chip-enable signaal van het programmageheugen inactief gemaakt.

Van het power-controlcircuit zijn nu nog slechts enkele zaken niet besproken, te weten:

1. Het 5 Volt back-up circuit.
2. Het batterij-controle circuit.
3. De batterij-leeg indicatie LED.

ad 1. Aangezien de schakeling in power-down nog steeds zo'n 5 Volt nodig heeft, maar slechts 30μA verbruikt, is een 5 Volt back-up circuit nodig dat het stroomverbruik niet essentieel verhoogt (anders zou de levensduur van de batterij onnodig worden bekort). De back-up spanning kan om voornoemde reden niet via een spanningsdeler worden gerealiseerd, daar deze alleen enige stabiliteit heeft als $R_{deler} \ll R_{belasting}$ en dus als $I_{deler} \gg I_{belasting}$. De stroom door de back-up spanningsregelaar zou in dit geval ong. 300μA worden, hetgeen ontoelaatbaar hoog is. Gekozen is voor

een seriële regelaar daar deze geen extra stroomverbruik introduceert. Er is een regelaar ontworpen (zie figuur 5.4) met een FET om er voor te zorgen dat de spanningsval weinig beïnvloed wordt door de te leveren stroom. We houden in dit verhaal geen rekening met de dalende batterijspanning, daar in het gedeelte over de batterijspanningsdetector zal blijken, dat de gebruikte batterij (oplaadbare accu) voornamelijk ong. 9 Volt levert.



Figuur 5.4: De FET-regelaar.

Als de regelaar 5 Volt moet leveren, zal zij zelf 4 Volt spanningsval moeten realiseren. Dat houdt in dat over de FET ong. 3.3 Volt moet vallen (9 -5 -0.7(diode) = 3.3). De FET is een BS250 p-channel enhancement VMOS FET. Voor de FET-schakeling geldt (zie figuur 5.4):

$$V_{gs} = k_1 \times V_{ds} \quad met\ 0 \leq k_1 \leq 1 \tag{1}$$

voor de BS250 geldt:

$$I_{ds} = -k_2 \times (V_{gs} - V_p)^2 \quad als\ V_{ds} < V_{gs} - V_p \leq 0 \tag{2}$$

waarbij $k_2 = 0.04$ en $V_p = -2.25$ Volt.
Aangezien V_{gs} een fractie is van V_{ds} (1), $V_{ds} \approx$ -3.3 Volt is en we niet geïnteresseerd zijn in $V_{gs} - V_p > 0$ (dan spert de FET namelijk), geldt (2) in de hier gebruikte schakeling altijd [1].

We gaan nu de FET-schakeling zo instellen, dat bij de minimale stroom door de FET-schakeling, de spanning over de belasting-schakeling op 5.5 Volt komt te liggen. De minimale stroom wordt bereikt als er geen stroom meer door de FET loopt. Er zal dan nog $\approx$ 2.9 Volt (9 - 5.5 - 0.6) over de FET staan, terwijl moet gelden: $V_{gs} = V_p$ ($I_{ds} = 0$). Hieruit kan k_1 berekend

[1] Aangezien we een p-channel FET gebruiken zijn alle spanningen negatief.

worden:

$$k_1 = \frac{V_p}{V_{ds}} = \frac{-2.25}{-2.9} = 0.77 \tag{3}$$

We weten nu ook de minimale stroom, die de schakeling moet leveren om binnen zijn specificaties te blijven. Als namelijk $I_{ds} = 0$ dan loopt er alleen stroom door de potmeter. Bij een spanning over de FET (en dus over de potmeter) van $\approx$ 2.9 Volt loopt er dus een minimale stroom van $\approx$ 3 μA.

De minimale spanning op de belasting-schakeling moet 4.5 Volt zijn (V_{FET} is dan $\approx$ 3.9 Volt), zodat we de maximale stroom die de regelaar kan leveren kunnen berekenen.

$$I_{dsmax} = -0.04(0.77\,V_{ds} + 2.25)^2 = -23\,mA. \tag{4}$$

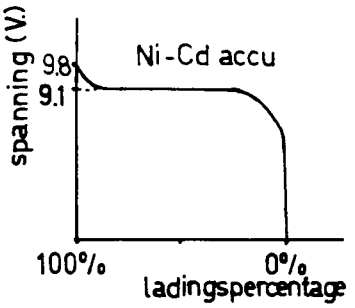
We kunnen dus stellen dat de regelaar de spanning ($\pm$10%) constant houdt bij een te leveren stroom tussen 3 μA en 23 mA. Tot slot kunnen we nog berekenen wat de belasting-spanning is bij een stroomafname van 30 μA.

$$V_{ds} = -\frac{\sqrt{\frac{I_{sd}}{0.04}} + 2.25}{0.77} = 3\,Volt \tag{5}$$

De belasting-spanning is dan 5.4 Volt. Bij het afregelen van de schakeling moet een volledig opgeladen accu aangesloten worden en moet de potmeter (R033) zodanig worden afgeregeld dat de back-up spanning 5.4 Volt is.

ad 2. Het batterij-controlecircuit werkt op het 5 Volt gedeelte van IC08. Het circuit werkt als de schakeling in de Active of Speaking mode is, hetgeen inhoudt, dat de spanning op de schakeling 5 Volt is (zolang $V_{bat} > \approx$7 Volt).

Aangezien de oplaadbare batterij (NiCd-accu) bij 8 Volt al voor meer dan 80% leeg is (zie figuur 5.5), kan deze spanning als laagst toegestane batterij spanning worden genomen. Door nu R013 zo in te stellen dat bij 8 Volt voeding uitgang O3 van IC08 net laag wordt, dan is de detector goed afgeregeld. De meekoppelweerstand R014 zorgt voor de nodige hysterese (zie figuur 5.3).



Figuur 5.5: Accu-karakteristiek.

ad 3. De uitgang O3 van IC08 wordt gelezen door de processor die, indien O3 laag is, een power-failure routine opstart (zie hoofdstuk 6.3). Een onderdeel van deze power-failure routine is het voor een aantal seconden laten branden van LED D006. De LED D006 wordt geschakeld door transistor T001, die op zijn beurt via R010 wordt aangestuurd door de processor.

6 Beschrijving van de software.

Dit hoofdstuk bevat de beschrijving van de software. Achtereenvolgens wordt ingegaan op de uitgangspunten bij het ontwerp van de software, uitleg over de programma-opzet en zijn werking, een bespreking van de gebruikte subroutines en tenslotte worden de verschillen in aansturing tussen de MEA8000 en de PCF8200 besproken.

6.1 Uitgangspunten van het programma.

De software van de Pocketstem moet een aantal zaken regelen, te weten:

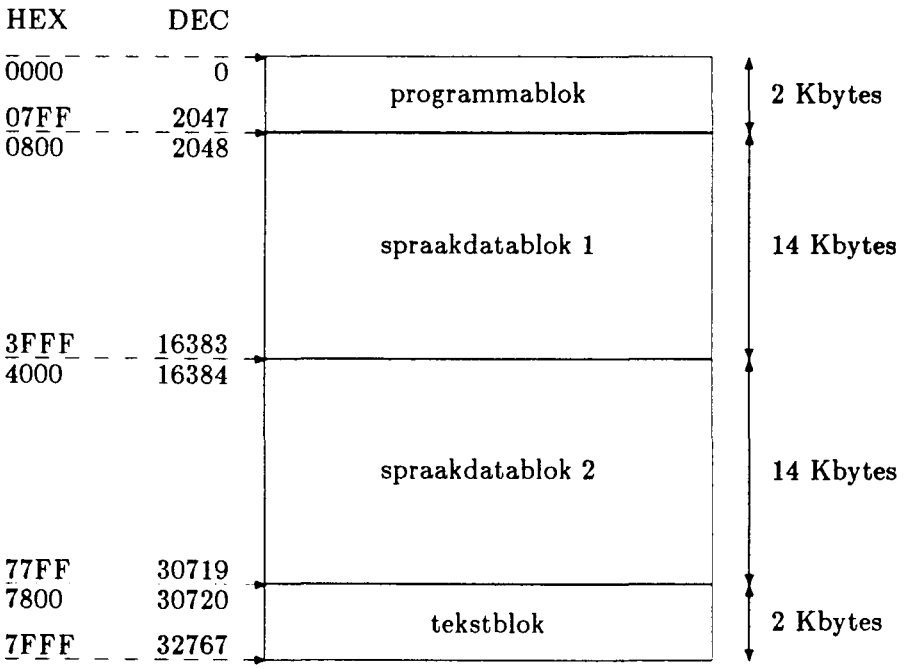
- Het uitlezen van de stand van de aan/uit-schakelaar (SW01).
- Het scannen van het keyboard en interpreteren van een toetsindruk (als een zin tweemaal achter elkaar wordt geselecteerd, dan moet de zin de tweede maal anders worden uitgesproken).
- Het uitschakelen van de Pocketstem zo gauw er niets meer behoefd te worden uitgesproken.
- Het registreren en displayen van het gebruik van de zinnen (ten behoeve van de evaluatie).
- Het ondernemen van actie als een te lage batterij-spanning wordt gedetecteerd.
- De audio-amp (en ingeval van de MEA8000 ook de synthesizer) van voedingsspanning voorzien en de spraaksynthesizer indien nodig van data voorzien.

6.2 Programmaopzet en beschrijving van de werking.

Alvorens op het besturingsprogramma in te gaan, zal eerst beschreven worden hoe het (programma- en data-)geheugen is ingedeeld. Ook zal eerst beschreven worden hoe het interne RAM van de μ-processor is ingedeeld en hoe de externe stuurlijnen in het programma zijn genoemd.

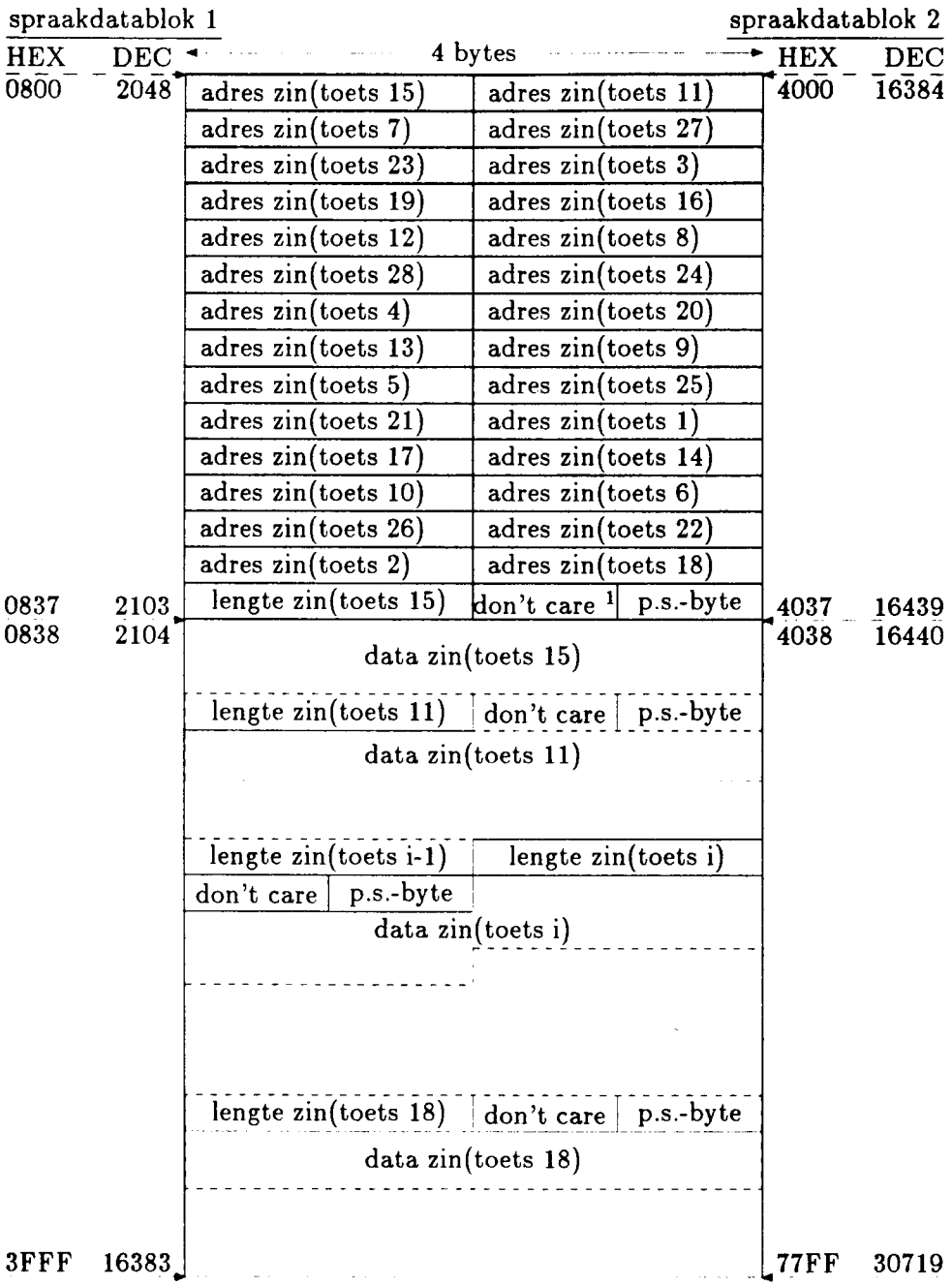
Het besturingsprogramma voor de μ-processor (max. 2 kbyte) is opgeslagen in het EPROM (27C256). Hierin bevindt zich ook de spraakdata en de tekstdata van de opgeslagen zinnen. De indeling van dit geheugen is weergegeven in figuur 6.1.

De routines die de spraakdata ophalen, zullen in hoofdstuk 6.3 weliswaar niet worden besproken (de routines zijn standaard en spreken voor zich), toch zal de indeling van de spraakdatablokken en het tekstblok hier beschreven worden. Het voor ogen hebben van de indeling van deze datablokken is namelijk essentieel voor het kunnen volgen van de programmalisting.



Figuur 6.1: Memory-map van het EPROM.

De twee spraakdata-blokken zijn qua indeling identiek (qua inhoud uiteraard niet). De indeling van zo'n spraakdata-blok is weergegeven in figuur 6.2.

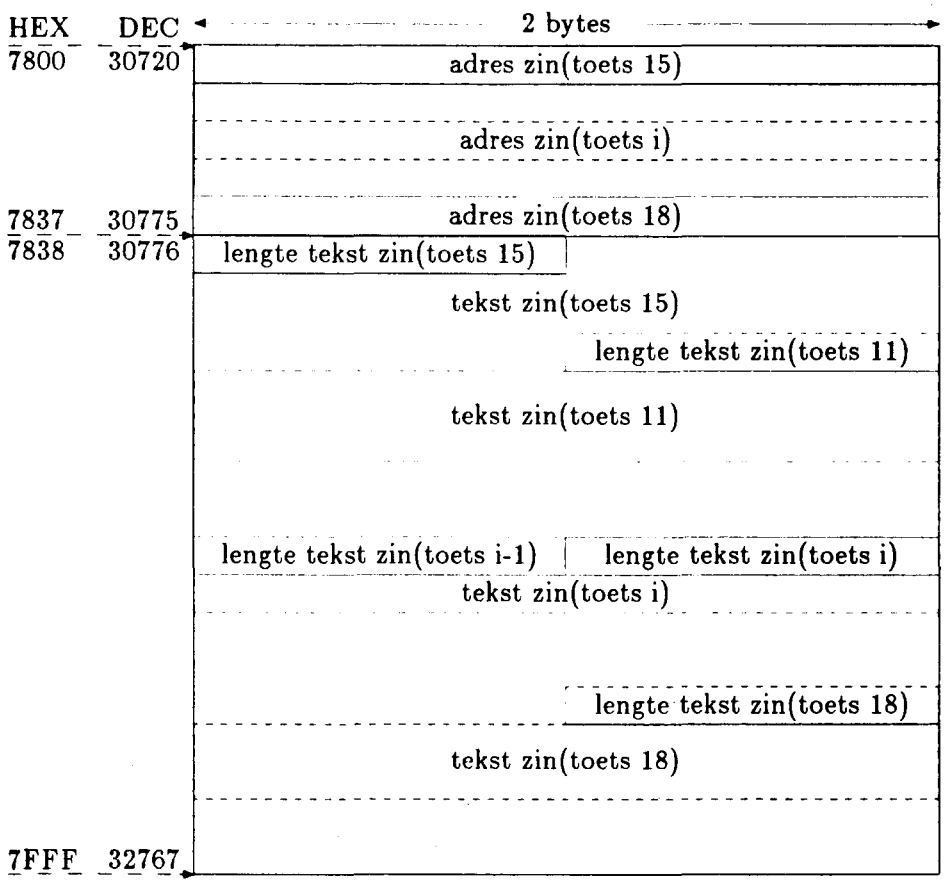


Figuur 6.2: Indeling van een spraakdatablok.

[1]Dit is voor de MEA8000, ingeval van de PCF8200 staat hier een don't care-byte gevolgd door een command-byte.

Bij de Pocketstem zijn in een spraakdata-blok maximaal 28 boodschappen opgeslagen. Toch ziet de configuratie van zo'n spraakdata-blok er altijd hetzelfde uit. Het spraakdata-blok begint met een adressenlijst (2 bytes/adres) die de plaats aangeeft waar de data van deze boodschappen zich bevindt. In geval van zin(toets 15) (zie figuur 6.2) zal dus in blok1 op adres 0800 het adres 0837 worden gevonden. Op adres 0837 wordt in 2 bytes de lengte van de boodschap vermeld (in figuur 6.2 is als voorbeeld zin(i-1) met lengte 0 opgenomen, dat wil zeggen dat die zin er niet in zit). Op het moment dat de processor dus het startadres en de lengte van de spraakdata heeft opgehaald, is het verder een kwestie van data ophalen en naar de synthesizer sturen totdat het einde van de zin is bereikt. Wat betreft de inhoud van de data wordt verwezen naar hoofdstuk 6.4, waar de verschillen tussen de MEA8000 en de PCF8200 worden besproken.

De indeling van het tekstblok is overeenkomstig de indeling van de spraakdatablokken en is weergegeven in figuur 6.3.



Figuur 6.3: Indeling van een tekstblok.

In de processor is 128 byte RAM aanwezig, hetgeen als volgt wordt gebruikt:
RAM registers 0 - 63: registers ten behoeve van het programma.
RAM registers 64 - 127: registers waarin het aantal malen dat een boodschap is gebruikt wordt opgeslagen.
De registers ten behoeve van het programma zijn onderverdeeld als beschreven in Tabel 6.1.

Tabel 6.1: Indeling van de eerste 64 RAM-plaatsen.

register(s)		functie
0	:	algemeen gebruik
1	:	algemeen gebruik
2	:	rij lengte (t.b.v. display routine)
3	:	tekst lengte (t.b.v. display routine)
4	:	L.O. tekst adres (t.b.v. display routine)
5	:	H.O. tekst adres (t.b.v. display routine)
6	:	speakbufferpointer (out)
7	:	speakbufferpointer (in) (reg. 0-7 liggen in reg. bank 0)
8	:	algemeen gebruik
9	:	algemeen gebruik
10	:	algemeen gebruik
11	:	algemeen gebruik
12	:	L.O. spraakdatapointer
13	:	H.O. spraakdatapointer
14	:	L.O. eindadres
15	:	H.O. eindadres (reg. 8-15 liggen in reg. bank 1)
16-23	:	niet in gebruik
24-31	:	speakbuffer [1]
32	:	Device Status Word (DSW)
33	:	niet in gebruik
34	:	pitch-startbuffer
35	:	spraakdatabuffer (PCF8200)
36-39	:	spraakdatabuffer (MEA8000 & PCF8200)
40-63	:	stack

De registers waarin het aantal malen gebruik van een boodschap is opgeslagen is onderverdeeld als beschreven in Tabel 6.2.

Tabel 6.2: Indeling van de RAM-plaatsen 65 t/m 128.

registernummer(s)		functie
64-65	:	aantal malen gebruik van zin 1 [2]
66-67	:	idem van zin 2
"	:	"
"	:	"
118-119	:	idem van zin 28

[1] Inputbuffer waarin de nummers van alle nog uit te spreken zinnen staan.
[2] BCD opgeslagen dus is maximaal 9999.

In de lijst met de door het programma gebruikte registers is sprake van het DSW (device status word). Dit DSW is een verzameling vlaggen die dienen om de status van de Pocketstem aan te geven. Overal in het programma kan men de status bekijken door simpelweg de bits van het DSW te checken. De verschillende bits hebben de betekenis zoals beschreven in Tabel 6.3.

Tabel 6.3: Beschrijving van de bits van het DSW.

Bit	Naam	Omschrijving
0	secout	Wordt geset als uit te spreken zin gelijk is aan de vorige.
1	-	Niet in gebruik.
2	leis0	Geeft aan dat de lengte van de uit te spreken zin nul is.
3	speane	Is hoog zolang het speakbuffer niet leeg is.
4	speaking	Geeft aan dat de Pocketstem spreekt.
5	key	Wordt geset als er de ingedrukte toets is gevonden.
6	devoff	Wordt geset als de aan/uit toets op 'uit' staat.
7	powfail	Wordt geset als er een power failure optreedt.

De meeste namen die in het programma voorkomen zijn hiermee verklaard. Er zijn echter nog een aantal namen in het programma, die enige beschrijving behoeven. In hoofdstuk 5 zijn alle poortlijnen benoemd, maar aangezien alle lijnen van poort 1 en 3 zgn. bit-adresseerbaar zijn hebben zij in het programma ook een naam gekregen. Deze namen komen niet altijd overeen en daarom zijn in de volgende tabel (Tabel 6.4) alle I/O-lijnen nog eens opgesomd met zowel de naam, die in het hardware-schema wordt gebruikt, als de naam die in het programma wordt gebruikt.

Tabel 6.4: Overzicht van de namen van de poortlijnen.

poortnummer	naam in schema	naam in programma
P1.7	$\overline{LED}$	nled
P1.6	bit2	pconb2
P1.5	bit1	pconb1
P1.4	$\overline{relais - on}$	relais
P1.3	$\overline{ce(keyboard)}$	keyen
P1.2	$\overline{ce(mea)}$	meaen
P1.1	A0	comm
P1.0	$\overline{LCD}$	lcden
P3.5	n.c. wisram	wisram
P3.4	switch	switch
P3.3	$\overline{int1}$	pcheckb[1]

We kunnen nu de werking van het programma nader bekijken omdat alle in het programma gebruikte namen zijn beschreven.

Aangezien de hoofdtaak van de Pocketstem is het uitspreken van een zin als er een toets is ingedrukt, is het hoofdprogramma tot een aantal elementaire routines terug te brengen. Wat moet er namelijk gebeuren?
De Pocketstem moet gaan werken als de aan/uit-schakelaar van stand verandert of als er op een toets van het keyboard wordt gedrukt (dit laatste als de Pocketstem aanstaat). Als eerste zal de processor controleren of de aan/uit schakelaar (SW01) in de stand 'aan' staat. Is dat niet het geval, dan moet het keyboard inactief gemaakt worden en de processor power-down gaan. Als de aan/uit schakelaar in de stand 'aan' staat, dan moet de processor de voedingsspanning controleren. Als deze niet voldoende is dan moet de spannings-controle LED een aantal seconden oplichten en moet de Pocketstem vervolgens weer 'uit' gaan. Als de voedingsspanning in orde is bevonden, dan moet de processor het keyboard gaan scannen. Wordt er geen toets gevonden die ingedrukt wordt, dan moet de Pocketstem stand-by gaan, d.w.z. keyboard actief en processor power-down. Wordt er wel een toets gevonden die ingedrukt is, dan moet de Pocketstem de onder die toets opgeslagen zin uitspreken (tijdens het uitspreken wordt het aantal malen gebruik geregistreerd en de tekst en het aantal malen gebruik van een zin naar een eventueel aangesloten display gestuurd). Na het uitspreken van een

[1]Deze lijn wordt door het programma getest en werkt pas tijdens het spreken als interruptlijn.

zin (c.q. meerdere zinnen) begint het programma weer van voren af aan. Dit is in figuur 6.4 van paragraaf 6.3, in een flowchart, uitgebeeld.

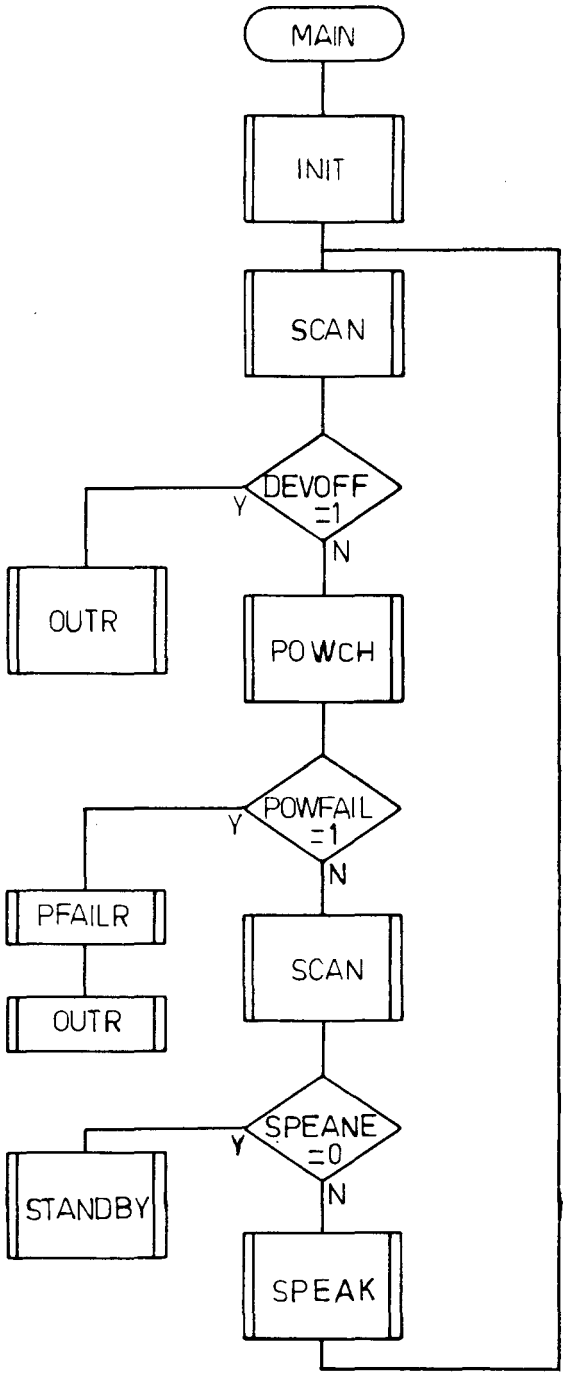
Teneinde de listing van het programma te kunnen lezen, is het handig om te weten dat het maken van het programma in drie stukken is gebeurd. Ieder stuk programma is in een aparte file gezet. Deze files heten:

- Decl.h (een file met alle declaraties)
- Program.s/Pcfpro.s (een file met het programma)
- CSHDIS.s (een file met alle display-routines)

In de file Decl.h worden alle al besproken naamtoewijzingen gedaan. In Program.s staat het feitelijke programma (hoofdprogramma) voor de Pocketstem met de MEA8000. De file Pcfpro.s bevat een analoog programma maar dan voor aansturing van de PCF8200. De file CSHDIS.s is dat gedeelte van het programma dat de aansturing van het display verzorgt en de gebruiksresultaten bijhoudt. Aangezien deze laatstgenoemde zaken slechts nodig zijn voor evaluatie-doeleinden, zijn ze in een aparte file ondergebracht die indien gewenst weggelaten kan worden.

6.3 Beschrijving van de gebruikte subroutines.

In dit gedeelte van het hoofdstuk zullen de subroutines van het hoofdprogramma wat nader beschreven worden. In figuur 6.4 staat de flowchart van het hoofdprogramma.

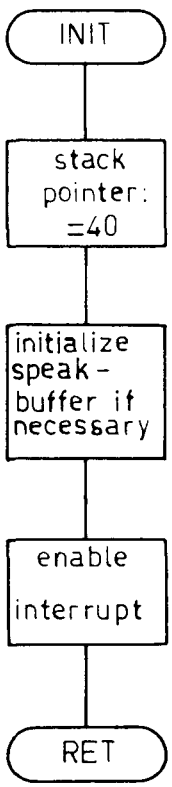


Figuur 6.4: Flowchart van het hoofdprogramma.

Deze flowchart vermeldt 8 subroutines, die hier zullen worden besproken. De genoemde subroutines zijn:

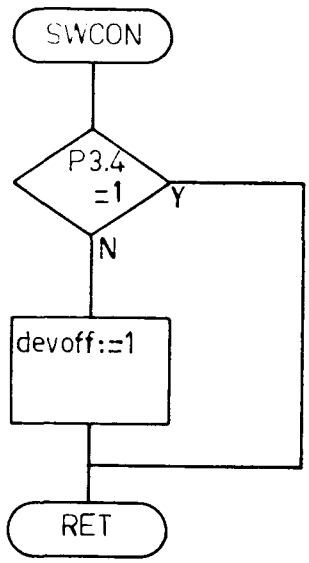
1. INIT (Initialisatie.)
2. SWCON (Controleert de stand van de aan/uit-schakelaar.)
3. OUTR (Schakelt de Pocketstem uit.)
4. POWCH (Controleert de voedingsspanning.)
5. PFAILR (Geeft aan dat de voedingsspanning te laag is.)
6. SCAN (Scant het key-board.)
7. STANDBY (Zet de Pocketstem stand-by.)
8. SPEAK (Zorgt voor het uitspreken van een of meer zinnen.)

ad 1. De routine INIT (zie figuur 6.5) zorgt voor de initialisatie van de Pocketstem. Deze routine zorgt voor het indelen van het RAM (zie Tabel 6.1). Ook wordt er gekeken of de speakbuffer-pointers nog op de juiste waarde staan. Als de Pocketstem namelijk voor het eerst van spanning wordt voorzien of een storing heeft gehad die het RAM heeft verminkt, dan moeten de pointers opnieuw geset worden. Bij het gebruik zal hiervan niet veel gemerkt worden, daar de Pocketstem gewoon zijn werk blijft doen en alleen als direkt na de storing een zin voor de tweede keer zou moeten worden uitgesproken, dan zal ditmaal niet de versie uit blok 2 genomen worden. Het gevolg voor het gebruik is dan wel niet zo groot, voor de registratie van het gebruik is zo'n storing natuurlijk funest.



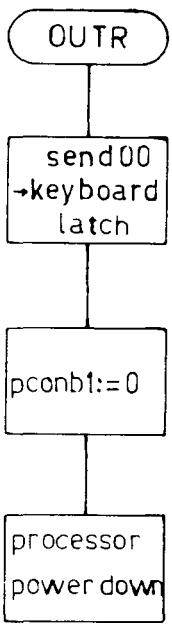
Figuur 6.5: Flowchart van de initialisatie-routine.

ad 2. De routine SWCON (zie figuur 6.6) controleert de stand van de schakelaar en geeft dat door aan het Device Status Word.



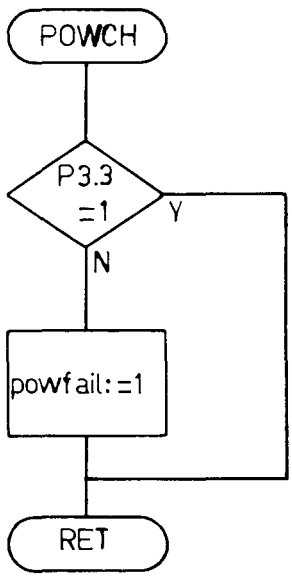
Figuur 6.6: Flowchart van de SWitch-CONtrol routine.

ad 3. De routine OUTR (zie figuur 6.7) maakt het keyboard inaktief door 00 naar de latch te schrijven, schakelt de voeding op stand-by en zet de processor in power-down mode (de processor wordt weer aktief d.m.v. een hardware-reset).



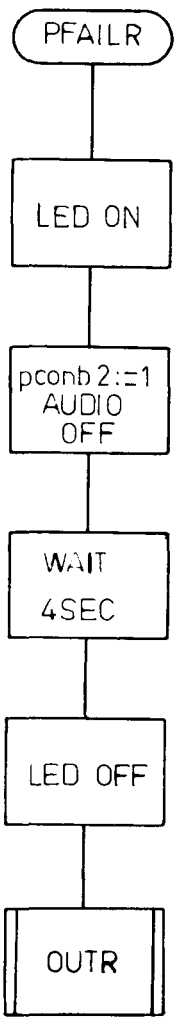
Figuur 6.7: Flowchart van de routine, die de Pocketstem uitschakelt.

ad 4. De routine POWCH (zie figuur 6.8) controleert de voedingsspanning door de, aan poort 3.3 gekoppelde, uitgang van de niveaudetector te testen.



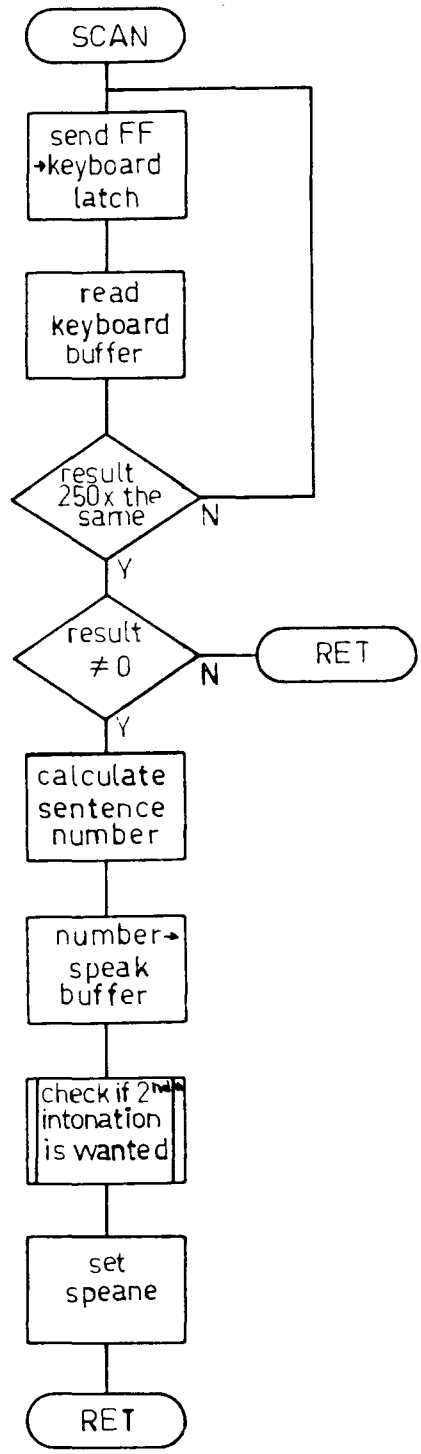
Figuur 6.8: Flowchart van de POWer CHeck routine.

ad 5. De routine PFAILR (zie figuur 6.9) treedt in werking als de voedingsspanning te laag is. Achtereenvolgens wordt een LED aangeschakeld, wordt het audiogedeelte (met evt. de MEA8000) uitgezet en wordt 4 sec. gewacht. Daarna wordt de LED weer uitgezet en wordt de routine OUTR aangeroepen. Deze routine wordt:
(1) In het hoofdprogramma bereikt (als powfail-bit in het DSW is geset).
(2) Via een interrupt als de Pocketstem spreekt.



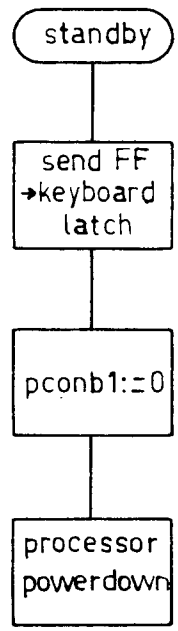
Figuur 6.9: Flowchart van de Power FAIL Routine.

ad 6. De routine SCAN (zie figuur 6.10) bepaalt of er een toets van het keyboard is ingedrukt en welke dat is. Allereerst wordt gecontroleerd of er een toets is ingedrukt door FF naar de latch te sturen. Als er een toets is ingedrukt dan vindt de processor bij het uitlezen van het buffer een waarde ongelijk aan 00. Om dender van de toetsen op te vangen wordt er gewacht tot er 250 maal hetzelfde resultaat uit het buffer wordt gelezen. 250 maal hetzelfde resultaat betekent dat (ingeval van de MEA8000, clockfrekwentie 4 MHz.) gedurende 9 msec. het resultaat gelijk moet blijven. Aangezien duur van de dender van een toets ongeveer 10 msec. is, wordt hiermee dender ondervangen (Voor de PCF8200, met een clockfrekwentie van 6 MHz. is de genoemde duur 12 msec). Als het resultaat stabiel is, dan kan worden bekeken welke toets is ingedrukt. Dit wordt gedaan door niet meer FF naar de latch te sturen, maar actereenvolgens 01, 02, 04, 08, 10, 20, 40 en 80. Op het moment dat in deze situatie het buffer een andere waarde dan 00 bevat is de ingedrukte toets gevonden. Uit de kombinatie van de inhoud van de latch en van het buffer is de toets een-eenduidig bepaald. Het nummer van deze toets wordt dan ook in deze routine berekend en in het speakbuffer geplaatst. Er wordt ook nog gekeken of de vorige zin in het speakbuffer hetzelfde is. Als de nummers gelijk zijn dan wordt in het DSW aangegeven, dat de zin uit BLOK2 genomen moet worden.



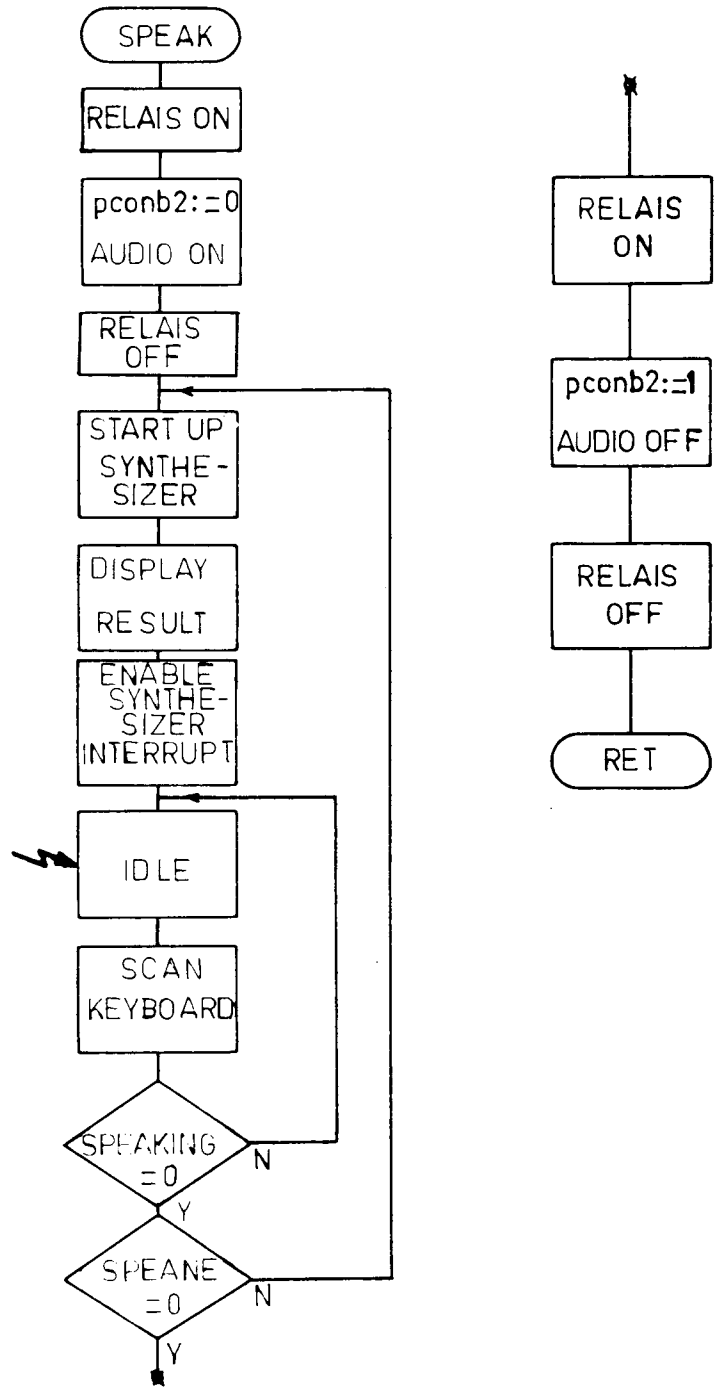
Figuur 6.10: Flowchart van de SCAN routine van het keyboard.

ad 7. De routine STANDBY (zie figuur 6.11) zet het keyboard stand-by door FF naar de latch te sturen. Vervolgens wordt de voeding stand-by en de processor power-down geschakeld. Merk op dat het verschil met de routine OUTR is, dat in dit geval een reset gegenereerd zal worden als een toets wordt ingedrukt.



Figuur 6.11: Flowchart van de routine STANDBY.

ad 8. De routine SPEAK (zie figuur 6.12) zorgt voor het uitspreken van een of meer zinnen. Op het moment dat het speakbuffer niet leeg is zal deze routine worden aangesproken. De routine begint met het aanzetten van het audiocircuit (en de MEA8000) en zorgt ervoor, door het bekrachtigen van het relais, dat er geen inschakelklik optreedt. Vervolgens wordt de interrupt die een power-failure aangeeft ge-enabled. Daarna wordt de synthesizer opgestart. De routines die daar voor nodig zijn, zullen niet meer worden besproken daar deze standaard zijn en terug te vinden zijn in [1] en [13]. De verschillen in aansturing tussen de MEA8000 en de PCF8200 wordt in hoofdstuk 6.4 nader besproken.



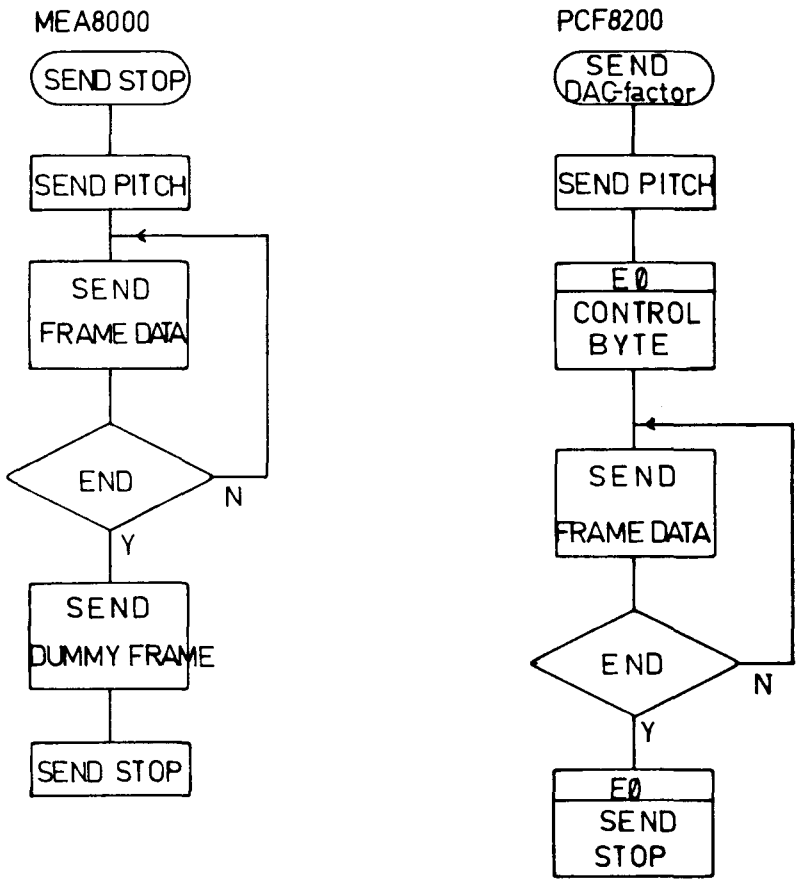
Figuur 6.12: Flowchart van de routine SPEAK.

Ten behoeve van de evaluatie bevat deze routine een subroutine, die een eventueel aangesloten display aanstuurt met informatie betreffende het aantal malen dat de boodschap is gebruikt. Vervolgens wordt de interruptlijn van de synthesizer ge-enabled en wordt de processor in idle-mode gezet. Een data-aanvraag van de synthesizer zal de processor uit de idle-mode halen. De processor handelt de data-aanvraag af in de interrupt service routine SEMEA c.q. SEPCF. Indien het eind van de zin is bereikt, dan geeft de interrupt service routine dit door aan de routine SPEAK door middel van resetten van het DSW-bit 'speaking'. Terug in de routine SPEAK wordt het keyboard gescand. Vervolgens wordt bekeken of de uitspraak al is afgelopen. Is dit niet het geval, dan wordt de processor weer in de idle-mode gezet en wacht deze weer op een data-aanvraag. Als de uitspraak wel is afgelopen dan wordt er gekeken of er nog meer moet worden uitgesproken (speane=1). Dit wordt hier gedaan en niet in het hoofdprogramma. De reden daarvoor is dat als er nog een zin moet worden uitgesproken, dat dan het audio-gedeelte niet eerst hoeft te worden uitgeschakeld om vervolgens weer te worden ingeschakeld. Als er niets meer hoeft te worden uitgesproken dan wordt het relais weer even bekrachtigd (teneinde een uitschakelklik te voorkomen) en het audio-gedeelte uitgeschakeld.

6.4 Verschillen in aansturing tussen de MEA8000 en de PCF8200.

Aangezien de PCF8200 een nauwkeurigere beschrijving van de formanten en de bijbehorende bandbreedtes nodig heeft (zie application notes [1] en [4]), is het dataformaat van een segment spraak anders dan dat van de MEA8000. Het voornaamste verschil voor de programmeur is het feit dat de PCF8200 5 bytes per frame (segment spraak) nodig heeft, in tegenstelling tot de MEA8000, die per frame 4 bytes data vraagt. Het feit dat de PCF8200 zowel een parameter tabel voor een mannenstem als een voor een vrouwenstem heeft houdt in dat de PCF8200 middels een command byte (zie figuur 6.13) duidelijk gemaakt moet worden, welke tabel moet worden genomen. Deze verschillen komen tot uitdrukking in het formaat van de spraakdata tabel (zie figuur 6.2). Zo is bij de data tabel voor de PCF8200 alles vijf byte georiënteerd (in tegenstelling tot de vier byte oriëntatie bij de MEA8000). Dit houdt ook in dat in plaats van één don't care byte voor het pitch start byte bij de MEA8000, de data tabel voor de PCF8200 behalve het don't care-byte ook een command-byte bevat.
Tevens bezit de PCF8200 een instelbare Digitaal Analoog Converter, het-

geen inhoud dat de instelling (de z.g.n. DAC-factor) van deze DAC ook verzonden moet worden. Een opvallend verschil in aansturing tussen beide synthesizers is het feit dat de PCF8200 geen hardware adresseerbaar commandregister heeft. Dit houdt in dat er middels een protocol in het commandregister moet worden geschreven. Tot slot nog een verschil in het lezen van de status van beide synthesizers. Bij de MEA8000 werd de status hardwarematig uitgelezen, via de $\overline{reqn}$-lijn. Bij de PCF8200 wordt de status ook door de *busy*-lijn bepaald. Daar er op de processor niet genoeg aansluitmogelijkheden over zijn om op eenvoudige wijze deze lijn te testen, is gekozen voor het softwarematig uitlezen van de status van de synthesizer.



Figuur 6.13: Flowcharts van de aansturing van de synthesizers.

7 Zinnenbestand voor de Pocketstem.

In dit hoofdstuk wordt beschreven, waarom een zinnenbestand van gesynthetiseerde zinnen is opgebouwd op floppy discs van een Apple IIe personal computer en aan welke eisen het programma-pakket op de Apple IIe moet voldoen.
Daarna wordt uiteengezet, hoe de omzetting van opgenomen zinnen tot synthetische spraak tot stand komt op de VAX 11/780 door middel van programma's uit het softwarepakket LVS [11].
Tenslotte wordt even stilgestaan bij de globale werking van het programmapakket op de Apple IIe, waarmee je het zinnenbestand kunt beheren en waarmee je EPROM's voor de Pocketstem kunt programmeren.
Uitgebreide informatie betreffende de bovenstaande Apple-programmatuur en het gebruik ervan is te vinden in de rapporten van Legdeur [6] en Tan [10].

7.1 Motivatie voor de opbouw van een zinnenbestand op een personal computer.

Bij de CSH I is het mogelijk om uit een vaste set zinnen te selecteren. De uit te spreken boodschappen zijn opgeslagen in een EPROM en ze bestaan uit gesynthetiseerde spraakdata. Uit de evaluatie van de CSH I is gebleken, dat er behoefte bestond aan een grotere en meer persoonlijke zinnenkeuze (Zie hoofdstuk 3).

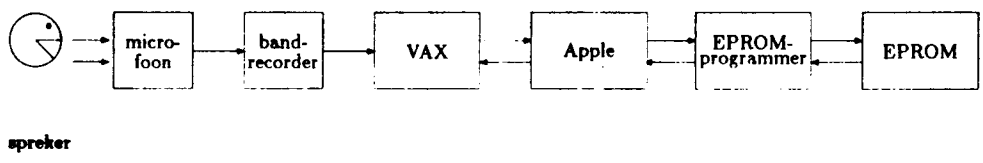
Bij de Pocketstem worden de zinnen ook geselecteerd uit een vaste zinnenset, die eveneens is opgeslagen in een EPROM. Bij de Pocketstem kan de gebruiker echter zijn eigen set kiezen uit een groot aanbod. Dit aanbod is opgeslagen in het zinnenbestand. Aan het opzetten en het gebruiken van het zinnenbestand zijn 3 aspecten verbonden:

1. De spraakbewerking vereenvoudigen.
   Om te komen tot het nivo, waarop de spraak in een bruikbare vorm voor de spraaksynthesizer is opgeslagen in een EPROM, moeten de volgende stadia worden doorlopen:

   - Het opnemen van gesproken boodschappen met een taperecorder.
   - Het synthetiseren van de opgenomen boodschappen op de VAX 11/780 computer d.m.v. programma's uit het LVS-softwarepakket [11].

- De boodschappen in een EPROM programmeren via een EPROM-programmer.

Om bij het aanbrengen van een andere zinnenset niet telkens opnieuw de hele weg te hoeven doorlopen (zie figuur 7.1), is een zinnenbestand van reeds gesynthetiseerde zinnen op floppy discs van een Apple IIe personal computer opgebouwd.



Figuur 7.1: Omzetting van spraak naar synthesizerdata.

Doordat de zinnen al in gesynthetiseerde vorm op het Apple-systeem aanwezig zijn, hoef je nu nog slechts de zinnen vanuit het Apple-bestand te kiezen en in de EPROM te programmeren. M.b.v. de programmatuur op de Apple IIe en de op Apple IIe aangesloten EPROM-programmer wordt de keuze van de 28 verschillende zinnen (samen met het besturingsprogramma van de Pocketstem) in een EPROM geprogrammeerd.
Het op de band opnemen van de zinnen en de verwerking van die zinnen op de VAX blijft op dezelfde manier gebeuren. Echter, door de programmatuur op de Apple IIe is het programmeren van de EPROM veel minder omslachtig geworden.

2. Het programma-pakket op de Apple IIe moet aan enkele eisen voldoen:
   - Het programma moet eenvoudig te bedienen zijn. Dit kan worden gerealiseerd door menusturing.
   - Via het "Apple"-programma moet het databestand opgebouwd en uitgebreid kunnen worden. Voorts moet een gewenste zinnenset gekozen kunnen worden uit het bestaande zinnenbestand.
   - Tenslotte moet vanuit dit programma de aangesloten EPROM-programmer kunnen worden aangestuurd.
3. Het, op een personal computer aanwezige, zinnenbestand met bijbehorende software werkt onafhankelijk van de grote VAX-computer,

waardoor je niet meer geconfronteerd wordt met de problemen van een multi-user systeem (b.v. lange responstijden en bezette terminals bij grote drukte). Een ander voordeel van dit nieuwe systeem is de transporteerbaarheid, zodat nu eventueel bij de gebruiker thuis uit de aanwezige database een persoonlijke zinnenset kan worden samengesteld en ter plaatse in de EPROM kan worden geprogrammeerd.
Op de printplaat var de Pocketstem is een IC-voet voor een EPROM gesoldeerd en de EPROM met de nieuwe zinnen kan daar gemakkelijk in worden geplaatst.

De reeds gesynthetiseerde zinnen van het Apple-bestand zijn gekozen in overleg met therapeuten en gebruikers (Zie [8]). Ze bevatten allerlei onderwerpen, zoals die in het dagelijkse leven van de gebruikers voorkomen. Aan deze database van zinnen kunnen steeds nieuwe zinnen worden toegevoegd, die dan later weer beschikbaar blijven voor toekomstige gebruikers.

7.2 Synthese van zinnen op de VAX.

Bij het synthetiseren van de, op de band opgenomen, zinnen op de VAX-computer wordt gebruik gemaakt van programma's uit het softwarepakket LVS [11].

Meteen na het inloggen begin je met het reserveren van een spraakkanaal via het programma ASSLPA. Daarna volgt de Analoog/Digitaal-conversie van een of meerdere zinnen via het programma INP. Hierin wordt de spraak bemonsterd (gesampled) en in een N-file (N:dummy) gezet. Een N-file is een disk file met bemonsterde spraakdata. Er wordt bovendien gecontroleerd of het opnamenivo van de A/D-converter niet te hoog of te laag is en wel als volgt:
Zinnen met dezelfde betekenis zijn meerdere keren en op verschillende manieren op de band ingesproken. Van deze zinnen wordt verondersteld, dat ze allemaal ongeveer hetzelfde geluidsvolume hebben (vandaar dat ze als groep kunnen worden ingenomen). Voor de verschillende groepen zinnen wordt telkens, door het veranderen van het output nivo van de bandrecorder en dan INP te herhalen, de optimale signaal-(kwantiserings)ruis verhouding verkregen.
De diverse zinnen van de zojuist gemaakte N-file kunnen, via het segmenteerprogramma SGF, worden afgebakend. Ook kun je in het programma SGF de resulterende zin laten uitspreken en als die voldoet kan deze worden gecopieerd naar een andere N-file (bv zin1a), waarvan je de naam (in tegen-

stelling tot *N:dummy*) wél zelf kunt bepalen. Op deze tijdrovende manier moeten alle boodschappen een voor een worden omgezet in N-files.
De files krijgen de opeenvolgende namen zin1a (zin1, versie a), zin1b (zin1, versie b), enz., zin2a, zin2b, zin2c enz. Hierna worden de N-files omgezet in A/P-files. In een A/P-file staat de spraak in, volgens het bron-filter model, geparametriseerde vorm opgeslagen. De conversie naar A/P-files gebeurt achtereenvolgens door de programma's AAB en JVH.
AAB verzorgt een standaard bron/filter analyse van N-files naar A/P-files. De toonhoogte (F0) krijgt hier een vaste waarde. Het programma JVH verzorgt een standaard toonhoogte analyse op de N-file en de "read-A/P"-file. Hierdoor wordt een variabele F0 verkregen. Deze wordt weggeschreven in de "write-A/P"-file. De spraakuiting is hierdoor weer voorzien van zijn oorspronkelijke toonhoogte en niet meer monotoon.
De volgende stap is de conversie van A/P-files in spraakchip-files, d.w.z. de output krijgt nu een zodanige structuur, dat deze als input gebruikt kan worden voor de spraakchips MEA8000 en PCF8200.
Aangezien de parameter-structuur voor de MEA8000 en de PCF8200 verschillend is hebben we hiervoor verschillende programma's nodig.
Voor de MEA8000 gebeurt dit via het programma KWC. Om de file tenslotte naar de Apple IIe te kunnen overzenden moet deze via CNA geconverteerd worden in een ASCII-file.
Voor de PCF8200 gebeurt zowel de conversie in spraakchip-files als de conversie in ASCII-files via het programma PCF.
Om het bedieningsgemak te vergroten en de kans op foutieve handelingen te verminderen, worden (voor de communicatie met de Apple IIe) alle ASCII-files in een aparte subdirectory ondergebracht. De gehele directory kan nu, via het communicatie-programma KERMIT, in een enkel commando worden overgezonden naar de Apple IIe, zonder dat de afzonderlijke zinnamen telkens opnieuw ingetypt moeten worden. (KERMIT is een programma, dat deel uitmaakt van de programma's op de Apple IIe en op de VAX.)
Je hoeft slechts de ASCII-files, die in deze subdirectory staan, en de N-files, die via INP en SGF zijn gemaakt, te bewaren. De A/P-files mogen worden weggehaald om geheugenruimte in de VAX te besparen.

De conversie N-files → ASCII-files kan plaatsvinden via een commando-file. Hierin kunnen de opeenvolgende bewerkingen voor één zin worden opgenomen.
Het eerste gedeelte, de conversie N-files → A/P-files, verloopt voor de beide spraakchips hetzelfde.

Deze conversie vindt plaats via de commando-file 'make.com' en deze bevat:

```
$ aab n:'p1' 'p1'
$ jvh n:'p1' 'p1' 'p1'
$ exit
```

De A/P-file → chip-N-file conversie voor een zin via de MEA8000 vindt plaats in de commando-file 'maakmea.com' en deze bevat:

```
$ kwc 'p1' n:'p2'
$ cna n:'p2' submea:'p2'
$ delete/noconfirm n:'p2'..
$ exit
```

Voor een zin via de PCF8200 verloopt die conversie via de commando-file 'maakpcf.com' en deze ziet er alsvolgt uit:

```
$ pcf 'p1'
$ copy 'p1'.pcf subpcf:*.*.
$ delete/noconfirm 'p1'.pcf
$ exit
```

De commando-files van de afzonderlijke zinnen kunnen worden samengevoegd tot een commando-file voor meerdere zinnen. Als voorbeeldje zou daarvoor de volgende commando-file 'auto.com' kunnen worden gebruikt:

```
$ @make zin1a
$ @make zin1b
$ @maakmea zin1a z1a
$ @maakmea zin1b z1b
$ @maakpcf zin1a
$ @maakpcf zin1b
```

In 'auto.com' worden de zinnen zin1a en zin1b omgezet in ASCII-files, geschikt voor resp. de MEA8000 en de PCF8200, en daarna in aparte sub-directories geplaatst.
Als de commando-file veel zinnen bevat kost dit veel computer-rekentijd, daarom kan dan deze laatste commando-file het beste via batch op de VAX worden verwerkt (SUBMIT/NOTIFY/NOPRINT <command file>).
(**Opm.:** Als je in de commando-files van de LVS programma's zowel alle inputfiles als alle outputfiles meegeeft, dan hoef je in die programma's niets meer interactief te specificeren en gaat alles automatisch volgens de default-waarden.)

7.3 Het "Apple-systeem".

Zoals reeds in de inleiding is opgemerkt, wordt de Apple-programmatuur uitgebreid behandeld in de rapporten van Legdeur [6] en Tan [10].
Nadat de op de band opgenomen zinnen op de VAX zijn verwerkt en onder de juiste namen in de vorm van ASCII-files in daarvoor bedoelde subdirectories zijn weggeschreven, moeten deze files worden overgehaald naar de Apple IIe personal-computer en vanuit die Apple IIe moet de EPROM geprogrammeerd kunnen worden.
Hiervoor is op de Apple IIe een programma-pakket ontwikkeld.

Globaal gezien bestaat dit Apple-pakket uit de volgende onderdelen :

- Het file transport van VAX naar Apple IIe.
- Het converteren van de ASCII-files naar files in de geschikte datastructuur.
- Het opbouwen van een algemeen zinnenbestand. (Van zowel de spraakdata van de zin, als de tekstdata.)
- Het selecteren van zinnen uit dat bestand om een verzameling van zinnen op te bouwen voor een bepaalde Pocketstem.
- Het in een EPROM programmeren van het Pocketstem-programma, de uitgezochte zinnenset (bij gebruik van de MEA8000 2 versies) en de bijbehorende tekstdata.

De programma's voor het file-transport van VAX → Apple IIe en het opbouwen van een algemeen zinnenbestand staan op een systeemdisc genaamd "EDIT-DATA-DISC".
Het filetransport wordt verzorgd door een communicatie-programma (KERMIT), dat de overgehaalde ASCII-files op de systeemdisc plaatst onder dezelfde naam, als waaronder ze in de VAX-directory stonden vermeld.
Voor verdere verwerking van deze ASCII-files moet het programma opnieuw worden opgestart, omdat KERMIT gedeeltelijk in hetzelfde deel van het APPLE-geheugen werkzaam is als het Disc Operating System (DOS) en daardoor DOS aantast.
Na opnieuw te zijn opgestart kun je met een andere optie uit het menu de ASCII-files verder verwerken. Je geeft dan een bepaalde filenaam en, als die file aanwezig is, wordt die ASCII-file in een binaire file omgezet in de

geschikte chip-datastructuur. Daarnaast moet je ook nog de tekst van die zin intypen.
Tenslotte worden de tekst en de spraakdata gezamenlijk op een datadisc (in disc-drive II) geplaatst onder de naam "zin<nr>". Op één datadisc mogen 100 zinnen op deze manier worden opgeslagen.
Naast dit toevoegen van nieuwe zinnen aan het zinnenbestand, bestaan er ook mogelijkheden om tekst en/of spraakdata van reeds bestaande zinnen te veranderen. Ook kun je zinnen verwijderen en verplaatsen in het zinnenbestand via deze programmatuur.
Uit dit algemene zinnenbestand kan door de individuele gebruiker een keuze worden gemaakt, welke zinnen in zijn/haar persoonlijke Pocketstem geplaatst moeten worden.

Op systeemdiscs, genaamd "APPARAAT<nr>", staan de programma's voor het selecteren van de gewenste zinnen uit het algemene zinnenbestand en voor het programmeren van de EPROM. Voor iedere Pocketstem is een aparte systeemdisc genomen.
Na het opstarten van zo'n systeemdisc krijg je het opstartmenu voorgeschoteld, waarin je o.a. de mogelijkheid hebt om een zinnenset voor de te programmeren EPROM samen te stellen. Van de gekozen zinnen worden zowel de tekstdata als de spraakdata in de zinnenset opgenomen. Je kunt de volgorde van de diverse zinnen zodanig bepalen, dat ze onder een door jezelf te bepalen toets op de Pocketstem terechtkomen.
Via een andere optie uit het menu kun je daarna de gekozen zinnen (spraakdata en teksten) in de EPROM programmeren. Ook kan het besturingsprogramma van de Pocketstem nog in de EPROM worden geprogrammeerd, indien dit nog niet gebeurd is.
Dit besturingsprogramma is reeds op de systeemdisc opgeslagen, nadat het op een PMDS (= Philips Microcomputer Development System) is ontwikkeld. Vanuit die PMDS werd het in een EPROM geprogrammeerd en vanuit dat EPROM kon het programma door het Apple-systeem op de systeemdisc worden gecopieerd.

8 Evaluatie resultaten.

In [8] worden de resultaten van de evaluatie besproken voor wat betreft de acceptatie van synthetische spraak in een communicatiehulpmiddel en de gebruikswaarde van zo'n hulpmiddel. In dit hoofdstuk worden de technische aspecten van de evaluatie besproken.

Gedurende de evaluatieperiode zijn een aantal gebreken naar voren gekomen, die als volgt onder te verdelen zijn:

1. Fouten in het ontwerp van de schakeling.
2. Fouten in de constructie c.q. in de assemblage.
3. Fouten door gebruik c.q. slijtage.

ad 1. Het reset-circuit was storingsgevoelig, hetgeen tot gevolg had dat er af en toe (ongeveer eens per 50 maal spreken) een te korte reset-puls werd gegenereerd. Deze te korte reset-puls verstoorde de inhoud van het RAM. Voor het gebruik was dat nauwelijks merkbaar. Alleen als de te korte reset-puls optrad op het moment dat een zin voor de tweede maal werd uitgesproken, dan werd niet de alternatieve versie hoorbaar gemaakt, maar werd de eerste herhaald. Deze storing in het reset-circuit mag dan voor het gebruik niet hinderlijk zijn, voor het onthouden van de gebruiksresultaten is deze storing funest (het RAM verliest zijn inhoud). Door in het reset-circuit een one-shot op te nemen (zie schema bijlage 2), wordt altijd een minimale lengte van de reset-puls gegarandeerd.

ad 2. Een duidelijke constructiefout is de inbouw van de schakelaar en het LED-je. Beide steken iets buiten de zijkant van het kastje uit (maar dit duurt meestal niet erg lang). Duidelijk is dat deze twee onderdelen verzonken moeten worden aangebracht zodat de schakelaar en/of het LED-je niet meer beschadigd kunnen worden.

Een drietal foutjes werd veroorzaakt door assemblagefouten. Tweemaal weigerde een Pocketstem na korte tijd dienst, doordat een soldeerverbinding geen goed kontakt maakte. De slechte verbindingen ontstonden als gevolg van het feit dat de print niet is doorgemetaliseerd (zie hoofdstuk 4). Eenmaal was bij het inzetten van het EPROM een pootje verbogen, hetgeen na een bepaalde tijd tot een storing heeft geleid. Ook is een keyboard na een bepaalde tijd een storing gaan vertonen, omdat bij de montage een klein

stickertje onder het keyboard is blijven zitten. Deze oneffenheid leidde na enkele maanden tot een storing.

ad 3. Het grootste aandeel in het soms niet functioneren van een Pocketstem ligt aan overmatige gevoeligheid m.b.t. gebruik c.q. slijtage van enkele onderdelen.

Zo lieten in het begin de rubber voetjes regelmatig los, hetgeen werd opgelost door een andere lijmsoort te gebruiken voor de bevestiging. Ook de interne bedrading verdient enige aandacht. In de gebruikte exemplaren zijn enkele onderdelen met 'niet-flexibele' draad verbonden (speaker/print, schakelaar/print en boven-/onderprint). Dit is geen bezwaar indien de Pocketstem niet wordt opengemaakt. Het feit dat de gebruikte exemplaren regelmatig van een andere zinnenset werden voorzien, noodzaakte echter wel het regelmatig openmaken van de behuizing. Hierdoor werden draadverbindingen op den duur zwak en was de kans op storing aanwezig.

Tenslotte kan nog opgemerkt worden dat de slijtage-gevoeligheid van het keyboard het gevolg was van een onbeschermd gedeelte van de staart van het keyboard. Dit heeft driemaal tot het vervangen van het keyboard genoodzaakt. Overigens is deze constructiefout door de leverancier van het keyboard onderkend en is bij de huidige exemplaren niet meer aanwezig.

9 Conclusies.

In dit hoofdstuk zullen de conclusies vermeld worden, die getrokken kunnen worden met betrekking tot de in voorgaande hoofdstukken behandelde onderwerpen. Tevens zullen enige algemene conclusies worden besproken.

Ten aanzien van de realisatie van de ontwerpeisen kan gesteld worden dat deze voor een groot deel gerealiseerd zijn. Drie eisen zijn niet gerealiseerd, te weten:
-Er is geen afdekplaat bijgeleverd. De reden daarvoor is, dat het membraan keyboard al gedeeltelijk in deze eis tegemoet komt en dat de evaluatie wel duidelijk zal maken in welke gevallen een afdekplaat toch nog nodig is. Het maken van zo'n afdekplaat werd in voorkomende gevallen overgelaten aan de instrumentatiedienst van het betreffende instituut.
-Ook het bevestigen van de Pocketstem op de rolstoel werd aan bovengenoemde dienst overgelaten, aangezien de gewenste bevestigingen per gebruiker erg verschilden.
-De keuze tussen een mannen- of vrouwenstem kon niet worden gerealiseerd, daar de daarvoor benodigde spraakchip (de PCF8200) nog niet werd geproduceerd.

Van de voor de Pocketstem ontworpen symbolen kan gezegd worden dat deze goed voldoen. Er is echter een probleem met betrekking tot het up to date houden van de symbolenset. Telkens als de zinnenset uitgebreid wordt, dan moet ook de symbolenset worden aangepast. Als eerste zal voor de desbetreffende zin een symbool moeten worden ontworpen en ten tweede moet van het symbool een slijtvaste sticker gemaakt worden. Het laatstgenoemde probleem is echter opgelost, aangezien de nieuwe keyboards met een inlegvel zijn uitgerust. Dit houdt in dat de stickers niet meer van slijtvaste kunststof hoeven te worden gemaakt. Het symbool kan gewoon op papier getekend, of gecopieerd worden en vervolgens op het inlegvel geplakt worden.

Het hardware ontwerp is wat betreft het functioneren van de Pocketstem behoorlijk geslaagd. De ontwerpeisen zijn allemaal gerealiseerd. Slechts één storing in de werking is terug te voeren tot een ontwerpfout. Het genereren van de reset-puls bleek niet betrouwbaar genoeg te zijn (zie hoofdstuk 8). Het euvel is inmiddels hersteld.

De software van de Pocketstem voldoet goed, behalve met betrekking tot

de gevoeligheid van het keyboard. De software is in staat toetsdender op te vangen, maar is in vele gevallen niet in staat de wat minder vloeiende toetsaanslagen te verwerken. Vele gebruikers van de Pocketstem drukken soms ongewild een toets meerdere malen kort na elkaar in. In één geval was dit zó hinderlijk, dat gekozen werd voor het negeren van het keyboard zolang de Pocketstem nog aan het spreken is. Als oplossing voor het vermelde probleem zou gedacht kunnen worden aan het, een korte tijd, niet scannen van het keyboard als een toets is ingedrukt. Deze tijd zou zelfs door de gebruiker moeten kunnen worden ingesteld, zodat snelheid en foutloze bediening tegen elkaar kunnen worden afgewogen.

Het zinnenbestand, op de Apple IIe, met bijbehorende beheer-software functioneert uitstekend. Het genereren van spraak-data blijft een wezenlijk probleem (zie hoofdstuk 7). Indien echter de spraakdata voorhanden is, dan is deze eenvoudig in het bestand onder te brengen. Als de spraakdata eenmaal is opgenomen in het zinnenbestand, dan is het genereren van een zinnenset (inclusief het programmeren van een EPROM) een eenvoudige zaak die slechts 20 minuten in beslag neemt.

Voor wat betreft het technisch functioneren van de Pocketstem kan gesteld worden dat er weinig echte problemen zijn opgetreden. De slijtagegevoeligheid van de staart van het keyboard (zie hoofdstuk 8) is geen probleem meer en de andere geconstateerde schoonheidsfoutjes zijn eenvoudig verholpen.

Als algemene conclusie kan gesteld worden dat de Pocketstem als hulpmiddel voor spraakgehandicapten bij verschillende gebruikersgroepen goed blijkt te voldoen. Echter het is ook duidelijk geworden, dat voor sommige gebruikersgroepen nog wensen overblijven. Een veel gehoorde wens is een uitgebreidere zinnenset in de Pocketstem, hetgeen echter direct bedieningsproblemen met zich meebrengt. Meer zinnen houden meer toetsen in (dus een groter apparaat) of een meer complexe bediening. Duidelijk is dat aan dit probleem nog de nodige aandacht zal moeten worden besteed. De wens om een vrouwenstem (of kinderstem) in de Pocketstem te hebben zal in de direkte toekomst worden vervuld.

Tot slot kan nog vermeld worden dat de volgende facetten van de Pocketstem in de directe toekomst de nodige aandacht zal krijgen. Ten aanzien van de beschikbare zinnenset wordt een aantal zaken onderzocht c.q. ontwikkeld:

- De Pocketstem zal naast de zinnenset ook met een spelalfabet worden uitgerust.
- De Pocketstem wordt aangepast, zodat deze aan de Tiepstem [3] kan worden gekoppeld. Door deze koppeling wordt het mogelijk de Pocketstem , via de Tiepstem, van nieuwe zinnen te voorzien. Een nadeel hieraan is wel, dat de Pocketstem dan voorzien wordt van zogenaamde difoonspraak, waarvan de kwaliteit nog niet optimaal is.
- De Pocketstem wordt zodanig aangepast dat de zinselectie kan geschieden op basis van een symbolentaal (b.v. Bliss of Dominolan).

Literatuurlijst

[1] Bierlaagh, Th.C.J. (1982)
MEA8000 Application Report.
Philips application report EDP8201, Elcoma-division.

[2] Bloemendaal, van, H. (1986)
Evaluatie van de symboliek voor de Compacte Spraak Hulp II, uitgevoerd bij afaciti.
IPO Rapport no. 567.

[3] Deliege, R.J.H. (1986)
Technische beschrijving van de Tiepstem.
IPO Rapport no. 548.

[4] Have, ten, M. (1985)
System specification voice synthesizer PCF8200 (M4790)
Philips laboratory report DPE85105.

[5] Intel Corporation (1985)
Micro-controller Handbook.
User's manual.

[6] Legdeur, G.J.A.A. (1987)
Aanpassing en documentatie voor programma's t.b.v. de Pocketstem.
Stageverslag Technische Universiteit Eindhoven, vakgroep EME.

[7] National Semiconductor (1980)
Linear databook.

[8] Oostinjen, E., Balkom, van, H., en Soede, M. (1985)
Evaluatie "Compacte Spraakhulp" IPO. Ervaringen met het eerste proefmodel.
Intern rapport IRV.

[9] Speth-Lemmens, I. (1987)
Praktijk-onderzoek Pocketstem. Een evaluatie-onderzoek naar de bruikbaarheid van de Pocketstem.
Intern rapport IRV/8 doc.(87)

[10] Tan, T.S.G. (1987)
Ondersteuningsprogrammatuur voor de Pocketstem.
Stageverslag Technische Universiteit Eindhoven, vakgroep EME.

[11] Vogten, L.L.M. (1985)
LVS - Speech processing programs on IPO-VAX 11/780.
IPO handleiding no. 67.

[12] Vroemen, M. (1986)
Symboliek voor de Compacte Spraakhulp II.
IPO Rapport no. 542.

[13] Waterham, R.P. (1986)
Technische beschrijving van de Compacte Spraak Hulp (CSH I).
IPO Rapport no. 525.

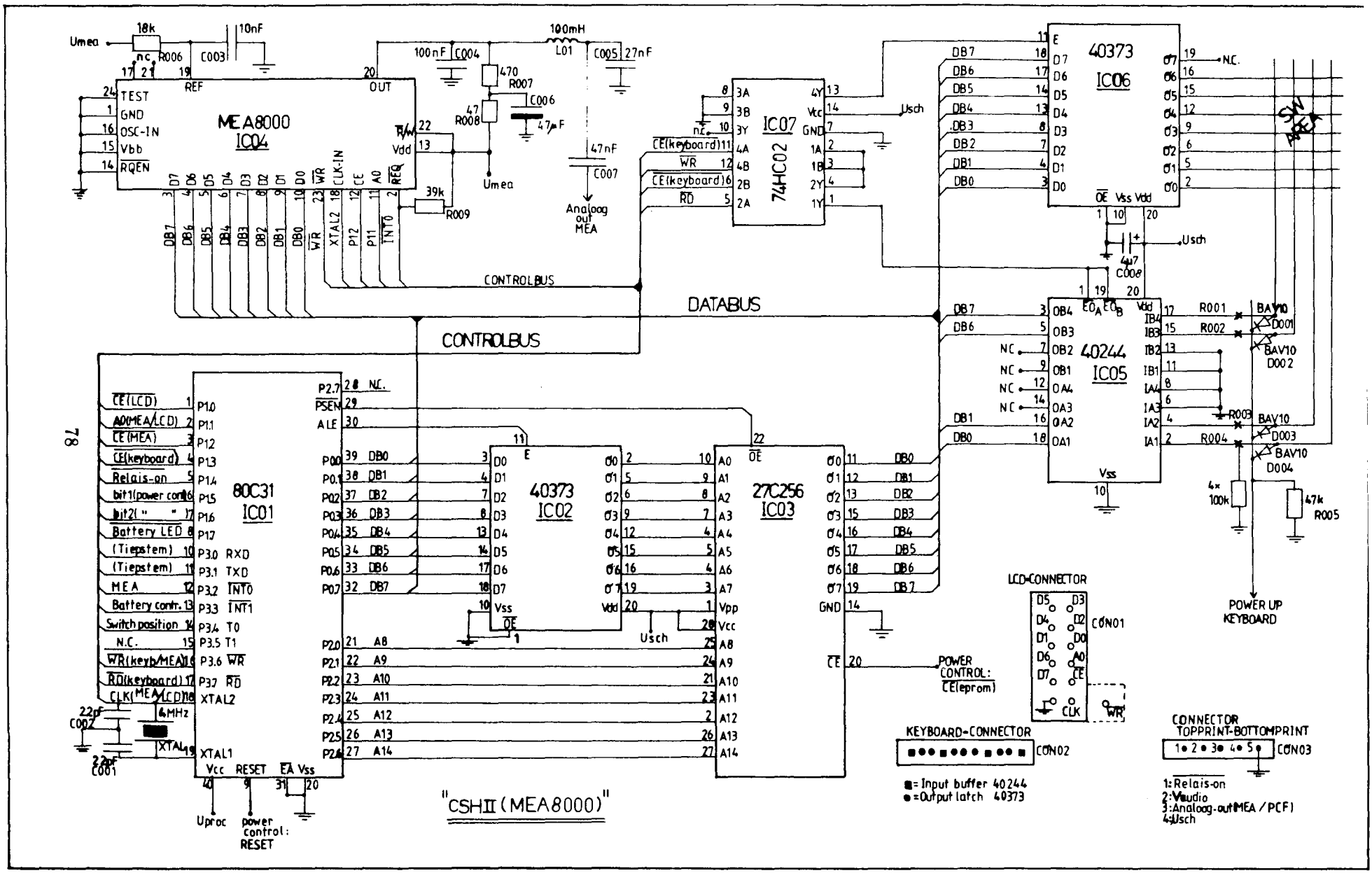
[14] Waterham, R.P. (1985)
Voorstellen voor de ontwerpeisen van de Compacte Spraak Hulp (CSH II), een eenvoudig hulpmiddel voor vocaal gehandicapten.
IPO Memorandum no. 296.

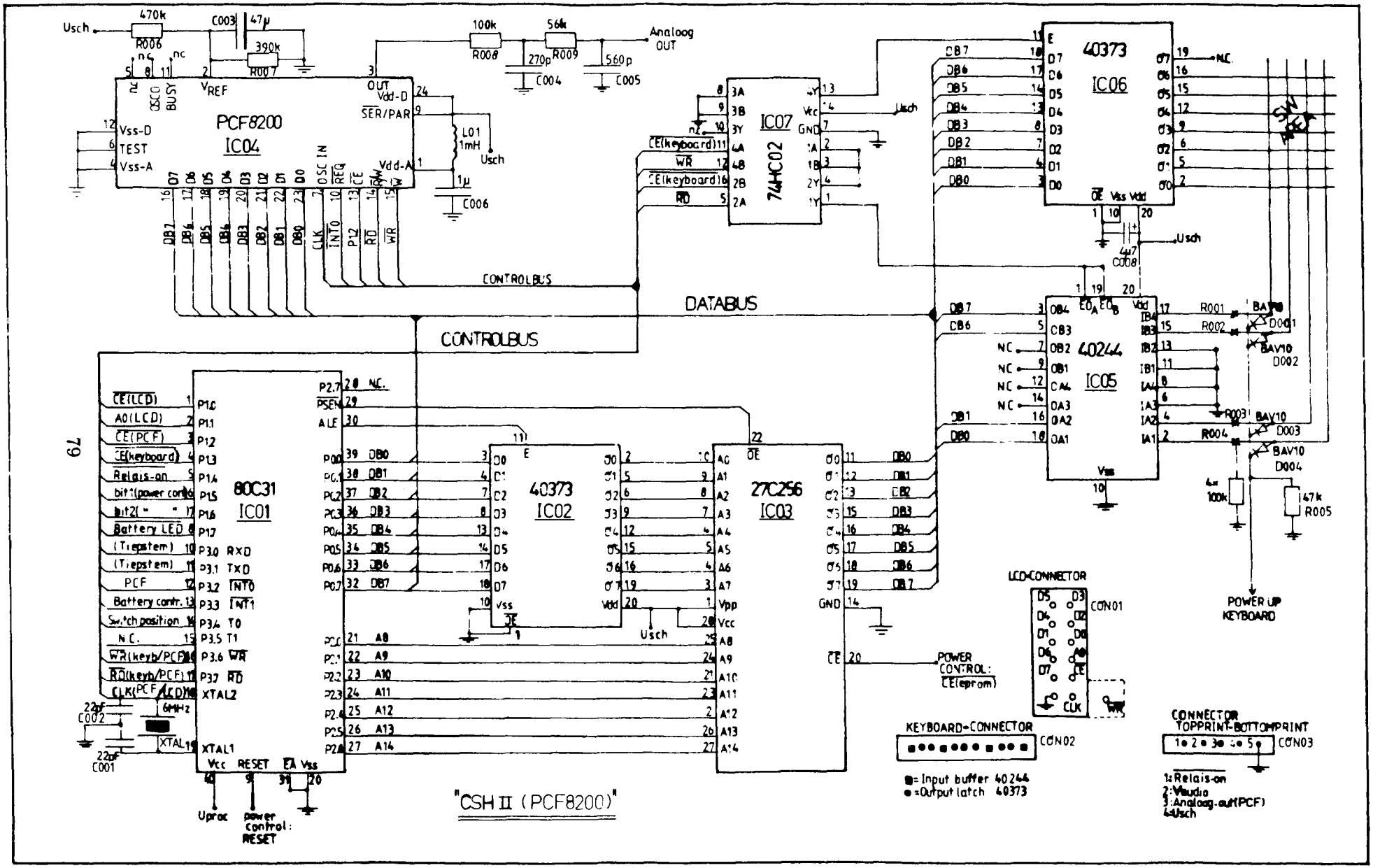
[15] Waterham, R.P. en Ossevoort, H.J.M. (1985)
Overwegingen t.a.v. de keuze van het keyboard van de CSH II.
IPO Memorandum no. 297.

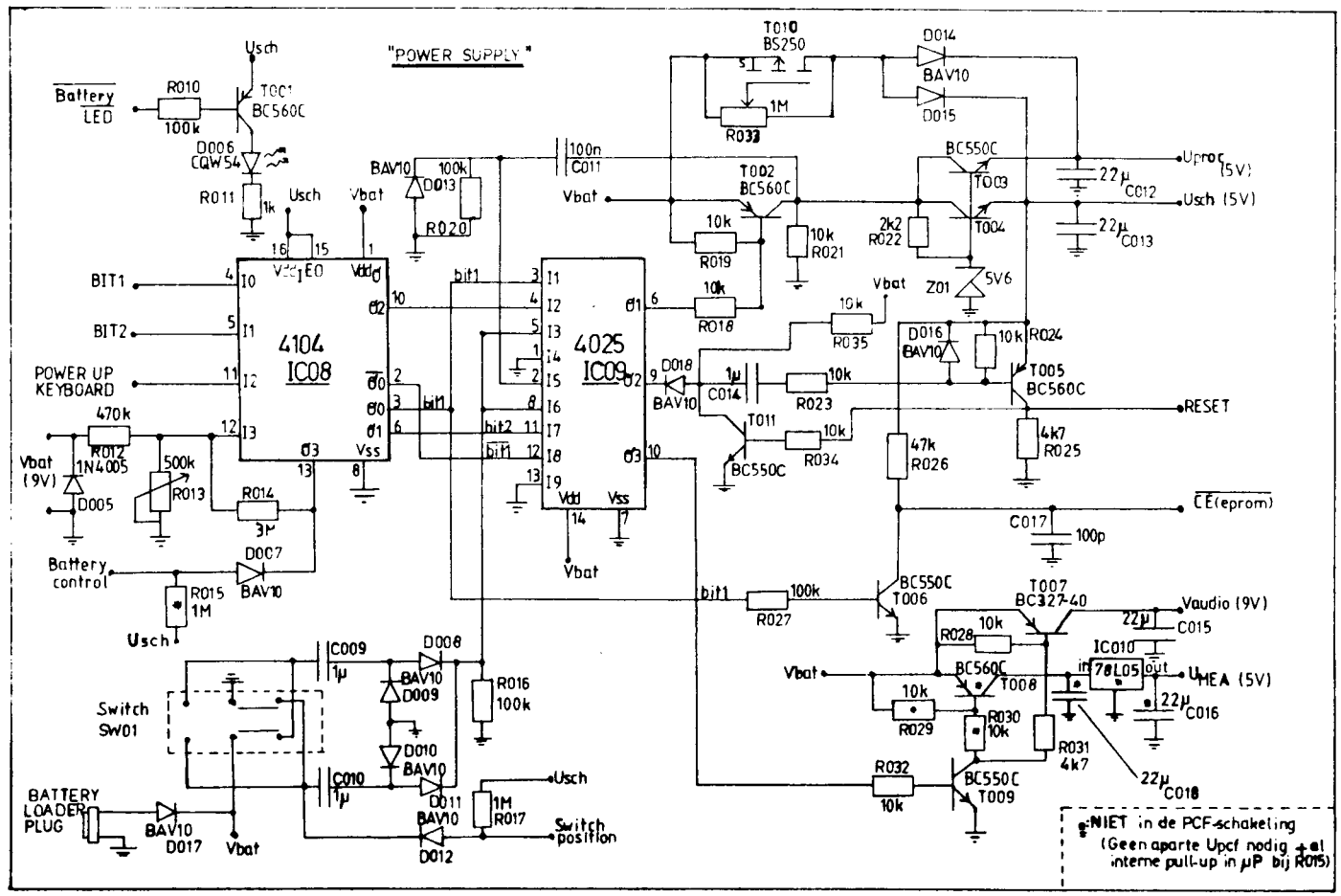
[16] Wintek sMartwork. (1986)
Reference Manual Wintek sMartwork.
Technical Bulletin (Revision v1.2r0).

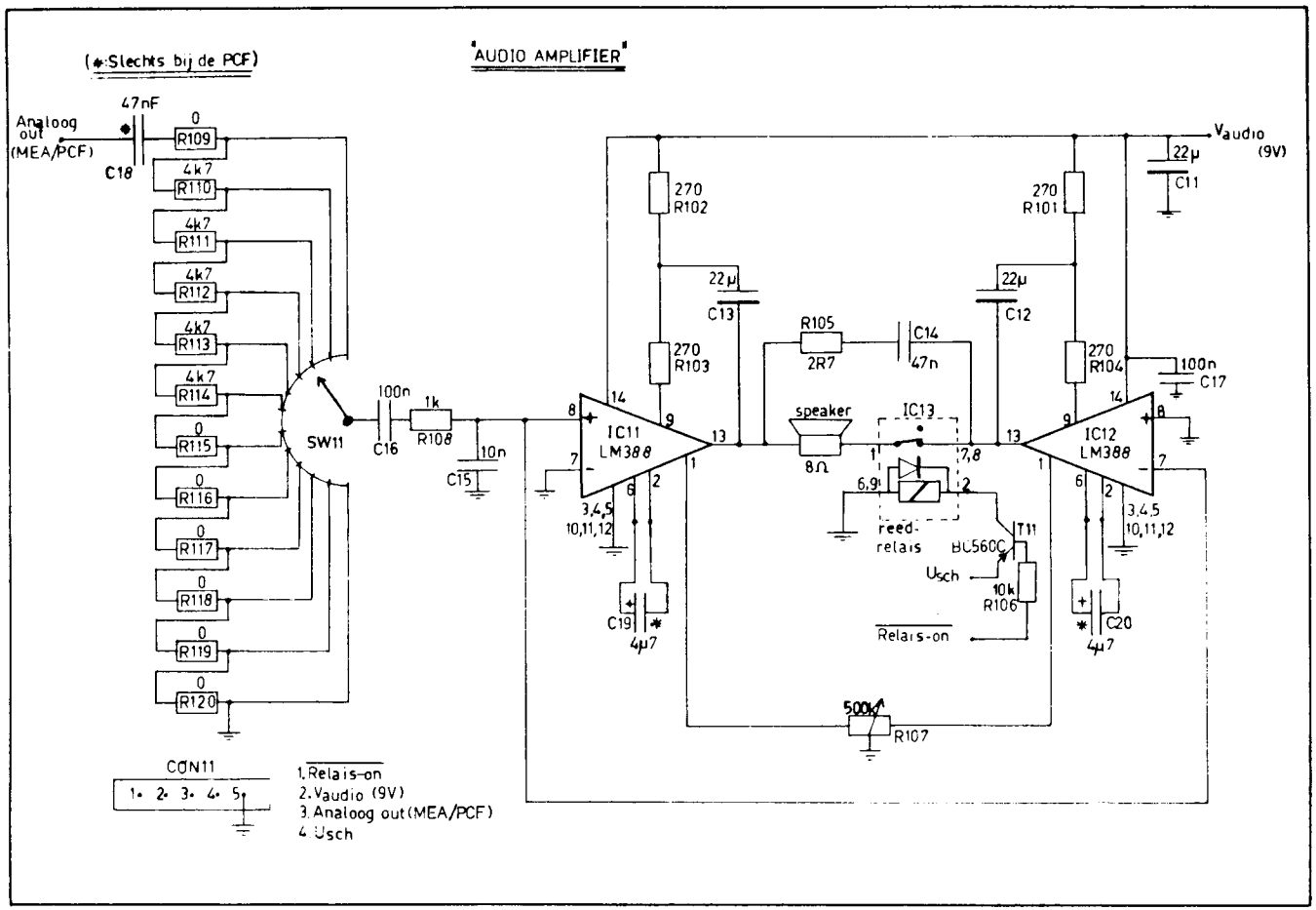
A Constructietekeningen.

A.1 Schema's.

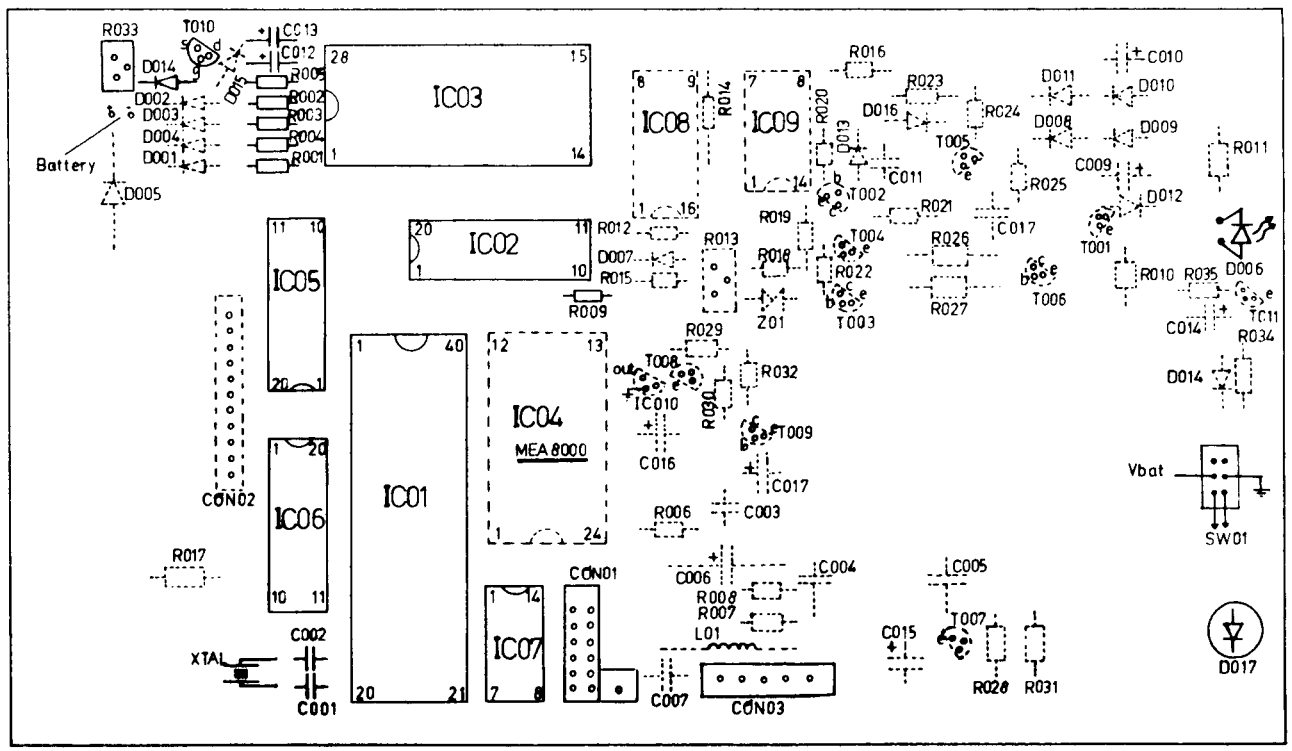

MEA8000 IC04
80C31 IC01
40373 IC02
27C256 IC03
74HC02 IC07
40373 IC06
40244 IC05
CONTROLBUS
DATABUS
"CSHII (MEA8000)"
LCD-CONNECTOR
KEYBOARD-CONNECTOR
POWER UP KEYBOARD
POWER CONTROL: CE(eprom)
CONNECTOR TOPPRINT-BOTTOMPRINT
1:Relais-on
2:Vaudio
3:Analoog-out MEA / PCF)
4:Usch
■= Input buffer 40244
●=Output latch 40373

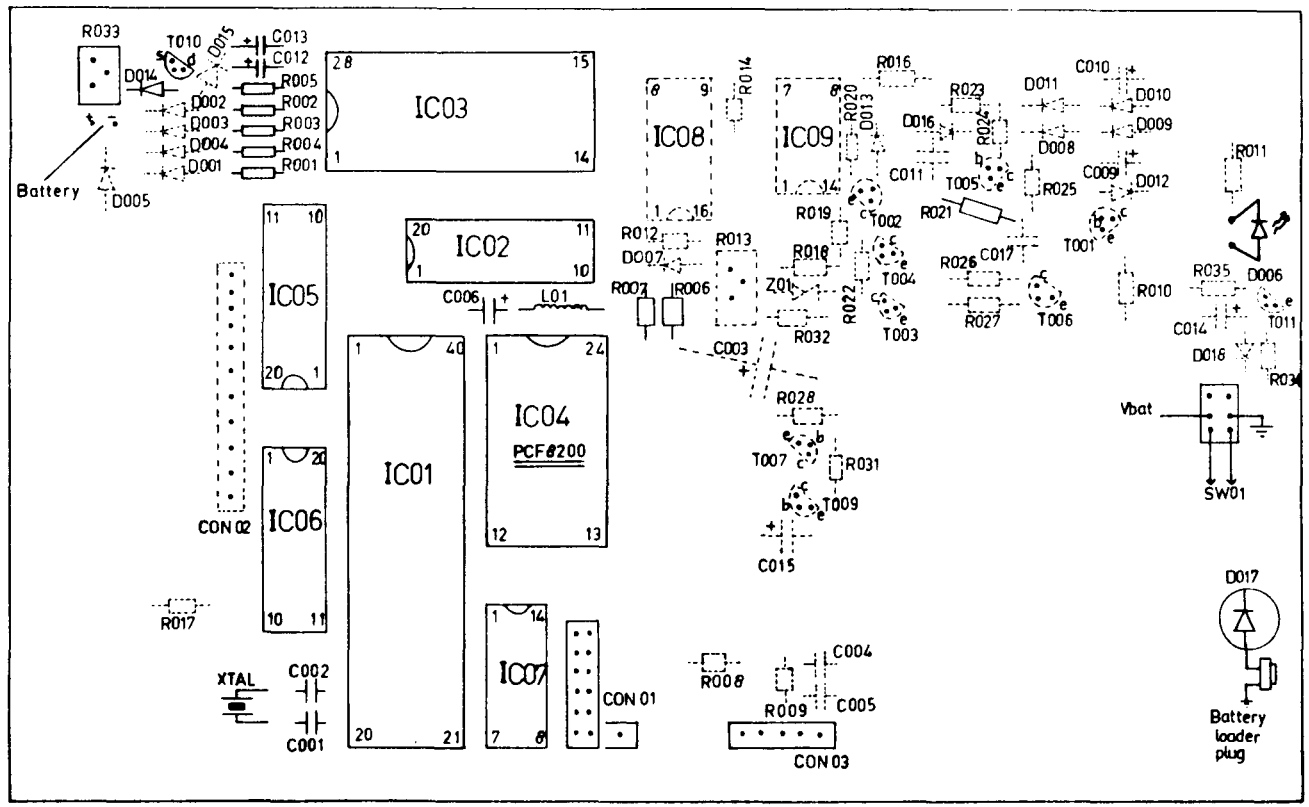

Usch
470k
R006
C003
47µ
390k
R007
100k
R008
56k
R009
270p
C004
560p
C005
Analoog OUT
PCF8200
IC04
Vss-D
TEST
Vss-A
OSCO
BUSY
VREF
OUT
SER/PAR
Vdd-D
Vdd-A
OSC IN
L01
1mH
1µ
C006
74HC02
IC07
40373
IC06
4µ7
C008
40244
IC05
R001
R002
R003
R004
R005
BAV10
D001
D002
D003
D004
100k
47k
80C31
IC01
22pF
C002
6MHz
C001
XTAL1
XTAL2
RESET
EA
Vcc
Vss
40373
IC02
27C256
IC03
CONTROLBUS
DATABUS
CE(LCD)
A0(LCD)
CE(PCF)
CE(keyboard)
Relais-on
bit1(power cont)
bit2(" ")
Battery LED
(Tiepstem)
(Tiepstem)
PCF
Battery contr.
Switch position
N.C.
WR(keyb/PCF)
RD(keyb/PCF)
CLK(PCF)
RXD
TXD
INT0
INT1
T0
T1
WR
RD
ALE
PSEN
Uprog
power control: RESET
POWER CONTROL: CE(eprom)
POWER UP KEYBOARD
LCD-CONNECTOR
CON01
KEYBOARD-CONNECTOR
CON02
CONNECTOR TOPPRINT-BOTTOMPRINT
CON03
■= Input buffer 40244
●=Output latch 40373
1:Relais-on
2:Vaudio
3:Analoog-out(PCF)
4:Usch
"CSH II (PCF8200)"
79

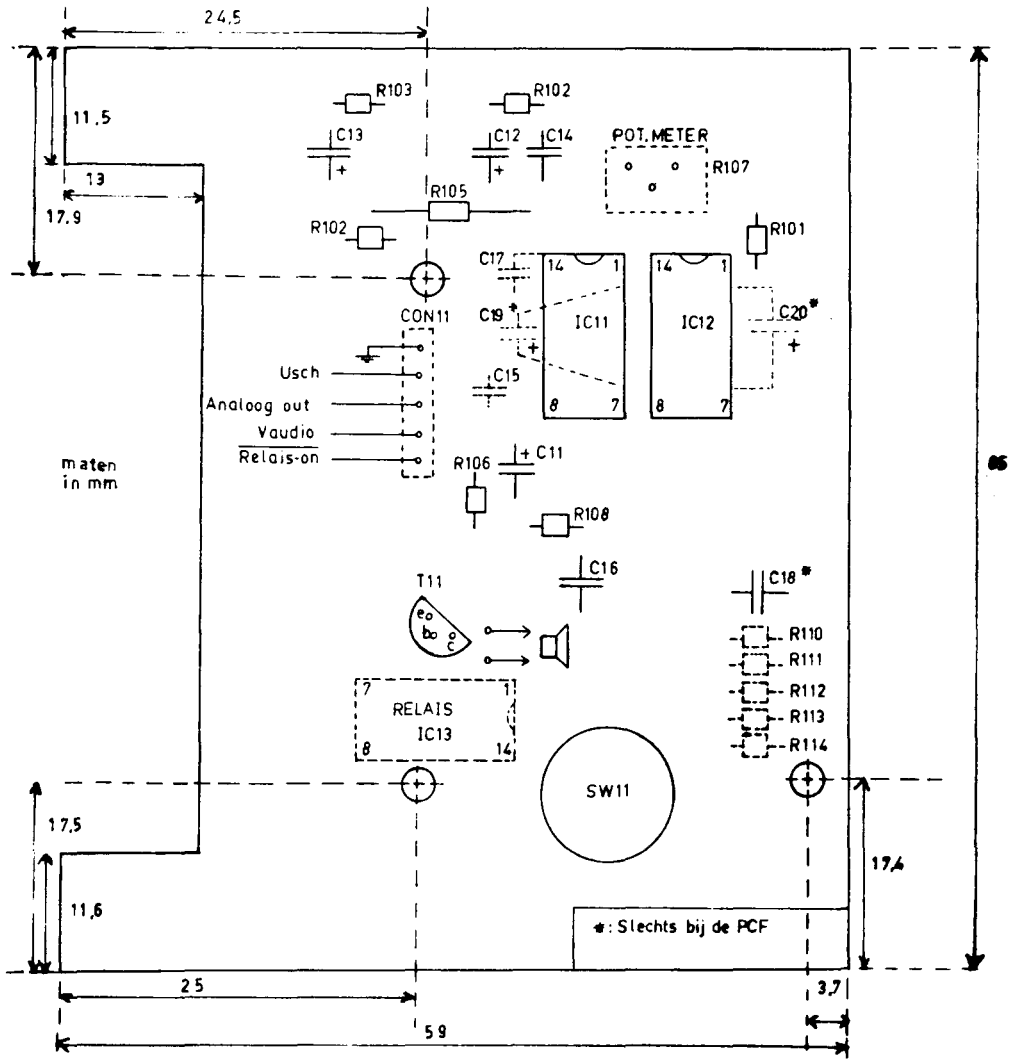

"POWER SUPPLY"
Battery LED
R010 100k
T001 BC560C
D006 CQW54
R011 1k
Usch
Vbat
BAV10 D013
100k
R020
100n C011
T010 BS250
1M R033
D014 BAV10
D015
BC550C T003
T002 BC560C
10k R019
10k R021
2k2 R022
T004
22µ C012
22µ C013
Uproc (5V)
Usch (5V)
Z01 5V6
10k R018
10k R035
D016 BAV10
10k R024
T005 BC560C
RESET
D018 BAV10
1µ C014
10k R023
T011 BC550C
10k R034
47k R026
4k7 R025
CE(eprom)
C017 100p
BIT1
BIT2
POWER UP KEYBOARD
4104 IC08
4025 IC09
bit1
bit2
470k R012
1N4005 D005
Vbat (9V)
500k R013
R014 3µ
D007 BAV10
Battery control
R015 1M
100k R027
BC550C T006
T007 BC327-40
Vaudio (9V)
22µ C015
10k R028
IC010 78L05
UMEA (5V)
22µ C016
BC560C T008
10k R029
R030 10k
R031 4k7
22µ C018
R032 10k
BC550C T009
C009 1µ
D008 BAV10 D009
R016 100k
Switch SW01
C010 1µ
D010 BAV10
D011 BAV10
1M R017
Switch position
BATTERY LOADER PLUG
BAV10 D017
D012
*NIET in de PCF-schakeling (Geen aparte Upcf nodig + al interne pull-up in µP bij R015)

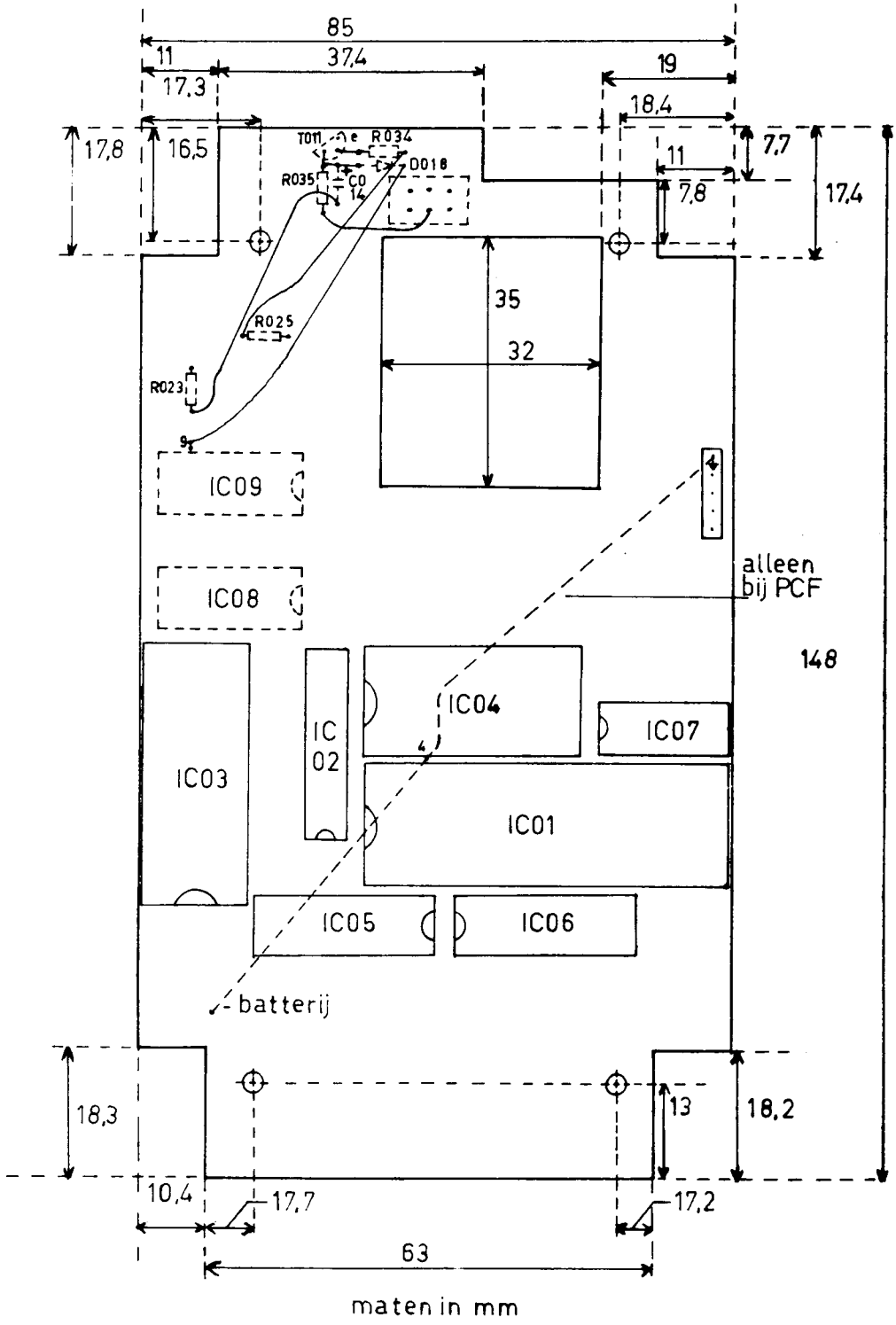

(*:Slechts bij de PCF)
"AUDIO AMPLIFIER"
Analoog out (MEA/PCF)
47nF
C18
0 R109
4k7 R110
4k7 R111
4k7 R112
4k7 R113
4k7 R114
0 R115
0 R116
0 R117
0 R118
0 R119
0 R120
SW11
100n C16
1k R108
10n C15
IC11 LM388
IC12 LM388
IC13
270 R102
270 R103
270 R101
270 R104
22µ C13
22µ C12
22µ C11
100n C17
R105 2R7
C14 47n
speaker 8Ω
reed-relais
T11 BC560C
Usch
10k R106
Relais-on
C19 4µ7
C20 4µ7
500k R107
Vaudio (9V)
3,4,5 10,11,12
6,9
7,8
CON11
1. Relais-on
2. Vaudio (9V)
3. Analoog out (MEA/PCF)
4. Usch

A.2 Printopstelling en onderdelenlijsten.


R033
T010
C013
C012
D014
D002
D003
D004
D001
R005
R002
R003
R004
R001
Battery
D005
IC03
28
15
1
14
IC08
8
9
16
1
R014
IC09
7
8
14
1
R016
R020
D013
R023
D016
R024
D011
D010
C010
D008
D009
C009
D012
R011
T005
C011
T002
R025
R021
C017
T001
R019
T004
R026
R022
R018
T006
R027
R010
R035
D006
T011
C014
R034
D014
Z01
T003
R012
D007
R015
R013
IC02
20
11
1
10
R009
IC05
11
10
20
1
CON02
R029
T008
R032
IC010
R030
C016
T009
C017
IC01
1
40
20
21
IC04
MEA8000
12
13
1
24
R006
C003
R017
CON01
C006
C004
R008
R007
L01
C005
C015
T007
R028
R031
XTAL
C002
C001
IC07
1
14
7
8
C007
CON03
Vbat
SW01
D017


R033
T010
D015
C013
C012
D014
R005
D002
R002
D003
R003
D004
R004
D001
R001
Battery
D005
IC03
IC08
R014
IC09
R016
R020
D013
R023
D011
C010
D010
D016
R024
D009
D008
C011
T005
R025
C009
D012
R021
T001
R011
R019
T002
R012
D007
R013
R018
T004
R035
D006
T011
R007
R006
Z01
R022
T003
R032
R026
C017
R027
T006
R010
C014
D018
R034
C003
C006
L01
IC02
IC05
R028
Vbat
T007
R031
T009
SW01
C015
IC04
PCF8200
IC01
CON 02
IC06
D017
R017
XTAL
C002
C001
IC07
CON 01
R008
C004
R009
C005
CON 03
Battery loader plug


24,5
11,5
13
17,9
R103
R102
C13
C12
C14
POT.METER
R107
R105
R102
R101
C17
C19*
CON11
IC11
IC12
C20*
C15
Usch
Analoog out
Vaudio
Relais-on
maten
in mm
R106
C11
R108
C16
T11
C18*
R110
R111
R112
R113
R114
RELAIS
IC13
SW11
17,5
11,6
17,6
*: Slechts bij de PCF
2,5
5,9
3,7


85
11
37,4
17,3
19
18,4
17,8
16,5
7,7
11
7,8
17,4
35
32
R034
D018
R035
C0
14
R025
R023
IC09
IC08
alleen
bij PCF
148
IC04
IC07
IC03
IC
02
IC01
IC05
IC06
batterij
18,3
13
18,2
10,4
17,7
17,2
63
maten in mm

A.2.1 Onderdelenlijst van de MEA-print.

Onderdelenlijst MEA-print						
R001	100 kΩ	IC01	80C31	μ-processor	C001	22 pF
R002	100 kΩ	IC02	40373	latch	C002	22 pF
R003	100 kΩ	IC03	27C256	EPROM	C003	10 nF
R004	100 kΩ	IC04	MEA8000	synthesizer	C004	100 nF
R005	47 kΩ	IC05	40244	buffer	C005	27 nF
R006	18 kΩ	IC06	40373	latch	C006	47 μF
R007	470 Ω	IC07	74HC02	NOR-gate	C007	47 nF
R008	47 Ω	IC08	4104	level-converter	C008	4.7 μF
R009	39 kΩ	IC09	4025	NOR-gate	C009	1 μF
R010	100 kΩ	IC010	78L05	volt. reg.	C010	1 μF
R011	1 kΩ				C011	100 nF
R012	470 kΩ				C012	22 μF
R013 [1]	500 kΩ	T001	BC560c	pnp	C013	22 μF
R014	3 MΩ	T002	BC560c	pnp	C014	1 μF
R015	1 MΩ	T003	BC550c	npn	C015	22 μF
R016	100 kΩ	T004	BC550c	npn	C016	22 μF
R017	1 MΩ	T005	BC560c	pnp	C017	100 pF
R018	10 kΩ	T006	BC550c	npn	C018	22 μF
R019	10 kΩ	T007	BC327-40	pnp		
R020	100 kΩ	T008	BC560c	pnp	D001	BAV10
R021	10 kΩ	T009	BC550c	npn	D002	BAV10
R022	2.2 kΩ	T010	BS250	p-channel FET	D003	BAV10
R023	10 kΩ	T011	BC550c	npn	D004	BAV10
R024	10 kΩ				D005	1N4005
R025	4.7 kΩ	L01	100 mH	choke	D006 [2]	CQW54
R026	47 kΩ				D007	BAV10
R027	100 kΩ	XTAL	4 MHz		D008	BAV10
R028	10 kΩ				D009	BAV10
R029	10 kΩ	Z01	BZX79C-5V6	zener	D010	BAV10
R030	10 kΩ				D011	BAV10
R031	4.7 kΩ				D012	BAV10
R032	10 kΩ				D013	BAV10
R033 [1]	1 MΩ				D014	BAV10
R034	10 kΩ				D015	BAV10
R035	10 kΩ				D016	BAV10
					D017	BAV10
					D018	BAV10

[1] Instel-potentiometer
[2] LED-rood

A.2.2 Onderdelenlijst van de PCF-print.

Onderdelenlijst PCF-print						
R001	100 kΩ	IC01	80C31	μ-processor	C001	22 pF
R002	100 kΩ	IC02	40373	latch	C002	22 pF
R003	100 kΩ	IC03	27C256	EPROM	C003	47 μF
R004	100 kΩ	IC04	PCF8200	synthesizer	C004	270 pF
R005	47 kΩ	IC05	40244	buffer	C005	560 pF
R006	470 kΩ	IC06	40373	latch	C006	1 μF
R007	390 kΩ	IC07	74hc02	NOR-gate		
R008	100 kΩ	IC08	4104	level-converter	C008	4.7 μF
R009	56 kΩ	IC09	4025	NOR-gate	C009	1 μF
R010	100 kΩ				C010	1 μF
R011	1 kΩ				C011	100 nF
R012	470 kΩ				C012	22 μF
R013 [1]	500 kΩ	T001	BC560c	pnp	C013	22 μF
R014	3 MΩ	T002	BC560c	pnp	C014	1 μF
		T003	BC550c	npn	C015	22 μF
R016	100 kΩ	T004	BC550c	npn		
R017	1 MΩ	T005	BC560c	pnp	C017	100 pF
R018	10 kΩ	T006	BC550c	npn		
R019	10 kΩ	T007	BC327-40	pnp		
R020	100 kΩ				D001	BAV10
R021	10 kΩ	T009	BC550c	npn	D002	BAV10
R022	2.2 kΩ	T010	BS250	p-FET	D003	BAV10
R023	10 kΩ	T011	BC550c	npn	D004	BAV10
R024	10 kΩ				D005	1N4005
R025	4.7 kΩ	L01	1 mH	choke	D006 [2]	CQW54
R026	47 kΩ				D007	BAV10
R027	100 kΩ	XTAL	4 MHz		D008	BAV10
R028	10 kΩ				D009	BAV10
		Z01	BZX79C-5V6	zener	D010	BAV10
					D011	BAV10
R031	4.7 kΩ				D012	BAV10
R032	10 kΩ				D013	BAV10
R033 [1]	1 MΩ				D014	BAV10
R034	10 kΩ				D015	BAV10
R035	10 kΩ				D016	BAV10
					D017	BAV10
					D018	BAV10

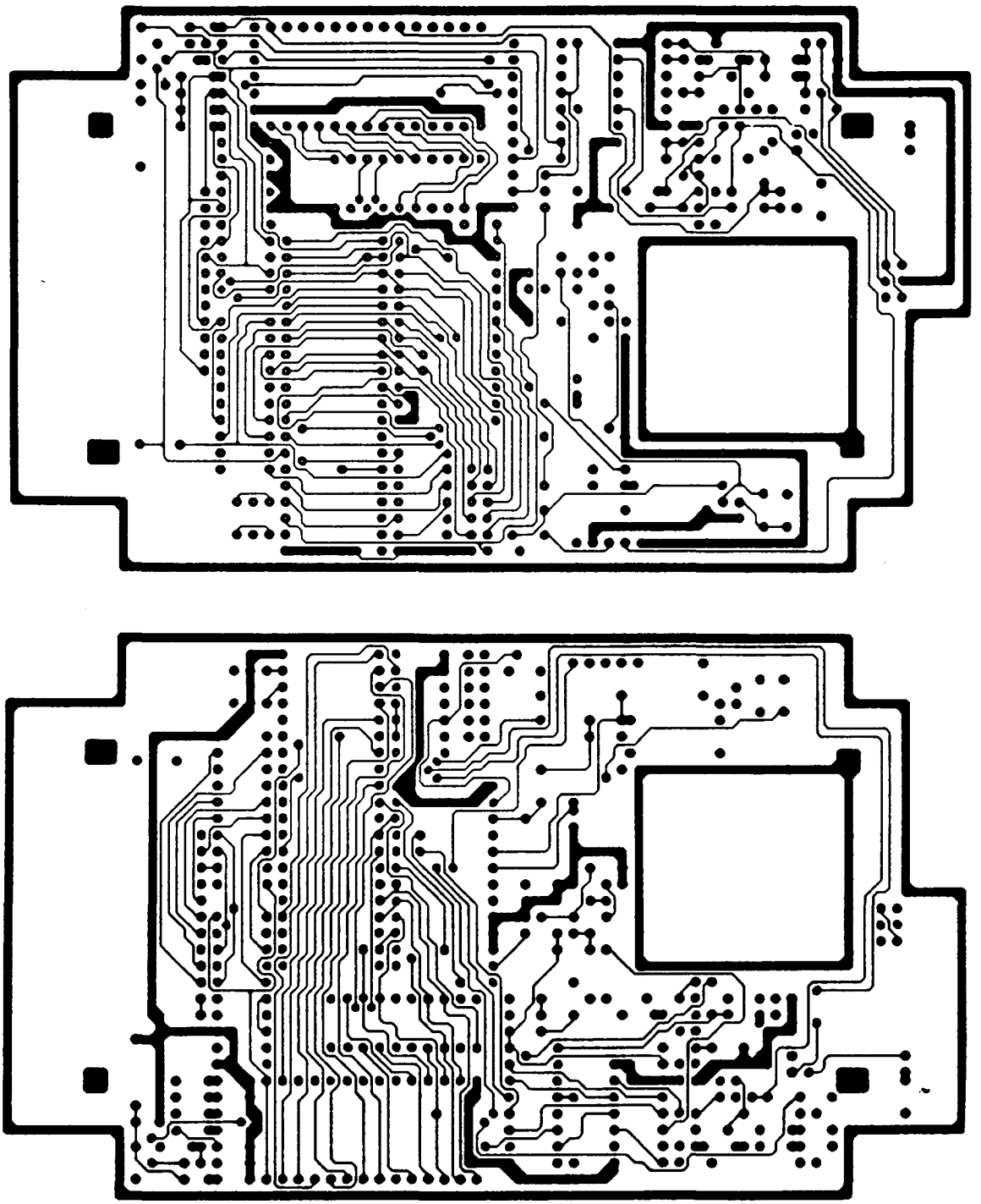
[1] Instel-potentiometer
[2] LED-rood

A.2.3 Onderdelenlijst van de Audio-print.

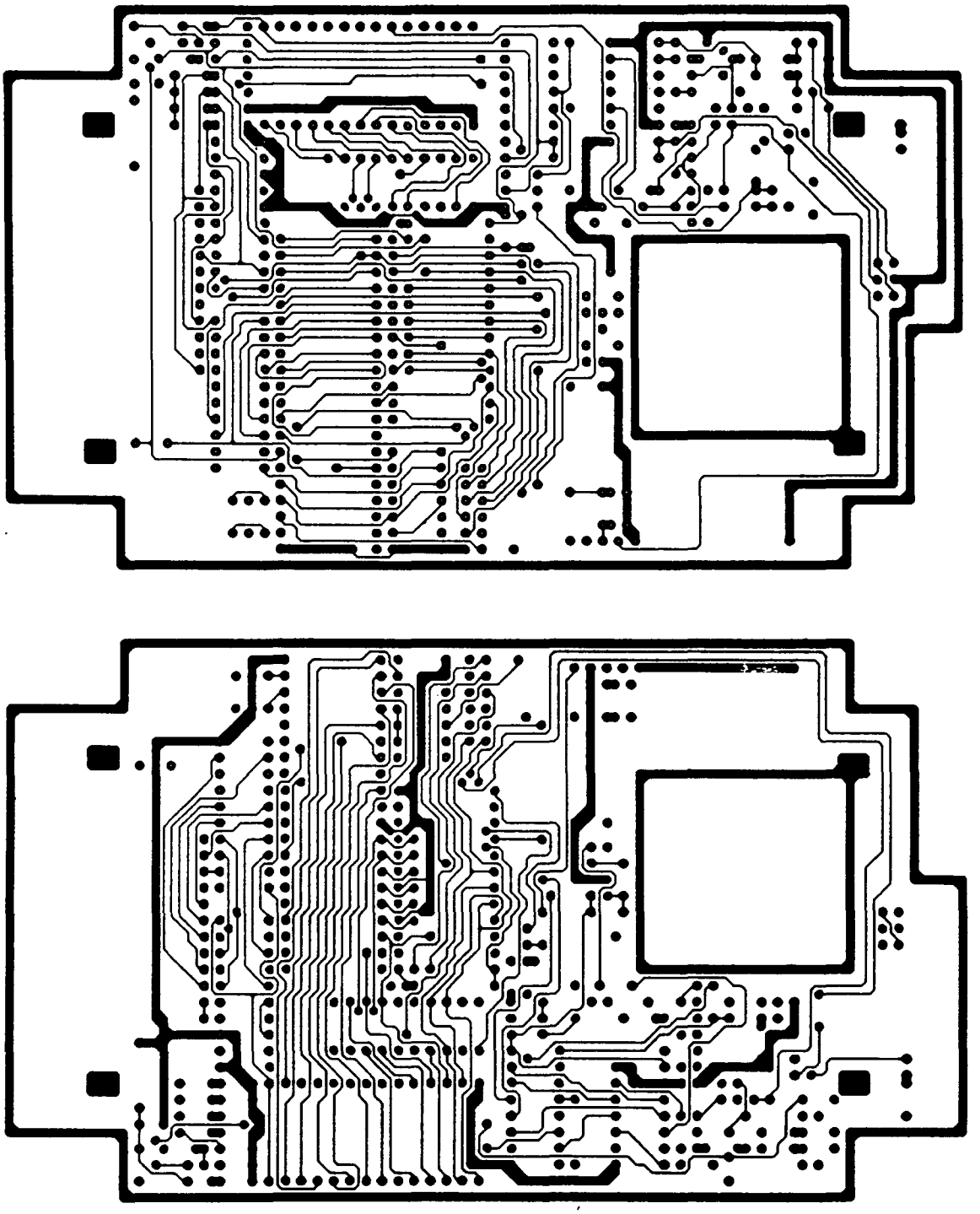
Onderdelenlijst Audio-print						
R101	270 Ω	IC11	LM388	audio-amp.	C11	22 μF
R102	270 Ω	IC12	LM388	audio-amp.	C12	22 μF
R103	270 Ω	IC13	Günther 3573 1231054	reed-relais	C13	22 μF
R104	270 Ω				C14	47 nF
R105	2.7 Ω	T11	BC560c	pnp	C15	10 nF
R106	10 kΩ				C16	100 nF
R107 [1]	500 kΩ				C17	100 nF
R108	1 kΩ				C18	47 nF
R110	4.7 kΩ				C19	4.7 μF
R111	4.7 kΩ				C20	4.7 μF
R112	4.7 kΩ					
R113	4.7 kΩ					
R114	4.7 kΩ					

[1] Instel-potentiometer

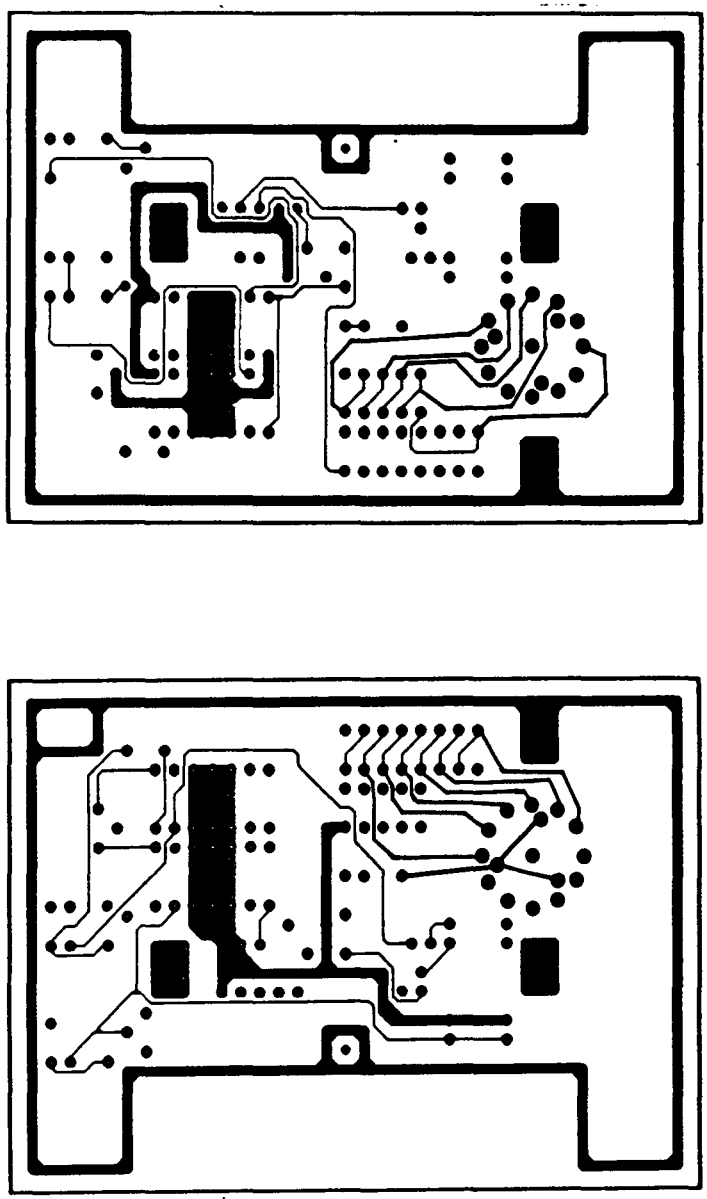
A.3 Printlay-out en aanzichttekeningen.



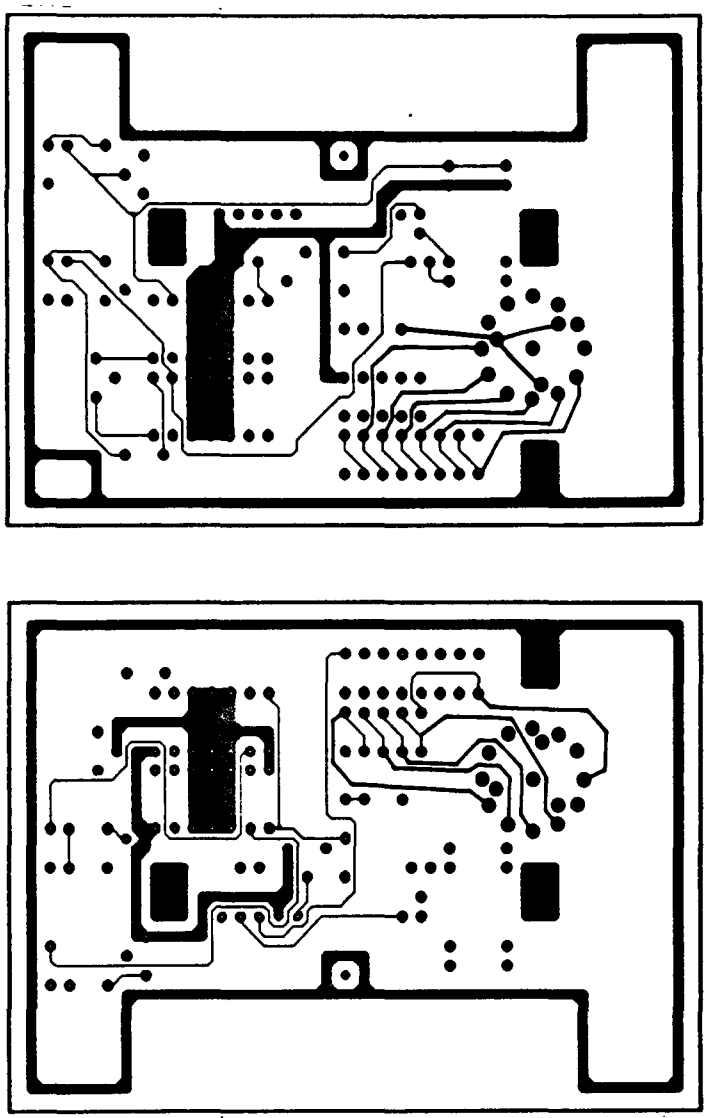
MEA8000 PRINT LAYOUT.



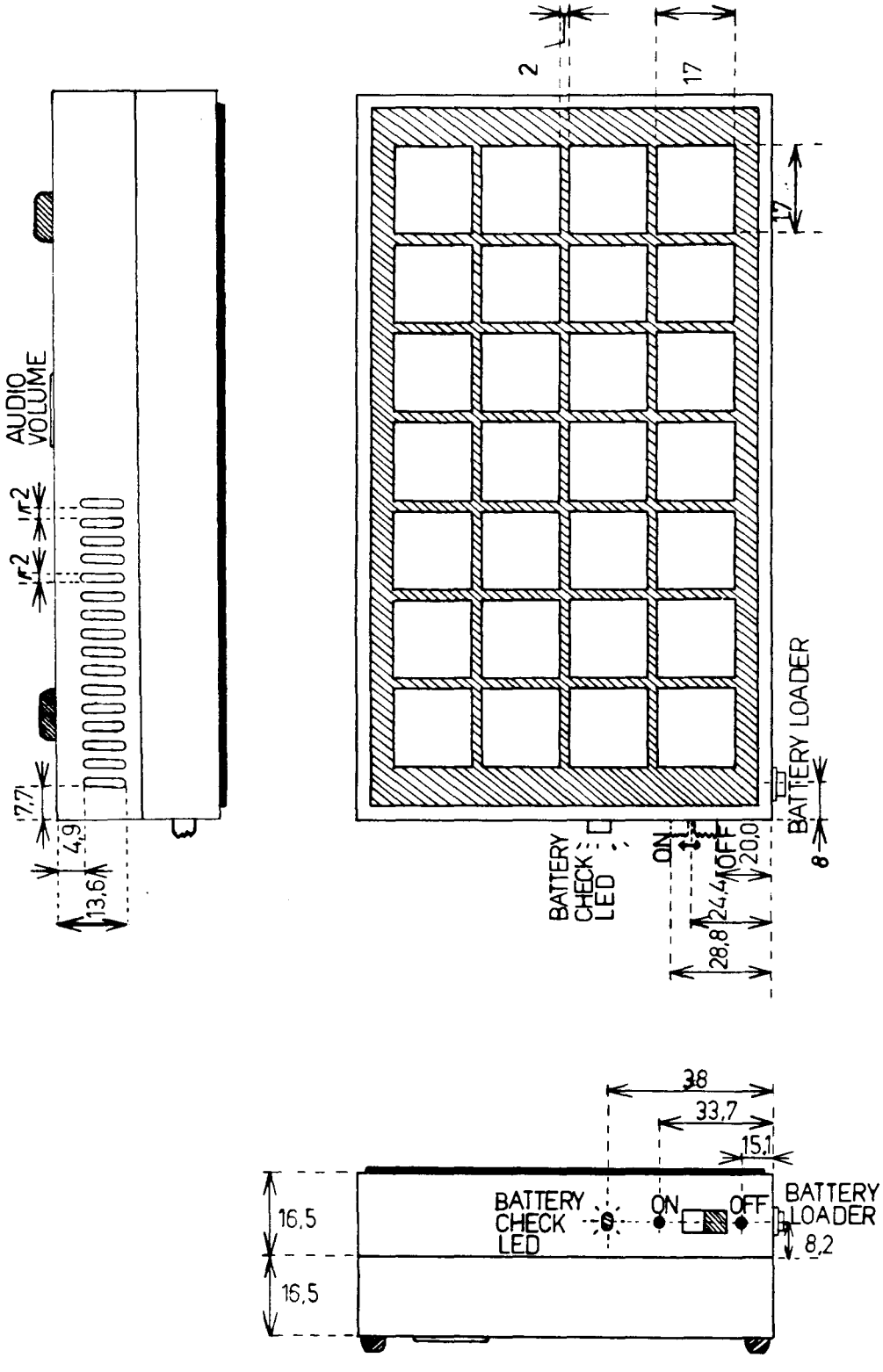
PCF8200 PRINT LAYOUT.

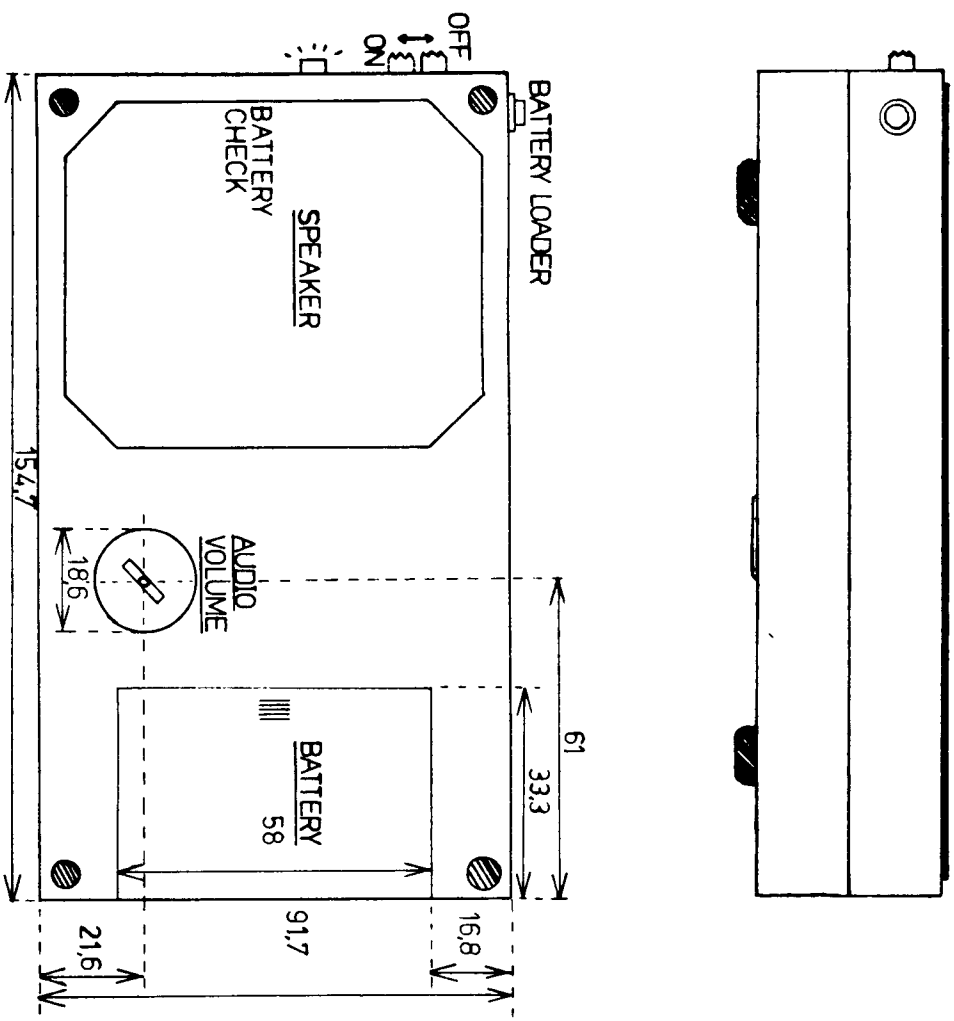


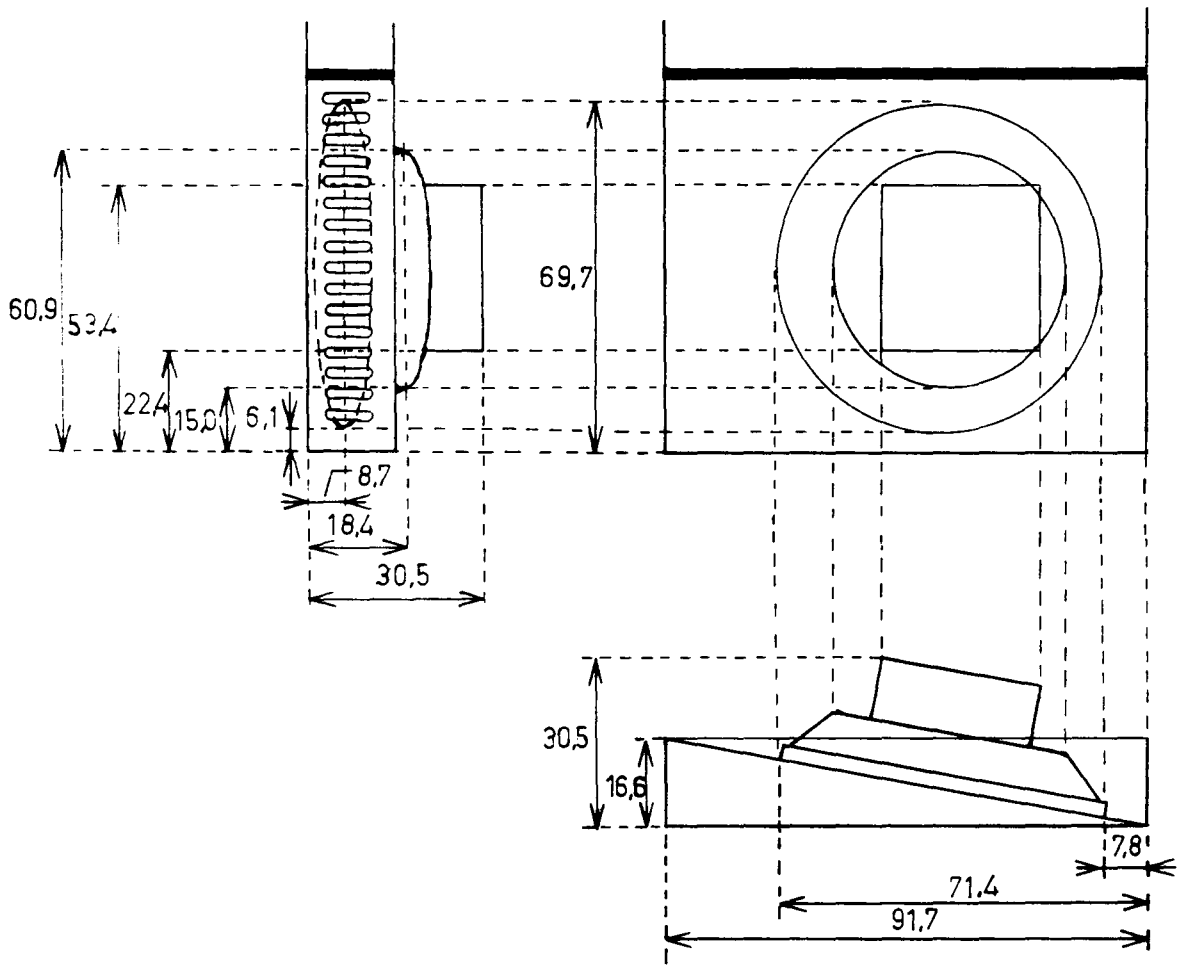
AUDIO PRINT LAYOUT (MEA8000).



AUDIO PRINT LAYOUT (PC F8200).

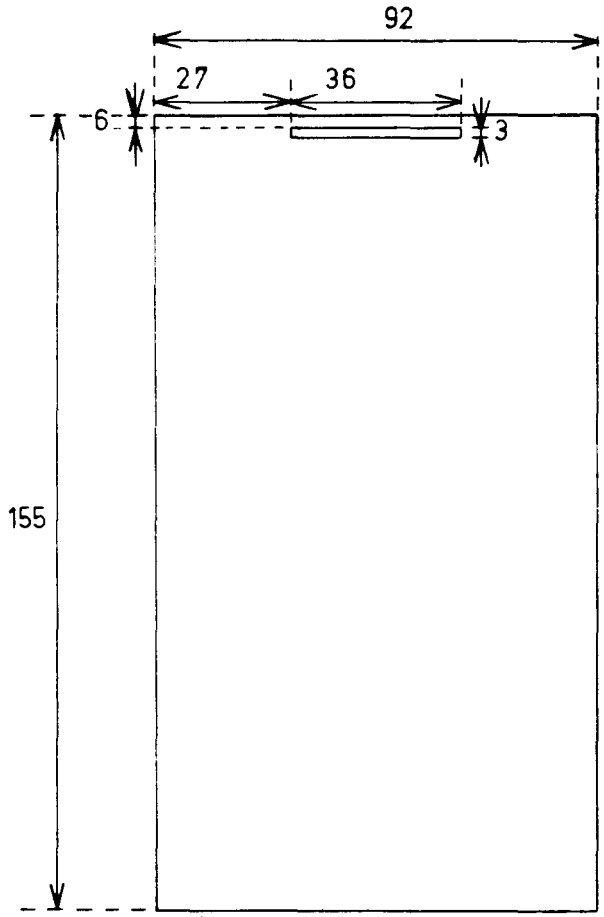

AUDIO VOLUME
2
17
17
BATTERY LOADER
BATTERY CHECK LED
ON
OFF
20.0
24.4
28.8
7.7
4.9
13.6
38
33.7
15.1
16.5
16.5
BATTERY CHECK LED
ON
OFF
BATTERY LOADER
8.2


OFF
ON
BATTERY LOADER
BATTERY CHECK
SPEAKER
AUDIO VOLUME
BATTERY
58
154,7
18,6
61
33,3
91,7
16,8
21,6



maten in mm gemeten van de buitenkant van het kastje

opstelling van de luidspreker



bovenaanzicht met
sleuf van keyboard

maten in mm

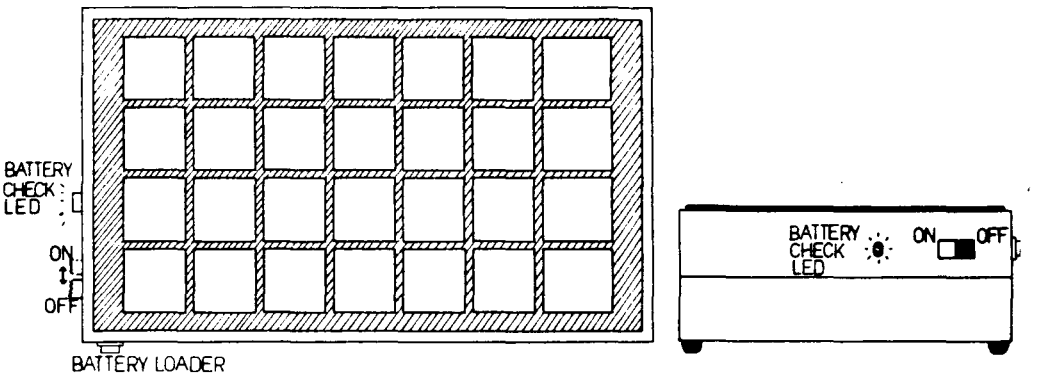
B Gebruiksaanwijzing.

AAN/UIT:

-UIT-stand is alleen nodig tijdens transport.

-AAN-stand in alle overige gebruikssituaties

BELANGRIJK: als het apparaat een storing vertoont dan volstaat het om de CSH II eenmaal uit en daarna weer aan te zetten. (mocht het apparaat dan nog niet goed werken dan moet het apparaat of opgeladen worden of technische hulp moet worden ingeroepen).



SPREKEN:

-de CSH II zal als het apparaat in de stand "AAN" staat op iedere druk op een van de toetsen reageren door de zin die onder die toets staat opgeborgen uit te spreken.

BELANGRIJK: dit geldt zolang de accu voldoende vermogen bezit om de zin uit te spreken. Is de accu te ver leeg dan zal de rode LED 4 seconden oplichten (zie ook "ACCU LEEG").

-het volume waarmee een zin wordt uitgesproken is regelbaar met de in de onderkant van de CSH II aangebrachte regelaar.

ACCU LEEG:

-indien de accu te ver leeg is geraakt dan zal de CSH II op een toets druk reageren door de "Batterij Controle LED" 4 seconden te laten branden. Na deze vier seconden schakelt het apparaat zich uit en zal niet meer op een toets reageren.

BELANGRIJK: om het apparaat weer te laten werken moet hij eenmaal UIT en weer AAN gezet worden (natuurlijk moet voor een goede werking eerst de accu worden opgeladen).

OPLADEN:

-na een aantal dagen gebruik, of na een dag intensief gebruik volstaat het om het apparaat gedurende een nacht aan de meegeleverde batterij-lader aan te sluiten. Gebruik geen andere dan de bijgeleverde oplader.

-opladen gebeurt door de lader in een wandkontaktdoos en de plug in de CSH II te steken (controle: de LED op de oplader zal gaan branden ten teken dat de accu wordt opgeladen).

C Programmalistings.

C.1 Het declaratie gedeelte.

```
!*****************************************************************************
!*                                                                           *
!*      General declarations                                                 *
!*                                                                           *
!*****************************************************************************
! P1.0 = LCD enable active low
#define lcden   p1(0)
! P1.1 = A0 command bit for MEA/LCD
#define comm    p1(1)
! P1.2 = chip enable for MEA or PCF active low
#define meaen   p1(2) ! pcfen for the PCF
! P1.3 = keyboard enable active low
#define keyen   p1(3)
! P1.4 = relais bit active low
#define relais  p1(4)
! P1.5 = bit 1 power control
#define pconb1  p1(5)
! P1.6 = bit 2 power control
#define pconb2  p1(6)
! P1.7 = LED bit when power failure
#define nled    p1(7)
!
! P3.4 = switch input
#define switch  p3(4)
! P3.3 = power check line
#define pcheckb p3(3)
! P3.5 = wis ram input
#define wisram  p3(5)
!
! DSW is device status word
!
!define DSW itself
#define dsw     0x20
!
!DSW bit 7 is power failure bit
#define powfail 0x07
!DSW bit 6 is device off
#define devoff  0x06
!DSW bit 5 is key-action bit
#define key     0x05
!DSW bit 4 is speaking bit
#define speaking 0x04
!DSW bit 3 is speakbuffer not empty
#define speane  0x03
!DSW bit 2 is length is zero (is length 2 in tabel)
#define leis0   0x02
!DSW bit 0 is second pronounciation bit
#define secout  0x00
!
!renumber pcon register (PMDS bug!!!)
#define pcon    0x87
```

C.2 Het programmagedeelte voor de MEA8000.

```
#
.list
!
!*****************************************************************************
!*                                                                           *
!*      M O D U L E     M E A P R O G                                        *
!*                                                                           *
!*                                                                           *
!*      Pocketstem program routines                                          *
!*                                                                           *
!*                                                                           *
!*                                                                           *
!*      Copyright march 1986                                                 *
!*      Institute for perception research, IPO                               *
!*      Eindhoven, the Netherlands                                           *
!*                                                                           *
!*      Author: Ir. R.P. Waterham                                            *
!*                                                                           *
!*****************************************************************************
!
#include "decl.h"
!
.sect    _reset
.base    0
.extern init
!
        ajmp    init
!
!*******************************************************************************
!
.sect    _mint
.base    0x03
.extern semea
!
        ajmp    semea           !go to serve mea routine
!
!*******************************************************************************
!
.sect    _pfintl
.base    0x13
.extern pfailr
!
        ajmp    pfailr          !if power fails then to power fail routine
!
!*******************************************************************************
!
.sect    _timint
.base    0x1b
!
        reti                    !no action necessary
!
!*******************************************************************************
!
.sect    _init
.define Init
.extern main
!
init:   mov     sp,#0x27        !set stack pointer on ram address 40, stack
        mov     a,r7            !check if r7 is within limits (24-31)
        clr     c               !always clear before substraction
        subb    a,#24           !borrow if a<24
        jnc     1f              !if carry then correct r7
        clr     c               !carry was set
        mov     r7,#24          !speakbuffer pointer on ram address 24
        mov     r6,#24          !speakbuffer pointer on ram address 24
        sjmp    2f              !finish initialisation
```

```
1:      mov     a,r7            !same for a>31
        subb    a,#32           !if carry then right values
        jc      3f              !
        mov     r7,#24          !speakbuffer pointer on ram address 24
        mov     r6,#24          !speakbuffer pointer on ram address 24
        sjmp    2f              !finish initialisation
3:      clr     c               !carry was set
        mov     6,7             !copy r7 to r6
2:      anl     dsw,#1          !reset DSW
        setb    pt1             !timer 1 high priority
        setb    ea              !enable all interrupts
        ajmp    main            !jump to main program
!
!*******************************************************************************
!
.sect   _main
.define main
.extern powch,swcon,scan,speak,standby,pfailr,outr
!
main:   lcall   swcon           !call routine to check position of switch
        jb      devoff,0f       !jump to local address 0 (then ajmp outr)
        lcall   powch           !call routine to check power
        jb      powfail,1f      !jump to local address 1 (then ajmp pfailr)
        lcall   scan            !scan keyboard (it sets speane bit)
        jnb     speane,2f       !if no key-action then go stand-by
        lcall   speak           !now speak sentence
        sjmp    main            !let main decide what to do
2:      ajmp    standby         !jump to routine to go stand-by
1:      ajmp    pfailr          !jump to routine to indicate powerfailure
0:      ajmp    outr            !jump to routine to go 'out'
!
!*******************************************************************************
!
.sect   _outr
.define outr
!
outr:   clr     keyen           !enable keyboard
        clr     a               !clear accu
        movx    @r0,a           !send accu to imaginary address (keyboard)
        setb    keyen           !disable keyboard
        clr     pconb1          !shut off power (go power down)
        mov     pcon,#0x2       !processor power down
!
!*******************************************************************************
!
.sect   _pfailr
.define pfailr
!
pfailr: clr     nled            !turn on powerfailure led
        setb    pconb2          !be sure mea is off!!!
        mov     tmod,#0x10      !set timer 1 ready to use
        mov     r0,#20          !realise 4 seconds time to wait
0:      mov     tl1,#00         !timer low reset
        mov     th1,#00         !timer high reset
        clr     tf1             !clear timer flag
        setb    tr1             !activate timer1
        setb    et1             !enable timer1 interrupt to come out of idle
        mov     pcon,#0x1       !go in idle mode
        djnz    r0,0b           !if r0 is not yet 0, then wait more
        clr     et1             !disable timer1 interrupt
        clr     tr1             !stop timer 1
        setb    nled            !led off again
        lcall   outr            !device has to go out!!
!
!*******************************************************************************
!
```

```
.sect   _standby
.define standby
!
standby:
        clr     keyen           !enable keyboard
        mov     a,#0xff         !write 1's to keyboard
        movx    @r0,a           !send accu to imaginairy address
        setb    keyen           !disable keyboard
        clr     pconbl          !shut off power
        mov     pcon,#0x02      !processor power down
        mov     pcon,#0x02      !to be sure!!
!
!*****************************************************************************
!
.sect   _swcon
.define swcon
!
swcon:  jb      switch,0f       !if switch is high then device is on
        setb    devoff          !if not then bit devoff is set in DSW
0:      ret
!
!*****************************************************************************
!
.sect   _powch
.define powch
!
powch:  jb      pcheckb,0f      !if power check bit (line P3.3) is high : oke
        setb    powfail         !if not then set powfail bit in DSW
0:      ret
!
!*****************************************************************************
!
.sect   _scan
.define scan
.extern calsno,incrr7,conspe,chr7
!
scan:   mov     r0,#250         !scan 250 times to prevent 'dender'
        mov     r1,#0           !reset r0
again:  clr     keyen           !enable keyboard
        mov     a,#0xff         !send 1's to keyboard
        movx    @r0,a           !send them to imaginairy address (keyboard)
        movx    a,@r0           !read keyboard
        setb    keyen           !disable keyboard
        anl     a,#0xc3         !be sure that only bits 7,6 and 1, 0 are good
        cjne    a,1,5f          !compare accu with reg 1
        sjmp    4f              !if equal then go on
5:      mov     r1,a            !save value in r1
        mov     r0,#250         !reset r0 on 250
4:      djnz    r0,again        !we want 250 times the same scan result!
        jnz     0f              !if accu 250 times equals zero then out
        clr     key             !no key action anymore
        lcall   conspe          !routine to control speane bit
        ret
0:      mov     a,#1            !now find which key it is
        mov     r0,a            !copy accu to r0
3:      clr     keyen           !enable keyboard
        movx    @r0,a           !write accu to keyboard (im. address)
        movx    a,@r0           !read keyboard line
        setb    keyen           !disable keyboard
        anl     a,#0xc3         !be sure that only bits 7,6 and 1, 0 are good
        jnz     1f              !jump if key is detected
        mov     a,r0            !copy r0 to accu
        rlc     a               !go to next scan line
        jc      2f              !if carry then end of scan
        mov     r0,a            !copy accu to r0 (backup)
        sjmp    3b              !try next line
```

```
2:      clr     c               !reset carry
        clr     key             !no key action after all
        lcall   conspe          !routine to control speane bit
        ret
1:      jnb     key,6f          !go on if no key action was present
!*************************************************************************
!a second key can only be detected if the first has been released in time.
!*************************************************************************
        lcall   conspe          !routine to control speane bit
        ret
6:      setb    key             !now indicate key action
        lcall   calsno          !calculate sentence number (result in r3)
        mov     1,7             !move content of r7 to r1
        mov     @r1,3           !move r3 to @r7
!*************************************************************************
!here action to determine whether sec. intonation is wanted!!!!!
!*************************************************************************
        lcall   chr7            !routine to check if previous sent was the same
        lcall   incrr7          !routine to incr speakpointer
        lcall   conspe          !routine to control speane bit
        ret
!
!*************************************************************************
!
.sect   _chr7
.define chr7
.extern incrr7,decrr7
!
chr7:   mov     r1,7            !load @r7 in accu
        mov     a,@r1           !done
        lcall   decrr7          !check with previous @r7
        mov     r1,7            !
        mov     0,@r1           !previous sentence in r0
        cjne    a,0,1f          !if not equal then return
        lcall   incrr7          !restore r7
        mov     r1,7            !reload @r7 in accu
        mov     a,@r1           !done
        setb    acc(7)          !set accubit 7
        mov     @r1,a           !store in @r7
        ret
1:      lcall   incrr7          !restore r7
        ret
!
!*************************************************************************
!
.sect   _calsno
.define calsno
!
calsno: mov     b,#0            !clear B register
        rrc     a               !look if 1 out of four bits is high
        jc      0f              !found a 1, so jump to counting the rest
        inc     b               !keep count in reg. B
        rrc     a               !do it again
        jc      0f              !until a 1 is found
        inc     b               !watch it; A looks like- 11xxxx11
        swap    a               !now look for bits 6 & 7
        rrc     a               !again
        jc      0f              !
        inc     b               !
        rrc     a               !
        jc      0f              !
        mov     b,#0            !no number found!! program never here!
        mov     r3,#0           !so 0 in r3
        ret
0:      clr     c               !clear carry because it was set
        mov     a,#7            !calculate b*7
```

```
        mul     ab              !muliply a(=7)*b
        mov     r3,a            !result in R3
        mov     r2,#0           !clear r2
        mov     a,r0            !scan address to accu
2:      rrc     a               !look for place of 1 in scan address
        jc      1f              !if carry is detected then jump
        inc     r2              !keep count in r2
        cjne    r2,#8,2b        !if r2=/8 then look again
        mov     r3,0            !no number found
        mov     r2,0            !then say first sentence
1:      clr     c               !reset carry
        mov     a,r3            !go for result
        add     a,r2            !sum is calculated
        mov     r3,a            !result in r3
        ret
!
!*******************************************************************************
!
.sect   _speak
.define speak
.extern ssas,sssl,wait12,xmtstp,gtpst,xmtpst,rdqfr,sqtmea,incrr6,display,conspe
!
speak:  clr     relais          !put relais on
        clr     pconb2          !that means power on mea
        clr     ie1             !to be sure that it was not set!!!!!
        setb    ex1             !enable pfail interrupt
        lcall   wait12          !wait about 12 msec. for power is in specs.
        setb    relais          !relais has done his job
1:      lcall   ssas            !search and store address sentence
        lcall   sssl            !search and store sentence length
        lcall   xmtstp          !transmit stop
        jb      leis0,3f        !if length is 0 then stop speaking
        lcall   gtpst           !get pitch start
        lcall   xmtpst          !transmit pitch start
        lcall   rdqfr           !read quartet from rom
        lcall   sqtmea          !send quartet to mea
        lcall   rdqfr           !read quartet from rom
        lcall   sqtmea          !send quartet to mea
        setb    speaking        !device is speaking now
        setb    ex0             !extern int 0 is enabled (level dep.)
!*******************************************************************************
        lcall   display         !display routine is still there!!!!!!
!*******************************************************************************
        lcall   incrr6          !now pointer r6 is increased
0:      mov     pcon,#0x1       !processor goes idle
        lcall   scan            !scan routine at each interrupt
!******* replace conspe for scan to anticipate multiple output ****************
        jb      speaking,0b     !if mea is still speaking then jump back
        jb      speane,1b       !jump if speak buffer is not empty
        clr     relais          !again relais on
2:      setb    pconb2          !mea power off
        lcall   wait12          !wait again about 12 msec.
        setb    relais          !relais has done his job
        ret
!
3:      lcall   incrr6          !correct r6
        sjmp    2b              !now go back
!
!*******************************************************************************
!
.sect   _wait12
.define wait12
!
wait12: mov     tmod,#0x10      !timer 1 ready for use
        mov     tl1,#0          !timer low reg. reset
        mov     th1,#0xd0       !high reg. 12288 counts from end = about 36 ms.
```

```
        clr     tf1             !reset flag
        setb    tr1             !activate timer 1
        setb    et1             !enable timer 1 interrupt
        mov     pcon,#0x1       !go in idle mode
        clr     et1             !disable timer 1 interrupt
        clr     tr1             !stop timer 1
        ret
!
!*******************************************************************************
!
.sect   _incrr7
.define Incrr7
!
incrr7: push    acc             !safety first!!!!!!!!
        inc     r7              !
        mov     a,r7            !move content of r7 to accu
        cjne    a,#32,0f        !compare accu (is r7) with 32 and act
        mov     r7,#24          !refill r7 with 24
0:      pop     acc             !safety first!!!!!!!
        ret
!
!*******************************************************************************
!
.sect   _decrr7
.define decrr7
!
decrr7: push acc                !safety first!!!!!!!
        dec     r7              !decrement r7  and watch for 23
        mov     a,r7            !check in accu
        cjne    a,#23,0f        !if r7 is 23 then reset to 31
        mov     r7,#31          !
0:      pop     acc             !safety first!!!!!!!
        ret
!
!*******************************************************************************
!
.sect   _incrr6
.define Incrr6
!
incrr6: push    acc             !safety first!!!!!!!
        inc     r6              !
        mov     a,r6            !move content of r6 to accu
        cjne    a,#32,0f        !compare accu (is r6) with 32 and act
        mov     r6,#24          !refill r6 with 24
0:      pop     acc             !safety first!!!!!!!
        ret
!
!*******************************************************************************
!
.sect   _conspe
.define conspe
!
conspe: mov     a,r7            !register 7 to accu
        cjne    a,6,1f          !compare accu with reg.6 and set speane
        clr     speane          !reset speak buffer not empty bit
        ret
1:      setb    speane          !set  speak buffer not empty bit
        ret
!
!*******************************************************************************
!
.sect   _ssas
.define ssas
!
ssas:   mov     r1,6            !move r6 to r1
        mov     a,@r1           !it is done
```

```
        setb    rs0             !select reg. bank 1
        mov     dph,#0x8        !now offset is 2k
        jnb     acc(7),1f       !if high bit if 1 then second intonation
        clr     acc(7)          !reset accu bit 7
        mov     dph,#0x40       !offset sec. int. is 4000 hex
1:      rl      a               !multiply the contents of a with 2
        mov     dpl,a           !this is lower order part of address
!*******************************************************************************
! now decide which offset is needed!!!!! 16k is 0x40
!*******************************************************************************
        clr     a               !data moves are relative to a
        movc    a,@a+dptr       !in fact it is a,@dptr
        mov     r5,a            !store including offset
        inc     dptr            !16 bit increment; get low order address
        clr     a               !same trick as before
        movc    a,@a+dptr       !get low order address
        mov     r4,a            !store in r4
        clr     rs0             !select reg. bank 0
        ret
!
!
!*******************************************************************************
!
.sect   _sssl
.define sssl
!
sssl:   setb    rs0             !routine works in reg bank 1
        mov     dph,r5          !fill data pointer
        mov     dpl,r4          !filled with address of sent. length
        inc     dptr            !first get lower order byte
        clr     a               !
        movc    a,@a+dptr       !lo sentence length in accu
        add     a,r4            !result is lo end address
        mov     r6,a            !store in r6
        dec     dpl             !now get high order (action with PCF)
        clr     a               !
        movc    a,@a+dptr       !ho sent. length in accu
        addc    a,r5            !result is ho end address (with lo overflow)
        mov     r7,a            !store in r7
        inc     dptr            !dptr on address of.....
        inc     dptr            !dptr on address of.....
        inc     dptr            !...pitchstart
        lcall   cilis0          !check if length is 0
        clr     rs0             !select reg. bank 0
        ret
!
!*******************************************************************************
!
.sect   _cilis0
.define cilis0
!
cilis0: mov     a,r7            !high end adress to accu
        clr     c               !reset carry before substraction
        subb    a,r5            !substract r5 from r7
        jnz     1f              !if unequal then no action
        mov     a,r6            !same trick for r6 and r4
        clr     c               !reset carry before substraction
        subb    a,r4            !substract r4 from r6
        cjne    a,#2,1f         !if result is not 2 then no action
        setb    leis0           !action is setting length is 0 bit
        clr     c               !clear carry (just in case)
        ret
1:      clr     c               !after substract carry can be set
        clr     leis0           !length is not 0 information to DSW
        ret
!
```

```
!*****************************************************************************
!
.sect   _xmtstp
.define xmtstp
!
xmtstp: setb    rs0             !select reg. bank 1
        jb      int0,.          !wait until int0 is low
        setb    comm            !mea in command mode
        clr     meaen           !enable mea
        mov     a,#0xfb         !mea in slow stop mode
        movx    @r0,a           !send accu to imaginairy address
        setb    meaen           !disable mea
        clr     rs0             !select reg. bank 0
        ret
!
!*****************************************************************************
!
.sect   _gtpst
.define gtpst
!
gtpst:  setb    rs0             !select reg. bank 1
        clr     a               !
        movc    a,@a+dptr       !get pitch start to accu
        mov     r0,#34          !save pitch start in reg 34
        mov     @r0,a           !done
        inc     dptr            !dptr to first data byte
        clr     rs0             !select reg. bank 0
        ret
!
!*****************************************************************************
!
.sect   _xmtpst
.define xmtpst
!
xmtpst: setb    rs0             !select reg. bank 1
        jb      int0,.          !wait until int0 is low
        clr     comm            !mea in speech mode
        mov     r0,#34          !get pst from @34
        mov     a,@r0           !pst to accu
        clr     meaen           !enable mea
        movx    @r0,a           !send accu to imaginairy address
        setb    meaen           !disable mea
        clr     rs0             !select reg. bank 0
        ret
!
!*****************************************************************************
!
.sect   _rdqfr
.define rdqfr
!
rdqfr:  setb    rs0             !select reg. bank 1
        mov     r3,#4           !counter on 4
        mov     r0,#36          !begin quartet buffer
0:      clr     a               !
        movc    a,@a+dptr       !first..to fourth byte from 4 to ram
        mov     @r0,a           !to ram address 36...39
        inc     r0              !next ram address
        inc     dptr            !next byte for mea
        djnz    r3,0b           !count until zero
        clr     rs0             !select reg. bank 0
        ret
!
!*****************************************************************************
!
.sect   _sqtmea
.define sqtmea
```

```
!
sqtmea: setb    rs0             !select reg. bank 1
        mov     r3,#4           !counter on 4
        mov     r0,#36          !begin quartet buffer
0:      jb      int0,.          !wait until int0 in low
        clr     comm            !put mea in speech mode
        clr     meaen           !enable mea
        mov     a,@r0           !move @r0 to accu
        movx    @r0,a           !send byte to mea
        setb    meaen           !disable mea
        inc     r0              !next byte address in ram
        djnz    r3,0b           !count until zero
        clr     rs0             !select reg. bank 0
        ret
!
!*******************************************************************************
!
.sect   _eoxmt
.define eoxmt
!
eoxmt:  setb    rs0             !select reg. bank 1
        mov     a,r6            !check if r6 == dpl
        cjne    a,dpl,0f        !if not == then no end yet
        mov     a,r7            !same for r7 and dph
        cjne    a,dph,0f        !if not == then no end yet
        clr     speaking        !end yet
0:      clr     rs0             !select reg. bank 0
        ret
!
!*******************************************************************************
!
.sect   _xmtdfr
.define xmtdfr
!
xmtdfr: setb    rs0             !select reg. bank 1
        mov     r3,#4           !counter on 4
0:      jb      int0,.          !wait until int0 is low
        clr     a               !send zero's
        clr     comm            !put mea in speech mode
        clr     meaen           !enable mea
        movx    @r0,a           !send to mea
        setb    meaen           !disable mea
        djnz    r3,0b           !count until r3 is zero
        clr     rs0             !select reg. bank 0
        ret
!
!*******************************************************************************
!
.sect   _semea
.define semea
.extern rdqfr,sqtmea,eoxmt,xmtdfr,xmtstp
!
semea:  push    acc             !accu on stack in this interrupt routine
        lcall   rdqfr           !read quartet from rom
        lcall   sqtmea          !send quartet to mea
        lcall   eoxmt           !end of transmission????
        jb      speaking,1f     !if speaking was reset then end of xmt.
        lcall   xmtdfr          !transmit dummy frame
        lcall   xmtstp          !transmit stop
        clr     ex0             !disable interrupt 0
1:      pop     acc             !accu back from stack
        reti
!
!*******************************************************************************
!
```

C.3 Het programmagedeelte voor de PCF8200.

```
#
.list
!
!*************************************************************************
!*                                                                       *
!*      M O D U L E     P C F P R O G                                    *
!*                                                                       *
!*                                                                       *
!*      Pocketstem program routines     with pcf synthesizer             *
!*                                                                       *
!*                                                                       *
!*                                                                       *
!*      Copyright march 1986                                             *
!*      Institute for perception research, IPO                           *
!*      Eindhoven, the Netherlands                                       *
!*                                                                       *
!*      Author: Ir. R.P. Waterham                                        *
!*                                                                       *
!*************************************************************************
!
#include "pcfdec.h"
!
.sect    _reset
.base    0
.extern init
!
         ajmp    init
!
!*************************************************************************************
!
.sect    _pint
.base    0x03
.extern sepcf
!
         ajmp    sepcf           !go to serve pcf routine
!
!*************************************************************************************
!
.sect    _pfint1
.base    0x13
.extern pfailr
!
         ajmp    pfailr          !if power fails then to power fail routine
!
!*************************************************************************************
!
.sect    _timint
.base    0x1b
!
         reti                    !no action necessary
!
!*************************************************************************************
!
.sect    _init
.define Init
.extern main
!
init:    mov     sp,#0x27        !set stack pointer on ram address 40, stack
         mov     a,r7            !check if r7 is within limits (24-31)
         clr     c               !always clear before substraction
         subb    a,#24           !borrow if a<24
         jnc     1f              !if carry then correct r7
         clr     c               !carry was set
         mov     r7,#24          !speakbuffer pointer on ram address 24
         mov     r6,#24          !speakbuffer pointer on ram address 24
         sjmp    2f              !finish initialisation
```

```
1:      mov     a,r7            !same for a>31
        subb    a,#32           !if carry then right values
        jc      3f              !
        mov     r7,#24          !speakbuffer pointer on ram address 24
        mov     r6,#24          !speakbuffer pointer on ram address 24
        sjmp    2f              !finish initialisation
3:      clr     c               !carry was set
        mov     6,7             !copy r7 to r6
2:      anl     dsw,#1          !reset DSW
        setb    pt1             !timer 1 high priority
        setb    ea              !enable all interrupts
        ajmp    main            !jump to main program
!
!*******************************************************************************
!
.sect   _main
.define main
.extern powch,swcon,scan,speak,standby,pfailr,outr
!
main:   lcall   swcon           !call routine to check position of switch
        jb      devoff,0f       !jump to local address 0 (then ajmp outr)
        lcall   powch           !call routine to check power
        jb      powfail,1f      !jump to local address 1 (then ajmp pfailr)
        lcall   scan            !scan keyboard (it sets speane bit)
        jnb     speane,2f       !if no key-action then go stand-by
        lcall   speak           !now speak sentence
        sjmp    main            !let main decide what to do
2:      ajmp    standby         !jump to routine to go stand-by
1:      ajmp    pfailr          !jump to routine to indicate powerfailure
0:      ajmp    outr            !jump to routine to go 'out'
!
!*******************************************************************************
!
.sect   _outr
.define outr
!
outr:   clr     keyen           !enable keyboard
        clr     a               !clear accu
        movx    @r0,a           !send accu to imaginary address (keyboard)
        setb    keyen           !disable keyboard
        clr     pconb1          !shut off power (go power down)
        mov     pcon,#0x2       !processor power down
!
!*******************************************************************************
!
.sect   _pfailr
.define pfailr
!
pfailr: clr     nled            !turn on powerfailure led
        setb    pconb2          !be sure audio is off!!!
        mov     tmod,#0x10      !set timer 1 ready to use
        mov     r0,#20          !realise 4 seconds time to wait
0:      mov     tl1,#00         !timer low reset
        mov     th1,#00         !timer high reset
        clr     tf1             !clear timer flag
        setb    tr1             !activate timer1
        setb    et1             !enable timer1 interrupt to come out of idle
        mov     pcon,#0x1       !go in idle mode
        djnz    r0,0b           !if r0 is not yet 0, then wait more
        clr     et1             !disable timer1 interrupt
        clr     tr1             !stop timer 1
        setb    nled            !led off again
        lcall   outr            !device has to go out!!
!
!*******************************************************************************
!
```

```
.sect   _standby
.define standby
!
standby:
        clr     keyen           !enable keyboard
        mov     a,#0xff         !write 1's to keyboard
        movx    @r0,a           !send accu to imaginairy address
        setb    keyen           !disable keyboard
        clr     pconbl          !shut off power
        mov     pcon,#0x02      !processor power down
        mov     pcon,#0x02      !to be sure!!
!
!*******************************************************************************
!
.sect   _swcon
.define swcon
!
swcon:  jb      switch,0f       !if switch is high then device is on
        setb    devoff          !if not then bit devoff is set in DSW
0:      ret
!
!*******************************************************************************
!
.sect   _powch
.define powch
!
powch:  jb      pcheckb,0f      !if power check bit (line P3.3) is high : oke
        setb    powfail         !if not then set powfail bit in DSW
0:      ret
!
!*******************************************************************************
!
.sect   _scan
.define scan
.extern calsno,incrr7,conspe,chr7
!
scan:   mov     r0,#250         !scan 250 times to prevent 'dender'
        mov     r1,#0           !reset r0
again:  clr     keyen           !enable keyboard
        mov     a,#0xff         !send 1's to keyboard
        movx    @r0,a           !send them to imaginairy address (keyboard)
        movx    a,@r0           !read keyboard
        setb    keyen           !disable keyboard
        anl     a,#0xc3         !be sure that only bits 7,6 and 1,0 are good
        cjne    a,1,5f          !compare accu with reg 1
        sjmp    4f              !if equal then go on
5:      mov     r1,a            !save value in r1
        mov     r0,#250         !reset r0 on 250
4:      djnz    r0,again        !we want 250 times the same scan result!
        jnz     0f              !if accu 250 times equals zero then out
        clr     key             !no key action anymore
        lcall   conspe          !routine to control speane bit
        ret
0:      mov     a,#1            !now find which key it is
        mov     r0,a            !copy accu to r0
3:      clr     keyen           !enable keyboard
        movx    @r0,a           !write accu to keyboard (im. address)
        movx    a,@r0           !read keyboard line
        setb    keyen           !disable keyboard
        anl     a,#0xc3         !be sure that only bits 7,6 and 1,0 are good
        jnz     1f              !jump if key is detected
        mov     a,r0            !copy r0 to accu
        rlc     a               !go to next scan line
        jc      2f              !if carry then end of scan
        mov     r0,a            !copy accu to r0 (backup)
        sjmp    3b              !try next line
```

```
2:      clr     c               !reset carry
        clr     key             !no key action after all
        lcall   conspe          !routine to control speane bit
        ret
1:      jnb     key,6f          !go on if no key action was present
!*****************************************************************************
!a second key can only be detected if the first has been released in time.
!*****************************************************************************
        lcall   conspe          !routine to control speane bit
        ret
6:      setb    key             !now indicate key action
        lcall   calsno          !calculate sentence number (result in r3)
        mov     1,7             !move content of r7 to r1
        mov     @r1,3           !move r3 to @r7
!*****************************************************************************
!here action to determine wether sec. intonation is wanted!!!!!
!*****************************************************************************
        lcall   chr7            !routine to check if previous sent was the same
        lcall   incrr7          !routine to incr speakpointer
        lcall   conspe          !routine to control speane bit
        ret
!
!*****************************************************************************
!
.sect   _chr7
.define chr7
.extern incrr7,decrr7
!
chr7:   mov     r1,7            !load @r7 in accu
        mov     a,@r1           !done
        lcall   decrr7          !check with previous @r7
        mov     r1,7            !
        mov     0,@r1           !previous sentence in r0
        cjne    a,0,1f          !if not equal then return
        lcall   incrr7          !restore r7
        mov     r1,7            !reload @r7 in accu
        mov     a,@r1           !done
        setb    acc(7)          !set accubit 7
        mov     @r1,a           !store in @r7
        ret
1:      lcall   incrr7          !restore r7
        ret
!
!*****************************************************************************
!
.sect   _calsno
.define calsno
!
calsno: mov     b,#0            !clear B register
        rrc     a               !look if 1 out of four bits is high
        jc      0f              !found a 1, so jump to counting the rest
        inc     b               !keep count in reg. B
        rrc     a               !do it again
        jc      0f              !until a 1 is found
        inc     b               !watch it; A looks like- 11xxxx11
        swap    a               !now look for bits 6 & 7
        rrc     a               !again
        jc      0f              !
        inc     b               !
        rrc     a               !
        jc      0f              !
        mov     b,#0            !no number found!! program never here!
        mov     r3,#0           !so 0 in r3
        ret
0:      clr     c               !clear carry because it was set
        mov     a,#7            !calculate b*7
```

```
        mul     ab              !muliply a(=7)*b
        mov     r3,a            !result in R3
        mov     r2,#0           !clear r2
        mov     a,r0            !scan address to accu
2:      rrc     a               !look for place of 1 in scan address
        jc      1f              !if carry is detected then jump
        inc     r2              !keep count in r2
        cjne    r2,#8,2b        !if r2=/8 then look again
        mov     r3,0            !no number found
        mov     r2,0            !then say first sentence
1:      clr     c               !reset carry
        mov     a,r3            !go for result
        add     a,r2            !sum is calculated
        mov     r3,a            !result in r3
        ret
!
!*******************************************************************************
!
.sect   _speak
.define speak
.extern ssas,sssl,wait12,xmtstp,gtpst,xmtpst,rdqifr,sqtpcf,incrr6,display,conspe
.extern sedac,secom
!
speak:  clr     relais          !put relais on
        clr     pconb2          !that means audio power on
        clr     iel             !to be sure that it was not set!!!
        setb    exl             !enable pfail interrupt
        lcall   wait12          !wait about 12 msec. for power is in specs.
        setb    relais          !relais has done his job
1:      lcall   ssas            !search and store address sentence
        lcall   sssl            !search and store sentence length
        lcall   sedac           !send dac amplitude factor
        jb      leis0,3f        !if length is 0 then stop speaking
        lcall   gtpst           !get pitch start
        lcall   xmtpst          !transmit pitch start
        lcall   secom           !send command to pcf
        lcall   rdqifr          !read quintet from rom
        lcall   sqtpcf          !send quintet to pcf
        lcall   rdqifr          !read quintet from rom
        lcall   sqtpcf          !send quintet to pcf
        setb    speaking        !device is speaking now
        setb    ex0             !extern int 0 is enabled (level dep.)
!*******************************************************************************
        lcall   display         !display routine is still there!!!!!!
!*******************************************************************************
        lcall   incrr6          !now pointer r6 is increased
0:      mov     pcon,#0x1       !processor goes idle
        lcall   scan            !scan routine at each interrupt
!******* replace conspe for scan to anticipate multiple output ****************
        jb      speaking,0b     !if pcf is still speaking then jump back
        jb      speane,1b       !jump if speak buffer is not empty
        clr     relais          !again relais on
2:      setb    pconb2          !pcf power off
        lcall   wait12          !wait again about 12 msec.
        setb    relais          !relais has done his job
        ret
!
3:      lcall   incrr6          !correct r6
        sjmp    2b              !now go back
!
!*******************************************************************************
!
.sect   _wait12
.define wait12
 !
wait12: mov     tmod,#0x10      !timer 1 ready for use
```

```
        mov     tl1,#0          !timer low reg. reset
        mov     th1,#0xa0       !high reg. 24576 counts from end = about 48 ms.
        clr     tf1             !reset flag
        setb    tr1             !activate timer 1
        setb    et1             !enable timer 1 interrupt
        mov     pcon,#0x1       !go in idle mode
        clr     et1             !disable timer 1 interrupt
        clr     tr1             !stop timer 1
        ret
!
!*******************************************************************************
!
.sect   _incrr7
.define incrr7
!
incrr7: push    acc             !safety first!!!!
        inc     r7              !
        mov     a,r7            !move content of r7 to accu
        cjne    a,#32,0f        !compare accu (is r7) with 32 and act
        mov     r7,#24          !refill r7 with 24
0:      pop     acc             !safety first!!!!
        ret
!
!*******************************************************************************
!
.sect   _decrr7
.define decrr7
!
decrr7: push acc                !safety first!!!!!!!
        dec     r7              !decrement r7  and watch for 23
        mov     a,r7            !check in accu
        cjne    a,#23,0f        !if r7 is 23 then reset to 31
        mov     r7,#31          !
0:      pop     acc             !safety first!!!!!!!
        ret
!
!*******************************************************************************
!
.sect   _incrr6
.define incrr6
!
incrr6: push    acc             !safety first!!!!
        inc     r6              !
        mov     a,r6            !move content of r6 to accu
        cjne    a,#32,0f        !compare accu (is r6) with 32 and act
        mov     r6,#24          !refill r6 with 24
0:      pop     acc             !safety first!!!!
        ret
!
!*******************************************************************************
!
.sect   _conspe
.define conspe
!
conspe: mov     a,r7            !register 7 to accu
        cjne    a,6,1f          !compare accu with reg.6  and set speane
        clr     speane          !reset speak buffer not empty bit
        ret
1:      setb    speane          !set  speak buffer not empty bit
        ret
!
!*******************************************************************************
!
.sect   _ssas
.define ssas
!
```

```
ssas:   mov     r1,6            !move r6 to r1
        mov     a,@r1           !it is done
        setb    rs0             !select reg. bank 1
        mov     dph,#0x8        !now offset is 2k
        jnb     acc(7),1f       !if high bit if 1 then second intonation
        clr     acc(7)          !reset accu bit 7
        mov     dph,#0x8        !offset sec. int. is 800 hex no memory space
                                !for blok 2 so no second intonation!!!
1:      rl      a               !multiply the contents of a with 2
        mov     dpl,a           !this is lower order part of address
!*****************************************************************************
! now decide which offset is needed!!!!! 16k is 0x40
!*****************************************************************************
        clr     a               !data moves are relative to a
        movc    a,@a+dptr       !in fact it is a,@dptr
        mov     r5,a            !store including offset
        inc     dptr            !16 bit increment; get low order address
        clr     a               !same trick as before
        movc    a,@a+dptr       !get low order address
        mov     r4,a            !store in r4
        clr     rs0             !select reg. bank 0
        ret
!
!*****************************************************************************
!
.sect   _sssl
.define sssl
!
sssl:   setb    rs0             !routine works in reg bank 1
        mov     dph,r5          !fill data pointer
        mov     dpl,r4          !filled with address of sent. length
        inc     dptr            !first get lower order byte
        clr     a               !
        movc    a,@a+dptr       !lo sentence length in accu
        add     a,r4            !result is lo end address
        mov     r6,a            !store in r6
!
!attention decr dpl can go over border of dph !!!action!!!
!it is because the pcf is 5 byte orientated!!
!
        mov     a,dpl           !check if dpl is zero
        jnz     1f              !if accu not zero then no action
        dec     dph             !low dp will underflow, that is corrected!!
1:      dec     dpl             !now get high order
        clr     a               !
        movc    a,@a+dptr       !ho sent. length in accu
        addc    a,r5            !result is ho end address (with lo overflow)
        mov     r7,a            !store in r7
        inc     dptr            !dptr on address of.....
!****************** here an extra inc of data pointer !!!!!!!!
        inc     dptr            !dptr on address of.....
        inc     dptr            !dptr on address of.....
        clr     a               !command byte goes to r0 of bank 2
        movc    a,@a+dptr       !first command bety from 4th byte 1st frame
        setb    rs1             !select reg bank 2
        clr     rs0             !select reg bank 2
        mov     r0,a            !copy accu to r0 (bank 2)
        clr     rs1             !select reg bank 1
        setb    rs0             !select reg bank 1
        inc     dptr            !...pitchstart
        lcall   cilis0          !check if length is 0
        clr     rs0             !select reg. bank 0
        ret
!
!*****************************************************************************
!
```

```
.sect   _cilis0
.define cilis0
!
cilis0: mov     a,r7            !high end adress to accu
        clr     c               !reset carry before substraction
        subb    a,r5            !substract r5 from r7
        jnz     1f              !if unequal then no action
        mov     a,r6            !same trick for r6 and r4
        clr     c               !reset carry before substraction
        subb    a,r4            !substract r4 from r6
        cjne    a,#2,1f         !if result is not 2 then no action
        setb    leis0           !action is setting length is 0 bit
        clr     c               !clear carry (just in case)
        ret
1:      clr     c               !after substract carry can be set
        clr     leis0           !length is not 0 information to DSW
        ret
!
!*******************************************************************************
!
.sect   _xmtstp
.define xmtstp
!
xmtstp: setb    rs0             !select reg. bank 1
        jb      int0,.          !wait until int0 is low
        clr     pcfen           !enable pcf
        mov     a,#0xe0         !pcf in command mode
        movx    @r0,a           !send accu to imaginairy address
        mov     a,#0x20         !pcf in stop mode and male voice
        movx    @r0,a           !send accu to imaginairy address
        setb    pcfen           !disable pcf
        clr     rs0             !select reg. bank 0
        ret
!
!*******************************************************************************
!
.sect   _gtpst
.define gtpst
!
gtpst:  setb    rs0             !select reg. bank 1
        clr     a               !
        movc    a,@a+dptr       !get pitch start to accu
        mov     r0,#34          !save pitch start in reg 34
        mov     @r0,a           !done
        inc     dptr            !dptr to first data byte
        clr     rs0             !select reg. bank 0
        ret
!
!*******************************************************************************
!
.sect   _xmtpst
.define xmtpst
!
xmtpst: setb    rs0             !select reg. bank 1
        jb      int0,.          !wait until int0 is low
        mov     r0,#34          !get pst from @34
        mov     a,@r0           !pst to accu
        clr     pcfen           !enable pcf
        movx    @r0,a           !send accu to imaginairy address
        setb    pcfen           !disable pcf
        clr     rs0             !select reg. bank 0
        ret
!
!*******************************************************************************
!
.sect   _rdqifr
```

```
.define rdqifr
!
rdqifr: setb    rs0             !select reg. bank 1
        mov     r3,#5           !counter on 5
        mov     r0,#35          !begin quartet buffer
0:      clr     a               !
        movc    a,@a+dptr       !first..to fifth byte from 5 to ram
!
!because the pcf in not perfect yet we have to catch some wrong bytes here
!
!
        cjne    r3,#5,1f        !if r3=5 then catch E0 (correct to D0)
        cjne    a,#0xe0,2f      !we catched E0
        mov     a,#0xd0         !we correct to D0
        sjmp    2f              !
1:      cjne    r3,#3,2f        !if r3=3 then catch xxxx 0000
        mov     r2,a            !save accu
        anl     a,#0x0f         !make accu 0000 xxxx
        cjne    a,#0,3f         !we catched xxxx 0000
        mov     a,r2            !restore accu
        add     a,#1            !we make accu xxxx 0001
        sjmp    2f              !
3:      mov     a,r2            !if we don't correct then restore accu
!
! this part was just for correcting the pcf!!!!!
!
2:      mov     @r0,a           !to ram address 35...39
        clr     c               !could have been set!!!
        inc     r0              !next ram address
        inc     dptr            !next byte for pcf
        djnz    r3,0b           !count until zero
        clr     rs0             !select reg. bank 0
        ret
!
!*******************************************************************************
!
.sect   _sqtpcf
.define sqtpcf
!
sqtpcf: setb    rs0             !select reg. bank 1
        mov     r3,#5           !counter on 5
        mov     r0,#35          !begin quintet buffer
0:      jb      int0,.          !wait until int0 in low
        clr     pcfen           !enable pcf
        mov     a,@r0           !move @r0 to accu
        movx    @r0,a           !send byte to pcf
        setb    pcfen           !disable pcf
        inc     r0              !next byte address in ram
        djnz    r3,0b           !count until zero
        clr     rs0             !select reg. bank 0
        ret
!
!*******************************************************************************
!
.sect   _eoxmt
.define eoxmt
!
eoxmt:  setb    rs0             !select reg. bank 1
        mov     a,r6            !check if r6 == dpl
        cjne    a,dpl,0f        !if not == then no end yet
        mov     a,r7            !same for r7 and dph
        cjne    a,dph,0f        !if not == then no end yet
        clr     speaking        !end yet
0:      clr     rs0             !select reg. bank 0
        ret
!
```

```
!*****************************************************************************
!
.sect   _sepcf
.define sepcf
.extern rdqifr,sqtpcf,eoxmt,xmtstp
!
sepcf:  push    acc             !accu on stack in this interrupt routine
        lcall   rdqifr          !read quintet from rom
        lcall   sqtpcf          !send quintet to pcf
        lcall   eoxmt           !end of transmission????
        jb      speaking,1f     !if speaking was reset then end of xmt.
        lcall   xmtstp          !transmit stop
        clr     ex0             !disable interrupt 0
1:      pop     acc             !accu back from stack
        reti
!
!*****************************************************************************
!
.sect   _sedac
.define sedac
!
sedac:  setb    rs0             !select reg. bank 1
        clr     pcfen           !enable pcf
1:      movx    a,@r0           !first read busy bit of status byte
        jb      acc(6),1b       !if pcf is busy then loop
        mov     a,#0x30         !dac factor is 3
        movx    @r0,a           !send factor to pcf
        setb    pcfen           !disable pcf
        clr     rs0             !select reg. bank 0
        ret
!
!*****************************************************************************
!
.sect   _secom
.define secom
!
secom:  setb    rs0             !select reg. bank 1
        jb      int0,.          !wait until interrupt is low
        clr     pcfen           !enable pcf
        mov     a,#0xe0         !set pcf in command mode
        movx    @r0,a           !send E0 to pcf
! whether voice is male or female depends on 4th byte of 1st frame
        setb    rs1             !select reg bank 2
        clr     rs0             !select reg bank 2
        mov     a,r0            !now command byte is in accu
        clr     rs1             !select reg bank 1
        setb    rs0             !select reg bank 1
        movx    @r0,a           !send 00 to pcf
        setb    pcfen           !disable pcf
        clr     rs0             !select reg. bank 0
        ret
```

C.4 Het display-aansturingsprogramma.

```
!
!*************************************************************************
!*                                                                       *
!*      M O D U L E     D I S P L A Y                                    *
!*                                                                       *
!*                                                                       *
!*                                                                       *
!*                                                                       *
!*                                                                       *
!*      Copyright march 1986                                             *
!*      Institute for perception research, IPO                           *
!*      Eindhoven, the Netherlands                                       *
!*                                                                       *
!*      Author: Ir. R.P. Waterham                                        *
!*                                                                       *
!*************************************************************************
!
.sect   _display
.define display
.extern store,rsds,calas,cale,gsch,line2,disres
!
display:push    dpl             !save datapointer
        push    dph             !both of them
        jb      wisram,1f       !if p35 is low then wis ram!!
        mov     r0,#64          !clear ram area 64 to 127
9:      mov     @r0,#0          !fill with 0
        inc     r0              !next ram address
        cjne    r0,#128,9b      !until reg. 128
        ret
1:      lcall   store           !store sentence number of times used
        lcall   rsds            !reset display
        mov     r2,#16          !length of row is 16
        lcall   calas           !calculate address of text
        lcall   cale            !get length of text
2:      lcall   gsch            !get and send character
        djnz    r2,3f           !still at line one
        lcall   line2           !go to second line of display
3:      cjne    r3,#0,2b        !max 28 char. to be displayed
        lcall   disres          !in the end display times used
        pop     dph             !get datapointer back
        pop     dpl             !both of them
        ret
!
!*************************************************************************
!
.sect   _line2
.define line2
!
line2:  mov     a,#0xc0         !C0 is address of new line
        clr     comm            !put lcd in command mode
        clr     lcden           !enable lcd
        movx    @r0,a           !send to lcd
        setb    lcden           !disable lcd
        ret
!
!*************************************************************************
!
.sect   _rsds
.define rsds
.extern wait3
!
rsds:   mov     a,#0x10         !system reset display
        clr     comm            !put lcd in command mode
        clr     lcden           !enable lcd
        movx    @r0,a           !write to lcd
```

```
        setb    lcden           !disable lcd
        mov     a,#0x1          !clear display
        mov     r0,#3           !wait a bit
        djnz    r0,.            ! ,,
        clr     lcden           !enable lcd
        movx    @r0,a           !send to lcd
        setb    lcden           !disable lcd
        lcall   wait3           !watch it specs. say nsec. but it is msec.
        mov     a,#0x0e         !cursor on
        clr     lcden           !enable lcd
        movx    @r0,a           !send to lcd
        setb    lcden           !disable lcd
        mov     a,#0x0d         !display on
        mov     r0,#3           !wait a bit
        djnz    r0,.            !  ,,
        clr     lcden           !enable display
        movx    @r0,a           !send to lcd
        setb    lcden           !disable lcd
        ret
!
!*******************************************************************************
!
.sect   _calas
.define calas
!
calas:  mov     1,r6            !in @r6 pointer of sentence
        mov     a,@r1           !get it to accu
        clr     acc(7)          !reset sec int bit
        rl      a               !double accu
        mov     dpl,a           !low order to dptr
        mov     dph,#0x78       !high order address is 78 (xx)
        clr     a               !
        movc    a,@a+dptr       !this is high order address
        mov     r5,a            !store in r5
        inc     dptr            !next is low order
        clr     a               !
        movc    a,@a+dptr       !get it from rom
        mov     r4,a            !store in r4
        ret
!
!*******************************************************************************
!
.sect   _cale
.define cale
!
cale:   mov     dpl,r4          !get dptr
        mov     dph,r5          ! ,,
        clr     a               !
        movc    a,@a,dptr       !first byte is length of sentence
        mov     r3,a            !length in r3
        inc     dptr            !pointer on first letter
        mov     r4,dpl          !store again
        mov     r5,dph          ! ,,
        cjne    a,#28,1f        !if length is 27 or lower carry will be set
1:      jc      2f              !if no carry then max length is 28
        mov     r3,#28          !
2:      clr     c               !to be sure
        ret
!
!*******************************************************************************
!
.sect   _gsch
.define gsch
!
gsch:   clr     a               !get character
        mov     dpl,r4          !
```

```
        mov     dph,r5          !
        movc    a,@a+dptr       !out of memory
        setb    comm            !put lcd in data mode
        clr     lcden           !enable lcd
        movx    @r0,a           !send char.
        setb    lcden           !disable lcd
        inc     dptr            !next address
        mov     r4,dpl          !back-up
        mov     r5,dph          !back-up
        dec     r3              !length decreases one
        ret
!
!*******************************************************************************
!
.sect   _store
.define store
!
store:  mov     1,r6            !
        mov     a,@r1           !@r6 to accu
        clr     acc(7)          !reset sec int bit
        rl      a               !place is no. * 2
        add     a,#64           !offset is 64
        mov     r1,a            !r1 is pointer
        mov     a,@r1           !get byte to accu
        clr     c               !to be sure
        add     a,#1            !decimal increment accu
        da      a               !is now done
        mov     @r1,a           !store result
        inc     r1              !next byte
        mov     a,@r1           !to accu
        addc    a,#0            !add carry to accu
        da      a               !and then dec. adjust accu
        mov     @r1,a           !store result
        clr     c               !to be sure (could be set if 10.000 occurs)
        ret
!
!*******************************************************************************
!
.sect   _disres
.define disres
!
disres: mov     a,#0xcc         !cc is place no. 12 on row two of lcd
        clr     comm            !lcd in command mode
        clr     lcden           !enable lcd
        movx    @r0,a           !send to lcd
        setb    lcden           !disable lcd
        setb    comm            !lcd in data mode
        mov     1,r6            !
        mov     a,@r1           !@r6 to accu
        clr     acc(7)          !reset sec int bit
        rl      a               !place is no. * 2
        add     a,#64           !offset
        mov     r1,a            !
        inc     r1              !first do high order
1:      mov     a,@r1           !high order decimal
        swap    a               !high nibble } low nibble
        anl     a,#0x0f         !clear high nibble
        add     a,#0x30         !make ascii number
        clr     lcden           !enable lcd
        movx    @r0,a           !send ascii to lcd
        setb    lcden           !disable lcd
        mov     a,@r1           !again for lower nibble
        anl     a,#0x0f         !clear high nibble
        add     a,#0x30         !make ascii number
        clr     lcden           !enable lcd
        movx    @r0,a           !send to lcd
```

```
        setb    lcden           !disable lcd
        dec     r1              !get previous byte
        mov     a,r1            !
        jnb     acc(0),lb       !if r1 is odd then once again
        ret
!
!*******************************************************************************
!
.sect   _wait3
.define wait3
!
wait3:  mov     tmod,#0x10      !timer 1 ready for use
        mov     tl1,#0          !reset timer low reg.
        mov     th1,#0xfc       !high reg. 1024 counts from end = about 3 ms.
        clr     tf1             !reset flag
        setb    tr1             !activate timer 1
        setb    et1             !enable timer 1 interrupt
        mov     pcon,#0x1       !go in idle mode
        clr     et1             !disable timer 1 interrupt
        clr     tr1             !stop timer 1
        ret
!
!*******************************************************************************
!
```