IEEE-488 MEA 8000 interface op basis van LABBUS systeem

INSTITUUT VOOR PERCEPTIE ONDERZOEK
Den Dolech 2 - Eindhoven

Rapport nr. 514

IEEE-488 MEA 8000 INTERFACE
op basis van LABBUS systeem

E.A. de Koning

Verslag van een afstudeeropdracht, verricht bij het IPO
in de periode sep 1985 t/m nov 1985
De opdracht werd verricht in de instrumentatiegroep
en werd begeleid door ing. J.G. Jonker

gezien en voor akkoord voor verzending naar school:
ing. J.M. den Hertog 5-12-85

ing. J.G. Jonker 5-12 '85

SAMENVATTING

Dit verslag behandelt de bouw en programmering van een LABBUS systeem voor een IEEE-488 naar MEA 8000 interface met bufferwerking.

Een deel van dit verslag is aan IEEE-488 gewijd omdat het omgaan met deze interface een gedegen kennis vereist.

Verder wordt kort het principe van de MEA 8000 spraaksynthesizer besproken.

Tenslotte volgt het programma dat een buffer, de IEEE-488 interface en een zelf ontwikkelde MEA 8000 interface laat samenwerken conform opdracht
Het programma is tijd-kritisch (MEA 8000 mag niet met horten en stoten spraak genereren) en is daarom in assembler (6809) geschreven.

INLEIDING

De afstudeeropdracht die in dit verslag wordt uitgewerkt luidt als volgt:

> "Ontwerp van een IEEE-488 interface voor de MEA 8000 spraakchip. Het interface omvat de verbinding met de bus, een klein intern buffergeheugen en de spraaksynthesechip inclusief filter en versterker."

Het IEEE-488 interface systeem is zeer uitgebreid.
Het doorgronden van dit systeem vereiste veel studie; het resultaat hiervan vormt dan ook een apart hoofdstuk.

De opdracht kan op twee manieren worden uitgewerkt:

1) zuivere hardware realisatie
2) opbouw uit bestaande en te ontwikkelen LABBUS microcomputer modules

Nadeel van methode 1is dat de kennis van alle te gebruiken systemen cq. componenten aanwezig dient te zijn voor dat er zinvol ontworpen kan worden.
Bovendien maakt het gebrek aan flexibiliteit eventueel noodzakelijke verbeteringen soms onmogelijk.
Daarom is voor methode 2 gekozen.
Door de programmeerbaarheid van het systeem is het dan ook eenvoudiger de modules afzonderlijk te testen.

Na de bouw van een LABBUS systeem waarin een eigen LABBUS MEA 8000 interface is opgenomen is het programma ontwikkeld dat de afzonderlijke modules samenvoegt tot een systeem conform de opdracht.
Dit programma bedient een tijd-kritische applicatie en is daarom geschreven in assembler (6809).

VOORWOORD

In het kader van mijn afstuderen aan de HTS Elektrotechniek ben ik in de gelegenheid gesteld een opdracht te verrichten binnen de instrumentatiegroep van het Instituut voor Perceptie Onderzoek (IPO) te Eindhoven.
Vanaf deze plaats zou ik daarvoor mijn dank willen uitspreken.

Met name bedank ik ing. J.G. Jonker voor de intensieve begeleiding die ik gedurende deze periode heb ontvangen.
Ook bedank ik ir. L.F. Willems voor het mij verstrekken van de afstudeeropdracht en voor mijn opname in de instrumentatiegroep gedurende het verrichten van de opdracht.
Bovendien wens ik allen hier niet genoemd te bedanken voor het mij verstrekken van adviezen en faciliteiten die nuttig bleken om mijn opdracht tot een goed einde te brengen.

Eelco de Koning

Eindhoven, 5 december 1985

1 IEEE-488 INTERFACE

1.1 ontstaan

Met de toegenomen beschikbaarheid van computers om gegevens te analyseren nam ook het belang om routinematige metingen te automatiseren (bijvoorbeeld om de betrouwbaarheid te verhogen) sterk toe.
Na een open vraag om suggesties voor een flexibele en algemeen toepasbare instrumentatiebus werkte het $IEEE_1$ een voorstel uit van Hewlett-Packard (1975: Digital Interface for Programmable Instruments).
De opgedane ervaringen leidden in 1978 tot een nieuwe specificatie.
Deze werd ook door het $ANSI_2$ goedgekeurd en staat sindsdien bekend als "ANSI IEEE-488 1978" of kortweg IEEE-488.
De IEC_3 nam de standaard over als "IEC-625"$_4$.
Verder werd de standaard bekend als $GPIB_5$ en natuurlijk als $HP\text{-}IB_6$.

De grote waarde van de standaard ligt besloten in zijn werkelijke praktische uniformiteit en het grote aantal instrumenten dat kan worden aangesloten.
Hieraan wil het andere standaards (bijvoorbeeld RS 232) nogal eens ontbreken.

1: Institute of Electrical and Electronic Engineering (U.S.A.)
2: American National Standards Institute
3: International Electrotechnical Commission
4: IEC-625-1 Standard interface systems for programmable measuring equipment
(IEC-625-2 Byte serial / bit parallel interface systems)
5: General Purpose Interface Bus
6: Hewlett-Packard Interface Bus

1.2 inleiding

IEEE-488 beschrijft een compleet interface systeem met hardware en protocollen.
Door het geven van algemene kenmerken probeert dit hoofdstuk inzicht te bieden in de principes van het IEEE-488 interface systeem.
Later worden voor dit project relevante zaken nader uitgewerkt.
In de bijlagen is een beknopt IEEE-488 overzicht opgenomen.

1.3 busstructuur

IEEE-488 gebruikt een parallelle bus met aktief lage signalen in wired-OR configuratie.
De bus omvat zestien signaallijnen in drie groepen:

groep		lijn	naam
	data	DIO1-DIO8	Data
control	handshake	DAV	Data Valid
		NRFD	Not Ready For Data
		NDAC	Not Data Accepted
	interface message	ATN	Attention
		REN	Remote Enable
		EOI	End Or Identify
		SRQ	Service Request
		IFC	Interface Clear

Instrumenten worden doorverbonden door middel van kabels met aan weerszijden "piggy-back" connectors.

1.4 systeem concept

In het algemeen bestaat een IEEE-488 interface systeem uit controllers (bestuurders, zenders van adressen en commando's), uit talkers (zenders van data) en uit listeners (ontvangers van data).
Elk instrument beschikt over een eigen IEEE-488 interface met een of meer functies.

Op elk moment treedt hoogstens één instrument op als controller.
Meestal is dit een vast instrument; vanwege zijn programmeerbaarheid is dit meestal een computer.
De controller bepaalt welk instrument als talker en welke instrumenten als listener optreden.
Op elk moment is er hoogstens één talker; er kunnen meerdere listeners tegelijk zijn.
Een drielijns handshake protocol garandeert twee belangrijke IEEE-488 eigenschappen:

1) elke ontvanger ontvangt alles wat de zender verstuurt
2) instrumenten met verschillende nominale datasnelheden kunnen zonder aanpassing met elkaar communiceren.

1.5 adressen / commando's

Elk instrument beschikt over een uniek 5-bit bus adres.
Er zijn 31 bus adressen beschikbaar.
In de data mode, nl als de lijn ATN=false, kunnen de instrumenten data bytes (DAB) uitwisselen over de datalijnen.
De controller brengt de bus in de command mode als hij ATN=true maakt.
Dan kan de controller bus-commando's, te weten adressen, universele en geadresseerde commando's over de datalijnen versturen (8-bit codes)

Adressen dienen om instrumenten listener of talker te maken.

- Er zijn 31 listen adressen (LAD).
  Ontvangt een instrument zijn LAD, dan wordt zijn listener-functie (indien aanwezig) ingeschakeld.
  Het adres unlisten (UNL) schakelt alle listener-functies uit.
- Er zijn 31 talk adressen (TAD).
  Ontvangt een instrument zijn TAD, dan wordt zijn talker-functie (indien aanwezig) ingeschakeld.
  Het adres untalk (UNT) en elk ander TAD schakelt zijn talker-functie weer uit.

Universele commando's zorgen dat ieder instrument op de bus een bepaalde interface operatie uitvoert.
Geadresseerde commando's hebben dit effect alleen op geadresseerde instrumenten.

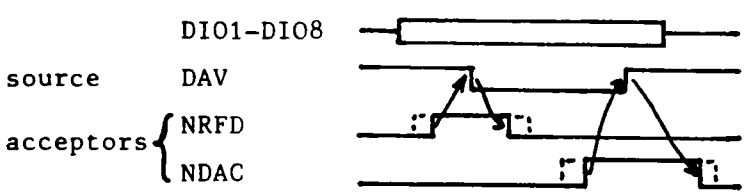
1.6 handshake

Twee functies, source handshake (SH) en acceptor handshake (AH), zorgen voor respectievelijk verzenden en ontvangen van bytes via de datalijnen.
Op elk moment kan hoogstens een source (talker of controller) bytes (respectievelijk data of bus-commando's) naar een of meer acceptors versturen.
De overdracht geschiedt per byte onder besturing van de handshake.
De handshake procedure garandeert dat iedere acceptor elk door de source verzonden byte inleest.
Deze procedure is asynchroon, waardoor instrumenten met verschillende datasnelheden via de IEEE-488 interface bus altijd kunnen communiceren.

De functies van de handshake lijnen zijn als volgt:

- De source maakt de lijn DAV=true om aan te geven dat het byte op de datalijnen geldig is en gelezen mag worden.
- Een acceptor maakt zijn NRFD=false om aan te geven dat hij gereed is voor een nieuw byte.
  Door de wired-OR configuratie wordt de lijn NRFD=false indien iedere acceptor gereed is voor een nieuw byte.
- Een acceptor maakt zijn NDAC=false om aan te geven dat hij het byte ontvangen heeft.
  Door de wired-OR configuratie wordt de lijn NDAC=false indien iedere acceptor het byte ontvangen heeft.

Onderstaande figuur is het timing diagram voor een byte overdracht:



N.B. aktief laag systeem: false = hoog
true = laag

1.7 polling

Door de aard van sommige taken die de instrumenten uitvoeren is niet altijd goed bekend wanneer zij service behoeven.
Gedurende die tijd zou de controller andere taken kunnen uitvoeren.
Er zijn twee mechanismen om die instrumenten pas weer te bedienen wanneer dat nodig is.

- Service request / serial polling:
  Dit is een interrupt mechanisme met terugmelding per instrument van een status byte STB (data).
  Een instrument dient een service request in door de lijn SRQ=true te maken.
  De controller kan daarop reageren met een serial poll procedure:
  1) UNL schakel alle listeners uit.
  2) LAD het instrument dat de gevraagde service moet verlenen (gewoonlijk de actieve controller zelf).
  3) SPE universeel commando serial poll enable.
     Instrumenten die een service request kunnen indienen weten nu dat de controller hun STB zal opvragen.
  4) TAD instrument dat service request kon indienen.
  5) De talker levert in de data mode zijn STB.
     Als dit instrument een service request indiende maakt het bit 7 van STB true (RQS, requested service).
     De andere bits (1 t/m 6 en 8) kunnen gebruikt worden om andere informatie over te dragen aan de listener.

  De stappen 4 en 5 worden voor iedere mogelijke indiener van de service request herhaald.

  6) SPD universeel commando serial poll disable.
     De procedure is beëindigd.

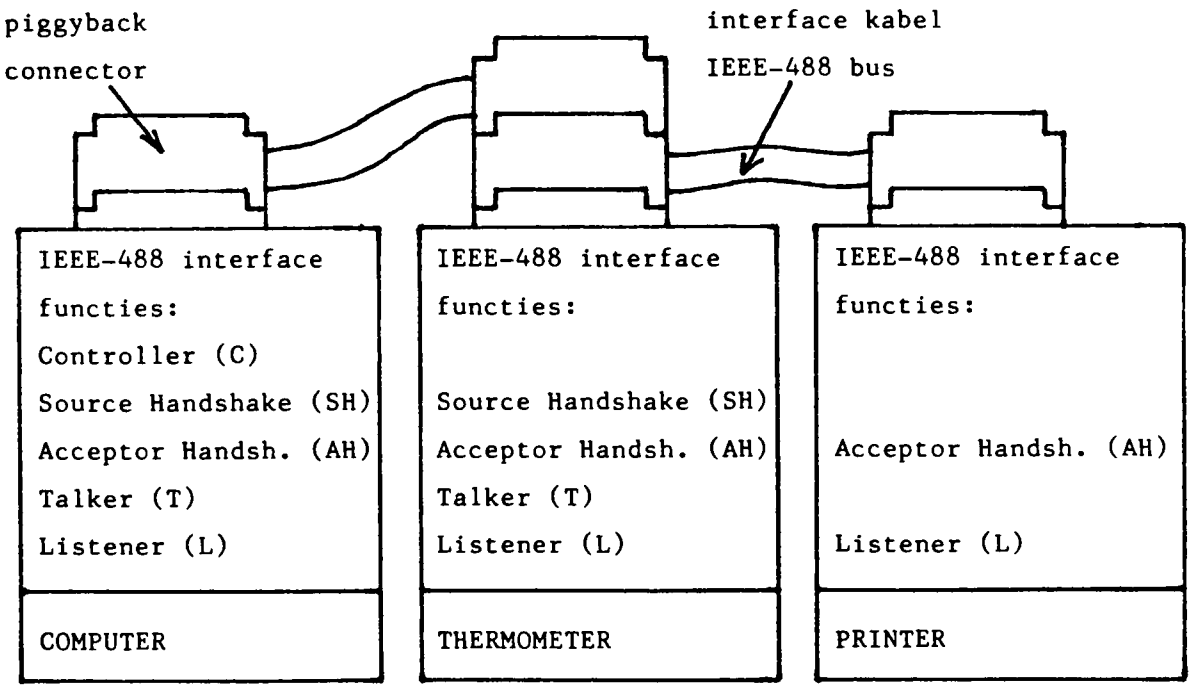
  De indiener van de service request weet dat de controller kan reageren.
  De conroller kan de passende aktie uitvoeren.

- Parallel polling:
  Bij dit mechanisme controleert (pollt) de controller regelmatig de status van maximaal acht instrumenten.
  De deelnemende instrumenten moeten voor parallel polling ieder apart geïnitialiseerd worden.
  Wanneer de controller in de command mode de lijn EOI=true maakt (Identify, IDY), dan geven de instrumenten ieder op de hun toegewezen datalijn een status bit af.

Voordeel van de parallel poll procedure is de veel grotere snelheid die verkregen kan worden.
Nadeel is dat de informatie slechts uit één bit bestaat en dat een uitgebreide initialisatie nodig is.

1.8 voorbeeld

Ter illustratie volgt hier een voorbeeld van een meetopstelling.



systeem besturen (C)
data (T): schaal thermo data (T): temperatuur
tekst printer
data (L): temperatuur data (L): schaal thermo data (L): tekst

2 LABBUS IEEE-488 BUS INTERFACE KAART

2.1 inleiding

Deze kaart vormt een complete IEEE-488 interface voor het LABBUS microcomputer ontwikkelsysteem.
Een MC68488 (General Purpose Interface Adapter GPIA) realiseert de meest voorkomende interface functies.
Voor programmering is dit IC uitgerust met 8 leesregisters (RØR t/m R7R) en 7 schrijfregisters (RØW, R2W t/m R7W) vanuit de CPU gezien.
Om de kaart ook als controller te kunnen gebruiken is middels extra logica een schrijfregister R1W toegevoegd dat ik Control register noem.
Vijf MC3448A IEEE-488 bus transceivers koppelen de GPIA en R1W aan de IEEE-488 interface bus.
Om time-out faciliteiten aan de LABBUS zijde mogelijk te maken (naar inzicht van de gebruiker is een MC6840 (Programmable Timer Module PTM) toegevoegd.
N.B. in dit hoofdstuk wordt vervolgens kortweg van "bus" gesproken waar IEEE-488 interface bus bedoeld wordt.

2.2 LABBUS aspecten

De GPIA (met het Control register erbij) en de PTM beslaan ieder acht adressen in LABBUS.
Een jumper in jumperveld W1 op de kaart maakt drie locaties voor deze adressen in het LABBUS VPA-gebied mogelijk.
In mijn toepassing is gekozen voor de middelste instelling.
Daardoor beslaan RØR/W t/m R7R/W respectievelijk de adressen $FF34 t/m $FF3B.
Overeenkomstige lees- en schrijfregisters delen hetzelfde adres.

De PTM en het Control register worden in mijn toepassing niet gebruikt.
Zowel de GPIA, het Control register als de PTM beschikken over interruptfaciliteiten naar LABBUS.
In mijn toepassing wordt hiervan geen gebruik gemaakt.
Na een LABBUS reset zijn de interrupts van de GPIA, van het Control register en van de PTM geblokkeerd, zodat geen extra software vereist is om deze interrupts te blokkeren.

Alle overgebleven jumpers dienen voor doorverbindingen van/naar/in de PTM of voor het genereren van interrupts.
In mijn toepassing worden deze jumpers niet geplaatst.

2.3 IEEE-488 interface aspecten

Niet alle faciliteiten van de GPIA zijn nodig om het gewenste interface gedrag te bewerkstelligen.
Daarom worden alleen de in mijn toepassing gebruikte registers en hun bit(s) in deze paragraaf besproken.
Bij elk register/bit worden de symbolische naam en het LABBUS adres vermeld zoals in mijn programma gekozen.
Een registernaam begint daarin met IB (interface bus); een bitnaam eindigt op MSK (mask).

2.3.1 initialisatie

R3W Auxiliary Command register (IBAXCM, $FF37):
- RESET;
  Dit bit wordt geset door de CPU (RESMSK) of door een LABBUS reset.
  Is het bit geset, dan:
    1) worden alle interrupts (Interrupt Status register) gewist
    2) worden alle interface-functies niet-aktief gemaakt
    3) kan de CPU alleen het Address register laden.
  Een LABBUS reset wist bovendien:
    1) de interrupt maskers (Interrupt Mask register en Control register) zodat er geen interrupts gegenereerd kunnen worden
    2) de IEEE-488 interface status bits
    3) het Address, het Address Mode en het Control register.
  Het bit kan worden gereset door de CPU.

R4R Address Switch register (IBADSW, $FF38):
- De kaart bevat een blokje met acht DIP switches sw1 t/m sw8 (S1).
  Via dit register worden de lijn SRQ en sw2 t/m sw8 gelezen (respectievelijk MSB t/m LSB).
  De CPU dient de inhoud van dit register gedeeltelijk naar het Address register te copiëren (zie aldaar).

R4W Address register (IBADRS, $FF38):

- bus adres (AD5 t/m AD1);
  Deze bits worden geladen met respectievelijk sw4 t/m sw8 en vormen het 5-bit bus adres.
  Na een LABBUS reset moet de CPU het bus adres naar dit register laden alvorens RESET=0 te maken om adressering van de GPIA vanuit de bus mogelijk te maken.
  In mijn toepassing wordt het bus adres "3" gekozen door sw4 t/m sw8 = 00011 in te stellen.
- least significant bit enable (lsbe);
  Als lsbe=1 dan wordt AD1, het laagste bit in het bus adres, een don't care bij adressering vanuit de bus.
  Zo krijgt de kaart effectief twee bus adressen (dual primary addressing).
  In mijn toepassing wordt deze mogelijkheid niet gewenst.
  Daarom wordt dit bit gemaskeerd (DDAMSK, disable dual addressing) alvorens dit register te laden.
- disable listener (dal), disable talker (dat);
  Bij het copiëren worden deze bits gelijk aan sw2 respectievelijk sw3 (UD2 respectievelijk UD1, user definable bits).
  In mijn toepassing worden zowel de listener- (voor het ontvangen van spraakfiles; data) als de talker-functie (voor het verzenden van STB na een service request) gebruikt.
  Daarom wordt sw2 sw3=0 0 ingesteld.

Line driver configuratie (sw1):

- Als sw1=1 dan vindt pull-up van de busdriver uitgangen plaats.
  In mijn toepassing wordt deze stand gekozen (voordeel: afsluiting).
- Als sw1=0 dan worden de busdriver uitgangen open -collector.
  Dan is parallel polling mogelijk.

2.3.2 listener

R2R Address Status register (IBADST, $FF36):

- Listener Active State (LACS);
  Aan dit bit kan de CPU zien (MLAMSK, My Listen Address) of de CPU als listener geadresseerd is.

RØR Interrupt Status register (IBINST, $FF34):
- Byte in (BI);
  Aan dit bit kan de CPU zien (BIMSK) dat de GPIA een DAB van de bus in het Data In register heeft geplaatst.
  Leest de CPU het Data In register, dan wordt het bit gereset.
- END;
  Aan dit bit kan de CPU zien (LSBMSK, last byte) dat de talker in de data mode EOI=true gemaakt heeft.
  Dit doet de talker alvorens het laatste DAB te versturen.

R7R Data In register (IBDATA, $FF3B):
- De CPU leest uit dit register de DAB's die de GPIA als listener toegezonden krijgt.
  De GPIA beëindigt de acceptor handshake voor een DAB pas nadat dat DAB door de CPU uit dit register gelezen is.

2.3.3 service request

R5R/W Serial Poll register (IBSPOL, $FF39):
- request service (rsv);
  Als de CPU dit bit set (RSVMSK) maakt de GPIA de lijn SRQ=true
  Hierdoor wordt de serial poll procedure door de controller gestart.
- Status Bits (S8, S6 t/m S1);
  De overige zeven bits van dit register kunnen door de CPU worden geladen met status bits naar keuze.

Als de GPIA tijdens de hierop volgende serial poll procedure als talker wordt geadresseerd verzend de GPIA de complete registerinhoud in de data mode als STB.
(De listener cq controller vat rsv dan op als RQS, Requested Service).
- Service Request State (SRQS);
  Dit bit blijft geset zolang het STB nog niet naar de bus is verzonden.
  Het verzenden (niet het ontvangen) van het STB reset dit bit.

R1R Command Status register (IBCMST, $FF35):
- Serial Poll Active State (SPAS);
  Als de GPIA het commando Serial Poll Enable (SPE) ontvangt wordt dit bit geset om aan te geven dat de controller de serial poll procedure gestart heeft.
  Dit bit wordt na de serial poll procedure weer gereset als de

GPIA het commando Serial Poll Disable (SPD) ontvangt.
De CPU kan dit verifiëren (SPDMSK) om te weten te komen dat het STB ook ontvangen is.

De CPU dient nu rsv=0 te maken om een volgende service request mogelijk te maken.

3 MEA 8000 SPRAAKSYNTHESIZER

Evenals de meeste andere spraaksynthesizers simuleert de MEA 8000 het menselijk stemmechanisme.
Uitgangspunt is dat geluid, dat in de stembanden wordt opgewekt door een luchtstroom uit de longen, een aantal trilholtes passeert alvorens het lichaam als een stem te verlaten

3.1 klinkers

Indien de stembanden bijna gesloten staan zal de luchtdruk uit de longen ze telkens opendrukken, waarna een afnemende luchtstroom ze weer naar de gesloten toestand moet laten terug bewegen.
De luchtdruk zal dus een zaagtandfunctie van de tijd zijn.
Voor dit doel beschikt de MEA 8000 over een zaagtandgenerator.

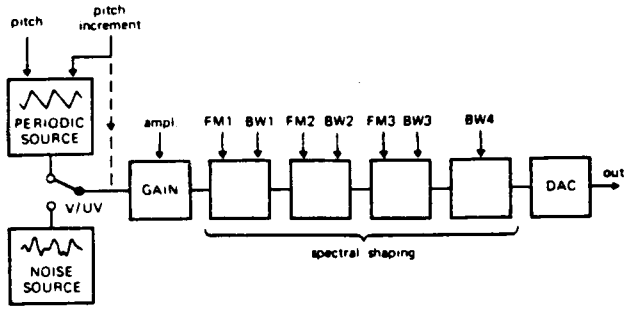
3.2 medeklinkers

Staan de stembanden een eindje open, dan wordt de luchtstroom niet afgesloten en zal de luchtdruk een stochastische functie van de tijd zijn.
Voor dit doel beschikt de MEA 8000 over een ruisbron.

3.3 formantspraak

De luchtstroom passeert trilholten: het strottehoofd, de keelholte en de mond- en neusholten.
De invloed van de vorm en afmeting van deze holten is te vertalen naar formant- (band-) filters.
Voor dit doel is de MEA 8000 uitgerust met vier bandfilters in cascade.
Van alle filters is de bandbreedte instelbaar.
Van de drie laagste filters is ook de resonantiefrequentie instelbaar.
Het hoogste filter heeft een vaste resonantie frequentie.

Onderstaande figuur schetst het principe van de formant synthesizer:

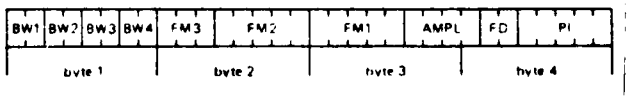


3.4 kwantisering

De synthesizerspraak wordt gemodelleerd naar menselijke spraak. Doel van de synthesizer is om met een lage bitrate spraak van redelijke kwaliteit te verkrijgen.
De spraakcodes voor de synthesizer worden als volgt verkregen:

1) neem menselijke spraak op.
2) analyseer de frequenties en bandbreedtes van de sterkste trillingen gedurende 8, 16, 32 of 64 ms.
3) kwantiseer de frequentiestijging (pitch increment) of kies ruis.

Het aldus verkregen spraakpakketje wordt een frame genoemd.
Een frame bevat 32 bit informatie in de volgende vorm:



code	bits	parameter
PI	5	pitch increment or noise selection
FD	2	speech frame duration
AMPL	4	amplitude
FM1	5	frequency of 1st formant
FM2	5	frequency of 2nd formant
FM3	3	frequency of 3rd formant
BW1	2	bandwidth of 1st formant
BW2	2	bandwidth of 2nd formant
BW3	2	bandwidth of 3rd formant
BW4	2	bandwidth of 4th formant

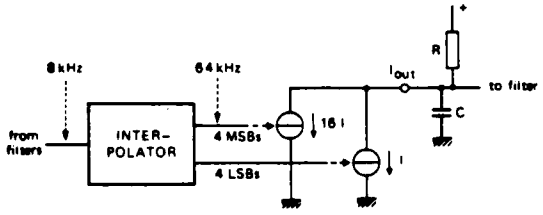
FM4, the frequency of the fourth formant, is fixed.

De synthesizer wordt gestart door het versturen van een pitch byte. Hierdoor wordt de hoogte van de stem geïnitialiseerd en kan de synthesizer zijn eerste frame gaan verwerken.

Het aldus verkregen codeerprincipe leidt tot een redelijke spraakkwaliteit. Weliswaar wordt de verkregen stem als niet-menselijk waargenomen, maar de gemiddelde informatie snelheid blijft beperkt tot zo'n 1000 bit/s (500 - 4000 bit/s).
Dit maakt het mogelijk allerlei intelligente apparatuur met spraak uit te rusten, zonder dat dit uitgebreide achtergrondgeheugens vereist.

3.5 digitalisering

Alle tot nu toe beschreven onderdelen van de synthesizer zijn digitaal uitgevoerd.
De formant synthesizer genereert met 8 kHz spraaksamples van 16 bit. Een interpolatie circuit berekent uit de 11 meest significante bits van de samples met 64 kHz acht 8-bit samples tussen twee samples die aan een DAC worden toegevoerd.
De DAC zelf bestaat uit twee stroombronnen I en 16*I die elk gedurende maximaal 15 pulsen van de klok ingeschakeld blijven, zie figuur:



Buiten de synthesizer moet nog een filter worden toegevoegd om een normaal audio signaal te verkrijgen uit de spanning die de DAC opwekt.

3.6 besturing

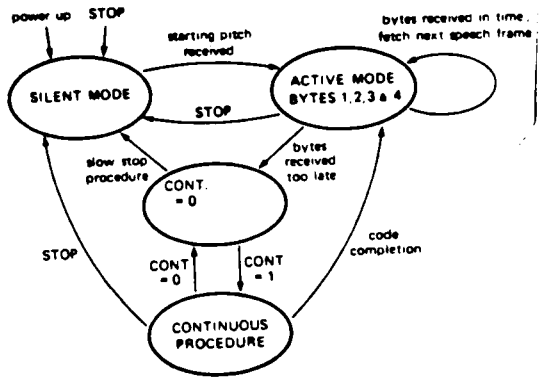
De nu volgende figuren geven de functie van het commando register van de synthesizer aan.
Indien $\overline{\text{REQ}}$ enabled wordt kan de synthesizer de CPU interrumperen ten teken dat hij data voor het volgende frame dient te ontvangen.

Indien CONT enabled wordt zal de synthesizer de laatst opgewekte klank niet laten uitsterven maar vasthouden indien het gewenste frame niet op tijd ontvangen wordt.
Het STOP commando legt de synthesizer meteen het zwijgen op.
Als volgende spraakcode wordt weer een pitch byte verwacht.

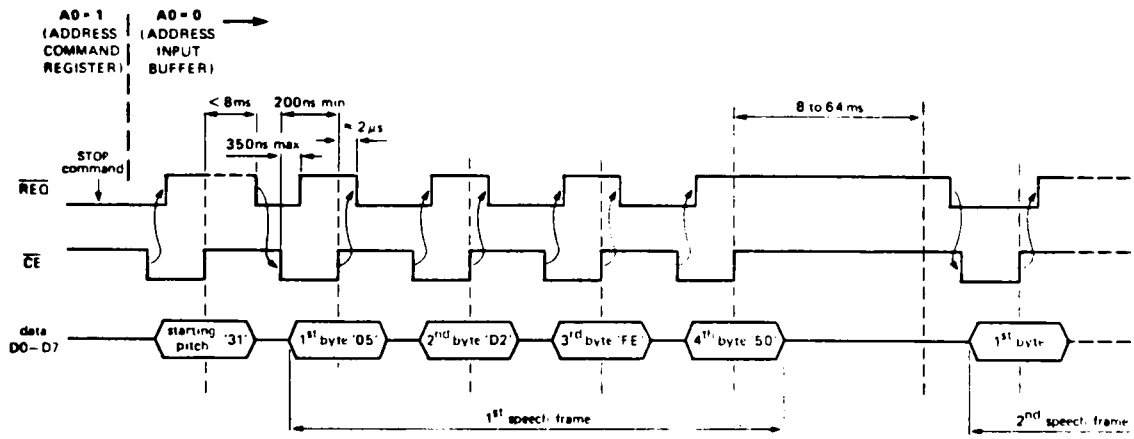
Command word bit allocation and truth table.

D4 STOP	D3 CONT enable	D2 CONT	D1 ROE enable	D0 ROE
0 = no action	0	0 = no action	0	0 = no action
1 = STOP	0	1 = no action	0	1 = no action
	1	0 = SLOW STOP procedure	1	0 = disable $\overline{REQ}$
	1	1 = CONTINUOUS procedure	1	1 = enable $\overline{REQ}$

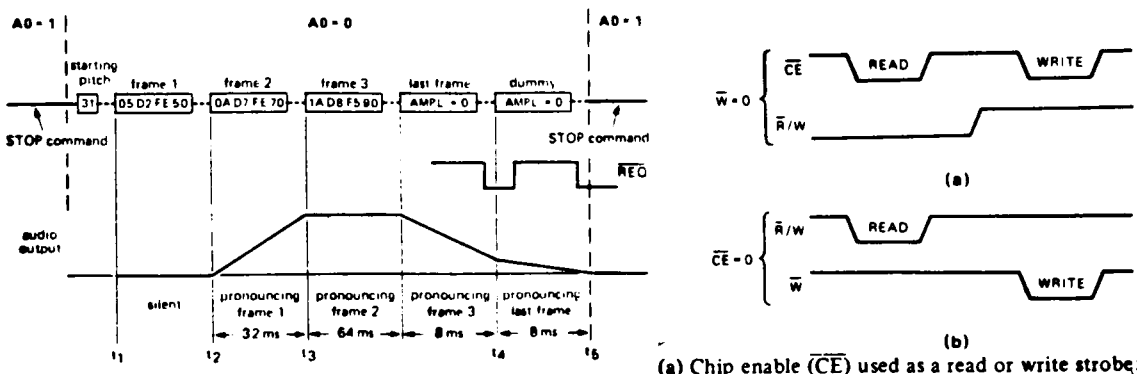
D7, D6 and D5 are not used;
ROE = request output enable.



Timing van de MEA 8000:



Timing of the first frame in detail. Data is written on the rising edge of $\overline{CE}$



(a) Chip enable ($\overline{CE}$) used as a read or write strobe;
(b) separate read and write strobes.

4 MEA 8000 INTERFACE VOOR LABBUS

Deze schakeling wordt in dit hoofdstuk kortweg aangeduid met "interface".

4.1 LABBBUS I/O

De MC6809 microprocessor (CPU) in het gebruikte LABBUS systeem maakt gebruik van memory-mapped I/O.
Voor de gebruiker vormen de adressen $FFØØ t/m $FF6F het "peripheral" gebied (programma: IOPAGE EQU $FF).
Ten bate van adressering in dit gebied definieert LABBUS speciale signalen om de implementatie van I/O modules te vereenvoudigen:

1) Het besturingssignaal $\overline{\text{VPA}}$ (Valid Peripheral Address) wordt aktief gemaakt.
2) Er worden I/O-blokken van vier adressen gecreëerd.
   De adreslijnen A2 t/m A9 worden gemodificeerd volgens een 2-uit-8 code.
   Elk I/O-blok wordt geselecteerd met twee enen (AX en AY) en zes nullen.
   Zo zijn er $\binom{8}{2}=28$ I/O-blokken.
   De adreslijnen A1 en AØ blijven ongewijzigd voor selectie binnen een blok.
   Het signaal $\overline{\text{WRT}}$ (write) configureert elk adres voor lezen en/of schrijven.

N.B. data wordt gelezen/geschreven op de dalende flank van DSTB (Data Strobe).

4.2 ontwerp (digitaal)

Registers:
De MEA 8000 spraaksynthesizer (IC6) bevat een lees- (MEASTS, status; gelezen wordt het enige bit REQ met REQMSK) en twee schrijfregisters (MEASPC, speech code input buffer; MEACMD, command register) en een Request uitgang ($\overline{\text{REQ}}$).
De tijd tussen twee frame (vier bytes elk) overdrachten van CPU naar synthesizer bedraagt ongeveer 8 (of 16, 32 of 64) ms.
In die tijd kan de CPU zo'n 1000 (of *2,*4 of *8) instructies uitvoeren.

Daarom wordt de synthesizer bediend op interrupt basis.
Het signaal $\overline{REQ}$ wordt binnen een frame na ontvangst van een spraak code byte naar verwachting binnen 3 µs weer aktief.
Gebruik van $\overline{NMI}$ (Non Maskable Interrupt; edge-triggered) zou resulteren in vier geneste interrupt services voor een frame.
Dit zou onnodig veel beslag op de stack en op de rekentijd leggen.
Daarom wordt $\overline{REQ}$ gebruikt om flip-flop RQ (IC4) te setten, die op zijn beurt per frame één interrupt zal genereren.
De CPU kan deze flip-flop weer resetten met een schrijfaktie (RESRQ).
Met jumperveld W2 kan de gewenste interrupt ($\overline{IRQ}$, normaal; $\overline{FIRQ}$, fast; $\overline{NMI}$. laagste respectievelijk hoogste prioriteit) worden gekozen.
In mijn toepassing wordt $\overline{FIRQ}$ ingesteld.

Interface selectie ($\overline{SEL}$):
Het selectiesignaal voor deze interface wordt gedefinieerd als SEL=VPA*AX*AY (IC4, IC5).
Dit signaal aktiveert de bus buffers (IC2) voor de datalijnen.
Middels jumperveld W1 zijn AX en AY vrij te kiezen uit A2 t/m A9.
De registers van deze interface passen binnen één I/O-blok.
In mijn toepassing begint dit I/O-blok op adres $FFØØ door AX*AY=A9*A2 in te stellen.

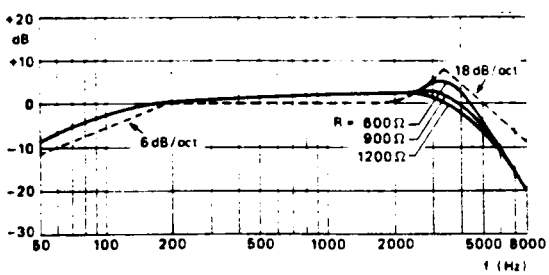
Register selectie:
De selectiesignalen voor de MEA 8000 ($\overline{CE}$) en de flip-flop RQ ($\overline{R}$) worden door een 3-bit demultiplexer (IC3) bepaald uit de signalen $\overline{WRT}$, A1 en AØ.
De toewijzing is als volgt:

$\overline{WRT}$	A1	AØ	uitgang	register	signaal
Ø	Ø	Ø	Ø	MEASPC	
Ø	Ø	1	1	MEACMD	IC6 $\overline{CE}$
1	Ø	Ø	4	MEASTS	
Ø	1	Ø	2	RESRQ	IC4 $\overline{R}$

Data overdracht:
Het signaal $\overline{WRT}$ bestuurt de richting van het datatransport door de bus buffers en de synthesizer.
De drie registers van de synthesizer gebruiken een gemeenschappelijk selectiesignaal $\overline{CE}$.
IC5 aktiveert dit signaal als een van de uitgangen Ø, 1 of 4 van de de demultiplexer aktief wordt.
De registers van deze interface gebruiken hun selectiesignalen tevens als lees- of schrijfpuls.
Daarom worden deze signalen gesynchroniseerd met DSTB door de enable-functie van de demultiplexer te definiëren als DSTB*SEL.

4.3 ontwerp (analoog)

Het uitgangscircuit van de synthesizer wordt gevormd door een 8-bit DAC.
De referentiestroom door de DAC (ter beveiliging van het uitgangs-circuit maximaal 100 µA) wordt bepaald door V(REF)=2,5 V, VDD=5 V en R6.
Gebruik van een DAC maakt een laagdoorlaatfilter noodzakelijk.
De 8 kHz samples uit de formant synthesizer worden in acht stappen lineair geïnterpoleerd, zodat de DAC met 64 kHz samples omzet.
Daardoor daalt de frequentiekarakteristiek van het signaal ($\sin^2 x/x^2$).
De filtering moet dus worden gecorrigeerd.
Aanbevolen wordt de volgende frequentiekarakteristiek:



Output filter transfer characteristic.
--- for optimum voice quality;
—— characteristic of the filter shown

De karakteristiek voor R=600 Ohm wordt goed benaderd door het toegepaste filter R1 C1 L1 R2.
C2 C3 vormt een wisselstroomkoppeling, R3 een volumeregeling.
De uitgangstrap wordt gevormd door IC7 R4 C4 C5.

5 BESTURINGSPROGRAMMA

5.1 inleiding

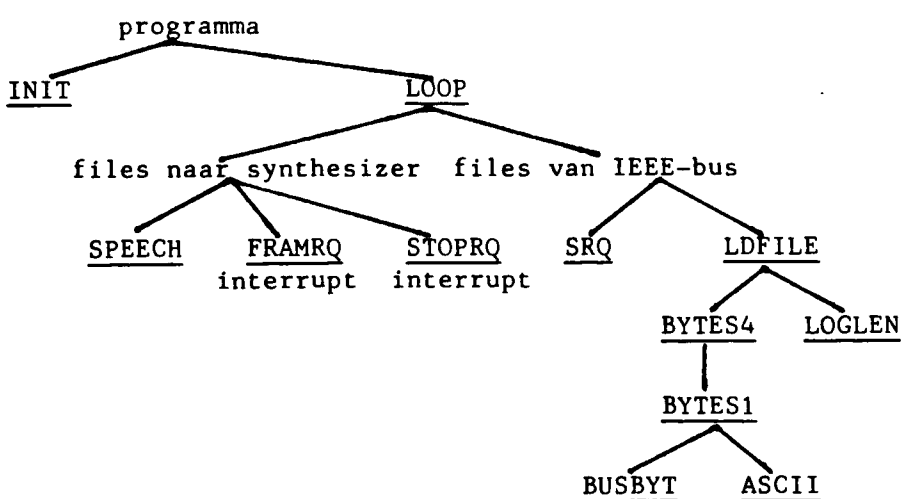
In dit stadium van het verslag zijn de hardware modules die in mijn toepassing gebruikt worden besproken.
Op een hoger plan staat het besturingsprogramma dat hun onderlinge samenwerking definieert.
De besturing van de spraaksynthesizer is tijd-kritisch.
Daarom is het programma in assembler (6809) geschreven.
Het geheel van hard- en software modules onder besturing van het programma vormt het "instrument" waarvan ik in dit hoofdstuk de werking beschrijf.

5.2 software modules

De opdracht valt uiteen in twee delen:

1) Laat de spraaksynthesizer spraak genereren uit codes die worden opgehaald uit een buffer.
2) Indien het buffer leeg dreigt te raken moet het instrument dit via een service request aan de IEEE-488 bus melden. Het instrument dient dan als listener spraakcodes te ontvangen om het buffer bij te vullen.

Voor het programma is de volgende top-down structuur op te stellen:
(onderstreepte termen zijn labels in het programma)



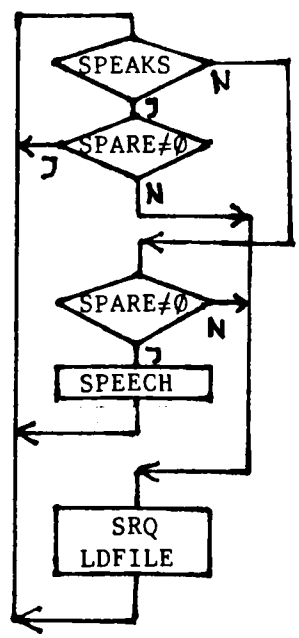
5.3 initialisatie INIT

De volgende instellingen worden vastgelegd na power-on reset:

- instellen IEEE-488 bus adres GPIA
- GPIA talker- en listener-functies niet blokkeren
- dual addressing blokkeren
- Y register wordt pointer voor codes naar synthesizer
- X register wordt pointer voor codes van IEEE-488 interface bus
- reset eventuele requests (RQ) van synthesizer interface
- SPEAKS:=false (synthesizer silent mode na power-on)
- SPARE:=Ø (geen files in voorraad)
- STB:=Ø (evenredig log(tijd dat synthesizer spreekt na start))
- initialiseer stackpointer
- vrijgave $\overline{\text{FIRQ}}$ (fast interrupt geeft MEA 8000 hoge prioriteit)

5.4 bewaking spraak/buffer LOOP

De programma LOOP bewaakt continu of de MEA 8000 nog wel spreekt en of er nog wel voldoende spraakdata is om de MEA 8000 aan de praat te houden.
Stopt de MEA 8000, dan wordt hij zo mogelijk van een nieuwe spraakfile voorzien.
Is er geen compleet file meer in voorraad, dan wordt eerst een service request gegenereerd (STB zie boven), waarna er op de komst van een spraakfile wordt gewacht voor de GPIA als listener.



LITERATUUR

Digital instrument course
part 4 IEC BUS INTERFACE
Philips Test and Measuring department

Technical publication 101
MEA 8000 voice synthesizer: principles and interfacing
Philips Elcoma

LABBUS 6809 single board computer / bus controller product £ 2001-2448-6
LABBUS systeembus
LABBUS IEEE-488 BUS INTERFACE product £ 2001-5943-3

6809 Assembly language programming
Lance A. Leventhal
OSBORNE/McGraw-Hill

De IEC/IEEE interface bus
Elektronica 81/13/14

Een IEC-bus-monitor
polytechnisch tijdschrift|elektrotechniek/elektronica (1981) nr. 10

Microprocessor data manual 1982
Motorola

News Release number: 9629E/May/85-087
Philips Elcoma (SPEECH ANALYSIS/EDITING SYSTEM USING THE IBM-PC)

Bijlage I: besturingsprogramma

```
IEEE-488<>MEA 8000          15-jan-00 12:15      RMA V1.16      page      1

                                NAM  IEEE-488<>MEA 8000
                           ****************************************
                           *           IEEE-488 INTERFACE         *
                           *                                      *
                           *                 FOR                  *
                           *                                      *
                           *      MEA 8000 SPEECH SYNTHESIZER     *
                           ****************************************
                           *     E.A. DE KONING   SEP-DEC 1985    *
                           ****************************************

                           * SPEECH FILES BUFFER
0000                       BOTTOM EQU  0
C000                       TOP    EQU  $C000       TOP OF EXTRA RAM CARD

                           * INTERRUPT MASK
00BF                       ENFIRQ EQU  %10111111   ENABLE FIRQ ACK

                           * ALL VPA ADDRESSES LOCATED ON PAGE $FF
00FF                       IOPAGE EQU  $FF

                           * MEA 8000  SPEECH SYNTHESIZER
                           * REGISTER ADDRESSES/MASKS
0000                       MEASTS EQU  0            STATUS (ROR)
0080                       REQMSK EQU  %10000000    READY CAUSES REQ

0000                       MEASPC EQU  MEASTS       SPEECH CODE (ROW)

0001                       MEACMD EQU  MEASTS+1     COMMAND (R1W)
0012                       COD1   EQU  %00010010    STOP, NO REQ
0013                       COD2   EQU  %00010011    STOP, REQ
1280                       COD3   EQU  256*COD1+REQMSK
001A                       COD4   EQU  %00011010    SLOW STOP, NO REQ
1A80                       COD5   EQU  256*COD4+REQMSK

                           * FLIP-FLOP RQ IS SET BY MEA REQ PORT
                           * AFFECTS FIRQ LINE
0002                       RESRQ  EQU  MEASTS+2     WRITE RESETS RQ (R2W)

                           * 68488  IEEE-488 INTERFACE
                           * REGISTER ADDRESSES/MASKS
0034                       IBINST EQU  $34          INTERRUPT STATUS (ROR)
0001                       BIMSK  EQU  %00000001    BYTE IN
0002                       LSBMSK EQU  %00000010    *END, LAST BYTE

0035                       IBCMST EQU  IBINST+1     COMMAND STATUS (R1R)
0004                       SPDMSK EQU  %00000100    *SERIAL POLL DISABLED

0036                       IBADST EQU  IBINST+2     ADDRESS STATUS (R2R)
0004                       MLAMSK EQU  %00000100    *MY LISTEN ADDRESS

0037                       IBAXCM EQU  IBINST+3     AUXILIARY COMMAND (R3W)
0080                       RESMSK EQU  %10000000    RESET

0038                       IBADSW EQU  IBINST+4     ADDRESS SWITCH (R4R)
0038                       IBADRS EQU  IBINST+4     ADDRESS (R4W)
```

```
               007F        DDAMSK EQU   %01111111  DISABLE DUAL ADDRESSING

               0039        IBSPOL EQU   IBINS1+5   SERIAL POLL (RSR/W)
               0040        RSVMSK EQU   %01000000  *REQUEST SERVICE
               0040        SRSMSK EQU   %01000000  *SERVICE REQUEST STATE

               003B        IBDATA EQU   IBINS1+7   DATA FROM IEEE-488 BUS

                           * POINTER AREA
     DFF0                         ORG   $DFF0      TOP OF RAM
     DFF0      0002        NFRAME RMB   2          NEXT FRAME
     DFF2      0002        NSPEAK RMB   2          NEXT SPEECH FILE
     DFF4      0002        START  RMB   2          INPUT FILE
     DFF6      0002        LFRAME RMB   2          LAST FRAME INPUT
     DFF8      0001        STB    RMB   1          *STATUS BYTE
     DFF9      0001        SPARE  RMB   1          FILES NOT IN USE
     DFFA      0001        SPEAKS RMB   1          FALSE IF MEA SILENT
     DFFB      0002        FIRVEC RMB   2          SERVICE MEA REQ


                           **************************************************
                           **************  P R O G R A M  ******************
                           **************************************************
     F000                         ORG   $F000      SYSTEM ROM
     F000 50 52 4F 50 45          FCC   /PROPERTY AND COPYRIGHT:/
     F017 49 4E 53 54 49          FCC   /INSTITUTE FOR PERCEPTION RESEARCH/
     F038 44 45 4E 20 44          FCC   /DEN DOLECH 2  EINDHOVEN NEDERLAND/
     F059 20 20 20 20 20          FCC   /     /
     F05E 44 49 53 43 4C          FCC   /DISCLOSURE TO THIRD PARTIES,/
     F07A 4F 52 20 52 45          FCC   /OR REPRODUCTION,/
     F08A 49 4E 20 41 4E          FCC   /IN ANY FORM WHATSOEVER/
     F0A0 57 49 54 48 4F          FCC   /WITHOUT WRITTEN CONSENT/
     F0B7 49 53 20 46 4F          FCC   /IS FORBIDDEN/
                           * INITIALISATION
     F0C3 86 FF            INIT   LDA   £IOPAGE    DIRECT PAGE
     F0C5 1F8B                    TFR   A,DP       SET DIRECT PAGE
                           * INSTALL LABBUS  IEEE-488 INTERFACE
     F0C7 86 80                   LDA   £RESMSK    RESET BUS INTERFACE
     F0C9 97 37                   STA   <IBAXCM    TO DISABLE BUS ACCES
     F0CB 96 38                   LDA   <IBADSW    READ SETTING
     F0CD 84 7F                   ANDA  £DDAMSK    NO DUAL ADDRESSING
     F0CF 97 38                   STA   <IBADRS    SET ADDRESS
                           * INSTALL LABBUS  MEA 8000 INTERFACE
     F0D1 CC 1280                 LDD   £COD3      STOP, NO REQ; REQMSK
     F0D4 D5 00            WTRES1 BITB  <MEASTS    READY FOR COMMAND ?
     F0D6 27 FC                   BEQ   WTRES1     NO, WAIT
     F0D8 97 01                   STA   <MEACMD
     F0DA D5 00            WTRES2 BITB  <MEASTS    READY FOR COMMAND ?
     F0DC 27 FC                   BEQ   WTRES2     NO, WAIT
     F0DE 97 01                   STA   <MEACMD    TWICE, MEA OLD VERSIONS
     F0E0 97 02                   STA   <RESRQ     REMOVE GHOST RQ MEA
                           * INSTALL PROGRAM VARIABLES
     F0E2 8E 0000                 LDX   £BOTTOM    POINTER IEEE-488 INPUT
     F0E5 108E 0000               LDY   £BOTTOM    POINTER OUTPUT MEA
     F0E9 7F DFF8                 CLR   STB        STATUS:=EMPTY
     F0EC 7F DFF9                 CLR   SPARE      NO FILES PRESENT
```

```
                        CLR   SPEAKS      MEA SILENT AFTER POWERON
                        LDS   #NFRAME     STACK BELOW VARIABLES
                        ANDCC #ENFIRQ     RELEASE FIRQ

                    * ACTUAL PROGRAM:
                    * CONTROL MEA AND BUFFER
F0EF 7F DFFA        LOOP    TST   SPEAKS      MEA SPEAKING ?
F0F2 10CE DFF0              BEQ   MEAOFF      NO, TRY TO RESTART
F0F6 1C BF                  TST   SPARE       FILES IN STOCK ?
F0F8 7D DFFA                BNE   LOOP        ALL OKAY
F0FB 27 0B          EMPTY   BSR   SRQ         NO FILES, ASK FOR FILE
F0FD 7D DFF9                BSR   LDFILE      FILL BUFFER FROM BUS
F100 26 F6                  BRA   LOOP        KEEP WATCHING STATUS
F102 8D 40          MEAOFF  TST   SPARE       FILES IN STOCK ?
F104 8D 56                  BEQ   EMPTY       NO FILES, ASK FOR FILE
F106 20 F0                  BSR   SPEECH      SPEAK NEW FILE
F108 7D DFF9                BRA   LOOP        KEEP WATCHING STATUS
F10B 27 F5
F10D 8D 02
F10F 20 E7

                    ************************************
                    * ROUTINE TO INITIATE MEA SPEAKING *
                    ************************************
                    * SPEECH  READ HEADER OF SPEECH FILE OUT NOW
                    * IN :Y=POINTER OUTPUT FILE
                    * OUT:Y=POINTER WHERE NEXT FILE EXPECTED
                    *      MEA STARTS REQUESTING
F111 CC F216        SPEECH  LDD   #FRAMRQ     POINTER RESPONSE
F114 FD DFFB                STD   FIRVEC      FOR MEA REQUESTS
F117 7F DFFA                CLR   SPEAKS
F11A 73 DFFA                COM   SPEAKS      :=TRUE
F11D 7A DFF9                DEC   SPARE       SYNTH OUTPUTS NEW FILE
F120 ECA1                   LDD   ,Y++        READ HEADER
F122 FD DFF2                STD   NSPEAK      POINTER NEXT FILE
F125 A6A0                   LDA   ,Y+         READ 2LOG(LENGTH)
F127 B7 DFF8                STA   STB         SHOW MAX SERVICE DELAY
F12A 3121                   LEAY  1,Y         IGNORE PITCH NOW
F12C 10BF DFF0              STY   NFRAME      POINTER FIRST FRAME
F130 313F                   LEAY  -1,Y        FIRST CODE = PITCH
F132 CC 1280                LDD   #COD3       STOP, NO REQ;REQMSK
F135 D5 00          WTCOD1  BITB  <MEASTS     READY FOR STOP,NO REQ ?
F137 27 FC                  BEQ   WTCOD1      NO, WAIT
F139 97 01                  STA   <MEACMD     COMMANDING TWICE FOR
F13B 86 13                  LDA   #COD2       MEA OLD VERSIONS
F13D D5 00          WTCOD2  BITB  <MEASTS     READY FOR STOP, REQ ?
F13F 27 FC                  BEQ   WTCOD2      NO, WAIT
F141 97 01                  STA   <MEACMD     MEA OLD VERSIONS
F143 39                     RTS               REQUESTS CAUSE FIRQ

                    ************************************
                    * ROUTINES FOR OBTAINING FILES *
                    ************************************
                    * SRQ  GENERATES SERVICE REQUEST
                    * UNTIL SERIAL POLL COMPLETED
                    * EXPECTS CONTROLLER ACTIONS:
                    *      1)*SPE
                    *      2)MY *TAD
                    *      3)*SPD
```

```
IEEE-488<>MEA 8000          15-jan-00 12:15      RMA V1.16      page      4

                        * IN :STB (2LOG(LENGTH FILE OUT NOW)
                        * OUT:*SRQ=TRUE UNTIL POLLED
                        *      GPIA ACCESSIBLE BY BUS
F144 0F 37              SRQ     CLR  <IBAXCM     RELEASE BUS ACCES
F146 F6 DFF8                    LDB  STB         MAX DELAY CONT SPEECH
F149 CA 40                      ORB  £RSVMSK     *REQUEST SERVICE
F14B D7 39                      STB  <IBSPOL     FORCE *SRQ=TRUE
F14D C6 40                      LDB  £SRSMSK     PREPARE TEST *STB SENT
F14F D5 39              WTSENT  BITB <IBSPOL     *SERVICE REQUEST STATE
F151 26 FC                      BNE  WTSENT      YES, WAIT FOR *STB SENT
F153 C6 04                      LDB  £SPDMSK     PREPARE TEST *SPD IN
F155 D5 35              WTSPD   BITB <IBCMST     *SERIAL POLL ACTIVE ?
F157 26 FC                      BNE  WTSPD       YES, WAIT FOR *SPD IN
F159 0F 39                      CLR  <IBSPOL     POLLED;ENABLE NEXT *SRQ
F15B 39                         RTS

                        * LDFILE  WAITS UNTIL GPIA IS LISTENER
                        *         THEN LOADS SPEECH FILE FROM IEEE BUS
                        *         ASCII FILE IN CONVERTED TO HEX FILE
                        *         AFTER LOAD GPIA NOT ACCESSIBLE BY BUS
                        * IN :X=STORAGE ADDRESS
                        *     FILE FORMAT EXPECTED:
                        *      HEADER=LENGTH (HIGH; BYTES INCL HEADER)
                        *             LENGTH (LOW)
                        *             DONT CARE
                        *             PITCH
                        *      FRAMES (4 BYTES EACH, AT LEAST 1 FRAME)
                        *       (*END PRIOR TO RECEPTION LAST ASCII BYTE)
                        * OUT:X=NEXT STORAGE ADDRESS
                        *     LFRAME=ADDRESS LAST FRAME LOADED
                        *     HEX FILE:
                        *      HEADER=ADDRESS NEXT FILE (HIGH)
                        *             ADDRESS NEXT FILE (LOW)
                        *             INT(2LOG(LENGTH)) (FOR *STB)
                        *             PITCH
                        *      FRAMES UNCHANGED
F15C C6 04              LDFILE  LDB  £MLAMSK     PREPARE TEST *MY LAD
F15E D5 36              WTLACS  BITB <IBADST     GPIA=LISTENER ?
F160 27 FC                      BEQ  WTLACS      NO, WAIT UNTIL SERVICE
F162 C6 01                      LDB  £RIMSK      PREPARE TEST BYTE IN
F164 BF DFF4                    STX  START       KEEP POINTER HEADER
F167 8D 5F                      BSR  BYTES4      GET FILE HEADER
F169 34 10                      PSHS X           KEEP POINTER FIRST FRAME
F16B EC9F DFF4                  LDD  ←START→     GET LENGTH OF FILE
F16F 83 0004                    SUBD £4          WITHOUT LAST FRAME
F172 F3 DFF4                    ADDD START       POINTER LAST FRAME
F175 25 06                      BCS  ADRERR      OVERRANGE ?
F177 1083 C000                  CMPD £TOP        BEYOND BUFFER ?
F17B 25 06                      BLO  ADRCOR      NO, ADDRESS CORRECT
F17D 83 C000            ADRERR  SUBD £TOP        MODULO BUFFER;
F180 C3 0000                    ADDD £BOTTOM     -(TOP-BOTTOM)
F183 FD DFF6            ADRCOR  STD  LFRAME      POINTER LAST FRAME
F186 EC9F DFF4                  LDD  ←START→     GET LENGTH OF FILE
F18A BE DFF4                    LDX  START       CHANGE IN HEADER
F18D 8D 71                      BSR  LOGLEN      DETERMINE 2LOG
F18F A702                       STA  2,X         REPLACES DONT CARE
```

```
225 F191 35 10          PULS X          CURRENT POINTER
226 F193 C6 01          LDB  £BIMSK     PREPARE TEST BYTE IN
227 F195 BC DFF6 LDFRAM CMPX LFRAME     REACHED LAST FRAME ?
228 F198 27 04          BEQ  LASTFR     YES, LOAD IT
229 F19A 8D 2C          BSR  BYTES4     LOAD 1 NORMAL FRAME
230 F19C 20 F7          BRA  LDFRAM     KEEP LOADING
231 F19E 8D 2A LASTFR   BSR  BYTES3     GET 3 PRELAST BYTES
232 F1A0 8D 4B          BSR  BUSBYT     GET PRELAST BUSBYTE
233 F1A2 8D 50          BSR  ASCII      IS ASCII; CONVERT
234 F1A4 48             ASLA            MAKE IT HIGH NIBBLE
235 F1A5 48             ASLA            (*16)
236 F1A6 48             ASLA
237 F1A7 48             ASLA
238 F1A8 A7E2           STA  ,-S        KEEP IT FOR LAST
239 F1AA C6 02          LDB  £LSBMSK    PREPARE TEST *END
240 F1AC D5 34 WTEND    BITB <IBINST    *LAST BYTE ?
241 F1AE 27 FC          BEQ  WTEND      NO, WAIT
242 F1B0 C6 01          LDB  £BIMSK     PREPARE TEST BYTE IN
243 F1B2 8D 39          BSR  BUSBYT     GET LAST BUSBYT
244 F1B4 8D 3E          BSR  ASCII      IS ASCII; CONVERT
245 F1B6 AAE0           ORA  ,S+        APPEND PREVIOUS HEX
246 F1B8 A780           STA  ,X+        TRAILER INPUT FILE
247 F1BA 86 80          LDA  £RESMSK    RESET BUS INTERFACE
248 F1BC 97 37          STA  <IBAXCM    TO DISABLE BUS ACCESS
249 F1BE 8D 10          BSR  CHKTOP     ADJUST IF REACHED TOP
250 F1C0 AF9F DFF4      STX  ←START→    NSPEAK TO HEADER
251 F1C4 7C DFF9        INC  SPARE      JUST OBTAINED 1 FILE
252 F1C7 39             RTS
253
254                     ********************************
255                     * LOW LEVEL LOADER ROUTINES *
256                     ********************************
257                     * BYTES4  LOADS 8 ASCII BYTES FROM IEEE-BUS
258                     *         STORES 4 DERIVED HEX BYTES
259                     * IN :8 ASCII BYTES FROM GPIA=LISTENER
260                     *      X=STORAGE ADDRESS IN BUFFER
261                     *      B=BIMSK
262                     * OUT:4 HEX BYTES TO BUFFER
263                     *      X=NEXT STORAGE ADDRESS
264 F1C8 8D 10 BYTES4 BSR  BYTES1      GET 4 HEX BYTES
265 F1CA 8D 0E BYTES3 BSR  BYTES1
266 F1CC 8D 0C         BSR  BYTES1
267 F1CE 8D 0A         BSR  BYTES1
268 F1D0 8C C000 CHKTOP CMPX £TOP      EXCEEDED BUFFER ?
269 F1D3 27 01          BEQ  BFRERR
270 F1D5 39             RTS
271 F1D6 8E 0000 BFRERR LDX  £BOTTOM   YES, CONTINUE THERE
272 F1D9 39             RTS
273
274                     * BYTES1  LOADS 2 ASCII BYTES FROM IEEE-BUS
275                     *         STORES 1 DERIVED HEX BYTE
276                     * IN :2 ASCII BYTES FROM GPIA=LISTENER
277                     *      X=STORAGE ADDRESS IN BUFFER
278                     *      B=BIMSK
279                     * OUT:1 HEX BYTE TO BUFFER
280                     *      X:=X+1
```

```
281 F1DA 8D 11          BYTES1 BSR   BUSBYT      1ST BYTE FROM IEEE
282 F1DC 8D 16                 BSR   ASCII       CONVERT TO NIBBLE
283 F1DE 48                    ASLA              MAKE IT HIGH NIBBLE
284 F1DF 48                    ASLA              (*16)
285 F1E0 48                    ASLA
286 F1E1 48                    ASLA
287 F1E2 A7E2                  STA   ,-S         KEEP IT
288 F1E4 8D 07                 BSR   BUSBYT      2ND BYTE FROM IEEE
289 F1E6 8D 0C                 BSR   ASCII       CONVERT TO NIBBLE
290 F1E8 AAE0                  ORA   ,S+         APPEND PREVIOUS
291 F1EA A780                  STA   ,X+         HEX BYTE TO BUFFER
292 F1EC 39                    RTS
293
294                     * BUSBYT  LOADS 1 BYTE FROM IEEE-BUS
295                     * IN :B=BIMSK
296                     * OUT:A=BYTE READ
297 F1ED D5 34          BUSBYT BITB <IBINST     BYTE IN ?
298 F1EF 27 FC                 BEQ   BUSBYT      NO, WAIT
299 F1F1 96 3R                 LDA   <IBDATA     GET BYTE FROM GPIA
300 F1F3 39                    RTS
301
302                     * ASCII  CONVERTS ASCII CODE(HEX) TO NIBBLE
303                     * IN :A=CODE FOR 0-9 OR A-F
304                     * OUT:A=NIBBLE (0000NNNN)
305 F1F4 84 7F          ASCII  ANDA £%01111111 STRIP OFF PARITY BIT
306 F1F6 81 40                 CMPA £$40       SEPERATION DIGIT/CHAR
307 F1F8 22 03                 BHI   CHAR
308 F1FA 80 30          DIGIT  SUBA £48         -CODE(0)
309 F1FC 39                    RTS
310 F1FD 80 37          CHAR   SUBA £55         -(CODE(A)-10)
311 F1FF 39                    RTS
312
313                     * LOGLEN  CALCULATES 2LOG(LENGTH INPUTFILE)
314                     * =POSITION HIGHEST 1 IN 16 BIT LENGTH
315                     * IN :D=LENGTH (=>B)
316                     * OUT:A=LOG (3<=RESULT<=15)
317 F200 4D             LOGLEN TSTA             LENGTH(HIGH)=0 ?
318 F201 27 08                 BEQ   SMALL       YES, SKIP A
319 F203 C6 0F                 LDB   £15         MAX RESULT
320 F205 48             LOGA   ASLA             FOUND HIGHEST 1 IN A ?
321 F206 25 0D                 BCS   FOUNDA      YES, STOP LOOKING
322 F208 5A                    DECB              RESULT:=RESULT-1
323 F209 20 FA                 BRA   LOGA        RETRY
324 F20B 86 07          SMALL  LDA   £7          MAX RESULT
325 F20D 58             LOGB   ASLB             FOUND HIGHEST 1 IN B ?
326 F20E 25 03                 BCS   FOUNDB      YES, STOP LOOKING
327 F210 4A                    DECA              RESULT:=RESULT-1
328 F211 20 FA                 BRA   LOGB        RETRY
329 F213 1F98           FOUNDB TFR   B,A         A:=RESULT
330 F215 39             FOUNDA RTS               A=RESULT
331
332                     ********************************************
333                     * INTERRUPT SERVICE ROUTINES FOR MEA SYNTH *
334                     ********************************************
335                     * FRAMRQ  FIRST CALL SUPPLY PITCH (1 BYTE)
336                     *         NEXT CALLS SUPPLY FRAME (4 BYTES)
```

```
                        * IN :X=POINTER CODE FOR SYNTHESIZER
                        * OUT:CODES TO SYNTHESIZER
                        *      X=POINTER NEXT CODE IN BUFFER
                        *      LAST CALL FIRVEC:=STOPRQ
F216 34 06              FRAMRQ PSHS B,A          SAVE STATUS
F218 C6 80                     LDB  #REQMSK      PREPARE TEST READY
F21A 20 04                     BRA  SPCODE       READY ON FIRQ ACK
F21C D5 00              WTREQ  BITB <MEASTS      READY FOR SPEECH BYTE ?
F21E 27 FC                     BEQ  WTREQ        NO, WAIT
F220 A6A0               SPCODE LDA  ,Y+          GET SPEECH CODE
F222 97 00                     STA  <MEASPC      TO SYNTH CODE BUFFER
F224 10BC DFF0                 CMPY NFRAME       REACHED NEXT FRAME ?
F228 26 F2                     BNE  WTREQ        NO, MORE REQ AHEAD
F22A 108C C000                 CMPY #TOP         REACHED END OF BUFFER ?
F22E 25 04                     BLO  FRMCOR       NO, POINTER CORRECT
F230 108E 0000                 LDY  #BOTTOM      BEGIN OF BUFFER
F234 10BC DFF2          FRMCOR CMPY NSPEAK       REACHED NEXT SPEECH ?
F238 26 08                     BNE  SAMERQ       NO, SAME FIRQ ROUTINE
F23A CC F24F                   LDD  #STOPRQ      INITIATE RESPONSE STOP
F23D FD DFFB                   STD  FIRVEC       FOR NEXT MEA REQ
F240 20 08                     BRA  EXFRAM       EXIT ROUTINE
F242 3124               SAMERQ LEAY 4,Y          FRAME AFTER REACHED
F244 10BF DFF0                 STY  NFRAME       IS NEXT FRAME NOW
F248 313C                      LEAY -4,Y         KEEP POINTER REACHED
F24A 97 02              EXFRAM STA  <RESRQ       RESET FLIP-FLOP RQ
F24C 35 06                     PULS A,B          RESTORE STATUS
F24E 3B                        RTI               REQUESTS CAUSE FIRQ

                        * STOPRQ  TERMINATE SPEECH SYNTHESIS
F24F 7F DFFA            STOPRQ CLR  SPEAKS       :=FALSE
F252 34 06                     PSHS B,A          SAVE STATUS
F254 CC 1A80                   LDD  #COD5        SLOW STOP,NO REQ;REQMSK
F257 97 01                     STA  <MEACMD      READY ON FIRQ ACK
F259 D5 00              WTCOD4 BITB <MEASTS      READY FOR SAME COMMAND ?
F25B 27 FC                     BEQ  WTCOD4       NO, WAIT
F25D 97 01                     STA  <MEACMD      TWICE, MEA OLD VERSIONS
F25F 35 06                     PULS A,B          RESTORE STATUS
F261 97 02                     STA  <RESRQ       RESET FLIP-FLOP RQ
F263 3B                        RTI               NO MORE REQUESTS

                        ************************************
                        * INTERRUPT VECTORS DEFINITION *
                        ************************************
F264 6E9F DFFB          FIRACK JMP  [#FIRVEC]    VIA POINTER IN RAM
F268 3B                 GHOST  RTI               RESUME FLOW CONTROL

FFF0                           ORG  $FFF0        INTERRUPT VECTORS 6809
FFF0 FFFF               RESERV FDB  $FFFF        RESERVED
FFF2 F268                      FDB  GHOST        SWI3
FFF4 F268                      FDB  GHOST        SWI2
FFF6 F264                      FDB  FIRACK       FIRQ USED BY MEA
FFF8 F268                      FDB  GHOST        IRQ
FFFA F268                      FDB  GHOST        SWI
FFFC F268                      FDB  GHOST        NMI
FFFE F0C3                      FDB  INIT         RESET
                               END
```

Bijlage II: overzicht MC 68488 GPIA

REGISTER CONTENTS

	7	6	5	4	3	2	1	0	
R0W	IRQ	BO	GET		APT	CMD	END	BI	Interrupt "Mask Register"
R0R	INT	BO	GET		APT	CMD	END	BI	Interrupt Status Register
R1R	UACG	REM	LOK		RLC	SPAS	DCAS	UUCG	Command Status Register
R1W									Unused
R2R	ma	to	lo	ATN	TACS	LACS	LPAS	TPAS	Address Status Register
R2W	dsel	to	lo		hide	hida		apte	Address Mode Register
R3R	RESET	DAC	DAV	RFD	msa	rtl	ulpa	fget	Auxiliary Command Register
R3W		rfdr	feoi	dacr			dacd		Auxiliary Command Register
R4R	UD3	UD2	UD1	AD5	AD4	AD3	AD2	AD1	Address Switch Register
R4W	lpbe	dal	dat	AD5	AD4	AD3	AD2	AD1	Address Register
R5R	S7	SRQS	S6	S4	S3	S2	S1	S0	Serial Poll Register
R5W		rsv							
R6R	B7	B6	B5	B4	B3	B2	B1	B0	Command Pass-Through Register
R6W	PPR8	PPR7	PPR6	PPR5	PPR4	PPR3	PPR2	PPR1	Parallel Poll Register
R7R	DI7	DI6	DI5	DI4	DI3	DI2	DI1	DI0	Data In Register
R7W	DO7	DO6	DO5	DO4	DO3	DO2	DO1	DO0	Data Out Register

Notes:
1. Upper case letters indicate a message resulting from the IEEE-488 Standard bus.
2. Lower case letters indicate a message resulting from the MPU data bus.
3. The bit terminology of the Data In and Data registers represent the numbering of the IEEE-488 Standard bus and not the 6800 MPU bus – see Section 3.1.2.

GPIA INTERNAL CONTROLS AND REGISTERS

There are fifteen locations accessible to the MPU data bus which are used for transferring data to control the various functions on the chip and provide current chip status. Seven of these registers are write only and eight registers are read only. The various registers are accessed according to the three least-significant bits of the MPU address bus and the status of the Read/Write line. One of the fifteen registers is external to the IC but an address switch register is provided for reading the address switches. Table 2 shows actual bit contents of each of the registers.

INTERRUPT MASK REGISTER R0W – The Interrupt Mask Register is a 7-bit storage register used to select the particular events that will cause an interrupt to be sent to the MPU. The seven control bits may be set independently of each other. If dsel (bit 7 of the Address Mode Register) is set high CMD bit 2 will interrupt on SPAS or RLC. If dsel is set low CMD will interrupt on UACG, UUCG, and DCAS in addition to RLC and SPAS. The Command Status Register R1R may then be used to determine which command caused the interrupt. Setting GET bit 5 allows an interrupt to occur on Group Execute Trigger Command. END bit 1 allows an interrupt to occur if EOI is true (low) and ATN is false (high). APT bit 3 allows an interrupt to occur indicating that a secondary address is available to be examined by the MPU if apte (bit 0 of Address Mode Register) is enabled and listener or talker primary address is received and a Secondary Command Group is received. A typical response for a valid secondary address would be to set msa (bit 3 of Auxiliary Command Register) true and dacr (bit 4 Auxiliary Command Register) true, releasing the DAC handshake. BI indicates that a data byte is waiting in the data-in register. BI is set high when data-in register is full. BO indicates that a byte from the data-out register has been accepted. BO is set when the data-out register is empty. IRQ enabled high allows any interrupt to be passed to the MPU.

THE INTERRUPT STATUS REGISTER R0R – The Interrupt Status Register is a 7-bit storage register which corresponds to the interrupt mask register with an additional bit INT bit 7. Except for the INT bit the other bits in the status register are set regardless of the state of the interrupt mask register when the corresponding event occurs. The IRQ (MPU interrupt) is cleared when the MPU reads from the register. INT bit 7 is the logical OR of the other six bits ANDed with the respective bit of R0W.

Interrupt Status Register
(Read Only)

INT	BO	GET	X	APT	CMD	END	BI

INT – Logical OR of all other bits in this register ANDed with the respective bits in the interrupt mask register.
BO – A byte of data has been output
GET – A Group Execute Trigger has occurred
APT – An Address Pass-Through has occurred
CMD – SPAS + RLC + dsel (DCAS + UUCG + UACG) has occurred
END – An EOI has occurred with ATN = 0
BI – A byte has been received

COMMAND STATUS REGISTER R1R – The command status register flags command or state as they occur. These flags or states are simply coupled on the MPU bus. There are five major address commands. REM shows the remote/local state of the talker/listener. REM bit 6, set low, implies the local state. LOK bit 5 shows the local lockout status of the talker/listener. RLC bit 3 is set when a change of state of the remote/local flip-flop occurs and reset when the command status register is read. DCAS bit 1 indicates that either the device clear or selected device clear has been received activating the device clear function. SPAS bit 2 indicates that the SPE command has been received activating the device serial poll function. UACG bit 7 indicates that an undefined address command has been received and depending on programming the MPU decides whether to execute or ignore it. UUCG bit 0 indicates that an undefined universal command has been received.

Command Status Register
(Read)

UACG	REM	LOK	X	RLC	SPAS	DCAS	UUCG

UACG – Undefined Addressed Command
REM – Remote Enabled
LOK – Local Lockout Enabled
RLC – Remote/Local State Changed
SPAS – Serial Poll Active State is in effect
DCAS – Device Clear Active State is in effect
UUCG – Undefined Universal Command

ADDRESS STATUS REGISTER R2R – The address status register is not a storage register but simply an 8-bit port used to couple internal signal modes to the MPU bus. The status flags represented here are stored internally in the logic of the chip. These status bits indicate the addressed state of the talker/listener as well as flags that specify whether the chip is in the talk only or listen only mode. The ATN, bit 4, contains the condition of the Attention Line. The ma signal is true when the chip is in:

TACS – Talker Active State
TADS – Talker Addressed State
LACS – Listener Active State
LADS – Listener Addressed State
SPAS – Serial Poll Active State

Address Status Register
(Read Only)

ma	to	lo	ATN	TACS	LACS	LPAS	TPAS

ma – my address has occurred
to – the talk-only mode is enabled
lo – the listen-only mode is enabled
ATN – the Attention command is asserted
TACS – GPIA is in the Talker Active State
LACS – GPIA is in the Listener Active State
LPAS – GPIA is in the Listener Primary Addressed State
TPAS – GPIA is in the Talker Primary Addressed State

AUXILIARY COMMAND REGISTER R3R/W – Bit 7, reset, initializes the chip to the following states: (reset is set true by external RESET input pin and by writing into the register from the MPU).

SIDS – Source Idle State
AIDS – Acceptor Idle State
TIDS – Talker Idle State
LIDS – Listener Idle State
LOCS – Local State
NPRS – Negative Poll Response State
PPIS – Parallel Poll Idle State
PUCS – Parallel Poll Unaddressed to Configure State
PPO – Parallel Poll No capability

rfdr (release RFD handshake) bit 6 allows for completion of the handshake that was stopped by RFD (Ready For Data) holdoff commands hlda and hlde.

fget (force group execute trigger) bit 0 has the same effect as the GET (Group Execute Trigger) command from the controller.

rtl (return to local) bit 2 allows the device to respond to local controls and the associated device functions are operative.

dacr (release DAC handshake) bit 4 is set high to allow DAC to go passively true. This bit is set to indicate that the MPU has examined a secondary address or an undefined command.

ulpa (upper/lower primary address) bit 1 will indicate the state of the LSB of the address received on the DIO1-8 bus lines at the time the last Primary Address was received. This bit can be read but not written by the MPU.

msa (valid secondary address) bit 3 is set true (high) when TPAS (Talker Primary Addressed State) or LPAS (Listener Primary Addressed State) is true. The chip will become addressed to listen or talk. The primary address must have been previously received.

RFD, DAV, DAC – (Ready For Data, Data Valid, Data Accepted) bits assume the same state as the corresponding signal on the MC68488 package pins. The MPU may only read this bit. These signals are not synchronized with the MPU clock.

dacd (data accept disable) bit 1 set high by the MPU will prevent completion of the automatic handshake on Addresses or Commands. dacr is used to complete the handshake.

feoi (forced end or identify) bit 5 tells the chip to send EOI low. The EOI line is then returned high after the next byte is transmitted. NOTE: The following signals are not stored but revert to a false (low) level one clock cycle (MPUϕ2) after they are set true (high):

1. rfdr
2. feoi
3. dacr

These signals can be written but not read by the MPU.

Auxiliary Command Register

reset	rfdr	feoi	dacr	msa	rtl	dacd	fget	Write
	DAC	DAV	RFE			ulpa		Read

reset – initialize the chip to the following status:
(1) all interrupts cleared
(2) following bus states are in effect: SIDS, AIDS, TIDS, LIDS, LOCS, PPIS, PUCS, and PPO
(3) bit is set by RESET input pin

RESET – The RESET input provides a means of resetting the GPIA from a hardware source. In the low state, the RESET input causes the following: ← LABBUS

- The Interrupt "Mask" register is reset;
- All status conditions are reset;
- The GPIA is placed in the Untalk/Unlisten state;
- The Parallel Poll, Serial Poll, Data In, and Data Out registers are reset;
- The Address register and Address mode register are cleared;
- All stored conditions in the Auxiliary Comand register except bit 7 are reset – (bit 7 is set);
- T/R1, 2 will go to the low state.

When RESET returns high (the inactive state) the GPIA will remain in the reset condition until the MPU writes bit 7 of the Auxiliary Command register (R3W) low. Prior to the release of the software reset bit, the only register that can be accessed is the Address register. The conditions affected by the RESET pin cannot be changed while this pin is low.

ADDRESS SWITCH REGISTER R4R – The address switch register is external to the chip. There is an enable line (ASE) to be used to enable three-state drivers connected between the address switches and the MPU. When the MPU addresses the address switch register the ASE line directs the switch information to be sent to the MPU. The five least-significant bits of the 8-bit register are used to specify the bus address of the device and the remaining three bits may be used at the discretion of the user. The most probable use of one or two of the bits is for controlling the listener only or talk only functions.

Address Switch Register
(Read Only)

UD3	UD2	UD1	AD5	AD4	AD3	AD2	AD1

AD1-AD5 – Device address.
UD1-UD3 – User definable bits.
When this "register" is addressed, the ASE pin is set which allows external address switch information from the bus device to be read.

ADDRESS REGISTER R4W – The Address Register is an 8-bit storage register. The purpose of this register is to carry the primary address of the device. The primary address is placed in the five least-significant bits of the register. If external switches are used for device addressing these are normally read from the Address Switch Register and then placed in the Address Register by the MPU.

AD1 through AD5 bits 0-5 are for the device's address. The lsbe bit 7 is set to enable the Dual Primary Addressing Mode. During this mode the device will respond to two consecutive addresses, one address with AD1 equal to 0 and the other address with AD1 equal to 1. For example, if the device's address is HEX 0F, the Dual Primary Addressing Mode would allow the device to be addressed at both HEX 0F and HEX 0E. The dal bit 6 is set to disable the listener and the dat bit 5 is set to disable the talker.

This register is cleared by the RESET input only (not by the reset bit of the Auxiliary Command Register bit 7).

When ATN is enabled and the primary address is received on the IB0-7 lines, the MC68488 will set bit 7 of the address status register (ma). This places the MC68488 in the TPAS or LPAS.

When ATN is disabled the GPIA may go to one of three states: TACS, LACS, or SPAS.

Address Register
(Write Only)

lsbe	dal	dat	AD5	AD4	AD3	AD2	AD1

lsbe – enable dual primary addressing mode
dal – disable the listener
dat – disable the talker
AD1-AD5 – Primary device address, usually read from address switch register.
Register is cleared by the RESET input pin only.

SERIAL POLL REGISTER R5R/W – The Serial Poll Register is an 8-bit storage register which can be both written into and read by the MPU. It is used for establishing the status byte that the chip sends out when it is serial poll enabled. Status may be placed in bits 0 through 5 and bit 7. Bit 6 rsv (request for service) is used to drive the logic which controls the SRQ line on the bus telling the controller that service is needed. This same logic generated the signal SRQS which is substituted in bit 6 position when the status byte is read by the MPU IB0-IB7. In order to initiate a rsv (request for service), the MPU sets bit 6 true (generating rsv signal) and this in turn causes the chip to pull down the SRQ line. SRQS is the same as rsv when SPAS is false. Bit 6 as read by the MPU will be the SRQS (Service Request State).

Serial Poll Register
(Read)

S8	SRQS	S6	S5	S4	S3	S2	S1

S1-S8 – Status bits.
SRQS – Bus in Service Request State

Serial Poll Register
(Write)

S8	rsv	S6	S5	S4	S3	S2	S1

S1-S8 – Status bits
rsv – generate a service request

DATA-IN REGISTER R7R – The data-in register is an actual 8-bit storage register used to move data from the interface bus when the chip is a listener. Reading the register does not destroy information in the data-out register. DAC (data accepted) will remain low until the MPU removes the bytes from the data-in register. The chip will automatically finish the handshake by allowing DAC to go high. In RFD (ready for data) holdoff mode, a new handshake is not initiated until a command is sent allowing the chip to release holdoff. This will delay a talker until the available information has been processed.

Data-In Register
(Read Only)

D17	D16	D15	D14	D13	D12	D11	D10

PROGRAMMING CONSIDERATIONS

The following is a list of considerations when using the MH version of the MC68488: MH7948

Interrupt Structure

The status bits in R0R, when set, cause an interrupt (drives the $\overline{IRQ}$ line low) if the appropriate interrupt mask bits in R0W are set. The $\overline{IRQ}$ line is sensitive to a low to-high transition produced by the logical OR of the appropriate bits in R0R. If, for example the BI status bit is set and causes an $\overline{IRQ}$ interrupt, the MPU reads R0R (this read will reset the $\overline{IRQ}$ line but not the status bit) and detect that the BI bit is set. The software should then direct the MPU to read the data byte from R7R, which in turn causes the BI bit to be reset. If after the status register (R0R) was read and before R7R is read, another interrupt status bit is set (e.g., the CMD bit) this second condition does not cause an interrupt. The BI bit being set at the time CMD occurred prevents the $\overline{IRQ}$ line from detecting the necessary low-to-high transition and an interrupt could be missed. To prevent this, the last set of instructions in the software interrupt handler should be a reset of the interrupt mask register, followed by programming this same register to its original state. This always produces the needed low-to-high transition, preventing missed interrupts.

END Status Bit

The END status bit in R0R is used to indicate to addressed listeners that the next byte received by the addressed talker is the last byte of a string. This bit is not qualified with the handshake and thus occurs ahead of the reception of the last data byte. This alerts the MPU that the final byte will soon follow. Because of this, two interrupts, if so programmed, will occur. One for the END bit and one for the BI bit when the final byte is transferred with a handshake. For those situations where it is inconvenient to have two interrupts the END status bit can be masked, not allowing it to cause

SPAS status bit

If the controller conducts a serial poll by sending Serial Poll Enable (SPE) and then sends the GPIA talk address, the SPAS status bit is set and can cause an interrupt. After the controller receives the Serial Poll Status byte it will send Serial Poll Disable (SPD) which resets the SPAS status bit. If the controller can perform this sequence of events before the interrupt handler can check the SPAS status bit, the MPU will not find any status bits set!

If this device had actually requested the service, then the MPU, after receiving the interrupt ("ghost" or not), should check bit 6 of the Serial Poll register. If this bit is reset the MPU knows that a Serial Poll was conducted and can reset the rsv as per normal Serial Poll handling procedures.

Serial Poll Procedure

The MPU initiates a service request by writing rsv (bit 6, R5W) high in the GPIA. At the same time, the appropriate code should be placed in the other 7 bits. Bit 6 being set causes the $\overline{SRQ}$ management line to go low. The GPIA enters the Serial Poll Active State (SPAS) when it receives SPE and is an active talker. When it enters SPAS, the following occurs: the SPAS status bit (bit 2, R1R) is set, the CMD status bit (bit 2, R0R) is set, the $\overline{SRQ}$ line is asserted passively false (high), the SRQ status bit (bit 6, R5R) is reset, and the contents of R5R is placed on the GPIB data bus.

When the GPIA enters SPAS, the SPAS status bit (R1R) is set. This, in turn, causes the CMD status bit in R0R to be set. In an interrupt driven system with the CMD and IRQ mask bits set, this causes an MPU interrupt. These status bits are not latched conditions and only monitor the current state of the GPIA. If the controller places the GPIA in SPAS (sends SPE and MTA), receives the Serial Poll status byte and removes the GPIA from SPAS (sends SPD) before the MPU reads the Interrupt Status register, the contents of this register shows hex 10. Since the MPU knows that this device issued the service request, it should check bit 6 of R5W if an MPU interrupt is generated but no status bit is set. If bit 6, R5R, is reset, the MPU will know the controller has performed a Serial Poll on it. However, the SRQ status bit being reset does not indicate that the status byte was accepted by the controller — that is, the handshake was completed. Rather, it indicates that the GPIA has been placed in SPAS and that the status byte has been placed on the GPIB. In systems with slow responding controllers, the SRQ bit in R5R can be reset while the SPAS status bit is still set. In this case to determine when the status byte was accepted, the MPU can monitor SPAS status bit. This bit is reset when the controller has removed the GPIA from the SPAS. Once in SPAS, the controller must accept the Serial Poll byte before removing the device from SPAS. The rsv bit cannot be written low until the status byte has been accepted, but should be written low as soon as the status byte has been accepted by the controller.

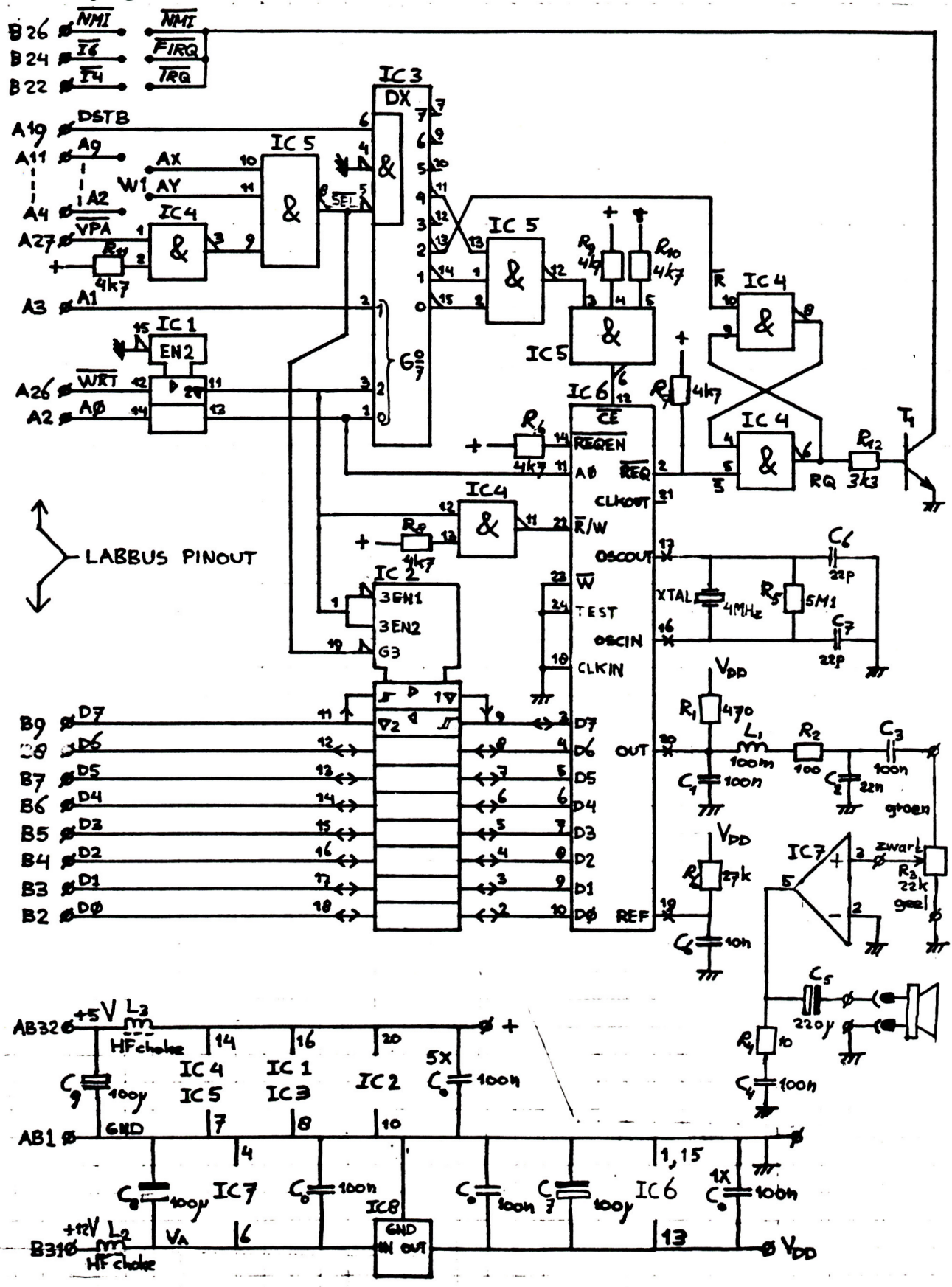
If this device has issued a service request to the controller, the following provides a procedure for handling a SPAS interrupt. The procedure only discusses Serial Poll (SPAS) interrupts. Interrupts resulting from other sources need to be incorporated as appropriate for the system application. In an interrupt driven system, the MPU normally reads the Interrupt Status Register to find the cause of the interrupt. The Interrupt Status Register must be read to release the $\overline{IRQ}$ line and, in most cases, it will be read to check if something other than SPAS caused the interrupt. However, since it is possible that the SPAS status can be set and then reset before the MPU reads the register, the following procedure should also be used (even though the SPAS status is reset).

1) The MPU should monitor the SRQ bit in the Serial Poll Register. This can occur as a result of either an interrupt or a polling routine.
2) When the SRQ bit returns to zero, it indicates that the MC68488 has been placed in the Serial Poll Active State (SPAS). This does not mean that the device is in SPAS, because the controller could have placed the MC68488 in SPAS and then removed the device from SPAS before the MPU reads the Serial Poll Register (R5R).
3) After the SRQ bit in R5R returns to zero, the MPU should read the Command Status Register and Monitor the SPAS status bit. When this bit returns to 0, it indicates that the Serial Poll Status byte has been accepted by the controller and that the MC68488 has been removed from the Serial Poll Active State (SPAS).
4) After the SPAS status bit returns to 0, the rsv bit (in R5W) should be written low.

NOTE

After a Serial Poll has been conducted on the GPIA, and the SRQ bit (bit 6, R5W = 0) is reset, the MPU must write the rsv (bit 6, R5W) low before another service request can be initiated.

Bijlage III: MEA 8000 interface voor LABBUS



IC1 : 74 LS 367 A
IC2 : 74 LS 245
IC3 : 74 LS 138
IC4 : 74 LS 00
IC5 : 74 LS 10
IC6 : MEA 8000
IC7 : LM 386
IC8 : 7805

Bijlage IV: layout MEA 8000 interface voor LABBUS