Techno talk
Chippen en pinnen

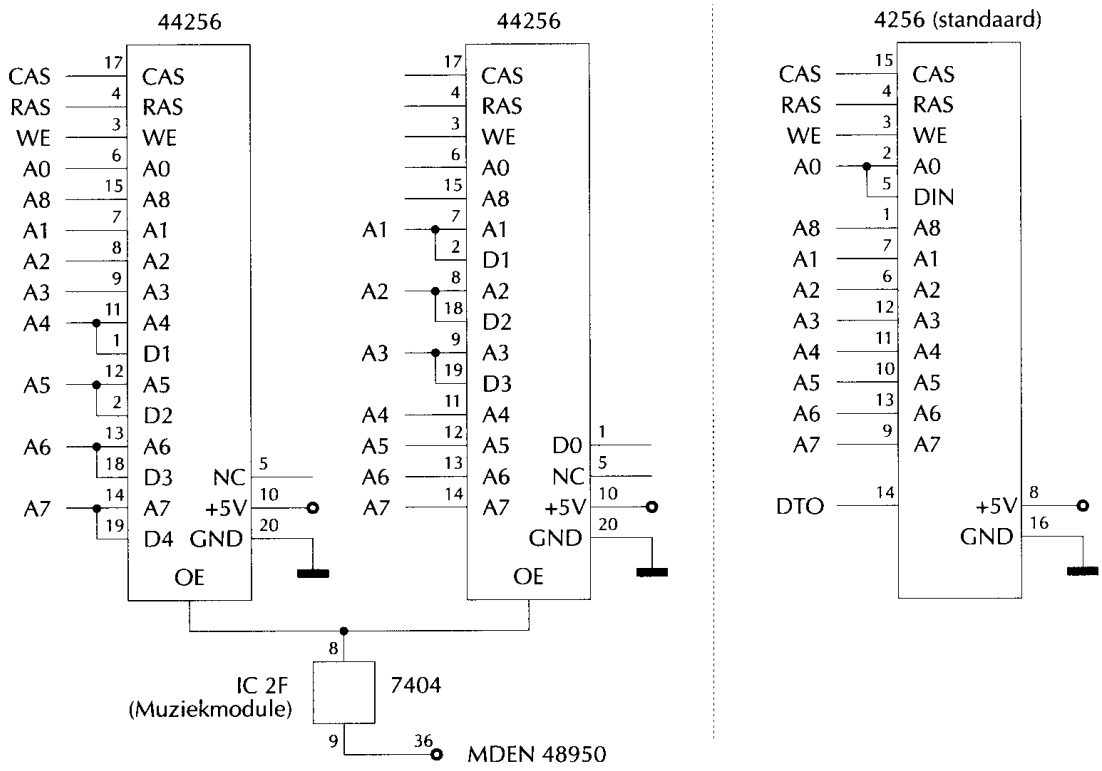
MSX Computer & Club Magazine nummer 88 - voorjaar 1997
Sandy Brand en Bas Vijfwinkel

Scanned, ocr'ed and converted to PDF by HansO, 2001

Het is weer een bijeenraapsel van onderwerpjes geworden, met als grootste gedeelte het uitbreiden van de sampleram van de muziekmodule en de VDP-registers met undocumented features. Jawel!

256 kB sampleram

Met twee ram-ic's is het mogelijk om het muziekmodulegeheugen uit te breiden van 32 naar 256 kB ram. In sampletijd dus een acht keer zoveel. In plaats van een sample van vier seconden op 16 kHz af te spelen, kun je er nu eentje van 32 seconden laten horen.
De ram-ic's zijn van het type 44256. Deze werden voorheen in de 286 moederboardjes gebruikt als main-ram. Nieuw kunnen ze nog wel wat kosten, maar een tweedehands moederbordje is nog wel eens goedkoper te krijgen. Bovendien zit er dan vier keer zoveel op, dus als je met z'n vieren deze ombouw doet, zul je waarschijnlijk redelijk goedkoop uit zijn. Als je het niet aan durft om in je muziekmodule te solderen, dan kun je het altijd nog laten doen door een professional. Maar voor degene die het lef wel hebben, kijken we eens naar figuur 1.



Figuur 1: sampleram–uitbreiding 256 kB

Je hebt alleen twee 44256 ramchips nodig. In figuur l staat nog een 7404 inverter, maar daarvan zit er nog één ongebruikt in de module. Dit is ic 2 (pen 8 en 9). Naar pen 8 moet dus het signaal (MDEN) dat op pen 36 van de soundchip (Y8950) zit. Pen 9 van ie 2 moet naar pen 16 van de ramchips en hiermee worden de ramchips geschakeld. We hoeven bij dit project geen ic's uit te solderen. De re-
den hiervoor is dat het dan onnodig ingewikkeld wordt. Dit grapje is ooit eens bedacht door Digital KC. Het eerste blok sampleram wordt namelijk iets anders geschakeld dan de rest. Hiervoor zouden we dus nog een extra schakel-ic erbij moeten doen als we die oude ram (ic 8 - 4256) eruit zouden gooien. Dat doen we dus niet en we laten het eerste blokje zitten zoals het nu zit.
We houden nu wel een blokje van 32 kB over uit de 256 kB die we erbij plaatsen, maar dat maakt het er eigenlijk alleen maar gemakkelijker op. Zoals je wel kunt zien in figuur l, zijn er redelijk veel pennen met hetzelfde signaal. Wat ik de eenvoudigste manier vond, was het opstapelen van de ic's. Als je ook nog een gedraaid voetje ertussen zet, kun je alles op je bureau in elkaar knutselen en daarna in een keer op de oude sampleram (ic 8) zetten en vast solderen.
Van het bovenste ic knip je pen l, 2, 5,18 en 19 zo kort, dat ze niet het onderste ic raken, maar wel zo dat je er nog een draadje aan kunt
solderen. De andere pennen kun je gewoon zo laten zitten en vast solderen aan het onderste ie. Pen l, 2,18,19 moeten verbonden worden met een draadje aan respectievelijk 11,12,13 en 15. Van het onderste ic moeten pen l, 2,3,4,5,6, 7,16,18, 19 en 20 kort geknipt worden, zodat ze de pennen van het voetje niet raken. Pen 2,18 en 19 moeten verbonden worden met respectievelijk 7,8 en 9.
Als het goed is zitten pen 8,9,10, 11,12,13,14,15 en 17 in pen 6, 7,8, 9,10,11,12,13 en 15 van het voetje. Nu zijn er nog vijf draadjes nodig om wat signalen te verbinden. Pen 3,4,6, 7 en 20 van het onderste ic moeten naar pen 3,4, l, 2 en 16 van het voetje. Nu zou het bouwwerkje af zijn.

Het kan nu op ic 8 gezet worden en de pennetjes van het voetje kunnen aan ie 8 worden gesoldeerd. Voor de draadjes kun je het beste wire-wrap-draad gebruiken. Dit is zeer fijn geïsoleerd koperdraad. Geïsoleerd wikkeldraad is ook wel geschikt. Als laatste moeten de twee draden van en naar de inverter (ic 2) nog worden vastgemaakt. Het is soms wat priegelwerk, maar als je de tijd neemt, dan moet het wel te doen zijn. Ik heb in totaal vier modules op deze manier verbouwd en ze werken nog alle. Maar heb je niet zoveel vertrouwen in je eigen soldeerkunsten, dan kun je het altijd laten doen.

Software

Het aansturen ervan is eigenlijk niets moeilijker dan voorheen. Je kon het adres voorheen tot 1FFFh instellen en nu kan dat tot adres FFFFh. Sommige demo's — bijvoorbeeld Unknown Reality — gebruiken de 256 kB sampleram, maar Moonblaster kan nog niets met het extra geheugen. Maar er is voldoende in MCCM en PTC geschreven over hoe je je eigen programma's voor de module kunt maken.

Samplerom

Behalve extra ram kun je ook extra rom aan de module hangen. Ik heb dit nog nooit gezien, maar de manual van de Y8950 legt wel uit hoe het moet worden aangesloten. Ook de Panasonic MSX Audio Module heeft de rom-samples niet aan de module hangen, maar aan de Z80. Die samples worden dan naar de sampleram verplaatst als je ze wilt horen. Wel heeft hij de AudioBasic en AudioBios aan boord. Allerlei standaardfuncties kunnen dan met een CALL-commando worden aangeroepen in plaats van alles te moeten doen in machinetaal. Ook de FM-pac heeft een verkapte vorm van deze AudioBasic en AudioBios aan boord. Zo stond eens in een Japans blad een replayroutine voor de FM-pac op twee pagina's, die blijkbaar gebruik maakte van deze bios-functies.

FM-pac samples

Het is trouwens ook mogelijk om met de FM-pac samples af te spelen. Er zit een test register in (0Fh). Hiermee kun je de chip in een testmode zetten, waarbij de data van register l0h direct naar de DAC van die chip gaat. Het is 4-bits en je moet zelf alle data erheen gooien, maar toch leuk dat hij het kan. Ik weet er zelf niet de details van, maar als iemand iets meer weet, dan wil ik er best wel een stukje aan wijden in de volgende aflevering.

Simsala-1A-bim

Ik ben bezig met het vertalen van de Mega-scsi handleidingen en steeds kom ik daar de regel 'om een korte beschrijving te krijgen, kunt u het volgende typen in de MSX DOS commandline: TYPE ESET.COM'— maar dan in het Japans — tegen. Dus ik heel ongelovig toch maar eens proberen. En warempel, het werkt nog ook. Na een diskmoni-tortje ter hand te hebben genomen, blijkt het nog heel erg eenvoudig te zijn. Als je TYPE invoert met de desbetreffende programmanaam — het programma is een gewoon werkend programma — dan krijg je netjes een uitleg hoe je het moet gebruiken en verder niets, dus geen troep of een reeks bagger op je scherm. Hoe doen

ze dat? Als je de code bekijkt, zie je het volgende:
org &H100 ;begin programma

```
jp startcode
defb &HOD ;begin regel
defb "TEXT"
defb &H0D, &H0A
defb "l 2 3"
defb &H1A
startcode: ;begin code
```

Bij het opstarten springt het programma naar startcode en voert het programma uit. Bij TYPE filenaam begint de command interpreter de bytes naar het scherm te sturen. De eerste drie bytes van de JP-instruc-tie leveren niets zinnigs op, maar omdat de vierde byte &H0D is, gaat hij toch weer terug naar het begin van de regel en wordt er over de eerste drie bytes heen geprint. Met &H0D, &H0A spring je weer naar een nieuwe regel voor de volgende tekst.
De truc zit hem in &H1A. Pas in een oud boekje, waarin de volledige ascii tekenset staat, kwamen we er achter dat &H1A aangeeft dat een tekst geëindigd is en niet meer verder hoeft te worden afgedrukt. Dit doet dos dan ook niet en springt terug naar de commandline. Hierdoor wordt dus alleen alles voor &H1A afgedrukt en de rest van de code, dat als tekst niets betekent, wordt niet afgedrukt.
Er zit echter een klein addertje onder het gras. Als er in de eerste JP-instructie ook een &H1A — bijvoorbeeld JP &H1A78 — zit, dan wordt er ook verder niets afgedrukt. Mocht je dus niets te zien krijgen, dan moet je dat even controleren. Alle ascii's onder de &H20 hebben een speciale betekenis, dus hou dat even in de gaten! Met &H07 kun je net als in basic ook een beepje laten horen. De truc is eenvoudig, maar kom er maar eens op.

VDP registers

In assembly moet je alles zelf doen, het bios helpt wel een handje, maar soms is snelheid nu eenmaal belangrijker. Voor degene die zelf de VDP willen instellen, zullen we een aantal registers nader bekijken, zie figuur 2. In deze registers worden altijd de bases van bepaalde tabellen gespecificeerd: het adres van de eerste byte in die tabellen dus. Aangezien aan de VDP 128 kB te koppelen is, worden de plaatsen van tabellen aangeduid met een 17-bits geheugenadres. De 64 kB ex-tended vormt een apart blok en is dus niet tegelijk te adresseren. In de meeste gevallen echter limiteert de VDP je in het aantal mogelijke tabelposities door de onderste bits te maskeren, maar hierover straks meer.

Sprites

Sprite pattern table
Specificeert de hoogste zes bits (All - Al6) in register R#6 (= VDP(6)). Er zijn dus 64 mogelijke posities in de vram, segmentgrootte = 2048 bytes.

Sprite attribute table
Hierin staan onder andere de x- en y-posities van de sprites. Specificeert de hoogste tien bits (A07 -Al6) in VDP registers R#5 en R#ll (= VDP(5) en VDPÜ2)). Echter, de

laagste twee instelbare bits (A07 en A08) worden genegeerd door de VDP. Dit zou nog steeds 256 mogelijke posities in de vram geven, met segmenten van 512 bytes (alleen de eerste 32*4 = 128 bytes zijn in gebruik). Echter, er zit nog een leuk addertje onder het gras. In het V9938 MSX video Technical Data Book van Yamaha staat netjes vermeld dat de positie van de Sprite color table wordt berekend door 512 van het Sprite attribute table-adres af te trekken. Dit is echter niet helemaal correct, in feite wordt namelijk alleen maar bit 3 in register R#5 (= A09) gereset voor deze berekening. 2A9 is inderdaad 512 en kies je een adres waarin A09 geset is, kun je inderdaad heel snel — en dat zal waarschijnlijk ook wel de reden zijn — het adres dat 512 bytes lager ligt, berekenen. Er wordt nog op de valreep wel melding van gemaakt, dat dit door de gebuiker zelf moet worden gedaan, maar een beetje flauw is het wel eigenlijk. Kies je namelijk een adres waarin dit bit al gereset is, kom je in de problemen omdat dan de sprite attribuut- en kleurtabel elkaar gaan overlappen! Uiteindelijk houden we dus maar 128 bruikbare posities in de vram over (segmenten van 1024 bytes, eerste 128 in gebruik). In de technische manual van Yamaha raadt men aan genegeerde bitjes (in dit geval A07 en A08) altijd op l te zetten, nou ja, laten we dit dus voor alle zekerheid ook maar blijven doen.

Scherm 5 tot en met 8

In de grafische schermmodi is eigenlijk nog maar één extra ding in te stellen en dat is welke pagina zichtbaar is. Dit staat in register R#2, echter op een beetje merkwaardige manier. De manual vermeldt dat in dit register de hoogste zeven bits van het basisadres worden gespecificeerd (AIO - A16). Voor scherm 5 en 6 kun je de zichtbare pagina specificeren met A15 en A16 (vier pagina's) en alleen met Al 6 voor scherm 7 en 8 (maar twee pagina's).
De manual vermeldt dan dat de lagere bitjes op een moeten staan; waarom staat er echter niet bij. Dus dan maar eventjes proberen door een aantal bits op 1 te zetten. Tot onze grote verbazing verknipt de VDP dan opeens het scherm in stukken en zet sommige meermalen op het scherm! Zet bits A10 tot en met A14 maar eens op 0 en plots worden over het hele scherm de eerste acht beeldlijnen steeds herhaald onder elkaar weergegeven!
Met een beetje experimenteren zijn we tot de volgende verklaring gekomen: het scherm is op verschillende niveau's verdeeld in stroken. Elk bit in register R#2 heeft betrekking op een verschillende strook-grootte. Verschillend niveau als het ware. Deze grootte in horizontale beeldlijnen is uit te rekenen met behulp van het adresbitnummer: grootte = 2^(adresbitnummer - 7). Door nu van bovenaf de stroken te nummeren, beginnend met 0 kun je per strookgrootte aangeven of de stroken met oneven nummers een herhaling van de vorige zijn of niet (0 = herhaal). Om dit te verduidelijken geven we een voorbeeld, zie figuur 3. We kijken naar adres bit A10. De stroken waarop deze bit effect heeft, zijn 2^(10-3) = 8 pixels hoog. Wanneer A10 geset is, zal het scherm er normaal uit zien, maar wanneer deze gereset is, zullen stroken l en 3 er hetzelfde uitzien als hun voorgangers 0 en 2. Hetzelfde gaat op voor A11, alleen zijn de stroken hier niet acht pixels hoog, maar zestien. Merk ook op dat de verschillende instellingen worden gecombineerd, dus de indeling binnen een zestien-pixelhoge strook wordt weer bepaald door de indeling van de acht-pixelhoge stroken van een niveau lager als het ware, et cetera. Het mooie hiervan is dat sprites hier helemaal geen last van hebben, de VDP laat ze ongemoeid. Ook register R#23 (= VDP(24)) blijft nog dezelfde functie vervullen. Ik heb een klein basic programmaatje gemaakt om te laten zien wat voor leuke effecten hiermee te maken zijn.

Experimenteer er maar op los!

Scherm 4

Ook al wordt dit scherm tot de MSX 2 schermen gerekend, eigenlijk is dit gewoon scherm 2, maar dan met de sprite mode 2, dus met meer kleuren sprites en met maximaal acht sprites zichtbaar per beeldlijn, met hier en daar wat minder mogelijke posities voor tabellen omdat sommige lagere adresbits genegeerd worden. Het is een char-acter based scherm dat in feite een uitbreiding is op scherm l. Het scherm is opgedeeld in drie blokken — boven, midden en onder — waarin per blok een aparte charac-ter pattern- en bijbehorende kleurta-bel te definiëren zijn. Al deze blokken staan, net zoals in de pattern name table, achter elkaar in de vram.
Pattern name table Specificeert de hoogste 7 bits (A10 -A16) in register R#2 (= VDP(2)) Er zijn dus 128 mogelijke posities in de vram, segmentgrootte = 1024 bytes, alleen de eerste 3*32*8 = 768 bytes in gebruik.
Pattern generator table Specificeert de hoogste 6 bits (A11 -A16) in register R#4 (= VDP(4)). Dit zou 64 mogelijk posities geven, maar bits A11 en A12 vervullen een heel andere functie. Met deze bitjes kun je namelijk gedeeltelijk aangeven welk patroontabelblok per blok gebruikt moet worden. Zie figuur 4. Voor het middelste en onderste blok kan men aangeven of het dienovereenkomstige blok pa-troondata gebruikt moet worden, of die van het bovenste scherm! Intern in de VDP zal het schakelen van die blokken waarschijnlijk op deze manier ook sneller zijn. Uiteindelijk houden we dus zestien mogelijke posities over, met segmenten van 2048 bytes, waarvan alleen de eerste 256*8*3 = 6144 bytes in gebruik. Color table Specificeert de hoogste elf bits (A06 - A16) in registers R#3 en R#10 (= VDP(3) en VDP(11)). Hier wordt echter een aantal bits ook anders gebruikt: de onderste vier (A06 - A10) worden genegeerd — de VDP ziet hier denkbeeldig allemaal nullen — en de daarop volgende twee (A11 en A12) hebben hetzelfde soort effect als bits A11 en A12 hebben in R#4, maar dan alleen met betrekking op de patroonkleuren, zie figuur 4. We houden dan ook precies hetzelfde aantal mogelijke posities over als voor de pattern generator table.

A16...A0 is een 17-bits adres

Register 11 - VDP(12): Sprite attribute table base address (high)

7	6	5	4	3	2	1	0
0	0	0	0	0	0	A16	A15

Noot: de sprite color table is afhankelijk van het sprite attribute table base address; deze ligt hier 512 bytes onder.

Register 5 - VDP(5): Sprite attribute table base address (low)

7	6	5	4	3	2	1	0		Effectief							
A14	A13	A12	A11	A10	A9	A8	A7	➜	A14	A13	A12	A11	A10	1	1	1

Register 6 - VDP(6): Sprite pattern generator table base address

7	6	5	4	3	2	1	0
0	0	A16	A15	A14	A13	A12	A11

Register 2 - VDP(2): Bitmap table base address (pixel based screen modes)

7	6	5	4	3	2	1	0		Effectief							
0	A16	A15	A14	A13	A12	A11	A10	➜	0	A16	A15	1	1	1	1	1

Schakelen van zichtbare pagina's: screen 5-6 [A16 A15]; screen 7-8 [A16]

Register 2 - VDP(2): Pattern name table address (character based screen modes)

7	6	5	4	3	2	1	0
0	A16	A15	A14	A13	A12	A11	A10

Register 10 - VDP(11): Color table base address (high)

7	6	5	4	3	2	1	0
0	0	0	0	0	A16	A15	A14

Register 3 - VDP(3): Color table base address (low)

7	6	5	4	3	2	1	0		Effectief							
A13	A12	A11	A10	A9	A8	A7	A6	➜	A13	1	1	1	1	1	1	1

Register 4 - VDP(4): Pattern generator table base address

7	6	5	4	3	2	1	0		Effectief							
0	0	A16	A15	A14	A13	A12	A11	➜	0	0	A16	A15	A14	A13	1	1

Figuur 2: VDP registers

Stroken van 8 pixels hoog (A10)

Normaal scherm

R#2 = 0xx11111b

	0	255	
0	ABCDEF		strook 0
8	abcdef		strook 1
16	12345		strook 2
24	67890		strook 3

Herhaling aan

R#2 = 0xx11110b

0	255
ABCDEF	
ABCDEF	
12345	
12345	

Stroken van 16 pixels hoog (A11)

Normaal scherm

R#2 = 0xx11111b

	0	255	
0	ABCDEF		strook 0
	abcdef		
16	12345		strook 1
	67890		

Herhaling aan

R#2 = 0xx11110b

0	255
ABCDEF	
abcdef	
ABCDEF	
abcdef	

Stroken van 8 en 16 pixels hoog (A10 en A11)

Normaal scherm

R#2 = 0xx11111b

	0	255	
0	ABCDEF		strook 0
8	abcdef		strook 1
16	12345		strook 2
24	67890		strook 3

Herhaling aan

R#2 = 0xx11110b

	0	255
str 0	ABCDEF	
	ABCDEF	
str 1	ABCDEF	
	ABCDEF	

Figuur 3: schermlay-out

	Normaal scherm	Herhaal 1	Herhaal 2	Herhaal 3
	R#3 = x11xxxxxb	R#3 = x00xxxxxb	R#3 = x01xxxxxb	R#3 = x10xxxxxb
	R#4 = xxxxxx11b	R#4 = xxxxxx00b	R#4 = xxxxxx01b	R#4 = xxxxxx10b
Blok 1	ABCDEFGH	ABCDEFGH	ABCDEFGH	ABCDEFGH
Blok 2	12345678	ABCDEFGH	12345678	ABCDEFGH
Blok 3	abcdefgh	ABCDEFGH	ABCDEFGH	abcdefgh

Figuur 4: Blokweergave in screen 4 met verschillende waarden voor A12 en A11 in registers 3 en 4