Het Morse Key Board

door Wim de Kruijf PAoWV

Inleiding

Het MKB is niet het Midden en Kleinbedrijf dat Japanse koopdozen en antennes in blisterverpakking levert, maar een Morse Key Board.

Voor een ander project kwam ik op het idee om een oud PC-AT keyboard te gebruiken voor het genereren van morsetekens, dat zou ik "even" doen maar dat viel heel erg zwaar tegen. Uiteindelijk is het gelukt en daarom, hier een apart verhaal over dat keyboard alleen. Je kunt een en ander makkelijk nabouwen, het schema is simpel de geprogrammeerde chip kun je bij mij verkrijgen, de bedoeling is echter dat er ook van kan worden geleerd, en dat als je het anders wilt doen met bijvoorbeeld een ander type controller, dat ook met de hier en op een website [1] verstrekte gegevens en wat zelfwerkzaamheid kan.

Het PC-AT keyboard

Op diverse plaatsen op Internet zijn gegevens te vinden over het PC-AT keyboard die cruciale fouten bevatten, voorts elkaar op punten tegenspreken, onvolledig zijn en bij gebruik dus leiden tot lastig op te lossen problemen.

Het keyboard wordt gevoed via de DIN 5 pens aansluit connector en mijn eerste exemplaar, een super ouwetje uit 1984, trekt bij 5V een niet onaanzienlijk stroom van 120 mA. Daarom heb ik er een wat moderner exemplaar ingezet dat bovendien meer toetsen heeft.
De keyboarden met PS-2 connector zijn hetzelfde behalve de connector waar een verloop voor verkrijgbaar is. Er zijn naast de 5 volt voeding, een datalijn en een kloklijn op de connector aanwezig.

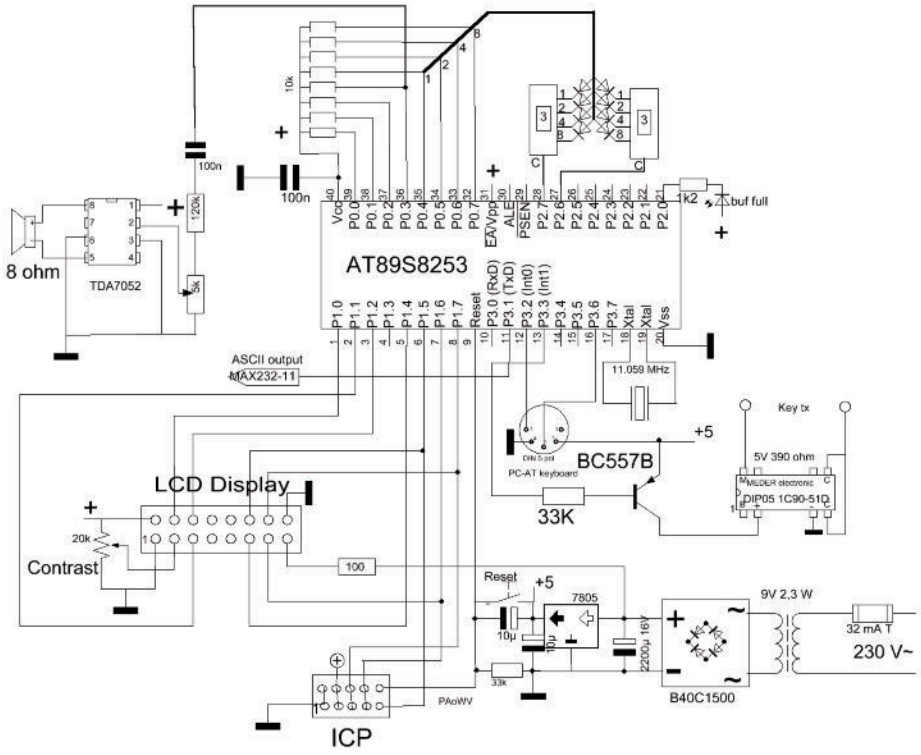
De klok wordt altijd door het keyboard gegenereerd. De datastroom kan naar het keyboard toe (commando's) en uiteraard van het keyboard af. De richting van de informatie en de dataflow control worden bepaald door de controller van het aangesloten apparaat, in dit geval dus het MKB.

Beide lijnen, klok en data, zijn open collector en in rust hoog. De controller van het op het keyboard aangesloten apparaat kan ze laag trekken. Het keyboard zendt alleen als de kloklijn niet laag getrokken wordt door de controller. De kloklijn fungeert dus tevens als CTS (clear to sent). Het keyboard genereert de klok en ontvangt een commando als de datalijn eerst laag getrokken wordt door de controller (RTS Request to sent) tot de kloklijn een serie klokpulsen begint af te geven. Er is altijd een startbit, een stopbit, 8 databits en een odd parity bit. Bij verzenden van een byte naar het keyboard toe, heeft het keyboard aan het einde van het verzonden teken een laag bit af ter bevestiging. Je kunt commando's stu-

ren bijvoorbeeld om een LED in het keyboard aan of uit te doen, of om het laatst ontvangen byte te herhalen.
In dat geval van een verzonden commando wordt na het commando tevens een byte 0xFA gestuurd als bevestiging, of 0xFE als het verkeerd overkomt of het commando niet kan worden uitgevoerd. En dat na elk byte van een meerbyte commando. Tot zover de details. Om poorten van de basisroutines get byte en put byte op een andere controller mogelijk te maken heb ik de flowcharts van die assembly routines getekend die zijn te vinden op [1].

Uiteindelijk heb ik de zaak goed kunnen uitzoeken en werkend gekregen middels een geheugenscope, die eenmalig ontvangen signalen in een geheugen vasthoudt en periodiek als een normaal scopebeeld weergeeft.

Het keyboard geeft zogenaamde scancodes af, zowel bij het maken als bij het loslaten van een toets. Die kunnen uit meerdere bytes bestaan (maximaal 8). Met opzoektabellen worden die in de controller omgezet in ASCII, de toetsen die geen ASCII zijn, zoals de Fn toetsen, pijltjes etc. worden als upper ASCII (bit 8 = hoog) volgens een in principe door mij willekeurig gekozen code afgegeven, zodat die toetsen ook ergens anders voor gebruikt

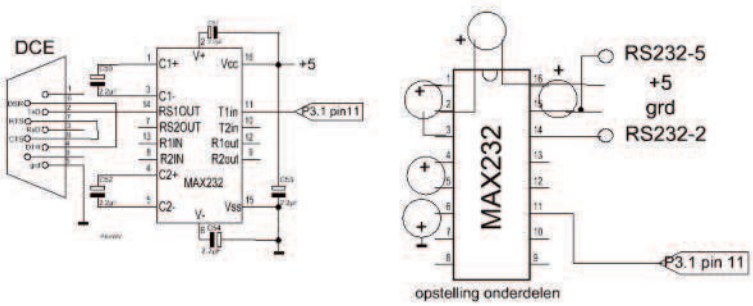


kunnen worden en dat in dit ontwerp ook worden. In de vertaalroutines van scancode naar ASCII wordt bijgehouden of sprake is van capslock, alt, shift, control en numlock, de bijbehorende LEDs worden (met commando's naar het keyboard toe) bediend. Als fouten in de keyboard communicatie worden gedetecteerd, wordt daar door hervragen adequaat op gereageerd. Helpt hervragen (3 keer) niet dan volgt een commando "keyboard reset".
Het keyboard zelf bevat een buffer voor scancodes die vooruit typen mogelijk maakt, (en die bij keyboard-reset leeg wordt opgeleverd) die is echter te klein om sneller typen dan zenden foutloos af te handelen, het zijn namelijk 16 bytes die kunnen de scancodes van maximaal 5 letters bevatten, daarom worden in de controller de ingetypte letters in ASCII code in een circulaire buffer met een capaciteit van 80 bytes opgeslagen. Tamelijk veel, maar het biedt de gelegenheid tijdens een QRS rag chew QSO tijdens de uitzending van de zenderbeschrijving tussendoor even weg te lopen om koffie te gaan drinken.

De inhoud van de buffer die nog niet is uitgezonden staat op een LCD display.

Backspace heeft tot gevolg dat in de buffer, indien niet leeg, het laatste karakter wordt weggehaald. Op die wijze kun je dus misslagen in de ingetypte tekst corrigeren. De delete toets op het nummerpad rechts, laat hele woorden tegelijkertijd verdwijnen, die wist namelijk tot hij een spatie tegenkomt.

De morsetekens zoals <SK> , dus ...-.- worden prosigns genoemd. De 7 prosigns zijn aparte fonts voor ontworpen die worden bij de initialisatie in de LCD geladen.



Seriele interface ASCII keyboard output

Komt de buffer voor meer dan 87% vol dan gaat een rode LED branden om te waarschuwen dat bij doortypen dataverlies dreigt. Dat kan natuurlijk ook een zoemertje of iets dergelijks zijn.

Verder kun je ingeval de ontvangst van BK tijdens een QSO de buffervoorraad data onderbreken (pauzetoets) of zelfs geheel weggooien met ctrl-alt-del, die de vertrouwde combinatie vormt om de hele zaak te resetten. De morsesnelheid is instelbaar tussen 6 en 99 wpm door Alt en dan twee cijfers in te toetsen terwijl je Alt ingedrukt houdt. Het is verstandig tijdens dat gebeuren even de pauzetoets in te drukken, want de cijfers schuiven achter elkaar op hun plaats, zodat na het eerste cijfer de snelheid een onverwacht zeer hoge waarde kan hebben. De morsesnelheid is in woorden per minuut, PARIS standaard. Die wordt dan ook in EEPROM gewijzigd, zodat de volgende keer na inschakelen van de netspanning het keyboard op de laatst gebruikte snelheid terugkomt.

Het blokje van 6 toetsen tussen het alfanumerieke linker deel van het keyboard en het nummerpad rechts, is gebruikt om de prosigns in te coderen, dat zijn de tekens SK, AR, KA etc. Die toetsen dus een plaketiketje opdoen, tot je ze uit je hoofd weet tegen de tijd dat die er afgesleten zijn. Of BK een prosign is of niet kan zelf bepaald worden door de letters b en k apart in te toetsen, of door de pijl omhoog onder het prosignblok, die bk als zevende prosign bevat.

Er is een 4 cijferige contestteller geïmplementeerd. Wordt op de grote + toets rechts op het nummerpad gedrukt dan komt die teller in de zendbuffer te staan en wordt tevens daarna 1 verhoogd voor de volgende keer. De - toets erboven decrementeert de teller 1, voor het geval het getal herhaald moet worden is die functie noodzakelijk. Bij een reset wordt de teller op 0001 geinitialiseerd. Je kunt de teller op een willekeurige gewenste stand zetten door die in de lege buffer in te tikken en uitzending te voorkomen met de pauzetoets. Vervolgens de entertoets op het keypad indrukken, dan verdwijnt het 4 cijferige getal in de teller. Ter bevestiging verdwijnt het van de display. Verdwijnt het niet dan waren het geen 4 cijfers en kan het naar wens alsnog gewijzigd worden.

Een en ander is goed uitgezocht zodat een betrouwbaar geheel is ontstaan.

Onder de F toetsen kun je een bericht opnemen, van 80 bytes maximaal per toets. Dat kan als volgt gebeuren.

1. Druk de pauzetoets in
2. Type de gewenste tekst in, en kijk op de display of die naar wens is. Zo niet editten met backspace en Del.
3. Druk Alt in en vervolgens tevens de F toets waar het bericht onder moet komen.

Je kunt vervolgens kiezen of je pauzetoets bedient en het bericht uit de buffer uitzendt, of dat je dat bericht in de buffer met backspace, delete of in een klap met ctrl-alt-del wist.

Druk je dan op de betreffende F toets dan komt dat bericht direct in de buffer te staan, je kunt het ook meerdere keren achter elkaar in de buffer zetten zelfs als daar geen plaats voor is in de buffer. De berichten onder de F toetsen staan in EEPROM dus die blijven bij uitschakelen van de netspanning aanwezig voor gebruik. Het is mogelijk per QSO op die manier de call van het tegenstation erin te zetten, dan hoef je die tijdens het QSO niet herhaaldelijk in te tikken. Met een paar van dergelijke macro's tik je met een paar toetsaanslagen zo een standaard hallo-goodbye-QSO bij elkaar in je buffer. Dan krijg je de bekende macro QSO's met hams, die zodra je iets afwijkends vraagt of doet schielijk QRT gaan al of niet met smoes dat ze BCI of TVI hebben of zojuist de zendbuizen beginnen te smelten, dan wel, erger nog: schoonmoeder zojuist is gearriveerd. (Dat kun je overigens ook in een macro zetten)

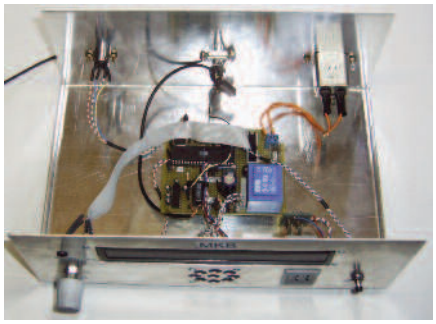
Je kunt ook een bericht onder een Fn toets editten als volgt:
1. Pauzetoets indrukken om uitzending te verhinderen.
2. De betreffende Fn toets indrukken, dat geeft het er onderliggende bericht in de buffer en dus op de display.
3. edit het bericht
4. Plaats het met Alt Fn terug in het EEPROM.

Na de vertaling tot ASCII, die dat deel van het apparaat universeel toepasbaar maakt voor andere doeleinden, en daarom de ASCII output op de UART van de controller naar buiten voert, wordt een volgende vertaalslag gemaakt, waarbij de keuze uit Morse en Hell code is geïmplementeerd. Je kunt van Morse naar Hell schakelen met de linker en rechter pijltoets op het pijltoetsenveldje onder het prosign blokje.

Ontwerpproblemen

De flowcharts van get byte en put byte zijn aanvankelijk als polling routines gerealiseerd. Prima voor een terminal die ASCII code via een UART afgeeft. Nu kun je echter hier niet de output van het keyboard in de controllerbuffer krijgen en daarbij voorlopen op de morse die de buffer leeghaalt en die ook met polling van een interrupt gedreven tijdteller werkt die de lengte van strepen en punten bepaalt.

Een oplossing zou kunnen zijn twee controllers te gebruiken, een voor het keyboard dat via de SSI of UART ASCII afgeeft en een tweede die ASCII aanpakt en in de circulair buffer zet, waar dan ook de morse of Hell verwerking in zit die de buffer leeghaalt.
Om niet direct op deze onelegante aanpak terug te vallen is om te beginnen de polling van het keyboard vervangen door een interrupt service routine. Elke klokflank van het keyboard geeft dan een interrupt die als taak het volgende keyboardbit opneemt. Je hebt dan dus 11 interrupts per keyboard byte. Dat volstaat echter niet om de buffer sneller te kunnen vul-



len met keyboard output dan de morse hem aftapt. Om dat te bereiken zou gestrooid in de morseroutines een subroutineaanroep om het keyboard te servicen multipel moeten worden opgenomen. Ook verre van fraai. En de morse hapert als je commandotoetsen bedient zoals capslock of numlock, want afhandelen van commando's betekent wachten op de acknowledge bytes, parameter bytes ingeven etc.

Daarom heb ik eerst gedacht aan twee stacks, Er draaien in de controller dan twee aparte programma's, het ene haalt het keyboard leeg met polling en geeft ASCII af aan de buffer in het tempo waarin getypt wordt, en het andere programma haalt naar behoefte die buffer in zijn eigen tempo leeg om de ASCII output te transformeren naar Hell- of Morsecode met de gekozen snelheid, en verzorgt de displayverversing.

Een interrupt routine die optreedt bij timer overflow, en die tevens de sidetoonfrequentie bepaalt, drukt dan als extra taak alle gebruikte registers van het ene programma op diens stack, en dan komt het: verwisselt de stackpointer met een tweede stackpointer en gaat dus bij einde interrupt de registerinhouden van de tweede stack halen en die zijn van het tweede programma. Iedere keer als die interrupt optreedt worden dus de stacks verwisseld. Dat is dus een context switch. Beide programma's komen dan even vaak en even lang aan de beurt en hebben beide nagenoeg de halve beschikbare tijd als timeslice beschikbaar. Voor elk van de programma's lijkt het alsof er een interrupt optreedt die ruim de helft van de tijd in beslag neemt.

Op die wijze lopen er twee programma's in de controller onafhankelijk van elkaar, om de beurt. De duur van de tijdsleuven wordt bepaald door de frequentie van de interrupt.

Dat kan echter in dit geval niet werken, omdat elke tijdsleufduur gebruikt voor de context switch en het andere programma te lang duurt om binnen 25 microseconde van een keyboardklokflank de datalijn gegarandeerd te sensen. En dat moet.

Nu kun je dat sensen echter met een andere externe interrupt doen, de klokflank van het keyboard veroorzaakt dan een externe interrupt en onafhankelijk van de even of oneven timeslice wordt die onmiddellijk geserviced. Enige uitzondering is dat de context switch interrupt net aan de gang is, maar die duurt kort. Dan lijkt dat fout te gaan omdat de interrupt de registers opbergt op de verkeerde stack als toevallig het morsedeel aan de gang is, maar dat geeft helemaal niks want die interrupt wordt door de hem toegekende prioriteit altijd eerst afgemaakt, dat wil zeggen dat hij de registers die hij opborg op een willekeurige stack ook weer afhaalt van diezelfde stack. Die keyboard interrupts lopen dan altijd door ook als het andere proces, dat de morse genereert op het moment van interrupt, aan de gang is. Dat is de wijze waarop een en ander is geïmplementeerd.

Testen.

Dit testen is niet bij nabouw, maar bij het porten naar een ander type processor of tijdens het ontwerpen van de schakeling van belang. De circulaire buffer wordt getest door die te vullen met keyboardaanslagen tot hij vol is (carry set) en vervolgens te legen tot hij leeg is (weer carry set). Het onder interrupt schrijven van de putbyte en get byte van het keyboard kan apart getest (kost dagen om onverwachte fouten op te sporen, de vertraging van de “Oh ja Erlebnis” groeit blijkbaar exponentieel met het vorderen van de leeftijd van de ontwerper).
In casu zet je voor putbyte de data klaar om onder interrupt bitsgewijs verstuurd te worden, en mag je niet met get byte (voor ontvangst van het acknowledge byte) beginnen, voor die data daadwerkelijk verstuurd is. Twee wachtlussen dus, een voor toegang tot putbyte, en vervolgens een na putbyte wachten tot de data effectief geheel verstuurd is, wat blijkt omdat de bitcounter van 11 naar 0 gedaald is. Open deur, maar ja..Enfin, na een halve fles port geconsumeerd te hebben zag ik het ineens. Als ik weer een weerbarstig probleem heb kan is dus op basis van deze ervaring weer naar die oplossingsmethode grijpen.

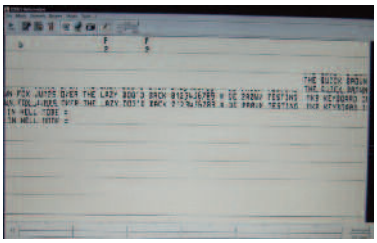
Bij het dual procesontwerp, wordt het tweede proces eerst heel eenvoudig gemaakt, namelijk keer steeds een poortpen om van polariteit. Hang je een scope daaraan, dan zie je dat dat gebeurt gedurende 100 us en dat er 150 us niks gebeurt op die poortpen. Daaruit is te concluderen dat beide processen 100 us van de 250 us beschikbaar hebben, en dat de twee contextswitches 50 us van die tijdspanne, dus 20% van de totale processortijd consumeren bij die schakelfrequentie.

Zo snel schakelen hoeft niet 2 ms lijkt me snel genoeg. Een punt of streep van morse kan dan in het nadeligste geval 2 ms te lang worden, bij 99 wpm, een morsesnelheid die door sommige hams in de wereld met keyboards gehaald en op het gehoor genomen wordt, duurt een punt 12,5 ms en die wordt dan dus maximaal 16% te lang, minder vaak context switchen wil ik dus niet doen, mede omdat de sidetoon die nu 250 Hz is dan te laag in frequentie wordt. Als ik me kwaad maak kan ik dat met een derde proces oplossen, maar ik maak me niet kwaad. Timer 1 wordt daarom voor de context switch ingericht met een herhalingstijd van 2 ms, omdat een timer default in mode 1 van de MCS-51 architectuur door 8192 deelt en dus geen aanpassing van het deeltal (en dus tijd) vergt gedurende
de context switch, uitgezonderd nu het ‘on the fly’ setten van het meest significante bit van de lopende teller.

Opletten dat de beide programma’s in de controller niet elkaars data wijzigen. Dat leidt tot onverwachte effecten. Dit werkt en na zoveel gedoe geeft dat aangevuld met de high makende harsdampen van de soldeerbout de vreugde die slechts de zelfontwerper/bouwer kan ervaren, je waant je dan in het Paradijs met 7 maal 70 maagden ter beschikking, hoewel me dat als uitruil voor mijn geheugenscope een slechte deal lijkt. Van het Paradijs naar de Hell:

Feld Hell

In de Hell mode levert de buffer voor elk erin staand ASCII teken een feld hell teken met een snelheid van 122,5 baud. Dat bepaalt de bandbreedte. De leesbaarheid wordt bij ontvangst verhoogd als je die pulsen van ruim 8 ms of een geheel veelvoud daarvan, kunt verplaatsen over 4 ms. Ze worden dan niet smaller, de bandbreedte neemt dus niet toe. Dat betekent hier dat elke letter bestaat uit kolommen van 14 bits, waarvan altijd elke zwart- en wittijd tenminste twee bits duurt.



Alle letters zijn als hoofdletters in een tabel opgenomen van 64 letters maal 12 bytes. Dat is genoeg voor 6 kolommen per letter, de zevende kolom is altijd een lege letterspatie.

Testen gebeurt door de audio van de sidetoon speaker via een microfoon te stoppen in de laptop geluidsingang waarop het programma van IZ8BLY draait.

Nabouw

Het keyboard zelf zit in een plastic kast en het niet afgeschermde aansluitsnoer werkt als antenne, dus als je veel HF in je shack hebt, krijg je problemen. Daar kom je als SSB-er achter als je je lippen aan de microfoon brandt. Ferriet kan je wellicht redden, een C -tje van de klok en datalijn naar aarde misschien ook. Ik heb de zaak ongevoelig gemaakt voor netschakelaarpulsen etc, door een interrupt van het keyboard te controleren of de spanning dan daadwerkelijk laag is verder en door een netfilter toe te passen.

De DIN connector heeft op de soldeerlippen gezien linksom draaiend de pennummers 1,4,5, 2 en 3 Dat lijkt een rare volgorde maar dat komt omdat er ook 3 pens DIN connectors zijn die de plaats van de pennen 1, 2 en 3 bepalen. Pen 1 is een kloklijn, pen 2 de datalijn, 4 is ground en 5 is +5 volt.

Het geheel is gezet op een half euroformaat (dat is niet de grootte van een 50 ct munt) gaatjesprint, voedinkje erbij, kristal het vlot verkrijgbare 11,0592 MHz type. Controller kun je geprogrammeerd bij mij bestellen via mijncall@amsat.org voor 15 euro inclusief porto en verpakking. Wil je zelf programmeren, dan kan dat ook, het programma is op de hiervoor genoemde website beschikbaar daarvoor. Kastje is gekocht voor minder dan 10 euro bij Baco in IJmuiden. Niet duur, wel een beetje krass- en deukerig. Het is al omgezet dus zagen en boren is dan ronduit lastig.

Het 2 regelige 40 karakters/regel display van Samsung zonder back lite is geleverd door Rein Pentina PAoRKP, die altijd op de vlooienmarkten te vinden is. De aansluitkabel heb ik wel naar de achterzijde verplaatst, dat is door de doorplatering een lastig karwei. Denk eraan dat dan de even en oneven nummers van de kabel van plaats verwisselen.

Als je geen behoefte hebt aan ASCII output op de RS232 connector, dan kun je die connector en de MAX232 met elco's gewoon weglaten. De ICP voet kan ook weggelaten worden als de zaak niet in circuit geprogrammeerd hoeft te worden, omdat je een geprogrammeerde controller chip monteert.

De foto van het apparaat en het schema tonen twee BCD duimwiel schakelaars voor de snelheidsbepaling. Die kunnen worden weggelaten uit het schema, omdat de snelheid makkelijker en goedkoper ingesteld wordt met het toetsenbord zoals hiervoor beschreven. Degenen die toch BCD duimwielen willen toepassen voor de snelheidsinstelling, de zogenaamde duimwielfetishisten, kunnen dat bij bestelling van de chip melden, dan wordt die daarop geprogrammeerd.

Alle verdere onderdelen zijn vlot verkrijgbaar en niet kritisch qua opstelling grootte etc. Eerst de voeding maken, monteer een voet voor de controller, meet tussen pen 20 en 40 of er 5 volt staat, dan de controller erin. Opletten dat er geen pennetjes dubbel klappen en dat hij niet een halve slag gedraaid in de voet wordt gezet. Dat de controller werkt kun je in geval van twijfel meten op pen 30, daar staat dan een signaal van 3 MHz. De LCD contrastregelaar instellen, zodat je beeld hebt. De audio testen door je vinger te drukken op pen 2 van de TDA7052. Ik heb een reed relais gemonteerd, potentiaalvrije contacten, daar kun je dus elke normale zender mee sleutelen, maar je kunt natuurlijk daarnaast of in plaats daarvan ook een transistordriver monteren eventueel met optocoupler voor negatief of positief schakelende seinsleutelaansluitingen. Hoe je zender sleutelt kun je gewoon meten op de contacten van een aangesloten seinsleutel door daar de open spanning en de kortsluitstroom te meten.

55 es bcnu in CW PAoWV

[1] www.xs4all.nl/~pa0wv/morsekb