TYPE `N TALK MEETS THE HERO Jr. ROBOT.

(BUILDING A REPLICA TYPE `N TALK)

Dr. H. Holden, July 2018.

Introduction:

Those familiar with the Hero Jr. Robot know that it contains a very early speech synthesizer IC, The Votrax SC-01A. This IC generates an unlimited vocabulary as it is programmed to create word sounds with Phonemes. To program the Robot to speak, the phoneme codes, in a Hex byte format, can be entered on the Robot's keypad. Alternatively, if the Robot is connected to a computer terminal, or terminal emulator,(via its RS232 serial link) the phoneme symbols, which make up each word sound, can be programmed in directly in Hero Jr. Basic, if the Basic programming cartridge is plugged into the Robot.

The above works, however it is a very time consuming process to program each word. Ideally there would be a way where typed text would automatically be converted to the phoneme string and seldom require correction. Such a device exists, of the same vintage as the robot. It is a mini-computer based on a Motorola 6802 processor with ROM & RAM and a UART (with two way communication & handshaking) and also the SC-01A IC. This unit converts the typed text into a phoneme code and to a speaker. However, due to the two way serial communication, it can also transfer the phoneme string (in Diphthong format) back to the terminal or computer that programmed it. This product is the Votrax Type `N Talk, which was produced in the early 1980's by Votrax, the makers of the SC-01A speech synthesizer IC.

The photos below show the two potential companions, the Heathkit Hero Jr. Robot and the Votrax Type `N Talk or "TNT".

Hero Jr. Robot:

Votrax's original Type 'N Talk unit:

My Replica Type 'N Talk unit:

Buying or Building a Type `N Talk or "TNT" unit:

The next move to see if the idea above would actually work was to acquire the TNT. These days it seems they are relatively rare, only coming up from time to time on Ebay. I decided the best move would be to build one.

While there are a few posts on the internet saying that one was being built, I could see not a single finished working example of a replica yet. So I did not know if it was a certainty that it could be done. The main risk of project failure would be if the only documented copy of the .bin file for the Rom was good.

Fortunately, the TNT has been reverse engineered (by Kevin Horton) to the extent that the original MROM in the unit had been un-potted and the .bin file recovered from it. Also Kevin provided photos of the pcb and a hand drawn schematic. Not all the pcb was visible in the photos, some of it of course hiding under components and IC's. However, I concluded there was enough information present to be able to construct a replica TNT from the photos. Not having a real TNT unit myself to study, it then became a matter of doing it from the pictures of the vintage unit.

THE TNT's POWER SUPPLY:

Firstly the TNT's power supply was subject to study. The circuit is shown below. After studying this and photos of Votrax's pcb, it became apparent that Votrax made two errors:

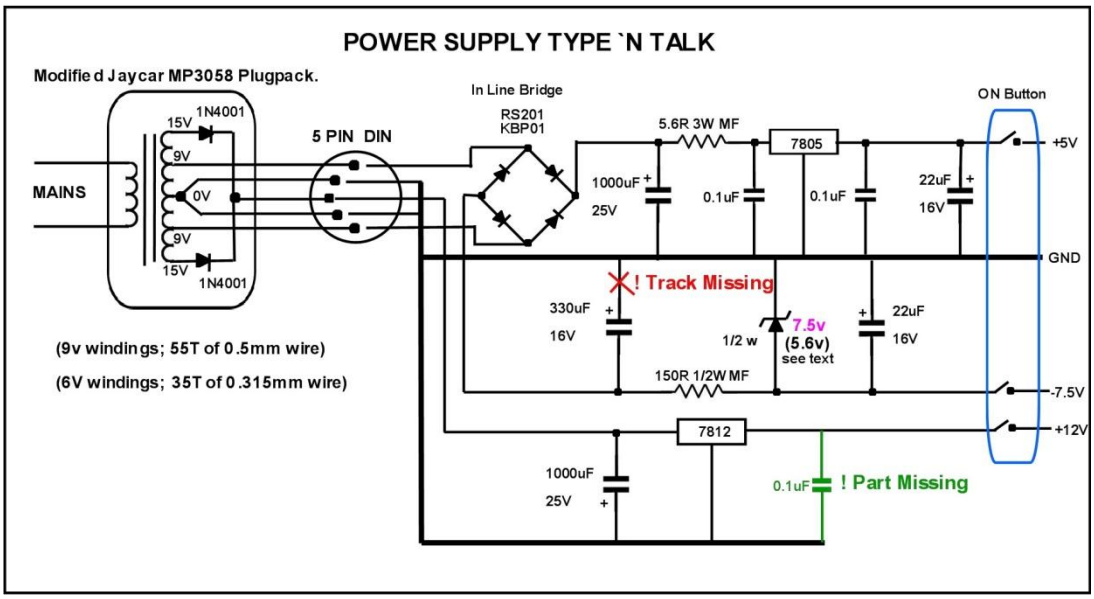
In one case they omitted a section of track-work on their pcb which left the positive terminal of the 330uF electrolytic in the -7.5V power supply section floating. This mistake was never noticed because the 22uF electrolytic capacitor stored enough charge, that despite the ripple voltage, the 7.5V zener diode shunt regulator still worked adequately! On my version of the PCB though, this mistake is corrected.

Secondly, the output of the 7812 regulator had no bypass capacitor near it. Instead its output passed directly to a power switch and then via a long length of pcb track-work to power the SC-01A and the LM386. This is a little risky as sometimes the regulator could become unstable and oscillate. So a 0.1uF capacitor was added to the 7812 circuit near the regulator output.

I had also wondered about the value of the -5.6V zener diode. This stabilizes the negative voltage required by the MC1488 line driver IC for the RS232 interface. It is close to the margin for the negative going serial data voltage, so I increased it to a 7.5V zener diode.

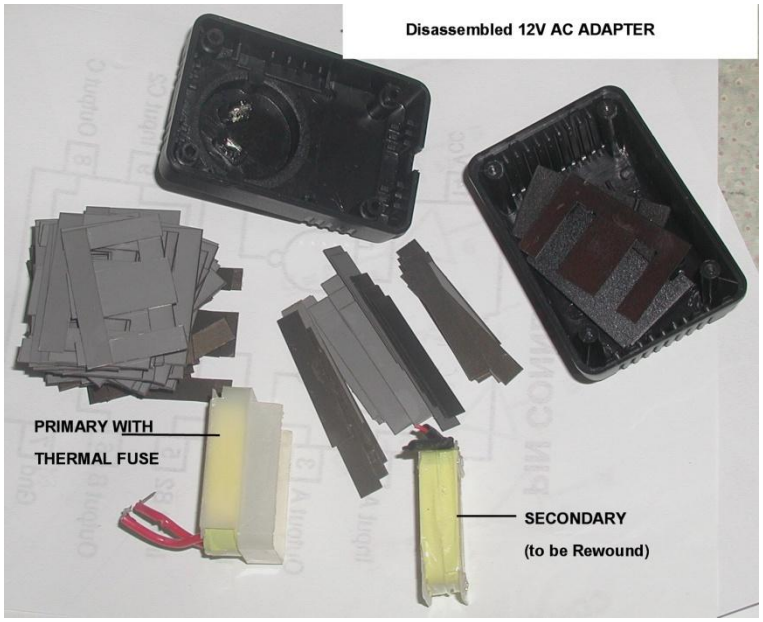
Powering the TNT:

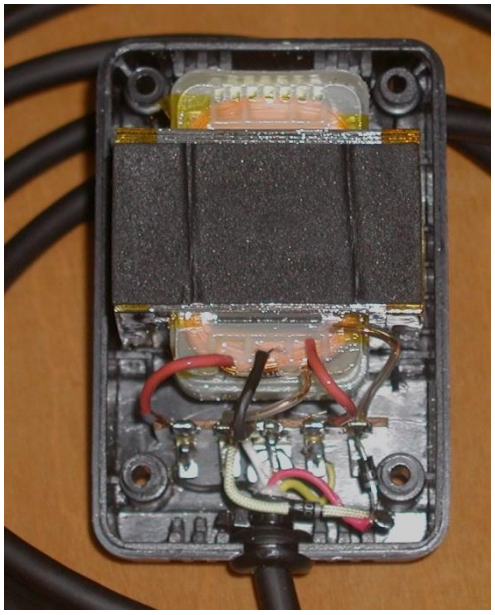
The original Votrax power adapter (from what I have gathered from photographs) was rated a 17VAC rms CT @ 1A and 20V DC @ 0.5A. The actual currents drawn by the TNT are much lower at around 390mA for the +5 supply and about 30mA for the -7.5v supply. Also the +12V supply runs just the SC-01A IC and the + supply for the 1488 line driver IC.



So I acquired a standard mains voltage (230V here) to 12VAC 1.67A rated Jaycar Powertech adapter, disassembled it and rewound its secondary bobbin.

The primary and secondary bobbins are independent (as it's a double insulated transformer). The primary bobbin with its thermal fuse was left alone. I wound the new secondary windings to be 18V and 30V CT and added two diodes. This way it would provide the appropriate voltages for the TNT. The AC adapter was a screw together type which was helpful. This adaptor would power an original TNT unit also.



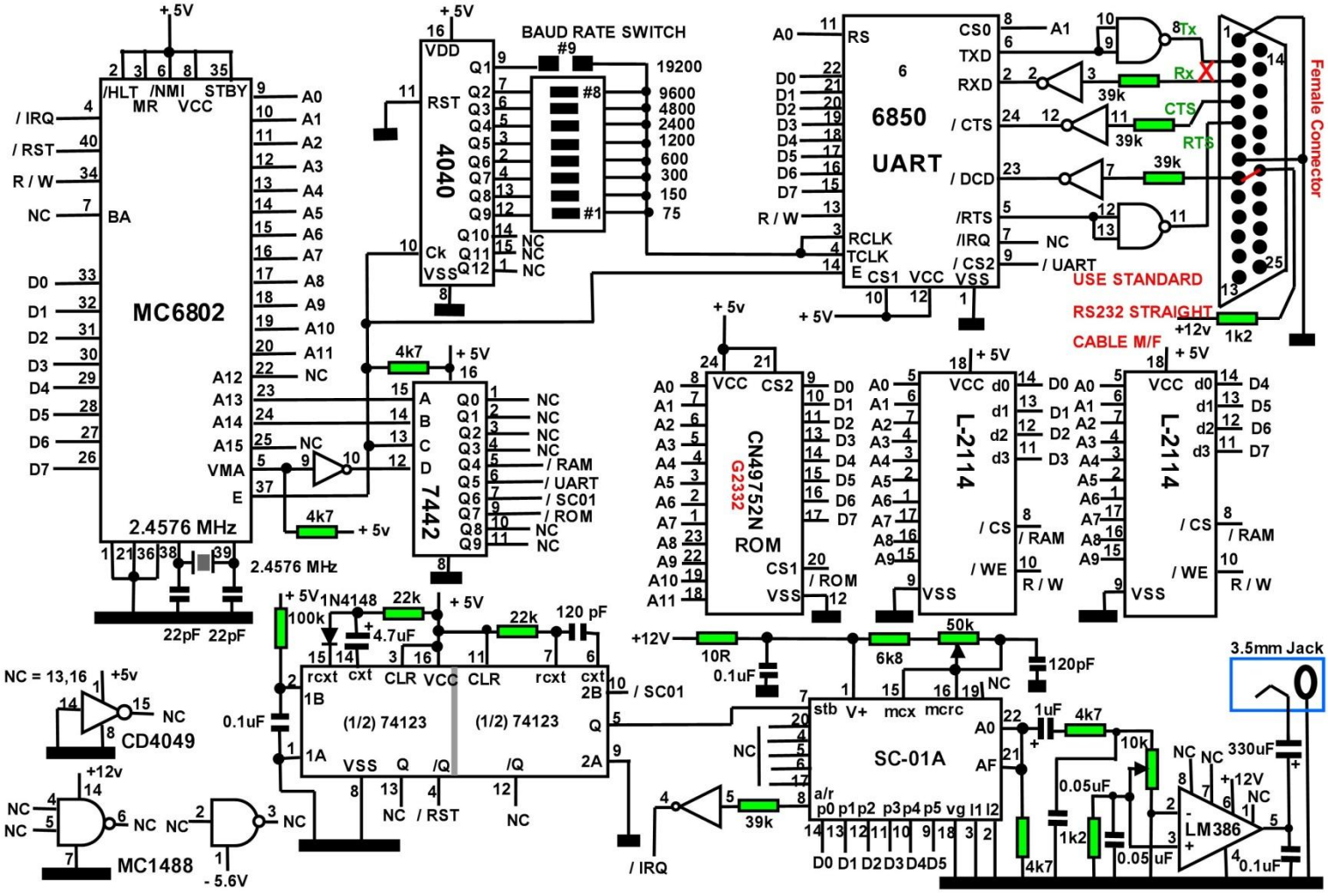


Next the hand drawn circuit diagram (which Kevin Horton had kindly posted) was re-drawn and checked against the pcb photos.

Only three issues were detected. There was a 22uF capacitor on the output of the 7812 regulator that was not shown. Pin 11 of the 4040 IC was shown connected to +5V

(when it should be grounded) and Pin 8 of the UART IC (6850) was shown as N/C when it should be connected to address line A1.

The original Votrax ROM, as far as I can tell, was most likely a GTE Microcircuits G2332 MROM or a Signetics equivalent of that, which is similar to the NEC product D2332C. A pin to pin replacement, in an erasable UV Eprom variety, is the Texas Instruments TMS2532. But as shown by Kevin Horton, a 2732 UV Eprom can be used with some re-wiring of the pins. The re-drawn schematic is shown below:



RS-232 hardware flow control & DB25 connections:

The topic of RS-232 serial connections appear to have created much confusion over the years. It is worth spending some time on this topic as it relates to the TNT. This is certainly evident, even in the TNT's user manual, where there is an error in the RS-232 connection documentation (see below).

Part of this confusion occurs because the Rx or RD labeled terminal on a **modem** (DCE), pin 3, is a transmitting terminal or ***output*** and Tx or TD terminal on pin 2 is ***input***. Likewise RTS pin 4 is an ***input*** and CTS pin 5 is an ***output.***

The reverse is true on a computer (DTE) where Tx is on pin 2 and is an ***output***, Rx on pin 3 and is an ***input***, RTS on pin 4 an ***output*** and CTS on pin 5 an ***input***.

This means the cabling for a "computer to modem link" is a straight pin for pin cable with the labels, in order computer to modem: Rx to Rx, Tx to Tx, RTS to RTS and CTS to CTS.

Obviously, in any RS-232 link of any kind, DTE to DCE, or DTE to DTE, or DCE to DCE, either simple or with hand shaking, outputs are connected to inputs (of the actual line driver and receiver IC's) in all cases, never outputs to outputs or inputs to inputs. Simply paying attention to this fact avoids a lot of grief.

Also to add into the mix, in the case of RS-232 links with handshaking, the RTS is used differently for the DTE to DTC link versus the DTE to DTE link. For example, in the connection of a computer to a modem (DTE to DCE) when there is data that the computer wants to ***send*** to the modem, the computer asserts its RTS output which the modem receives on its RTS input. When the modem is ready to receive, it asserts its CTS output, which signals the computer on its CTS input, to then send the data. If the modem de-asserts its CTS (say if its buffer is full) the data transmission stops or pauses until CTS is reasserted. And when the computer (DTE) has finished transmitting it de-asserts its RTS.

However, to throw a cat into the pigeons, if both of the connected devices are DTE (two computers for example) the handshaking protocol is different. It requires that the RTS and CTS of each computer are cross connected as in Computer.a to Computer.b; RTSa to CTSb, RTSb to CTSa.

For example, in this arrangement, computer.a asserts RTSa to signal computer.b that it is ready to ***receive*** data from computer.b. Computers.a and computer.b check their CTS inputs to determine when it is ok to ***send*** to the other computer.

This is why two computers or two DTE's can be linked with a Null Modem cable which has the Tx & Rx and RTS & CTS connections ***both*** crossed.

However, of note, the DB25 connector's connections that Votrax used in their TNT is neither wired as a DTE or a DTC, it's a hybrid. Of the two types it is more like a DTE as it transmits on pin 2. However it is a female connector and DTE's are more often male.

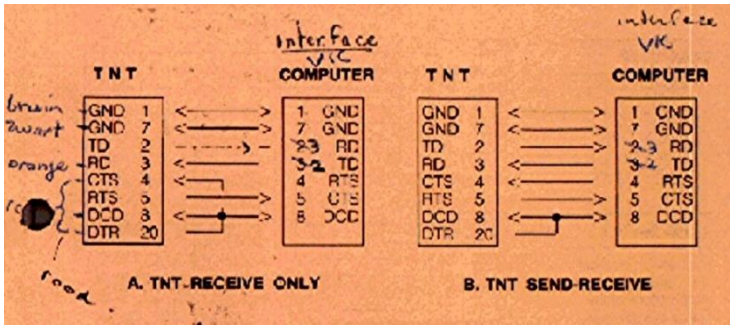
As noted in the schematic above, with the red cross on the DB25 pins 2 & 3 connections, the easiest thing to do, if one wants two way communication and handshaking with the computer and the TNT, is to reverse pin 2 and 3 at the connector on the TNT's pcb. In other words alter Votrax's original pcb design and then use a straight through RS232 cable (pin for pin).

The reason for this is that Votrax have the CTS (an input) and RTS (an output) of the 6850 UART already reversed on the TNT's pcb's female connector. So using a standard female to male RS232 cable (if pin 2 & 3 are reversed as shown in the diagram above) the correct connections are then made, with Tx to Rx and RTS to CTS in both directions.

As Votrax had originally arranged it, with their connector wiring, it would require a Null Modem cable to work, where the TNT is a ***receiver alone*** (so that Rx and Tx are suitably connected). However, the Null Modem cable would not work properly for ***two way communication*** because it also reverses the CTS and RTS connection. However, the CTS and RTS connections are already reversed by Votrax at their pcb's DB25 connector. So an illegal and non functional hardware state of "two outputs connected together" and "two inputs connected together" would occur.

Also, as a Null Modem cable is more often a female to female on each end to allow two DTE's to connect (as DTE's are normally male connectors) it would also need a M-M adapter to be able to plug onto the female connector of the TNT. Although M-F Null modem cables are available too.

Therefore, for two way communication at least, between the TNT and a computer, Votrax used a non standard cable, being neither a "straight through" nor a "Null Modem" cable. However, people using the TNT would have got it working right away as a ***receiver*** at least, with just the Null Modem cable, but it wouldn't have worked to send a phoneme string back to the computer. In addition to this, there is an error in the wiring diagram in the TNT manual on the RS232 connection. Somebody had kindly corrected it with ball point pen on the online manual I found. A computer's serial card with a DB25 connector outputs (transmits) on pin 2 and receives on pin 3:



(From Votrax TNT user manual)

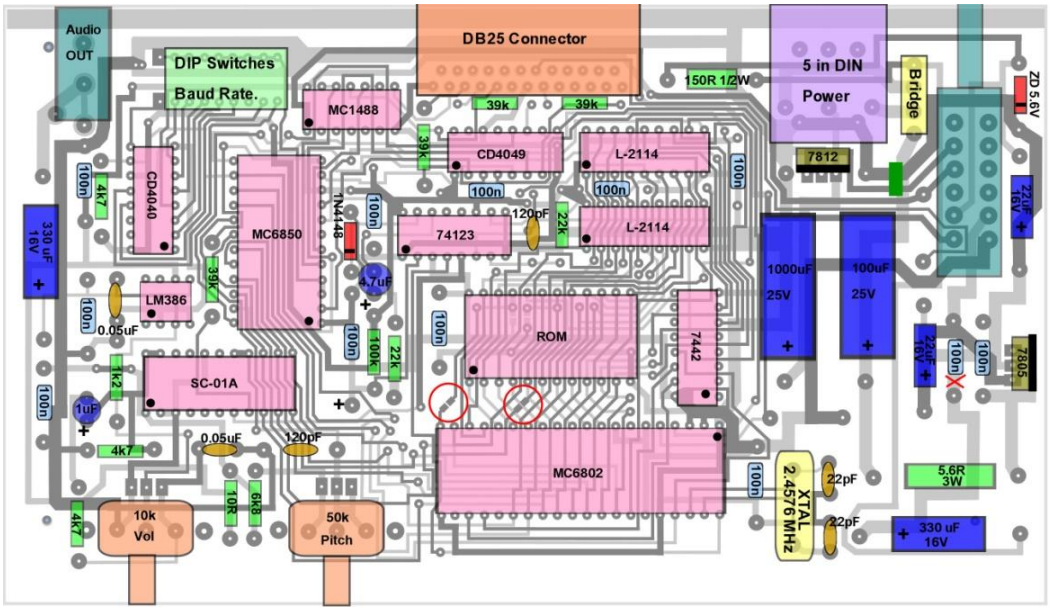
As noted by Votrax, pin 20 (which is tied high with a pullup resistor in the TNT) needs to connect to pin 8 to make sure the carrier detect input is always correct.

(One other possible option to solve the connection/cable problem, to allow two way communication between the TNT and the computer, could be by reversing the RTS and CTS connections on the TNT's connector instead, leaving the Rx and Tx connections as Votrax had them. In this instance then the Null modem cable would work for two way communications. However, I decided that this was an inferior option to reversing the Rx and Tx connections at the pcb's connector and using the straight cable, because there are a few variations of Null Modem cables out there, more variations in these, than just the straight through cable types).

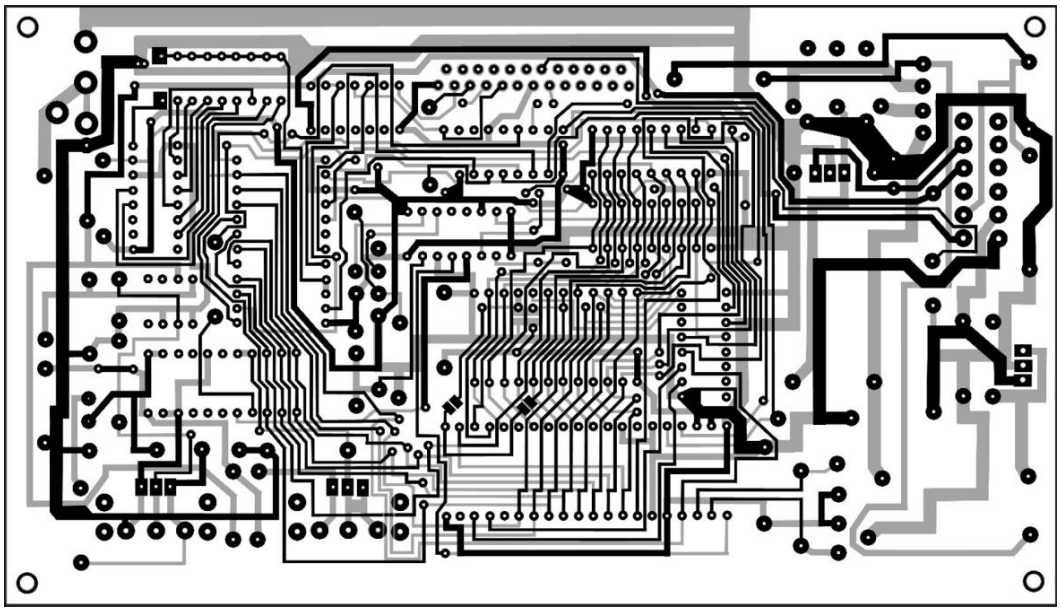
Making the PCB:

I do not have specific pcb drawing software, but I have a good drawing program, the vintage "Microsoft Picture It". So I drew the pcb that way as a .png and .jpg file. Fortunately my pcb maker can get these into Altium with some work & software trickery.

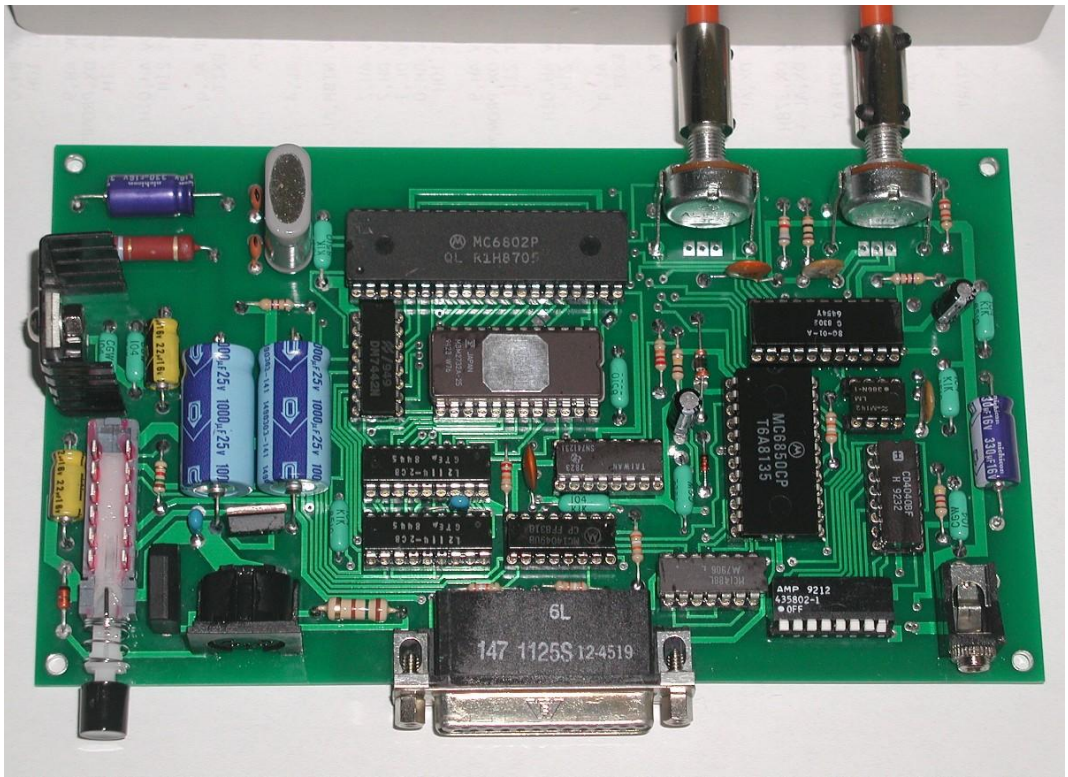
The images below shows the pcb. It took a while to figure out some of the track routings under the IC's. I made it conform as closely as possible to the original, though some changes were made. I could not get exactly the same potentiometers, so their footprint had to be altered a little to fit Alpha potentiometers. I added the missing track (shown in the red X) which was on the lower pcb surface. Of note, the DB25 connector is unique in that its mounting pins extend much further back on its body than the common type. Also there is the added capacitor shown in dark green near the 7812 output. The red circles show added pcb link solder pads (also there are two others on the lower pcb surface), to facilitate using either the TMS2532 or the 2732 UV Eprom variants.



The image below shows the two board sides overlaid on each other. The PCB is close to 7 x 4 inches.

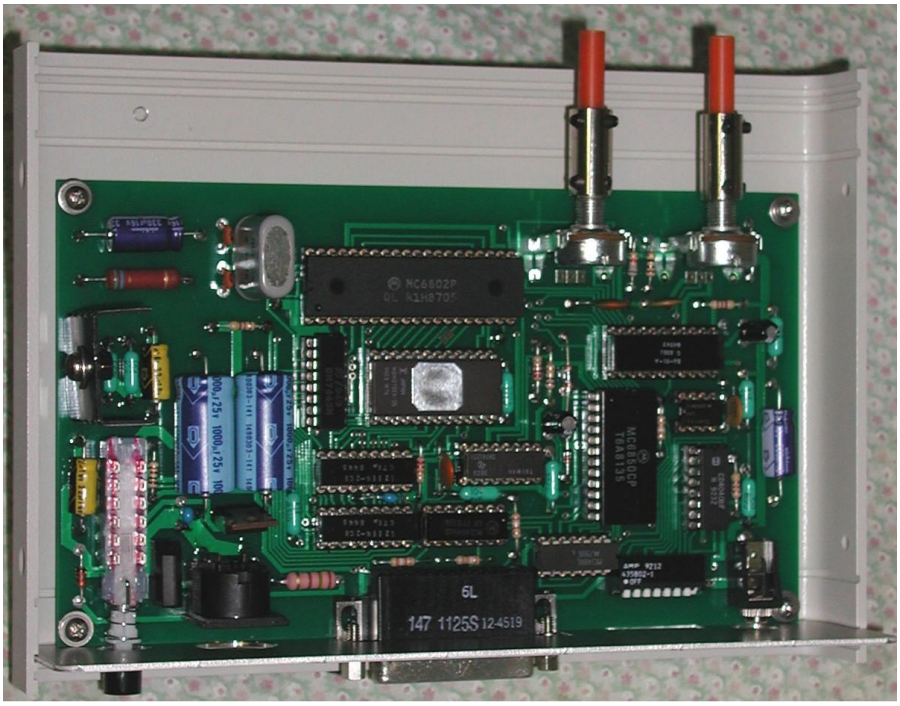


The photo below shows the assembled and tested pcb:



The 0.1uF ceramic bypass capacitors I used are a vintage type made by the Corning Glass Works. The originals on Votrax's pcb were blue. Mostly axial 0.1uF ceramic capacitors are a mustard-yellow color these days.

The enclosure I used to house this pcb is an extruded aluminium case made by TAKACHI in Japan. It has a good scratch resistant coating. These are excellent cases with thick aluminium walls. Very good for tapping threads such as 4-40 UNC, for screws to mount feet and mount PCB stand offs etc. Also optional tilt feet are available for them. Due to the case being a little deeper than the original Votrax case and the potentiometer shafts a little shorter than Votrax's originals, shaft extensions were required.



I stuck to the original style Din power connector, DIP baud rate switch, a long reach DB25 connector, the same design power switch and audio output jack. Therefore the rear panel looks near identical to Votrax's original unit:

The original TNT pcb was mounted in its case into a slot in Votrax's extruded aluminium housing. My pcb is mounted on posts, so 4 holes were placed near the corners of the pcb to facilitate this. I used the same type of hex post seen on either side of the DB25 connector and cut 4-40 UNC threads into the case to avoid a nut on the outer lower case surface.

Downloading Translated Phonemes to the 5155 Computer.

To get the unit up and running in receive mode I connected it to the serial port on my vintage 5155 computer which has Procomm installed. I was delighted when the unit worked first pop confirming a few things: I had not made any errors in the pcb and I had successfully programmed the UV Eprom with a good .bin file.

In default mode (when the unit is powered up) if the RS232 connections are set up with two way communication, or hand shaking, the typed text is not only spoken, but it is echoed to the terminal. Software switches can change the configuration. I found that this worked, however using com1 on the computer the echoed text, sometimes was not there. When I investigated this I found that it is the IBM serial async card in my computer that has an intermittent fault. It fails to receive on the receive input sometimes (I also tested it with an RS232 loopback tester). Luckily I had a COM2 port (on the AST 6 Pak Plus card in the 5155 computer) so I shifted to that port, it is fine. I will have to investigate the IBM async card later, to find out why the serial input Rx line is intermittent.

Then came the task of attempting to operate the software switches to return the translated phoneme data back to the computer. The manufacturer's instructions are shown below:

- **MODE SELECTION** There are several optional modes of operation for a TNT that can be selected using control codes. These modes can be dynamically set in accordance with the user's application. TABLE 3 lists the various control codes along with a mnemonic descriptor of the feature. They will be discussed further below. To invoke a particular mode, an ESCAPE code (HEX 1 B, DECIMAL 27) followed by the corrsponding mode control code must be sent to a TNT while its input mode is ON. NOTE: If the TIMER is enabled then an ESCAPE code must be followed by any code within the timer interval (about 3-4 seconds) or TNT will lock-up and must be powered-down. A block diagram showing data flow within a TNT is given in FIGURE 5. This figure will aid the reader in understanding the effect of the mode controls as they are described in detail below.

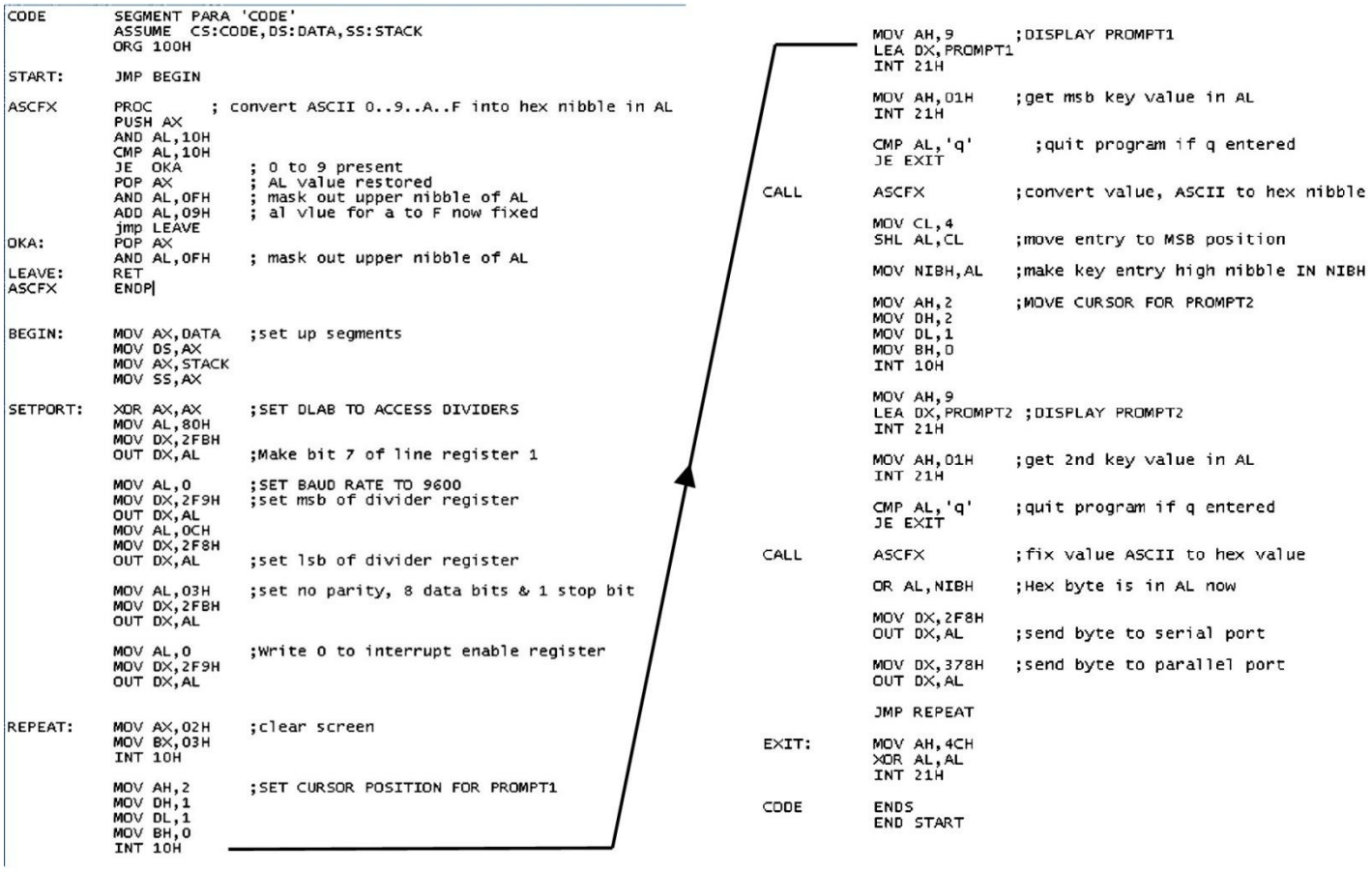
Table 3 Mode Control Codes

	—CONTROL CODES—			
MODE	HEX	DEC	ASCII	
PSEND ON	11	17	DC1	CRSR
*PSEND OFF	12	18	DC2	RVS ON
*ECHO ON	13	19	DC3	CLR HOM
ECHO OFF	14	20	DC4	DNST DE
CAPS ON	15	21	NAK	
*CAPS OFF	16	22	SYN	
TIMER OFF	17	23	ETB	
RESET	18	24	CAN	

*indicates automatic power-up selections

The Procomm program is a terminal emulator, designed to send out typed text as ASCII characters. It does have the ability to translate incoming ASCII data. However, no matter how hard I tried I could not get it to successfully send the 1B Escape code and other control codes to the TNT to operate the software switches.

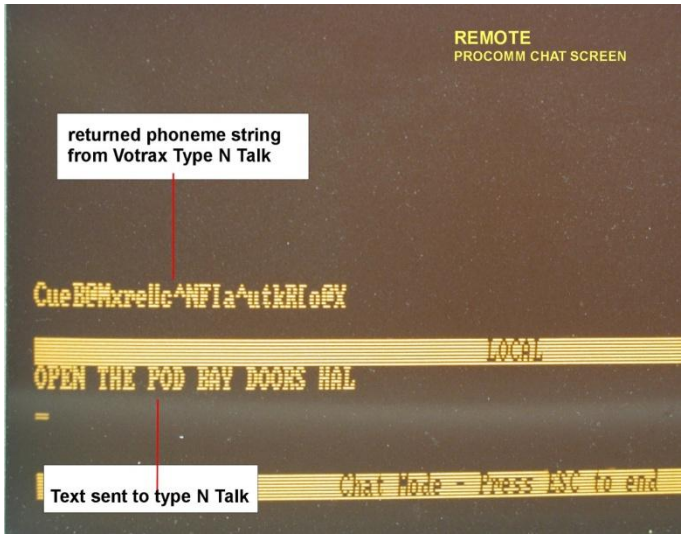
In the end I decided to write a small .EXE utility in assembly language to directly control the serial port and send out the escape and control code bytes that way. The program is shown below (It assembles to an .obj file and links to an .exe file on my 5155 using MASM & LINK).



```
CODE        SEGMENT PARA 'CODE'
            ASSUME  CS:CODE,DS:DATA,SS:STACK
            ORG 100H

START:      JMP BEGIN

ASCFX       PROC      ; convert ASCII 0..9..A..F into hex nibble in AL
            PUSH AX
            AND AL,10H
            CMP AL,10H
            JE  OKA         ; 0 to 9 present
            POP AX          ; AL value restored
            AND AL,0FH      ; mask out upper nibble of AL
            ADD AL,09H      ; al vlue for a to F now fixed
            jmp LEAVE
OKA:        POP AX
            AND AL,0FH      ; mask out upper nibble of AL
LEAVE:      RET
ASCFX       ENDP

BEGIN:      MOV AX,DATA     ;set up segments
            MOV DS,AX
            MOV AX,STACK
            MOV SS,AX

SETPORT:    XOR AX,AX       ;SET DLAB TO ACCESS DIVIDERS
            MOV AL,80H
            MOV DX,2FBH
            OUT DX,AL       ;Make bit 7 of line register 1

            MOV AL,0        ;SET BAUD RATE TO 9600
            MOV DX,2F9H     ;set msb of divider register
            OUT DX,AL
            MOV AL,0CH
            MOV DX,2F8H
            OUT DX,AL       ;set lsb of divider register

            MOV AL,03H      ;set no parity, 8 data bits & 1 stop bit
            MOV DX,2FBH
            OUT DX,AL

            MOV AL,0        ;Write 0 to interrupt enable register
            MOV DX,2F9H
            OUT DX,AL


REPEAT:     MOV AX,02H      ;clear screen
            MOV BX,03H
            INT 10H

            MOV AH,2        ;SET CURSOR POSITION FOR PROMPT1
            MOV DH,1
            MOV DL,1
            MOV BH,0
            INT 10H

            MOV AH,9        ;DISPLAY PROMPT1
            LEA DX,PROMPT1
            INT 21H

            MOV AH,01H      ;get msb key value in AL
            INT 21H

            CMP AL,'q'        ;quit program if q entered
            JE EXIT

CALL        ASCFX           ;convert value, ASCII to hex nibble

            MOV CL,4
            SHL AL,CL       ;move entry to MSB position

            MOV NIBH,AL     ;make key entry high nibble IN NIBH

            MOV AH,2        ;MOVE CURSOR FOR PROMPT2
            MOV DH,2
            MOV DL,1
            MOV BH,0
            INT 10H

            MOV AH,9
            LEA DX,PROMPT2  ;DISPLAY PROMPT2
            INT 21H

            MOV AH,01H      ;get 2nd key value in AL
            INT 21H

            CMP AL,'q'      ;quit program if q entered
            JE EXIT

CALL        ASCFX           ;fix value ASCII to hex value

            OR AL,NIBH      ;Hex byte is in AL now

            MOV DX,2F8H
            OUT DX,AL       ;send byte to serial port

            MOV DX,378H     ;send byte to parallel port
            OUT DX,AL

            JMP REPEAT

EXIT:       MOV AH,4CH
            XOR AL,AL
            INT 21H

CODE        ENDS
            END START
```

The job of this small utility is to get the ASCII typed values of 0 to 9 and A to F, typed consecutively and then convert this to a single Hex byte of those key values and send it out the serial port on COM2. For example if 1 then B is typed, the Hex byte 1B is sent out (Not the ASCII value of 1 then B which is not 1B hex).

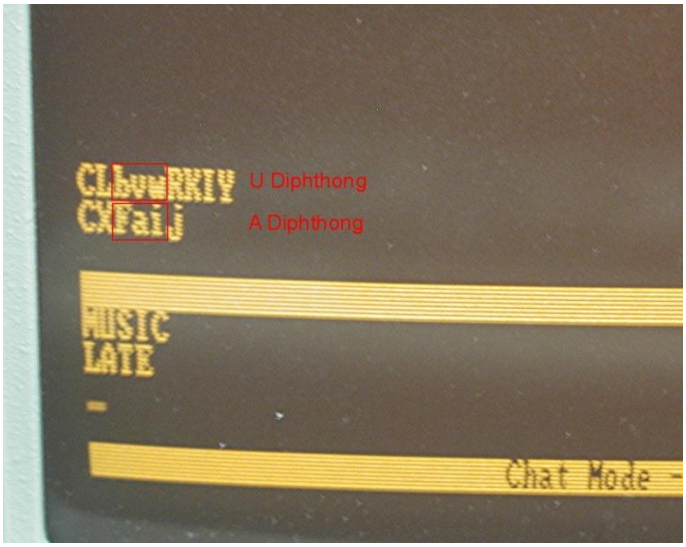
To operate this program, it is necessary to have the TNT working in Chat Mode with Procomm, use **esc** to get back to the main Procomm screen, shell out of Procomm to DOS using ALT-F4, to then run Sendbyte.exe. Then type Exit to return to Procomm and ALT-O to return to Chat mode. If say 1B 14 1B 11 is sent out, this switches off the echo of the typed text and switches on the PSEND switch, which returns the phoneme codes to the Procomm remote area on the Chat screen:



Phoneme symbols & Phoneme Hex code & Diphthongs:

The returned translated typed text data is in a Diphthong format. This had me a little baffled initially as I was expecting that the phoneme data would probably be in its Phoneme symbol format.

A *Monophthong* is a typical single vowel sound. A *Diphthong* is a combination of two vowels where one leads into another to create a single sound.



The Diphthong for “U” sound in Music is “bvw” and for “A” in Late is “Fai”.

The text to speech translator function of the TNT can be bypassed by preceding the string with “~” and terminating it with a “?” So the Diphthong strings above will be spoken if sent back to the TNT with these delimiters around them.

However, the idea of this project was to get the phoneme strings to send to the Hero Jr. Robot. These string are accepted in the phoneme symbol form and delivered to the robot when it is running Hero Jr. BASIC (A version of Wintek tiny Basic)

Further experimentation revealed how the Diphthong strings relate to the hex code and the phoneme symbols:

The word “ready” for example is returned in diphthong format as “CkB@^i”. The leading “C” appears at the start of and statement or string. Dropping the C off and translating this ASCII string to Hex code gives: 6B 42 40 5E 69

Looking at Votrax’s Phoneme chart in their Speech Dictionary that they published for the SC-01A IC, the phoneme symbols for “ready” are R, EH1, EH3, D ,Y . Converting those to phoneme Hex code (from their provided chart) gives:

2B 02 00 1E 29 and aligning these with the hex code of the returned diphthongs:

6B 42 40 5E 69

These codes are essentially the same, except for a 40h offset.

Therefore it is possible with a translation to convert the returned diphthong format code into phoneme symbols. I am currently working on a software utility to do it.

This will mean I can type large amounts of text quickly and use that to program the Hero Jr. Robot, with the aid of the TNT.

Other Investigations:

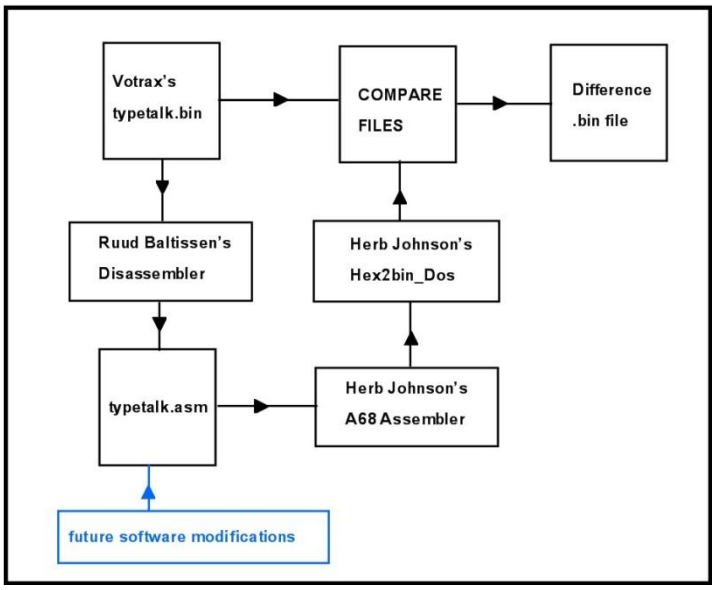
In the meantime, there have been some interesting developments. I had thought it would be good to learn to write assembly programs for the Motorola 6800 series CPU’s and do that on my vintage 5155 computer. Also it would be good, if possible, to disassemble Votrax’s TNT .bin file.

I sent the .bin file to Mr. Ruud Baltissen. He has designed a custom disassembler program which is very advanced. He kindly disassembled Votrax’s .bin file into the .asm or source file.

To test if this had worked, this .asm file was then assembled in an A68 cross assembler (provided by Mr. Herb Johnson, who has a website supporting vintage cross

assemblers). It worked. The regenerated .bin file was practically the same as the original .bin file except for a few addresses related to jumps.

The differences and how it came about, is summarized in the diagrams below to avoid any confusion. The difference file was very small:



THE DIFFERENCE FILE:

```
Bestanden TYPETALK.BIN en TYPETALK.ROM vergelijken
00000A41: 6C 7C
00000A4B: 6B 7B
00000A50: 6B 7B
00000A97: 6B 7B
00000AFF: 6A 7A
00000B0B: 6A 7A
00000B6C: 6C 7C
00000B7E: 6C 7C
00000BA5: 6C 7C
00000BAA: 6C 7C
00000C7D: 6D 7D
00000C8D: 6B 7B
00000C96: 6B 7B
00000CF5: 6B 7B
00000D24: 6D 7D
00000E07: 6F 7F
00000E10: 6E 7E
00000E27: 6F 7F
00000E2A: 6E 7E
00000E2D: 6F 7F
00000E30: 6F 7F
00000E33: 6F 7F
00000F2B: 6E 7E
```

The purpose behind this exercise is to be able to modify what would be the equivalent of Votrax's original source file, to alter the function of the TNT. This process can also be applied to any vintage 6800 computer ROM data setup, to help to acquire the original source codes and subsequently modify or repair a vintage ROM resident program.

****