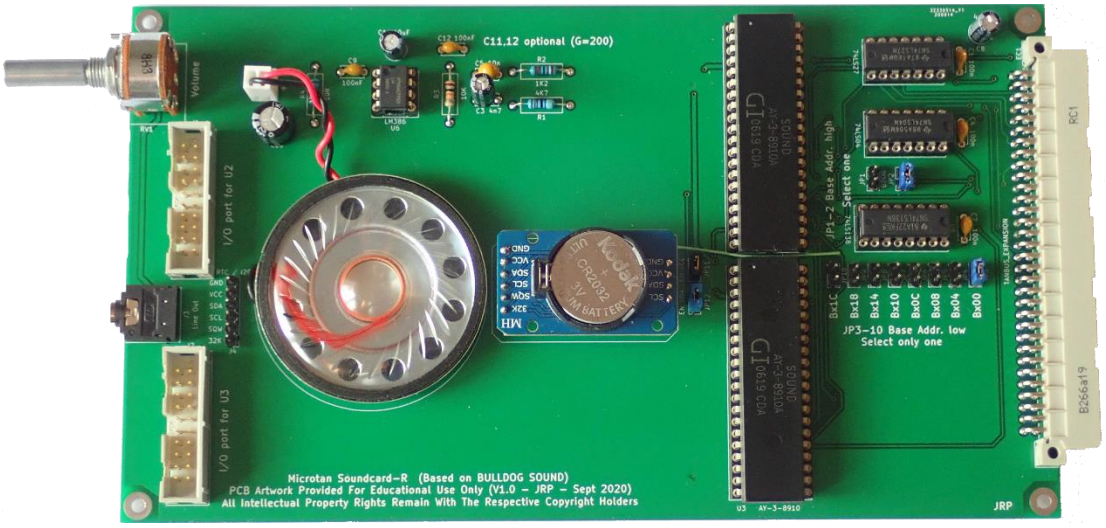
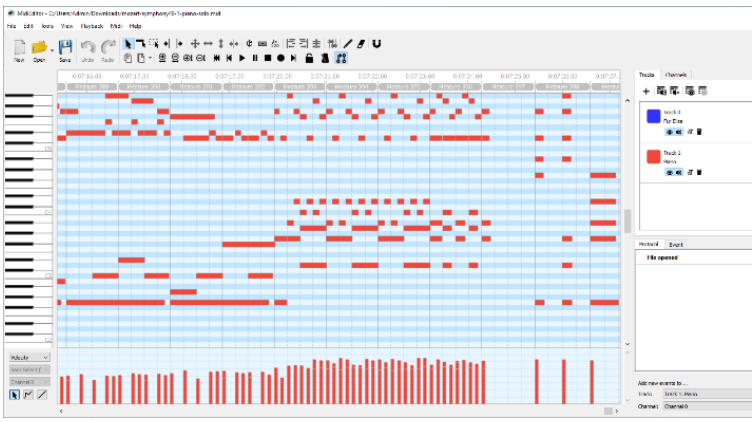
ADVENTURES WITH A MICROTAN-R SOUNDCARD





ISSUE 1.0
James Price 2021

1 INTRODUCTION 2
2 AVAILABLE ITEMS 2
3 INITIAL SYSTEM DESIGN 3
3.1 MEMORY MAP 3
4 C COMPILER 4
4.1 CC65 INSTALATION ON A WINDOWS PC 4
4.2 BUILDING THE CORE LIBRARY FILES 5
4.3 EXAMPLE C PROGRAM 6
5 HIGH SPEED PC TO MICROTAN DOWNLOAD 7
6 THE MICROTAN AS A MIDI PLAYER 10
6.1 MIDI NOTE EVENTS 11
6.2 MIDI KEY NUMBER TO NOTE FREQUENCY 11
6.3 KEEPING TRACK OF NOTES 13
6.4 BUILDING THE MICROTAN MIDI APPLICATION 14
6.5 PUTTING IT ALL TOGETHER 15
7 FUTURE ENHANCEMENTS 18
8 HARDWARE IMPROVEMENTS 18
9 AY-3-8910 *or is it*? 20
10 APPENDIX 22
10.1 MT65rom FILES 22
10.2 HELLOWORLD FILES 42
10.3 HIGH SPEED LOADER 45
10.4 MIDI PLAYER FILES 47
11 REFERENCES 56
12 DOCUMENT HISTORY 57

1 INTRODUCTION

The MICROTAN–R Sound Card (MSC) is a modern replacement for the Bulldog sound card that was produced in the 1980's. It provides for six individual tones (voices) plus a noise channel and an envelope shaper for the audio output. The active devices used by the card are the ubiquitous (for the time) General Instrument AY-3-8910 chips.

This document details the various steps that I took to investigate what could be done to make the MSC produce some interesting tunes. My main motivation was to try and use the MSC as a MIDI instrument so that I could get a MIDI sequencer to 'play' some recognisable tunes with that distinctive square wave video game timbre that the AY-3-8910 was (in)famous for in my youth.

I also wanted to investigate the feasibility of using the 'C' programming language on my Microtan-R system as I suspected that BASIC was just not going to be fast enough also I have some experience with C and there is a huge library of software 'out there' to make use of.

I have no musical knowledge to speak off (as will become apparent) so I have had to gather information about MIDI protocols and note frequencies, keys, octaves and other musical terms as I have gone along - so apologies in advance for any misuse of musical terms that I may make.

I have really written this document as an aide-mémoire for myself, if others find the information within it useful, that's a bonus.

Details of the Microtan Soundcard-R can be found on the excellent Microtan website at: - http://www.microtan.ukpc.net.

2 AVAILABLE ITEMS

The following list itemises the various hardware items that I have at my disposal.

- A Microtan-R CPU
- A Tanex-Plus (48K RAM)
- Two 'Production quality' Soundcard-Rs with 2 AY-3-8910 chips
- A 'Prototype' Soundcard-R with 2 AY-3-8910 chips
- A Windows 10 PC
- An Arduino Micro microcontroller.
- A couple of Bluetooth type amplified speakers with 3.5mm input jacks.
- Generic 8 port USB 24Mhz logic Analyser & USB to Serial converter dongle
- A soldering iron, solder, wires, various connectors etc.
- The Microtan boards are hosted on a Microtan-R backplane.

The following list details the software that I have available for the windows 10 PC:-

Tera Term	A Terminal Emulator.	**SigRok Pulse View**	A Logic analyser app.
TextPad	A Text Editor.	**MidiEditor**	A MIDI editor and sequencer.
Arduino Development IDE 1.8.12.			

All the software is freeware or has some kind of evaluation licence.

3 INITIAL SYSTEM DESIGN

My initial idea is to somehow couple the MIDI sequencer software to the Microtan system so that readily available MIDI tune files could be played through the Soundcard-R by a program resident on the Microtan.

i.e.

When working with my Microtan system I generally have it connected to the Windows PC via a USB to RS-232 Serial converter that plugs into the UART on the Tanex-Plus card via connector P7. I use Tera Term to communicate with the Microtan. This setup works quite well allowing me to interact with Microtan's monitor (e.g. Tugbug) and BASIC as well as capture any output as well as 'paste in' long text sequences copied from a text editor. My normal mode of working is to develop program code on the Windows PC using TextPad and then do a copy and paste operation to the Microtan via Tera Term.

3.1 MEMORY MAP

Table 1. below shows the current memory map for my Microtan-R system.

PCB	ADDRESS RANGE			TYPE	DESCRIPTION
MT65-R	F800	2K	FFFF	EPROM	MONITOR
TANEX-PLUS	F000	2K	F7FF		X-BUG
	C000	12K	EFFF		USER RAM / EPROM
MT65-R	BFF0	16b	BFFF	SYSTEM IO	MT65 IO
TANEX-PLUS	BFE0	16b	BFEF		6522 VIA
TANEX-PLUS	BFD0	4b	BFD3		UART
TANEX-PLUS	BFC0	16b	BFCF		6522 VIA
SOUNDCARD-R 3	BC08	4b	BC0A	USER IO	SOUNDCARD
SOUNDCARD-R 2	BC04	4b	BC07		SOUNDCARD
SOUNDCARD-R 1	BC00	4b	BC03		SOUNDCARD
TANEX-PLUS	A000	6K	BBFF	RAM	USER RAM
HRG	8000	8K	9FFF		HIGH RES GRAPHICS
TANEX-PLUS	400	30K	7FFF		USER RAM
MT65-R	200	512b	3FF		VDU RAM
	100	256b	1FF		CPU STACK
	0	256b	FF		ZERO PAGE

Table 1. My Microtan System Memory Map.

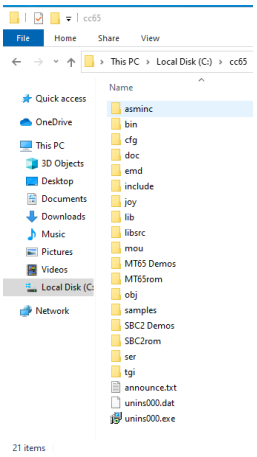
4 C COMPILER

There are several compilers that could possibly be used to produce code for the Microtan-R, the common ones are listed at: - http://6502.org/tools/lang/. Of these I chose to look more closely at the CC65 variant, mainly because someone by the name of Daryl Ricter has done most of the heavy lifting in getting the compiler ported for their 6502 Single Board Computer system which bares many similarities with the Microtan. In my opinion, a most useful contribution by Daryl is the construction of a couple of Windows compatible batch files that remove the need to get '**make**' working on my Windows PC. The documentation provided by the CC65 developers is a bit sketchy and the work done by Daryl speeded up the process of getting a C program compiled for, and running on my Microtan considerably. Nice one Daryl!

4.1 CC65 INSTALATION ON A WINDOWS PC

To start with, get Daryl's CC65 support files from:- https://sbc.rictor.org/download/cc65sbc2.zip (make sure you have a good look around Daryl's interesting web site while you are there) and follow the instructions that can be found within the .zip file to get CC65 installed. I believe this will install a slightly older version of the compiler suite than can be obtained from the CC65 developers (see: - https://cc65.github.io/) but no matter, Daryl's version will do nicely for the moment.

You should now have the CC65 compiler installed on your C drive with a folder layout similar to that shown if Figure 1.



Note:- you will not have the MT65 Demos or the MT65rom folders yet

Figure 1. CC65 Folder Layout

Next, create the two specific Microtan folders – **MT65 Demos** and **MT65rom** and copy the contents of the **SBC2 Demos** and **SBCrom** folders respectively into them.

In the MT65rom folder - rename the files as shown in table 2.

Original Name	New name
sbc.h	mt65.h
sbc2.inc	mt65.inc
sbc2rom.cfg	mt65rom.cfg
makesbc2.bat	makemt65.bat
copysbc2.bat	copymt65.bat

Table 2. Names for Microtan rom files.

Now, we need to edit various files that the compiler will use to place all the various code and data elements in the proper places to suit the Microtan Memory map along with some specific C routines coded especially for the Microtan. Using a text editor - edit the following files listed in Table 3. to be similar to those shown in Appendix. Note: this configuration is for my Microtan's memory layout (see Memory Map) different memory layouts may require different config file contents.

C:\cc65\MT65rom\ File	Description / Notes
_irq_proc.s	Interrupt routine.
_scrsize.s	Screen size. Not used for now.
cgetc.s	Gets input char from Microtan monitor.
clrscr.s	Clear the Microtan-R VDU screen (200 to 3FF).
color.s	Not used .
conio.h	Console IO header file.
copymt65.bat	Copies MT65 ROM files correct location ready to build libraries.
cputc.s	Character output routine for Microtan-R.
crt0.s	'C' RunTime. Start up code, gets Microtan ready to execute 'C' code.
ctype.s	No change. Ignore for now.
filedes.inc	No change. Ignore for now.
gotoxy.s	Ignore for now. Cursor location routine.
kbhit.s	'C' kbhit() function for Microtan.
makemt65.bat	Substuitutes for a 'make' file to build the 'C' libraries etc.
mt65.h	Microtan constants.
mt65.inc	Microtan constants.
mt65rom.cfg	Tells the compiler where to put the code and data.
oserrlist.s	No change.
oserror.s	No change.
peek.s	C version of BASIC command.
poke.s	C version of BASIC command.
randomize.s	'Seeds' the random number generator.
read.s	'C' read function. Might need converting for Microtan-R later on.
rwcommon.s	No change
write.s	'C' write function. Might need converting for Microtan-R later on.

Table 3. C:\cc65\MT65rom\files

4.2 BUILDING THE CORE LIBRARY FILES

Having completed the compiler installation, we can now perform an initial build of the libraries to suit the Microtan-R system.

Open a Windows command window in the **`C:\cc65\MT65rom\`**folder and execute the **`copymt65.bat`** file as shown in figure 2. The warning from **`ar65.exe`** is normal and can be ignored.

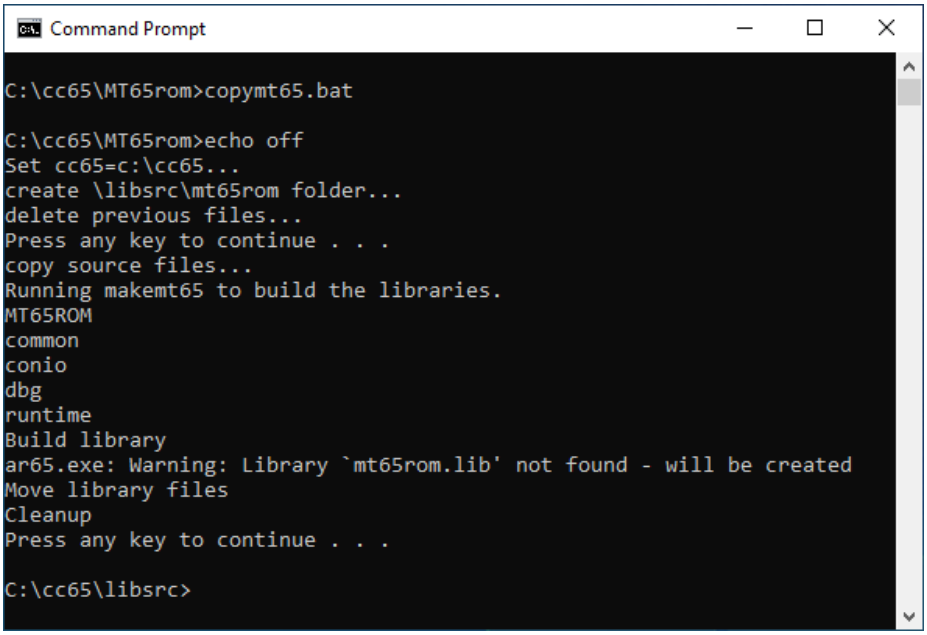


Figure 2 Building the libraries.

4.3 EXAMPLE C PROGRAM

With the libraries built we can now write a simple 'hello world' type program in 'C' and get it running on the Microtan-R.

In the **C:\cc65\MT65 Demos** folder create a **helloWorld** folder edit the **helloWorld.c** and **make.bat** files to match the similarly named files listed in the Appendix.

For convenience I have coded a binary to ASCII hex converter program in Java – **bin2txt.java** (see the Appendix for the Java source) this needs to be placed in to the **C:\cc65\MT65 Demos** folder. The Java runtime also needs to be installed on the Windows PC (I used Java version 15.0.1).

This converter will come in handy later when we need to get the code bytes onto the Microtan-R.

To build the **hello.c** program:- open a Windows command window in the **C:\cc65\MT65 Demos\helloWorld** folder and execute the make.bat batch file.

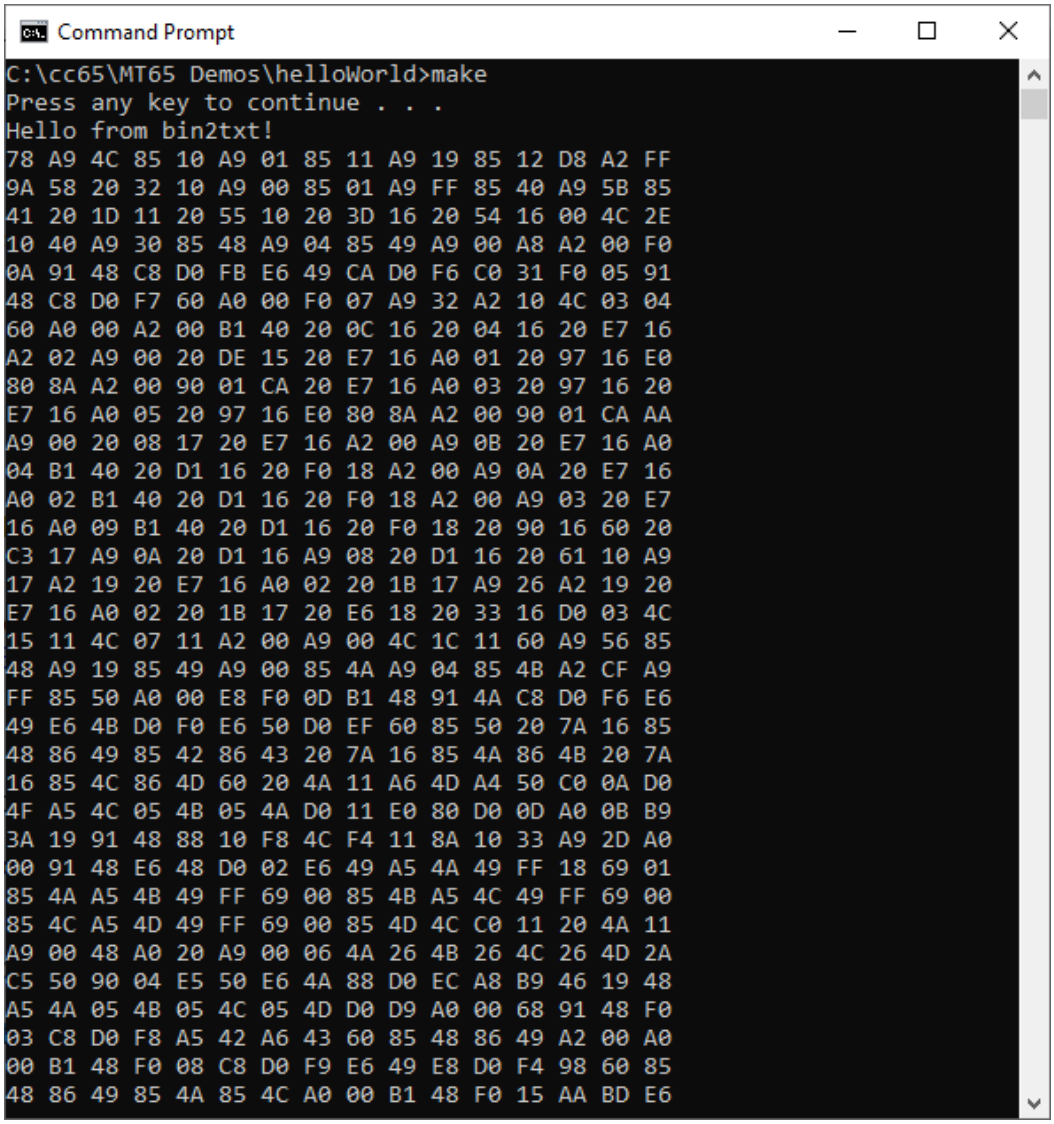


Figure 3. Building the hello.c program

The output of the build is the helloWorld.65b file, this is processed by the **bin2text.java** file to give the ASCII bytes shown in the command window. If you were to type these byte codes into the Microtan-R using the monitor's 'M' command and then issued the monitor's **G1000** command the C program would run. However, typing in all those codes (potentially 30+Kbytes) is not really practical.

What to do?

Well there may be various ways in which we can get the bytes into the Microtan's memory one of which is explored in the SOUNDCARD-R manual(see:- http://www.microtan.ukpc.net/) that involves 'auto typing' via the Tera Term terminal emulator. This 'auto typing' is OK for smallish programs but is still slows down the edit – compile- load- test cycle too much. Can we get the data from the PC to the Microtan in a faster fashion?

Yes, we can, by using an Arduino board to take the data from the PC via USB at a much higher baud rate than that supported by the Microtan's 6551 UART chip which is limited to 19200 Baud. The Arduino can support much faster baud rates - 200000 baud or more!

In order to get the data from the Arduino to the Microtan we can use one of the 6522 parallel ports available on the TANEX-PLUS card – this should be able to transfer data many times faster than the 6551 UART. The details of this scheme are explored in the next section.

5 HIGH SPEED PC TO MICROTAN DOWNLOAD

In order to get my compiled code into the Microtan in as short a time as I could, I devised the following scheme: -

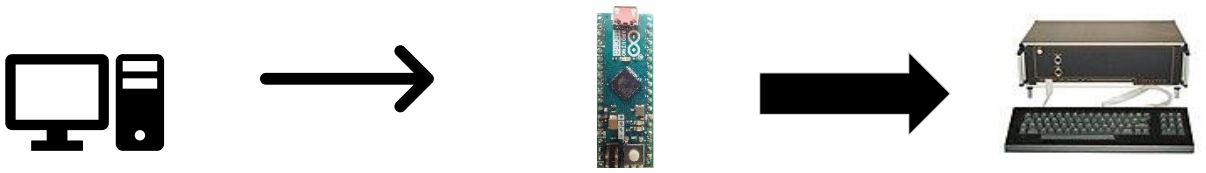


Figure 4 High speed transfer scheme

So, we need an Arduino, (I chose an Arduino Micro) some code on the Arduino to receive the bytes in a serial fashion from the PC and to then transmit them in a parallel fashion to the Microtan's 6522 'A' port, also we need some code running on the Microtan to receive the bytes from the Arduino via the 6522 port and place them into the Microtan's RAM. We need to ensure that the code running on the Microtan is small in size (we have to type this in manually so size is paramount) and needs to reside 'out of the way' in RAM so that it won't get overwritten by the incoming code bytes.

For the Arduino code see the program **`mt65_loader.ino`** in the appendix and for the Microtan code see the **`mt65_loader.asm`** section in the appendix.

A small interface cable is needed to couple the i/o ports of the Arduino to the 6522 'A' port on the TANEX-PLUS card.

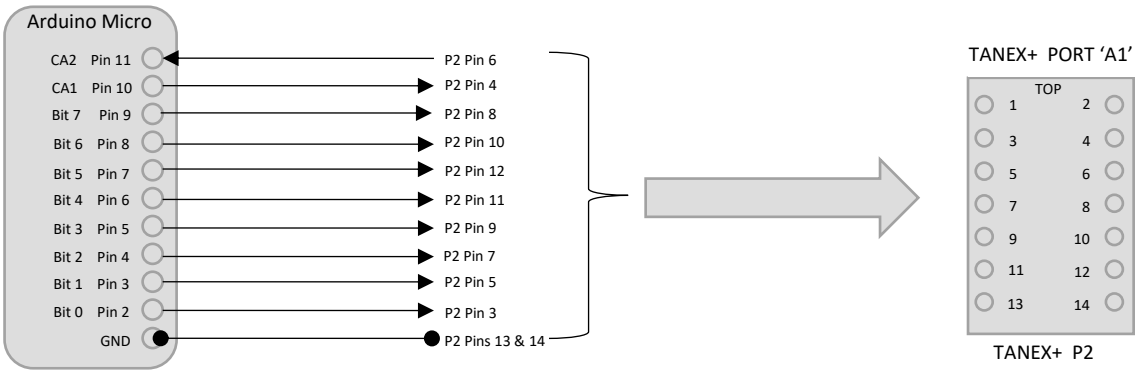


Figure 5. Arduino to Microtan cable connections.

The Arduino I/O pins 2 to 9 form the 8-bit parallel bus and should be connected to PORT 'A' of the 1st 6522 (U19) with Arduino pin 2 joined to PORT A bit 0, Arduino pin 3 joined to PORT A bit 1 and so on. The two handshake signals from the Arduino (pin 10 & 11) should be connected to PORT A CA1 and CA2 respectively.

For my cable I used a small piece of Veroboard, a length of ribbon cable, some header sockets to connect the Veroboard to the Arduino and some header pins to allow for connection to a logic analyser so that I could check what was happening.

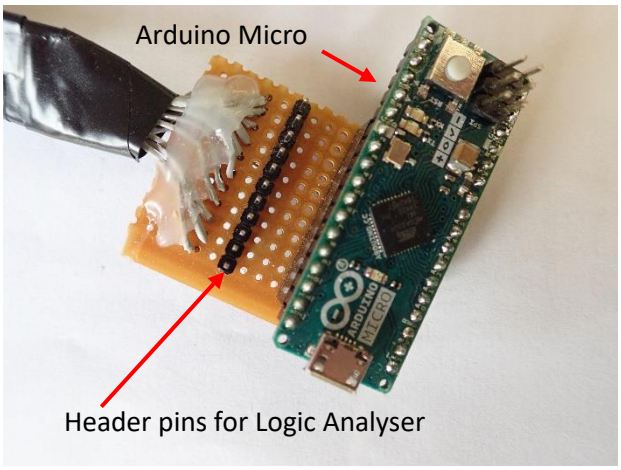


Figure 6. Arduino Micro end of the cable

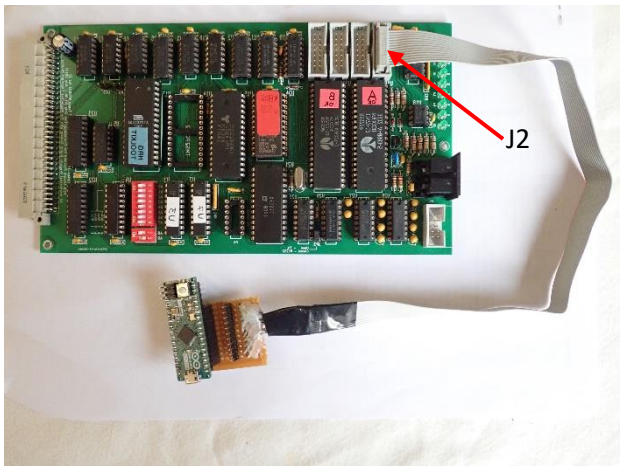


Figure 7. Cable assembly in situ on TANEX- PLUS.

The Microtan's loader code is purposely kept very simple and therefore very small (you may have to type this in manually, so small is good). All it does is read a byte from the 6522-port using the CA handshake signals to arbitrate the transfer and place the received byte into memory, increment the memory pointer, print the memory pointer and then repeat. There is no start / stop sequencing or checksum checking. I have located the loader at **$BB00** which is near the top of user RAM in my system and therefore about as out of the way as I can get it. I have found that for the most part I only have to type this code in once per Microtan power cycle, it seems to survive resets and errant compiled programs more often than not. In the future I would like to add it to the Microtan's monitor to avoid typing it in at all.

On the Arduino side, the code reads the ASCII coded bytes received from the PC, reformats them into single binary bytes and outputs them to the I/O pins to be 'clocked' into the Microtan using a couple of handshake signals.

To get the bytes from the PC to the Microtan the following sequence should be followed: -

1. Turn the Microtan OFF.
2. Connect the Arduino to the Microtan's TANEX port A1 (socket P2).
3. Connect the Arduino to the PC via a USB cable.
4. Turn the Microtan ON.
5. Using the Arduino IDE – Download & launch the Arduino Loader program. Also, using the IDE open a Serial Monitor window.
6. Using the 'M' command or otherwise, load the Microtan's loader program starting at address **$BB00**.
7. Build the helloWorld.c program and copy the ASCII code bytes from the command window into the paste buffer. Make sure you select all the bytes and then press 'Control – C'.

8. Reset the Microtan and then execute the loader by typing '**GBB00**'. If the Microtan prints out an address at this point – reset the Microtan and re-issue the '**GBB00**' command- we want the Microtan to just sit there waiting for the first byte.
9. In the Arduino IDE's Serial Monitor window's input box (at the top of the window) paste in the bytes you have in the PC's paste buffer and then press the 'Send' button. The Arduino should now start sending the bytes and you should see the Microtan printing a stream of consecutive addresses. If this does not happen then repeat the process from step 5. Assuming the transfer proceeds OK - after a few seconds the Microtan should stop printing addresses and you can regain control by resetting the Microtan.
10. As a sanity check – list the contents of the Microtan's memory from Address **$1000** for a few bytes and check that they match the ASCII sequence that you cut and pasted in the previous step. If there are incorrect bytes – close & reopen the Arduino IDE and start again.
11. If all is well you can execute the helloWorld program on the Microtan by issuing the '**G1000**' command. The program should run, print the "HelloWorld" message on the Microtan's VDU and then wait for you to input any key before exiting with a register dump printed.

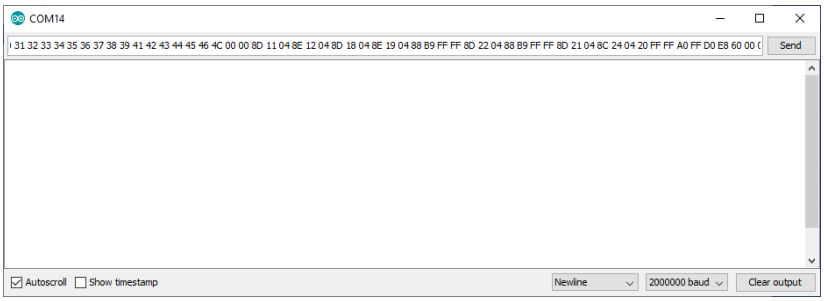


Figure 8. Arduino IDE Terminal window - helloWorld program about to be sent....

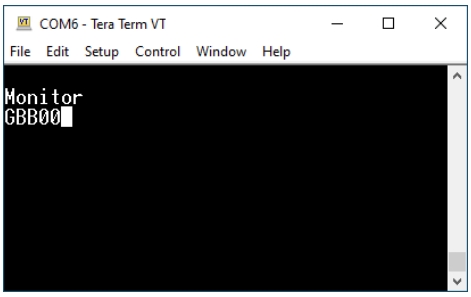


Figure 9. Microtan Correctly waiting for data.

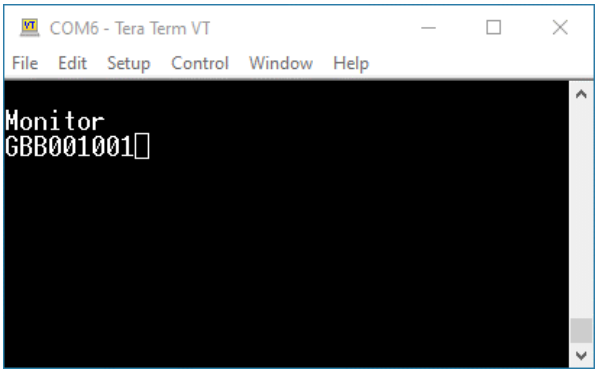


Figure 10. Not correct – Reset & try again.

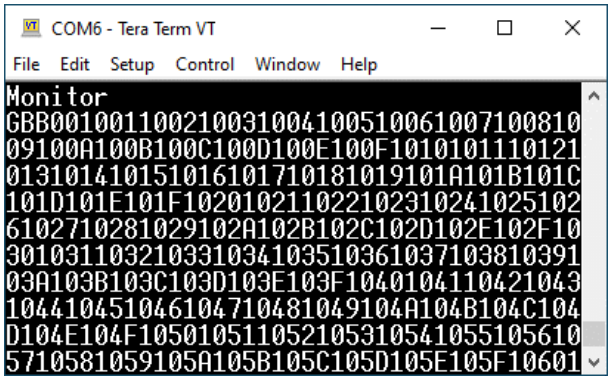


Figure 11. Data transfer in progress...

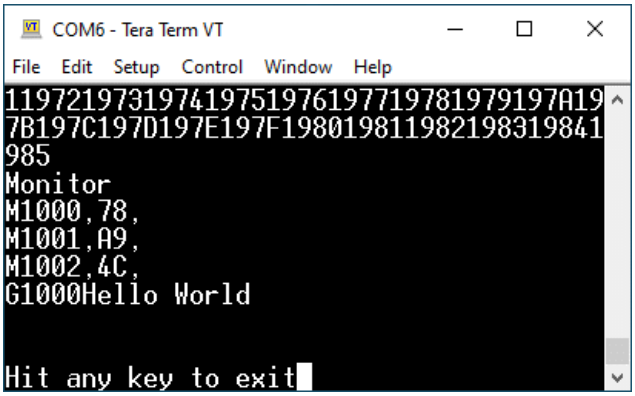


Figure 12. Output in Tera Term. After transfer finished - hit reset, check a few bytes and run…

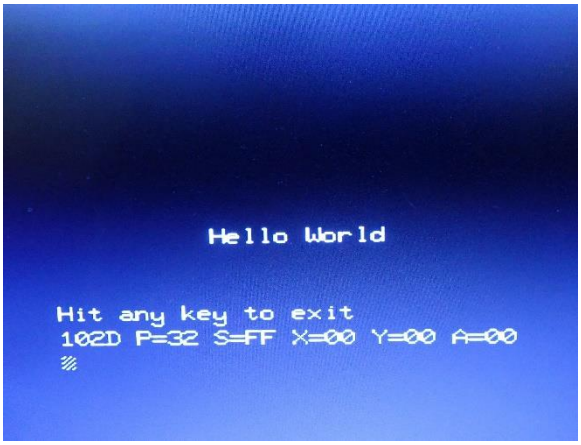


Figure 13. Output on Microtan VDU after hitting a key to exit.

6 THE MICROTAN AS A MIDI PLAYER

So, now we have the C compiler and high-speed transfer working it is time to move on to the main event – making the Microtan play notes sent from a MIDI sequencer. The MIDI sequencer I have chosen to use is the Windows hosted MidiEditor program by Markus Schwenk (See https://midieditor.org/). I'm not all that familiar with MIDI sequencer software but this seems to be quite a capable free application and continued use should probably result in a donation to the author.

The MIDI software can take standard MIDI files (tunes), display the component notes in a 'piano roll' editor and play the notes via a connected MIDI instrument. The MIDI hardware interface traditionally comprises of a serial interface running at 31K Baud via a set of DIN connectors.

The Microtan does not have a suitable MIDI connection but we can leverage the High-Speed interface described earlier. We can do this because fortunately the Arduino Micro can act as a USB MIDI end point (Note: - not all Arduino variants can do this), will be recognised by the MidiEditor software as such and it has a USB MIDI support library available.

When the MidiEditor plays the MIDI file it effectively sends note ON and note OFF events to the Arduino. We can write a simple program for the Arduino that can read these note events, reformat them to be suitable for the Microtan and send them via the 6522 Port A in a very similar fashion to the way the High-Speed interface does. Then we will need to write an application for the Microtan that receives the note events, determines what values to send to the Soundcard in order to produce audible sounds of the correct frequency. The Note events contain a 'velocity' term that can be used to modulate the volume of the note being played.

It transpires that the Arduino Micro can support the USB MIDI interface simultaneously with the standard Arduino Serial interface. This will be handy for debugging and allow for a single Arduino program that can support the High-Speed interface and the USB MIDI preventing unnecessary program swapping during development. Figure 14. shows the general arrangement.

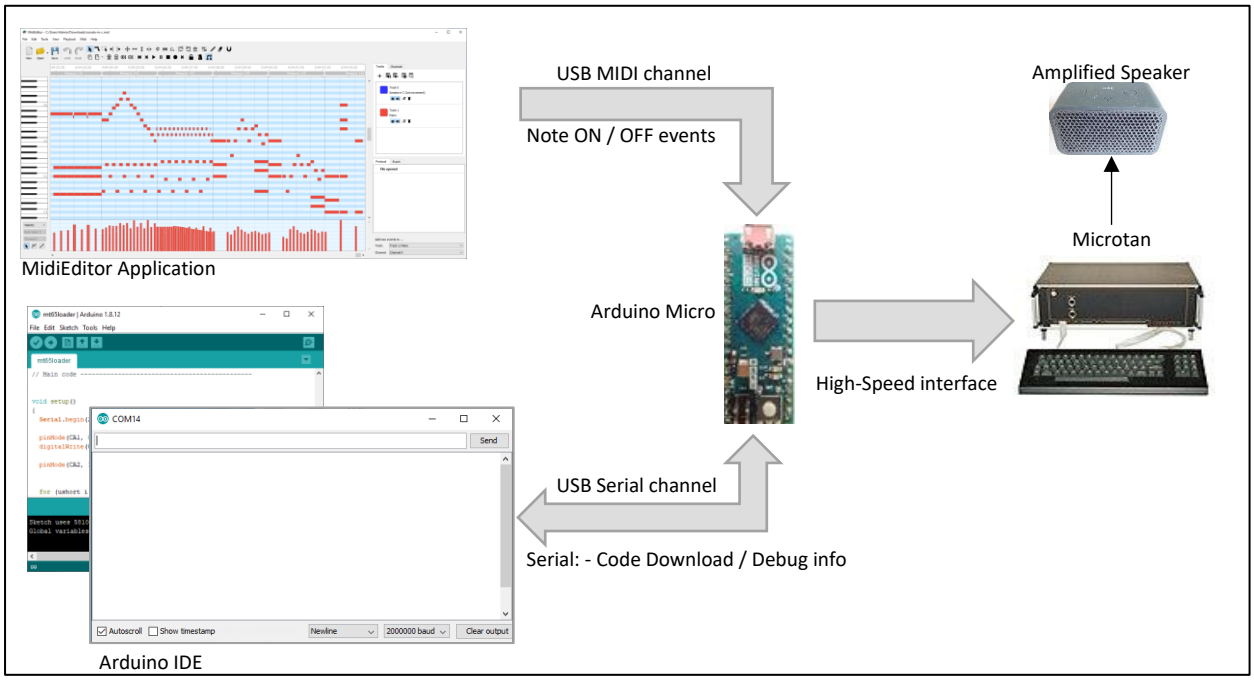


Figure 14. MIDI System setup.

6.1 MIDI NOTE EVENTS

The MIDI note events are arranged as follows in Table 4.

Header	Channel	Key Number	Velocity
0x9 = Note ON 0x8 = Note OFF	0 to 0xF	0 to 0x7F	0 to 7F

Table 4.

The Arduino MidiUSB library is able to deliver these note events to our code via the midiEventPacket structure so our code will easily be able to determine the contents of each note event.

The Arduino code to receive the events would look something like this: -

```
#include "MIDIUSB.h"

void loop()
{
  midiEventPacket_t rx = MidiUSB.read();

  if ( rx.header==0x9 ) // Note on
  {
    noteOn( rx.byte1 & 0xF, rx.byte2, rx.byte3 );
  }
  else if ( rx.header==0x8 ) // Note off
  {
    noteOff( rx.byte1 & 0xF, rx.byte2, rx.byte3 );
  }
```

Where byte1, byte2 and byte3 contain the Midi Channel, Midi key number and key velocity respectively.

The channel information is can be used to select the type of instrument voice to render the note information. Initially it will be ignored and just pure notes will be rendered, in the future it could be used to support for percussion events (drums, cymbals etc).

6.2 MIDI KEY NUMBER TO NOTE FREQUENCY

Each midi key is assigned a note frequency. It is normal (in Western music at least) to assign each key to a specific frequency.

After a bit of Googling I came up with a table of integer rounded frequencies - Table 5. The lowest octaves are not going to be very accurate as the error between the exact frequency and the integer representation will be quite large, but no matter - most tunes use the higher octaves anyway. (I presume). There are 12 notes to an octave and it seems usual to assign note A in octave 4 to 440 Hz. Higher and lower octaves are referenced to this which results in negative octave numbers at the low end!

In table 5. We can see that there are 11 octaves of 12 notes giving 132 notes in total. MIDI notes can only range from 0 to 127 i.e. 128 notes in total so we won't be able to play the top few notes but again I don't think this will be too limiting. So, MIDI note 0 will map to C_{-1} & MIDI note 127 will be G_9

In order to play a note, its frequency needs to be loaded into the AY-3-8910 chip on the Microtan Soundcard. The AY-3-8910 uses the System clock - 750Khz divided by 16 further divided by a 12-bit binary value to produce the signal frequency. The 12-bit divisor is held in two 8-bit registers. Bits 0-7 in the low register and bits 8-11 in the high register. We could calculate the note frequencies to divisor values at run time or just include a readymade lookup table, I chose to get the Microtan to earn its keep by doing the calculations at run time to build an array of AY-3-8910 divisor values from a table of tone frequencies.

Octave	Note											
	C	C#/Db	D	D#/Eb	E	F	F#/Gb	G	G#/Ab	A	A#/Bb	B
-1	8	9	9	10	10	11	12	13	13	14	14	15
0	16	17	18	19	20	22	23	24	26	27	29	31
1	33	35	37	39	41	44	46	49	52	55	58	62
2	65	69	73	77	82	87	92	98	104	110	117	123
3	131	139	147	156	165	175	185	196	208	220	233	247
4	262	277	294	311	330	349	370	392	415	440	466	494
5	523	554	587	622	659	698	740	784	831	880	932	988
6	1046	1109	1175	1245	1319	1397	1480	1568	1661	1760	1865	1976
7	2093	2217	2349	2489	2637	2794	2960	3136	3322	3520	3729	3951
8	4186	4435	4699	4978	5274	5588	5920	6272	6645	7040	7459	7902
9	8372	8870	9397	9956	10548	11175	11840	12544	13290	14080	14917	15804

Table 5. Midi Note frequencies in Hz.

Example calculation for note A in the fourth octave: -

A_4 = 440 Hz: Microtan Clock = 750000Hz: AY-3-8910 Pre-divisor = 16

AY-3-8910 FClock = $\frac{750000}{16} = 46875$

AY-3-8910 divisor value = $Int(\frac{46875}{440}) = 106$

AY-3-8910 High Reg. = $Int\left(\frac{106}{256}\right) = \mathbf{0}$ **(0x0)**

AY-3-8910 Low Reg. = $Int(106 - (High\ Reg * 256))$

= $Int(106 - (0 * 256)) = \mathbf{106}$ **(0x6A)**

The code to do the calculations for all 128 MIDI keys might look something like: -

```
#define FCLOCK (750000 / 16)
for (i=0; i<=127; ++i)
{
        freq = FCLOCK/noteFreq[i];
        freqHigh[i] = freq/256;
        freqLow[i]     = freq - (freqHigh[i]*256);
}
```

6.3 KEEPING TRACK OF NOTES

Having determined how to convert MIDI note values to AY-3-8910 register values we now need to consider how to coordinate the notes that are playing.

The MIDI note events will either be ON or OFF events and although we could get the Microtan to play notes of a defined duration (like a real piano would) and thus ignore the OFF event I have decided to honour the OFF event in the Microtan code (at least initially).

The Microtan Soundcard can support the playing of up to six simultaneous notes (voices), three per AY-3-8910 chip and for each incoming note ON event we need to check that there is a free voice available, if there is - then we need to record which voice is going to be used and program the AY-3-8910 frequency and amplitude registers appropriately. If there are no free voices available then the simplest course of action is just to ignore the note ON event. More complex ways of managing the recorded voices could be devised including 'first in first out' or channel priority schemes but we'll stick with the simple method to begin with.

When a MIDI note OFF event is received the Microtan needs to search for the voice being used to play that particular note, free up the recorded voice and set the appropriate AY-3-8910 amplitude register to zero. Again, if no corresponding note has been allocated a voice we can just ignore the note OFF event.

So, how many simultaneous notes are possible? Well, each Soundcard can provide six voices if both AY-3-8910 chips are present and there are potentially sixteen I/O base addresses available for a total of 6 x 16 = 84 voices! The main limitation is going to be the number of spare backplane slots available. In my system with the Microtan-R motherboard backplane, I have four free slots of which three are occupied by fully populated Soundcards giving a total of eighteen voices.

However, most of the MIDI files I have downloaded from https://www.mfiles.co.uk/ sound quite good with six voices and some are ok with just three. Most of the 'classical' piano and 'rag-time' piano files sound quite acceptable, the more contemporary 'pop' tunes less so. It is not uncommon for midi files to allocate several channels to various different instruments but our initial simple minded Microtan application will just render everything as a square wave tone. The possibility exists to enhance the midi application to support percussion sounds and other instruments, indeed in the old days some amazing sonic effects were produced by the game programmers using the venerable AY-3-8910.

6.4 BUILDING THE MICROTAN MIDI APPLICATION

The C program that comprises the Microtan MIDI player can be found in the APPENDIX as **midi.c** along with its associated **make.bat** file. These files need to be added a **C:\cc65\MT Demos\midi** folder. The code can be compiled by opening a Windows command window in the a **C:\cc65\MT Demos\midi** folder and executing the make.bat file.

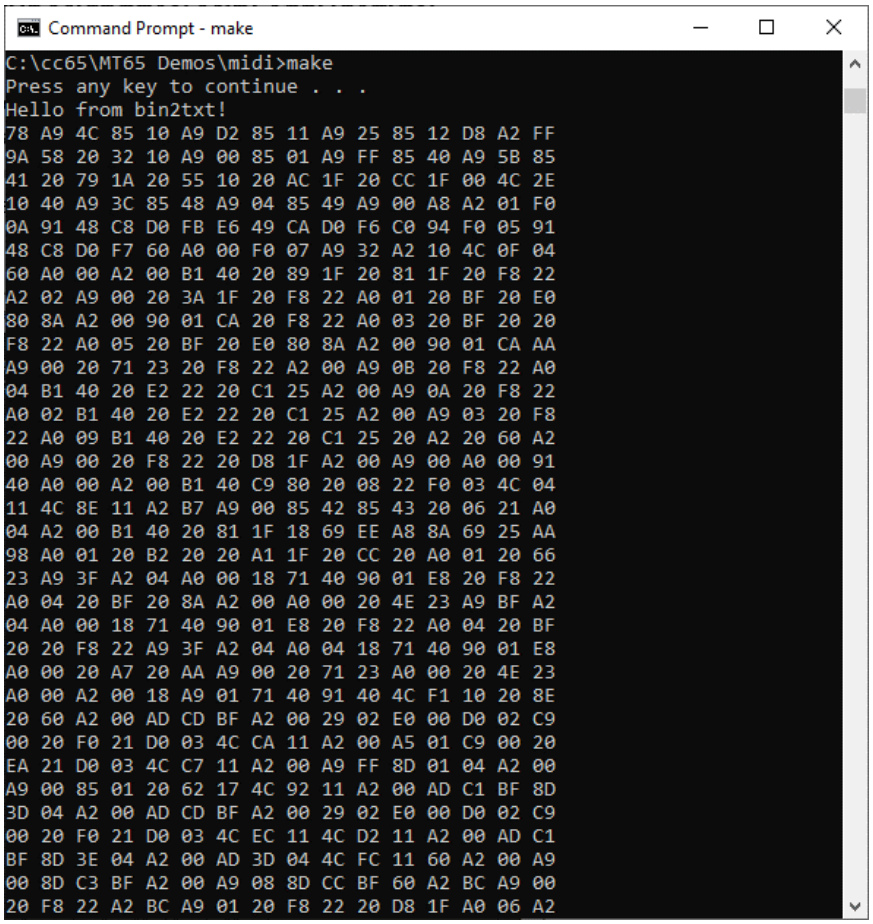


Figure 15. Building the Microtan Midi code.

6.5 PUTTING IT ALL TOGETHER

We should now have all the components we need to put together a functional MIDI player. The following steps assume that the High-speed code transfer cable described earlier is in place between the Arduino Micro and the Microtan and that the Arduino is connected to the PC via the USB cable.

1. Compile and upload the **midi_loader.ino** (see APPENDIX) file into the Arduino Micro. This file supports both the High-speed code transfer and the MidiUSB channel.
2. Type in or otherwise get the mt65loader program loaded and running on the Microtan.
3. Copy and paste the Microtan code bytes from the Windows Command session to the Arduino Serial Monitor window in a similar fashion to that described in the example C program (helloWorld) section.

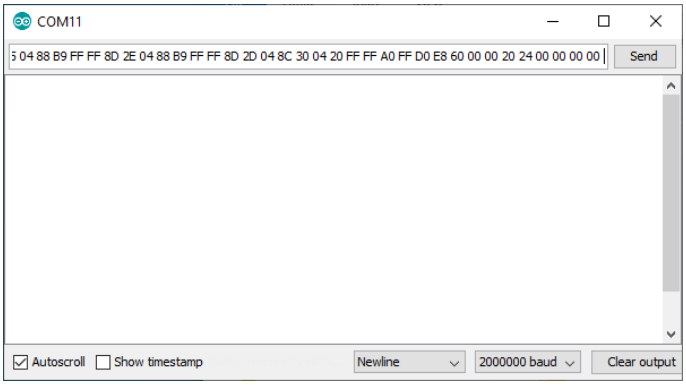
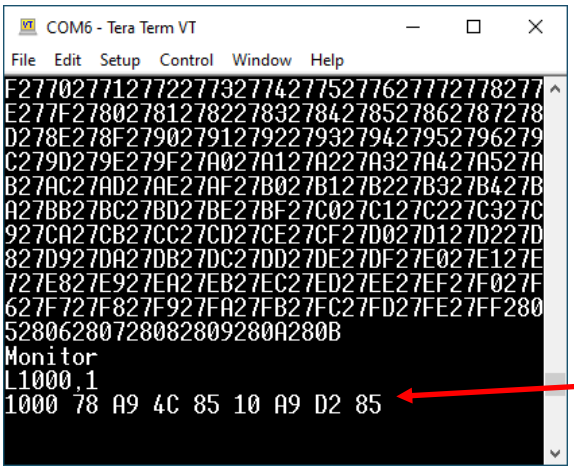


Figure 16. Ready to send...

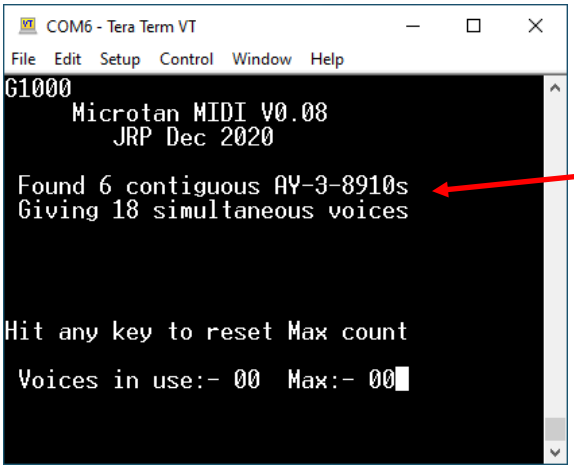
4. Press the send button in the Arduino Serial Monitor window and confirm that the code bytes are transferred correctly to the Microtan.



Cursory check of code bytes.

Figure 17. sent OK

Staying on the Microtan execute the midi program with **G1000**.



You may have different values here depending on how may Soundcards / chips are installed.

Figure 18. The Microtan MIDI App running...

5. Now fire up the **MidiEditor** program on the PC and ensure that the Arduino Micro is selected as the Midi output device in the **midi->settings** page.

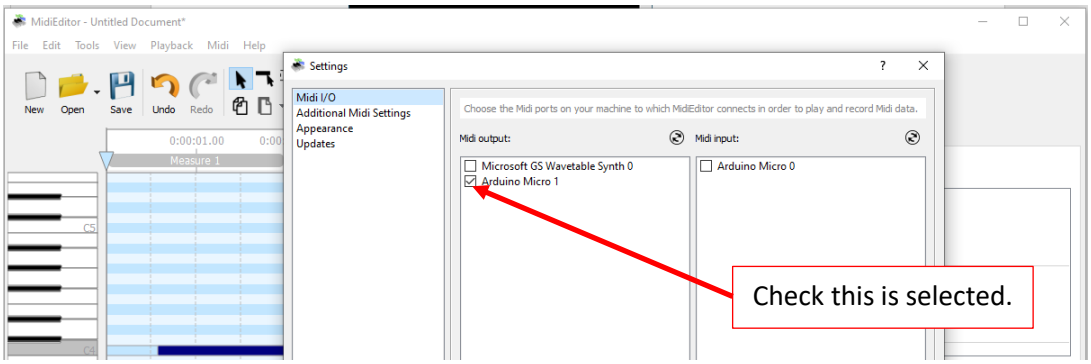


Figure 19. Making sure the Arduino is selected.

6. Next, we need a MIDI file. I found the following site has a good selection that can be tried out for free: - https://www.mfiles.co.uk/midi-files.htm.

   To start with, download the following: -https://www.mfiles.co.uk/downloads/The-Entertainer.mid and open it with the MidiEditor application.

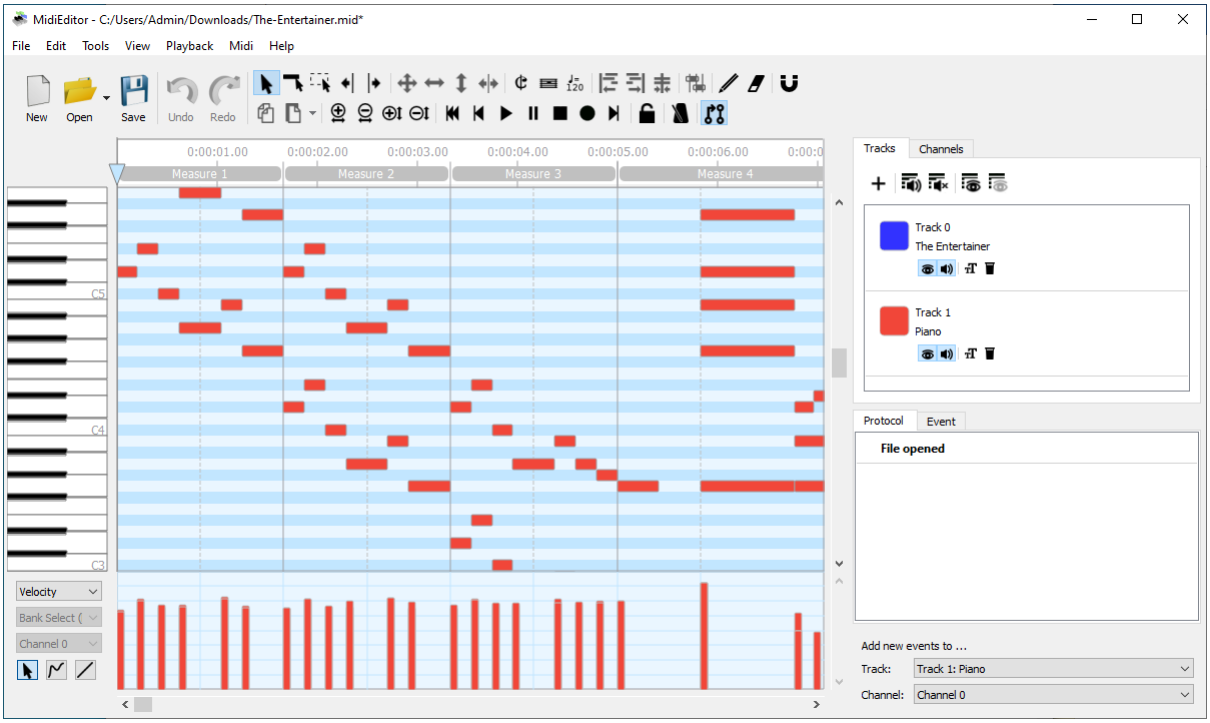


Figure 20. Digital Scott Joplin

7. Press the MidiEditor's play button and you should hear a familiar tune emanating from the Microtan. The quality of the rendition will be determined by the number of voices available. For this particular midi file a maximum of six voices are used.

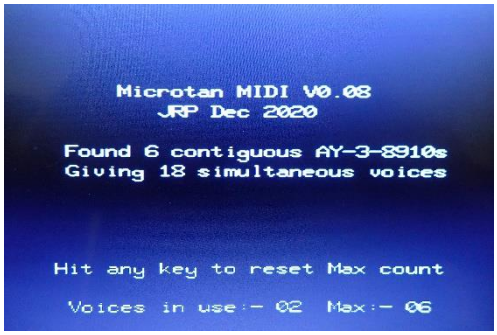


Figure 21. The Microtan playing Scott Joplin's 'The Entertainer'.

6.5.1 TROUBLE SHOOTING

If it all worked first time that's great but if not, don't worry, try the following steps: -

1. Make sure the volume is turned up on the output speaker.
2. Attach a USB logic analyser to the high-speed interface handshake pins to see if the note events are being sent to the Microtan.

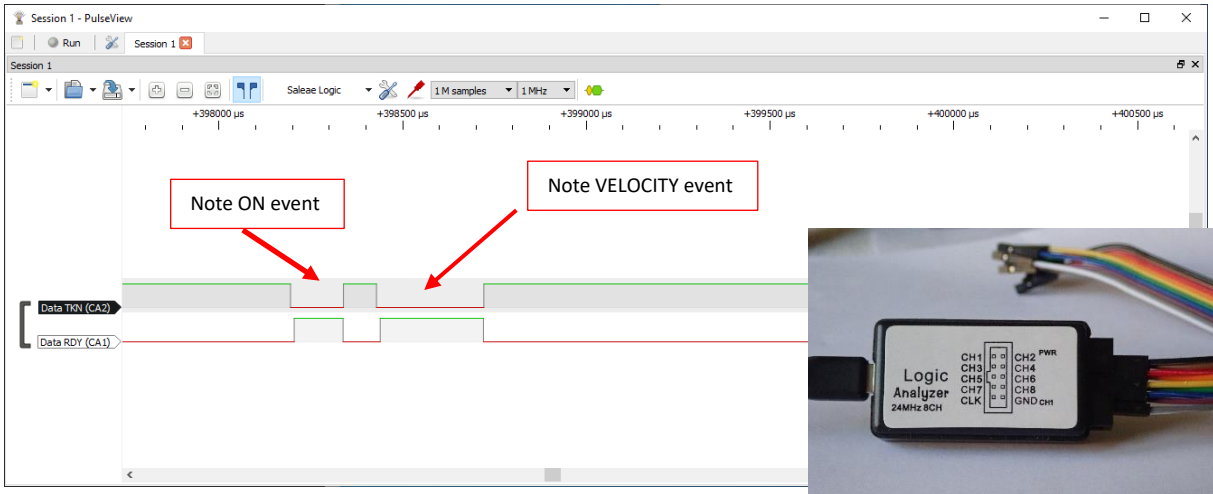


Figure 22. SigRok PulseView in action

Figure 23. USB logic analyser. Yours for < £20!

3. Reset everything! This includes the Arduino Micro and the Microtan.
4. Reload the Arduino IDE and the **midi_loader.ino** program.
5. Check that the MidiEditor is seeing the Arduino Micro as an output device – check the editor's settings page. Sometimes I found that deselecting and the reselecting the Arduino as the output device got things working.
6. If things are still not right then start adding debug print statements to the various software components: - **midi_loader.ino** and **midi.c** to try and determine what is blocking the operation of the system.

7 FUTURE ENHANCEMENTS

Here is a list in no particular order, of the improvements that could be made going forward: -

- Add support for percussion sounds.
- Support multi-channel operation.
- Draw the music on staves as it is being played.
- More robust high-speed interface protocol, error checks etc.
- Improve the hardware amplification on the Soundcard.
- More complex voice reuse algorithm.
- Look at other compilers. E.g. vbcc. 'Highly optimizing' apparently.

8 HARDWARE IMPROVEMENTS

During development and testing of this project it became obvious that the sound quality of the Soundcard-R could be better. When just a single voice was being used, as might be the case in games, the sound was reasonably acceptable but when playing multi voice midi files the sound seemed to degrade somewhat. Why? When I designed the Soundcard-R PCB I basically used the circuitry published in the GI AY-3-8910 datasheet and the original Bulldog soundcard documentation for the output stages. Indeed, after extensive Googling it seemed that all implementations of the AY-3-8910 followed essentially the same design.

After poking about with a scope, I deduced that one of the main problems seemed to be the very poor frequency response (or quality) of the onboard speaker – solution use the line out connection to drive a cheap (£20) Bluetooth type speaker that has a 3.5mm jack input. The sound quality did improve but still more '1960's miss-tuned transistor radio' than '1980's HiFi'.

Another problem centred around the input to the amplification stage. Here is my original implementation: -

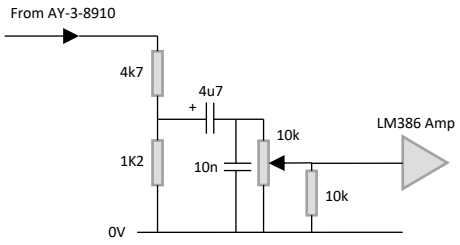


Figure 24. My original circuit

Changing this to: -

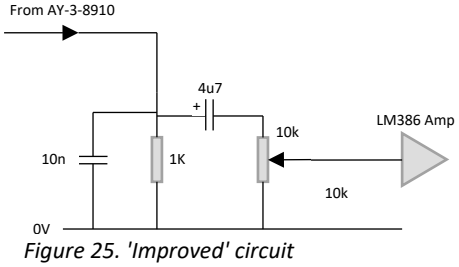


Figure 25. 'Improved' circuit

made a noticeable improvement to the 'cleanness' of the sound.

Another problem seemed to be that the voices were interfering with each other, in almost all the historical & contemporary designs I could find, the AY-3-8910 output channels (A, B & C) are directly connected together before feeding into the amplification stage (almost always an LM386). I did spot one developer that had used series resistors between the AY-3-8910 outputs and the amplification stage so I decided to try this approach by using 10K resistors. Unfortunately, this meant cutting some tracks on the Soundcard so that the resistors could be inserted. However, the effort was worthwhile with the improvement definitely being audible.

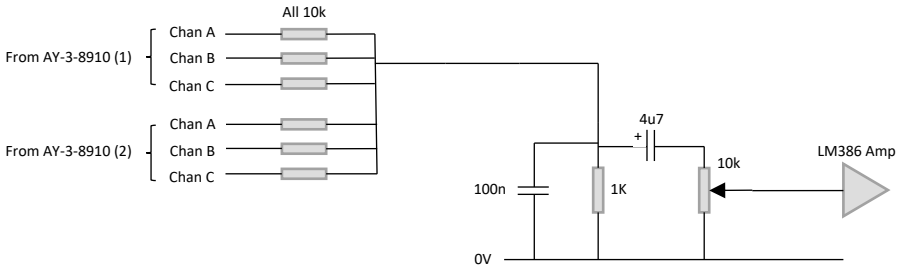


Figure 26. Better channel separation?

9 AY-3-8910 *or is it*?

The AY-3-8910 was initially manufactured by General Instrument in the late 70's and there were a few variants produced that had various pin counts these had slightly different feature sets, mainly the number of I/O ports but the sound generator side of things remained the same. The A-3-8910 that we are concerned with is the 40 pin DIP package. In the late 80's GI was spun off to Microchip Technology so it is possible to find AY-3-8910's marked GI or Microchip.

Also, a licensed version of the AY-3-8910 was manufactured by Yamaha that was virtually identical except that it had a clock divider pin and the envelope generator was modified slightly. Yamaha called their version of the chip the YM2149F and it was to all intents and purposes fully pin compatible with the GI AY-3-8910.

The AY-3-8910 chip is no longer manufactured and I suspect there are very few 'new old stock' devices left languishing in suppliers' warehouses. So most of the devices you can buy today are 'pulls', that is they are devices that have been removed from old scraped computers from the 80's.

So far so good. However, when trying to source an AY-3-8910 you will find that the price varies wildly with some suppliers quoting £30 or more per device while others are only asking a couple of pounds per device. So, if you want to build a Microtan Soundcard-R and fully populate it you could be looking at north of £60 just for the AY-3-8910 chips! This might make the Soundcard-R too expensive to consider or mean that you limit the card to one device, which would be OK for games but a little limiting for 'music' reproduction. Most of the low-priced devices are made available via 'no name' sellers on well-known on-line auction sites while the premium priced ones are usually found on local specialist sites. Well, just buy the devices from those asking a more reasonable price then, well OK but there's probably a reason why the cheap devices are so cheap.

I suspect that devices available from local suppliers with a reputation to maintain are genuine GI or Microchip devices and are 'new old stock' or good quality 'pulls' i.e. chips from sockets rather that unsoldered and cleaned up. The other suppliers may be providing genuine devices or they may be engaging in the practise of 're-marking'. 'Re-marking' is the dubious practise of sanding or scouring off the chips original marking and printing on a new designation. Why do this? One reason is to pass off one manufacturer's device as another (possibly more valuable or common) manufacturer's product. Another reason is just to place completely incompatible or different chips on to the market in the hope that most buyers who are just buying one or two devices at a time won't notice and/or won't be bothered to chase up what they think was just a defective chip.

What did I do? To begin with I bought two 'affordable' devices from a local (that is with a UK trading address) supplier listed on our favourite on-line auction site for a grand total of £8. The devices duly arrived properly packaged in anti-static tubes. The chips were obviously 'pulls' and were all identically marked as GI and just for good measure three devices had been supplied. The date code of *0619* looked a bit odd though, three 'pulls' all with the same date, a date that could not be correct at that. Hmm. Better test the devices in my prototype Soundcard-R. Would they work? Actually, yes, they did, two of the devices were fully functional and one had a 'dead' sound channel. But I had two working devices I had ordered so I'll claim that as a result.

Are the chips re-marked YM2149F devices though? How can you tell? Apparently, the re-marking process is not very stable and cleaning the top of the chip with a little acetone (nail polish remover) will reveal if the chip's marking is original or not. I used a cotton bud soaked in acetone and gave one of the AY-3-8910 chips a light swabbing- see figure 27. As you can see the black surface is easily removed. As a control I repeated the test on a known genuine chip (not an AY-3-8910) but a 'pulled' 6522 chip and this did not leave any black residue on the cotton bud.

So, it's looking like my AY-3-8910 chips are hiding something - what other checks can we do? After a little Googling it would appear that the YM2149F devices have fully provided read/write bits for all the chips registers even those bits that are not allocated any function and will read back the same value written to these bits, whereas the GI versions will always return '0' for the unimplemented bits. Using the Microtan's 'M' command I wrote various values to my device's registers and indeed all bits seemed to be fully read / write.

There is another check that I could do and that is to see if the clock divisor pin can be activated. If the device is a YM2194F device then bringing pin 26 (!SEL) low should halve the frequency of any tones produced, it's not clear what effect this will have on the AY-3-8910 but it will either do nothing or affect the chips operation in some other fashion. I tested the chip in my prototype card and I can confirm that the frequencies did indeed get halved. One other reported difference is that the YM2194F has an approx. 2-volt dc offset on the audio outputs, the AY-3-8910 has a much lower offset. I scoped my chips and they definitely had a 2-volt dc lift present. I do wonder if this has some interaction with my original output circuitry and was the cause of less than perfect sound quality. If I ever obtain a genuine AY-3-8910 I will re-instate the original circuit to check.

So, my chips are very likely re-marked YM2194F chips. Does it matter? Not in this case, at least the chips are compatible and I'm not left wondering if I've damaged them during the build process and they were affordable. Would I buy from the same source again? Well, yes, and I did, I bought another four AY-3-8910s from the same supplier, again they arrived in good time and again they all look like re-marked YM2194F devices but they are all fully functional. I'm guessing that the supply of YM2194F chips is far more plentiful than that of the AY-3-8910 thus making the re-marking game economic. I may have been lucky in this case but as is usual when buying cheap – caveat emptor!

Figure 27. Makeup?

Figure 28. Tattoo?

10 APPENDIX

10.1 MT65rom FILES

10.1.1 _irq_proc.s

```
;
; JRP 2020
;
; Microtan irq routine
;
;
;

    .export _irq_proc
    .export _irq_count


    .include   "mt65.inc"

.proc   _irq_proc

        inc _irq_count
        pha             ; save acc

        lda  U19_IFR    ; get 6522 interrupt flags
        bpl exit        ; quit if no interrupts to service
        rol             ; get Timer 1 flag in 'S' bit
        bpl exit        ; quit if not timer1 interrupt
        lda U19_IFR     ; get the flags back
        and #$C0        ; reset Timer 1 interrupt
        sta U19_IFR     ;

 exit:  pla
        rti


.endproc

.bss
_irq_count: .res    1
```

10.1.2 _scrsize.s

```
;
; Ullrich von Bassewitz, 26.10.2000
;
; Screen size variables
;
; Modified for SBC-2 by Daryl Rictor
; Modified for Microtan-R by JRP

    .export screensize

    .include   "mt65.inc"

.proc   screensize

    ldx #32
    ldy #16
    rts

.endproc
```

10.1.3 cgetc.s

```
;
; Ullrich von Bassewitz, 06.08.1998
;
; char cgetc (void);
;
; Modified for SBC-2 by Daryl Rictor
; Modified for Microtan-R by JRP
;


    .export   _cgetc
    .import   cursor

    .include  "mt65.inc"

_cgetc: lda ICHAR           ;Get any input character
        beq _cgetc          ;Wait until there is a character
        pha                 ;Save char
        lda #$00            ;Blank ichar
        sta ICHAR
        pla

        ldx   #$00          ; Clear high byte
    rts
```

10.1.4 clrscr.s

```
;
; Ullrich von Bassewitz, 16.11.2002
;
;
; void clrscr (void);
;
;
; Modified for SBC-2 by Daryl Rictor
; Modified for Microtan-R by JRP
;

    .export     _clrscr

    .include    "mt65.inc"

; store a space charcter in every VDU location.
_clrscr:
        ldx #$00
        lda #$20
next:   sta VIDSTART,x
        sta VIDMID,x
        dex
        bne next
        rts
```

10.1.5 color.s

```
;
; Ullrich von Bassewitz, 06.08.1998
;
; unsigned char __fastcall__ textcolor (unsigned char color);
; unsigned char __fastcall__ bgcolor (unsigned char color);
; unsigned char __fastcall__ bordercolor (unsigned char color);
;
;
; Modified for SBC-2 by Daryl Rictor
; Modified for Microtan-R by JRP but not currently used.
;

    .export     _textcolor, _bgcolor, _bordercolor

    .include    "mt65.inc"

_textcolor:

_bgcolor:

_bordercolor:
    lda #0
    rts
```

10.1.6 conio.h

```
/*****************************************************************************/
/*                                                                           */
/*                                  conio.h                                  */
/*                                                                           */
/*                            Direct console I/O                             */
/*                                                                           */
/*                                                                           */
/*                                                                           */
/* (C) 1998-2005 Ullrich von Bassewitz                                       */
/*               Römerstraße 52                                              */
/*               D-70794 Filderstadt                                         */
/* EMail:        uz@cc65.org                                                 */
/*                                                                           */
/*                                                                           */
/* This software is provided 'as-is', without any expressed or implied       */
/* warranty.  In no event will the authors be held liable for any damages    */
/* arising from the use of this software.                                    */
/*                                                                           */
/* Permission is granted to anyone to use this software for any purpose,     */
/* including commercial applications, and to alter it and redistribute it    */
/* freely, subject to the following restrictions:                            */
/*                                                                           */
/* 1. The origin of this software must not be misrepresented; you must not   */
/*    claim that you wrote the original software. If you use this software   */
/*    in a product, an acknowledgment in the product documentation would be  */
/*    appreciated but is not required.                                       */
/* 2. Altered source versions must be plainly marked as such, and must not   */
/*    be misrepresented as being the original software.                      */
/* 3. This notice may not be removed or altered from any source              */
/*    distribution.                                                          */
/*                                                                           */
/*****************************************************************************/


/*
 * This is the direct console interface for cc65. I do not like the function
 * names very much, but the first version started as a rewrite of Borland's
 * conio, and, even if the interface has changed, the names did not.
 *
 * The interface does direct screen I/O, so it is fast enough for most
 * programs. I did not implement text windows, since many applications do
 * not need them and should not pay for the additional overhead. It should
 * be easy to add text windows on a higher level if needed,
 *
 * Most routines do not check the parameters. This may be unfortunate but is
 * also related to speed. The coordinates are always 0/0 based.
 */


/*  Modified for SBC-2 by Daryl Rictor */
/*  Modified for Microtan-R by JRP     */


#ifndef _CONIO_H
#define _CONIO_H



#if !defined(_STDARG_H)
#  include <stdarg.h>
#endif

/* Include the correct machine-specific file */
#if defined(__APPLE2__)
```

```
#  include <apple2.h>
#elif defined(__APPLE2ENH__)
#  include <apple2enh.h>
#elif defined(__ATARI__)
#  include <atari.h>
#elif defined(__ATMOS__)
#  include <atmos.h>
#elif defined(__CBM__)
#  include <cbm.h>
#elif defined(__GEOS__)
#  include <geos.h>
#elif defined(__LUNIX__)
#  include <lunix.h>
#elif defined(__LYNX__)
#  include <lynx.h>
#elif defined(__NES__)
#  include <nes.h>
#else
#  include <mt65.h>
#endif



/*****************************************************************************/
/*                               Functions                                   */
*/
/*****************************************************************************/



void clrscr (void);
/* Clear the whole screen and put the cursor into the top left corner */

unsigned char kbhit (void);
/* Return true if there's a key waiting, return false if not */

void __fastcall__ gotox (unsigned char x);
/* Set the cursor to the specified X position, leave the Y position untouched
*/

void __fastcall__ gotoy (unsigned char y);
/* Set the cursor to the specified Y position, leave the X position untouched
*/

void __fastcall__ gotoxy (unsigned char x, unsigned char y);
/* Set the cursor to the specified position */

unsigned char wherex (void);
/* Return the X position of the cursor */

unsigned char wherey (void);
/* Return the Y position of the cursor */

void __fastcall__ cputc (char c);
/* Output one character at the current cursor position */

void __fastcall__ cputcxy (unsigned char x, unsigned char y, char c);
/* Same as "gotoxy (x, y); cputc (c);" */

void __fastcall__ cputs (const char* s);
/* Output a NUL-terminated string at the current cursor position */

void __fastcall__ cputsxy (unsigned char x, unsigned char y, const char* s);
/* Same as "gotoxy (x, y); puts (s);" */

int cprintf (const char* format, ...);
/* Like printf(), but uses direct screen output */
```

```
int __fastcall__ vcprintf (const char* format, va_list ap);
/* Like vprintf(), but uses direct screen output */

char cgetc (void);
/* Return a character from the keyboard. If there is no character available,
 * the function waits until the user does press a key. If cursor is set to
 * 1 (see below), a blinking cursor is displayed while waiting.
 */

int cscanf (const char* format, ...);
/* Like scanf(), but uses direct keyboard input */

int __fastcall__ vcscanf (const char* format, va_list ap);
/* Like vscanf(), but uses direct keyboard input */

unsigned char __fastcall__ cursor (unsigned char onoff);
/* If onoff is 1, a cursor is displayed when waiting for keyboard input. If
 * onoff is 0, the cursor is hidden when waiting for keyboard input. The
 * function returns the old cursor setting.
 */

unsigned char __fastcall__ revers (unsigned char onoff);
/* Enable/disable reverse character display. This may not be supported by
 * the output device. Return the old setting.
 */

unsigned char __fastcall__ textcolor (unsigned char color);
/* Set the color for text output. The old color setting is returned. */

unsigned char __fastcall__ bgcolor (unsigned char color);
/* Set the color for the background. The old color setting is returned. */

unsigned char __fastcall__ bordercolor (unsigned char color);
/* Set the color for the border. The old color setting is returned. */

void __fastcall__ chline (unsigned char length);
/* Output a horizontal line with the given length starting at the current
 * cursor position.
 */

void __fastcall__ chlinexy (unsigned char x, unsigned char y, unsigned char
length);
/* Same as "gotoxy (x, y); chline (length);" */

void __fastcall__ cvline (unsigned char length);
/* Output a vertical line with the given length at the current cursor
 * position.
 */

void __fastcall__ cvlinexy (unsigned char x, unsigned char y, unsigned char
length);
/* Same as "gotoxy (x, y); cvline (length);" */

void __fastcall__ cclear (unsigned char length);
/* Clear part of a line (write length spaces). */

void __fastcall__ cclearxy (unsigned char x, unsigned char y, unsigned char
length);
/* Same as "gotoxy (x, y); cclear (length);" */

void __fastcall__ screensize (unsigned char* x, unsigned char* y);
/* Return the current screen size. */

void __fastcall__ cputhex8 (unsigned char val);
void __fastcall__ cputhex16 (unsigned val);
/* These shouldn't be here... */
```

```
/*************************************************************************/
/*                                 Macros                                */
/*************************************************************************/



/* On some platforms, functions are not available or are dummys. To suppress
 * the call to these functions completely, the platform header files may
 * define macros for these functions that start with an underline. If such a
 * macro exists, a new macro is defined here, that expands to the one with the
 * underline. The reason for this two stepped approach is that it is sometimes
 * necessary to take the address of the function, which is not possible when
 * using a macro. Since the function prototype is still present, #undefining
 * the macro will give access to the actual function.
 */

#if defined(_textcolor)
#  define textcolor(x)          _textcolor(x)
#endif
#if defined(_bgcolor)
#  define bgcolor(x)            _bgcolor(x)
#endif
#if defined(_bordercolor)
#  define bordercolor(x)        _bordercolor(x)
#endif



/* End of conio.h */
#endif
```

10.1.7 copymt65.bat

```
echo off
rem  This batch file will move the MT65ROM library files to a sub folder
rem  under the libsrc folder.  It assumes the complete libsrc file structure
rem  is installed.  It further assumes that CC65 is installed under the
rem  "C:\CC65" folder.  Once the files are moved, it will call the
rem  "makesbc2.bat" file to compile the library.
rem  on
rem  Use this at your own risk.  I assume no liability for its use.
rem  on
rem  (c) Daryl Rictor 2012
rem  Modified for Microtan-R JRP 2020 I assume no liability either.

path = C:\cc65\bin;%PATH%
echo Set cc65=c:\cc65...
set CC65=C:\cc65
echo create \libsrc\mt65rom folder...
if not exist %CC65%\libsrc\mt65ROM\nul mkdir %CC65%\libsrc\MT65ROM
echo delete previous files...
del %CC65%\libsrc\MT65ROM\*.* /q >nul
pause
echo copy source files...
copy *.s %CC65%\libsrc\MT65ROM >nul
copy *.inc %CC65%\libsrc\MT65ROM >nul
copy *.h %CC65%\include >nul
if not exist %CC65%\cfg\nul mkdir %CC65%\cfg
copy mt65rom.cfg %CC65%\cfg >nul
copy makemt65.bat %CC65%\libsrc >nul
cd %CC65%\libsrc >nul
echo Running makemt65 to build the libraries.
makemt65
pause
```

10.1.8 cputc.s

```
;
; Ullrich von Bassewitz, 02.06.1998
;
;
; void cputcxy (unsigned char x, unsigned char y, char c);
; void cputc (char c);
;
;
; Modified for SBC-2 by Daryl Rictor
; Modified for Microtan-R by JRP
;

    .export     _cputcxy, _cputc, cputdirect, putchar
    .import     popa, _gotoxy

    .include    "mt65.inc"

_cputcxy:
       pha          ; save char
       jsr popa        ; Get Y
       jsr popa        ; Get X
;      jsr _gotoxy     ; not used in SBC-2
       pla             ; restore char and fall thru

; Plot a character - also used as internal function

_cputc:

cputdirect:                     ; Write the character to the screen
        pha                     ; save registers
        jsr     OUTALL          ; Tugbug print char routine
        pla
        rts                     ; done

; Write one character to the screen without doing anything else.
putchar:
        pha                     ; save registers
        jsr     OUTALL          ; Microtan monitor routine
        pla
        rts
```

10.1.9 crt0.s

```
; C RunTime code - sets up C runtime envionment.
; This version relies on the Microtan Monitor to,
; have setup the serial UART. Entry is via the,
; monitors 'G' command. JRP 2020.
; Ullrich von Bassewitz, 17.06.1998
;
; Startup code for cc65
;
; Modified for SBC-2 by Daryl Rictor
; Modified for Microtan-R by JRP
;
    .export     _exit
    .export     __STARTUP__ : absolute = 1  ; Mark as startup
    .import     initlib, donelib
    .import     copydata
    .import     callmain
    .import     zerobss
    .import     _irq_proc

    .include    "zeropage.inc"
    .include    "mt65.inc"

.segment   "STARTUP"

; ---------------------------------------------------------------------------
; Actual code

_coldstart:             ; reset vector points here!
    sei                 ; Disable interrupts while we meddle
    lda #$4C            ; setup irq vector
    sta INTSL1
    lda #<_irq_proc
    sta INTSL2
    lda #>_irq_proc
    sta INTSL3
    cld
    ldx #$ff            ; setup CPU stack
    txs
    cli                 ; Enable interrupts


; Clear the BSS data

    jsr zerobss
    LDA #$00        ;CLEAR ICHAR
    STA ICHAR

; setup the C stack

    lda #<TOPMEM        ; from sim.inc
    sta sp
    lda #>TOPMEM
    sta sp+1            ; Set argument stack ptr

; Copy data segment from ROM image to RAM
    jsr copydata

; Call module contructors

    jsr initlib

; Push arguments and call main

    jsr callmain
```

```
; Call module destructors. This is also the _exit entry.

_exit:  jsr donelib
        brk                     ; END with BRK - will dump registers on
Microtan

stop:
        jmp stop                ; endless loop


NMIjump:
Interrupt:
INTret:         RTI             ; Null Interrupt return

;Vectors segment not required for Microtan for now.
;.segment    "VECTORS"


;  65C02 Firmware Notes
;
;  NMIjmp      =     $FFFA
;  RESjmp      =     $FFFC
;  INTjmp      =     $FFFE

;.word   Interrupt
;.word   _coldstart
;.word   NMIjump

.bss
```

10.1.10 gotoxy.s

```
;
; Ullrich von Bassewitz, 16.11.2002
;
; void gotoxy (unsigned char x, unsigned char y);
;
;
; Modified for SBC-2 by Daryl Rictor
; Modified for Microtan-R by JRP
;
    .export     _gotoxy
    .import     popa

    .include "mt65.inc"

_gotoxy:
        jsr popa
        rts
```

10.1.11 kbhit.s

```
;
; int kbhit (void);
;
;
; Modified for SBC-2 by Daryl Rictor
; Modified for Microtan-R by JRP
;

    .export     _kbhit
    .import     return0, return1

    .include    "mt65.inc"

_kbhit:
    lda ICHAR           ; Get input char.
    bne L1              ; return 0 if nothing there.
    jmp return0
L1: jmp return1         ; return 1 if there is a char waiting.
```

10.1.12 makemt65.bat

```
echo off
rem
rem  This batch file will compile the MT65ROM library files and place
rem  them in proper folders.  It will then cleanup all non-source files.
rem
rem  Use this at your own risk.  I assume no liability for its use.
rem
rem  (c) Daryl Rictor 2012
rem  Modified for Microtan-R JRP 2020 I assume no liability for its use either
rem
@echo off
path = C:\cc65\bin;c:\cc65;c:\cc65\include;c:\cc65\asminc;%PATH%
echo MT65ROM
cd MT65ROM
for %%i in (*.s) do ca65 -l %%i

echo common
cd ..\common
for %%i in (*.c) do cc65 -Osir -g -I..\..\include %%i
for %%i in (*.s) do ca65 -l %%i

echo conio
cd ..\conio
for %%i in (*.s) do ca65 -l %%i

echo dbg
cd ..\dbg
for %%i in (*.s) do ca65 -l %%i

echo runtime
cd ..\runtime
for %%i in (*.s) do ca65 -l %%i

echo Build library
cd ..
if exist mt65rom.lib del mt65rom.lib >nul:

ar65 a mt65rom.lib common\*.o
ar65 a mt65rom.lib runtime\*.o
ar65 a mt65rom.lib conio\*.o
ar65 a mt65rom.lib dbg\*.o
ar65 a mt65rom.lib mt65rom\*.o

echo Move library files
if not exist %CC65%\lib\nul mkdir %CC65%\lib
copy mt65rom.lib ..\lib >nul

echo Cleanup
rem del mt65rom\*.lst
del mt65rom\*.o
del common\*.o
del conio\*.o
del dbg\*.o
del runtime\*.o
del mt65rom.lib

pause
```

10.1.13 mt65.h

```
/* mt65.h */
/* Modified for SBC-2 by Daryl Rictor */
/* Modified for Microtan-R by JRP */

#ifndef MT65_H
#define MT65_H

/* Zero-page locations */
/* define Ptr (*(unsigned char*)0xe0)  */
/* define temp (*(unsigned char*)0xe3) */
/* define Addr (*(unsigned char*)0xe4) */

#define CH_ULCORNER     '+'
#define CH_URCORNER     '+'
#define CH_LLCORNER     '+'
#define CH_LRCORNER     '+'
#define CH_TTEE         '+'
#define CH_BTEE         '+'
#define CH_LTEE         '+'
#define CH_RTEE         '+'
#define CH_CROSS        '+'


/* MT65 specific constants */
#define ICURS  0x0a                  // cursor address
#define ICURSH 0x0b
#define VDUIND 0x03                  // screen index reg


#define U19_DDRA       0x0BFC3               //6522 Timer 1 Port A Direction
#define U19_DDRB       0x0BFC2               //6522 Timer 1 Port B Direction
#define U19_IORA       0x0BFC1               //6522 Timer 1 Port A Input/Output reg
#define U19_IORB       0x0BFC0               //6522 Timer 1 Port A Input/Output reg
#define U19_IFR        0x0BFCD               //6522 Timer 1 Interrupt Flags
#define U19_IER        0x0BFCE               //6522 Timer 1 Interrupt Enable
#define U19_PCR        0x0BFCC               //6522 Timer 1 Peripheral Control
#define U19_ACR        0x0BFCB               //6522 Timer 1 Auxiliary Control
#define U19_SR         0x0BFCA               //6522 Timer 1 Shift Register
#define U19_T1L_H      0x0BFC7               //6522 Tim#r 1 Latch High
#define U19_T1L_L      0x0BFC6               //6522 Timer 1 Latch Low
#define U19_T1C_H      0x0BFC5               //6522 Timer 1 Counter High
#define U19_T1C_L      0x0BFC4               //6522 Timer 1 Counter Low
#define U19_T2C_H      0x0BFC9               //6522 Timer 2 Counter High
#define U19_T2C_L      0x0BFC8               //6522 Timer 2 Counter Low
#define U19_ORA0x0BFCF               //6522 Timer 1 Port A Output reg (no handshake)


/* MT65 specific functions */
unsigned char __fastcall__ peek (long);
void poke (int,unsigned char);

/* Useful macros */
#define hibyte(x) (__AX__=(x),asm("\ttxa\n\tldx\t#$00"),__AX__)
//#define hibyte(x) (x >> 8)
#define lobyte(x) (__AX__=(x),asm("\tldx\t#$00"),__AX__)
//#define lobyte(x) (x & 0xFF)
#define hiword(x) (x >> 16)
#define loword(x) (x & 0xFFFF)

#endif
```

10.1.14 mt65.inc

```
;
; SBC-2 include File by Daryl Rictor (c) 2014
; Modified for Microtan-R JRP 2020
;
; Top of available memory
TOPMEM := $5BFF

.PC02                   ; use 65C02 assembly


; Zero-page locations

Ptr     := $00          ; two byte pointer (used by peek and poke)


; UART addresses for Microtan-R

ACIA1dat    :=     $BFD0
ACIA1sta    :=     $BFD1
ACIA1cmd    :=     $BFD2
ACIA1ctl    :=     $BFD3

;TUGBUG / TANBUG ENTRY POINTS

OUTALL      :=     $F80E
JPLKB       :=     $F81D

;TUGBUG VARAIABLES

ICHAR       :=     $0001
INTFS1      :=     $04       ;Fast interrupt vector
INTFS2      :=     $05
INTFS3      :=     $06
INTSL1      :=     $10       ;Slow interrupt vector
INTSL2      :=     $11
INTSL3      :=     $12
VIDSTART    :=     $0200
VIDMID      :=     $0300


;TANEX_PLUS 6522 constants.

U19_DDRA    :=  $BFC3           ;6522 Timer 1 Port A Direction
U19_DDRB    :=  $BFC2           ;6522 Timer 1 Port B Direction
U19_IORA    :=  $BFC1           ;6522 Timer 1 Port A Input/Output reg;
U19_IORB    :=  $BFC0           ;6522 Timer 1 Port A Input/Output reg; end
U19_IFR     :=  $BFCD           ;6522 Timer 1 Interrupt Flags;
U19_IER     :=  $BFCE           ;6522 Timer 1 Interrupt Enable
U19_PCR     :=  $BFCC           ;6522 Timer 1 Peripheral Control
U19_ACR     :=  $BFCB           ;6522 Timer 1 Auxiliary Control
U19_SR      :=  $BFCA           ;6522 Timer 1 Shift Register
U19_T1L_H   :=  $BFC7           ;6522 Tim#r 1 Latch High
U19_T1L_L   :=  $BFC6           ;6522 Timer 1 Latch Low
U19_T1C_H   :=  $BFC5           ;6522 Timer 1 Counter High
U19_T1C_L   :=  $BFC4           ;6522 Timer 1 Counter Low
U19_T2C_H   :=  $BFC9           ;6522 Timer 2 Counter High
U19_T2C_L   :=  $BFC8           ;6522 Timer 2 Counter Low
U19_ORA     :=  $BFCF           ;6522 Timer 1 Port A Output reg (no handshake)
```

10.1.15 mt65rom.cfg

```
#
#  cc65 Config for SBC-2 ROM image generation
#  By Daryl Rictor 2014
#  Modified for Microtan-R JRP 2020
#
MEMORY {
    ZP: start = $0040, size = $00BF, type = rw, define = yes;  #Zero Page < 40 is for Microtan monitor use.
    RAM:start = $0400, size = $0FFF, define = yes;             #4K (approx) for program data and 'C' stack.
    ROM:start = $1000, size = $6FFF, file = %O, define = yes;  #28K This is where the code will live.
}
SEGMENTS {
        STARTUP:   load = ROM, type = ro;
        INIT:      load = ROM, type = ro,  optional = yes;
        CODE:      load = ROM, type = ro;
        RODATA:    load = ROM, type = ro;
        DATA:      load = ROM, type = rw,  define   = yes, run = RAM;
        ZEROPAGE:  load = ZP,  type = zp,  define   = yes;
        BSS:       load = RAM, type = bss, define   = yes;
        HEAP:      load = RAM, type = bss, optional = yes;

# Don't need the Intrrupt vectors at the moment
# The code will set them up in the crt0 file.
# VECTORS:   load = ROM, type = ro,  start    = $8000;
}
FEATURES {
    CONDES:    segment = STARTUP,
               type    = constructor,
                   label   = __CONSTRUCTOR_TABLE__,
                   count   = __CONSTRUCTOR_COUNT__;
        CONDES:    segment = STARTUP,
                   type    = destructor,
                   label   = __DESTRUCTOR_TABLE__,
                   count   = __DESTRUCTOR_COUNT__;
}
SYMBOLS {
    __STACKSIZE__ = $800;   # 2K 'C' stack
}
```

10.1.16 peek.s

```
;
; Peek memory function
;
; Modified for SBC-2 by Daryl Rictor
; Modified for Microtan-R by JRP
;

    .include    "mt65.inc"
    .importzp   sreg
    .importzp  ptr1
    .export   _peek

.code
;---------------------------------------------------------------------------
; peek: read memory byte

.proc   _peek
        sta ptr1
        stx ptr1+1
        ldy #$00
        lda (ptr1),y
        ldx #$00
        rts

.endproc
```

10.1.17 poke.s

```
;
; Poke memory function
;
; Modified for SBC-2 by Daryl Rictor
; Modified for Microtan-R by JRP
;

    .include    "mt65.inc"
    .import popa, popax
    .importzp   sreg
    .importzp   ptr1
    .export _poke

.code
;---------------------------------------------------------------------------
; poke: write memory byte

.proc   _poke
    jsr popa
    pha
    jsr popax
    sta ptr1
    stx ptr1+1
    ldy #$00
    pla
    sta (ptr1),y
    rts

.endproc
```

10.1.18 randomize.s

```
;
; Ullrich von Bassewitz, 07.11.2002
;
; void _randomize (void);
; /* Initialize the random number generator */
; /* using RTC register values */
;
; Modified for SBC-2 by Daryl Rictor
; Modified for Microtan-R by JRP
;
    .export     __randomize
    .import     _srand
    .importzp       tmp1, tmp2

    .include        "mt65.inc"

__randomize:

    lda #$5A        ; bogus seed
    sta tmp2
    lda #$2E
    sta tmp1
    jmp _srand      ; Initialize generator
```

10.1.19 read.s

```
;
; Ullrich von Bassewitz, 16.11.2002
;
; int read (int fd, void* buf, unsigned count);
;
;
; Modified for SBC-2 by Daryl Rictor
; Modified for Microtan-R by JRP
;
    .export     _read
    .constructor    initstdin

    .import     __oserror
    .import     rwcommon , _cgetc
    .importzp       ptr1, ptr2, ptr3, tmp1, tmp2,
tmp3

    .include        "mt65.inc"
    .include        "fcntl.inc"
    .include        "filedes.inc"

;-----------------------------------------------
--------------------------
; initstdin: Open the stdin file descriptors for
the keyboard

.segment  "INIT"

.proc    initstdin

    lda #SBC2_STDIN
    sta fdtab+STDIN_FILENO
    rts

.endproc

;-----------------------------------------------
--------------------------
; _read

.code

.proc  _read

;  rwcommon set this up:
;  ptr1= count (unsigned int, # bytes to read)
;  ptr2= buffer pointer (bank 0 only)
;  ptr3= bytes read(unsigned int)
;  tmp2= file handle

    jsr rwcommon        ; Pop params, check
handle
    bcs errout      ; Invalid handle, errno
already set

; Check if the handle is valid and the file is
open for writing

    tax
    lda fdtab,x     ; Get flags for this handle
    and #$01            ; File open for reading?
    beq notopen

; Check the EOF flag. If it is set, don't read
anything

;   bit fdtab,x     ; Get flags for this handle
    lda fdtab,x     ; Get flags for this handle
    bmi eof
        asl
    bmi kybrd           ; STDIN
    jmp notopen     ; NO FILE SYS

kybrd:
    ; Check for zero count
    lda ptr1
    ora ptr1+1
    beq chk2

    ; Read from kybrd
next2:
    jsr _cgetc          ; wait for keypress

    ; Put char into buf
    ldy #$00
    sta (ptr1),y

    ; Increment pointer
    inc ptr2
    bne :+
    inc ptr2+1

    ; Increment counter
:   inc ptr3
    bne chk2
    inc ptr3+1

    ; Check for counter less than count
chk2:   lda ptr3
    cmp ptr1
    bcc next2
    ldx ptr3+1
    cpx ptr1+1
    bcc next2

    ; Return success, AX already set
    rts


; Return the number of chars read

eof:
    cmp #$0F            ; EOF flag
    bne error           ; system error abort
    lda ptr3
    ldx ptr3+1
    rts

; Error entry, file is not open

notopen:
    lda     #$08        ; File not open

; Error entry, status not ok

error:
    sta __oserror
errout:
    lda #$FF
    tax             ; Return -1
    rts


.endproc
```

10.1.20 write.s

```
;
; Ullrich von Bassewitz, 16.11.2002
;
; Modified for SBC-2 by Daryl Rictor
; Modified for Microtan-R by JRP
;

;
; int write (int fd, const void* buf, unsigned
count);
;
    .export     _write
    .constructor    initstdout

    .import     __oserror
    .import     rwcommon, _cputc
    .importzp   sp, ptr1, ptr2, ptr3

    .include    "fcntl.inc"
    .include    "filedes.inc"
    .include    "mt65.inc"


;-----------------------------------------------
--------------------------
; initstdout: Open the stdout and stderr file
descriptors for the screen.

.segment    "INIT"

.proc   initstdout

    lda #SBC2_STDOUT
    sta fdtab+STDOUT_FILENO
    sta fdtab+STDERR_FILENO
    rts

.endproc

;-----------------------------------------------
--------------------------
; _write

.code

.proc   _write

    jsr     rwcommon        ; Pop params, check
handle
    bcs     errout          ; Invalid handle,
errno already set

    tax             ; save handle
    lda ptr1
    EOR #$FF
    sta ptr1
    lda ptr1+1
    EOR #$FF
    sta ptr1+1      ; Remember -count-1
    inc ptr1
    bne :+
    inc ptr1+1      ; add 1 to count


; Check if the handle is valid and the file is
open for writing

:   lda fdtab,x     ; Get flags for this handle
    and #$02        ; File open for writing?
    beq notopen

;   bit fdtab,x     ; Get flags for this handle
    lda fdtab,x     ; Get flags for this handle
        asl
    bmi console     ; STDOUT
    jmp notopen     ; NO FILE SYS

console:
@L0:    ldy #0
    lda (ptr2),y
    inc ptr2
    bne @L1
    inc ptr2+1      ; A = *buf++;
@L1:jsr  _cputc

; Count characters written

    inc ptr3
    bne @L2
    inc ptr3+1

; Decrement count

@L2:    inc ptr1
    bne @L0
    inc ptr1+1
    bne @L0

; Wrote all chars, close the output channel
; Return the number of chars written

    lda ptr3
    ldx ptr3+1
    rts

; Error entry, file is not open
notopen:
    lda #$08        ; File not open

; Error entry, status not ok

error:  sta __oserror
errout: lda #$FF
    tax         ; Return -1
    rts

.endproc
```

10.2 HELLOWORLD FILES

10.2.1 helloWorld.c

This file should be put in the **C:\cc65\MT65 Demos\helloWorld** folder.

```
// Hello World program for Microtan-R.
// If this compiles and runs,
// then the toolchain is OK.

#include <conio.h>
#include <stdlib.h>

// prototypes
void setcursor(unsigned char x, unsigned char y);


int main (void)
{
        clrscr();                       // clear the Microtan VDU screen
        setcursor(10,8);                // Put cursor (X,Y) somewhere near the screen centre
        cprintf("Hello World\r\r\r");   // and print using console i/o, '\r' equiv to \n,
                                        // cc65 does not support 'normal' printf for Microtan.
        cprintf("Hit any key to exit");
        while(!kbhit())
        {
                ;
        }

    return EXIT_SUCCESS;
}


void setcursor(unsigned char x, unsigned char y)
{
        // Microtan specific cursor positioning routine.
        int memaddr = (0x200 + (y*32));
        int addrh = (memaddr/256);
        int addrl =  memaddr - (( (int)(memaddr/256))*256);

        //Microtan specific constants picked up from mt65.h via conio.h
        poke(ICURSH, addrh);
        poke(ICURS, addrl);
        poke(VDUIND,x);
}
```

10.2.2 helloWorld make.bat file

This file should be put in the **C:\cc65\MT65 Demos\helloWorld** folder.

```
@echo off
set CC65LIB=c:\cc65\lib
if exist helloWorld.65b del helloWorld.65b >nul:
cl65 -t none -C mt65rom.cfg -o helloWorld.65b -T -m helloWorld.map -l helloWorld.c %CC65LIB%\mt65rom.lib -
Wl -vm
del *.o /Q >nul
if exist helloWorld.65b (
pause
java ..\bin2txt.java helloWorld.65b
pause) else (echo compilation failed)
@echo on
```

10.2.3 bin2txt.java file

This file should be put in the **C:\cc65\MT65 Demos** folder.

```
package bin2txt;
//bin2txt - simply converts a binary file produced by cc65 to ASCII characters,
//          and prints them out.

import java.io.*;
import java.nio.file.*;

public class bin2txt
{

    public static void main(String[] args)
    {
        System.out.println("Hello from bin2txt!");
        String inputFile = args[0];


        try
        {

            int itemsPerLine = 0;
            byte[] allBytes = Files.readAllBytes(Paths.get(inputFile));
            for(int i=0; i < allBytes.length; i++)
            {
                {
                    String st = String.format("%02X",allBytes[i]);
                    System.out.print(st);
                    System.out.print(" ");

                    itemsPerLine++;
                    if (itemsPerLine == 16)
                    {
                        System.out.println();
                        itemsPerLine = 0;
                    }
                }

            }

        } catch (IOException ex)
        {
            ex.printStackTrace();
        }

    }

}
```

10.3 HIGH SPEED LOADER

10.3.1 mt65loader.ino

This is the Arduino code.

```
/*
   Simple Microtan-R code loader.
   Reads ASCII coded bytes from the serial port,
   converts them to an 8 bit binary byte and then,
   outputs it to the Microtan via pins 2-9.
   Arbitration of the transfer is done by CA1 & CA2,
   on pins 10 & 11.
   JRP Dec 2020
   Note: There is NO WARRANTY, LIABILITY etc.
*/

#include <avr/io.h>

typedef unsigned short int ushort;

// Pins
static const int dbus[8] = { 2, 3, 4, 5, 6, 7, 8, 9 };

//6522 handshake lines
static const ushort
CA1 = 10,
CA2 = 11;

static void setData(unsigned char db)
{
  unsigned char bit = 1;
  for (ushort i = 0; i < 8; i++)
  {
    digitalWrite(dbus[i], (db & bit) ? HIGH : LOW);
    bit <<= 1;
  }
}

  int incomingByte[4]; // for incoming serial data
  int i = 0;
  int x;
  unsigned char high_nibble, low_nibble, value;

// Main code ----------------------------------------------
--
void setup()
{
  Serial.begin(2000000);

  pinMode(CA1, OUTPUT);
  digitalWrite(CA1, HIGH);

  pinMode(CA2, INPUT);

  for (ushort i = 0; i < 8; i++)
  {
    pinMode(dbus[i], OUTPUT);
    digitalWrite(dbus[i], LOW);
  }
}
bool dataTaken()
{
  if (digitalRead(CA2) == LOW)
    return true;
  else
    return false;
}


void setDataReady()
{
   digitalWrite(CA1, LOW);
}


void setDataNotReady()
{
   digitalWrite(CA1, HIGH);
}


void sendData(unsigned char data)
{
  setData(data);
  setDataReady();

  while (!dataTaken())
  {
  }
  setDataNotReady();
}


void printHex(unsigned char num)
{
  char tmp[16];

  sprintf(tmp, "%02X", num);
  //Serial.println(tmp);
  sendData(num);
}


unsigned char h2d(unsigned char hex)
{
  //Serial.println(hex,HEX);
  if (hex > 0x39) hex -= 7; // adjust for hex letters
  hex = (hex & 0x0f);
  //Serial.println(hex,HEX);
  return (hex);
}


void loop()
{
  // Serial port data is destined to be loaded into RAM:
    if (Serial.available() > 0)
    {
      incomingByte[i] = Serial.read();
      if (incomingByte[i] == 32)
      {
        high_nibble = h2d(incomingByte[0]);
        low_nibble  = h2d(incomingByte[1]);
        value = 0;
        value = (high_nibble << 4) | low_nibble;
        printHex(value);
        i = 0;
      }
      else
      {
        if (incomingByte[i] >= '0' &&  incomingByte[i] <=
'F')
        {
          i++;
        }
      }
    }
}
```

10.3.2 mt65_loader.asm

This is the Microtan code.

```
;mt65_loader.asm
;Simple byte code loader for the Microtan-R
;Uses the 6522 Port A1 on TANEX-PLUS
;JRP 2020 No WARRANTY or LIABILITY etc.

ADDL:     EQU    $40
ADDH:     EQU    $41
IRA:      EQU    $BFC1
DDRA:     EQU    $BFC3
PCR:      EQU    $BFCC
IFR:      EQU    $BFCD
HEXPRN:   EQU    $F81A
LDADDRL:  EQU    $00     ; Bytes will be loaded at $1000...
LDADDRH:  EQU    $10
VDUIND:   EQU    $03

    ORG $BB00

BB00  A900    INIT:   LDA  #$00
BB02  8DC3BF          STA  DDRA        ; Set port a to input
BB05  A8              TAY              ; Set index reg y to 0

BB06  A908            LDA  #$08        ; CA2 handshake mode
BB08  8DCCBF          STA  PCR

BB0B  A900            LDA  #LDADDRL    ; Set base address
BB0D  8540            STA  ADDL
BB0F  A910            LDA  #LDADDRH
BB11  8541            STA  ADDH

BB13  ADCDBF  WAIT:   LDA  IFR
BB16  2902            AND  #$02        ; Is CA1 active?
BB18  F0F9            BEQ  WAIT        ; No keep looking
BB1A  ADC1BF          LDA  IRA         ; Yes then read data
BB1D  9140            STA  (ADDL),Y    ; Store the data

BB1F  E640            INC  ADDL        ; Increment the addr
BB21  D002            BNE  END         ;
BB23  E641            INC  ADDH        ;
BB25  A900    END:    LDA  #$00        ; Set VDU index to 0
BB27  8503            STA  VDUIND      ;
BB29  A541            LDA  ADDH        ; Get current address
BB2B  201AF8          JSR  HEXPRN      ; Print it
BB2E  A540            LDA  ADDL        ;
BB30  201AF8          JSR  HEXPRN      ;
BB33  4C13BB          JMP  WAIT        ;


There have been no errors.

  A9 00 8D C3 BF A8 A9 08
  8D CC BF A9 00 85 40 A9
  10 85 41 AD CD BF 29 02
  F0 F9 AD C1 BF 91 40 E6
  40 D0 02 E6 41 A9 00 85
  03 A5 41 20 1A F8 A5 40
  20 1A F8 4C 13 BB
```

10.4 MIDI PLAYER FILES

10.4.1 midi_loader

This is the Arduino code.

```
/*
   Microtan-R Midi & code loader.
   JRP Dec 2020
   Note there is NO WARRANTY.
*/

#include <avr/io.h>
#include "MIDIUSB.h"
#define NOTE_OFF 0

typedef unsigned short int ushort;
typedef unsigned char note_t;
typedef unsigned char midictrl_t;


// Pins
static const int dbus[8] = { 2, 3, 4, 5, 6,
7, 8, 9 };

//6522 handshake lines
static const ushort
CA1 = 10,
CA2 = 11;

static void setData(unsigned char db)
{
  unsigned char bit = 1;
  for (ushort i = 0; i < 8; i++)
  {
    digitalWrite(dbus[i], (db & bit) ? HIGH
: LOW);
    bit <<= 1;
  }
}

  int incomingByte[4]; // for incoming
serial data
  int i = 0;
  int x;
  unsigned char high_nibble, low_nibble,
value;


// Main code ----------------------------
----------------

//static unsigned long lastUpdate = 0;

void setup()
{
  Serial.begin(2000000);

  pinMode(CA1, OUTPUT);
  digitalWrite(CA1, HIGH);

  pinMode(CA2, INPUT);

  for (ushort i = 0; i < 8; i++)
  {
    pinMode(dbus[i], OUTPUT);
    digitalWrite(dbus[i], LOW);
  }
}


bool dataTaken()
{
  if (digitalRead(CA2) == LOW)
    return true;
  else
    return false;
}


void setDataReady()
{
   digitalWrite(CA1, LOW);
}


void setDataNotReady()
{
   digitalWrite(CA1, HIGH);
}


void sendData(unsigned char data)
{
  setData(data);
  setDataReady();

  while (!dataTaken())
  {
  }
  setDataNotReady();
}


void printHex(unsigned char num)
{
  char tmp[16];

  sprintf(tmp, "%02X", num);
  Serial.println(tmp);
  sendData(num);
}


unsigned char h2d(unsigned char hex)
{
  //Serial.println(hex,HEX);
  if (hex > 0x39) hex -= 7; // adjust for
hex letters upper or lower case
  hex = (hex & 0x0f);
  //Serial.println(hex,HEX);
```

```
  return (hex);
}


static void noteOn( midictrl_t chan, note_t
note, midictrl_t vel )
{
      Serial.print("Note On ");
      Serial.println(note, HEX);
      printHex(note | 0x80);
      printHex(vel);
}


static void noteOff( midictrl_t chan,
note_t note, midictrl_t vel )
{
      Serial.print("Note Off ");
      Serial.println(note, HEX);
      printHex(note);
      printHex(vel);
}

void loop()
{
  midiEventPacket_t rx = MidiUSB.read();

  if ( rx.header==0x9 ) // Note on
  {
    noteOn( rx.byte1 & 0xF, rx.byte2,
rx.byte3 );
  }
  else if ( rx.header==0x8 ) // Note off
  {
    noteOff( rx.byte1 & 0xF, rx.byte2,
rx.byte3 );
  }

  // Serial port data is destined to be
loaded into RAM:
    if (Serial.available() > 0)
    {
      incomingByte[i] = Serial.read();

      if (incomingByte[i] == 32)
      {
        //Serial.println("found space");
        high_nibble = h2d(incomingByte[0]);
        low_nibble = h2d(incomingByte[1]);
        value = 0;
        value = (high_nibble << 4) |
low_nibble;
        printHex(value);
        i = 0;
      }
      else
      {
        if (incomingByte[i] >= '0' &&
incomingByte[i] <= 'Z')
        {
          i++;
        }
      }
    }
}
```

10.4.2 midi.c

```
#include <conio.h>
#include <stdlib.h>

/*Microtan constants */
#define ICHAR_ADDR          0x0001         // Microtan Character input buffer
#define ICURSL_ADDR         0x000A
#define ICURSH_ADDR   0x000B
#define VDUIND_ADDR   0x0003
#define AY_ADDR             0xBC00         // 1st possible base address of a soundcard
#define AY_DATA             0xBC01
#define AY_ADDR_ALT         0xBD00         // Alternate 1st base address of a soundcard
#define AY_PORT_A  0x0E
#define U19_6522_IRA  0xBFC1               // U19 6522 on TANEX-PLUS
#define U19_6522_DDRA 0xBFC3
#define U19_6522_IFR  0xBFCD
#define U19_6522_PCR  0xBFCC
#define VOICE_IN_USE_TENS  0x03F1          // Microtan screen location
#define VOICE_IN_USE_UNITS 0x03F2
#define MEASURED_PH2  749570               // Nominally 750KHz
/*^^^ Microtan related constants ^^^*/

#define NO_VOICE            0xFF
//#define MAX_VOICES        18

#define MAX_POSSIBLE_SOUNDCARDS 16
#define MAX_POSSIBLE_CHIPS_PER_SOUNDCARD 2
#define VOICES_PER_CHIP 3
#define MAX_POSIBLE_VOICES (MAX_POSSIBLE_SOUNDCARDS * MAX_POSSIBLE_CHIPS_PER_SOUNDCARD * VOICES_PER_CHIP)
//#define TOTAL_CHIPS (MAX_VOICES / VOICES_PER_CHIP)
#define ENABLE_REG 7

#define AY_PRE_DIVISOR 16
#define FCLOCK (MEASURED_PH2 / AY_PRE_DIVISOR)
#define true 1
#define false 0


void setcursor(unsigned char x, unsigned char y);
void calculateFrequencies();
unsigned char getMidiNote();
void setup6522();

void noteOn(unsigned char note, unsigned char voice, unsigned char noteVelocity);
void noteOff(unsigned char note);
void enableVoice(unsigned char voice);
void disableVoice(unsigned char voice);
void disableAllVoices();
void enableAllVoices();
unsigned char getVoice(unsigned char note);
unsigned char getVoiceUsedByNote(unsigned char note);
void clearVoicesInUse(void);
void displayVoicesInuse(void);
void cursorOff(void);
unsigned char detectNoOfChips(void);

unsigned char irq_count2;
unsigned char midiNote;
unsigned char noteVelocity;
unsigned char const voiceFreqLReg[] = {0,2,4};
unsigned char const voiceAmpLReg[] = {8,9,0xA};
unsigned char freqHigh[128];
unsigned char freqLow[128];
unsigned char voiceInUse[MAX_POSIBLE_VOICES];
unsigned char voicesInUse = 0;
unsigned char maxVoicesUsed = 0;
unsigned char maxVoicesAvailable = 0;
unsigned char totalChips = 0;

char *voicesInUseTens      = (char*)VOICE_IN_USE_TENS;
```

```
char *voicesInUseUnits     = (char*)VOICE_IN_USE_UNITS;
char *maxVoicesUsedTens    = (char*)VOICE_IN_USE_TENS + 10;
char *maxVoicesUsedUnits   = (char*)VOICE_IN_USE_UNITS + 10;

// Frequencies for equal-tempered scale, Octaves 0 to 8, A4 = 440 Hz
// Frequecies are rounded to nearest integer.
// lowest octave (-1) not particularly accurate.
int const noteFreq[] =
                          {8 ,9 ,9 ,10 ,10 ,11 ,12 ,13 ,13 ,14 ,14 ,15,
                           16,17,18,19,20,22,23,24,26,27,29,31,
                           33,35,37,39,41,44,46,49,52,55,58,62,
                           65,69,73,77,82,87,92,98,104,110,117,123,
                           131,139,147,156,165,175,185,196,208,220,233,247,
                           262,277,294,311,330,349,370,392,415,440,466,494,
                           523,554,587,622,659,698,740,784,831,880,932,988,
                           1046,1109,1175,1245,1319,1397,1480,1568,1661,1760,1865,1976,
                           2093,2217,2349,2489,2637,2794,2960,3136,3322,3520,3729,3951,
                           4186,4435,4699,4978,5274,5588,5920,6272,6645,7040,7459,7902,
                           8372,8870,9397,9956,10548,11175,11840,12544,13290,14080,14917,15804};



int main (void)
{
        unsigned char note;
        unsigned char currentVoice;

        totalChips = detectNoOfChips();
        maxVoicesAvailable = totalChips * VOICES_PER_CHIP;

        clrscr();
        cprintf("\r     Microtan MIDI V0.08 \r        JRP Dec 2020 \r\r");
        cprintf(" Found %d contiguous AY-3-8910s\r", totalChips);
        cprintf(" Giving %-2d simultaneous voices\r\r\r\r\r",maxVoicesAvailable);
        cprintf("Hit any key to reset Max count\r\r");
        cprintf(" Voices in use:- %02d  Max:- %02d",0,0);
        cursorOff();
        calculateFrequencies();
        setup6522();
        disableAllVoices();
        clearVoicesInUse();
        enableAllVoices();

        while(1)
        {
                displayVoicesInuse();
                note = getMidiNote();

                if((note & 0x80) == 0x80)
                {
                        currentVoice = getVoice(note & 0x7f);
                        noteOn(note & 0X7f, currentVoice, noteVelocity);
                }
                else
                {
                        noteOff(note);
                }

        }
    return EXIT_SUCCESS;
}

unsigned char getVoice(unsigned char note)
{
        unsigned char i;
        unsigned char voiceAvailable = NO_VOICE;

        for(i=0; i < maxVoicesAvailable; i++)
        {
                if (voiceInUse[i] == NO_VOICE)
                {
                        voiceAvailable = i;
                        voiceInUse[i] = note;
```

```
                        break;
                }
        }
        return voiceAvailable;
}


void calculateFrequencies()
{
        unsigned int freq = 0;
        unsigned char i;

        for (i=0; i<=127; ++i)
        {
                freq = FCLOCK/noteFreq[i];
                freqHigh[i] = freq/256;
                freqLow[i]       = freq - (freqHigh[i]*256);
                //cprintf("Note=%d, NoteFreq = %d, freq=%d, High=%02X, Low=%02X\r", i, noteFreq[i], freq,
freqHigh[i], freqLow[i]);
        }
}


void noteOn(unsigned char note, unsigned char voice, unsigned char noteVelocity)
{
        char *ayAddr = (char*)AY_ADDR;
        char *ayData = (char*)AY_DATA;
        unsigned char chip;

        if (voice != NO_VOICE)
        {
                chip = voice / VOICES_PER_CHIP;

                ayAddr+=(2*chip);
                ayData = ayAddr+1;

                voice %=VOICES_PER_CHIP; //Voice mod 3

                *ayAddr = voiceFreqLReg[voice];   // voice freq fine
                *ayData = freqLow[note];
                *ayAddr = voiceFreqLReg[voice]+1; // voice freq coarse
                *ayData = freqHigh[note];

                *ayAddr = voiceAmpLReg[voice];             // voice Amplitude
                *ayData = (noteVelocity/16) + 8;  // velocity (0-127) mapped to Amplitude (8 - 15)
                voicesInUse++;
        }
}


void noteOff(unsigned char note)
{
        unsigned char voice;
        unsigned char chip;

        char *ayAddr = (char*)AY_ADDR;
        char *ayData = (char*)AY_DATA;

        voice = getVoiceUsedByNote(note);

        if (voice != NO_VOICE)
        {
                chip = voice / VOICES_PER_CHIP;
                ayAddr+=(2*chip);
                ayData = ayAddr+1;

                voice %=VOICES_PER_CHIP;          // Voice mod 3

                *ayAddr = voiceAmpLReg[voice];             // voice Amplitude
                *ayData = 0;                               // Amplitude = 0 = OFF
                voicesInUse--;
        }
}
```

```
void displayVoicesInuse(void)
{
        unsigned char tens = 0;
        unsigned char units = 0;

        tens = voicesInUse / 10;
        units = voicesInUse % 10;

        *voicesInUseUnits = '0' + units;
        *voicesInUseTens  = '0' + tens;


        if(voicesInUse > maxVoicesUsed || maxVoicesUsed == NO_VOICE)
        {
                maxVoicesUsed = voicesInUse;

                *maxVoicesUsedUnits = '0' + units;
                *maxVoicesUsedTens  = '0' + tens;
        }

}

void enableVoice(unsigned char voice)
{
        char *ayAddr = (char*)AY_ADDR;
        char *ayData = (char*)AY_DATA;
        unsigned char chip;

        chip = voice / VOICES_PER_CHIP;

        ayAddr+=(2*chip);
        ayData = ayAddr+1;

        voice %=VOICES_PER_CHIP;  //Voice mod 3

        /*enable ay-3 voice (chan A(0),B(1) or C(2))*/
        *ayAddr = ENABLE_REG;
        *ayData = *(char*)ayAddr & (0xff & (~(1 << (voice))));
}


void enableAllVoices()
{
        unsigned char i;
        for (i=0; i < maxVoicesAvailable; ++i)
                enableVoice(i);
}


void disableVoice(unsigned char voice)
{
        char *ayAddr = (char*)AY_ADDR;
        char *ayData;
        unsigned char chip;

        chip = voice / VOICES_PER_CHIP;

        ayAddr+=(2*chip);
        ayData = ayAddr+1;

        voice %=VOICES_PER_CHIP;  //Voice mod 3

        /*Disable ay-3 voice (chan A(0),B(1) or C(2))*/
        *ayAddr = ENABLE_REG;
        *ayData = *(char*)ayAddr | (1 << voice);
}


void disableAllVoices()
{
        char *ayAddr;
```

```
        char *ayData;
        unsigned char chip;

        for (chip=0; chip < totalChips; ++chip)
        {
                ayAddr = (char*)AY_ADDR + (2*chip);
                ayData = ayAddr+1;

                *ayAddr = ENABLE_REG;
                *ayData = 0x3F;  // turn all voices and noise channel OFF
        }
}

void clearVoicesInUse(void)
{
        unsigned char i;
        for(i=0; i < maxVoicesAvailable; i++)
        {
                voiceInUse[i] = NO_VOICE;
        }
}


unsigned char getVoiceUsedByNote(unsigned char note)
{
        unsigned char i;
        unsigned char voiceInUseFound = NO_VOICE;

        for(i=0; i < maxVoicesAvailable; i++)
        {
                if(voiceInUse[i] == note)
                {
                        voiceInUseFound = i;
                        voiceInUse[i] = NO_VOICE;
                        break;
                }
        }

//      if (voiceInUseFound == NO_VOICE)
//              cprintf("Note OFF without a voice\r");

        return voiceInUseFound;
}


unsigned char getMidiNote()
{
        while( (*(char*)U19_6522_IFR & 0x02) == 0 )
        {
                if (*((char*)ICHAR_ADDR) != 0)
                        {
                                maxVoicesUsed = NO_VOICE;
                                *((char*)ICHAR_ADDR) = 0;
                                displayVoicesInuse();
                        }
        }
        midiNote = (*(char*)U19_6522_IRA);

        while( (*(char*)U19_6522_IFR & 0x02) == 0 )
        {
                ;
        }
        noteVelocity = (*(char*)U19_6522_IRA);

        return midiNote;
}


void setup6522()
{
        *(char*)U19_6522_DDRA = 0x00;      // port A input mode
        *(char*)U19_6522_PCR  = 0x08;      // handshake mode
```

```
}


void setcursor(unsigned char x, unsigned char y)
{
        int memaddr = (0x200 + (y*32));
        int addrh = (memaddr/256);
        int addrl =  memaddr - (( (int)(memaddr/256))*256);
        poke(ICURSH, addrh);
        poke(ICURS, addrl);
        poke(VDUIND,x);
}


void cursorOff(void)
{



         *(char*)(  (*(char*)ICURSH_ADDR*256) +
         (*(char*)ICURSL_ADDR) +
         (*(char*)VDUIND_ADDR)  ) = (char)0x20;
}


unsigned char detectNoOfChips(void)
{
        unsigned char noChipDetected = false;
        unsigned char offset = 0;
        unsigned char noOfChips = 0;
        char *chipAddr = (char*)AY_ADDR;

        while(noChipDetected == false)
        {
                *(chipAddr + offset) = AY_PORT_A;
                //cprintf("chipAddr = %04X, offset = %02X Data = %02X\r",chipAddr, offset, *(chipAddr +
offset));
                if (*(chipAddr + offset) == 0xFF)
                {
                        noOfChips++;
                        offset +=2;
                }
                else
                {
                        noChipDetected = true;
                }

                if ( chipAddr == (char*)AY_ADDR && offset == 32)
                {
                        chipAddr = (char*)AY_ADDR_ALT;
                        offset = 0;
                }
                else if (chipAddr == (char*)AY_ADDR_ALT && offset == 32)
                {
                        noChipDetected = true;
                }
        }
return noOfChips;
}
```

10.4.3 make.bat

```
@echo off
set CC65LIB=c:\cc65\lib
if exist midi.65b del midi.65b >nul:
cl65 -t none -C mt65rom.cfg -o midi.65b -T -m midi.map -l midi.c  %CC65LIB%\mt65rom.lib -Wl -
vm
del *.o /Q >nul
if exist midi.65b (
pause
java ..\bin2txt.java midi.65b
pause) else ( echo compilation failed)
@echo on
```

11 REFERENCES

All things Microtan.
http://www.microtan.ukpc.net/

AY-3-8910 / 8912 Programmable Sound Generator Data Manual
http://www.microtan.ukpc.net/Products/SoundDataManual.pdf

mfiles – lots of free MIDI files.
https://www.mfiles.co.uk/midi-files.htm

Maidavale.org – Ay vs YM sound IC differences.
https://maidavale.org/blog/ay-ym-differences/

Michigan Tech - musical note frequencies.
https://pages.mtu.edu/~suits/notefreqs.html

MidiEditor – A graphical interface to edit, play and record Midi data.
https://midieditor.org/

Sigrok – the home of the PulseView application.
https://sigrok.org/

Sound Generator Board for Tangerine systems – Bulldog Video Limited
http://www.microtan.ukpc.net/Products/BulldogSound.pdf

WilsonMinesCo.com - All kinds of 6502 resources - in particular a I^2C bit banging implementation
http://wilsonminesco.com/6502primer/GENRLI2C.ASM

12 DOCUMENT HISTORY

Date	Comment	Version	By
12/01/2021	Initial draft	0.1	JRP
13/01/2021	After review	0.2	DG / JRP
16/01/2021	Added copyright	0.3	JRP
18/01/2021	First Issue	1.0	JRP

Image Attribution

Image	Attribution and License
Tangerine Microtan 65 In System Rack (white bg).jpg	https://commons.wikimedia.org/wiki/File:Tangerine_Microtan_65_In_System_Rack_(white_bg).jpg This file is licensed under the Creative Commons Attribution-Share Alike 3.0 Unported license. **Attribution: The original uploader was Ian Dunster at English Wikipedia.**
Musical Notes on Staves	CC0 Public Domain. No attribution required.