

Retrofit mICE Interface for Sinclair QL

Background

A friend of mine gave me a Eidersoft mICE interface for repair as there was the startup screen visible, but no action from mouse.

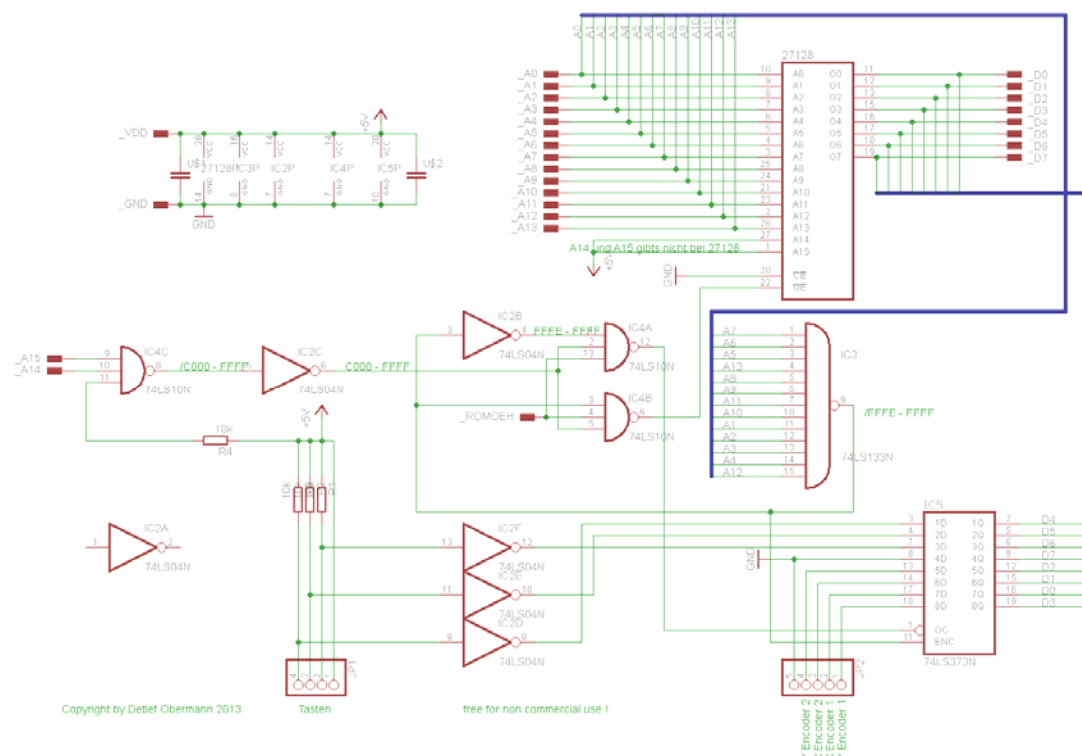
The mICE consists of two items, one is the interface itself as ROM cardridge and 3 button mouse. I had no knowledge about either the interface nor the mouse, so I start to analyse the ROM modul first. As there was no error found, I continued with mouse and wiring. The mouse has a quite simple way to convert movement from the ball to the interface. The system is working with so call quadratur-encoders, which means that 4 digital signals, 2 for each axle. In addition there are 3 signals for each mouse key. The encoders work with optical signals from one IR-LED to 4 receiver transistors. The light is conducted with polymerfibre to the transistors. The error was the degrading of the LED, so there wasn't enough light to detect motion anymore. After replacing the LED everythings works fine again. The ROM files can be found in the internet.

Detlef Obermann 2012/13 detlef@dieobis.de

The Project Part 1 “The mICE Interface”

Due to the fact that I had analysed the circuit I was able to do a circuit drawing. For things like this I am using the EAGLE tool from CADSOFT.

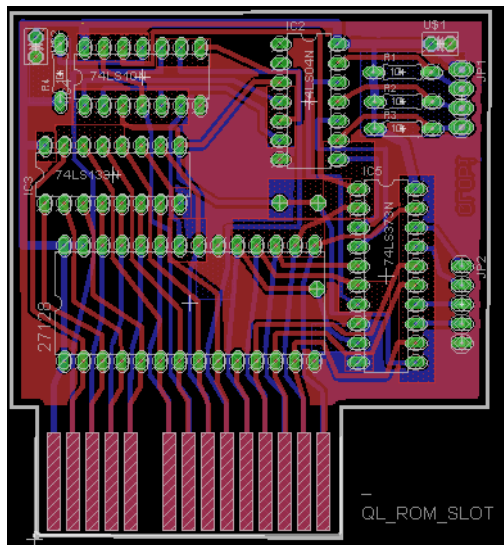
If the drawing is already in the editor you can easily make a printed circuit board from that. First you see the complete and tested drawing of the mICE.



So the general function is like for the most ROM plugins, except for 2 addresses were a 8bit-latch is visible for the software. The EPROM is a normal 16K-byte one (27128), were the last two addresses are occupied by the 8-bit latch. As already mentioned, 4 bit are used for detecting the movements in X and Y direction and 3 bits for each key. One bit is permanently pulled to low.

The latch address is hFFFF with a mirror at hFFFE, therefore this two bytes aren't available for EPROM content. The EPROM contains the software for the main grafical user interface and is started direct after the F1/F2 selection phase on startup.

If you make a printed board from the drawing you will end up with this:



Micev2.rar

The EAGLE-Files are embedded in this document as ZIP-file for both .SCH and .BRD files. These files can be opened with the free version of EAGLE and forwarded to a circuit board manufacturer.

The ready and working (home brewed) prototype is looking like this:



If you build this and plugin you will encounter that you need to connect a mouse. That will be a problem unless you have the original one. A modern PC mouse will not work due to fact it has a serial Interface as USB, PS/2 and plain RS232. Ever older ATARI mices won't work due to the differend signals. If you have an old mouse or trackball you can salvage it and convert and connect the signal direkt to the mouse. To help on that there is some info and help in appendix A. That circumstances lead to the second stage of the project:

The Project Part 2 “The PS/2 Mouse Adapter”

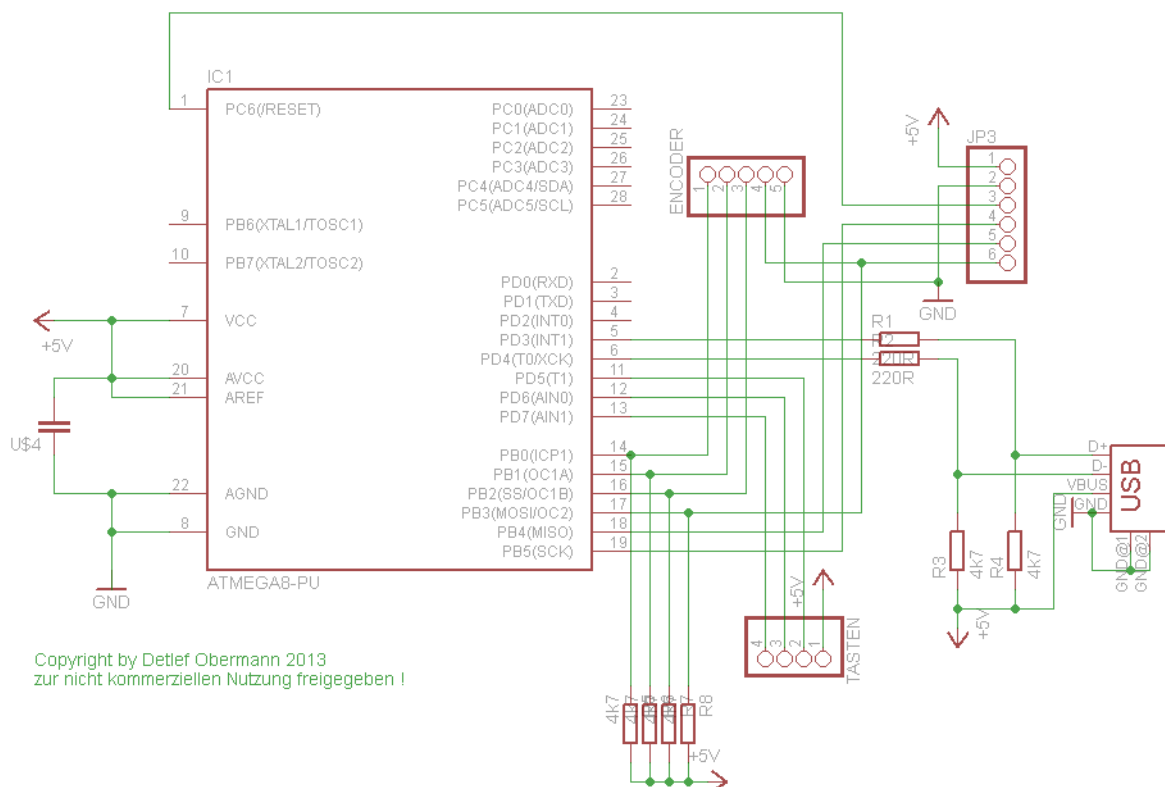
The idea is to convert signals from modern mice to free yourself from vanished hardware. If you have a closer look to modern mice you will find them with a USB interface. The idea is having a microcontroller reading the mouse and than simulate the key and encoder signals. But the microcontroller needs to be a USB master, which is possible, but complicated. On the other hand, next to all mice have a PS/2 compatability mode, which is a synchronous serial protocoLL and that is a bit easier to deal with. The mouse can be set into a polling mode so the timing is determined by the controller and not by the mouse. This prevents an overflow of the controller in the case to frequend data from mouse.

The controller reads out the mouse frequently and simulates the encoder signals for the mICE interface. This is done with an ATMEL ATmega8 microcontroller, this is because I am familiar with these. The software is written with the BASCOM compiler and published at the end of this article.

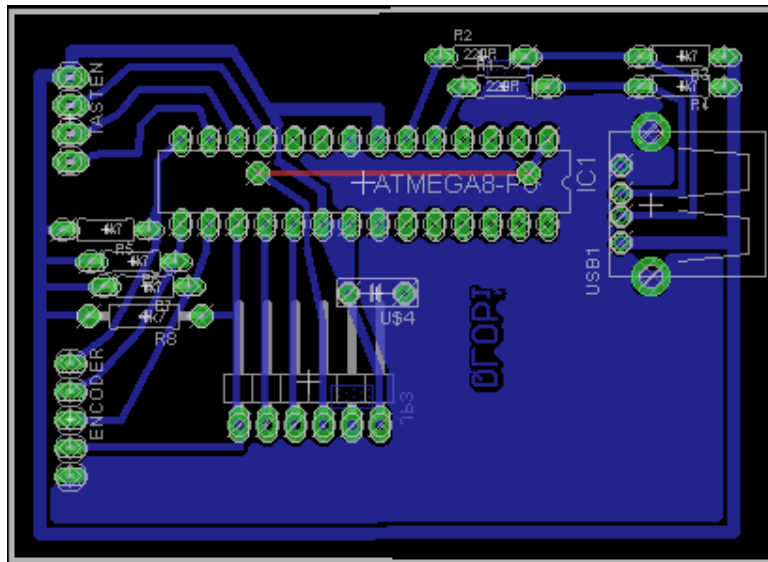
The design is for a direct plugin of a PS/2 compatible USB mouse. The use of the original PS/2 connector was a problem, because of the dimensions. The PS/2 block is too high to fit piggy pack on the mICE interface. The USB A connector is shallow enough to fit.

With this piggyback controller you can also adapt to other types of mice (ATARI or serial RS232 mice) but you have to redesign the input interface and the software. The software for the PS/2-to-encoder protocol is published later in this dcoument for own modifications.

Here is the circuit drawing of the piggyback adapter:



The layout looks like this:

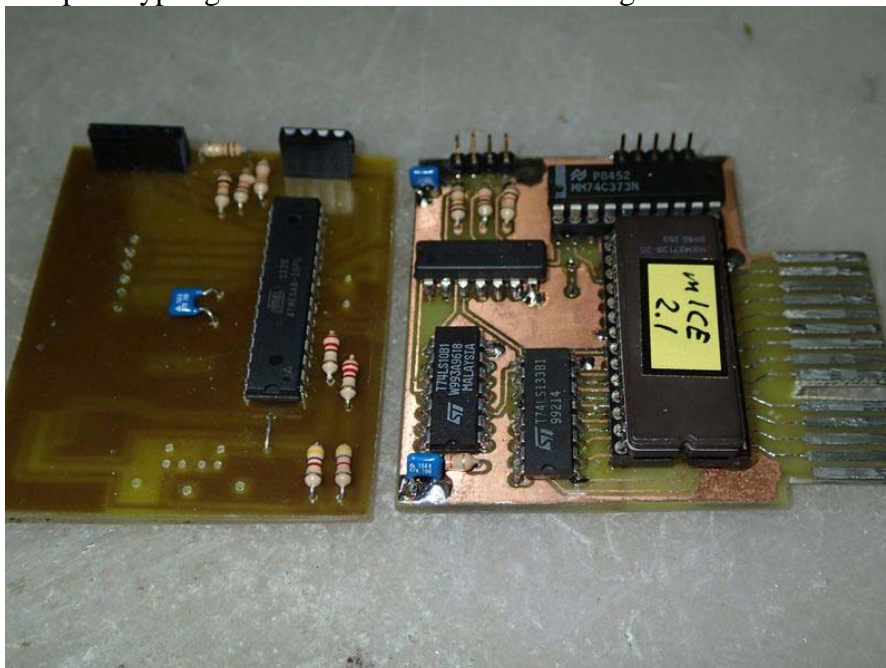


The latest EAGLE files for the adapter:

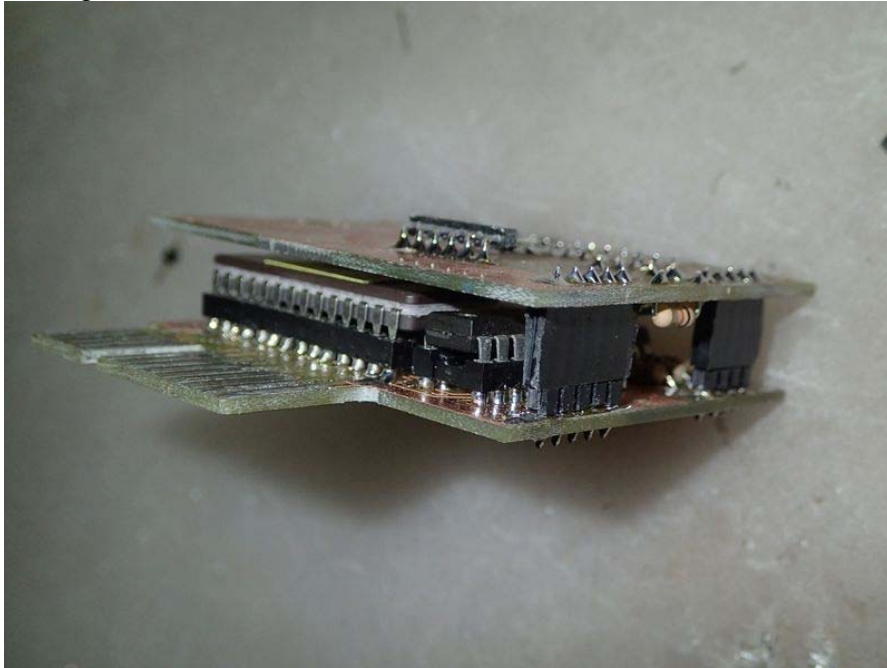


mICE Adapter.rar

The prototyp together with the mICE it's looking like this:



Put together:



The commented BASCOM software sources are here:



QL PS2 Maus V1.03.bas

The software consists of two main parts. One part is the PS/2 transmit and receive routine for the communication with the mouse. The mouse is set into polling mode, which allows the controller to determine the communication timing.

The other is the implementation of the encoder simulation, with the speciality that one signal of each encoder is inverted, compared with standard hardware encoders. If you rewrite these routines take care of this!

Due to the fact that the PS/2 protocol is a synchronous serial one, there is no special requirement for the accuracy of the controller clock. Also the encoder signals don't need to be precise. Just the speed is a matter, so clock is set to 8 Mhz on internal oscillator (+/- 3%). Other (slower) speeds may be possible, but aren't tested so far.

The only difference for the mICE interface is that signal B is inverted. The key signals are pulled down if pressed. For the opposite direction B has to start first and A is deferred.

APENDIX B

The Softwareroutines

This is a reassembly for the ROM idle routine (no key pressed and no movement). It's also possible to patch this routine and relink to your own software to establish a software version of mouse interface*. Good luck!

```
00E7FE LEA     FFFF.L, A0
00E804 JSR     280E0.L
00E80A CMPI.B  #0F, D0
00E80E BHI.S   00E818
00E810 MOVE.L  A4, D1
00E812 BNE     00E89A
00E816 BRA.S   00E81A
00E818 MOVEA.W D0, A4
00E81A MOVE.B  D0, D1
00E81C MOVE.B  D1, D2
00E81E ANDI.B  #03, D1
00E822 LSR.B   #2, D2
00E824 ANDI.B  #03, D2
00E828 CMP.B   0(A1,D3.W), D1
00E82C BEQ.S   00E866
00E82E CMP.B   0(A2,D3.W), D1
00E832 BEQ.S   00E872
00E834 CMP.B   0(A1,D4.W), D2
00E838 BEQ.S   00E880
00E83A CMP.B   0(A2,D4.W), D2
00E83E BEQ.S   00E88C
00E840 MOVE.W  D6, 96(A6)
00E844 MOVE.B  D7, 99(A6)
00E848 CMPI.B  #06, 1BC(A6)
00E84E BHI.S   00E860
00E850 ADDI.W  #4000, D7
00E854 CMPI.B  #03, 1BC(A6)
00E85A BCS.S   00E860
00E85C ADDI.W  #4000, D7
00E860 MOVE.B  D1, D3
00E862 MOVE.B  D2, D4
00E864 BRA.S   00E7FE
00E866 TST.B   D7
00E868 BEQ.S   00E834
00E86A TST.W   D7
00E86C BMI.S   00E834
00E86E SUBQ.W  #1, D7
00E870 BRA.S   00E834
00E872 CMPI.B  #F7, D7
00E876 BEQ.S   00E834
00E878 TST.W   D7
00E87A BMI.S   00E834
00E87C ADDQ.W  #1, D7
00E87E BRA.S   00E834
00E880 TST.W   D6
00E882 BEQ.S   00E840
```

```
D0 00000000 A0 0000FFFF
D1 00000000 A1 0000E8C0
D2 00000000 A2 0000E8C4
D3 00000000 A3 0002B680
D4 00000000 A4 00000000
D5 00000000 A5 00028000
D6 00000100 A6 0002B680
D7 00000080 A7 000BFFD0
SSP 00028480
```

```
PC 0000E7FE SR -----Z--
```

```
0280E0 MOVE.B (A0), D0
0280E2 RTS
```

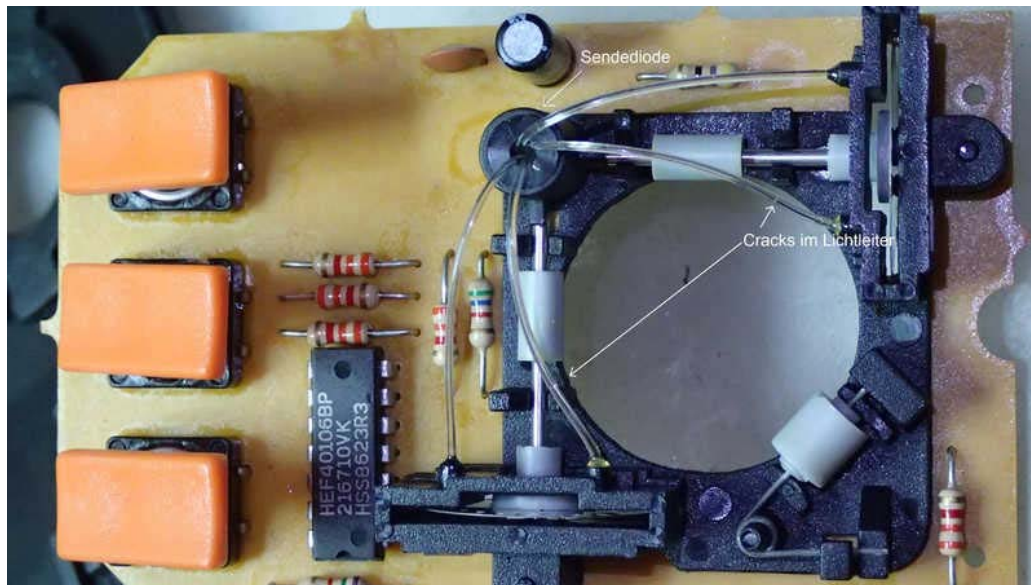
D0	D1	D2	D3	D4	D5	D6	D7
Encoder Y1	Encoder X1	Encoder X2	Encoder Y2	0	Taste 1	Taste 2	Taste 3

For the english readers "Taste" means "key".

*Interesting is the Jump at 00E804 into the RAM area. There is only the readout (move.b (A0),D0) of the external signals and a jump back. Here you can place your own routines for a solution without an hardware adapter.

APENDIX C

Views to the open EIDERSOFT mouse:



The encoder wheels with the 2 transistors:



The IR-LED and the polymer fibres.

