I2C Interface for QL Emulators Part 1

Originally Published in QL Today, Vol 15, Issue 4, May-June 2011

One of the features that the Minerva II ROM provided for the original QL hardware was the I2C I/O (Input/Output) interface. TF Services also supported this interface with it's own interface products for both parallel and analogue applications.

As always with my projects this is to give you ideas and the basics. This is not a complete project in it's own right with a real practical applications. I leave that up to you. However in another article I plan to show you what you can do with an old Sony radio to give it remote control. The techniques used can be used for other applications. More on this, another day. Also this is all at your own risk. But do have fun playing.

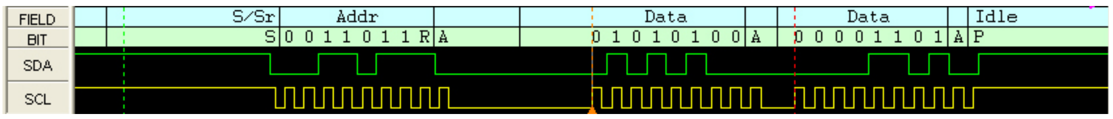
What I hope to show you in this short series is how to implement the use of the I2C interface using a QL Emulator such as QPC2. All the examples given have been tested on QPC2. But there is no reason why this should not work with other emulators. The only basic requirement is you need USB port on your computer.

Some of you may still have the original TF Services I2C interfaces, if you have, then dig them out you can reuse these.

Now there is a way of recreating this interface using a product from ByVac. Their BV4221 product provides a USB to I2C interface. This device appears as COM port just like a RS232 converter that I have used in my previous projects published in OL Today.

So back to the beginning, what is the I2C interface. This was an interface developed by Phillips. It is a two wire (plus ground) system. One wire is used a clock (SCL) and the second (SDA) as a bi directional data line. The computer, in our case a QL with a Minerva II ROM or a QL Emulator such as QPC2 with a ByVac BV4221 converter connected is the master station. This controls device(s) connected to it, which are called slaves. Each connected device has it's own address. So up to 254 devices can be connected to an I2C bus. Theses can be such things as bi directional parallel ports, analogue to digital converters, digital to analogue converters, RTC (Real Time Clock), RAM(Random Access Memory) and digitally controlled potentiometers as examples. These are most likely, the most useful to us.

Here is an example of how these signals look, and gives a little insight to the protocol of the I2C system. More on this later.



The slave devices in most cases have up to 8 addresses available via 3 address connections in the chips themselves. Each type of device has it's own range of addresses. So using for example PCF8574 parallel I/O IC, you can use eight of these, in the address range 64 to 78 (Decimal), one device for each even address, I will come back to the odd numbers later. In fact you can have a further eight devices connected by using the PCF8574A device, which have the address range 112 to 126, again the even numbers. There are exceptions, for example the DS1307 RTC(Real Time Colock) device has only a single address. But it is unlikely that you would need more than one real time clock. However if you did there is nothing to stop you using two BV4221 converters on an emulator. Something you could not do with the original Minerva II QL solution.

Now as I have pointed out above the addresses I have quoted are even numbers. The reason for this is the 7 most significant bits are in fact the device address in the true term. The least significant bit is used to define reading or writing to a device. The rule being, even numbers are write addresses and odd numbers are read addresses. So for example if we take PCF8574 with all address pins held low, the write address would be 160 and the read address would be 161. See the address table. This show all the addresses for all the devices featured in this series of articles.

I2C Interface
Addresses

Address Ranges	Read	Write

256x8 bit RAM

1010000	161	160
1010001	163	162
1010010	165	164
1010011	167	166
1010100	169	168
1010101	171	170
1010110	173	172
1010111	175	174

Parallel (PCF8574)

0100000	65	64
0100001	67	66
0100010	69	68
0100011	71	70
0100100	73	72
0100101	74	74
0100110	75	76
0100111	76	78

Parallel (PCF8574A)

0111000	113	112
0111001	115	114
0111010	117	116
0111011	119	118
0111100	121	120
0111101	123	122
0111110	125	124
0111111	127	126

A/D and D/A
(PCF8591)

1001000	145	144
1001001	147	146
1001010	149	148
1001011	151	150
1001100	153	152
1001101	155	154
1001110	157	156
1001111	159	158

RTC [Real Time
Clock](DS1307)

1101000	209	208

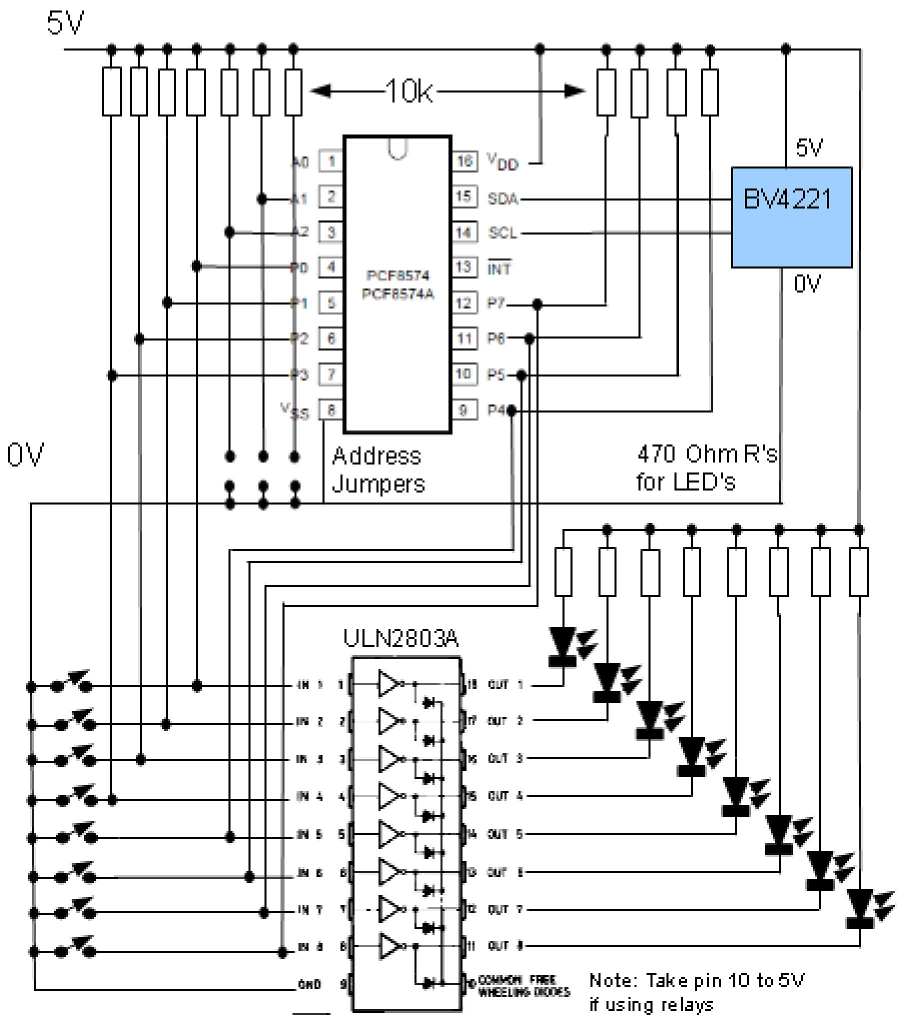
Digital Potentiometer
(DS1803)

0101000	81	80
0101001	83	82
0101010	85	84
0101011	87	86
0101100	89	88
0101101	91	90
0101110	93	92
0101111	95	94

So that I hope gives you a basic idea if what I2C interfacing is. So now how to implement this in practice.

I built myself a test board built on Vero Board, containing a PCF8574(A) parallel chip with LED’s and switches to input and output to the chip. A PCF8591 AD and DA chip, a DS1307 RTC(Real Time Clock and finally a PCF8570 RAM chip. You do not have to build all of this if you don’t wish to. For the purposes of the series of articles I have spilt this in to each chip in it’s own right.

I am going to start with the PCF8574(A) parallel chip.



The circuit shown above has the I2C parallel IC (PCF8574) shown at the top. This is the heart of this circuit the rest are the LED's and a driver (ULN2803A) and switches to provide output indication and inputs. This is so you can fully experiment with this device. The reason for using the ULN2803A driver is two fold. First, the output from the PCF8574 could drive LED's directly, but the LED and it resistor has to be from +5V supply to the port this means when the output from the PCF8574 is high, that is a "1" the LED will not light. So can get a bit confusing. The ULN2803A acts as an inverter, so that when the port are high, the LED's light. The second point is the output current from the PCF8574 is limited, so if you want to drive relays for example you need a driver for these as well. The ULN2803A also contains the back EMF diodes so the device does not blow when the relay is released. If you are using relays please take pin 10 of the ULN2803A to +5V this connects all the output diodes together and provides the required protection.

All the ports are "pulled high" with the 10K resistors, so that the switches, when 'on', pull the inputs low. It is best to start with the all the switches are off. You have to ensure the PCF8574 outputs are high first. I will come back to this when we talk about the software.

The three address lines, A0, A1 and A2 are also “pulled” high as well. The jumpers pulling these low when required. You will see from the address table above this will make address for the PCF8574 ‘78’ and for the PCF8574A ‘126’. If you wish to use more than one of each device type on a I2C bus then you need to fit jumpers as required. By using both PCF8574’s and PCF8574A’s you can have up to 128 I/O lines on a single I2C bus. That is 8 x PCF8574’s providing 8 lines each and 8 x PCF8574A’s again providing 8 lines each. That should be more than enough for most applications.

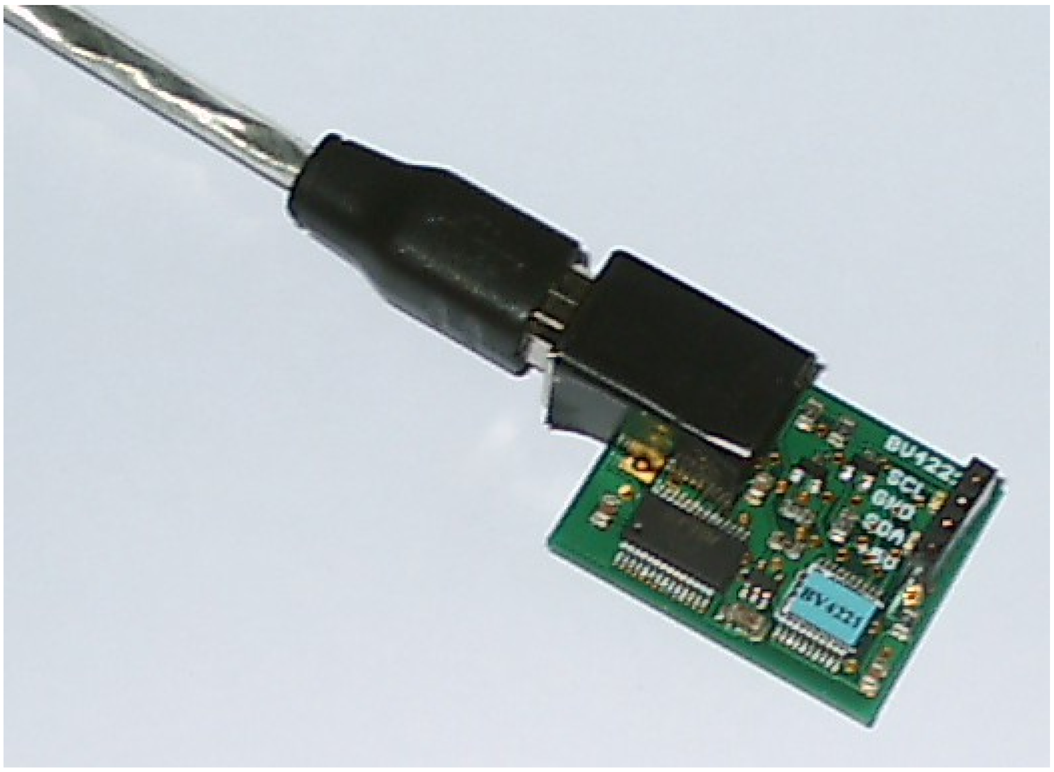
As to construction, stripboard is fine for this, just remember the cut the tracks as required on the reverse of the board. The photograph of my test board above should help you with the layout of this and the wire jumpers to build the circuit. Just follow the circuit diagram.

Main Parts list

Part	Supplier	Part Number	Price each	Total
BV4221-V2	ByVac	BV4221-V2	£24.98	£24.98
PCF8574	RS Components	517-0687	£1.31 (Come in packs of five)	£6.55
PCF8574A	RS Components	517-0249	£1.63 (Come in packs of five)	£8.15
ULN2803A	RS Components	714-1167	£0.42 (Come in packs of five)	£2.10
8 x 470R Resistor	Maplin	M470R	25p	£2.00
11 x 10K Resistor	Maplin	M10K	25P	£2.50
8 x LED’s	Maplin	CJ58N	54P	£4.32
Stripboard	Maplin	JP51F	£5.29	£5,29

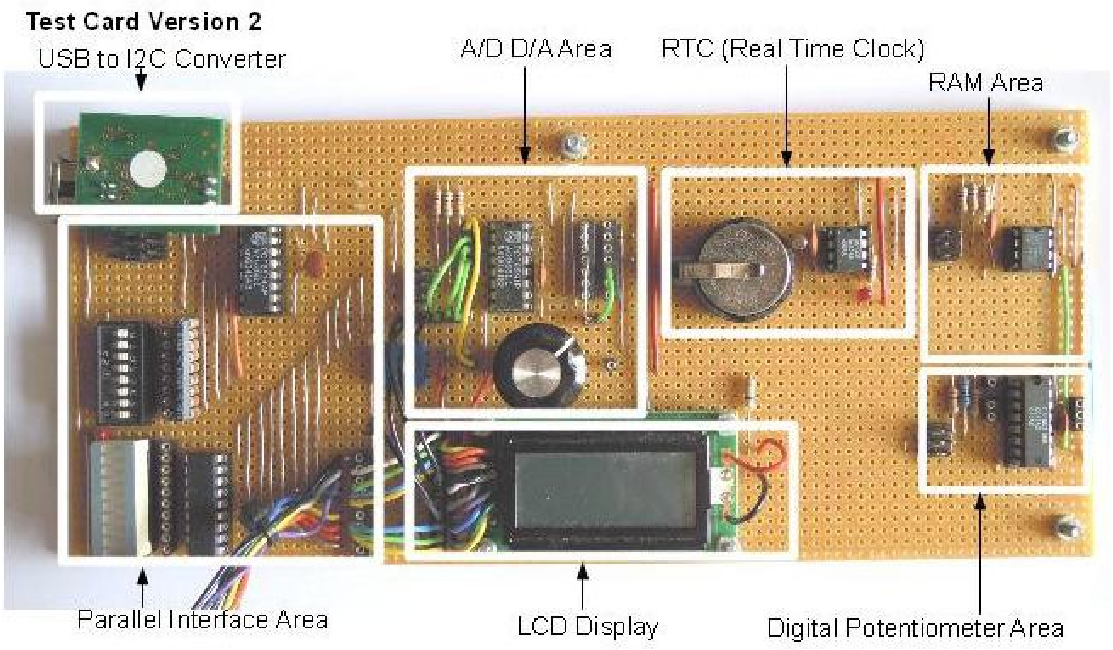
You may wish to use IC sockets, which is not a bad idea, also you will need wire for the links etc. The BV4221-V2 is the most up to date version of the USB to I2C converter it is not the same size as the version I used. The prices are correct at time of writing, You may wish to find other suppliers that may be cheaper or do not have the minimum order quantities.

One of the nice features of this project is that the power comes from the USB port of your computer. However be careful not to load it too much. This simple test circuit is fine. But care needs to be taken if more than 100mA of current is required, this can cause problems and even damage to your computer. So if you think you will require more power such as powering relays for example, then using a stand alone PSU is a good idea. In this case, take the 5V wire from the BV4221 converter and connect your 5V PSU here.



For those not so happy with soldering there is a simpler way, depending on your requirements. There is a range of pre built cards from ByVac, I have listed some of the options below, but do please look at the ByVac's web site for the most up to date information since they are constantly updating and changing their products. I must point out I have no connection with ByVac, I just use their products. There are others in the market. So worth a search on the internet.

Test card with labels



Now the software

```
5 CLS
10 BAUD 115200
15 parallel=126:REMark PCF8574A address, all address links open
17 adda=158:REMark PCF8591 address, all address links open
20 OPEN#3;ser2ir:REMark i=ignor hardware handshake, r=raw data
30 PRINT#3;CHR$(13);:REMark Carriage Return to set the baud rate in the USB to
I2C converter, required on first pass to initialise USB to I2C converter.
40 print_reply
50 PRINT
60 PRINT#3;"V";CHR$(13);:REMark Command to USB to I2C coverter for
firmware version.
65 PRINT "Return USB Converter Version Number:-";
70 print_reply:REMark Prints version number reply from USB to I2C converter
80 PRINT#3;"D";CHR$(13);:REMark Sets USB to I2C coverter to receive decimal
numbers, default is hex numbers.
85 PRINT "Decimal Mode Selected"
90 print_reply
100 ledflash

110 ledcount
120 input_test
200 PRINT "End":CLOSE#3:STOP
210 :
```

```
1000 DEFine PROCedure print_reply
1010 REPeat loop
1020 a$=INKEY$(#3)
1030 b$=a$
1040 PRINT b$;
1050 IF a$=">" THEN EXIT loop
1060 END REPeat loop
1070 END DEFine print_reply
1080 :
1090 :
1100 DEFine PROCedure non_print_reply
1110 REPeat loop
1120 a$=INKEY$(#3)
1140 IF a$=">" THEN EXIT loop
1150 END REPeat loop
1160 END DEFine non_print_reply
1170 :
2000 DEFine PROCedure ledflash
2010 FOR a=1 TO 10
2020 PRINT#3;"s-";parallel;" 255 p";CHR$(13);:REMark s=start message to USB to
I2C converter, p=end of message to USB to I2C converter.
2030 PAUSE 25
2040 PRINT#3;"s-";parallel;" 0 p";CHR$(13);
2050 PAUSE 25
2060 NEXT a
2070 END DEFine ledflash
2080 :
3000 DEFine PROCedure ledcount
3010 FOR a=0 TO 255
3015 PRINT a
3020 PRINT#3;"s-";parallel;" ";a;"p";CHR$(13);
3025 print_reply
3030 PAUSE 25
3040 NEXT a
3050 END DEFine ledcount
3060 :
4000 DEFine PROCedure input_test
4010 REPeat input_loop
4020 PRINT#3;"s-";parallel;" 255 p";CHR$(13);:REMark need to ensure any lines
used as an input are set high.
4030 non_print_reply:REMark Stops printing the USB to I2C reply
4040 PRINT#3;"s-";parallel+1;" g p";CHR$(13);:REMark reads input data, note: the
need to add 1 to the address variable, this puts the I2C IS's into read data mode.
4050 print_reply:REMark prints both the address reply plus the data.
4060 IF INKEY$=" " THEN EXIT input_loop

4070 END REPeat input_loop
4080 END DEFine input_test
4090 :
```

The above routine test the parallel port, both the output and input features.
So if all is OK, run this program will return the firmware version of the USB to I2C converter and then all the LED's should start to flash. They will flash 10 times at 0.5s interval's. Then there is a binary count from 0 to 255, this tests all permutations of the outputs. Then the final part of the program tests the input, just switch any or all of the switches 'on' and the numbers change on screen.

This simple routine show how to use the protocol and who this works with the PCF8574(A) chip.

Line 10 set the baud rate of the port to be used. This can be any where in the range 9600 to 115,200 baud in there standard increments.

Line 30 sends CR (carriage return) to the USB to I2C converter. The converter sees this and it is this that set the baud rate automatically on the converter itself.

Line 60 sends the character "V" followed by a CR, all commands to the converter require a CR at the end to tell the converter this string has ended.

Line 80 set the converter into 'Decimal' mode, default is 'Hex' mode. I change it to decimal because it is easier to handle.

You will note that when the converter replies it returns the address of the first device it sees. So for example "126>". The reply routines are there to read these replies and stop them filling up the COM input buffer. Also the data from read commands, will be extracted from these replies as well.

You will see from the first diagram with the waveforms, that in the bit stream row the line start with an "s" this is the start command. Which is then followed by the address for the device we wish to control. If there is only one device then you need only send the "s" command. However I feel it is good practice to send the address of the device we wish to control, it saves rewriting thinks later if there are more devices added later. So the start will look like "s-126(The Device Address)….

Now we can add the data we wish to send to the device. In this case make all the output go 'high'. "s-126 255 p". So the "s" is the start as before followed by the device address, then the number 255 which will set all the output 'High then "s" which is the "Stop Condition". Don't forget are CR at the end for it all to happen.

Now that sends commands to the device so who about reading data. This is were the odd address number come in. Remember the address is the 7 MSB (Most Significant Bits) but there are 8 bits. The LSB (Least Significant Bit) is the one that controls the read/Write function of the I2C bus. So we just add "1" to the address to read data from the chip. Hence the address is 126 to write to the device and 127 to read the device. So the command now looks like this, "s-127 g p". So what does the "g" mean I hear you ask. It is the command to 'Get' data from the device. So I have just you some basic ideas of the concepts involved so do read the BV4221 pdf for more details on how to use the converter.

Next time we will look at the AD/DA, RTC and RAM chips and their use.

Some ByVac I2C products:-

BV4502	I2C 2 Relay
BV4627-B	I2C 8 Relay
BV4236	I2C RTC and Temp
BV4237	I2C RTC, Temp and ADC
BV4506	I2C Keyboard Controller
BV700-B	Small Black keypad for use with the above keyboard controller
BV4240	I2C Bus Extender

Range of display controllers, to many to list here, so please see ByVac's web site. www.byvac.com. The select 'shop', where you will find all the information on these products.
(Please note, The software with this article was not design for these interfaces, so may need adapting)

References

http://www.byvac.com/bv3/index.php?route=product/product&product_id=88
(Please note, I used the original V1 of the BV4221, ByVac now supply V2 which also has a SPI interface. The commands are the same, so the programs listed in the article should still work.)

http://www.byvac.com/bv3/index.php?route=product/category&path=44

PCF8570 Ram Data Sheet
http://www.nxp.com/documents/data_sheet/PCF8570.pdf

PCF8574(A) Data Sheet
http://www.nxp.com/documents/data_sheet/PCF8574.pdf
http://focus.ti.com/lit/ds/symlink/pcf8574.pdf

PCF8591 Data Sheet
http://www.nxp.com/documents/data_sheet/PCF8591.pdf

DS1307 RTC (Real Time Clock)
http://datasheets.maxim-ic.com/en/ds/DS1307.pdf

DS1803 Digital Potentiometer Data Sheet
http://datasheets.maxim-ic.com/en/ds/DS1803.pdf

I2C Tutorials
http://www.robot-electronics.co.uk/acatalog/I2C_Tutorial.html
http://www.i2c.byvac.com/ar_foundation.php

TF Services I2C manual
http://www.dilwyn.me.uk/docs/manuals/index.html

Advanced I2C information, but still worth a read to understand I2C protocols
http://www.nxp.com/documents/user_manual/UM10204.pdf