RAGCHEW

JANUARY 2019

HAPPY NEW YEAR!

FROM THE EDITOR - G4CIB

May I take this opportunity to wish all members of GARES a Happy New Year and a Healthy and Prosperous 2019.

On 9th January 1969 I turned up at the "Lamb Inn" by the entrance of Kings Square for a meeting of the then Gloucester Amateur Radio Society. The little "signing in" book does not record the meeting details and time has dimmed my memory!

I had first become aware of Amateur Radio in the late 1950s when I listened to the 40m band on my father's radiogram. CW and AM (amplitude modulation) were the modes of choice, with the occasional SSB (single sideband) station. When I left school I went to what was then Rugby College of Engineering Technology to study electrical engineering. The college had an amateur radio station with its own call sign - G3VWI - and my interest was rekindled which resulted in me taking the City and Guilds Radio Amateurs Exam (RAE) in the summer of 1968 just before I left college. To get a Class A call sign enabling you to operate on the HF bands, a pass in the RAE and a 12wpm Morse test was mandatory, but a Class B licence was available with the RAE pass. The lowest frequency available, however, was 432MHz (70cm) and with no commercially available equipment, this was definitely an experimenters band. Around this time the rules were relaxed allowing Class B licence holders access to the 144MHz (2m) band. Returning to Gloucester I started a Graduate Apprenticeship at Smiths Industries in Bishop's Cleeve in the autumn of 1968 and amateur radio was again put on the "back-burner". Once again my memory is hazy how I found out about the GARS meetings at the "Lamb Inn" and I seem to remember that I only attended a few meetings there before the club moved to the RAFA (Royal Air Force Association) Club in Spa Road as the whole of Kings Square and the surrounding area was being razed to the ground for redevelopment.

The technical advances in our hobby have been enormous during the 50 years I have been a member of the club and the opportunities for experimentation continue to expand exponentially. Member **John Rowing M0NRZ** has submitted an article on the development of a CI-V (remote) interface for his IC7300. Light years away from the technology available when I joined the club 50 years ago!

73 and good DX!

Brian G4CIB

New Year - New Challenges

How about trying something new in 2019? In the spirit of self training how about trying a new mode? Perhaps you have always run high power? How about trying some QRP? Whatever new challenge you take on, how about writing an article for "Ragchew"? Email your article to me at **g4cib@outlook.com**

A New 2m Beacon

Many thanks to **Dave G4BCA** for alerting me to a new 2 metre beacon **GB3SEV** on 144.432MHz, located at IO82UI, Stourport-on-Severn. The antenna is a 4 element yagi but at the time of writing this I am not sure of the beam heading. The licence holder is well-known 2m UKAC contester **M0VXX.**

Heathkit Now Becoming Collectable!

A few weeks ago I was casually browsing on ebay for nothing in particular when I came across some Heathkit items. Further investigation revealed that many of their products have become sought after and are fetching serious money!

A further search and I discovered the following web sites:-

https://shop.heathkit.com/shop

http://heathkit.glosnet.com/

Development of a CI-V (Remote) Interface for the Icom IC-7300

By John Rowing M0NRZ

This is a commentary on a work in progress using an Arduino Mega 2560, an Ethernet Shield, a prototyping shield and a handful of low cost components. The Arduino IDE used to program the device is available for free download at www.arduino.cc but once you master it, I suggest that it's worth making a small donation to the Arduino site to ensure that the IDE can continue to be developed. There are also some very good tutorials on You-Tube : I recommend the ones by Jeremy Blum.

Also take a look at Processing.org for development of 'sister' apps to run on your PC or Mac.

Evolution of a multi-purpose Arduino based CI-V interface

Back in the summer of 2016, I began the process of becoming a licensed radio amateur. It turned out to be a very fortunate starting point as the ICOM 7300 had just become available in the UK, so I bought one.

I do like my rig, and it fits neatly into the fairly small space which is barely worthy of the name 'shack', let's just say it's 'cosy', tucked into a remote corner of the house.

After a few 100 or so hours of listening, occasionally calling CQ and a few brief QSO's I began to think that maybe I should have bought a bigger radio.

To be specific, I would have liked a few more buttons on the front panel, to give me direct access to the mode selections and of course direct access to the 20m, 40m, 80m bands.

I do like my radio, but I don't like multi-layer 'soft buttons'.

I did a little more reading of the manuals, and discovered the huge potential offered by the CI-V (remote) interface. It seemed like just about every internal function of the 7300 can be interrogated or set via this interface. And to my great delight setting the band, modes and VFO frequency are just a few bytes away.

The plan :

1. create a device which has a hardware CI-V interface for direct connection into the REMOTE ('one wire bus') port.

2. write the software (C++) to send specific short strings of bytes to the radio via the CI-V interface.

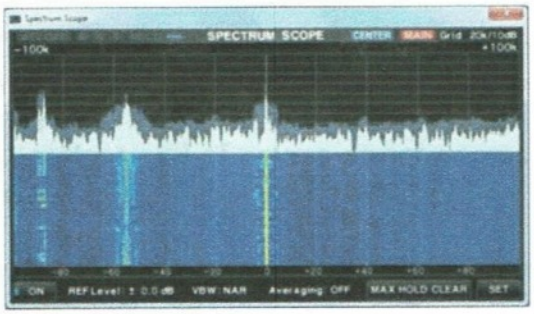
3. Assemble a few buttons (on a panel) to sit adjacent to the radio that are scanned by my software too.

I started working on the project late in 2016, but other things cropped up which needed my attention.

One day perhaps I will recreate something like this panel to go alongside my 7300.

I should mention that I did also purchase the icom RS-BA1 software together with the RC-28 remote dial. I kind of got it to work, but I think there were some issues with the early versions of the software; I am NOT a great fan of MS Windows. The RS-BA1 is quite impressive and can be a real joy when it works, but I think my PC's don't always play ball.

I was however inspired by it to attempt to make something which might be a little more robust, and perhaps add a few functions.



I'm not going to describe much detail of the CI-V message structure as there are several documents available which do precisely that. But a word or two about the 'one wire bus' are necessary to understand some of the contents of this document.

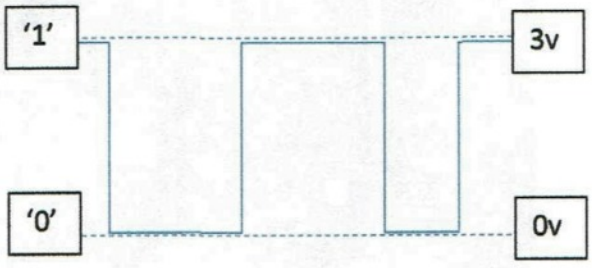
First, being pedantic it's actually two conductors, one is at CHASSIS potential (notionally GROUND), the other is the single wire on which all data will be carried to all connected devices.

There are similarities to ETHERNET (10BASE2) computer networking in the 1980's. *Does anyone remember using BNC 'T-pieces', RG-58 coax and 50 ohm terminators to play Collaborative or DeathMatch DOOM in the office ?*

On a network configuration like this, all devices 'listen' for traffic, and ideally only one device 'speaks' at a time.

Like ethernet, the icom CI-V protocol has mechanisms to handle data collisions.

In the idle state (i.e. most of the time) the CI-V data line is at a LOGIC 1 state, a potential of about +3v above ground. The magic happens when this line is pulled down to a LOGIC 0 (a potential of about GND) by any connected device.



Data from a radio can be requested from another device (the request includes the address of both the originator and destination), OR data can be broadcast to the bus for any device to act on or ignore.

I noticed that when I change mode or alter the VCO on my R8600 it outputs a short message of about 10 bytes, the source address is it's own id (96 - hexadecimal) but the destination address is 00 which I assume is the equivalent of "broadcast" i.e. any device can accept. There is a wealth of information in the ICOM CI-V manuals but I didn't spot this detail.

Short VCO change message example

HEXADECIMAL

FE	FE	00	96	00	20	50	55	00	FD
254	254	000	150	000	032	080	085	000	253

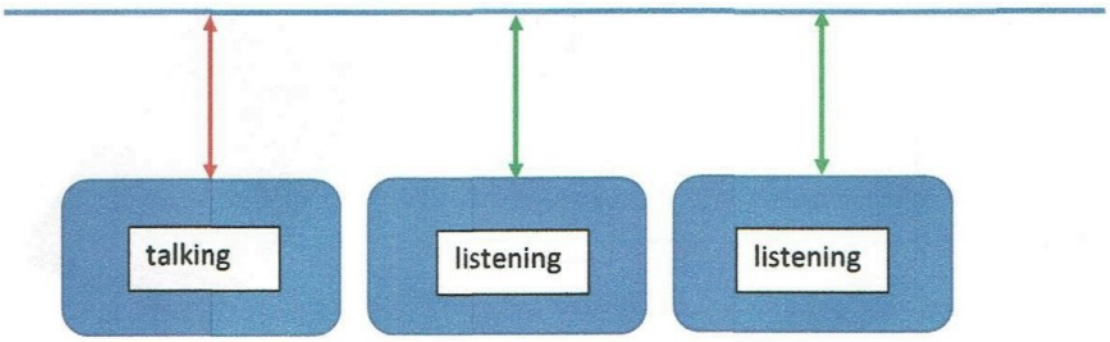
DECIMAL

I connected my 7300 to the 8600 using the remote ports 3.5mm jack to jack, if I change the VCO on either radio, they both change frequency simultaneously. Similarly, when I change from for example LSB to AM - both devices stay in sync. I haven't actually looked at the traffic - more on that later perhaps.

Note :

In the Connectors section of the Function menu, the CI-V Tranceive function needs to be ON for this to work.
The default condition in both the 7300 and the 8600 is ON

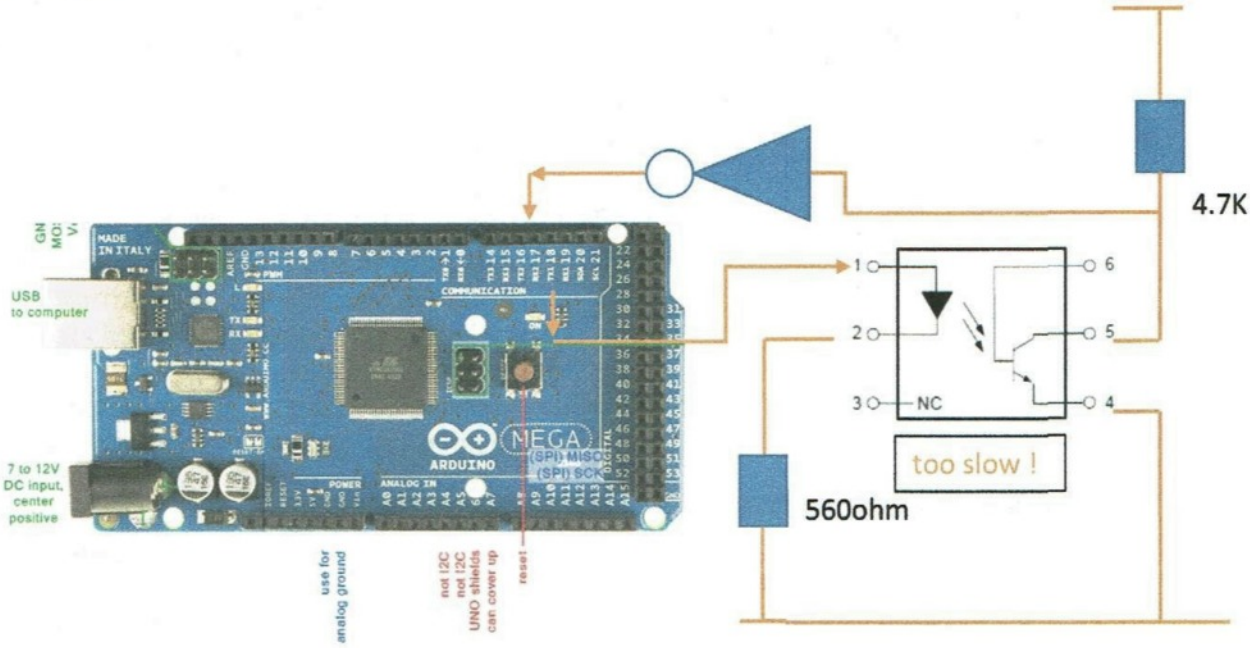
Getting back into the detail, as your device transmits data to the 'one wire bus' it must simultaneously read back each bit and byte on the 'bus', and check that what was read is identical to what was sent. If it isn't, then another device probably sent some data during 'your transmission' and all or part of your (and their) transmitted data was corrupted.



Fast forward to September 2018, and the project restarted.

Task 1 : create a device which has a hardware CI-V interface

I had decided to use an Arduino Mega 2560 as the platform for Task 2 (and despite it having 5v logic), I thought that by using a couple of opto-isolators (that I had in stock) I might resolve the physical connection mis-match as the 'one wire bus' uses 3.3v logic.



During my initial testing I sent a few ASCII characters out of pin18(TX1) to drive the opto-isolator. It is possible to either source or sink current via the Arduino's pins... I chose to have a LOGIC 1 light the internal LED. That turns on the internal transistor and 'grounds' pin 5 - a LOGIC 0 at the collector is inverted (becomes a LOGIC 1) and is fed into the Arduino's pin17(RX2).

I could simultaneously send and receive ASCII characters, but at 9600 baud I saw errors passing through the opto-isolator. Digging deeper, my oscilloscope revealed that the rising edges of the output (on the transistors collector) were profoundly rounded. At low baud rates the data appeared to be ok, but as the bit duration decreased errors occurred. Note : in fig 1 the yellow trace is the input signal, but in fig 2 & 3 the magenta trace is the input signal.

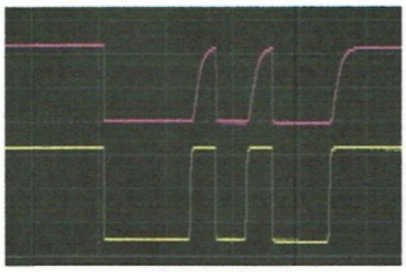


Fig.1

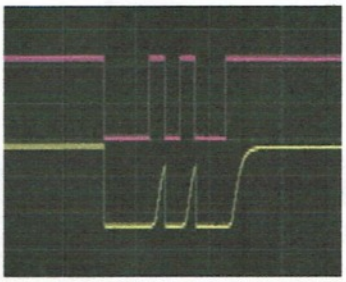


Fig.2

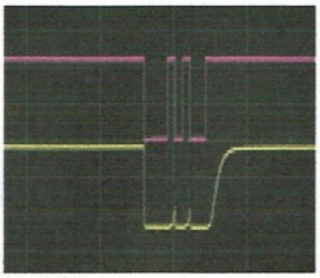
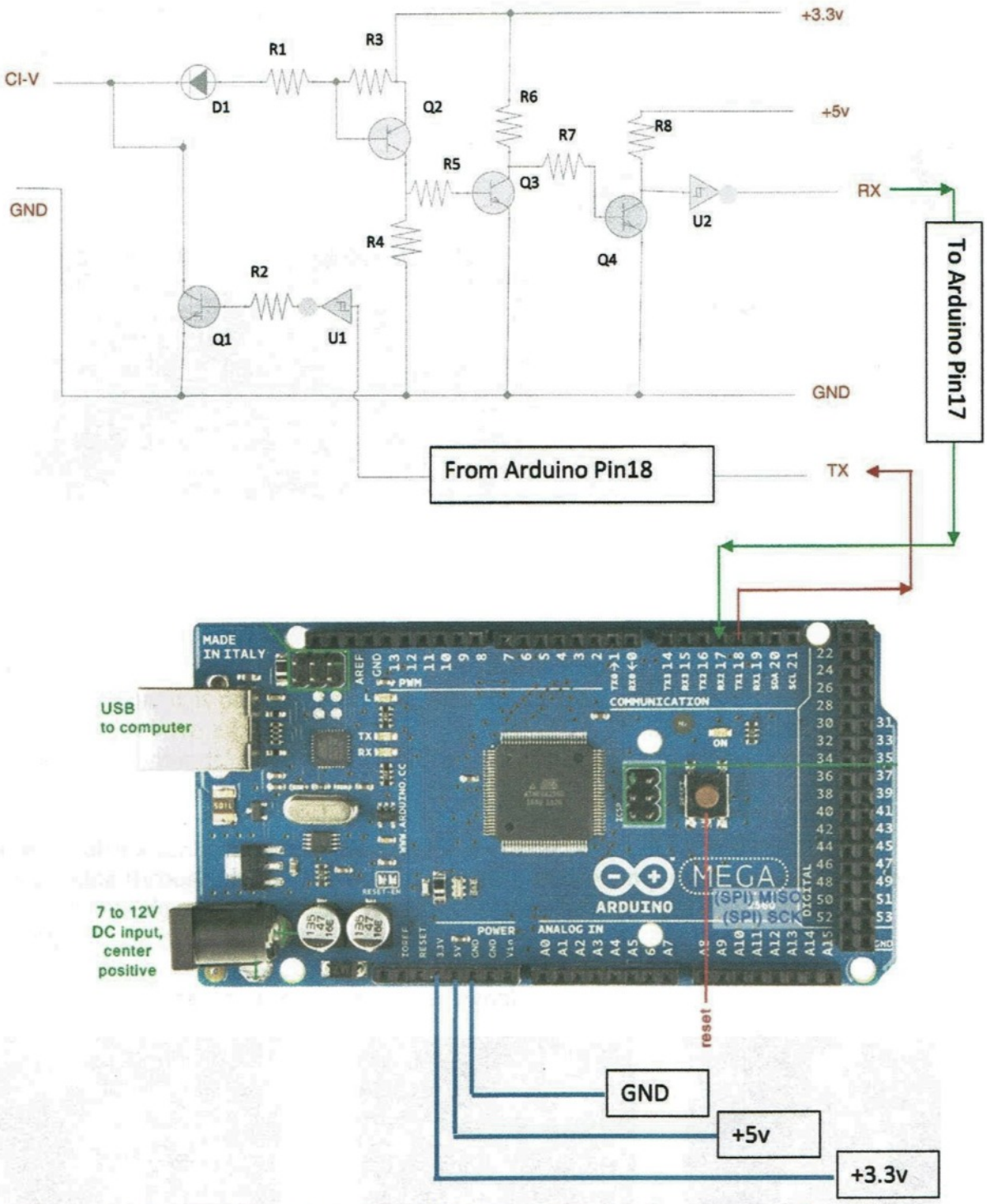


Fig.3

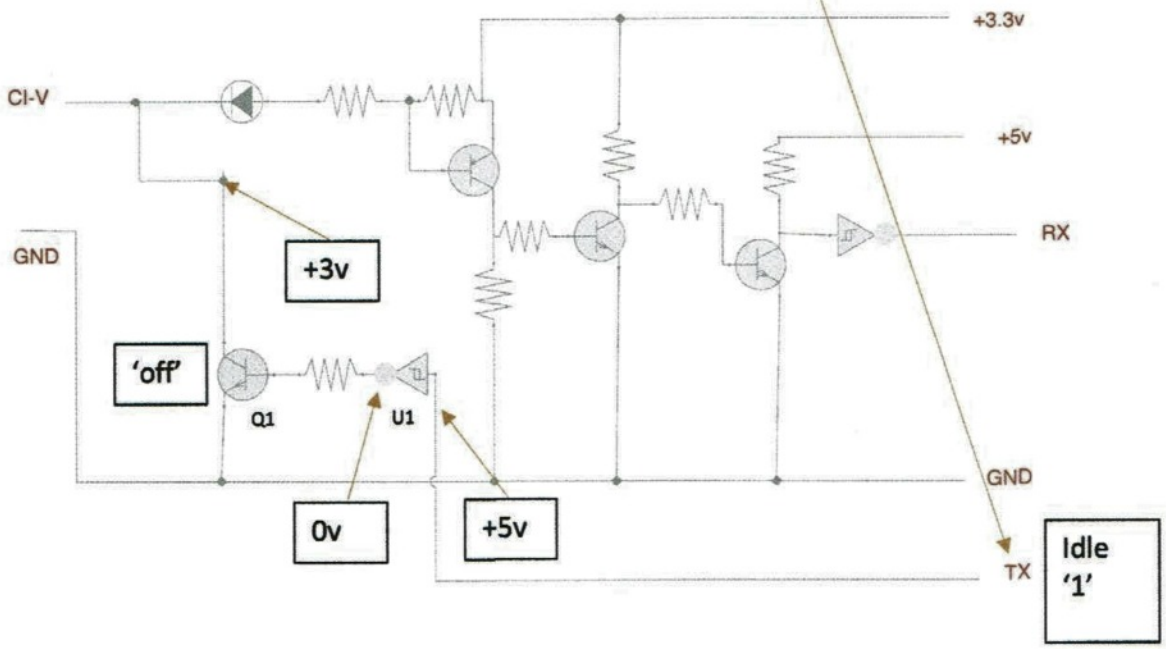
I did try various modifications to the previous circuit, but eventually abandoned it in favour of using this circuit which is very similar to that inside the ICOM 7300. In the text and diagrams below I have provided a fairly thorough (and lengthy) description of what occurs to aid anyone who is not familiar with this sort of stuff.



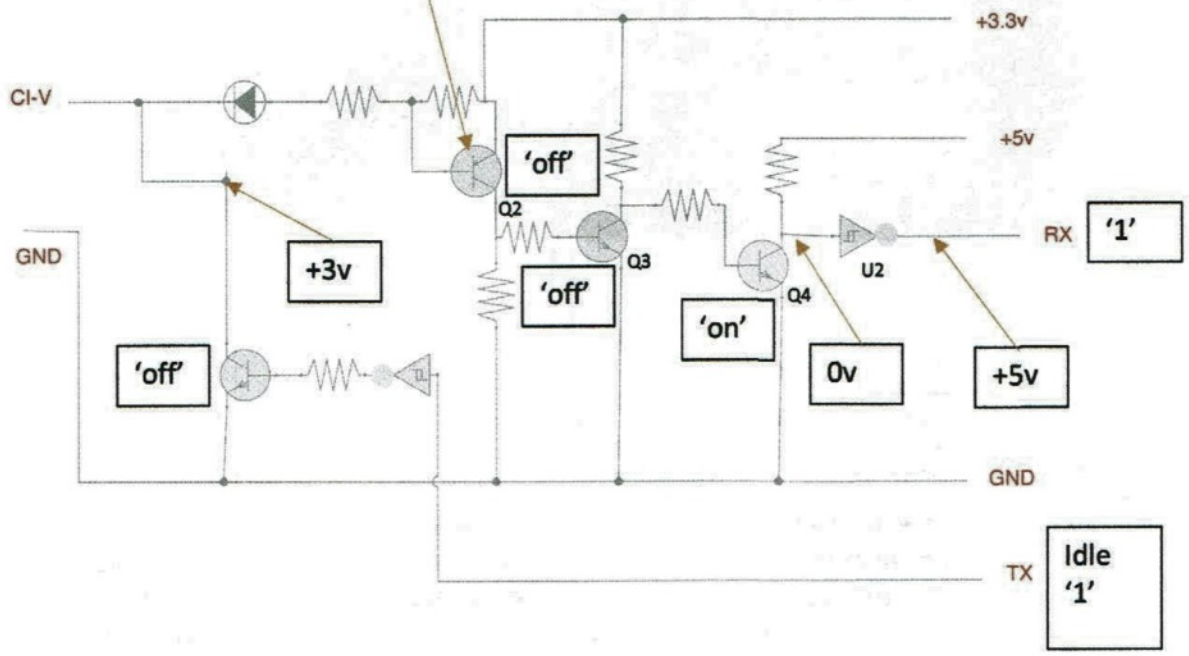
Arduino essentials: the board has many I/O pins. Some also have special functions. Most pins can be set (by code) to be IN or OUT, the code which runs is used to READ or SET the logic state of most of the pins, and make decisions. That might sound scary - it isn't, and it unleashes your imagination too.

In the IDLE state (below) I've left some component ID's off to reduce the clutter.

The 'TX' line on the RHS is connected to the Arduino pin18. When idle, it is at a LOGIC 1 - which is inverted by U1, so that a LOGIC 0 is applied to the base of Q1, which is biased off. This allows the CI-V line to be pulled (via D1) up to about +3v.



At the same time transistor Q2 is 'off', the base of Q3 is pulled to GND so it's also 'off' too. The collector of Q3 is pulled 'high' (up to about +3.3v), Q4 base voltage is thus higher than it's emitter biasing Q4 on, and taking it's collector low (near GND potential) - U2 inverts that to produce a LOGIC 1 at the RX line which is connected to the Arduino pin17.



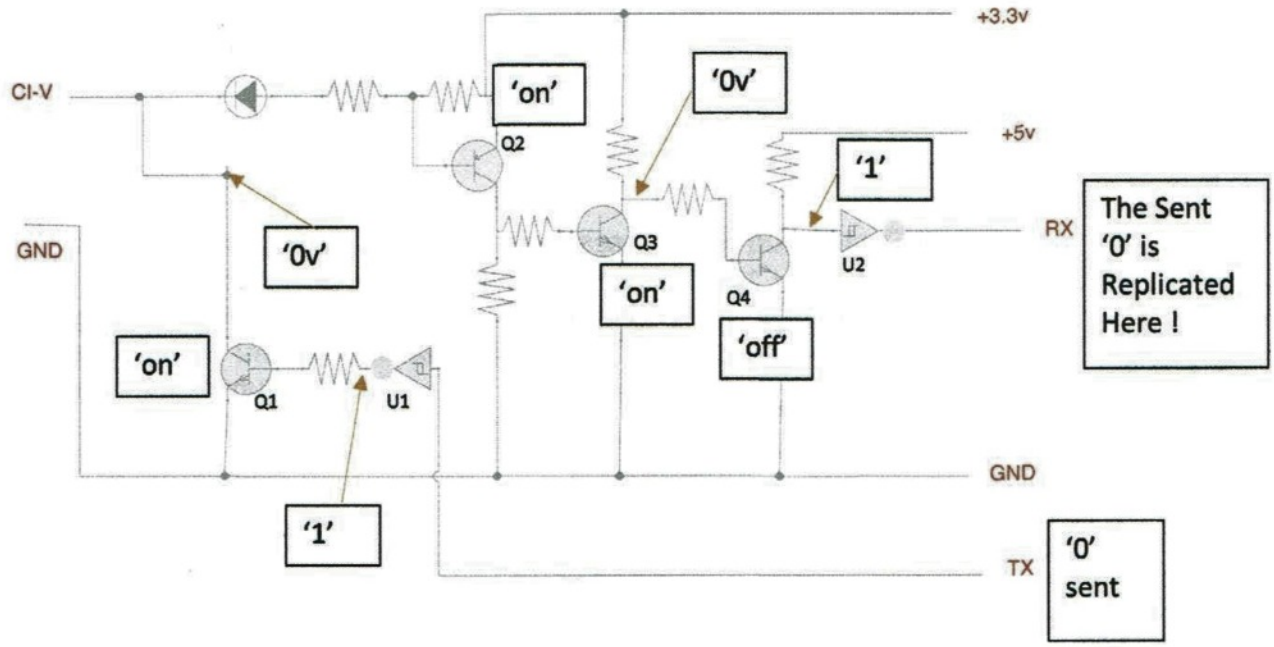
If the Arduino pin 18 and the TX line goes to a LOGIC 0 (i.e. a '0' bit is being output from the Arduino), U1 inverts it, and turns Q1 on which 'grounds' the CI-V line... this causes a small current to flow through R3 which develops a potential difference between the base and emitter of Q2... enough to turn it on.

The voltage at the base of Q3 rises, turning Q3 on, pulling it's collector low (near to ground potential).

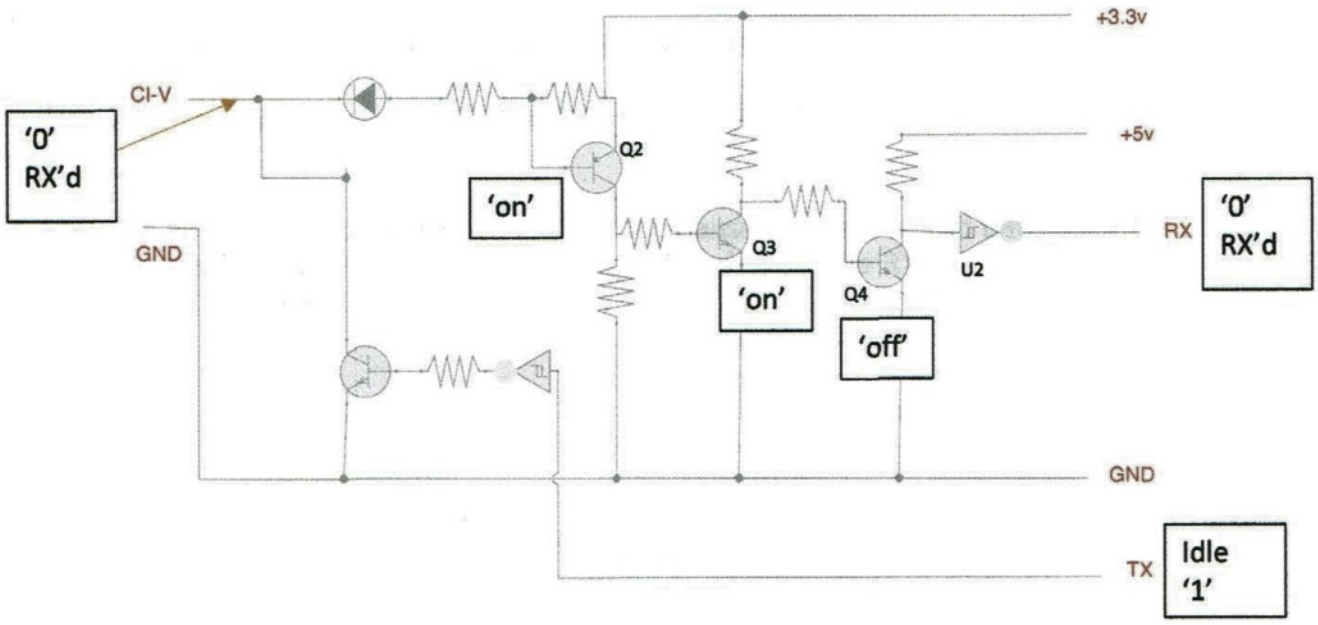
Q4 turns off and it's collector is pulled towards the +5v rail.

Finally, the inverter outputs a LOGIC 0 to feed Pin17 of the Arduino.

This is how change of state of the CI-V line cause by transmitted data from the Arduino is fed back through D1, Q2, Q3, Q4 and U2 to the RX line : this enables what is SENT to be READ and VERIFIED.

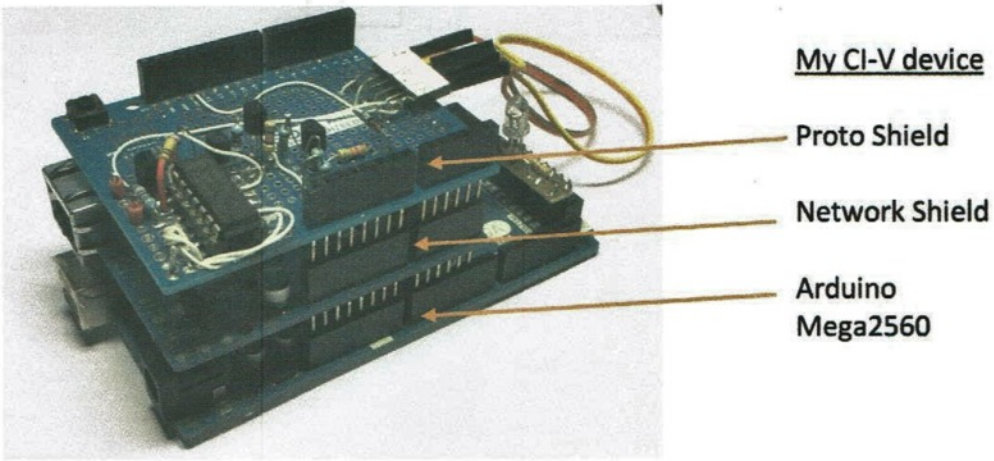


If another device on the CI-V line pulls the line low - to a LOGIC 0, just like the description above...a small current through R3 develops a potential difference between the base and emitter of Q2 which turns it on... and via Q3, Q4 and U2 a LOGIC 0 is passed to Pin18 of the Arduino.



With this circuit, I was able to connect my R8600 to an Arduino, and READ data sent from the radio. It can also send data to the radio ! - job done.

The circuit was assembled on an Arduino 'proto-typing' board (an additional stacked board is called a 'shield'). I also added a 'network shield' which provides an ethernet RJ45 port and networking chipset. That allows even more potential for connectivity and control.



Task 2 : write the software to interact with the CI-V interface

The programming language is more or less C++, I am NOT an expert! although I have been learning and using C and more recently C++ as part of my electronic hobby for about 3 decades.

The language allows the creation of re-usable chunks of code called functions, and better still there are 'code objects' that help to keep the whole project in a manageable state.

Anyone can write code, but I think the art of good code creation is to produce stuff that can be revisited many months or years later, and can be relatively easily understood, and modified without too much grief.

I won't dive too deeply into coding detail of this task, the design principle is that the Arduino Mega2560 will operate in only a number of very well defined configurations.

There are currently three high level modes for the whole device, and perhaps a 4th mode, and several 'states', some or all of which will be used within each mode.

Mode 1 : CI-V Hardware Button Control (physical buttons on a panel)

Mode 2 : CI-V Virtual Button Control (virtual buttons on an ipad)

Mode 3 : CI-V to Ethernet Bridge (CI-V to Ethernet and Ethernet to CI-V)

Mode 4 : CI-V to PROCESSING Display Meter display on MAC or PC

The code structure looks a little like this :

State 1 : Initialising
Run setup :
Setup {
define I/O pins,
set Serial Port Baud Rates,
set MAC and IP address, create internal Objects, set variables to known values,
set Device Mode
...Switch to State 2
then EXIT setup
}

now run Main Programme Loop.

Main loop {
Any buttons pressed ?

USB serial buffer empty ?

ETHERNET buffer empty ?

RX2 serial buffer empty ?

Is there a message to send ?

What is the device State ?
}

State 2 : Listening
If RX flag = true change to State 3

State 3 : Receiving
If ALL bytes read change to State 2.

State 4 : Lookup... not defined yet

State 5 : Sending
If ALL bytes sent change to State 6.

State 6 : Verifying
Are 'received bytes same as sent ?.
MORE?
State 7 : Collisions ! ... not defined yet

Each of the functions called by the Main Loop may set a FLAG to record that something occurred, and the flag(s) may in turn cause the device to branch to a different state.

In device Mode 1 or 2, after receiving the CI-V data, not much happens. It would be possible to perform a lookup or (as it actually does) just spit out the received bytes to the USB port for display on a monitor. The output can also be sent to a 2 or 4 line LCD module.

I'll skip the other (more obvious modes) and consider Mode 4. This evolved quite recently following some FaceBook comments regarding 'external meter displays'.

The device I have assembled can send a specific string of bytes to a radio to request that it responds with for example it's current VCO frequency. I can do this, it works. The question arose : can a radio output other 'meter' values? ...to perhaps provide a real time an S-meter display on a PC screen.

This table extract indicates that it can.

Cmd.	Sub cmd.	Data	Description
15	01	00/01	Read noise or S-meter squelch status (00=Close, 01=Open)
	02	0000 ~ 0255	Read the S-meter level (0000=S0, 0120=S9, 0241=S9+60dB)
	03		Read the meter level in the "AAAABBCC" Format: AAAA= Absolute signal strength (in 0.1 steps) BB= Plus or minus sign (00=+, 01=–) CC= Meter type (00=dBμ, 01=dBμ EMF, 02=dBm)
	04	0000 ~ 0255	Read the center meter level (0000= Left edge, 0128= Center, 0255= Right edge)

So in Mode 4 we might SEND a request like FE FE 96 E0 15 02 FD

... and receive back the current S-meter level as a value between 0 and 255.

The request could be sent perhaps every 500ms.

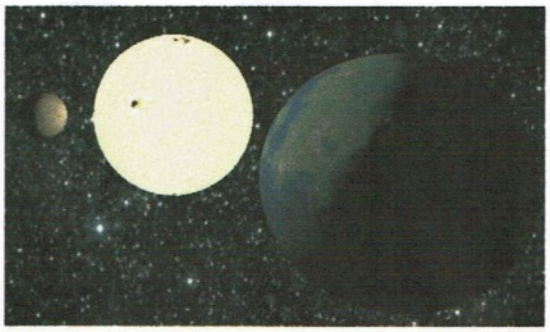
What can be done with this data ?

It can be sent to a PC or Mac via ethernet or USB serial interface.

There is a programming language called PROCESSING. It is very similar to the language used to programme Arduinos. PROCESSING can produce executable files that run on a MAC or a PC.

PROCESSING is very graphically orientated, in that it is designed to enable data to be presented visually.

I have used it to literally produce dynamic graphs depicting Temperature V. Time. Producing a custom S-meter should be quite straight forward.



This stunning image (including the Sun spots!) is in the PROCESSING demo files called 'planets'. By the way it is an animation...it has some 'WOW' factor.

Christmas Holiday Operating

Leta G4RHK and Brian G4CIB on Dumbleton Hill in QSO with Anne 2E1GKY - Boxing Day.

Rig - FT817ND

Looking South from our /P operating position

Archive Photos Supplied by 2E1GKY

Club Picnic 2002

Club School Picnic - Recognise anyone? 20/6/02.

Reliable Summer - Picnic.