THE PROBLEM:

I am the owner of an Icom 756 Pro II and the PW-1 KW. If you have a PW-1 you most like think it is a good amplifier but a lousy client on the CI-V bus, the single wire serial communication bus that runs between ICOM equipment. The problem with the CI-V bus are many making for a very fragile system. The problem is made worse that Icom has so many radios that uses the same flawed system that a complex grouping of equipment is guaranteed to fail.

The bus is used to pass messages between equipment. Messages have a two byte start and a one byte end, but the length of a message is not given in the message, and there is no check sum or CRC in the message that can be used to make sure the message was properly received before acting upon it. Each device on the bus could listen and make sure it's message was properly sent. If they did this, the sender can send a collision message, where the receiver is suppose to disregard the previous message. This protocol is not adhered to in most Icom equipment and few computer programs that connect to the bus. The problem is, when two users of the bus decide to speak at the same time, what appears on the bus is the “wired-or” of the two messages. There is no way for the receiver to know if the message is corrupted. The message that results is at best random garbage, and at worse an instruction for disaster.

For the casual user, this is not a problem. There are few messages on the bus. Make a larger network and thing go south fast. Connect a logging program, such as N1MM or WriteLog to the bus causes a large increase in traffic. There is a mode where you can turn off “transceive” (will describe later) and all that is on the bus is the computer asking questions and getting back the response. So a single transceiver and a computer can work perfectly. The computer is master of the bus, the radio is the slave and speaks only when spoken to.

Now add the Icom PW-1 into the mix and things get messy fast. Icom will tell you nothing about the PW-1 serial protocol. So here is my understanding of how it operates after doing reverse engineering.

In order for the PW-1 to quickly follow the transceiver, the transceiver in placed in “transceive” mode. In Icom protocol there are two messages (message 0 and 1) where the transceiver can broadcast a message on the bus every time the frequency changes, the mode change, or the receiver bandwidth changes. These message date back to the earliest days of CI-V and were designed, I would guess so that separate receivers and transmitters can be tuned together using the CI-V bus. These are broadcast message in that they are always addressed to device zero, one that is prohibited on the bus. When you change bands, the broadcast message is sent out immediately. The PW-1 picks up the message and follows to the new band in an instant. If “transcieve” is turned off, the broadcast messages from the transceiver stop.

Many computer programs will send out messages that collide with the broadcast messages. If it is just the computer and the transceiver on the network, the message will be lost. If the amplifier is on line, it can be unlucky enough to take that as instruction to change bands. Fortunately, most people do not transmit while turning the frequency knob, so the timing of the devastating command is not during transmit.

Okay, why not turn off these broadcast messages, turn of “transceive” mode. If the PW-1 does not receive a broadcast message for approximately ten seconds, it sends a request for frequency, mode and bandwidth to the transceiver. The result is, that it now takes up to ten seconds for the amplifier to follow a transceiver band change. The unfortunate side effect is that now the PW-1 is asserting itself as the bus master. Again, if the computer program is sending a message at the same time, then the transceiver will receive a command that has it doing something.   I have had it turn on split, change bands, change modes, many bad things. The problem is that this can happen any time. Most logging programs turn off sending messages while transmitting, this saves you from destruction.

So the problem is that you just can not have a logging program, a transceiver and the PW-1 all on the CI-V network at the same time and get it to operate properly.

Icom has done nothing to address this problem. The reason, most likely is that there is no easy fix. There are too many radios out there and **most** of the time the customers are happy. It also works okay with simple configuration, so it makes for a good demo and they get the sales. This however does nothing but help Elecraft who actually listen to customers.

THE MARKET

When I first discovered the problem, I figured someone out there must have a solution. I was also looking for a band decoder, but the band decoder would have to be smart enough not to be faked out by collisions. Top Band Decoder exited the business. Checking all the usual suspects at Dayton 2011 yielded many comments about how screwed up CI-V is. No solutions were found. I stopped by Icom and complained and asked for more information about the exact CI-V protocol. The Icom factory rep just gazed at me.

At Dayton I realized that I was not alone. I came across the fine web site by W8UT

http://www.k8ut.com/tiki-index.php?page=Riding+the+CI-V+Bus+-+Details

He had reached the conculsion that I had, but offered no solution.

ROLL YOUR OWN

I decided that there were no solutions on the market place. Having been involved with packet handling for more than a decade professionally, I figured that it would be fairly easy to build an intelligent CI-V message router. I also realized that it would be easy for a CI-V message router to parse frequency messages and map that into an antenna selection for filters or antenna selection.   The last small microprocessor I designed for was the 8052 Intel processor when it was brand new. I started looking for a modern processor that had multiple serial ports.   Ideally, at least three hardware serial ports that would be interrupt operated. That limits field. By chance I stumbled across the Arduino Mega board and the entire Arduino set of boards.

http://arduino.cc/en/Main/ArduinoBoardMega2560

The folks at Arduino also packaged the software development package with a very large and well written set of libraries. The Mega board has four hardware serial ports, 256K of Flash program memory and 8K of RAM.   It clocks at 16 MHz with single instruction per clock, plenty big. They also are readily available boards. They have a built in USB to serial converter, so the board could also act as the computer interface to modern computers. The board comes with Windows USB drivers. It also works for Linux, MAC-OS It can be powered from the USB port or via a wall wart, with automatic switching. This was just what I wanted. You even can buy “prototype shields” that plug into the Arduino processor board where one can build their interface circuitry.   To build the interface to CI-V from TTL is trivial. More on that later.

THE SOFTWARE

The software deals with messages. It is a store and forward router with each piece of ICOM equipment and the computer on it's own serial port. The Arduino can then control the traffic on each of networks. The computer interface can be run full duplex, so collisions there are not an issue.

The ICOM transceiver is run in the “tranceive” mode. So it reports any changes of frequency to the Arduino. The arduino keeps track of the frequency, mode and bandwidth and if anyone asks for that information, it responds. Any message that comes in that the Arduino does not know the answer to, it forwards to proper device. When it sends a message, it does so somewhat intelligently. It looks before it sends the message and makes sure that the port has not been active for 8 milliseconds. It also monitors the message that it sends and if the message was corrupted, it will reschedule the message. Also if a message is received, it goes checks the message length against the command to make sure it is logical and it does a syntax check on the message to see if it makes “sense”. This isn't as good as a CRC but at least it applies a sanity check. For an example, a “frequency” message passes the frequency in a BCD coded form. If any of the frequency bytes are not in a valid BCD format, then the message is ignored. Also to reduce traffic, the Arduino generate “transceive” messages and directs them towards the PW-1. This suppresses the random generation of frequency and mode query messages. This feature is not required, since the Arduino will automatically respond to these query messages without propagating the message to another port, however it keeps the PW-1 from being a bad master on it's port keeping the rogue messages off the port in case another piece of equipment is placed on the same port.

In normal operation the transceiver CI-V port only has the “transceive” messages and the very occasional message when the computer sends a change frequency, mode or VFO command. That command is sent at least through an intelligent interface, and if a collision occurs it is done against a message being sent by the transceiver, in which it already is ignoring, so no harm no foul. On the PW-1 port, only broadcast messages are sent whenever the transceiver changes frequency or mode, and Arduino generated broadcast messages that are sent at an interval to keep the PW-1 from sending any messages, keeping it quiet and out of trouble.

The band decoder is easy, since the exact frequency is always known. When a new frequency if received, it looks up in a table which band segment the frequency is located. For each band segment in the table there is a corresponding code which represents the selected filter or antenna that is desired. a In my case I am driving an Array Solutions Six Pack. With six antennas, I decided to reduce the number of wires between the Arduino CI-V router and the antenna control box, to three active signals. These signal are put into a CMOS decoder, which then drives the relays through larger switching transistors.

Another feature is also a possible confusion factor. Since the Arduino is acting as a store and forward router, the baud rate of each serial port can be set independently. Remember each message is fully received, goes through a sanity check before it is acted upon. The action is to forward it to the proper destination, or generate an automatic response. Now there are even more baud rates to set. Also, the USB serial port that feeds acts as the computer interface an operate at any baud rate up to 115200 bits per second. This means that a logging program can query status really fast. This line is also a full duplex line, and when operating with a normal simple CI-V interface, every message the computer sends out is received back immediately, the message appears on the one wire. If the software actually checks to see if it's message was sent properly, it will think the connection isn't working. Currently I have tested the device with N1MM and N3FPJ and it works fine in both cases. Neither program expects to see it's own message come back. I could program a “loopback”, but since the software structure is based on message store and forward, the easiest thing that could be done would be to receive the message and drop the message back into the message queue, hopefully keeping the program happy. I do not know how all the Icom CAT software operates, only time will tell. Likewise I have noticed with N1MM that when set for ICOM IC756Pro2 at 9600 baud, the program only queries about frequency and mode. I can crank up the USB rate to 56000 bits per second in N1MM and when I do that and automatically respond to those basic queries, the program for some unknown reason starts to query where the volume control is positioned and if the transceiver is in data mode. These commands are not normally seen at the lower bit rates. This was unexpected and not understood behavior. So, the conclusion is that CAT software can operate differently at different bit rates. Consequently raising the bit rate above the slow CI-V rates of 9600 baud may cause problems with some CAT software. As I said, this is a very bad interface that has not been implemented well by anyone. I am sure problems can arise. When an unexpected or unknown command comes into the Arduino router, it will forward it to the proper destination, where the destination will deal with it. Unknown commands are only checked to see if the length is within bounds.

Running the computer at a high rate also provides the potential where the computer can flood a Icom device with messages at a rate far greater than they can be relayed. Rapidly, messages will run out and queues will fill. When this happens, the Arduino does a reset. Fortunately all tested computer software does a query and waits for a response, as compared to blasting out many many queries at one time. The query response model is the only thing that would normally work with the CI-V bus, so there is little chance this will be a problem.

OTHER OBSERVATIONS

The PW-1 does not respond to directed frequency messages. It will only query the transceiver and then move, or once synced, will respond the broadcast messages from the transceiver. To sync, it appears that at power on it queries the transceiver, and if it gets a response within a period of time, it will respond to “transceive” messages. There is a sequence where one teaches the PW-1 the address of the mating transceiver. Once that is done, it will remember it.

Icom gives no information about the PW-1 CI-V control. If one could send it a simple command to the PW-1 to change bands, life would be so much easier. To get the PW-1 to operate under commands, one has to mymic a transceiver. It is possible that a contesting program could do this, and send broadcast messages on a single bus to keep the PW-1 quiet. However, this is at best problematic seeing that the transceiver might also respond to the broadcast message. So again, there is no sure fire way make things right from the logging software.

The Ardunio boards can operate off an external supply, or off power from the USB port. The Ardunio uses linear regulators with small heat sinks. This limits the external supply voltage to 9 volts. If a ham wishes to run it off of 13.6 volts, this will cause the board to smoke. The minimum external voltage to the Arduino is 7 volts. The regulated wall warts that are sold for the Arduino works fine. To run off a voltage greater than 9 volts a regulator or a series zener or resistor is required. The board takes about 120ma of current when running full bore. You have been warned.

INTERFACING TO CI-V

The CI-V bus is a bus that is normally high, 5 volts and is pulled low through an open collector transistor or an open drain FET. The line can be directly applied to the RX input of the Arduino serial port. The TX output must NOT be inverted and must drive an open collector transistor or open drain FET to the bus. Consequently, the output must be inverted and and then drive an open collector or open drain FET. I personally love the 2N7000 FETs for this application. They are cheap and the reduce the number of resistors need as compared to transistors. For a transistor, any fast medium signal NPN would work fine. I recommend the 2N4401 for this application.

HOW DOES THE SOFTWARE WORK

First off, all serial ports are essentially identical in the way they operate. The USB computer port is full duplex, the CI-V are simplex, a single wire. Each serial port operates interrupt driven with a 64 byte input and output buffer. The "main" program polls the each serial port receive and send data. Everything is driven by a serial message.

So 512 bytes of the Arduino memory are used for interrupt serial receive and transmit buffers. A area of the RAM is a memory pool of "Icom Messages". A message is a storage area where the incoming message is assembled, and then passed on. When the message is done, it is returned to the pool for reuse. Serial bytes are received and a state machine looks for the two Icom message start bytes. Upon receiving the start, it then stores the message until an Icom Message end byte is received. If too many bytes are received, the message is discarded. Once a complete message is received, it then goes through the message "sanity checker", where command specific checks are done on the message. If it passes this stage, it goes to the device routing table builder. Each Icom message has a source and and a destination address (byte). The source address is looked up, and if it is not in the table, it is placed in the table along with the address for the serial port. If the message contains either the frequency, mode or receiver bandwidth, this information is supplied to the device routing table. The message destination is looked up device routing table. If the message is a query for frequency, mode or receiver bandwidth, and if the answer is known from the device routing table, an automatic response is generated and returned. If it is another message, the message is forwarded only to the port where the destination is located. If the destination has not been heard, the message is sent out to all ports, hoping for a response. Optionally, if the frequency, mode or bandwidth is known, a broadcast message is sent to all ports other than the device is on at a period if 5 seconds, this is to keep the PW-1 from jumping in and sending messages.

Each serial port has a message queue. Messages are placed in the queue and when the "main" program checks for something the transmit, the message maybe moved to the serial output buffer which is handled by the interrupt handler. Before a message is passed to serial output buffer of a duplex (CI-V) port, it is checks to see if there is, or has had activity for the last 5 ms. When a message is placed in the output serial buffer for a duplex (CI-V) port, a flag is set in the receiver so that it knows that the message on the wire is one that is being sent, and the message is passed to the receiver for checking upon completion. If the message is received and matches, it is deleted.

The timing are based on CI-V baud rates of 9600 baud.