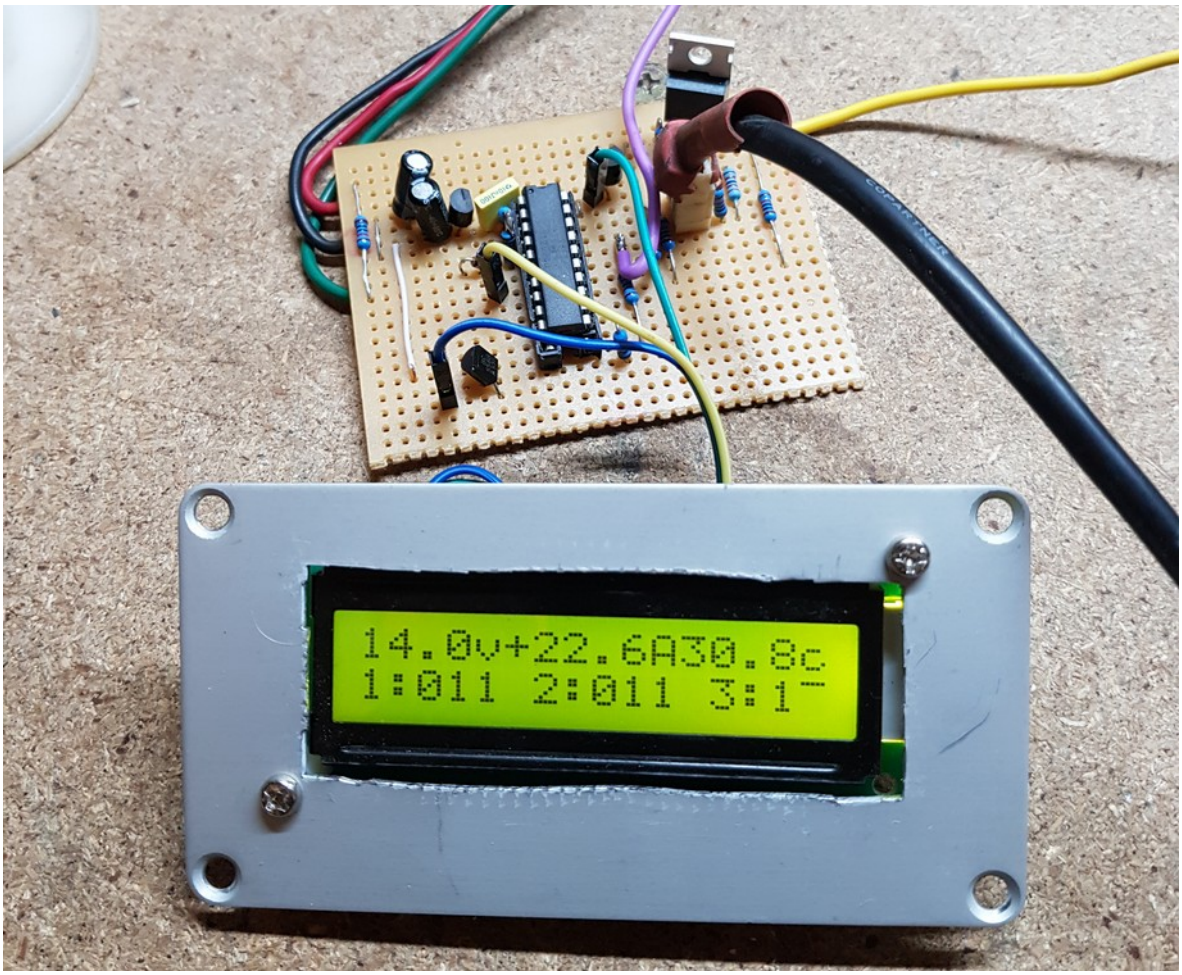
VK3RWO Telemetry unit



Another article by VK3VS/VK3SMB

Table of Contents

The idea behind it

When the idea came about to put VK3RWO on a remote site, I had to think about ways of keeping tabs on it. There may be something commercially available however I would imagine that it would be a something that fits all type approach, and given this is a hobby, most likely cost prohibitive.

Basically I had to design this controller around the following items:

- Low cost,
- Low power,
- Customisable; and
- Communication with the outside world.

The following things had to be monitored/controlled

- Controlling of 3 repeaters,
- Monitoring/control of battery voltage,
- Monitoring/control of temperature,
- Monitoring of current; and
- Security of the site

Electrical Components

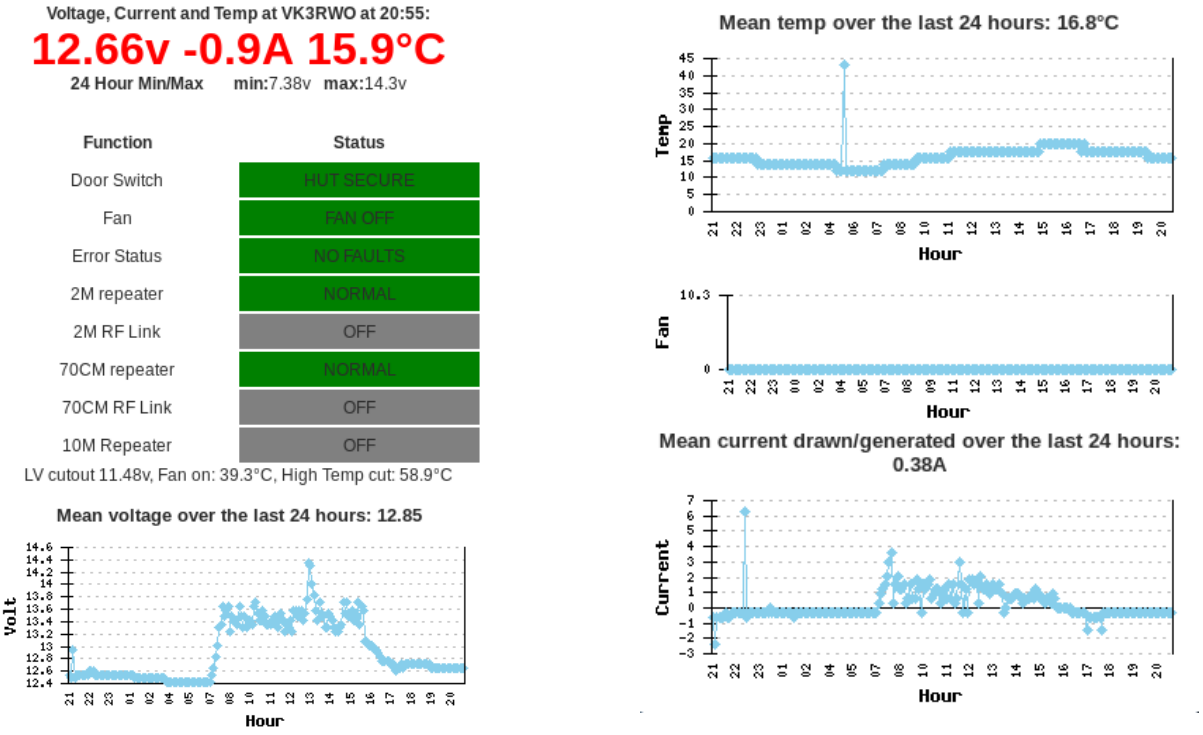
This controller is made up of several pieces. The interconnection of the equipment, and other possibilities of connection will be shown later in this article.

Onsite, there is the controller PCB, and its associated sensors, a data modem and a radio.

Offsite, there is a radio, data modem, TTL – USB converter and a Raspberry Pi with internet access

In a server room, somewhere in the ‘States, there is a server with a LAMP (Linux Apache Mysql and Php) stack installed, which has http://vk3rwo.vklink.com.au pointed at it.

This is all the pieces that give us a nice front end for your viewing pleasure.



Microprocessor

As with a lot of my projects, I use the PICAXE micro controllers. I use these purely and simply for ease of use, and being able to smash out a project with ease. Some will ask

why not the PIC??. It is simple, While I can program html, php, mysql, bash, C and cut my teeth on MS-BASIC, I have not had the time to study the data-sheets and process how to program in assembler. Now some of you will say, “Why not program the PIC in C?”. I probably could, again if I had the time to sit down and study data-sheets and incorporate PWM stacks, UART stacks, and all the other bits that are built into the firmware of the PICAXE. The PICAXE have enough speed and functionality for all of my projects, without getting down onto bit shifting type stuff. All PICAXE chips are programmed in PICAXE BASIC, which is a language very similar to BASIC, with added commands for the extra options available in a chip.

They were designed for students to get a grip on programming and interfacing to the real world. There is massive amounts of official documentation, with 5 volumes now, and huge amounts of info on the internet in the official and unofficial forums.

For this particular project, I have used the PICAXE 20X2. I have chosen this chip for the simple fact it has a 256 byte RS232 buffer with a built in UART that does background receive. What this means for the simple folk, is I can send data (from a data radio or offsite!) to the chip, then it can finish what it is doing and then process what is in the buffer. I don't have to interrupt the running program to deal with the RS232 data.

It has 17 usable pins, mostly bi-directional that are set up into 3 addresses (A, B, C), and lastly, it also has a built in resonator running at 8Mhz that is reasonably accurate. As you can see later, I can also send fire and forget RS232 data out of other pins to control other devices.

Voltage monitor

The voltage monitor is a simple resistor voltage divider. Unless something goes horribly wrong with the batteries and solar charger in the cabinet, the 15V limit I have aimed for should be sufficient.

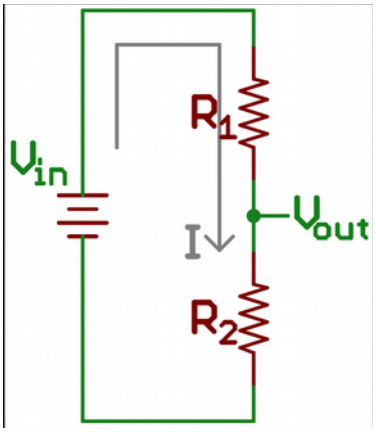


Illustration 1: Simple Voltage divider

The PICAXE has several ADC channels available to use. One has been used for monitoring the voltage. For ease of data transmission, I have only used the 8 bit ADC. Using the 8 bit ADC means there is 256 steps in the 0 to 5V rail, meaning we have a resolution of 0.0195V. As I have used a resistor divider for the voltage, the resolution becomes 0.0585V per ADC step. More than enough resolution for an estimation of how the batteries are holding up.

The ADC values are worked out as follows:

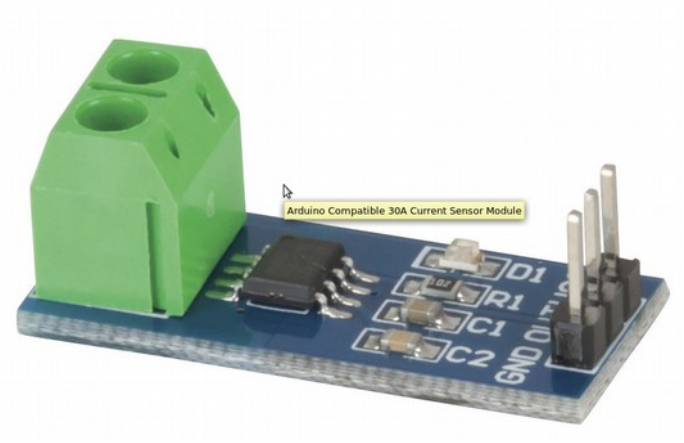
$$Measured\,Volt \div \left(\frac{R1+R2}{R2}\right) \div \left(\frac{PICAXE\ supply\ V}{Total\ ADC\ steps}\right) = ADC\,Value$$

For example, 12.6V (A near full charged battery): $12.6 \div (\frac{33\,kohm}{11\,kohm}) \div (\frac{5}{256}) = 215\,ADC\,steps$

The PICAXE reads this ADC value and stores it for use in the program.

The divider is wired across the batteries to get a voltage at the batteries.

Current monitor



Picture 1: Current module

The current monitor is a bit more advanced. I found while shopping at Jaycar one day that for under $10, they had a little module designed for the arduino system for measuring current. There is bugger all of a data-sheet available for the module alone, but there is one for the IC on it. However, basically I worked out that it runs on 5V, has a range of .5V to 4.5V for +/- 30A and sits at 2.5V for 0A. So working this out for interfacing to the PICAXE went something like this:

- 20% of the ADC steps aren't used, so we have $256 \times 80\% = 205\,steps$
- Step 127 is 0A.
- Which leaves 204 steps to deal with. As we want plus and minus readings there is 102 steps either way.
  - $127 + 102 = 229\,steps = +30\,A$
  - $127 - 102 = 25\,steps = -30\,A$

So to work out the current, the following is used:

$$A \div (\frac{30}{102}) + 127 = ADC\,steps$$

and into practice:

$20 \div (\frac{30}{102}) + 127 = 195$ or a minus value: $-20 \div (\frac{30}{102}) + 127 = 59$

Again the PICAXE reads this and stores it in for use in the program.

The module is wired inline with the batteries. This is done so that we can see how much goes in and out of the batteries. It is pointless installing it on the solar side, as it will only ever show a recharge when the sun is up, and pointless on the radio side as it will only ever show what the radios are drawing. This way we can keep tabs on the batteries themselves.

Temperature monitor

Measuring the temperature with the LM335 doesn't have great resolution, but it has enough for the job. It starts at 0 degrees Kelvin, and works its way up from there. (For those who don't know 0 Kelvin is minus 273 degrees Celsius…).

Picture 2: LM335 Temp sensor

Looking at the data-sheet, this device measures from -273 to +140 degrees C. the formula for working out its temperature with ADC gets a little more confusing.. you convert it to K and then change it to an ADC.

$$(273.15+degC)\div(\frac{5}{256}\times 100)=ADC$$

So, a Cabinet temperature of 14 degrees C would equal:

$$(273.15+14)\div(\frac{5}{256}\times 100)=147$$

Once again, this is stored for the program to use later.

The sensor is mounted behind the radios, in the airflow, in free space. It requires a 2k resistor from the 5V supply rail to the sensor pin and a ground. There is provision for adjustment on the third pin, however as it is only and indication, it is not used.

LCD Display

An LCD display is NOT necessary for this project, as it is remote equipment, nobody stands there and stares at the display. I incorporated it to give me a real time show of the repeater status' and what's going on inside the cabinet. The LCD display only turns on when the door is open to the cabinet.

Picture 3: Google image of a display

The PICAXE is capable of controlling a number of LCD displays. It can use the old bit shift display like the one Jaycar sells, or in this case, I have a number of AFMicro displays laying around from a previous project. These displays accept RS232 data in a 115600 baud 8N1 format. The PICAXE is capable of this speed.

It is wired to the PICAXE via a PNP transistor. When the PICAXE senses the door has been opened, it pulls the base of the PNP to ground via a 10k resistor and turns the display on. It then sends the data via another pin with TTL RS232.

If you look hard enough on the internet, there is actually a pretty comprehensive data-sheet for this device. You can move the cursor anywhere on the screen and display all 255 ASCII visible characters

Cooling fan

The cooling fan is controlled from the controller as well. It is simply an NPN transistor that controls a P Channel FET. The program commands the fan to come on when any of the repeaters are commanded to broadcast the news, or, if the controller senses the current draw of the repeater has exceeded 5A for more than 1 minute continuously, or if the temperature in the cabinet exceeds a preset value. The value has been set to just under 40 degrees, however this can be changed remotely which is discussed further in the article.

The fan is mounted to an external vent on the cabinet. It draws air from inside the cabinet and expels it.

Door switch

The door switch is actually a PTT switch off a PRM80 covert microphone. There is nothing special here. It does however serve a couple of purposes:

1. Security. If the door changes state, the controller within Microseconds, stops what its doing and sends a message via the telemetry that the cabinet has been opened, this in turn sends me an email to say its open… I know the door has been opened within a second of it being opened. So, even the flash would not be able to break open the door and disconnect the batteries before the controller told someone there was something wrong.
2. If the fan is running due to high temps and the door is opened, due to the negative pressure in the cabinet from the fan, it actually makes it hard to open the door, so as soon as it senses door movement, it shuts the fan off to get the door open; and
3. Serves as the switch for the LCD display. There is no point having the display running if the door is shut.

BCD Outputs

My Repeater controllers are unique to the world. When I built them, I did not want DTMF commands used, as so they can be copied by those with nothing better to do. I thought about telemetry control to each controller, but that didn't allow for expansion easily, and was one of those cases of putting ones eggs in one basket.

So I elected to give the repeater controllers BCD inputs, with each input serving a different purpose. I chose 3 inputs, as this gave me 8 different options of repeater operation.

It also means that IF the telemetry controller dies, I can wander up to the site and manually set the options with toggle switches and replicate which option I want to use. The repeater controllers will keep operating without the telemetry unit, they just wont tell us anything.

Understanding BCD

BCD or Binary Coded Decimal, is simply counting in binary. Hence in the repeater configuration, if I used inputs in normal use, I would have 3 functions, however, using BCD, I get 8. Here's how it works:

Input 1	Input 2	Input 3	Decimal Value	Function (examples)
0	0	0	0	Repeater normal, No sleeping, No Link
1	0	0	1	Repeater normal, Sleeping allowed, no link
0	1	0	2	Repeater normal, No sleeping, Link Active
1	1	0	3	Repeater normal, Sleeping allowed, Link Active
0	0	1	4	TOT disabled, No sleeping, no link
1	0	1	5	Shutdown
0	1	1	6	TOT disabled, No Sleeping, Link Active
1	1	1	7	Shutdown

As you can see, function 5 and 7 are the same, as there is no point having a shutdown repeater and an active link….

Communicating with the outside world

It is all fine and wonderful having all this information available on-site. But, I'm not going for a 7km walk every day to look at it. The whole idea of this was to have serial data getting off the site and down to my QTH to enable me to keep tabs on what is going on up there.

As mentioned previously, the PICAXE 20X2 has a built in UART and buffer. This means I can send the data over an RS232 connection. The LPID data radio and modem form the really long RS232 cable we need to get the data off the hill. The controller for VK3RWO is set up as follows:

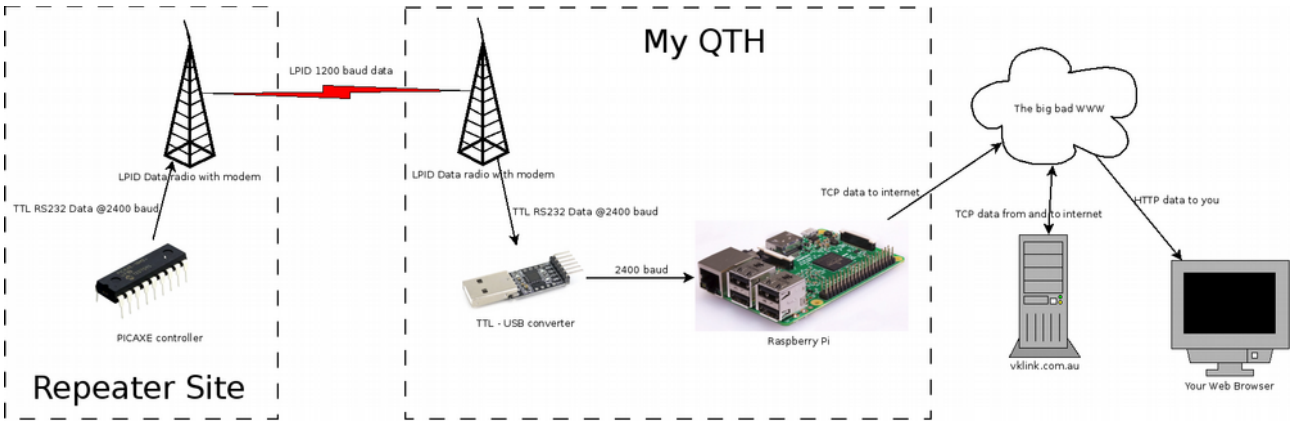


Illustration 2: Layout of the telemetry for VK3RWO

Even though it appears complex, it is relatively simple. Shown later is another configuration that could be used.

It is entirely possible to use a web server on the raspberry pi, and forward port 80 on your router to your raspberry pi. The reasons I have elected to use a server is:

a) Security. I don't have to worry about the wrong port being open on my router

b) automatic backups as part of my plan

c) I already have a server with a LAMP stack for the VKLink project.

Connection options

Whilst I have used LPID data radios as my preferred method of getting the bytes of data off the site, two other methods could be used, with no software change.

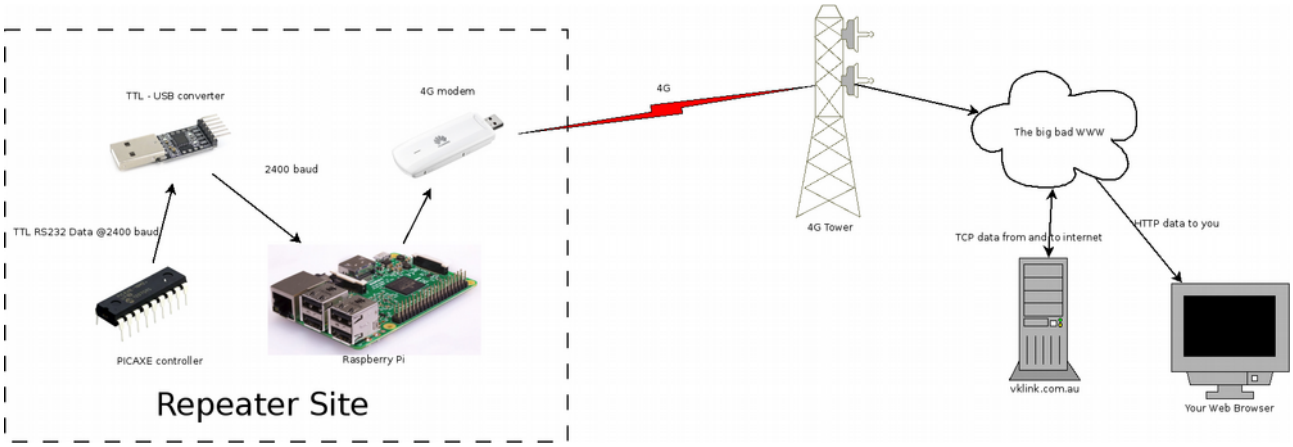


Illustration 3: Connection via 4G

There is a downside to this. Your Pi is not on your network, and if you have to login to make changes, you either have to do a reverse SSH tunnel, OR spend money on a network provider who give a public IP address to your 4G device. As you will see later in the article, you will have trouble controlling the telemetry controller.

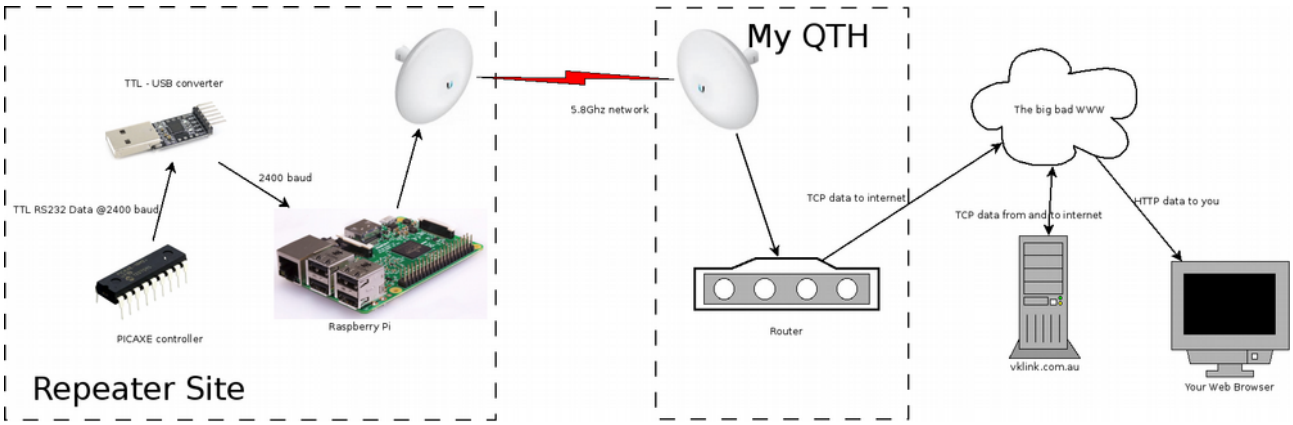


Illustration 4: 5.8Ghz connection. This would be my preferred method

If I had the line of site to the repeater, This would be my preferred method. The raspberry Pi could take over most of the role of the telemetry controller, and just use the temperature, voltage, current and door switch readings from the telemetry controller, and let the pi control everything else.

You can access all ports of the pi from your internal network. If the Pi was to take control of the repeater boards, you could program much more without going up the hill. As so long as you didn't kill the pi and lock it up (any sensible person would have a watchdog installed), you could change anything, how often it reports, etc.

Offsite Electrical Components

As seen from the Illustrations above, there are components onsite, which we have gone through, and there are components offsite. In the setup running on VK3RWO, the offsite components consist of, the data radio and modem, the TTL to USB converter and the raspberry pi.

The Data radio and modem receive the RS232 data and convert it to 2400 baud 8 bit, No parity and 1 stop bit.

This data is then shoved into a CP2102 TTL to USB converter, which is installed as /dev/ttyUSB0 on the raspberry pi.

Repeater Control

As mentioned in BCD Outputs, each repeater controller has 3 BCD inputs on their board. These, at the moment are programmed exactly the same way between all the repeaters.

This however, can change as I see other options etc in the firmware.

Shown already is the RS232 data path for getting data from the repeater site to the outside world. The reverse is also very true. Data can be sent back. Exactly how is shown further down this article.

Brief intro to the VK3VS Repeater controller

I created the VK3VS repeater controller as there wasn't anything out there that did exactly what I wanted. An article will be written on these controllers later, but for now here is a list of items I wanted in my repeaters:

- High impedance audio paths – meaning I could add link radios etc without the levels needing adjustment
- Controlling of the audio path between main and links
- Custom Roger Beeps so that my repeaters can be bought out of the mud during ducting
    - with that the roger beep changes depending on what mode its in.
- Dual mode squelches – meaning CTCSS and COR squelches could be used (for example in VK3RWO 2Mx, it has both available. CTCSS will hear you into the noise floor, COR will only hear you if your signal is stronger than 2uV into the repeater).
- Ability to turn the TOT off for news broadcasts
- A controlling ability that doesn't use DTMF to do so

- Being able to shut the repeater down remotely
- Being able to power down the exciter and PA when its not being used; and
- The big one. The ability for the repeater to go to sleep. What I mean for it to sleep is, it turns off for 2 seconds, powers up for 1 second and listens, then repeats. This drops the current consumption down to a 1/3, vastly improving battery life when it is sitting there idle.

As you can see all that is not in any repeater controller out there in a Ham's budget.

Program

The program that does all the onsite processing is written in PICAXE BASIC. Appendix 1 has a complete listing of the program, along with comments explaining what is going on.

There are four sections here that need explaining a little further.

Getting an average

As I don't want the telemetry controller sending data every 30 seconds, some code has been written into the controller that it grabs a reading every 30 seconds, stores it, then does that another 9 times (5 minutes), averages it out, then sends it down as part of the 24 byte packet. This is how the averages are shown on the graphs at http://vk3rwo.vklink.com.au

Automatic Fan control

The Fan in the cabinet is controlled in 4 ways:

1. Off when the door is open.
2. On when the temperature exceeds the value set in eeprom address 6
3. On whenever a repeater has the TOT disabled (getting ready for a news broadcast); and
4. I can command it via RS232 data to be on (expecting a hot day or similar)

There is no provision to turn the fan off if the controller decides it needs to be on. You can command it to turn off, but the controller will override it next time it reads the temperatures and status's

Changing parameters remotely

As stated, the data goes two ways. the first 8 bytes in the PICAXE eeprom memory store the repeater BCD status's, the fan status, the high temp trigger value and the low voltage value. By sending a 6 byte string to the telemetry controller after the preamble, it adjusts

the eeprom value and the program then deals with it. The string goes like this:

Data	XX	XX	XX	XX	10	13
Description	ID From	ID To	Eeprom address	Value to change to	CR	LF

Each controller has an ID. I have the one at VK3RWO as ID 2, ID 3 is reserved for VK3RWD. Further improvements on the controller will see it store and forward info from another controller and forward it to the internet. For now:

ID from is 1 (The raspberry pi at my QTH)

ID to is 2 (The repeater is ID 2)

The eeprom addresses are as follows:

Address	Description
01	ID of the Telemetry controller
02	BCD value for repeater #1 and #3
03	BCD value for repeater #2
04	Error status, 255 OK, 1 low voltage, 2 high temps
05	Fan control status. 255 for off, 1 for on, 3 for forced on
06	Fan cut in temp. 15 to 255 value
07	Low voltage cutout temp. 15 to 255 value

10 and 13 are ASCII CR and LF. These are used by most programs to signify the end of a line and that “I” need to do something with the string. The PICAXE actually discards them when it receives them.

Fail-safes

The telemetry has the following fail-safes installed:

- **Low voltage cutout.** If the voltage drops below the ADC value set in eeprom location 07, the software turns both repeaters off until such time as the voltage increases.
- **High temp fan.** If the temperature goes above the ADC value set in eeprom location 06, the fan is turned on until the temperature drops below the value.
- **High temp cutout.** The high temp cutout is hard coded to ADC value 170, or 58.9

degrees C in our readable language. If the temp goes above this, the repeaters are all switched off and the fan comes on. I have yet to see this in action as the cabinet of VK3RWO has never exceeded 48 degrees. I did produce this by aiming a pain removal heat gun into the cabinet at the sensor.

- **Door warning.** If the door opens, the controller drops what its doing and sends an alert that the door is open.

As I think of more things, the firmware will improve and change. This is a pretty impressive list for a device built from scrap.

Receiving the data from the telemetry controller

At this point everything right up to the TTL – USB converter has been explained. This converter does nothing to the data, only gives it a USB hole for the data to be delivered to the Raspberry Pi.

Once in to the Raspberry Pi, a small bash script has been written to capture the data coming in from the serial port (/dev/ttyUSB0) and send it to the vklink.com.au server.

The program listing:

```
!/bin/bash
stty raw -F /dev/ttyUSB0 2400 -echo > /dev/null 2>&1
cat /dev/ttyUSB0  | while read serin  ; do
array=$(echo  $serin | xxd -p)
echo $array
curl "http://vk3rwo.vklink.com.au/update.php?values=$array" -k >/dev/null 2>&1
done
```

This program is called at reboot via crontab and runs in the background. A sample crontab entry would look like this:

```
@reboot chmod 777 /dev/ttyUSB0 && /home/pi/remoteser.sh > /dev/null 2>&1
```

The file */home/pi/remoteser.sh* would have to be made executable by running the command:

```
chmod +x /home/pi/remoteser.sh
```

Storing the data

Once the remoteser.sh program has grabbed a chunk of data, it calls the URL update.php on the vklink.com.au server. This program simply grabs the array, splits it up into 1 byte segments and shoves it into a mysql database.

A sample php file is here. The database logins are removed. This is something you will need to work out:

```
<?php
$values=$_GET[values];

$values = strstr($values, '7575');
$values = str_replace("75757575757575", "", $values);
$values = str_split($values, 2);
$now = time();

include 'sqlserver.php';
mysql_connect(localhost,$username,$password);

@mysql_select_db($database) or die("Unable to select database");

$hours24 = $now - 86500;

mysql_query("DELETE FROM `rptstat` WHERE `timestamp` < '$hours24';");

if ($values[12] && $values[14]){

mysql_query("INSERT INTO `rptstat` (`id`, `timestamp`, `idf`, `rptid`,
`rpt13sw`, `rpt2sw`, `fault`, `fan`, `hightemp`, `lowvolt`, `avolt`, `acur`,
`atemp`, `volt`, `cur`, `temp`, `door`) VALUES (NULL, '$now', '$values[0]',
'$values[1]', '$values[2]', '$values[3]', '$values[4]', '$values[5]',
'$values[6]', '$values[7]', '$values[8]', '$values[9]', '$values[10]',
'$values[11]', '$values[12]', '$values[13]', '$values[14]');");

}
mysql_close();


?>
```

Processing the data

There is not a lot of processing the data that goes on. The http://vk3rwo.vklink.com.au web page simply brings the data stored in the database out into a webpage. The fancy CSS encoding of the vklink server makes it look pretty.

Displaying the data for humans

Displaying the data for us to read is done in similar fashions at both the repeater site, and online. The main difference is, the PICAXE cannot deal with decimal points. For example, if we had the equation: 5.7×8 , which we know calculator can work that out to 45.6. PHP can work that out. However that formula would be illegal in the PICAXE BASIC interpreter, so we would re-wite the formula to $57 \times 8 \div 10$ which still gives the answer of 45.6 (even though the PICAXE would interpret that as 46), and is quite acceptable in the program.

Onsite display

The onsite display is a simple 16 x 2 LCD. The program on the PICAXE runs the following formulas over the ADC data, splits the answer down to single ASCII characters and stores them individually. Refer to the appropriate sections above for comparison

Voltage: $Human\,readable = ADC \times 195 \div 10 \times 3 \div 100$

Current: It runs 2 different formulas, depending if the ADC value is greater than 127 or not:
$Human\,readable = ADC - 127 \times 300 \div 100$ or $Human\,readable = 127 - ADC \times 300 \div 100$

Temperature: $Human\,readable = ADC \times 195 \div 10 - 2732$

I did forget to mention that PICAXE mathematics does not follow BODMAS (or PEDMAS depending on where (and when) you went to school). It is strictly left to right.

The display also gives a Binary representation of the repeater status's

Remote display

The remote display, or website, reads everything out of a database, applies the simple math and adds some prettys to it. Without giving the entire PHP code out to the masses, here is bits of the code to get an idea:

```
current_volt = hexdec(mysql_result($query,0,"volt"));
$current_volt = round(0.01953 * $current_volt * 3,2);

$current_temp = hexdec(mysql_result($query,0,"temp"));
$current_temp = round((0.01953 * $current_temp * 100) - 273.15,1);

$current_current = hexdec(mysql_result($query,0,"cur"));
      if ($current_current > 127)
{$current_current = round(($current_current - 127)* 30/100,2);}
      else
{$current_current = 0 - (round((127 - $current_current) * 30/100,2));}
```

The graphs are created with php_plot. The data has to be put in an array, and then the php_plot function is called with the array:

```
<?php

//Include the code
require_once 'phplot/phplot.php';
require_once 'phplot/contrib/prune_labels.php';

//Define the object
$plot = new PHPlot(350,150);

//Define some data
include 'sqlserver.php';

mysql_connect('localhost',$username,$password);
@mysql_select_db($database) or die("Unable to select database2");

$graphing=mysql_query("SELECT * FROM `rptstat` WHERE `rptid` = '02' ORDER BY
`timestamp` DESC LIMIT 288;");
$num=mysql_numrows($graphing);
mysql_close();

$i= ($num - 1);
while ($i>0){
$voltage=mysql_result($graphing,$i,"avolt");
```

```
      $volt = hexdec($voltage);
      $volt = round(0.01953 * $volt * 3,2);
      $time=mysql_result($graphing,$i,"timestamp");
      $readable = date('H:i',$time);
if($volt > 11 && $volt < 15){
$volt_data[]=array($time,$volt);}
$i--;
}

prune_labels($volt_data, 20);

$plot->SetDataValues($volt_data);

//Turn off X axis ticks and labels because they get in the way:
$plot->SetXTickLabelPos('none');
$plot->SetXTickPos('none');
//$plot->SetXDataLabelPos('none');
$plot->SetXLabelAngle(90);
$plot->SetXLabelType('time', '%H');
//Set titles
$plot->SetYTitle('Volt');
$plot->SetXTitle('Hour');
//Draw it
$graph=$plot->DrawGraph();
?>
```

Summary

And that is how the Telemetry controller is put together. Its a big bit of mis-match of everything to get the job done. I had all these bits laying around. Whilst there is several different pieces of technology and coding being used, the standard RS232 format of data between them makes it all work very effectively.

Future plans include putting this controller on a second site, with a second ID, and the hope is that the second site will talk to the first and then the first will tell my QTH all about it.

‘73, Matt, VK3VS/VK3SMB.

Appendix 1 – PICAXE code listing

```
#picaxe 20x2
setfreq m16
'Control board for multiple repeater sites

'Set inputs and outputs
let dirsb = %10001110
let dirsc = %00111111

let pinsb = 0
let pinsc = 0

'Serial background recieve setup
hsersetup B2400_16,%1


'set adc up
let adcsetup = %10001000010
symbol temp = 1
symbol volt = 6
symbol curr = 10

symbol door = pinc.6
symbol fan = C.5
symbol lcd = B.7
symbol lcddata = C.4

symbol usebit = b0
symbol usebit2 = b1
symbol usebit3 = b2
symbol usebit4 = b3
symbol useword = w2
symbol errorflag = b6
symbol ownnumber = b7
symbol counter = b54
symbol smallcount = b55
symbol timer_1 = w26
symbol voltones = b51
symbol voltdec = b50
symbol currones = b49
symbol currdec = b48
symbol currdir = b45
symbol tempones = b47
symbol tempdec = b46
symbol norepeat = b44


eeprom 0, (3,2,4,4,255,255,160,196)
eeprom 14, (255,10,13,4,4)

start:
read 1, ownnumber
gosub zeroall
gosub readins
gosub zeroall
gosub setouts
pause 129
let timer_1 = timer_1 + 1
let counter = timer_1 / 300              'will be 600
```

```
let smallcount = timer_1 // 300                 'same
if smallcount = 0 then
      gosub readadcs
end if
if counter = 10 then
      gosub writeflags
end if

if hserinflag = 1 then
      gosub zeroall
      gosub rxdata
end if

goto start
end

rxdata:
'hsersetup B2400_8, 0
let hserptr = 0
for ptr = 0 to 50
      let usebit = @ptr
      if usebit <> "u" then exit
next ptr
if @ptr > 9 then goto finishrx
if @ptr = ownnumber then goto finishrx
if @ptr > 3 then
      put ptr, 3
end if
ptr = ptr + 1
if @ptr = ownnumber then goto takevalues
pause 500
for ptr = 0 to 30
      let usebit = @ptr
      hserout 0, (usebit)
      if usebit = 13 then goto finishrx
next ptr
pause 100
goto finishrx

takevalues:
ptr = ptr + 1
usebit = @ptrinc
usebit2 = @ptrinc
if usebit > 1 and usebit < 8 then
      write usebit, usebit2
      if usebit < 4 then
            usebit = usebit + 15
            write usebit, usebit2
      end if
      let ptr = 0
      let hserptr = 0
      let hserinflag = 0
end if
gosub zeroall
gosub sendinfo

finishrx:
let ptr = 0
let hserptr = 0
let hserinflag = 0
'hsersetup B2400_8,%1
```

```
return

writeflags:
      useword = 0
      for usebit = 200 to 209
            read usebit,usebit2
            let useword = useword + usebit2
      next usebit
      let usebit2 = useword / 10
      write 8,usebit2

      useword = 0
      for usebit = 210 to 219
            read usebit,usebit2
            let useword = useword + usebit2
      next usebit
      let usebit2 = useword / 10
      write 9,usebit2

      useword = 0
      for usebit = 220 to 229
            read usebit,usebit2
            let useword = useword + usebit2
      next usebit
      let usebit2 = useword / 10
      write 10,usebit2

      gosub sendinfo

      'exit'do a serial out here
      let timer_1 = 0

return

validateusebit:
if usebit < 14 then
      usebit = 14
end if
return

sendinfo:
      gosub zeroall
      pause 100
      hserout 0, ("uuuuuuu")
      for usebit = 0 to 128
            read usebit,usebit2
            hserout 0 ,(usebit2)
            if usebit2 = 13 then exit
      next usebit
pause 100
return

readins:
'     read voltage
readadc volt,usebit
gosub validateusebit
write 11,usebit
useword = usebit  * 195 / 10 * 3 / 100
usebit2 = useword / 10
voltones =  usebit2
usebit2 = useword // 10
```

```
voltdec =  usebit2
read 7, usebit2
'     if voltage is too low....
if usebit2 > usebit then
      write 2,2,2,1
      errorflag = 1
else
      read 17, usebit3
      read 2, usebit4
      if usebit3 <> usebit4 AND errorflag < 1 then
            write 2, usebit3
      end if
      read 18, usebit3
      read 3, usebit4
      if usebit3 <> usebit4  AND errorflag < 1 then
            write 3, usebit3
      end if
      write 4, 255
      errorflag = 0
end if

'     read the current
readadc curr,usebit
gosub validateusebit
write 12,usebit
if usebit >127 then
      useword = usebit - 127 * 300 / 100
      usebit2 = useword / 10
      currones = usebit2
      usebit2 = useword // 10
      currdec = usebit2
      currdir =  1
else
      useword = 127 - usebit * 300 / 100
      usebit2 = useword / 10
      currones = usebit2
      usebit2 = useword // 10
      currdec = usebit2
      currdir =  0
end if

'     read the temp
readadc temp,usebit
gosub validateusebit
write 13,usebit
if usebit > 139 then
      useword = usebit * 195 / 10 - 2732
      usebit2 = useword / 10
      tempones = usebit2
      usebit2 = useword // 10
      tempdec = usebit2
end if
read 6, usebit2
usebit3 = usebit2 -1

'deal with high temps.
select case usebit
case 0 to usebit3       'no fan
      read 5, usebit
      if usebit <> 3 and smallcount = 0 then
            write 5,255
```

```
      end if
      read 2, usebit, usebit2
      if usebit = 14 or usebit = 134 or usebit = 142 or usebit = 6 or usebit2 =
14 or usebit2 = 6 then
            write 5,1
      end if
      read 17, usebit3
      read 2, usebit4
      if usebit3 <> usebit4 AND errorflag < 1  then
            write 2, usebit3
      end if
      read 18, usebit3
      read 3, usebit4
      if usebit3 <> usebit4 AND errorflag < 1  then
            write 3, usebit3
      write 4, 255
      end if
case usebit2 to 169     'fan on
      write 5,1
      read 17, usebit3
      read 2, usebit4
      if usebit3 <> usebit4 AND errorflag < 1  then
            write 2, usebit3
      end if
      read 18, usebit3
      read 3, usebit4
      if usebit3 <> usebit4 AND errorflag < 1   then
            write 3, usebit3
            write 4, 255
      end if
case 170 to 255         'fan on and radios off
      let errorflag = 2
      write 2,2,2,2,1
end select

'     Read door switch
read 14,usebit
if usebit = 255 and door = 1 then
      write 14, 1
      gosub sendinfo
end if
if usebit = 1 and door = 0 then
      write 14,255
      gosub sendinfo
end if

return

zeroall:
let w0 = 0
let w1 = 0
let w2 = 0
return

setouts:
read 2,usebit,usebit2
let pinsb = usebit AND 14
let pinsc = usebit2 AND 14
if usebit > 14 then
      let pinsa = 1
else
```

```
      let pinsa = 0
end if
read 5, usebit
if usebit < 255 then
      high fan
else
      low fan
end if
read 14,usebit
if usebit = 255 then
      low lcd
      serout lcddata, T19200_16,(17,22)
      gosub lcddisplay
else
      high lcd
end if

return


readadcs:
      readadc volt,usebit
      let usebit2 = counter + 199
      gosub validateusebit
      write usebit2,usebit

      readadc curr,usebit
      let usebit2 = counter + 209
      gosub validateusebit
      write usebit2,usebit

      readadc temp,usebit
      let usebit2 = counter + 219
      gosub validateusebit
      write usebit2,usebit

return

lcddisplay:
serout lcddata, T19200_16, (128,#b51,".",#b50,"v")
if b45 = 1 then
      b45 = 43
else
      b45 = 45
end if
serout lcddata, T19200_16, (133,b45,#b49,".",#b48,"A ")
serout lcddata, T19200_16,  (139,#b47,".",#b46,"c")
read 2,usebit,usebit2
if usebit > 15 then
      usebit3 = 128
      usebit = usebit - 128
end if
serout lcddata, T19200_16, (148,"               ")
serout lcddata, T19200_16, (148,"1:")
do
      usebit = usebit / 2
      usebit4 = usebit // 2
      serout lcddata, T19200_16, (#usebit4)
loop until usebit = 1
serout lcddata, T19200_16, (154,"2:")
do
```

```
      usebit2 = usebit2 / 2
      usebit4 = usebit2 // 2
      serout lcddata, T19200_16, (#usebit4)
loop until usebit2 = 1
if usebit3 = 128 then
      usebit3 = 1
else
      usebit3 = 0
end if
serout lcddata, T19200_16, (160,"3:",#usebit3)

return
```

Appendix 2 – Schematic

Good enough representation of what I did.