ThermIIC Series

Dr. Simon J. Melhuish

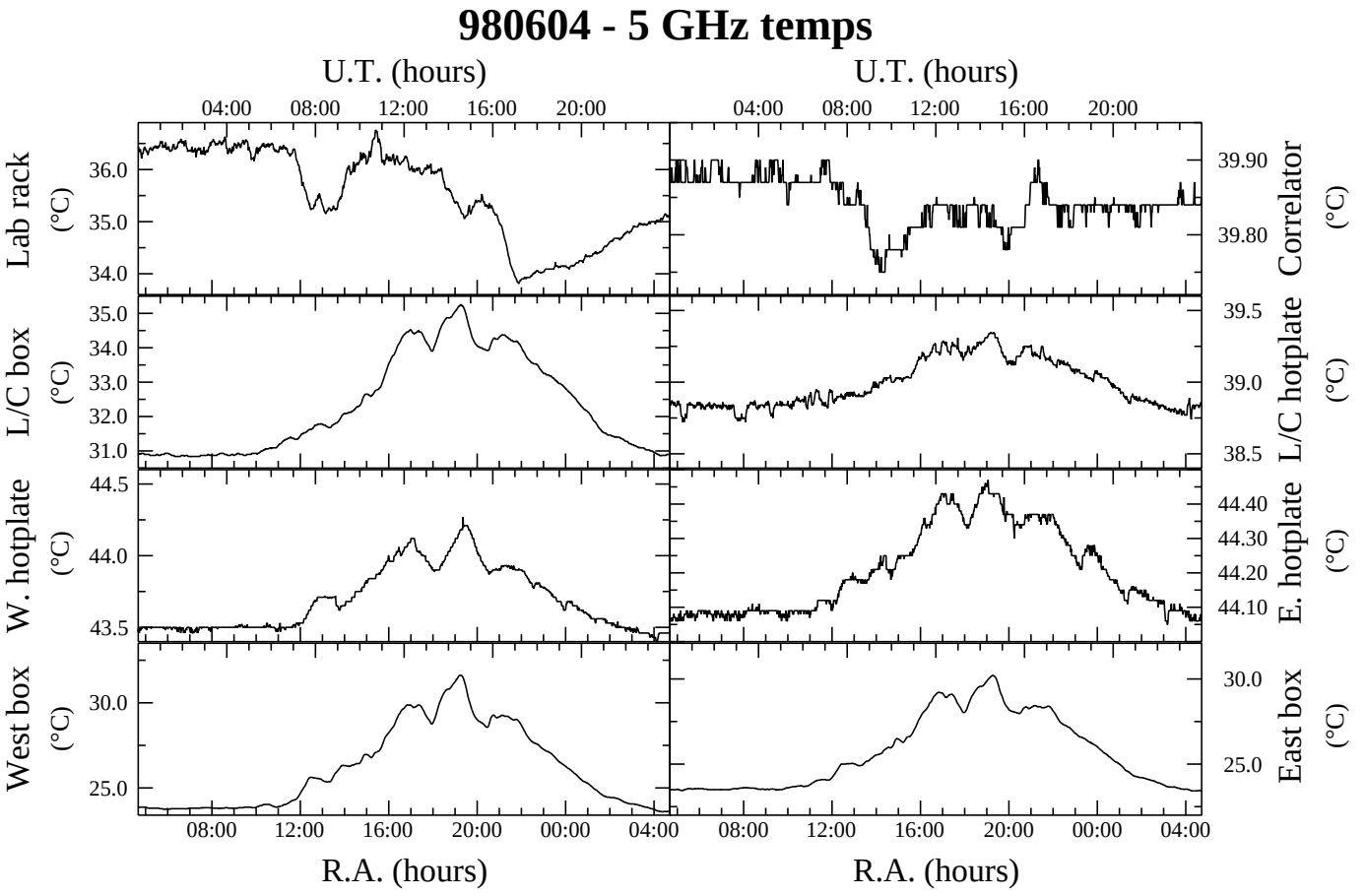
Part 3

In the first two parts of this series I showed you how to connect DS1624 high resolution temperature sensors to your computer, using my *ThermIIC* program to record the results. This time I'll take a look at a couple of applications.

For most uses of *ThermIIC* you'll want to save the data to log files on disc. To recap briefly, two logging options, *Daily file* or *One file*, are available from the setup window. Having started logging, clicking *Save setup* will ensure that logging is started every time *ThermIIC* is run. You could arrange for *ThermIIC* to be run from your computer's boot sequence, for instance by placing the program inside the !Boot.Choices.Boot.Tasks directory if you have a RiscPC-style !Boot application. That way you can be sure that always all your temperatures are being logged, even if the computer gets reset.

The log files, in *comma separated value* format, may be imported into other programs, such as spread sheets. For example, I have used *Eureka*. The date / time values ***should*** be recognized when the program loads the CSV file. However, my copy of *Eureka* seems to have trouble doing this. I find it necessary to make a second column for time values, with each cell using the *value()* function to read the time from the first column. The *value()* function gets it right even though the CSV import doesn't.

One of my current applications is to monitor equipment temperatures on our 5 GHz *cosmic microwave background* interferometer at Jodrell Bank. In pursuit of improved astronomical data I've been trying to reduce the day / night temperature variations in the electronics. With a high-sensitivity radio receiver temperature changes can been a problem. The plot shows how well (or not) I'm doing at the moment.



Results from eight sensors monitoring telescope equipment temperatures

In a school setting temperature logging could find its way into many parts of the curriculum. For example, at *key stage 2* maths “Handling data” or “Collecting, representing and interpreting data” are obvious candidates. Likewise for “Experimental and investigative science”, “Obtaining evidence”. In *key stage 2* thematic studies “Weather” you'd not need much time to think of some applications. This strand leads-in to *key stage 3* “Weather and climate”.

Used indoors the strip-board set up will do just fine, for example to record air temperature in a classroom (see it shoot up as a room-full of children come in!). However, if your environment is not so kind, for example if you want to measure outside temperature in a Stevenson Screen, corrosion would probably become a problem. At the very least you should spray the board with a protective lacquer, such as the so-called “conformal PCB spray”.

It's quite handy to be able to measure the temperature of wet things. For example, a beta tester has been monitoring the fluid temperatures of a hydroponic garden. Obviously electricity and water don't generally go together too well. Dunking your computer's +5 V and ground lines into a tank of water, or a pond, for example, is not something I'd normally recommend! But if you're very careful you ***can*** make the sensors waterproof. The trick is a substance known as *potting compound*. These compounds are usually epoxy, polyurethane or silicone-based, supplied as a resin and a hardener. I used a particular sort with low thermal resistance. It's intended for making sealed power supply units, so that the heat gets conducted out quickly, but heat flows just as well in the opposite direction. The two (liquid) parts are mixed together for use. Fill a mould with the potting compound, with your electronics sat in the middle. After a time *curing* the potting compound will set rock hard. Usually it's simplest to leave the mould in situ, rather than to try to remove it for re-use. There are very small plastic boxes made just for this purpose, called “potting boxes”.

To make the electronics small enough I used PCBs similar to the pattern I provided with part 1. However, the wiring-on-a-socket technique would do just as well so long as it won't come apart before the compound cures. Unfortunately, because of the extra *thermal mass* potting the sensors stops them responding so quickly to temperature changes. If you want a quick(ish) response keep the size down. I found that even the smallest potting boxes were bigger than I wanted, so I cut off approximately one third from the end of one. I used a small rectangle of aluminium to blank-off the cut end, with the DS1624 glued (I used special glue with a low thermal resistance) to the back of it. Making the aluminium over-sized with a hole near the end gives a handy means of mounting the sensor. You can see the finished item in the picture.

A “potted” sensor

The DS1624 sensors have a high ***resolution***, but as a test of sensor ***accuracy*** eight sensors were immersed in water that was progressively cooled. The recorded temperatures are shown in the chart. Ideally, so long as

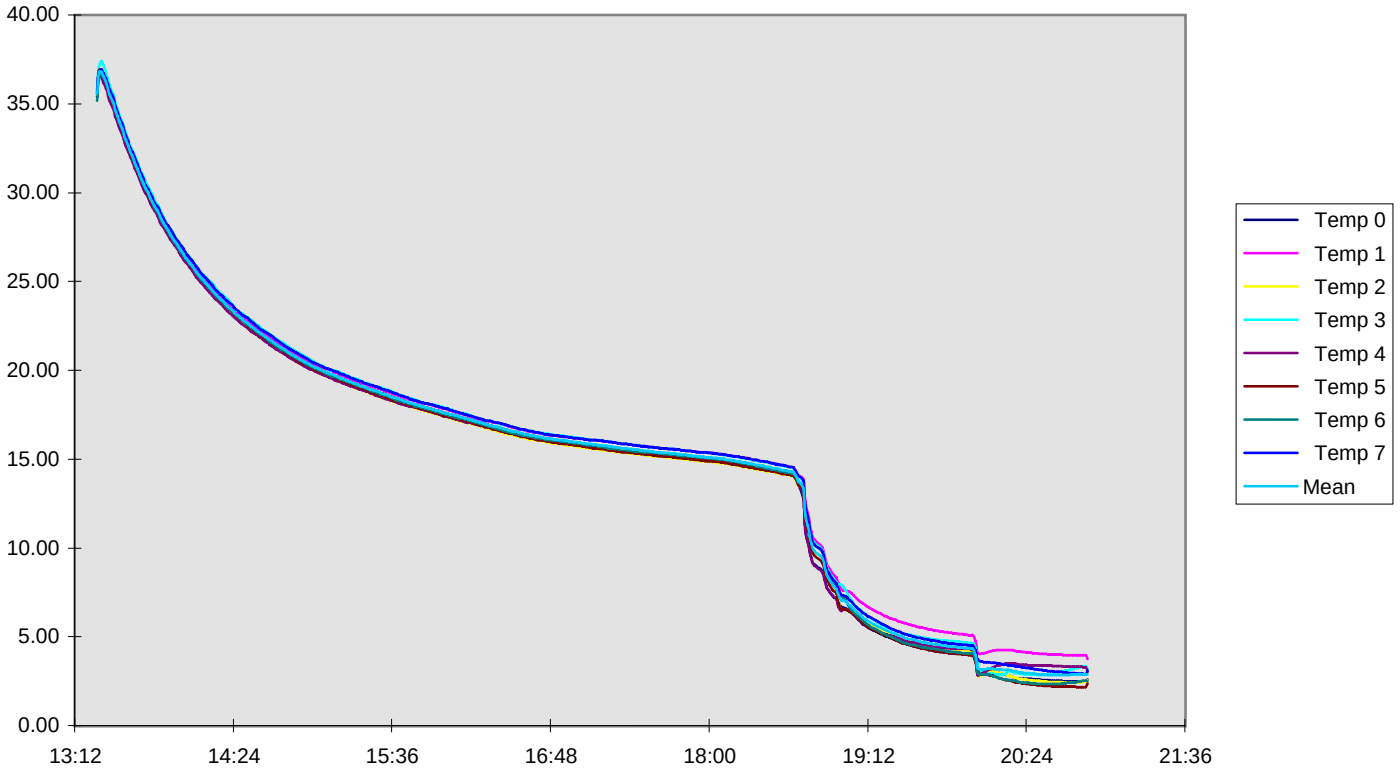
the water is all at an even temperature, the sensors should all read the same. The fact that they don't suggests that there may be a small systematic error (as opposed to a random error) on each one. This might or might not bother you. For myself, I'm only looking at temperature stability with time. I don't need to know absolute temperatures accurately. But in case it's important to you I have provided a temperature compensation facility in *ThermIIC*.

If you type `1624writeC` at the command line you will be able to set compensation values for each sensor. The syntax to use is:

```
1624writeC No. x y x
```

If the temperature recorded by the sensor is *T*, the displayed value will be $T + x + y * T / z$. *x* is in units of 0.01 C°. Set *z* to 0 to disable compensation on a sensor. `1624writeC` uses the DS1624 non-volatile RAM to store the compensation values, and a checksum to ensure their validity. See *!ThermIIC.history* for some examples. Repeatedly Alt-clicking the *ThermIIC* iconbar icon will show each of the compensation values in turn.

The question of measurement accuracy is a big subject in itself for this and for any other measurement device. Much fun could be had playing with insulated beakers of water, &c, repeating this test on your own sensors. But if you want results that are truly accurate you will need to find a truly accurate reference. Bear in mind that most domestic thermometers will be no better (and probably worse) than this. At the very least you will need a Met-grade thermometer to achieve 0.1° accuracy. Always remember that just because something gives you a digital read-out it doesn't mean it's right!



A plot of an example log file

I will conclude this part with a treat for the net-heads amongst you. I think it's fairly cool to be able to read live data over the internet. There are three programs on the cover disc to facilitate this with the DS1624 sensors. Firstly, the *Finger* directory contains a BASIC program, *1624*. This is for use with the *FingerD* finger server, by Andrew Freeman (`http://freenet.barnet.ac.uk/freenet/a.freeman/`). Open your copy of *!FingerD* and copy the BASIC file as *!FingerD.Librarys.1624*

Now, having run *FingerD*, a finger request to your host for *1624* (i.e `finger 1624@your_host_name`) will respond with the readings from all connected sensors.

The two BASIC programs in the *Web* directory are for use with a web server (for example *AlphaNet* or *Netplex*). You should place them in a directory accessed by the web server, typically in *cgi-bin*. *1624CGI* dynamically creates a web page listing the temperatures of all sensors. For those browsers that support it, a "Refresh" meta tag is given, so that the display will update every 30 seconds. Using the URL `http://your_host/cgi-bin/1624CGI` would then run the program on your server, providing the page of temperatures for the remote host. *1624table* is a variation on this, laying-out the data in a table. The eight lines of `DATA` statements at the start of the program give the sensor name and min / max temperatures. The table cells showing the temperatures are normally green, but turn blue / red if the given limits are exceeded. So long as my A4000's power supply hasn't died again by the time you read this, you should be able to see this program in action if you follow the links from my home page, `http://www.jb.man.ac.uk/~sjm/`

In the next issue I will return to a "soldering iron" theme, to show you how to use I²C over a long cable. This could be of use for other I²C projects, as well as this one.