## Contents

# Ten Year Temperature Logger
by Dr Richard Whitaker

When I first saw the adverts for the DS1516 chip from Dallas, I was intrigued by the way it could capture temperatures and store them away, and all in one chip. The really amazing thing is that you only need seven components to make a working system.

## What's the Temperature in the Greenhouse?

I have always wanted to know how the temperature in my greenhouse varied during the day and on into the night. A max min thermometer gives some idea of the extremes but can be fooled by the sun shining directly on the bulb or sensor, giving a false maximum temperature. A data logger giving a graph of temperature would tell the real story and this is the chip to make that possible.

Designs have to start somewhere so the first thing was to get the datasheet, which fortunately was found on the www.dallas.com Web Site. Providing data for new devices on the web is a great way to let people get started. Unfortunately the datasheet is only the start of the detective process which allows you to use a programmable chip such as this. I wish data sheets had simple but real examples of how to use the device. I bet many clever chips aren't used because there are no good examples of how to use them.

The circuit is easily built on a piece of Veroboard, though a PCB gives a smaller size and better controls the noise sources which can disturb the stability of the clock. I also wired up the switch and the red and green LEDs though these are not essential to run the device.

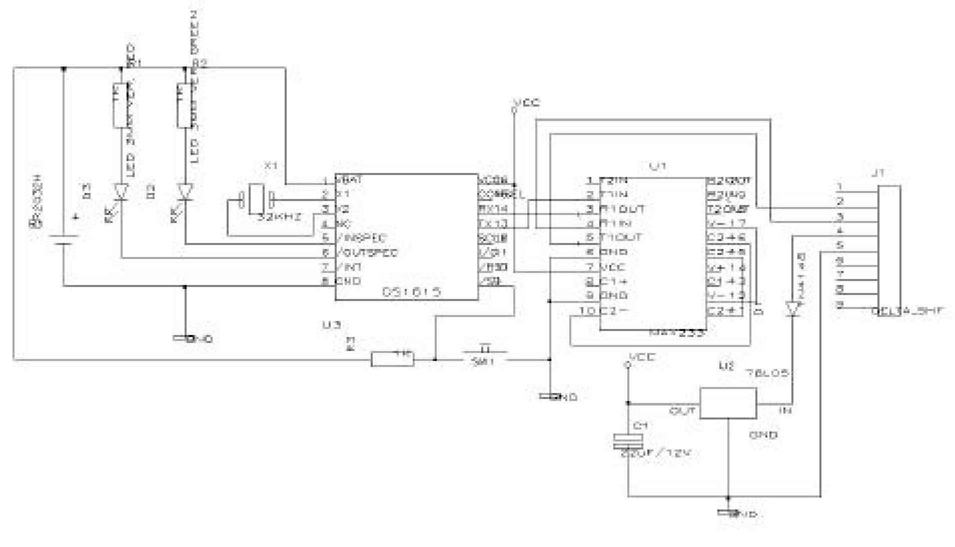


Figure 1 – Circuit Diagram

## The Circuit

The circuit in Figure 1 is made up of the DS1615 logger chip and a MAX233 RS232 interface. The MAX233 doesn't need any capacitors to generate the ±10V for the transmitted data. It would be cheaper to use a MAX232, but it would need four capacitors, which increases the component count.

The 5V supply for the circuit is collected from the RS232 interface. From the program, the DTR line is set high, which gives about +8V to +12V on pin 4 of the connector. The series diode D1 protects against having -12V on the pin, which is its low state. Finally the low drop out 5V regulator U2 provides power to the 5V side of the DS1615 and the MAX233. These together require about 2.7mA as measured in my circuit. Most RS232 sources will provide a short circuit limited current of 10 to 20mA, so we are well within range of the power which can be taken from the RS232 connection.

When the logger is disconnected from the PC and its RS232 interface, the DS1615 runs on the 3V lithium battery. Current consumption depends on how often the temperature is read, but calculations suggest that the lithium battery, type CR2032, will last for about 10years at all sample rates. This doesn't take into account using current for the LEDs. I first used ordinary LEDs since they were to hand, but they are not bright at such low currents. It would be better to use special low current LEDs, which have a maximum current of 7mA. The current through the LEDs is given by the equation:

$I_{LED} = (V_{BAT} - V_{LED})/R1$ which is (3-1.4)/1000

=1.6mA

A low current saves battery power. For a brighter LED decrease the value of R1 to say 100 ohms for a current of about 16mA.

The switch SW1 and R3 are used to start a logging mission. The switch is debounced in the chip by needing to be held closed for at least half a second.

Neither the LEDs nor the switch is needed to run the circuit. All control can be done through the serial interface, though for testing it is good to see the lights flash.

The crystal is a 32KHz watch crystal. I used one from an old watch and also tried another from Maplin Electronics. Both seemed to work OK, but for accurate time keeping you should look at the Dallas data sheet for the best crystals to use.

There are other pins on U3, which can be used if the chip is directly connected to a microprocessor. Allowing pin 15, COMSEL, to go high selects the serial ASCII pins 13 and 14 instead of the microcomputer clocked serial interface. The serial ASCII is set to run at 9600 baud with 8 data bits. There is no choice of baud rate.

Pin 7, /INT, can be enabled to generate a signal when the measured temperature exceeds the alarm values set in the chip. It can be used to generate an interrupt to a connected microprocessor. In this design I do not use it, at least not at the moment.

Be careful soldering the components. The crystal is sensitive to heat, so solder it quickly to prevent it getting too hot. The rest should be straightforward. Lithium cells are capable of sourcing high currents, which can be a hazard, so take great care not to short out the battery connections. The cell also has a potentially long life of 10 years. Any grease and moisture from handling the cell can add to the discharge rate and run it down before its time, so try not to bridge the positive and negative contacts with your fingers.

That's about all for the hardware of the logger, but it is no use without some software to set registers and read back data, so lets move on to making it active.

## The Detective Work

My first thought was that if all was working I could press the button and at least one of the LEDs would flash.

It was not so easy. When this did not work I looked around thinking about what may not be right. Was I providing power? The chip has two power supplies. One is the battery, which runs the circuit when it is recording data away from the PC, and the second is the 5volts used to run the RS232 data link. The datasheet is a bit hazy on the battery voltage. I wasn't using a lithium battery for these tests. My experiments were with three rechargeable nickel cadmium cells giving about 3.6V, and the 5V came from a stabilised power supply. What if I joined both power supplies together? Again it did not work. How about sending a command through the RS232 link to flash the LEDs? Still no success. After checking all the wiring and trying it all again I began to wonder if the chips I had were faulty! If in doubt blame the manufacturer. But I went back to the Web site and found an application note had appeared. There was more information about the battery and the 5V supply. For the RS232 interface to become active the 5V had to be applied to one pin and the battery had to be no more than 90% of the 5V. In addition the battery voltage must be at least 2.7V.

Armed with this new detail I used a 3V lithium cell for the battery, and a 5V stabilised supply for the VCC pin. The LEDs still wouldn't flash, so I had to think again. I guessed that the crystal could be a problem, because I'd used one salvaged from an old watch which no longer had a working display. The datasheet says you should use a crystal designed for a 6pF load capacitance. I had no idea what my crystal needed. Again the datasheet says the crystal pins should have a 'guard ring' to prevent interference to the oscillator! My Veroboard didn't have any 'guard rings'; another source of doubt. I tried to use my oscilloscope to detect the oscillation, but it looked as though there was no sign of life. Perhaps with the whole circuit using so little power, by probing I was damping the circuit to the extent that it stopped. Another inconclusive test! I later found that you can probe pin 3 of the crystal circuit and see it oscillating if it is turned on.

## Attacking the Serial Interface

It was back to the datasheet, and a fresh look at the RS232 commands that can read the data in the chip. Could I get it going and perhaps read some of the status bits? I am always nervous of commands, which consist of more than one byte. There are problems, of 'which bit is which'? Is D0 the least significant bit or the most significant bit? What are d0 to d7, a0 to a5, and a8 to a15? The more bytes the more chance of an error in understanding. Lets press on, though, the chip is split into 32byte blocks of memory called pages, so it should be possible to read page 0, without knowing the order of the bits. I use Power Basic, an MSDOS based Basic programming language, much like QBasic, running within Windows 95, to send the commands and read back the reply.  MSDOS is still the easiest way to read and write to the serial port. The Dallas data sheet says that the bytes of a command must not be separated by more than 10 bit times, otherwise the system ignores everything. It seems that provided the bytes are sent in the same print statement, this is not a problem.

The following is an excerpt of code, which I would have liked to see in the data sheet:

```
1 Open "COM2:9600,N,8,1,CS0,DS0,CD0,ME,FE" as 1    'the RS232 works at 9600 baud
2 delay 0.5                                         'allow +5V to settle
3 print #1,chr$(&H33);chr$(&H00);chr$(&H00);        'read data page 0
4 aa$=input$(32,#1)                                 'get 32 reply bytes
5 for i=1 to 32
6  print asc(mid$(aa$,i,1))                         'print the bytes 1 by 1
7 next i
8 ba$=input$(2,#1)                                  'read CRC which is 2 bytes
```

· Line 1 opens the serial port Com2 at 9600 baud and ignores all pins except 2 and 3, Receive and Transmit data.
· Line 2 is a delay of at least 0.5 seconds to allow the +5V to settle
· Line 3 prints the three-byte command to Com2 and onto the DS1615.
· Line 4 waits for 32 bytes to be read back.
· Lines 5 to 7 print the values on the screen.
· Line 8 reads the two byte Cyclic Redundancy Check that can be used to make sure the data was correctly received.

It worked! At last I could see the chip was alive. Prospecting through the data sheet reveals that the oscillator does not start if the EOSC bit is set to 1, which is the default value. Or at least that is true when the battery is the power source. When the 5V VCC is applied, the EOSC bit is ignored and the oscillator runs anyway. I guess the default being that the oscillator is off, is so that when you deliver a logger, it can have the battery in it but with almost no current drain. There are also a number of other bits to set up before the circuit is ready to start logging, though. This is the sequence to use:

1. Set the CLR bit to 1 to enable a Clear Memory

2. Set SE to 1 if you want to use the press button to start logging, or set SE to 0 if you want to use a program command to start logging
3. Send the Clear Memory Command
4. Send a Sample Rate command to set the number of minutes between samples and to start logging if a program set start is set up.

The code below resets the chip, then sets the temperature samples to one-minute intervals, and waits for you to press the switch.

```
1 print #1,chr$(&H22);chr$(&H0E);chr$(&H50);          ' clr=1 se=1
2 print #1,chr$(&HA5);                                ' Clear Memory - command
3 delay 0.5                                           ' time for the memory to clear
4 print #1,chr$(&H22);chr$(&H0D);chr$(&H01);          ' interval 1 minute
```

With the press button enabled, temperature logging does not start until the button is pressed and held for at least half a second. If you have the LEDs in the circuit, one or both will flash four times to confirm the start.

The built in real time clock will need programming to make best use of the collected data. I used the date and time in the PC to program the logger chip. When running on the battery, the clock acts like a watch keeping good time, with very little current drain. A battery such as the CR2032 should run the chip for about 10 years, if you don't use the LEDs!

## The Test Program

Here is a short program, which I set up to test the serial interface and read back some fairly raw data.

```
open "COM2:9600,N,8,1,CS0,DS0,CD0,ME,FE" as 1
10
cls
print
print "Commands"
print ""
print " q - Specification Test command - if recording data the LEDs will flash"
print " a - Read Page - zero, parameters, 32 bytes returned"
print " v - read all DS1615 data into the file data.txt"
print " z - Read Temperature - command, use 'a' to see the result"
print " c - Clear Memory and set the reading interval to 1 minute"
print " e - end"
do
 a$=inkey$
loop until len(a$)>0
if len(a$)>0 then
 if a$="e" then
  print "Finished"
  stop
 elseif a$="q" then
' send the Specification Test command - if recording data the LEDs will flash
  print "Specification Test"
  print #1,chr$(&H44);
 elseif a$="a" then
  cls
  ' Read Page - zero, parameters, 32 bytes returned
  print #1,chr$(&H33);chr$(0);chr$(0);
  aa$=input$(34,#1)
  print "-- page 0 ---"
  print "day ";hex$(asc(mid$(aa$,4,1)));" time ";hex$(asc(mid$(aa$,3,1)));
  print ":";right$("0"+hex$(asc(mid$(aa$,2,1))),2);
  print ":";right$("0"+hex$(asc(mid$(aa$,1,1))),2)
  for i=1 to 32 step 2
   if i=17 then print 0.5*(asc(mid$(aa$,i+1,1)))-40
   print hex$(i-1),bin$(asc(mid$(aa$,i,1))),bin$(asc(mid$(aa$,i+1,1)))
  next i
  do                        ' wait for a key press
  loop until instat
 elseif a$="v" then
  cls
  open "data.txt" for output as #2
  l=1
  print #1,chr$(&H33);chr$(&H00);chr$(&H00);             'base data 0
  aa$=input$(32,#1)
  for i=1 to 32
   print #2,asc(mid$(aa$,i,1))
  next i
  ba$=input$(2,#1)                                        'crc
  print l
  l=l+1
  print #1,chr$(&H33);chr$(&H00);chr$(&H20);             'base data 1
  aa$=input$(32,#1)
  for i=1 to 32
   print #2,asc(mid$(aa$,i,1))
  next i
  ba$=input$(2,#1)                                        'crc
  print l
  l=l+1
  print #1,chr$(&H33);chr$(&H02);chr$(&H18);             'serial number
  aa$=input$(8,#1)
  for i=1 to 8
   print #2,asc(mid$(aa$,i,1))
  next i
  ba$=input$(2,#1)                                        'crc
  print l
  l=l+1

  for k=&H220 to &H27F step 32
   hi%=k
   shift right hi%,8
   low%=k and &HFF
   print #1,chr$(&H33);chr$(hi%);chr$(low%); 'alarms
   aa$=input$(32,#1)
   for i=1 to 32
    print #2,asc(mid$(aa$,i,1))
   next i
   ba$=input$(2,#1)                                       'crc
   print l
   l=l+1
  next k

  for k=&H800 to &H87F step 32
   hi%=k
   shift right hi%,8
   low%=k and &HFF
   print #1,chr$(&H33);chr$(hi%);chr$(low%); 'histogram
   aa$=input$(32,#1)
   for i=1 to 32
    print #2,asc(mid$(aa$,i,1))
   next i
   ba$=input$(2,#1)                                       'crc
   print l
   l=l+1
  next k

  for k=&H1000 to &H17FF step 32
   hi%=k
   shift right hi%,8
```

```
   low%=k and &HFF
   print #1,chr$(&H33);chr$(hi%);chr$(low%); 'temperature
   aa$=input$(32,#1)
   for i=1 to 32
    print #2,asc(mid$(aa$,i,1))
   next i
   ba$=input$(2,#1)                                        'crc
   print l
   l=l+1
  next k
  close #2

  print #1,chr$(&H33);chr$(&H10);chr$(&H20);
  aa$=input$(32,#1)
  delay .1
  ba$=input$(2,#1) 'crc
  print #1,chr$(&H33);chr$(&H10);chr$(&H20);
  ba$=input$(32,#1)
  aa$=aa$+ba$
  for i=1 to 64 step 4
   print hex$(i-1), 0.5*(asc(mid$(aa$,i,1)))-40,
   print 0.5*(asc(mid$(aa$,i+1,1)))-40,0.5*(asc(mid$(aa$,i+2,1)))-40,
   print 0.5*(asc(mid$(aa$,i+3,1)))-40
'   print hex$(i-1),bin$(asc(mid$(aa$,i,1))),bin$(asc(mid$(aa$,i+1,1)));
'   print bin$(asc(mid$(aa$,i+2,1))),bin$(asc(mid$(aa$,i+3,1)))
  next i
 elseif a$="z" then
 ' Read Temperature - command
  print #1,chr$(&H55);
 elseif a$="x" then
 ' Write Byte - to address
  address%=&H0E
  idata%=&H10
  print #1,chr$(&H22);chr$(address%);chr$(idata%);
 elseif a$="c" then
  print "interval 1 minute"
  print #1,chr$(&H22);chr$(&H0E);chr$(&H50); ' clr=1 se=1
  print #1,chr$(&HA5);        ' Clear Memory - command
  delay 0.5
  print #1,chr$(&H22);chr$(&H0D);chr$(&H01); ' interval 1 minute
 else
  print #1,a$;
 end if
end if
goto 10
```

It works quite well but it is not very friendly to use. I wanted something better.

## Displaying the Data - a better program

Having worked out how to program the chip I could now collect temperature data.

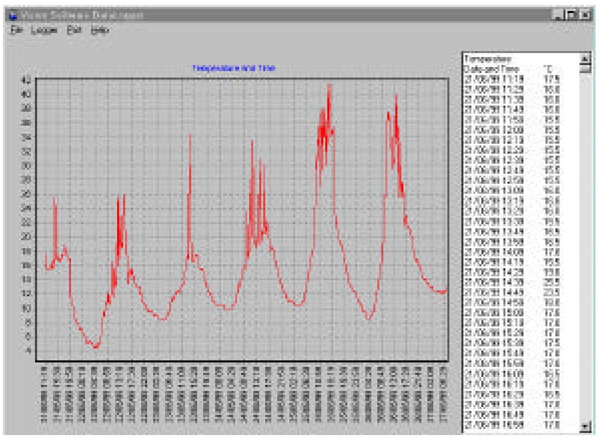
The temperature is measured in 0.5°C steps from -40°C to 85°C. Any temperatures, which have not been set in the logger memory show up as number zero, which decodes as a temperature of -40°C.

You can set the interval between temperature readings in one-minute steps, up to 255 minutes. At one minute the 2048 samples gives recording just over 34 hours. At 255 minutes the total time is just over 362 days, but samples are some 4¼ hours apart. So there are quite a range of times that can be used before the 2048 bytes of memory are filled. If the memory overflows you can set the chip to record the last 2048 readings or freeze the first measurements. The histogram still gives the range of readings, when more than 2048 samples are taken.

To begin with I wrote the data to a file and used a spreadsheet to draw a graph. This worked well but took quite a time to work through. In the end I made a Windows program written in Delphi to control the logger and to draw graphs of the results. Delphi has a clever graph plotter, which draws the data quickly, allowing you to examine the detail through panning, and zooming. With the new program I could also extract extra information that the chip collects. In particular the histogram, which shows the count of the temperatures in two-degree regions, is read and displayed. The chip also logs high and low alarm temperatures, which the new program can set and display. Here are a couple of screens to show the output.

## Results

Now I could go back to the reason for building the circuit. The screen shot in Figure 2 shows the temperature variation through six days in my greenhouse. There is a clear pattern of temperature rising during the day and falling in the night. Some of the other peaks correspond with the sun shining directly onto the DS1615 chip. It can raise the temperature well above the average. A max min thermometer would only show the peak without showing the average. The logger gives much more useful information than a simple thermometer.



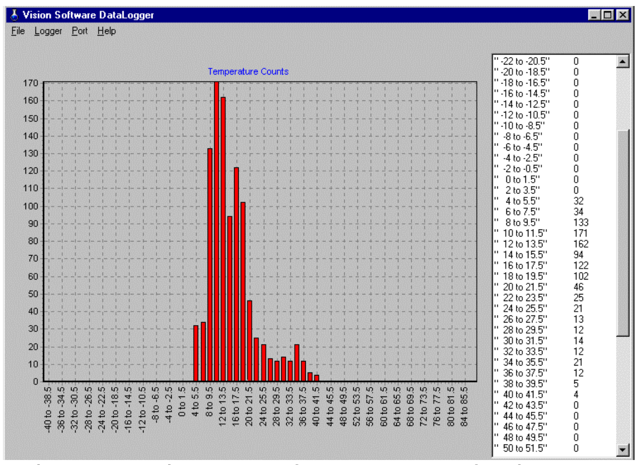
| Temperature Date and Time | °C |
|---|---|
| 21/06/99 11:19 | 17.5 |
| 21/06/99 11:29 | 16.0 |
| 21/06/99 11:39 | 16.0 |
| 21/06/99 11:49 | 16.0 |
| 21/06/99 11:59 | 15.5 |
| 21/06/99 12:09 | 15.5 |
| 21/06/99 12:19 | 15.5 |
| 21/06/99 12:29 | 15.5 |
| 21/06/99 12:39 | 15.5 |
| 21/06/99 12:49 | 15.5 |
| 21/06/99 12:59 | 15.5 |
| 21/06/99 13:09 | 16.0 |
| 21/06/99 13:19 | 16.0 |
| 21/06/99 13:29 | 16.0 |
| 21/06/99 13:39 | 15.5 |
| 21/06/99 13:49 | 16.5 |
| 21/06/99 13:59 | 16.5 |
| 21/06/99 14:09 | 17.0 |
| 21/06/99 14:19 | 15.5 |
| 21/06/99 14:29 | 19.0 |
| 21/06/99 14:39 | 25.5 |
| 21/06/99 14:49 | 23.5 |
| 21/06/99 14:59 | 18.0 |
| 21/06/99 15:09 | 17.0 |
| 21/06/99 15:19 | 17.0 |
| 21/06/99 15:29 | 17.0 |
| 21/06/99 15:39 | 17.5 |
| 21/06/99 15:49 | 17.0 |
| 21/06/99 15:59 | 17.0 |
| 21/06/99 16:09 | 16.5 |
| 21/06/99 16:19 | 17.0 |
| 21/06/99 16:29 | 16.5 |
| 21/06/99 16:39 | 17.0 |
| 21/06/99 16:49 | 17.0 |
| 21/06/99 16:59 | 17.0 |

Figure 1 - Temperature and time graph

Zooming into the data displayed in the graph can be done by drawing a rectangle with the mouse from the top left corner to the lower right whilst holding the left mouse button down. You can also pan the graph by pressing the right mouse button and holding it whilst moving left, right, up, or down. The graph data follows as the mouse moves. To reset the zoom, draw a rectangle with the left mouse button but start at the lower right and end at the upper left. If you zoom in enough you will see the 0.5°C steps in the readings. The time scale on the bottom of the graph will also show individual sample times.

The table on the right of the screen shows the measured data values, so you can look at the detail as numbers. You can mark the data in the table and copy it to a spreadsheet for special processing if you wish.

The DS1615 also collects counts of temperature readings in 2°C intervals. These can be plotted as a histogram as shown in Figure 3. It shows which temperatures are most often recorded. Clearly the higher temperatures around 40°C only happen infrequently with most of the readings around 10 to 12°C, but with quite a few around 17°C.



```
" -22 to -20.5"    0
" -20 to -18.5"    0
" -18 to -16.5"    0
" -16 to -14.5"    0
" -14 to -12.5"    0
" -12 to -10.5"    0
" -10 to -8.5"     0
"  -8 to -6.5"     0
"  -6 to -4.5"     0
"  -4 to -2.5"     0
"  -2 to -0.5"     0
"   0 to 1.5"      0
"   2 to 3.5"      0
"   4 to 5.5"      32
"   6 to 7.5"      34
"   8 to 9.5"      133
"  10 to 11.5"     171
"  12 to 13.5"     162
"  14 to 15.5"     94
"  16 to 17.5"     122
"  18 to 19.5"     102
"  20 to 21.5"     46
"  22 to 23.5"     25
"  24 to 25.5"     21
"  26 to 27.5"     13
"  28 to 29.5"     12
"  30 to 31.5"     14
"  32 to 33.5"     12
"  34 to 35.5"     21
"  36 to 37.5"     12
"  38 to 39.5"     5
"  40 to 41.5"     4
"  42 to 43.5"     0
"  44 to 45.5"     0
"  46 to 47.5"     0
"  48 to 49.5"     0
"  50 to 51.5"     0
```

Figure 3 – Histogram of temperatures for the same data as in Figure 2

Now I need a second logger to compare the inside and outside temperatures!

At last I have a way to log the temperature in my greenhouse. Perhaps I'll have a go at using the alarm output to turn on a heater in the cold of winter.

The program code can be downloaded as a file from the Vision Software site at www.visionsoftware.freeserve.co.uk. Both the PowerBasic and Delphi executable files are available.

1. References: DS1615 datasheet, Dallas Semiconductor, www.dallas.com
2. Application Note 116, Dallas Semiconductor

Thanks to Dallas Semiconductor and Silver Birch Marketing Ltd. for the samples.

## Components List:

| | |
|---|---|
| Resistors: | |
| R1, R2, R3 | 1K |
| Capacitors: | |
| C1 | 22uF/12V electrolytic |
| Semiconductors: | |
| D1 | 1N4148 |
| D2 | LED low current green |
| D3 | LED low current red |
| U1 | MAX233 |
| U2 | 78L05 |
| U3 | D1615 |
| Miscellaneous: | |
| X1 | 32kHz watch crystal |
| SW1 | push to make |
| J1 | 9pin D socket |
| B1 | CR2032 lithium battery |

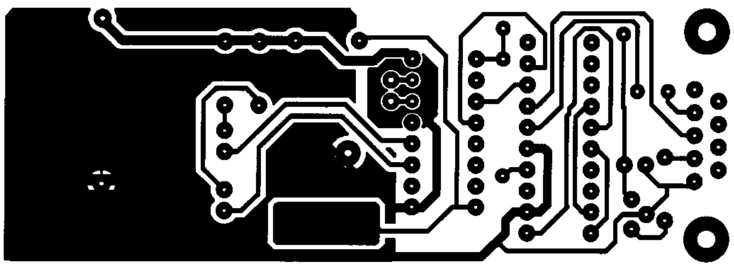


Figure 4 – PCB copper side foil

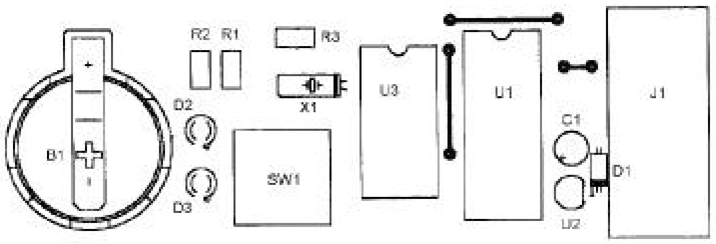
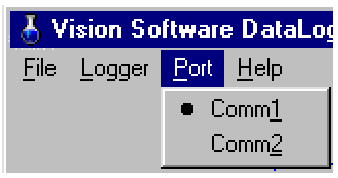


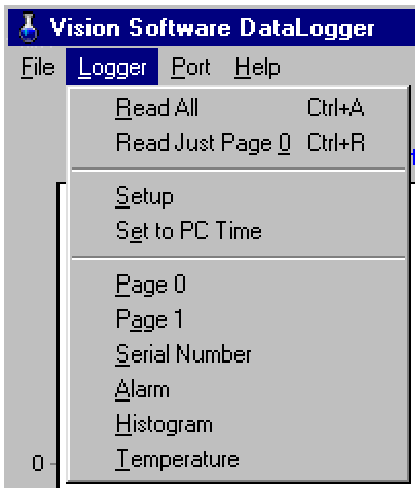
Figure 5 – Component Overlay

# Program Instructions

First select the serial communications port. If you select the wrong one the program may crash.



The logger needs to be set up by setting its clock to the same time as the PC. This only needs to be done if the time becomes wrong, or the first time the logger is powered.

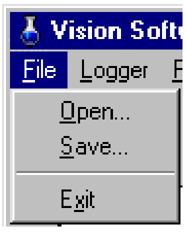


To start logging, use the *Setup* menu to choose the interval between temperature readings.

The *Read Just Page 0* is used to check that the clock has been set and that measurements have started.

The *Read All* collects all data stored in the DS1615 and displays the data as it is received. The *Temperature* data takes several seconds to transfer.

The Logger menu items *Page 0*, *Page 1*, *Serial number*, *Alarm*, *Histogram* and *Temperature* display the associated data in memory



The File menu allows saving of data that has been read from the DS1615. Two files are saved. One can be read into a spreadsheet, and the other can be read back into the program for graphing.

Zooming and Panning is done in the graph window.

1. Press the left mouse button and hold whilst dragging to the lower right, creating a box on the graph. Release the mouse button and the graph should zoom to the size of the box. If this does not work it may be because the box has to be reasonably large. Try a larger box.

2. To pan, press and hold the right mouse button. The graph should move with the movement of the mouse.

3. Making a left button box, but starting at the lower right and dragging to the upper left resets the Zoom.