MG002 Real Time Clock

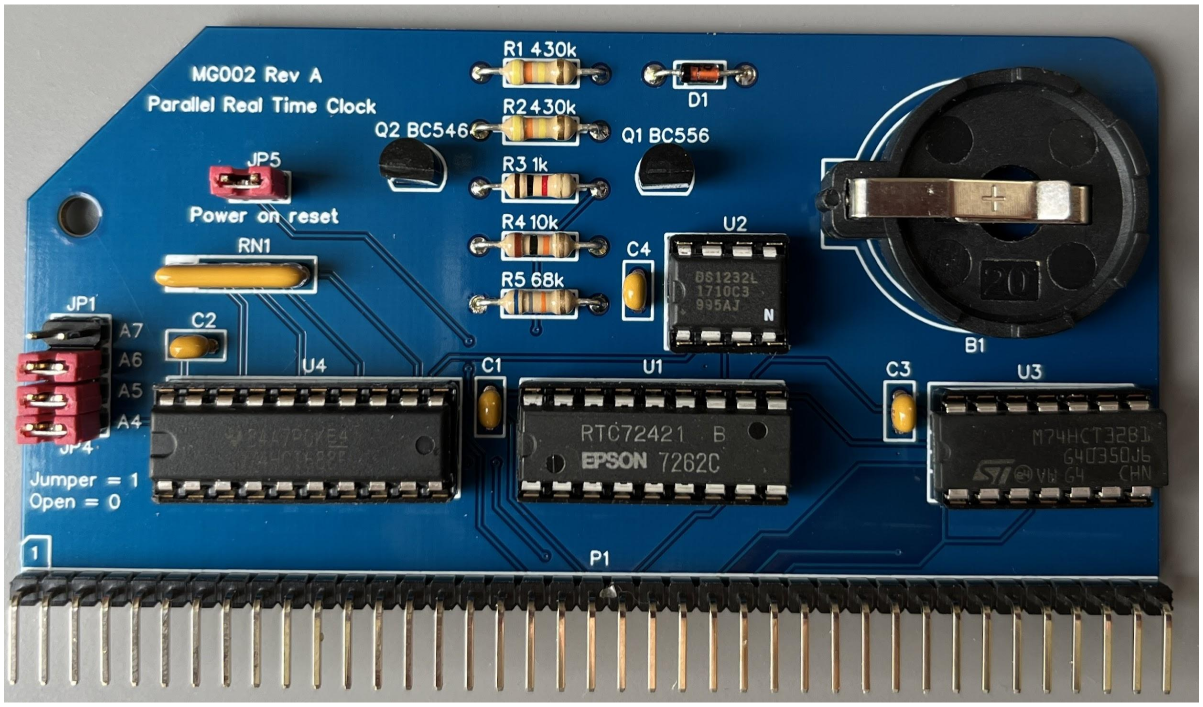
What is it?

MG002 is a real time clock designed for RC2014. It communicates using a 4 bit parallel address bus, and can be initialised and read using standard RC2014 BASIC commands.

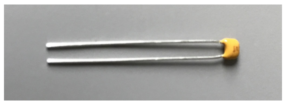
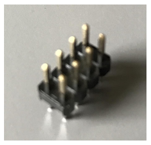
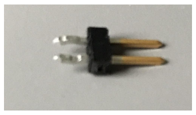
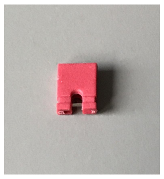
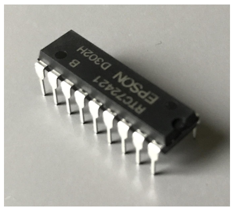
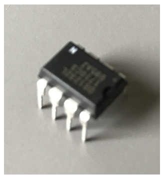
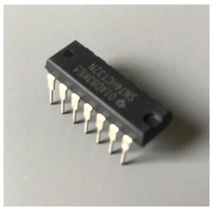
Its address can be set anywhere in the range 0 to 255, and occupies 16 address locations.

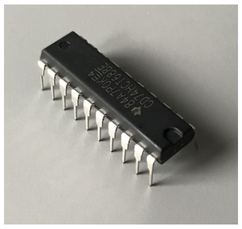
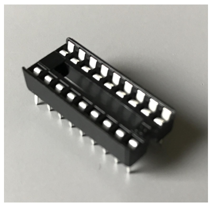
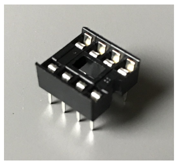
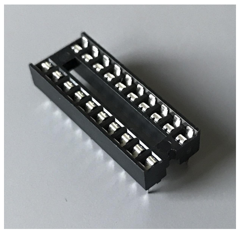
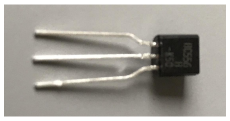
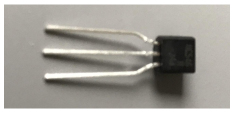
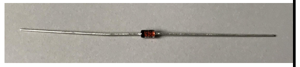
The MG002 can be used to maintain and output time, date and/or numeric day of week. A lithium backup battery (not supplied) keeps the clock running when power is removed. When combined with a display such as the MG001, and a host RC2014, it can be used to provide a fairly accurate clock and/or calendar.

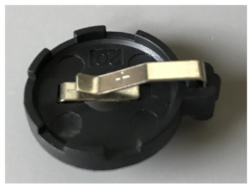
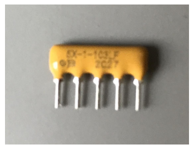
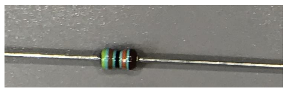
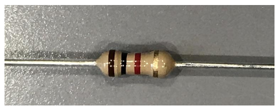
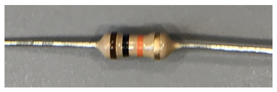
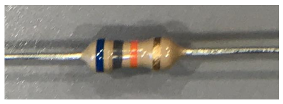
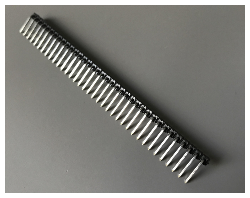
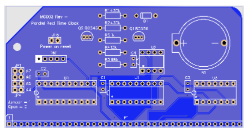
Reasonably authentic 1980’s technology is used throughout, without modern capabilities such as serial I2C busses (nothing wrong with them, it’s just not what I’m trying to do here).



What’s in the kit?

Name	Quantity	Description	Picture	Present?
C1-4	4	Capacitor, ceramic, 100 nF		,
JP1-4	1	Header, male, 2 x 4 pin, straight		,
JP5	1	Header, male, 1 x 2 pin, straight		,
JP1-5 Shunts	5	Jumper shunt		,
U1	1	RTC72421B		,
U2	1	DS1232LP		,
U3	1	74HCT32		,

U4	1	74HCT688		,
U1 Socket	1	18 Pin DIP socket		,
U2 socket	1	8 Pin DIP socket		,
U3 socket	1	14-pin DIP socket		,
U4 Socket	1	20-pin DIP socket		,
Q1	1	BC556 Transistor		,
Q2	1	BC546 (Or BC548) Transistor		,
D1	1	1N4148 Diode		,

B1	1	Battery Holder, Coin Cell - 20mm		,
RN1	1	Fixed Network Resistor, 10 kohm		,
R1, R2	2	430k Resistor, 250mW		,
R3	1	1k Resistor, 250mW		,
R4	1	10k Resistor, 250mW		,
R5	1	68k Resistor, 250mW		,
P1	1	Pin Header, Right Angle		,
PCB	1	MG002 PCB		,

What's not supplied?

A CR2032 battery to fit in B1. I have omitted this as the rules are complex as to whether these can be mailed. You probably have one lying around, and if not they are easily available. MG002 will work without it, but obviously there will be no time backup with power removed.

How do I build it?

There’s a good chance you will have some soldering experience, as you’re likely to have built an RC2014 or equivalent to plug your MG002 into. If you haven’t, I recommend searching for an online tutorial, there are some good ones on YouTube.

Recommended tools include:

- Soldering iron (ideally temperature controlled)
- Multicore solder
- Small snips to cut off leads
- Small pliers
- Desoldering pump and/or braid
- Anti-static wrist strap (or steer clear of materials that cause static and touch a grounded object every now and then).

The normal rule of thumb is to solder the lowest height components first, working up:

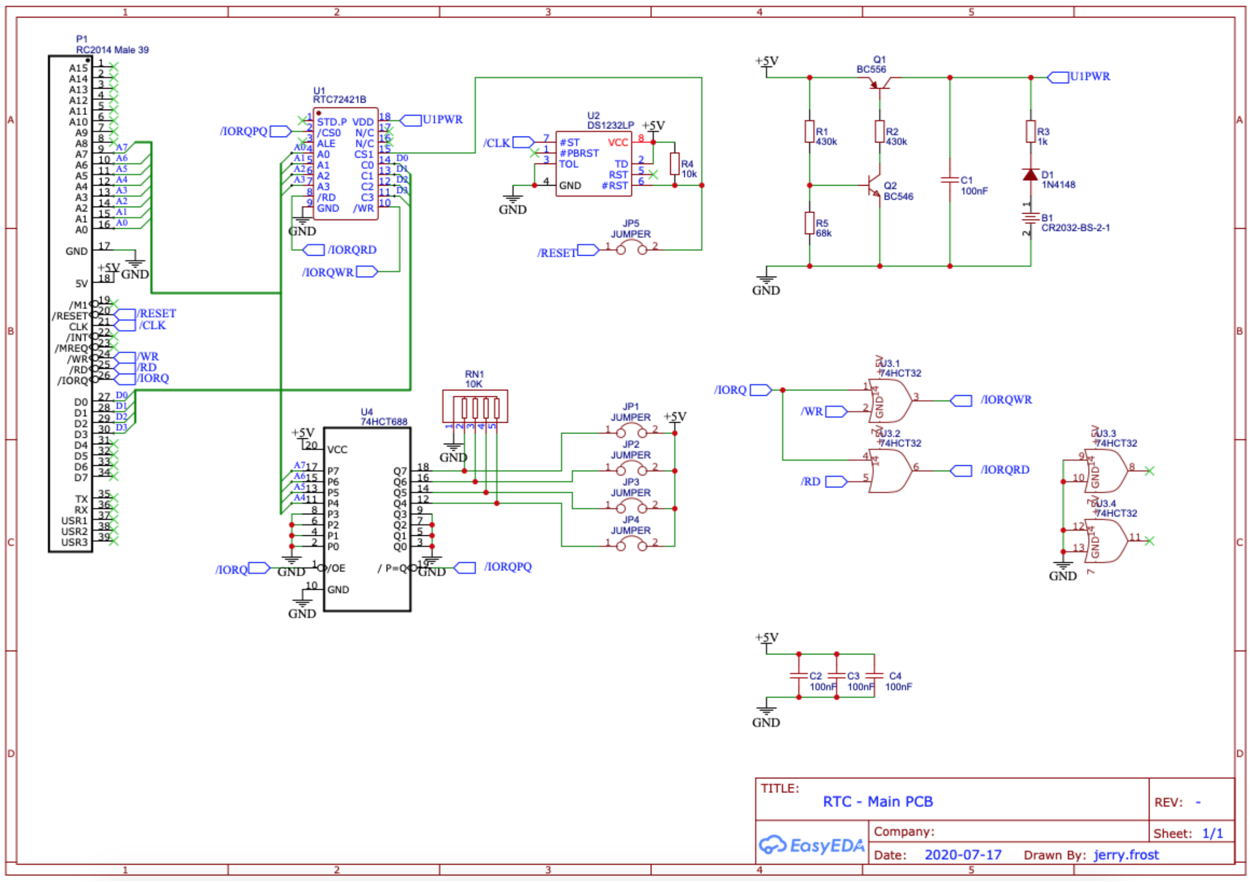
- R1-5. Orientation doesn’t matter. R1 to R5 are different values (with the exception of R1 and R2, which are identical). Please take care to fit the right resistor in the right place
- D1. Orientation matters for D1. The cathode (end of diode with band) should be aligned with the legend, in other words it should be pointing away from the battery holder B1. If D1 is reversed, time will not be maintained when power is removed, and the circuit will attempt to charge B1. R3 is there to keep this current to a level that is safe for B1, should D1 be reversed (or fail short-circuit)
- P1. This is normally supplied with 40 pins, therefore one needs to be carefully cut off using a sharp knife. Then, solder one joint only, check the alignment, melt solder and correct alignment if required before soldering remaining joints. If supplied with 36 pins, fit at the Pin 1 end of the PCB
- C1-4. Orientation doesn’t matter
- Sockets for U1-4 (do not fit ICs yet). Similarly to P1, solder two opposite corners, check the socket is flat on the board before continuing. Make sure the notches at the end of the sockets match with the PCB graphics, to reduce the risk of installing the ICs the wrong way round

- RN1. Note that the dot on one end should align with the marking on the PCB RN1 graphic (the dot should go closest to JP1)
- Q1-2. Align these as per the graphics on the PCB. Please note that Q1 and Q2 are different types of transistor and should not be mixed up (the type numbers for each are printed on the PCB). Their leads may need straightening in order to fit neatly through the PCB holes
- B1. Align as per the graphic on the PCB
- JP1 to JP5. JP1 to JP4 is a single assembly. JP5 will fit through the PCB holes better if its leads are straightened out

If you have flux cleaner, clean all joints. Now inspect them carefully for issues (a magnifying glass of some sort can be very helpful, the camera on some phones works quite well).

The final step prior to plugging into the host system and testing is to fit the ICs into their sockets. The IC legs will probably need a bit of gentle bending on a table or similar surface, to bring the two rows a little closer to each other. Pay attention to orientation (even after all this hard work, it’s easy to get wrong).

How does it work?



JP1-4 are used to set the address of the upper 4 address bits (A4 to A7). When the address bus matches this, U4 sends a "/IORQPQ" signal to U1, which enables reading or writing. U3.1 and U3.2 tell U1 whether a read or write is requested. U1 does all the Real Time Clock hard work, inputting, outputting and maintaining time and date (all without an external crystal).

One thing that U1 does not do is manage the switching between 5V and battery power. This is handled by Q1, Q2 and associated circuitry:

- When 5V is present, Q1 and Q2 pass it to U1 with a very small voltage drop (they are both biassed hard on)
- When 5V disappears, Q1 and Q2 switch off and prevent the battery voltage being used to power anything except U1
- D1 prevents the 5V attempting to charge the battery. Should D1 fail (go short circuit), R3 limits the current to a level that is safe for the battery

U2 fulfills a couple of functions:

- When 5V power disappears, it puts U1 into sleep (running on battery) mode quickly
- When 5V power comes back, it holds U1 in sleep mode for 250ms (to enable power to stabilise) before setting it going in powered mode
- As a bonus, by jumpering JP5, U2 can be used to provide a power up reset onto the RC2014 reset line (which should avoid the need to press reset after power up). It works pretty well on my RC2014, but I've not tested it extensively

How do I use it?

The following example uses BASIC as employed by the RC2014 Classic etc. For users that have an RC2014 Pro or similar, my website (https://jerryfrost1.wixsite.com/tech) provides C source code and executables suitable for CP/M.

The first step is to set the address. If you are using the standard RC2014 serial I/O board, then setting A7 to zero (jumper removed) ensures the MG001 will not clash with it. If you have other I/O boards fitted, then you will need to take them into account also.

To use an example:

Jumper	A7	A6	A5	A4
Status	Open	Jumper	Jumper	Jumper
Value	0	64	32	16

Summing the values gives a decimal address of 112 (if A0, A1, A2 and A3 are all 0), through to 127 (if they are all 1). Therefore there are 16 addresses to read from/write to, all of which have 4 data bits. I won't bore you with full details here, because a fully annotated example of code is coming up (and also because Epson don't like their documentation being reproduced), but Epson's Application Manual has all the detail and can be found at:
https://www5.epsondevice.com/en/products/rtc/rtc72423.html

(Although the link is for the RTC-72423 device, the Application Manual also covers RTC-72421).

Once you’ve set the address and powered the board up, you’ll want to set the time and/or date and then be able to read it. The following code gives an example (reading and writing 24 hour time) which can be modified in conjunction with the Epson Application Manual. The initialisation and set up takes a few lines, but once it’s running the code to read the time is very simple:

BASIC CODE	Notes
10 INPUT "WARM START";W\$ 20 IF W\$="y" THEN GOTO 300 30 IF W\$="Y" THEN GOTO 300	*If RTC is running then only lines 300 onwards are required*
50 INPUT "Tens of Hours (24Hr Clock)";G 60 INPUT "Hours";H 70 INPUT "Tens of Minutes";L 80 INPUT "Minutes";M	*Input the time as 4 separate digits into 4 variables*
100 OUT 127,4 110 OUT 125,0	*Lines 100, 110 start the counter and initialize the control registers*
120 OUT 125,1	*Set the hold bit*
130 IF (INP(125) AND 2) = 0 THEN GOTO 200	*Check whether busy bit (bit 2) is set, if it is:*
140 OUT 125,0	*Take off hold bit….*
150 GOTO 120	*...and try again*
200 OUT 127,7	*Otherwise stop and reset counter*
210 FOR Y=112 TO 124 220 OUT Y,0 230 NEXT Y	*Clear all time and date registers to zero*
240 OUT 117,G 250 OUT 116,H 260 OUT 115,L 270 OUT 114,M	*Lines 240 to 270 write the 4 time digits into RTC*
280 OUT 127,4 290 OUT 125,0	*Start the counter and take off hold bit. RTC is now up and running…*
300 LET G=(INP(117) AND 3)	*Read in the time digits. The “AND*

```
310 LET H=(INP(116) AND 15)       15" gets rid of the 4 most significant
320 LET L=(INP(115) AND 15)       bits (the RTC only uses the other 4).
330 LET M=(INP(114) AND 15)       In the case of tens of hours we only
                                  need the 2 least significant bits
                                  (hence "AND 3")
340 OUT 56,M                      Send time to an MG001 seven
345 OUT 57,L                      segment display (Rev B onwards)
350 OUT 58,H
355 OUT 59,G
360 FOR X=1 TO 1000               This loop is purely to slow down the
370 NEXT X                        rate of time requests
375 PRINT G""H":"L""M             Print out time via terminal
380 GOTO 300                      And repeat!
```

Notes

The Epson Applications Manual calls for the following sequence prior to reading time/date etc:

1. Set hold bit
2. Check busy bit is not set (if it is take off hold bit and go back to 1.)
3. Read data required
4. Release hold bit

This is similar to the sequence in lines 120 to 150 above. The rationale is that this prevents reading a value when it is in the middle of an update (which could cause a corrupted number). I've never implemented this and have never seen an issue. For the use case of a clock, a digit being corrupted for a second or so is relatively trivial, in any case.

Acknowledgements/Legal

MG002 has been designed for RC2014 with reference to the RC2014 Module Template. All pinouts used and the physical outline are in compliance with the RC2014 Module Template.

RC2014 is a trademark of RFC2795 Ltd.

MG002 has been designed for hobbyist use only and is not to be used for safety or business critical applications.