

## The MSF Signal

The MSF signal originally transmitted from Rugby is now transmitted from Anthorn in the UK (latitude 54° 54' N, longitude 3° 16' W).

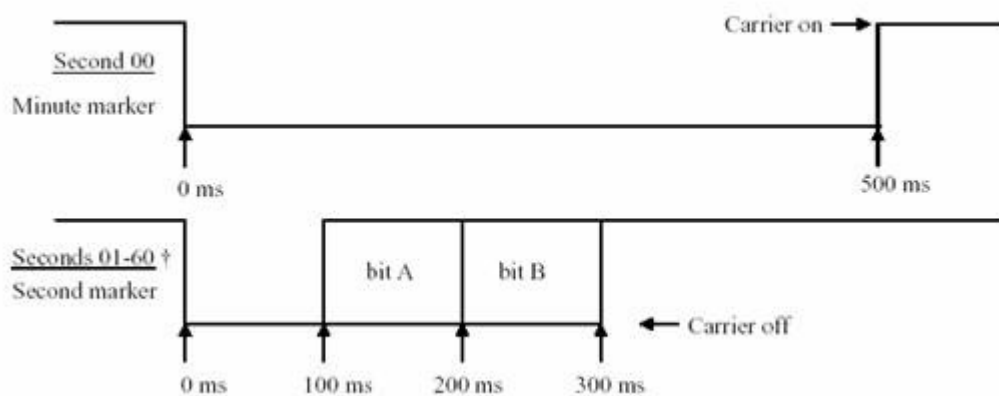
It is the principal means of discriminating the UK national time and frequency which are maintained by the National Physical Laboratory (NPL).

The signal strength is >10mV/m at 100km and >100uV/m at 1000km from the source it has a carrier frequency of 60kHz within 2 parts in 10000000000000.

The MSF time and date code is serially transmitted using simple on-off carrier modulation. The timing edge is governed by the seconds and minutes of Coordinated Universal Time (UTC) which is always within 1s of Greenwich Mean Time (GMT).

The start of each UTC second is preceded by 500mS of carrier and the second marker is transmitted with 1ms +/- accuracy.

The data bits following the minute marker contain 2 parts bitA which carries the main data and bitB which carries parity and the daylight savings time (DST) flag.



The principal is simple, we first look for 500ms carrier and once found we then restart a 1ms timer and reset all msf\_data registers, next we sample the signal every 1ms.

Within the 1ms timer interrupt a counter is incremented and at specific counts the signal is sampled.

1. At 50ms bits, bitA and bitB are set to 1.
2. At 150ms the signal is sampled, if high bitA is cleared to 0. If low, bitA remains 1.
3. At 250ms the signal is sampled, if high bitB is cleared to 0. If low, bitB remains 1.
4. At 650ms the bit is shifted into the appropriate register and the msf\_second register is incremented. Every MSF bit is followed by at least 700ms.

The whole process then repeats every 1second until all bits are clocked in. The 1second is determined by the MSF signal.

At the end of the minute, the parity is checked and if found to be odd, the RTC registers are updated and the 1s timer interrupt is synchronised.

The msf\_second register is used to determine which bit number is being sampled and if within a specific range the bitA data is shifted into the appropriate register, bitB is only used for msf\_second number 53 thru 57 which will be described later.

If msf\_second =0 then ignore.

If msf\_second >=01 and <=16 then ignore.

If msf\_second >=17 and <=24 then bitA is right shifted into msf\_year 8 bit register.

If msf\_second >=25 and <=29 then bitA is right shifted into msf\_month 8 bit register.

If msf\_second >=30 and <=35 then bitA is right shifted into msf\_date 8 bit register.

If msf\_second >=36 and <=38 then bitA is right shifted into msf\_day 8 bit register.

If msf\_second >=39 and <=44 then bitA is right shifted into msf\_hour 8 bit register.

If msf\_second >=45 and <=51 then bitA is right shifted into msf\_minute 8 bit register.

If msf\_second =53 then bitB is stored into msf\_offset

If msf\_second =54 then bitB is stored into msf\_year\_parity

If msf\_second =55 then bitB is stored into msf\_month\_date\_parity

If msf\_second =56 then bitB is stored into msf\_day\_parity

If msf\_second =57 then bitB is stored into msf\_hour\_minute\_parity

If msf\_second =58 then ignore

When msf\_second reaches 59, we check the parity and make sure everything is as it should be before setting the RTC registers. We then also resynchronise the RTC 1s timer interrupt

The parity should always be odd, so if the msf\_year is 07 represented as 00000111 binary should have a msf\_year\_parity of 0 to be valid. Which will make 00000111 + 0, there are an odd number of bits in the value in this case 3 bits are set to 1.

Another example if the time is 11:32 represented as 001011 0100000, we can see there are 4 bits set to 1, so the msf\_hour\_minute\_parity should be 1 to be valid. Which will make 001011 0100000 + 1 now there are an odd number of bits set, in this case 5 bits are set to 1.

More detailed information about the MSF time signal is available on the NPL website at [www.npl.co.uk](http://www.npl.co.uk)

### **The EM2S MSF Module**

The signal is received through the EM2S MSF module from Galleon, they also supply a matched ferrite wound antenna.

The module measures just 13x25mm and just requires a supply voltage and antenna to function.

The output signal is already de-modulated from the module and connects to the INT0 external interrupt of the microcontroller, the signal is open-collector and the microcontroller has an internal pull-up resistor so there is no need for an external resistor. The interrupt is generated on both rising and falling edges which is a neat feature of the Atmel AVR range of microcontrollers.

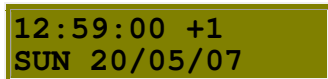
The module and antenna are low cost available from Galleon [www.galleon-uk.com](http://www.galleon-uk.com) and other suppliers.

### **LCD Module**

The LCD module used is a 16 character 2 line display with the common HD44780 display driver. It is operated in 4 bit mode to save on IO pins.

The firmware could easily be adapted for other types of display such as 7 segment, VFD or even nixie tubes.

The display will show time, GMT offset, day, date, month and year.

A photograph of a small, rectangular LCD display with a yellowish-green background. It shows two lines of text in black. The first line reads "12:59:00 +1" and the second line reads "SUN 20/05/07".

12:59:00 +1  
SUN 20/05/07

### **Sync LEDs**

As a debug tool, two optional LEDs are there to show the status of the clock.

The first one RTC tick is toggled for each tick of the RTC 1s interrupt and the second MSF tick is toggled for each bit of the MSF signal.

If both the MSF and RTC clocks are in sync then both LEDs will flash together or alternate.

### **Other Hardware**

I chose a ATmega32L part for this project, the code would quite easily fit a smaller micro, but I had plenty of the 32L parts to hand.

The micro is clocked at 8MHz and has a separate 32.768KHz crystal to control the 1s interrupt for keeping the time when the MSF signal is not being received.

The whole circuit runs from a single 5 volt power supply and will consume about 100mA when all LEDs are on. An onboard 7805 voltage regulator is used to set the 5 volts.

### **Firmware**

The firmware is written in C for the Mega32 AVR and was compiled using the IAR AVR compiler. The firmware can easily be adapted for other micros if required.