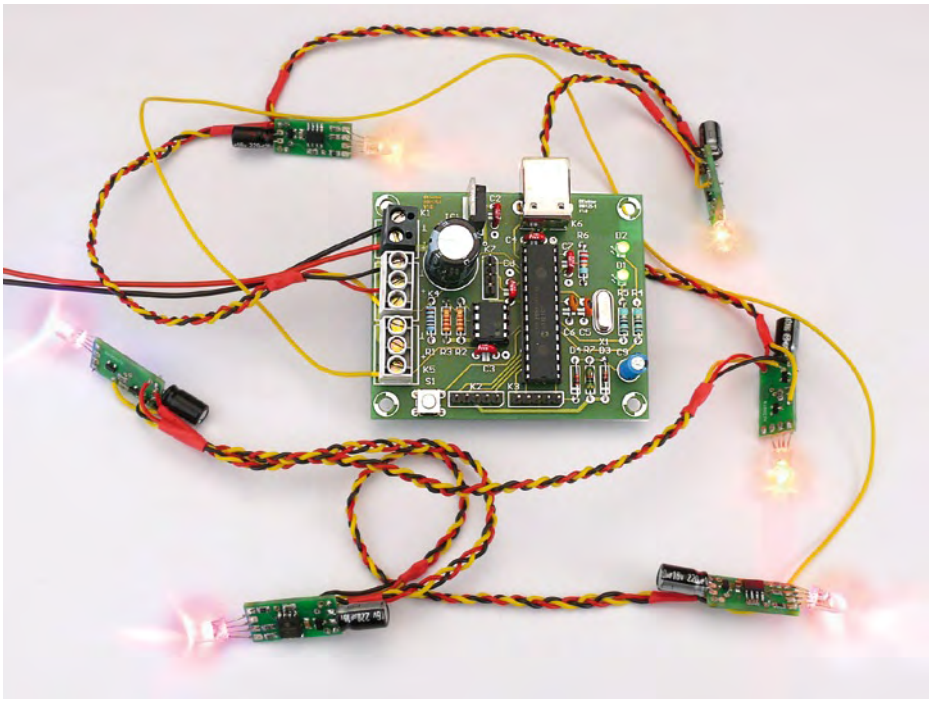
Top-of-the-Bill Lights Sequencer

Programmable fairy lights



By Boris Lecourt (France)

As Christmas approaches, rope-lights and fairy-lights are starting to appear in supermarkets everywhere. They're ridiculously cheap, it's true — but don't you think they're a bit short on originality? The project described in this article attempts to put this right, admittedly at greater cost, but what fun to build your very own Christmas lights, entrancing and totally unique — and what a fine present they'd make, too!

These lights let you produce the most dramatic lighting effects, by altering the countless available movement patterns (variable-speed chaser, random illumination, short or long flashes, etc.), and adjusting the colour (32 levels), saturation (16 levels, including white), and brightness (24 levels) — that means over 10,000 possible colour permutations!

Broadly speaking, the system consists of a master module that drives from 1 to 62 slave modules, identical in all respects (the 'lights'), with a 9 V PSU, supplying at least 2 A for 30 lights. The power supply is distributed to the master and lights in parallel (**Figure 1**).
The master drives all the lights over a single wire that carries a serial control signal. During the initialization stage, a return signal allows it to count the number of lights, which is then stored in the PIC's internal EEPROM. Thus this return signal is used only once in the fairy-lights' lifetime, and can then be disconnected, meaning that only one end of the string of lights has to be connected to the master module.
The master sends regular control signals to the lights to set the hue, brightness, and saturation for each light. The current master program runs through four types of multi-coloured chase sequences and random coloured or white flashes, but it's very easy to add others by programming the master's microcontroller.

The cost to build...

... is mainly affected by the price of the components forming the individual lights. By searching around on the Internet (for example, LEDs can be found on eBay for around 10 pounds for 50 pcs), the cost of each individual light can easily be brought below £2.50 (excluding PCB).

Hardware

The master module (**Figure 2**) consists of a 5 V regulator (IC1) and associated components, a PIC18F2550 microcontroller running at 20 MHz, and a reset circuit for the PIC (R7, D3, D4, and C9). It includes other sections for the extensions that are **optional**, and so **not yet implemented**:

- visual indicators (D1 and D2) to indicate the status of the master;
- a push-button (S1) for changing the movement pattern;
- a TTL-level serial port connector (K7) for communicating with the PIC;
- an EEPROM (IC2) for storing movement sequences;
- an analogue input (K2) so the movement can follow the rhythm of the music — for example, by connecting a mic/amplifier circuit;
- a USB connector (K6) so the fairy-lights can be driven from a computer.

Only two pins of the PIC are needed to provide the interface with the individual lights. One pin configured as an output (CCP1) transmits the control signal to the first light via connector K4. One other pin configured as an input (CCP2) receives the

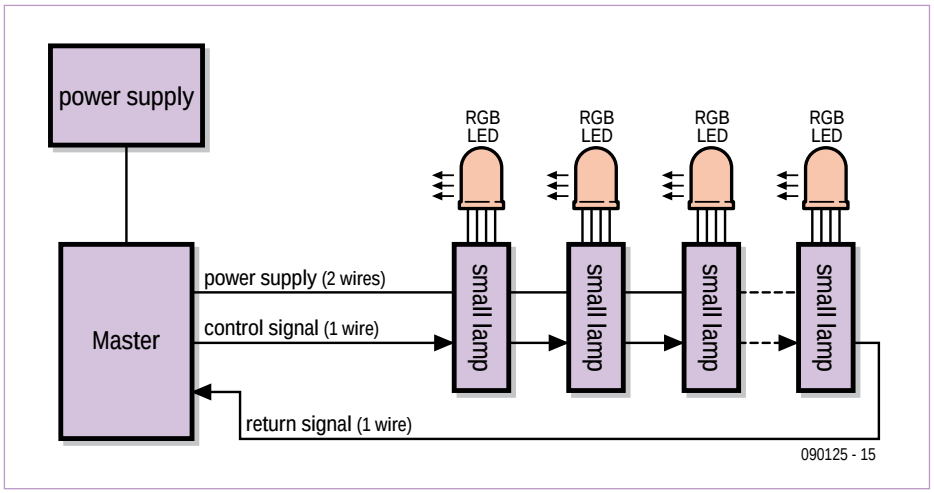


Figure 1. Block diagram of the light string.
You can connect up to 62 slave modules (individual lights).

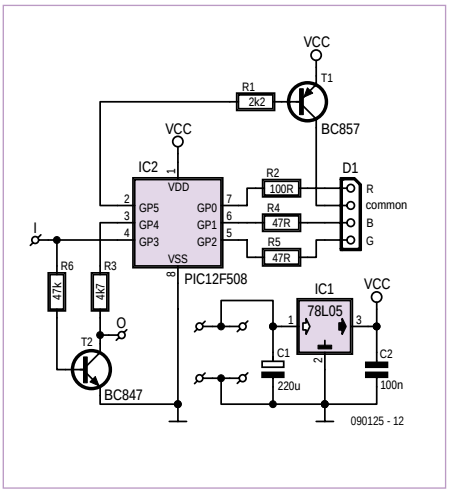


Figure 3. The circuit diagram for one of the lights. To be repeated 62 times, if you can afford it...

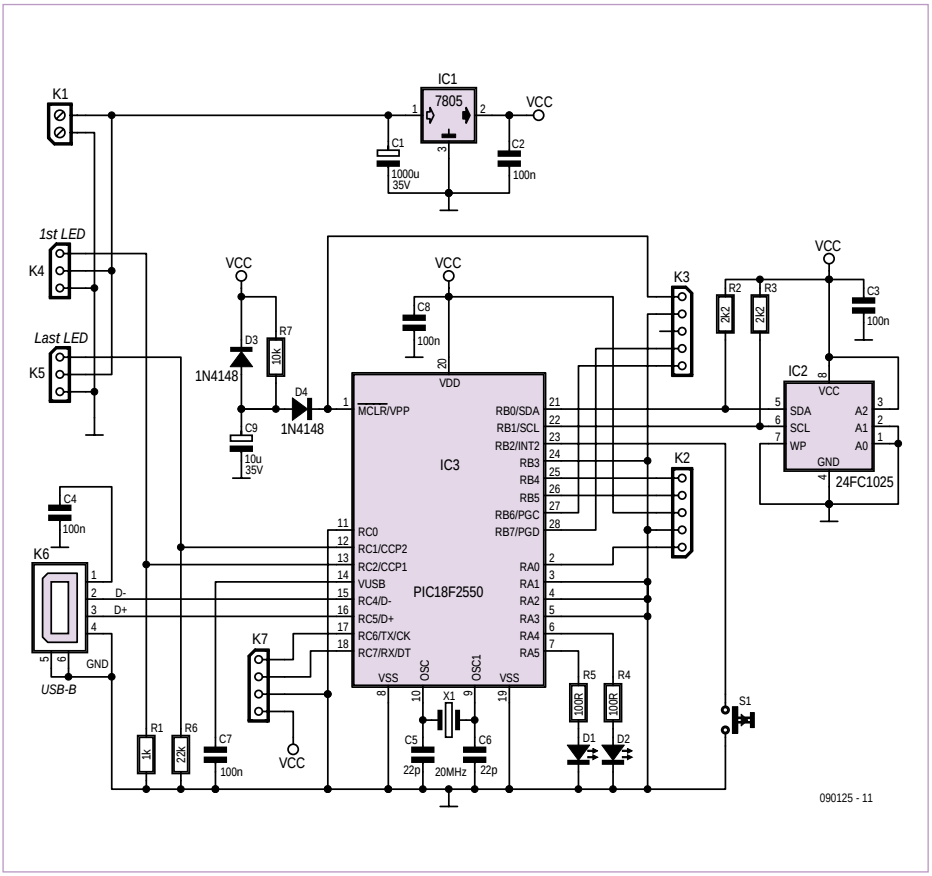


Figure 2. The master module circuit diagram.
The EEPROM IC2 could be used for storing sequences.

return signal from the last light via connector K5. These signals are pulled down to 0 V via resistors R1 and R6, for reasons which will be explained later.
Connectors K4 and K5 convey the power rail to all the individual lights in the string.

Circuit of the individual lights

Each individual light (**Figure 3**) consists of a small 8-pin 12F508 PIC microcontroller (IC2), a 78L05 5 V regulator, two switching transistors (T1 and T2), and an RGB LED (D1) with current limiting resistors. For better colour rendering, you can matt the front of the LEDs using fine glass-paper, which helps improve the mixing of the R, G, and B colour components.
All the input/output (I/O) pins of the PIC are used. Three pins GP0 to GP2 are configured as outputs and are connected to the LED's R, G, and B cathodes. A '0' on these pins allows current to flow in the LED elements. One pin, GP5, also configured as an output, is used for controlling all three elements together. It drives the LED anode via transistor T1, allowing a current of 60 mA to be switched, greater than the PIC output alone could handle (20 mA). PWM (pulse width modulation) signals from these pins make it possible to adjust the brightness of each element, as well as the overall brightness of the LED triad as a whole.

Pin GP3, configured as an input, is devoted to receiving the drive signal from the previous module (or from the master module, if this is the first module in the series). This signal is regenerated and inverted by transistor T2 before being passed on to the next module (see below). The collector of this transistor is fed from pin GP4 of the PIC, configured as an output. The function of GP4 is to be able to inhibit the signal sent to the next module when it is taken to 0 V. This function is used in the counting stage when the fairy-lights are first initialized.

The PIC and the LED are fed via the 5 V regulator, which can supply up to 100 mA, enough to light all three LED elements continuously at their maximum permitted current (20 mA each). This regulator is powered from the 9 V rail that comes directly from the master module's PSU.
We have designed PCBs for the master and slave modules, available from the Elektor website [1].

Software

The master module's PIC program was produced in C with the help of the MPLAB MCC18 compiler [2] (free version 3.21).
The PIC program for the individual lights was produced in C using the CC5X compiler [3] (free version 3.3A) which generates simple optimized assembler code that is very close to the C code.
The two pieces of software can be downloaded from the web page for this article [1].

All the ingenuity in these fairy-lights lies in the software, so it's considerably more complicated than the actual hardware itself. This software uses some interesting techniques that can be employed in other applications. Even though all the individual lights contain the same software, each light can be addressed individually, without needing to be configured first. The individual lights can be interchanged, or a failed one can be replaced, without changing the behaviour of the string as a whole.

We're using this technique here to produce a string of fairy-lights, but by replacing the RGB LEDs with relays and using the appropriate hardware (and modifying the software, of course), you could easily produce, for example, a modular garden watering and irrigation system — or a home automation system for adjusting the lighting in the various rooms in your house. What's more, the master module can be expanded using a USB port, for example, or an EEPROM. So there's no shortage of potential applications.

So, how does this software work? Well, take a look at **Figures 4** and **5** for an overview, and read the following description carefully.

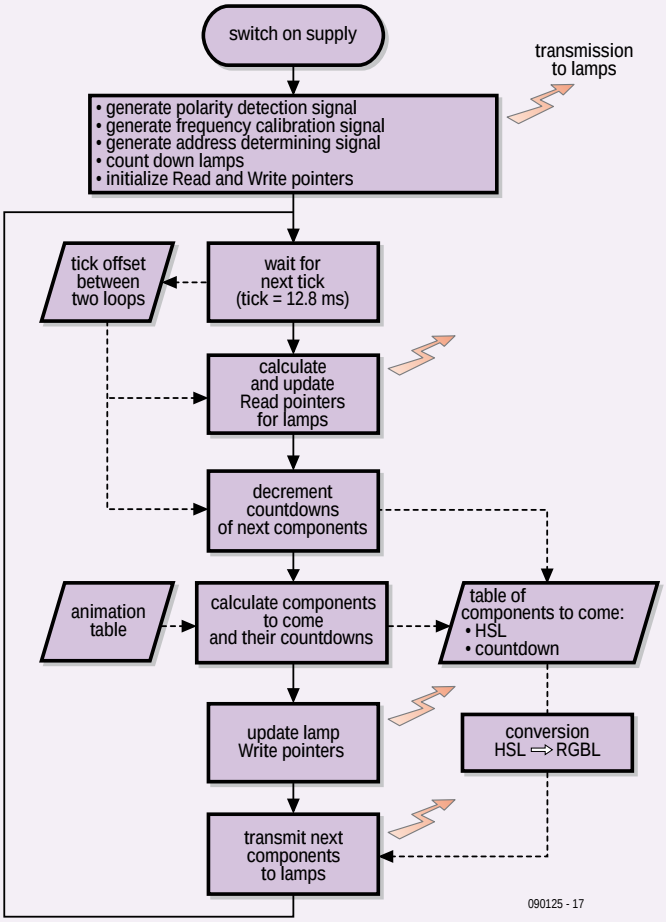


Figure 4. Flow chart for the master program.

Initialising the light string

When power is first applied, the master and the individual lights start a 3-stage initialisation process:

1. Polarity detection;
2. Frequency calibration;
3. Addressing and counting.

Polarity detection

This stage allows each individual light to determine if it is separated from the master by an odd or even number of other lights, in order to allow for the inversions caused by the T2 transistors in decoding the drive signals.

At initialization, the master PIC outputs are at high impedance, and so resistor R1 pulls the CCP1 output down to 0 V. The second light and all the others in even positions now detect a '1' on their GP3 inputs. Using a program variable that stores this polarity, these lights will from now on invert the GP3 input before interpreting it.

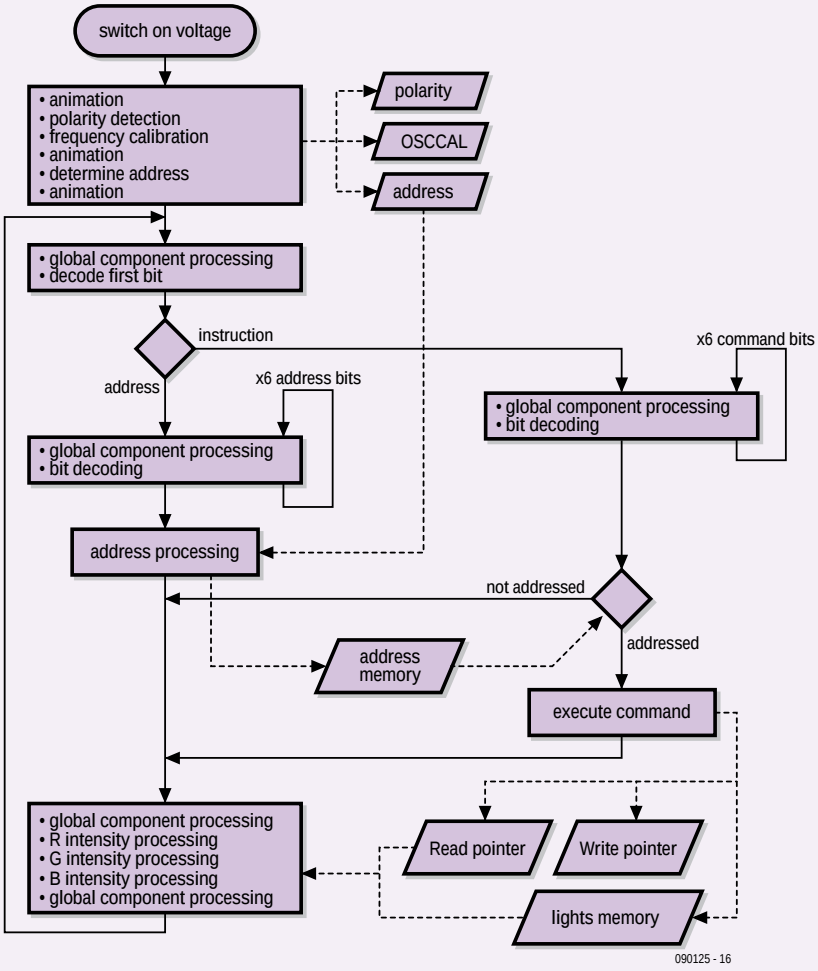


Figure 5. Flow chart for the lights program.

In the following explanations, we'll use 'N' to describe a '0' (0 V) and 'P' to describe a '1' (5 V) on the GP3 inputs of the 'odd' lights (with no inversion of the CCP1 signal). The opposite applies to the even lights (with CCP1 inversion).

Frequency calibration

After detecting the polarity, each light waits for a P on its GP3 input. At this moment, the master starts transmitting a square-wave signal with a period of 200 µs for one second. At the same time, each light starts a process of measuring the period of the square-wave signal received on GP3 using the PIC's T0 timer register. Each time GP3 changes state, a measurement of the period is available to regulate the PIC's clock rate. This is achieved by adjusting the OSCCAL register so as to get closer to the measured 200 µs period. The process stops when the measured difference falls below a certain threshold. Lastly, to end this stage, each light waits for the master to settle the GP3 signal at N for longer than 130 µs.

Addressing and counting

After completing the preceding step, the master generates a sequence of 64 cycles of a square-wave signal identical to the previous one (**Figure 6**). The modules will then 'collaborate' so that each can determine its own address, in the following manner:

1. wait for a P on GP3;
2. wait a few microseconds for all the lights to detect this P transition;
3. enable (goes to 0) the GP4 inhibit output (through T2, which forces the following module's GP3 input to 0 V);
4. wait for two successive P/N transitions on GP3;
5. disable the GP4 inhibit output (goes to 5 V);
6. count the number of N/P transitions;
7. subtract this number from 63 to obtain the address.

If the end-of-string connector is connected back to the master module, the master too can count the N/P transitions on its CCP2 pin and in this way count how many individual lights the string has. It then stores this number in the PIC's internal EEPROM and will use it to drive the light-string sequences correctly. If the connector is not connected, the master counts zero lights, and in this event uses the value stored in its EEPROM.

Control signal

After the string has been initialized, the master module starts the sequencing. As the master only has one wire to carry its quite complex control signals, the communication protocol is also a little complicated.
The master can transmit over 1,000 words a second to the lights. A word is coded using seven bits. The value of the first bit indicates if the following six bits are an address (bit 1 = 0) or a command (bit 1 = 1). Each word is separated from the previous one by an 'End' marker. This marker makes it possible to re-synchronise any lights that might have got out of sync with the control signal.
Single-wire transmission is achieved by an asynchronous serial signal using a proprietary protocol. To transmit a '1' bit, the master module sets the control signal to 0 V for 30 µs, then to 5 V for 58.3 µs. To transmit a '0' bit, the master reverses these timings, setting the signal to 0 V for 58.3 µs, then to 5 V for 30 µs. To transmit the 'End' marker, the master sets the control signal to 0 V for 160 µs, then to 5 V for 20 µs. Hence the total period for transmission of an address or command word is 798 µs.

In order to decode this word, the individual light synchronise themselves by waiting for an N-to-P transition on their GP3 inputs. They then measure the duration of the P state using their T0 timers and deduce from this whether the master sent a 0 or a 1.

Address word

The A word (Address) enables the master to

Table 1. Command word values and functions						
5	**4**	**3**	**2**	**1**	**0**	**Operation**
0	0	0	0	0	0	Set the component read pointer to memory 0
0	0	0	0	0	1	Set the component write pointer to memory 0
0	0	0	0	1	0	Set the component read pointer to memory 2
0	0	0	0	1	1	Set the component write pointer to memory 2
0	0	0	1	0	0	Increment the component read pointer
0	0	0	1	0	1	Increment the component write pointer
0	0	I3	I2	I1	I0	(I3 to I0) – 6 = overall intensity setting (0–9)
0	1	I3	I2	I1	I0	I3 to I0 = red component intensity setting (0–15)
1	0	I3	I2	I1	I0	I3 to I0 = green component intensity setting (0–15)
1	1	I3	I2	I1	I0	I3 to I0 = blue component intensity setting (0–15)

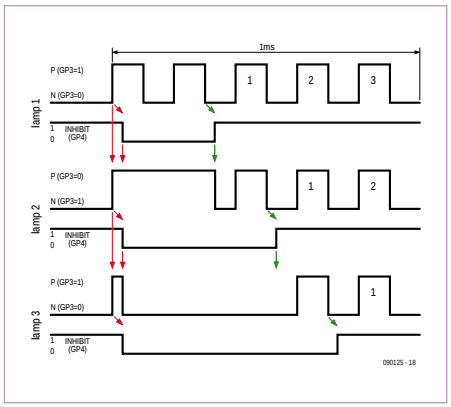


Figure 6. Timing diagram for the counting stage that enables each light to determine its own address.

address up to 62 individual lights. The first light connected directly to the master has the address 1, the next has the address 2, and so on. Addresses 0 and 63 have special functions. Address 0 is used as a 'neutral' word that doesn't change the state of the lights. This word enables the master to provide a clock to the lights, which need this to be able to light up at the required brightness, even when there are no command or address words. Address 63 allows the master to address all the lights at the same time (this is a broadcast address).

As soon as a light has decoded its own address, it ignores any other addresses that follow immediately, and then executes all command words that arrive until the next address word is received.

Command words

Each individual light has four memory locations and two pointers that address them. Each memory location stores a set of four intensities, three for the RGB components and one for the overall component. These locations are addressed by write and read pointers. The master can move these pointers at will from one memory to the next, via six special commands (**Table 1**).

Four other commands enable the master to write an intensity value (from 0 to 15) for the RGB components and an intensity value (from 0 to 9) for the overall compo-

Table 2: R, G, or B component generation table.																															
BRI	COUNTER																														LEVEL
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	
0																																	0 %
1	1																1															6 %
2							1										1											1					9 %
3					1								1								1								1				13 %
4					1							1						1						1							1		16 %
5				1					1					1						1					1					1			19 %
6	1					1				1					1				1					1				1					22 %
7			1				1				1				1				1				1				1				1		25 %
8			1			1			1			1			1				1			1			1			1			1		31 %
9		1			1			1		1			1			1		1			1			1		1			1			1	38 %
10		1		1		1			1		1		1			1		1		1		1			1		1		1			1	44 %
11		1		1		1		1		1		1		1		1		1		1		1		1		1		1		1		1	50 %
12	1	1		1		1		1	1	1		1		1		1	1	1		1		1		1	1	1		1		1		1	63 %
13	1		1	1	1		1	1	1		1	1	1		1	1	1		1	1	1		1	1	1		1	1	1		1	1	75 %
14		1	1	1	1	1	1	1		1	1	1	1	1	1	1		1	1	1	1	1	1	1		1	1	1	1	1	1	1	84 %
15	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	100 %

nent into the memory addressed by the write pointer.
The master moves the read pointer ('read' from the point of view of the individual lights) over a set of data it has previously written. In so doing, it selects this set as the illumination set-point for the light. By using the broadcast address 63, the master can change the illumination set-points of all the lights in the string at once, producing a simultaneous effect.
For example, to make the lights all light up red at the same time, the master can send the following words one after the other (assuming that at the outset, the individual lights' read and write pointers are at the first memory location 0):

A-3F: address word 0x3F, 63 in decimal, broadcast address,
C-05: command word, moves write pointers to location 1,
C-0F: sets overall components to maximum (9),
C-1F: sets red (R) components to maximum (15),
C-20: sets green (G) components to 0 (unlit),
C-30: sets blue (B) components to 0 (unlit),
C-04: moves the read pointers to location 1 (illuminated red).

Applying the illumination drives

The individual light modules generate the illumination drives by altering the chosen lit/unlit duty cycles of the LED elements. This generation takes place periodically in two cycles:

1. a 25.5 ms cycle for the RGB components and
2. a 798 µs cycle for the overall component.

RGB component generation cycle

Systematically every 798 µs, a light receives a command or address word from the master. Each time, it increases a counter that runs in a loop from 0 to 31. This counter, with the intensity set-point to be applied, enables it to step through a component illumination table (**Table 2**). If the table contains a 1, the light lights up the component by setting a 0 on the relevant GP0, GP1, or GP2 output. If the table contains a 0, the light extinguishes the component by setting the output to 1.

As this counter can take 32 values, the intensity set-point generation cycle has a period of 32 × 798 µs = 25.5 ms. As this duration is longer than the eye's 20 ms persistence of vision, this could cause a slight impression of twinkling. A number of choices have been made to reduce this effect:

- a minimum of two illuminations are carried out during the generation cycle and
- these illuminations are positioned in a specific way within the cycle.

You will notice that the intensity set-point is not exactly proportional to the element's illumination level during the cycle. This lets us compensate for the Weber-Fechner law [4] (stating that "the sensation varies as the logarithm of the stimulus") and allows our eyes to perceive an intensity that is substantially proportional to the set-point.

Overall component generation cycle

The successive operations (bit decoding, command execution) an individual light performs during the 798 µs word transmission cycle are broken down into nine segments by special processing devoted to generating the intensity of the overall component (**Figure 5**). At the start of the cycle, the light resets a counter to 0. At each step of the processing, the light sets the PIC's GP5 output to 0 (lit) if the counter value is lower than the intensity set-point, or 1 if it is higher. Each time, the light increments the counter by 1.

Combining the components

Combining the RGB and overall components lets us adjust the brightness over a wide range. This is particularly useful where a colour is obtained from a mixture of two components, as in the case of orange, obtained by mixing the red and green components. A good orange colour is obtained by setting the RGB components to 15, 6, and 0 respec-

COMPONENT LIST

Master module

Resistors
R1 = 1kΩ
R2,R3 = 2.2kΩ
R4,R5 = 100Ω
R6 = 22kΩ
R7 = 10kΩ

Capacitors
C1 = 1000µF 35V, radial, lead pitch 5.08mm (0.2 in.)
C2,C3,C4,C7,C8 = 100nF
C5,C6 = 22pF
C9 = 10µF 35V, radial, lead pitch 5.08mm (0.2 in)

Semiconductors
D1, D2 = LED, 3mm (see text)
D3, D4 = 1N4148
IC1 = 7805
IC2 = 24FC1025-I/P (Microchip) (see text)
IC3 = PIC18F2550 (Microchip)

Miscellaneous
K1 = 2-way terminal block, lead pitch 5.08mm (0.2 in.)
K2, K3 = 5-way SIL pinheader, lead pitch 5.08mm (0.2 in.)
K4, K5 = 2-way terminal block, lead pitch 5.08mm (0.2 in.)
K6 = USB-B socket for IC (see text)
K7 = 4-way SIL pinheader, lead pitch 5.08mm (0.2 in.)
S1 = pushbutton, 1 make contact (see text)
X1 = 20MHz quartz crystal, HC49/U case
PCB, ref. 090125-1 [1]

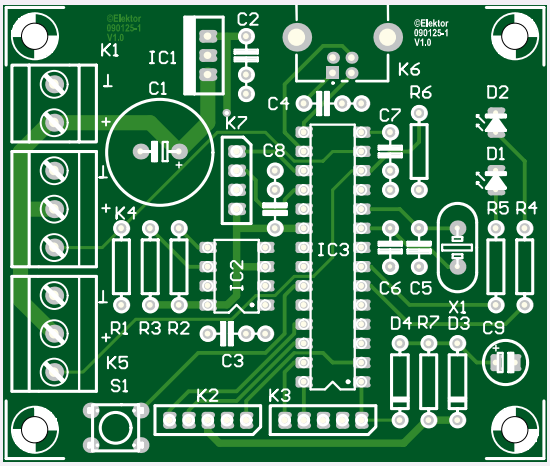
Miniature lamp (each)

Resistors (SMD 1206)
R1 = 2.2kΩ
R2 = 100Ω
R3 = 4.7kΩ
R4, R5 = 47Ω
R6 = 47kΩ

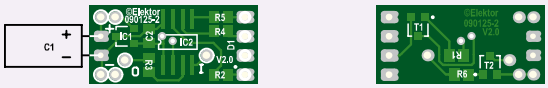
Capacitors
C1 = 220µF 25V radial, lead pitch 2.54mm (0.1 in.)
C2 = 100nF (SMD 1206)

Semiconductors
D1 = LED, RGB, common anode
IC1 = TS78L05CX, SOT-23 case
IC2 = PIC12F508-I/SN (Microchip, SOIC-150)
T1 = BC857, SOT-23 case
T2 = BC847, SOT-23 case

Miscellaneous
PCB, ref. 090125-2 [1]



Master module component layout.



Slave module layout, component side (left) and soldering side (right).

tively. By reducing these values moderately and proportionately (for example, to 10, 4, and 0), we can obtain a lower intensity without changing the orange colour too much.

However, it becomes hard to reduce the intensity still further, as this would lead to a more drastic change in the colour. The ratio between the perceived luminous intensities of the R and G components would depart too far from the initial value that gave us the orange colour. To obtain a greater intensity reduction, it's better to act upon the overall component.

Just to finish off...

After reading the rather detailed description of the software, you may be feeling like a bit of a change. Well, make the most of that to wire up the lights — you've got another 62 to go! Warning: the master module software published on the site works fine with around 30 lights, but has not been tested with a greater number — the limit will be related to the maximum rate at which commands can be sent out from the master module. And while you're trying to solder the SMD components, you'll be able to have a think about other applications for this dynamically-addressed 'single-wire' network (using four wires).

Send us your suggestions, and photos or videos of your fairy-lights, and we'll publish the best of them in a future issue and/or on our website.

Happy Christmas!

(090125-I)

Internet links

[1] www.elektor.com/090125
[2] www.microchip.com/stellent/idcplg?IdcService=SS_GET_PAGE&nodeId=1406&dDocName=en010014
[3] www.bknd.com/cc5x/index.shtml
[4] en.wikipedia.org/wiki/Weber%E2%80%93Fechner_law