Colourful Computer Light

Controlling the Living Colors lamp with USB

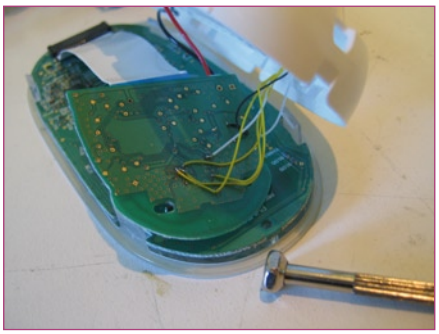
Jeroen Domburg

We live in a colourful environment these days. Everything is in colour: TV, advertising billboards, mobile phone displays and LEDs. Philips added a further dimension to all this with their Ambilight, Wake-up Light and Living Colors lamp. We will work with the latter in this M&T article. The wireless remote control offers interesting possibilities once the protocol has been cracked...

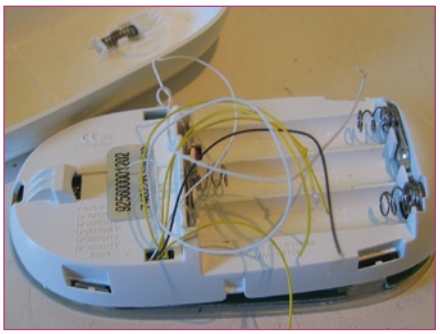
In the February 2008 issue the editors disassembled a Living Colors lamp from Philips. In this article we will once again do something with this lamp. One of the disadvantages of the lamp is that it can only be controlled with the supplied remote control. Nice enough perhaps, if all you want to do is use it as a glorified table lamp. But controlling it with a PC offers many other possibilities. Turn the room red when you've received mail, let the colour of the wall follow the movie you're watching, illuminate the room when it's time to get up, you mention it!

Lively colours

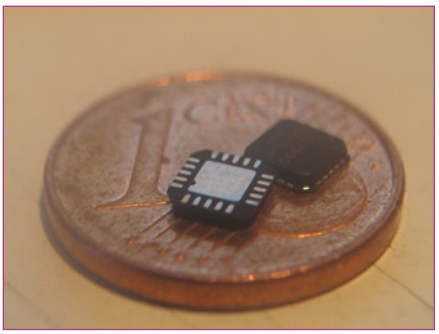
For those who missed the article mentioned earlier: a Living Colors lamp is an appliance made by Philips that with a few bright, coloured LEDs can illuminate a room in just about any conceivable colour. In this way you can create or enhance a particular mood. The Living Colors lamp comprises the lamp itself and a remote control. The two are linked via a CC2500, a little IC from Texas Instruments, which can send data over a 2.4 GHz radio link.
To be able to control the lamp we will first have to figure out how the data is sent. Measuring this without open-



In the remote control we find two printed circuit boards that are interconnected with a ribbon cable.



This is how we reverse engineer the protocol. It may not look like it, but apart from the wires the remote control is still completely intact.



The all-important CC2500 chip. They don't come any bigger than this, unfortunately...

ing the device is difficult. Firstly, because the CC2500 has several methods available for sending the data (MSK, FSK, OOK, with or without data whitening, Manchester-encoded, etc.) so it is a lousy job trying to dig out the transmitted data from the actual radio signal transmitted. Secondly, the author, in contrast to the RF people at the editorial offices, does not have SHF measuring equipment at his disposal, something that's crucial with this approach. We will therefore have to decode the information using some other method...

Eavesdropping

Taking a look at the datasheet for the CC2500, we read that the chip gets its data from the host-processor via a 4-wire serial connection, with the option of two more wires for status information. If we eavesdrop on the traffic on this 4-wire bus we should be able to learn a whole lot more about what is being transmitted.

Although there are two CC2500s, namely one in the remote control and one in the lamp itself, we decided to listen only to the one in the remote control. The reason for this is less philosophical than you may think: it proved to be impossible to open the lamp without damaging it, but it turned out that opening the remote control was a lot easier. The remote control consists of two printed circuit boards. The PCB for the touch sensitive 'push buttons' plus the controller for these, a QT1106 is connected with a ribbon cable to the smaller main PCB that contains the MSP430 processor and the CC2500.

Tapping into the bus is rather difficult, but with the aid of thin wire and some instant glue it was eventually possible to make a mechanically strong *tap*.

Because interpreting the protocol using only and oscilloscope is rather tedious, we use an AVR with hardware SPI support for the actual 'sniffing'. This AVR then sends the eavesdropped signal via a serial port to the PC where the actual decoding can begin.

When we push a few buttons on the remote control, it is immediately clear that the protocol is more complicated than we had initially anticipated. When the remote control is first turned on, The CC2500 is initialised with data regarding the frequency, the type of modulation and the data rate. The actual communication is based on packets. A packet is loaded into the CC2500 and transmitted by the chip in RF form. Reception is done in the same way. The CC2500 is set to receive mode and as soon as a packed has been received a particular pin goes high and the packet can be read by the microcontroller.

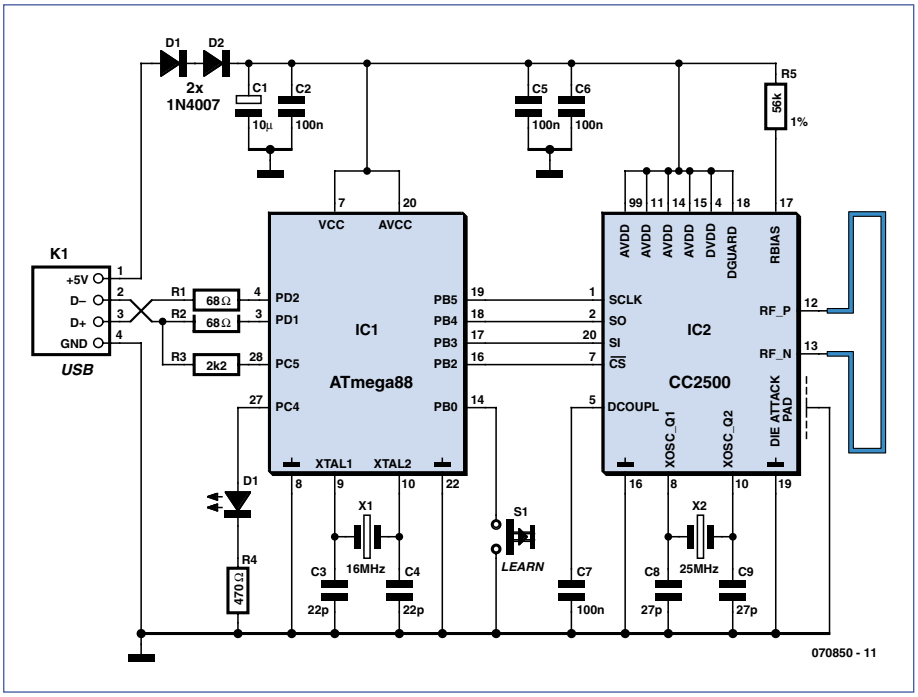


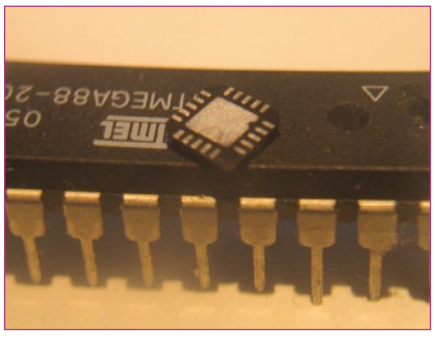
Figure 1. It can't be much simpler than this: we do the control of the CC2500 with an ATmega88 via USB.

Data format

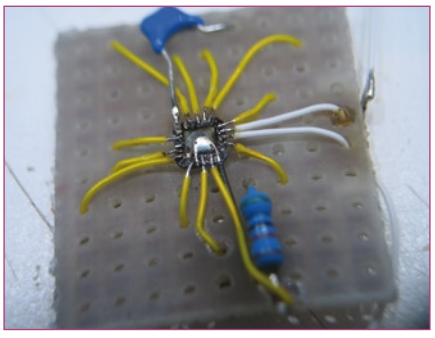
The packets consist of a number of fields. The first thing that emerges is that both the remote control and the lamp have a, probably unique, address. Therefore, the packets for setting the colour, for example, start with the address of the lamp followed by the command.

The commands correspond with the buttons on the remote control. There is, among others, a command to turn the lamp on, to turn it off again, to set the colour and to set the lamp in demo mode.

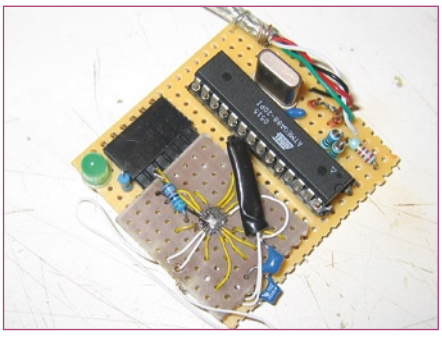
The command is followed by a se-



In comparison with the ATmega88 the latter is indeed quite 'mega'.



The size does not make it impossible to solder. A steady hand, a magnifier and thin wire go a long way.



This is how the transmitter is mounted on the USB PCB. The thing with black tape around it is a 25-MHz SMD crystal.



quence number. This is a number that increments by one after each command is sent. When the lamp sends a response, this same sequence number is sent back so that the remote control can determine which response goes with which command.
It gets more interesting after the sequence number byte. There now follow three bytes with colour information. The fact that colour information is being sent is somewhat remarkable, since the average remote control only passes on which button is being pushed. The decision to store the selected colour in the remote control makes sense. In this way Philips ensures that if you use the remote control with multiple lamps they will all be set to the same colour. For our purposes this is also very practical: it is, after all, much easier to sent the desired colour than to emulate all sorts of button pushing.
To send the colour, Philips decided to use the HSV system. The Hue gives the colour, the Saturation the intensity of that colour and Value the amount of light the lamp has to generate. By giving the appropriate command with certain HSV-values the desired colour can be set immediately. And because the wireless connection operates at a speed of 500 kbaud, this is relatively quick as well.

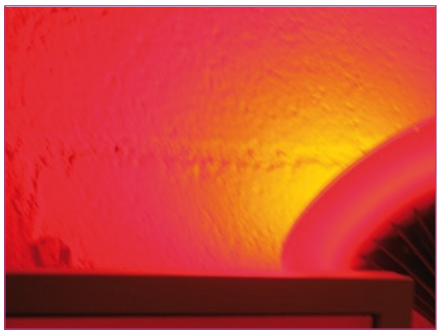
Control

Okay, we have the protocol, we have the initialisation data and we know how we can set the colour of the lamp. What are we now going to do with that knowledge? The author decided that an Ambilight-ish functionality would be nice to do. The plan therefore, was to build a device that could be connected to the PC and control several lamps.
For the control we can use existing software: on the internet there is a community of people who make their own PC controlled Ambilight clones. This has resulted in a few nice Linux and Windows applications that are very useful for this project. The most common protocol used in this software is the MoMoLight protocol, which is actually nothing more than sending the RGB values for three different light sources directly to the serial port.
To be compatible with the software we need a few things. Firstly we'll have to emulate a serial port over the USB bus and secondly we'll have to convert the incoming RGB data to the HSV format that's expected by the lamps.
The first requirement is easily met with one of several ready-made solutions: a number of companies make USB-to-RS232 converter ICs that can be directly connected to the bus. For this project however, we chose a different approach. The heart of the circuit consists of an ATmega88 which is connected directly to the USB port. If we look at the datasheet for this AVR we will however not find any mention of hardware to support USB. So how does this work then?
The solution is to be found in a trick: with some clever programming most of the AVRs can be made to 'mimic' a low-speed USB device. There even exist special libraries for this purpose [1]. Several projects have been made around these libraries: USB-programmers, bootloaders, display controller, just name it. One of these projects is called AVR-CDC and its purpose is to implement a USB to serial converter in software. That's just what we need! The software is licensed under the GPL, which means that if you build a device using it, you also have to supply the source code. That is not a problem for this project.
An RGB to HSV converter is also easily picked from the Internet. There are multiple solutions on various websites, but they are often based on floating point, which means that the already busy AVR has to do even more. After an extensive search we fortunately also found an integer version, which costs far fewer clock ticks. This software is released under the MIT license, which, after a little searching, appears to be compatible with the GPL. So after a copy-paste operation we've already gathered half of the required code.
The code to control the wireless chip is all that remains. Because this chip has a comprehensive datasheet and we have a good example obtained by eavesdropping on the data from the remote control, this is not a big deal.

Hardware

Because we've solved a number of requirements in software, the circuit that remains is not tricky at all (**Figure 1**). On the left is the USB connection, which is connected with a few, and according to the USB specification, mandatory resistors to the AVR. The CS2500 and the USB data lines require a power supply voltage of 3.3 to 3.6 V. This is obtained in a simple way from the 5 V on the USB connector. Connect two diodes in series with this 5 V and the voltage drops to about 3.5 V.
On the right of the schematic is the CC2500, in a configuration which is nearly entirely a direct copy from the datasheet. The loop between RP_P and RP_N is the antenna. Although there are quite specific requirements for this antenna in the datasheet, a wire about 11 cm long and bent into the shape indicated suffices in practice and works well over a short distance.
The schematic looks quite simple, but the assembly of the circuit is much trickier than it looks. This is because the CC2500 chip, which deals with the necessary RF communication, is only available in a QFN package. For those that are not familiar with SMD pack-

And the end result: the Living Colors lamps



ages: the five pins on each side of this tiny chip all fit between two pins of a normal DIP package. As if that is not bad enough, most of the 20 connections to the IC have to be actually connected as well. How do we solve this as hobbyist without access to an expensive SMD equipped workshop?
Of course, there are conversion PCBs available, but they are generally quite expensive and certainly the versions for QFN are not readily available. The author therefore chose for the 'dead bug' method: the chip is glued upside down with a drop of instant glue to a small piece of prototyping board. The connections are now made with thin wire to the copper tracks of the prototyping board. This type of wire is sold with the name Kynar- or wirewrap wire, but a cheaper alternative is salvaging an 80-way IDE cable; the individual wires are about the same size.
Once the module with the CC2500 is done, the remainder is not too much trouble. That is because these are all through-hole parts. In the end the diligent effort results in a little PCB about the size of a match box, with the USB connector as its only connection.

Compatibility problems

All that is left to do is plugging in the connector and testing of the assembly. The first tests appear to go really well, but several colours look absolutely nothing like those on the screen. How can this be? A quick test with a graphics program that can generate

in use as an Ambilight clone.

HSV colours indicates that the HSV-to-RGB conversion in the lamps does not follow the official standard entirely. Although the saturation and value are correct, there is a certain non-linearity in the hue curve. Fortunately this can be fixed. After a few observations of the differences in colour, a table can be constructed which converts 'real' hue-values to their equivalent Living Colors hue values. The table is not really an ideal solution, but if you notice the colour differences when watching a movie you will have to ask yourself whether that movie is really worth your time...
Because there is little chance that other lamps have the same addresses as the lamp we used, there is a learning routine in the AVR. This works as follows. First make sure that all lamps that have to be controlled can be operated with one remote control. You can 'add' a lamp to a remote control by holding the remote against the Philips logo on the front and pushing the '1'-button on the remote. Do this for all the lamps and if all is well, all lamps will now react to that remote control.
Once the remote control knows all the lamps it is possible to transfer the addresses to the AVR: push button S1 and press the '0' button on the remote control until the LED on the PCB (D1) turns off. What is happening? The remote control attempts to turn off all the lamps by sending each lamp the 'off' command. The AVR also listens on this channel and stores every passing address. These addresses are saved in EEPROM. 'Acquired' addresses remain in the AVR until replaced by other ones after the learn-button is pressed again. The addresses are also retained when the power supply voltage is removed.

The last mile

How does all this work on the PC side? As already mentioned, the AVR presents itself as a serial port that understands the so-called MoMoLight protocol. This means that any program that supports this protocol can control the Living Colors lamps. A few examples of these are, just like the firmware for the Atmel, on the website of the author [2] and on the project page at www.elektor.com.
For programmers who would like to write their own software: the MoMoLight protocol supports up to three RGB light sources. To set the lamps to the desired colour the emulated serial port needs to be opened at a baud rate of 4800, no parity and 8 data bits. The RGB values for the lamps can now be sent in nine bytes in the order of R1,R2,R3,G1,G2,G3,B1,B2,B3.
A final remark: it has come to the author's attention that the software USB stack is not quite as compatible with all computers as it should have been. Should there be a problem with a particular PC, you can try to connect the device via a USB2.0 hub to the PC. If this is all to no avail then there is also a serial version available on the author's website.

(070850-I)

Web Links

[1]: www.obdev.at/products/avrusb/index.html

[2]: http://meuk.spritesserver.nl/projects/livcol

About the author:

Jeroen Domburg is a student at the Saxion Technical University in Enschede, the Netherlands.

He is an enthusiastic hobbyist, with interests in microcontrollers, electronics and computers.

In this column he showcases his personal handiwork, modifications and other interesting circuits, which do not necessarily have to be useful. In most cases they are not likely to win a beauty contest and safety is generally taken with a pinch of salt. But that doesn't concern the author at all. As long as the circuit does what it was intended for then all is well. You have been warned!