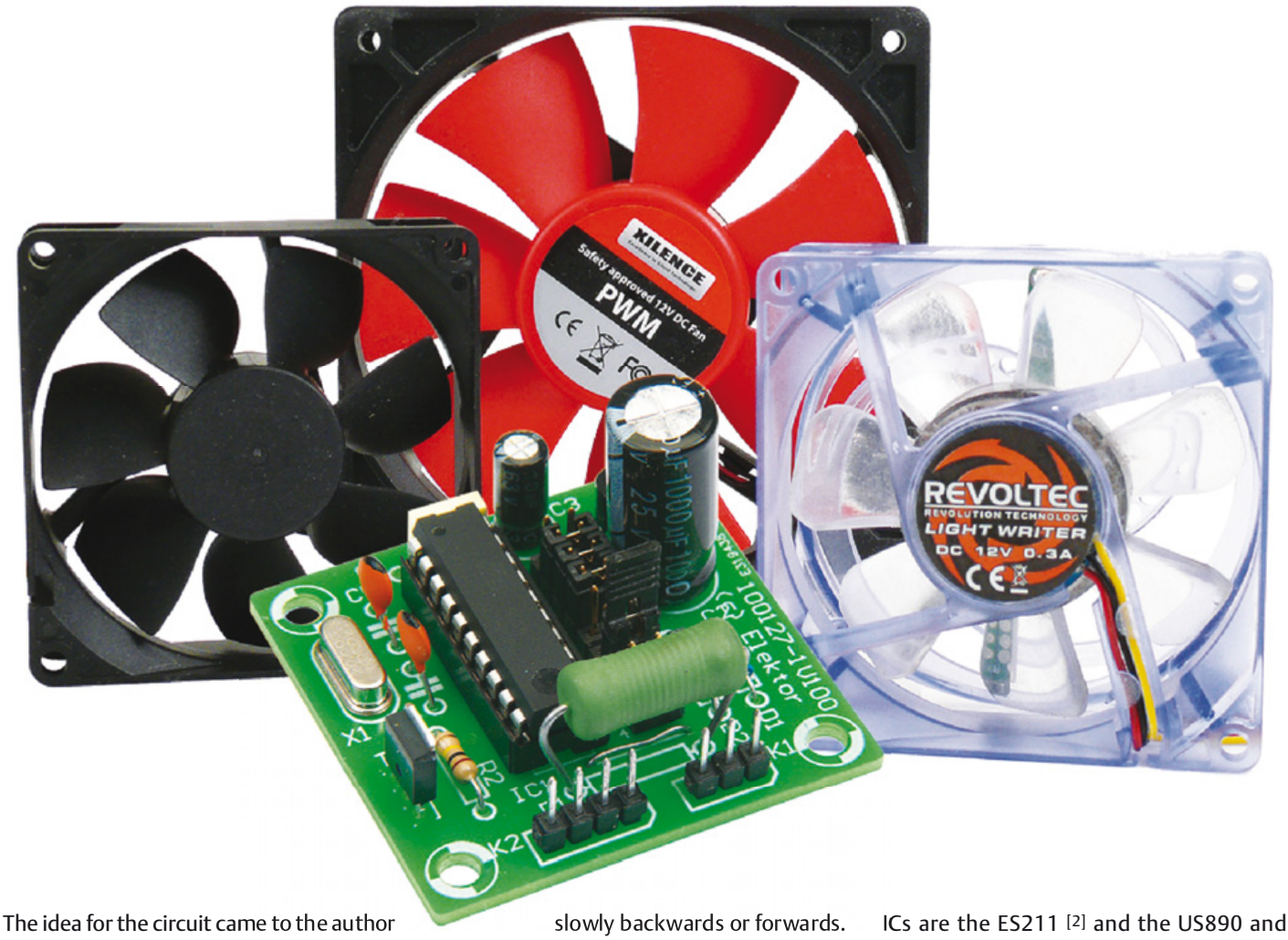
Stroboscopic PC Fan

No flash in the pan

A PC case with a side window gives plenty of opportunities for pimping the machine out with visual effects. Stand out from the crowd with the fan stroboscope circuit described here: based on an Atmel microcontroller, it drives an LED in such a way as to make the vanes of the fan seem to stop, turn slowly backwards or forwards, or suddenly change position.

By Martin Ossmann (Germany)



The idea for the circuit came to the author when his son bought a new computer. The case had a window in the side through which the gleaming components could be admired, but something was missing. An eye-catching centrepiece to the machine was needed, and the idea soon evolved into a plan to illuminate the CPU fan with a stroboscope using power LEDs. By flashing the LEDs with the right timing, it would be possible to make the fan appear to stop or turn slowly backwards or forwards.

The author's YouTube channel has videos showing how effective the illusion is: search for 'Ossimodding' [1].

All about fans

PC fans come in all shapes, sizes and colours. **Figure 1** shows the construction of the simplest modern design. Drive is by a brushless DC motor with two windings controlled by a dedicated IC. Typical driver ICs are the ES211 [2] and the US890 and US891 [3], which also include a Hall sensor to detect the position of the rotor.

To get a proper stroboscopic effect, the flashes of light illuminating the fan must be synchronised with its rotation. For PC fans with a three-wire [4] or four-wire [5] connection, this is relatively easy, as they have a special output (called the 'tacho' output) that delivers typically two pulses per revolution. Some fans deviate from this 'Intel

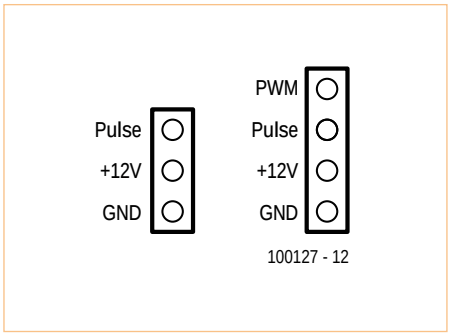


Figure 2. Pinouts of three- and four-pin fan connectors.

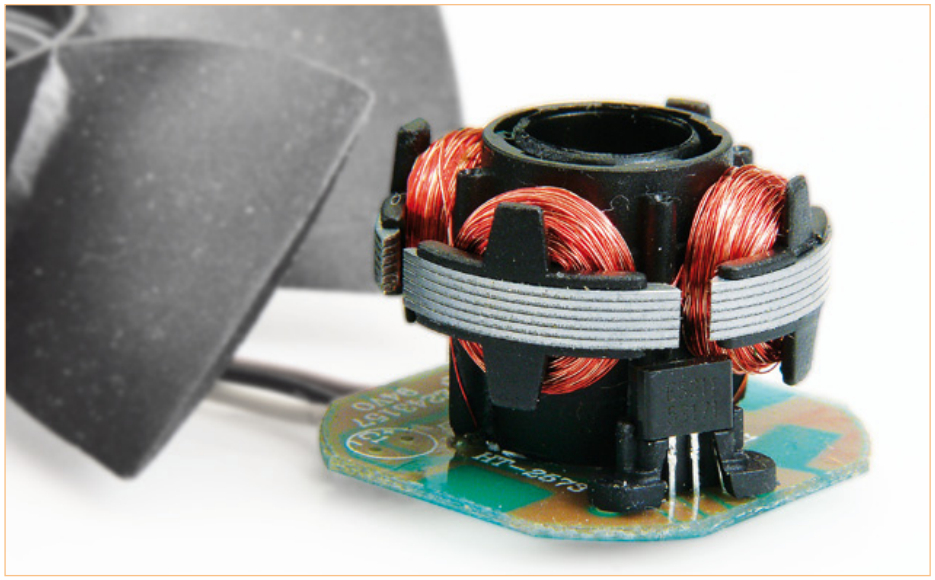


Figure 1. The secrets of a fan: two coils and a dedicated control IC.

standard' and deliver three pulses per revolution; the necessary software modification to deal with this is given in the download for this project.

Figure 2 shows the pinouts of the three- and four-pin connectors normally used, in each case an extension of the classical two-pin design providing just +12 V power and ground. The three-pin connector adds the tacho signal, which is an open-collector output from the fan. The PC motherboard will normally include a pull-up resistor to the +12 V rail, and this is taken into account in the design of our circuit.

The four-pin connector adds a PWM input to the fan controller. A TTL-level PWM signal provided on this input, nominally at 25 kHz, allows the fan speed to be controlled. If the input is left open-circuit, a pull-up resistor inside the fan ensures that it will run at maximum speed.

The circuit

Using a microcontroller we can read the tacho signal and use its timing to control one or more power LEDs accordingly. Driving the LEDs is best done via a logic-level MOSFET. The whole circuit is shown in **Figure 3**.

The clock for the microcontroller is provided by a 20 MHz crystal. A row of nine jumpers (JP1) allows the various options offered by the software to be configured: see the **Table**. The first four jumpers (on port pins PB4 to PB7) are used to tell the microcontroller the number of blades on the fan, from 0 to 15. The next three jumpers (on port pins PB0, PB1 and PB3) select one of eight different modes of operation. The jumper on PD6 controls the amplitude of the apparent fan movements, and the jumper on PD3 their speed.

Power is obtained from the +12 V line on the fan connector. The easiest way to connect the circuit to the fan is to make a small adaptor (**Figure 4**) which passes through the four fan signals and brings out the GND, +12 V and tacho wires for connection to K1 on our printed circuit board.

Construction and LEDs

Although the circuit only consists of a few components, we have designed a small printed circuit board (**Figure 5**) in the *Elektor* lab for the project to make the project

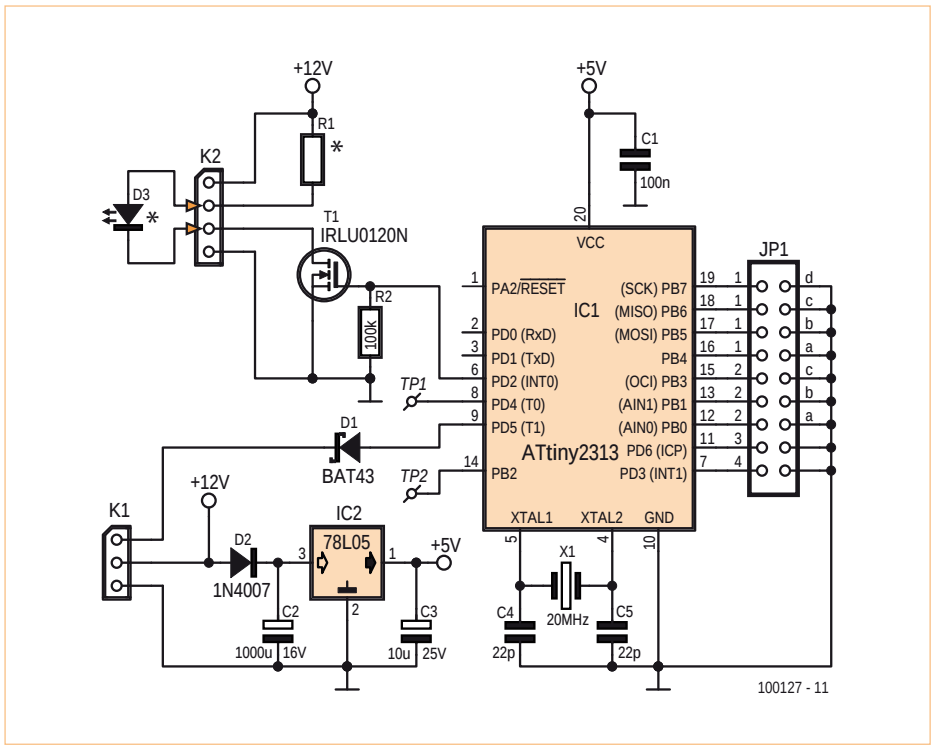


Figure 3. Circuit diagram: the jumpers allow various effects to be selected.

Table: Jumper settings

(PB7)	(PB6)	(PB5)	(PB4)	Blade count
1	1	1	1	0
1	1	1	0	1
1	1	0	1	2
1	1	0	0	3
.	.	.	.	.
.	.	.	.	.
.	.	.	.	.
0	0	0	0	15

(PB7)	(PB6)	(PB5)	Option N°.	Function
1	1	1	0	Stationary
1	1	0	1	Combination of various movements
1	0	1	2	Oscillation
1	0	0	3	Forwards and backwards
0	1	1	4	Movements with pauses
0	1	0	5	Sudden movements
0	0	1	6	Slow movement
0	0	0	7	Stationary

PD3 = select speed of effect

PD6 = select amplitude of effect

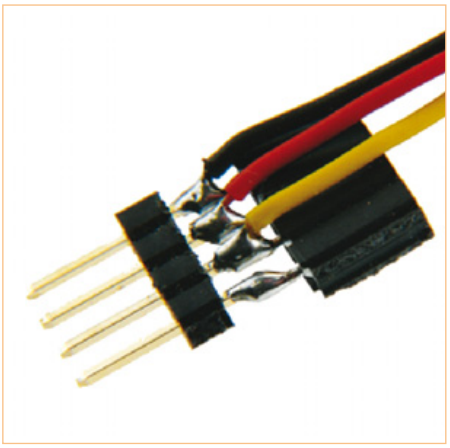


Figure 4. Adaptor for connecting to a fan cable.

Software variants

Version using ATtiny2313:

Source file: fan_flash_2313_v01.c
Compiler WINAVR GCC
Compileroption: -O2
Fuses: external crystal
brown_out at 4.3 V
no CKDIV8

Version using ATtiny25/45:

Source file: fan_flash_45_v01.c
Compiler WINAVR GCC
Compileroption: -O2
Fuses: external crystal
brown_out at 4.3 V
no CKDIV8

COMPONENT LIST

Resistors
R1 = see text
R2 = 100kΩ

Capacitors
C1 = 100nF
C2 = 1000µF 16V
C3 = 10µF 10V
C4,C5 = 22pF

Semiconductors
D1 = BAT43
D2 = 1N4007
D3 = power LED (see text)
IC1 = ATTiny2313-20PU, programmed, Elektor # 100127-41
IC2 = 78L05
T1 = IRLU120N (International Rectifier)

Miscellaneous
K1 = 3-pin pinheader
K2 = 4- pin pinheader
JP1= 18-pin (2x9) DIL pinheader with Jumper
X1 = 20MHz quartz crystal
PCB # 100127-1 (www.elektor.com/100127)

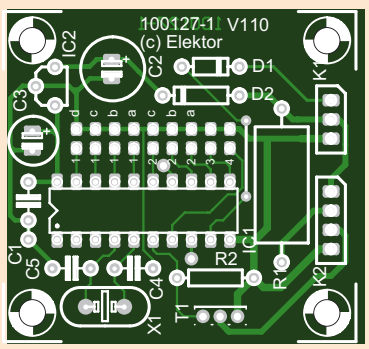


Figure 5. The printed circuit board designed in the *Elektor* lab.

easier for beginners to tackle. The software for the microcontroller (including source code) is available, as ever, for free download from the project web pages [6]. Compiler options and fuse settings are listed in the text box 'Software variants'. If you are not able or not inclined to program the microcontroller yourself, ready-programmed microcontrollers can be purchased, again via the project web pages. The device then just needs to be plugged (the right way around!) into its socket on the board: see **Figure 6**.

As shown in the circuit diagram, an LED is connected between pins 2 and 3 of K2. The LED only lights very briefly, and so needs to be driven at a high current to give bright enough results. Power LEDs rated at up to 5 W are ideal. The value of series resistor R2 will depend on the type of LED chosen. In our prototype we used a value of 5 Ω; with a normal LED a value of 50 Ω would be more suitable. It is also possible to wire several LEDs with individual current-limiting resistors in parallel, as the MOSFET has plenty of drive capability (I_{Dmax} = 10 A).

Software

The program running in the ATtiny2313 configures timer 0 so that it generates an interrupt every 200 clocks. The interrupt rate is thus 100 kHz, and all the important functions of the system are carried out in the service routine. The routine monitors

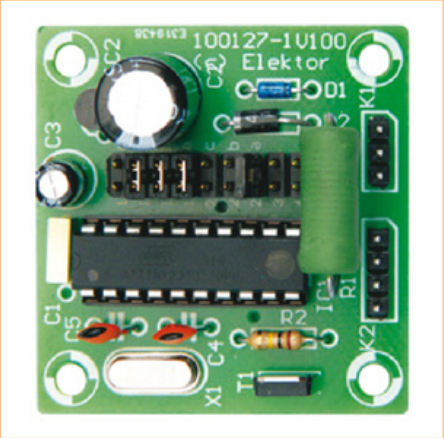


Figure 6. The assembled prototype.

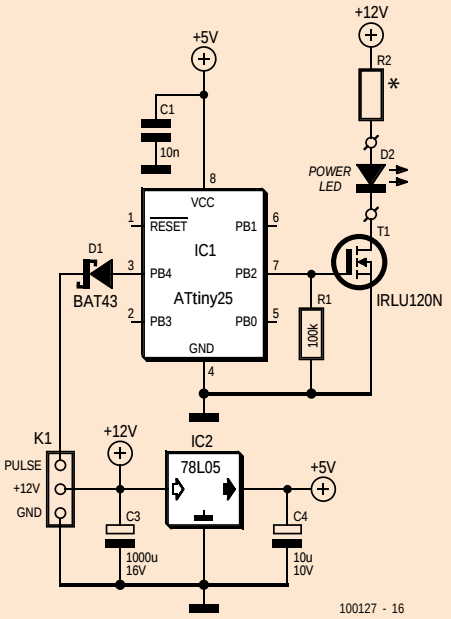
Minimalist's version

If the ability to configure the circuit is not required, and the settings can be fixed in the software, then it is possible to dispense with the jumpers.

Furthermore, if the internal RC oscillator in the microcontroller is used, the crystal can be dispensed with too.

The result is a minimalist's version that can run in an ATtiny25 or ATtiny45.

The software can again be downloaded from the *Elektor* website [6]: see the text box on software variants. In view of the small number of components, it is easy to build this version on a small piece of prototyping board.



the tacho signal from the fan to detect edges. Two rising edges correspond to one revolution of the fan. **Figure 7** shows the timing of events during each revolution. The variable slowTimer is incremented by one on each interrupt and reset to 0 at the start of each complete revolution. The changing value of this variable can be thought of as a sawtooth wave synchronised with the rotation of the fan, and with a maximum value dependent on its speed. For example, if the fan is turning at 1500 rpm each revolution takes 20 ms, or 2000 interrupts. For maximum brightness, we would like to flash the LED once as each blade of the fan goes by, and to this end we generate another sawtooth variable, called fastTimer. This variable is reset at the same time as slowTimer, but counts up only to a limit called fastPeriod, which is equal to the maximum value reached by slowTimer divided by the number of blades of the fan. The result is that fastTimer varies as a sawtooth synchronised with the blades of the fan.

It is also necessary to be able to adjust the relative phase of the LED flash and the fan blade position, which is done using a further fan-blade-synchronous sawtooth variable called PLLtimer. The LED is flashed each time this variable is reset. The period of the variable is equal to that of fastTimer and a simple software PLL controls the phase shift between the two variables. On each falling edge of PLLtimer (at the time indicated by the dashed line) the value of fastTimer is compared against a reference value. If the edge is to the left or right of the desired position the phase of PLLtimer is corrected by slightly increasing or decreasing the length of its next period. The **Listing** 'PLL code' shows the relevant part of the program.

The period of PLLtimer at any time is given by fastPeriod plus PLLcontrol. Here fastPeriod is the nominal period of the variable and PLLcontrol is the correction to cause a phase shift. When PLLtimer reaches its maximum value it is reset and the LED is

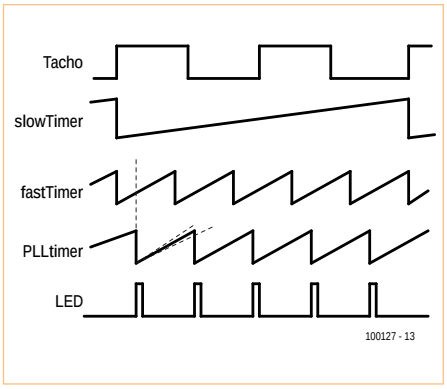


Figure 7. Timing diagram for the rotation of the fan.

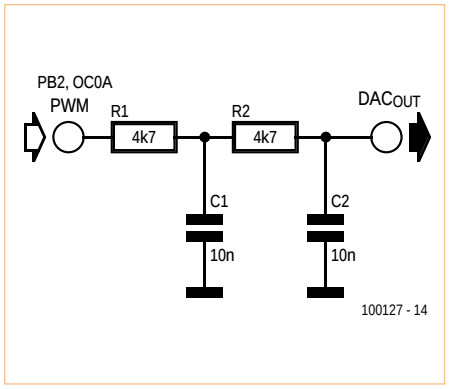


Figure 8. Low-pass filter suitable for converting the PWM test signal output on PB2 of the microcontroller into a sawtooth wave.

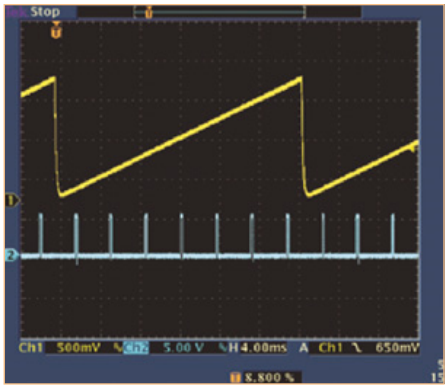


Figure 9. Oscilloscope traces showing the sawtooth waveform (filtered PWM signal) and the LED drive pulse.

flashed. At the same time the phase difference between wantedPhase and fastTimer is calculated in order to generate a new PLL-control value.

Debugging

If you are planning to make any modifications to the software, it will be helpful to have a debugging facility: real-time debugging is a must, as most bugs will lead to timing errors. To simplify this, pin 14 of the ATtiny2313 (TP2) outputs the current fan position in the form of a PWM signal. This can be filtered using the low-pass circuit shown in **Figure 8** and the result shown as a sawtooth signal on an oscilloscope. The second channel of the oscilloscope can be used to monitor the gate signal to the MOSFET to see where the light pulses are being generated. **Figure 9** shows a typical picture obtained using this set-up.

(100127)

Internet Links

[1] www.youtube.com/user/ossimodding
[2] www.eastera.com.cn/data/ES211-ENb_a.pdf
[3] www.melexis.com/Assets/US890US891_DataSheet_4851.aspx
[4] www.nidec.com/fanpdfs/t92t200901.pdf
[5] www.formfactors.org/developer%5Cspecs%5C4_Wire_PWM_Spec.pdf
[6] www.elektor.com/100127

Listing: PLL code

```
if (PLLtimer<fastPeriod+PLLcontrol){                  // period adjusted by PLLcontrol
 PLLtimer++ ;                                         // count up
}
else {
 PLLtimer=0 ;                                         // reset PLLtimer
LED_PORT |= ( 1 << LED_BIT ) ;                        / start LED flash
PLLcontrol=wantedPosition-fastTimer ;                 // get PLL phase-difference

if (PLLcontrol >  fastPeriodHalf ) {                  // adjust to the range
  PLLcontrol -= fastPeriod ;
  }
 if (PLLcontrol < -fastPeriodHalf ) {
   PLLcontrol += fastPeriod ;
   }

 if (PLLcontrol >  100 ) {                            // limit to -100..100
   PLLcontrol =  100 ;
   }
 if (PLLcontrol < -100 ) {
   PLLcontrol = -100 ;
   }
}
```