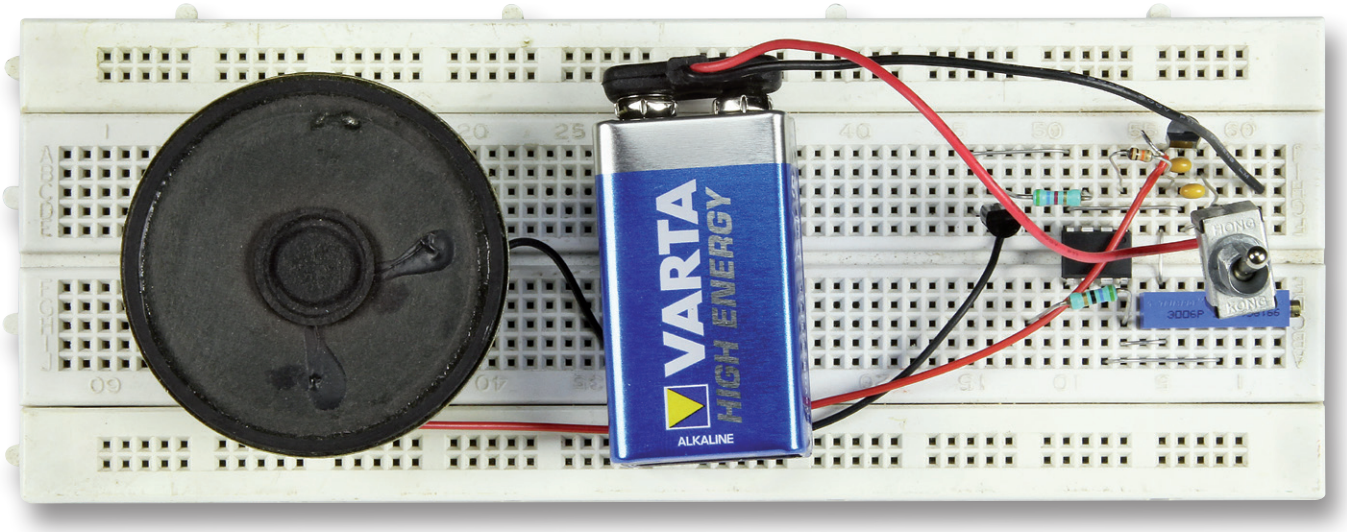
Simple Sound Effects 2.0

More noise, fewer components

By **Friedrich Lischeck**
(Germany)

In the May 1979 edition of Elektor we published a “Simple Sound Effects” unit. The author was rightly proud of the design which used a CD4040 counter, a CD4049 Hex inverter and a few odds and ends. Anyone who considered building the circuit all those years ago will be happy to know that a newer version is now available. Thanks to the marvels of modern technology we can now make the same sort of racket but with fewer components. That’s progress!



What we can’t see from the circuit diagram in **Figure 1** is how much hardware and wiring of the original design containing counters and inverters has been saved. The 21st century version does the same job in software and uses little more than just an 8-pin ATtiny45 microcontroller instead of the clever feedback counter design of the original circuit. The design also uses a single transistor buffer at the output to increase the level of annoyance produced from the small 8-Ω loudspeaker. A trimpot is included in the design to give some degree of frequency adjustment of the generated rumpus.

Software

Engineers these days are less likely to spend their time working out how to integrate hardware circuitry into their design and more likely to be solving the problem in software with a microcontroller. Even for relatively simple tasks, their low cost and flexibility means that microcontrollers should never be ruled out. The complete software for this design using an AVR controller (see **below**) is written in the popular Bascom-AVR [1] language. The original version 1.0 of the Simple Sound Effects featured 34 years ago was designed to produce a rising tone which switches back to a low tone when it reaches a maximum. The rate of increase is influenced by the value of a preset pot.

The same signal is now produced in this Simple Sound Effects 2.0 version using an ATtiny45. A preset variable resistor applies

a DC voltage level between 0 and 5 V to the analog input ADC.1. The A/D converter in the chip converts the DC level into a digital value in the range of 0 to 1023. The value produced is used to generate the frequency 'Fg'. Timer0 is used here; each time it overflows it generates an interrupt which calls the interrupt routine 'Oszillator'. In this routine the timer is loaded with a new value and the digital output PB.4 is toggled. The output signal produced is a symmetrical square wave. Each interrupt also increments the variable 'Fd' and then checks if it has reached 1000, if it has it sets it back to 100.
In the main program the ADC value is used to calculate the Preload and value of 'F': 'Fd' is added to the value 'Fg' produced by the ADC from the pot position. The result is a frequency somewhere between 200 Hz and 4 kHz, which then rises by a value of around 1000 Hz before it gets reset to the start value again.

Start value

Here we look at how the start value 'preload' of the timer is calculated. The time period 'T' of the 8-bit timer is derived as follows:

```
T = Tmax – Preload * Tclk
```

Preload can only have a value in the range of 0 and 255. 'Tclk' is derived from the microcontroller's clock signal and is divided down according to the prescaler setting (1/8/64/256). 'Tmax' is the maximum period which can be achieved with the timer:

```
Tmax = Prescale * 256 / f
```

where 'f' is the clock frequency, substituting we get:

```
Preload = (Tmax – T) / Tclk
```

and

```
Preload = (Tmax – 1 / 2 * F) / Tclk
```

Here 'F' is the frequency to be generated and 'T' is the corresponding period divided by two.

And finally

The resulting code is so small it only takes up a fraction of the available flash memory space. It will also happily run on an ATtiny25 (2 KB flash), it just so happened that an ATtiny45 was available at the time. The free demo version of the Bascom-AVR compiler is used to compile the program, this version is good for programs resulting in a compiled code of up to 4 KB. When it comes to flashing the microcontroller using either a home brew or off-the-shelf programmer be sure to remember to define the type of microcontroller you are using. Don't forget also to set up the fuses;

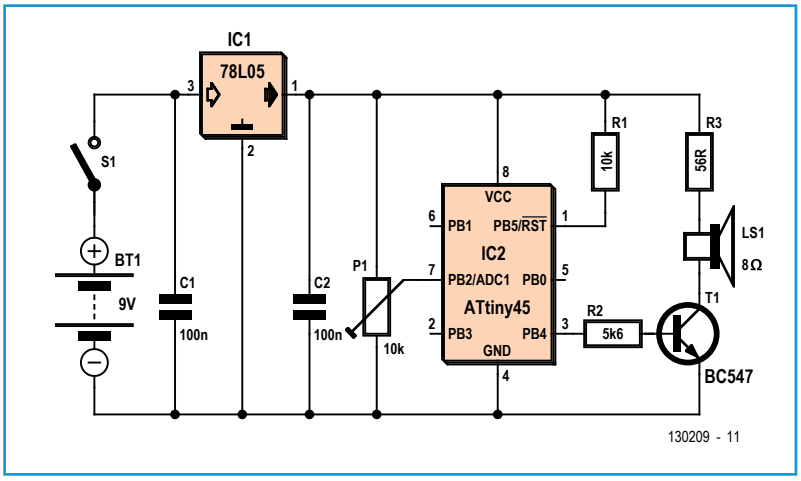
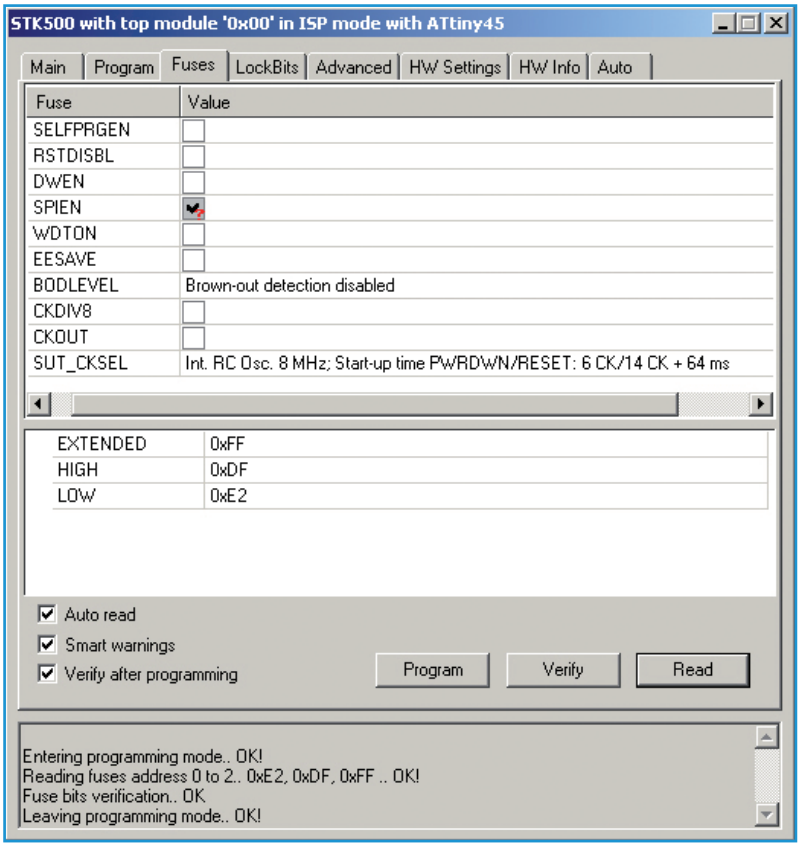
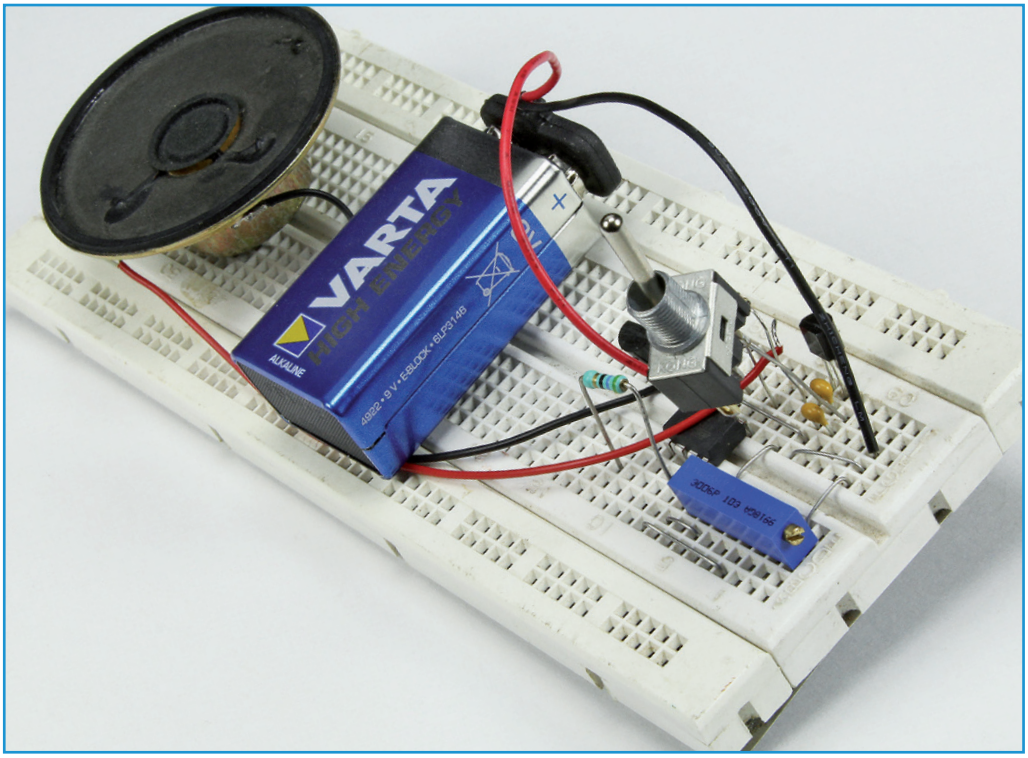


Figure 1.
The Simple Sound Effects 2.0 circuit is a lot simpler than the 1979 version.

Figure 2.
The ATtiny45 fuse setup.





Component List

Resistors
(all .25W)
R1 = 10kΩ
R2 = 5.6kΩ
R3 = 56Ω
P1 = 10kΩ trimpot

Capacitors
C1,C2 = 100nF 16V ceramic

Semiconductors
T1 = BC547
IC1 = 78L05
IC2 = ATtiny45, programmed

Miscellaneous
DIL-8 socket for IC2
Battery clip for 9V battery
B = 9V Battery
S1 = switch, 1 make contact
LS = loudspeaker, 8Ω 0.5W

screen shot **Figure 2** shows the fuse settings as displayed in Atmel Studio. Note that the clock divider 'CKDIV8' is disabled and the internal 8 MHz oscillator is selected with an extended start time. This prevents errors resulting in too low or no sound output. The source code and pre-compiled Hex file are free to download from [2].

There isn't too much to say about circuit construction, the small number of components can be easily accommodated on a small square breadboard—use an IC socket for IC2.

(130209)

Internet Link

[1] Bascom-AVR: www.mcselec.com

[2] www.elektor-magazine.com/130209

The Code

```
'Simple Sound Effects Tiny45
$regfile = "attiny45.dat"
$crystal = 8000000

'Pin B.4 is output:
Ddrb = &B00010000
'Start with output low:
Portb.4 = 0

Dim Preload As Byte
Dim F As Word
```

```
Dim Fg As Word
Dim Fd As Word
Dim Tmax As Single
Dim Tclk As Single
Dim H As Single

On Timer0 Oszillator
Config Timer0 = Timer , Prescale = 256
Enable Timer0
Enable Interrupts
Preload = &H64:                                    '100 Hz
Timer0 = Preload

Config Adc = Single , Prescaler = Auto , Reference = Avcc
Start Adc
'------------------------------------------------------------
'Init
Tclk = 256 / 8000000
Tmax = 256 * Tclk
Fg = 100
Fd = 100
'------------------------------------------------------------
'Main program
'F=200-4000
Do
   Fg = Getadc(1)
   H = Fg + 100
   Fg = Fg + H
   Fg = Fg + H
   F = Fg + Fd
   H = 2 * F
   H = 1 / H
   H = Tmax - H
   H = H / Tclk
   Preload = Int(h)
   H = Preload * Tclk
   H = Tmax - H
   H = 0.5 / H
   F = Int(h)
Loop
End
'------------------------------------------------------------
Oszillator:
   Timer0 = Preload
   Portb.4 = Not Portb.4
   Incr Fd
   If Fd > 1000 Then Fd = 100
Return
'------------------------------------------------------------
```