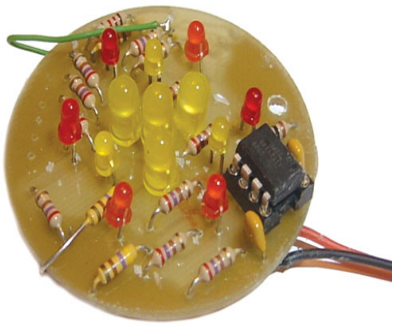


Pulse-width modulation is really useful. One of my ideas was to make a real electronic flame that was modulated with PWM. Many of you probably help decorating a Christmas tree or making a nativity scene, containing a shepherd’s fire near the shed at Bethlehem.



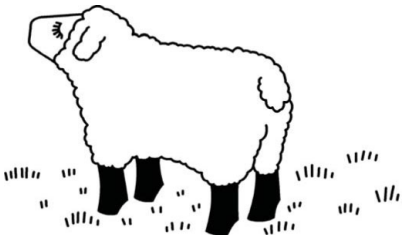
As a child, I wondered how to make a shepherd’s fire near the Holy shed. At that time, I did it with a small lamp covered with red paper, surrounded with some small, slightly burnt wooden branches. The lamp was battery-powered which, unfortunately, did not last very long. Also, I was not pleased with a “fire” that glowed constantly. A few years later, as my electronics knowledge grew, I designed a blinking fire – more like a car’s turn indicator...

Now I know enough to make an electronic flame that flutters like a real one. The prototype is shown in Figure 2, and is powered by a 5V mobile phone adapter. Because of this, I didn’t bother to include a 5V regulator on the board, but there is a place for one if you need it.

Properties of Flames

If one watches a real fire for an extended time, you'll notice that it's unpredictable in terms of the fluttering, intensity and colour of the flame. To address this, I have designed an electronic solution that is programmed into a small ATtiny45 microcontroller using Bascom-AVR. The ATtiny45 microcontroller contains plenty of PWM outputs. If we change the PWM parameters then we can simulate a really unpredictable fire. The PWM parameters that I change are the PWM frequency, duty cycle and direction –either rising or falling. On each of five output pins, I change all of these parameters independently, which really mimics the effect of an unpredictable flame. In this circuit, I have used eight LEDs connected as shown in the schematic diagram (Figure 3). On the PCB, the LEDs are placed in a circle, as a flame is normally circular in shape as well. In the center there are four yellow LEDs that represent the main flame and should be brighter. Around the perimeter are mixed yellow and red LEDs as desired.

Software



A project normally starts with the microcontroller selection, in terms of the number of pins and available functions etc. For this project I have chosen the ATtiny45: an 8-pin AVR with five (or six) I/O ports which is supported by the MegaPin & MiniPin II development boards. Because ATtiny45 is small and comes in a PDIP package, it's ideal for those who do not like SMD parts. It's also attractive to electronic newbies because it shares the same case as the popular NE555, yet offers much more!

When designing shepherd's fire, I had a few surprises. The first was that one I/O pin is connected to the microcontroller's Reset pin, which means that if I use this pin as an I/O pin then I can no longer program it using an SPI programmer. Great!

So, I settled on using only five I/O pins. Then I started testing Timer0 and Timer1 individually. Timer0 performed normally, but Timer1 did not. The reason for this is that Timer1 is an 8-bit Timer and not the usual 16-bit Timer normally found in AVRs. Therefore, Bascom-AVR refused to compile my code. After a thorough reading of the datasheet, I discovered the proper settings for Timer1. In the TCCR1 register you must set a clock prescale factor, after which Timer1 will

perform as it should.

Shepherd’s fire

2012_AVR_UK_199



Continue reading

Shop area

