Impress-4-Less with the Elektor Virtual Fireplace

Where there's no smoke there's PIC & mp3

By **Piero Di Stefano**
(Canada)

A softly burning and quietly crackling fireplace has imminent attraction to people and pets, but not everyone can afford to have the real thing in their living room. So without further ado here's an all-embedded, all-mp3 alternative to smoke, odors, flying embers and wood cleaving (out there in the cold). Although not generating any significant heat, the project ends all of your *will-it-burn-in-time* anxiety.

Spot-on for the Festive Season upon us now, here is a device that can simulate a burning fireplace to a fair degree. It needs 230-V (US: 115-V) light bulbs (or anything that is dimmable). You can use a red lamp to create a deep red background (which is not dimmed), plus one yellow and one white lamp (or any other color you like) that will flicker in rhythm with the crackling and sputtering noises of the virtual fire. In fact the device comes with a small (but loud) amplifier that plays the sound of a real fireplace. The mp3 sound file can be downloaded from [1]; if you do not like it, there's tons more on Youtube.

How it works

Looking at the circuit diagram in **Figure 1**, the audio file is stored on a memory card and played back by an mp3 sound module, SoundModule1 or 2. The author used the Tenda mp3 player, which gets connected as “SoundModule1”. Elektor Labs however purchased and mounted the type WTV020-SD-mini MP3 player and then connected the device as ‘SoundModule2’. SoundModule1 and SoundModule2 have slightly different PCB positions reserved for them.

During boot-up the micro tells the mp3 module to play the first file, and the BUSY line (pin 15) goes LOW until file plays. When the playback is over, the BUSY line goes High for a moment in order to replay the file. The Microchip 12F683 (IC1) sends the Play command to the mp3 module (actually just a pulse) over and over using the levels detected on GP0. Hooray the 12F683 is a DIP-8 device.

The lamp flicker is created by the micro and two triacs. The red lamp is connected straight to the AC powerline while the yellow and white lamps are driven by their respective triacs Tri1 and Tri2, with optocouplers IC5 and IC4doing the interfacing to the microcontroller.

A good old LM386 audio power amplifier IC drives an 8-ohm loudspeaker up to an output power of about 500 mW. A volume control is provided with trimpot P1. The other control, P2, may seem odd here but does serve a purpose—see the author’s notes on it in the mikroC source code listing.

The DC power supply is conventional around a 3.3 V regulator, IC3. The unregulated supply voltage of about 8 volts is used to power the audio amplifier IC only. All other devices run off 3.3 volts.

The microcontroller is made aware of the zero-crossing instants of the powerline voltage by means of transistor T3. This timing information is essential to make the triacs conduct at the right instant and so control the voltage reaching the lamp while prevent massive electrical noise on the lamp lines. The lamp

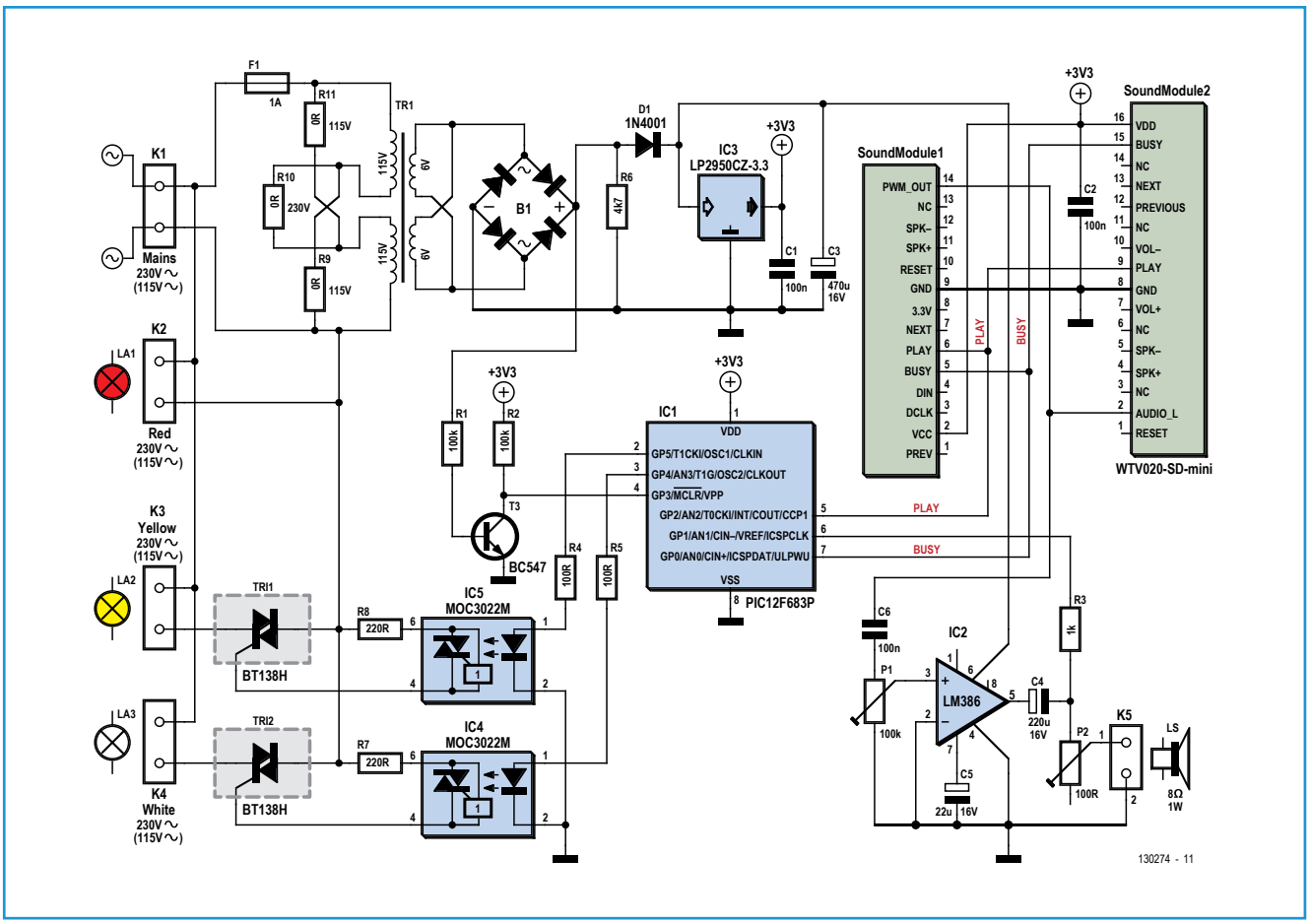

Figure 1.
The Virtual Fireplace is the Internet-Age equivalent of wood cleaving, fire starting, gasoline pouring, smoke, damp newspapers and the excruciating wait for a good fire. It lacks the ability to generate much heat though.

dimming method is called *phase angle control*. The component values and the transformer primary connections shown in in the schematic are for 230 VAC 50 Hz grids. To adapt the circuit to 115 VAC 60 Hz operation, install the 0-Ω jumpers R9, R11, and change fuse F1 to 2 A, slow. Change: R7, R8 to 100 Ω.

Build it

Tim at Elektor Labs designed a spaciously laid out and electrically safe printed circuit board for the project, see **Figure 2**. The board also accommodates the 2 x 6 V power transformer. Using the parts list and the photos in this article building the project should not pose problems as no SMD or other miniature parts are involved. The result is pictured in **Figures 3** and **4**.

CAUTION. The powerline Live (L) potential exists on several PCB tracks, solder points, screws and component terminals on the board, hence **the board and any wiring, devices or equipment connected to it must never be touched when in operation**. Also, the board MUST be built into an approved non-metallic enclosure, observing all regulations and precautions for electrical safety.

Software: it's all free

The firmware was written using mikroC Free Edition from Mikroelektronika and is less than

COMPONENT LIST

Resistors
(0.25W unless otherwise stated)
R1,R2 = 100kΩ
R3 = 1kΩ
R4,R5 = 100Ω
R6 = 4.7kΩ
R7,R8 = 220Ω
R9,R10,R11 = 0Ω or wire bridge (see text on changes for 115V grid voltage)
P1 = 100kΩ trimpot
P2 = 100Ω trimpot 0.5W

Capacitors
C1,C2,C6 = 100nF 50V
C3 = 470µF 16V
C4 = 220µF 16V
C5 = 22µF 16V

Semiconductors
B1 = bridge rectifier, e.g. Vishay type 2W005G-E4/51; Newark/Farnell # 1497581
D1 = 1N4001
IC1 = PIC12F683P-I/P, programmed, Elektor Store # 130274-41
IC2 = LM386N-1
IC3 = LP2950CZ-3.3/NOPB
IC4,IC5 = MOC3022M
T3 = BC547
TRI1,TRI2 = BT138-800

Miscellaneous
F1 = fuse, 1A, slow (2A @ 115V)
HS1, HS2 = heatsink, 21 K/W, Fischer Technik type FK 230 SA L1, Newark/Farnell # 1892318
K1,K2,K3,K4 = PCB screw terminal block, 7.5mm pitch
K5 = PCB screw terminal block, 5mm pitch

SoundModule1 = Tenda MP3 player *
SoundModule2 = WTV020-SD-mini MP3 player *
DIP6 socket for IC4 & IC5
DIP8 socket for IC1 & IC2
TR1 = power transformer, PCB mount, 2 x 115V prim., 2 x 6V sec., Block type AVB1.5/2/6, Newark/Farnell # 1131474
Fuse holder, PCB mount, with cap
Enclosure, e.g. Hammond type 1591USBK, Newark/Farnell # 1426582
LS = miniature 8Ω 1W loudspeaker
PCB, V1.1, Elektor Store # 130274-1

* use either SoundModule1 or SoundModule2

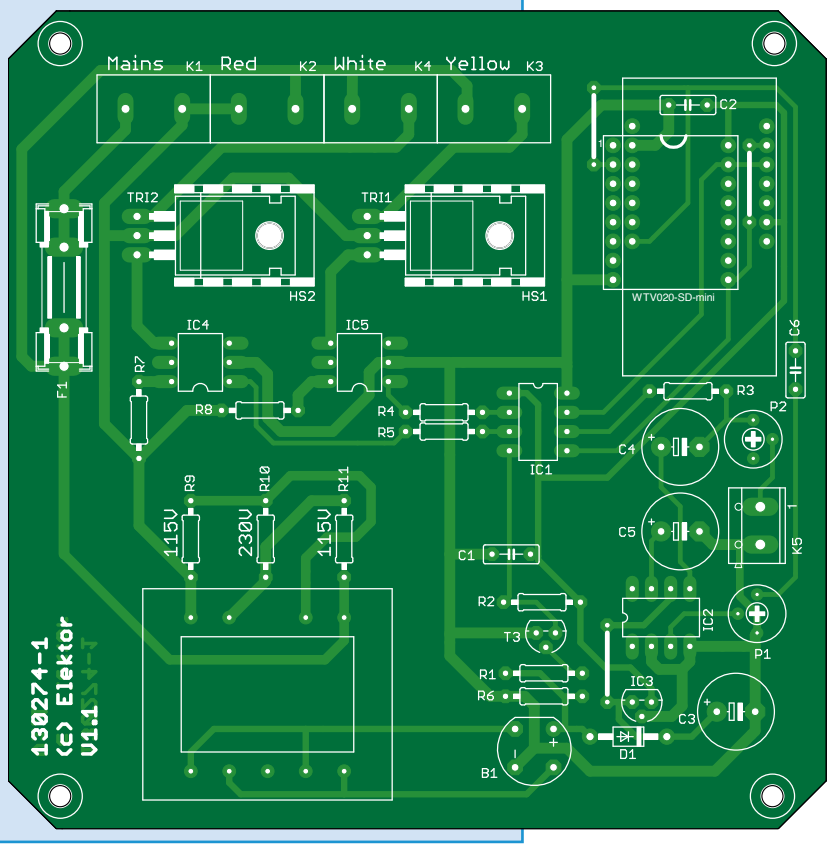

Figure 2.
In consideration of electrical safety the Virtual Fireplace is preferably built on this printed circuit board.

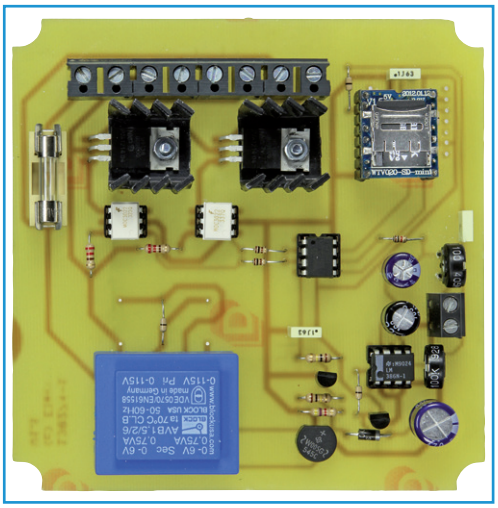

Figure 3. Assembled and tested board.

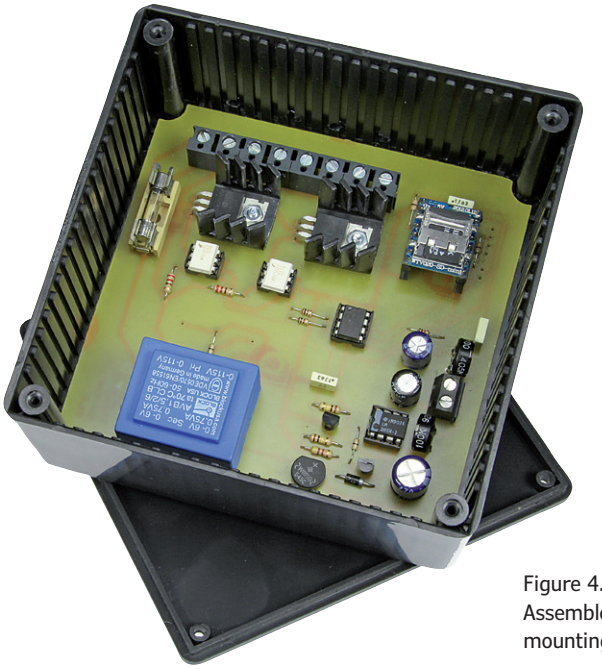

Figure 4.
Assembled board ready for mounting in the enclosure.

2 Kbytes in size—actually there's just over 200 words worth of executable code). **Listing 1** appended at the end of this article shows the mikroC program, which may be downloaded free from [1]. The content of TMR0 should match the frequency of your local grid, i.e. 50 Hz or 60 Hz, so study that listing carefully. Our European readers at this point might care to know that Elektor and Elektor.Post are also published in the USA. Conversely, US and Canadian readers should realize that an originally Canadian circuit got post engineered in The Netherlands.

With all the microcontroller files in place, understood, discussed, debugged, and compiled… inescapably there are the PIC fuse settings to grapple with, so these appear in summary in **Figure 5**. Next, burn the PIC, not the wood.

In action and on YouTube too

The author's Virtual Fireplace can be seen in more or less blazing action on YouTube [2]. Elektor Labs also built their version and the photos in this article go to show the results of their efforts. They also shot a video of the VF in action, with Tim explaining how it works [3].

Although the ancient fireplaces in several rooms in Elektor House (believed to be built in the mid 1600's) were professionally refurbished a few decades ago, the associated chimneys are closed and it's no longer possible or allowed by our CEO, CFO, quasi-resident architect, and Health & Safety Officers to enjoy a real fire with wood, smoke, strong drinks and all that. In good Christmas spirit it did not deter us from building a state-of-the-art *ersatz* though and "burning" a PIC12F683 or two instead of cleft wood logs.

(130274)

Internet Links

[1] Mp3 file, software, PCB layout files: www.elektor-magazine.com/130274, archive file # 130274-11.zip

[2] Author's Virtual Fireplace on YouTube: http://youtu.be/xK7Jj2aXOXI

[3] Elektor's Virtual Fireplace on YouTube: http://youtu.be/L_4yy8giTHk

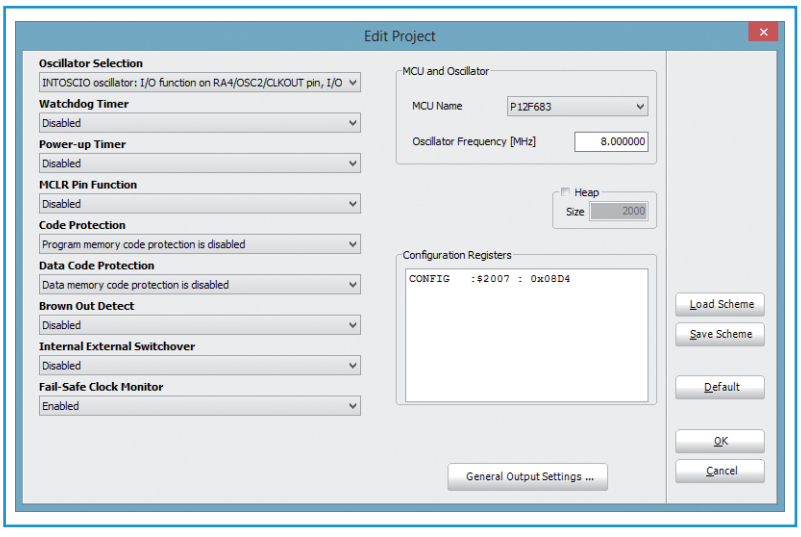

Figure 5.
Before programming the PIC, you need to get (and set) the internal fuses right.

Listing 1. Virtual Fireplace MikroC source code listing.

```
/*
 * Project name:
     PIC12F683 Electronic fireplace
 * Piero Di Stefano, MAY 8th 2013.
   The triacs are driven generating a delay that is inversely proportional to the sound level.
  a 60Hz half-wave takes 1/30= 16.66ms to complete, so we have to handle any time between zero and
      8333us.
  Here we go up to 8160, then above such a value, we keep the triac triggered. It's impossible to
      notice it.

  At 50Hz the timespan will be longer, 10ms, so we should set TMR0 prescaler to 128. That means
      that TMR0 interrupt will fire every
  128*255/2000000 = 16320us

  So first thing, in the main() remember to change the following statement:
  OPTION_REG = 0b10000101;  //TMR0 prescaler: 1/64

  with
  OPTION_REG = 0b10000110;  //TMR0 prescaler: 1/128

  In order to limit its range between 0 and 10000us, we must use an init value for TMR0 of:
      255*10000/16320 = 156.25
  Obviously we must use a smaller value, so no less than 156
  Countercheck:

  TMR0 = 156
  Tmr0 interrupt will fire every (156)*128/2000000 =  9984us

  Last important step to be taken:
  inside the TMROIF routine, we must change the following line:

  TMR0 = ADC_reading << 2;

  that prepares TMR0.
  Since we must use a minimum value of 156 for TMR0 to properly trigger the thyristor at 50Hz, we
      must limit our range to 255 -156 = 99.
  I suggest to limit the ADC reading to 127, shifting its value 3 times instead of just 2 and
      adding no more than 29 to it

  TMR0 = (ADC_reading << 3) + 29;

  A smaller value would give us more allowance

  So the problem should be solved.


 * Test configuration:
```

```
     MCU:             PIC12F683
     Dev.Board:       EasyPIC6
     Oscillator:      Internal, 8MHz
     Ext. Modules:    none

     SW:              mikroC PRO for PIC
                      http://www.mikroe.com/eng/products/view/7/mikroc-pro-for-pic/
 * NOTES:
 Tenda mp3 mplayback module:
http://www.google.ca/#hl=en&sclient=psy-ab&q=tenda+electronics+tdb380&oq=tenda+tdb&gs_
      l=hp.3.1.0j0i22i30.848.5801.0.8070.9.9.0.0.0.0.120.811.6j3.9.0...0.0...1c.1.14.psy-ab.Cd-
      q3yzoZcI&pbx=1&bav=on.2,or.r_qf.&bvm=bv.46751780,d.dmQ&fp=ca03a074a5a6f34d&biw=1017&bih=596
*/

     unsigned short dummy;
     unsigned int ADC_reading;

void interrupt()
{
   if(INTCON.T0IF)
   {
     GPIO.F5 = 1;          //past some time between 0 and 8160 us, brings LOW the output
     INTCON.T0IE = 0;      //disables TMR0 IRQ preventin it to fire until the next rising edge
      (GPIF)
     INTCON.T0IF = 0;      //clear bit
   }

   if(INTCON.GPIF)
   {
     dummy = GPIO;
     /*IMPORTANT, this is the core of the program
     First, we multiply ADC_reading by 4 to compensate for the low voltage level from the
      loudspeaker.
     Can't always amplify the mp3 file level, otherwise it will sound distorted. So keep the volume
      down and change the voltage threshold
     Note that the louder the volume, the higher the number we get from the ADC, the higher value
      TMR0 will be initialized at.
     So we get a smaller delay on triggering the triac and the light will appear brighter.
     */
     TMR0 = ADC_reading << 3 + 29;
     //If the sound level is all the way up to the max (at 245/255 to be precise), disable TMR0 and
      set GP5 HIGH permanently,
     //it will keep the triac triggered for the whole cycle
     if (TMR0 > 245)
     {
       INTCON.T0IE = 0;
       GPIO.F5 = 1;
     } else
       {
```

```
         delay_us(160);       //fine adjust zero-crossing synchronism, to compensate the early
      saturation of the bjt and trigger the triac precisely
         INTCON.T0IE = 1;
         GPIO.F5 = 0;         //triac gate LOW  NOTE: GP2 is LOW to begin with, then goes high.
       }

     INTCON.GPIF = 0;
   }

}


void main() {

OSCCON = 0b01110001;      //Enables 8MHz int osc  (we must SUPERSEDE Project prop. dialog box in
      order to get it to work)

OPTION_REG = 0b10000110;  //TMR0 prescaler: 1/128
INTCON = 0b10001000;      //general irq enabled  + GPIF (TMR0 enabled programmatically)
ANSEL = 0x02;             //AN1 enabled
CMCON0 = 0x07;            //disables comparators
IOC.IOC3 = 1;             //enables RBINT on GPIO.F0

trisIO = 0b00001011;      //GP0 digital input (~BUSY) to replay the file, GP1 = analog input from
      audio player, GP3 digital input from ZCD
GPIO = 0b00000100;        //GP2 must should start high.
TMR0 = 156;               //Init value


    // play mp3 file at startup
    GPIO.F2 = 1;
    delay_ms(100);
    GPIO.F2 = 0;
    delay_ms(100);



 while(1)
 {
    ADC_reading = ADC_Read(1);

    //White light bulb: when the sound level is higher, it goes on, and stays that way until next
      reading.
    if (ADC_reading > 1000) GPIO.F4 = 1; else GPIO.F4 = 0;


    if (GPIO.F0)     //F0 = HIGH only when device is in IDLE mode, goes LOW when playing
    {
      delay_ms(500);
```

```
      GPIO.F2 = 1;
      delay_ms(100);
      GPIO.F2 = 0;
      delay_ms(100);
    }

 }//end while

}// end main
```