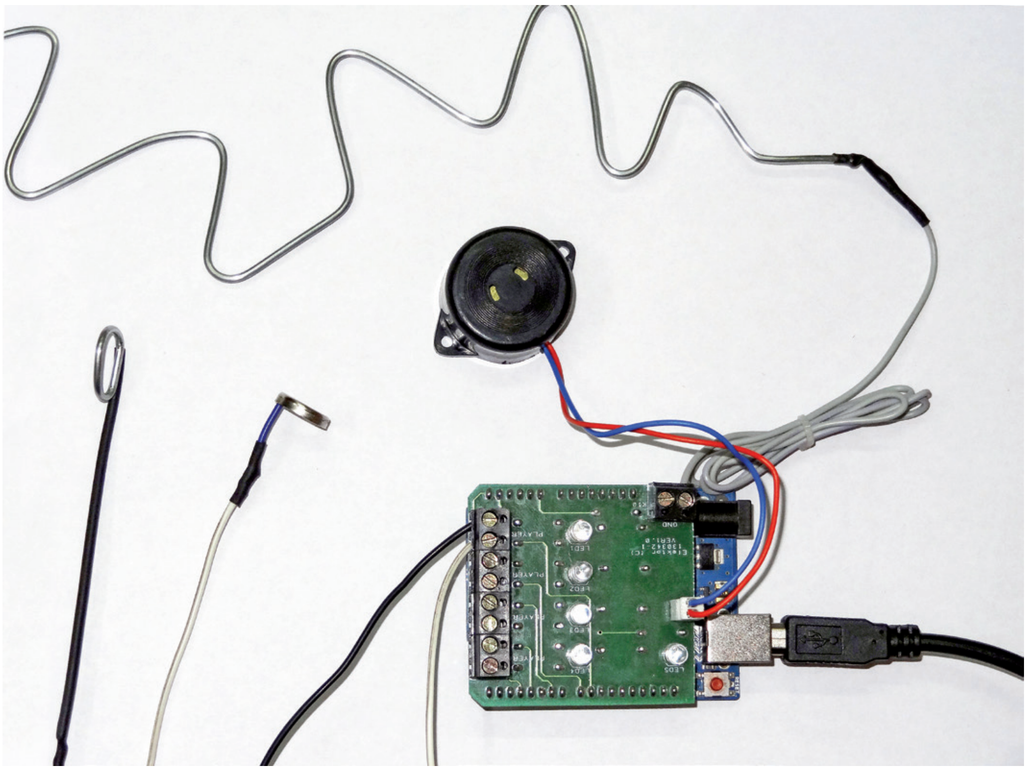
Wire Loop Game

Nerves of Steel – and Arduino in control



Here's a multi-player (max. 4) version of that nerve-racking game called Wire Loop where contestants have to pass a metal loop around a stretch of purposely warped wire or tubing without touching the 'track'. First one to finish without setting off the dreaded FAULT!! alarm, wins. Inspired by a circuit from Elektor's dim analog past we decided to add an Arduino Uno to make it suitable for all you Bitwise People out there.

By **Sunil Malekar**
(Elektor Labs India)

The game as we want it

Wire Loop is a dexterity-only game, meaning it has nothing to do with intelligence, cheats, or strategic skills. Our Arduino'd version is designed for up to four players, each with his/her own loop to "pass the test".

The goal is to pass the loop from the beginning of the track to the finish without touching the track wire. If the player touches the track wire a fault tone sounds, and a status LED identifying the player lights up. If a player successfully reaches the end of his/her track and touches his/her Finish point there, he/she wins the game. A short melody sounds and all the LEDs will blink.

The circuit – from basics to schematic

The main parts of the circuit are the loops, the wire track, the fault indicator and the variable-sound buzzer. This overall size of the circuit depends on the number of players playing the game. Our version supports up to four players—each one has a loop which is interfaced with an Arduino microcontroller port pin with an external pull-up to V_{CC}. Each participant's personal 'Finish' touch point is interfaced with the Arduino microcontroller with the help of a pull-down resistor to GND.

The player holds the loop, which normally produces the logic High signal. The warped track made from stiff copper or steel wire is connected to the GND terminal. When the game starts the player is challenged to move his/her loop from the start to the personal finish point without touching the track wire. If the player touches the track wire the logic High signal will drop to Low. This is flagged by the Arduino micro, which responds instantly by lighting the respective LEDs and making the buzzer sound. Each player has an individual buzzer sound and LED pattern. When a player touches his/her personal 'Finish' point he/she wins the game and there is a musical flourish from the Arduino microcontroller.

Referring to the schematic in **Figure 1**, the heart of this project is Arduino Uno R3 board using an ATmega328P microcontroller. The microcontroller GPIO pins are connected to our dexterity game shield by way of connectors K1, K2, K3, K4 of the Arduino "shield". K1, K2, K3 and K4 link the winners' "Finish" points and loops using Arduino microcontroller pins (Loop1, Loop2, Loop3, Loop4 and Win1, Win2, Win3, Win4) fitted with external pull-ups (R11, R12, R13 and R14) and pull-downs (R7, R8, R9 and R10). K10 is used to connect the solid track wire to Ground. The loop and winner correlation is established using GPIO PINs of the ATmega328p controller (3, 4, 6, 7, 8, 9, A4 and A5—according to the Arduino Uno R3 schematic). The four indicator LEDs are connected to Arduino microcontroller pins 13, 12, 11 and 10, which indicate individual faults when the loop (loop1, loop2, loop3 and loop4) touch the track wire. The buzzer generates individual tones for individual players when the loop touches the track wire or tube. LED5 on Arduino A0 indicates Start and Win status of the game.

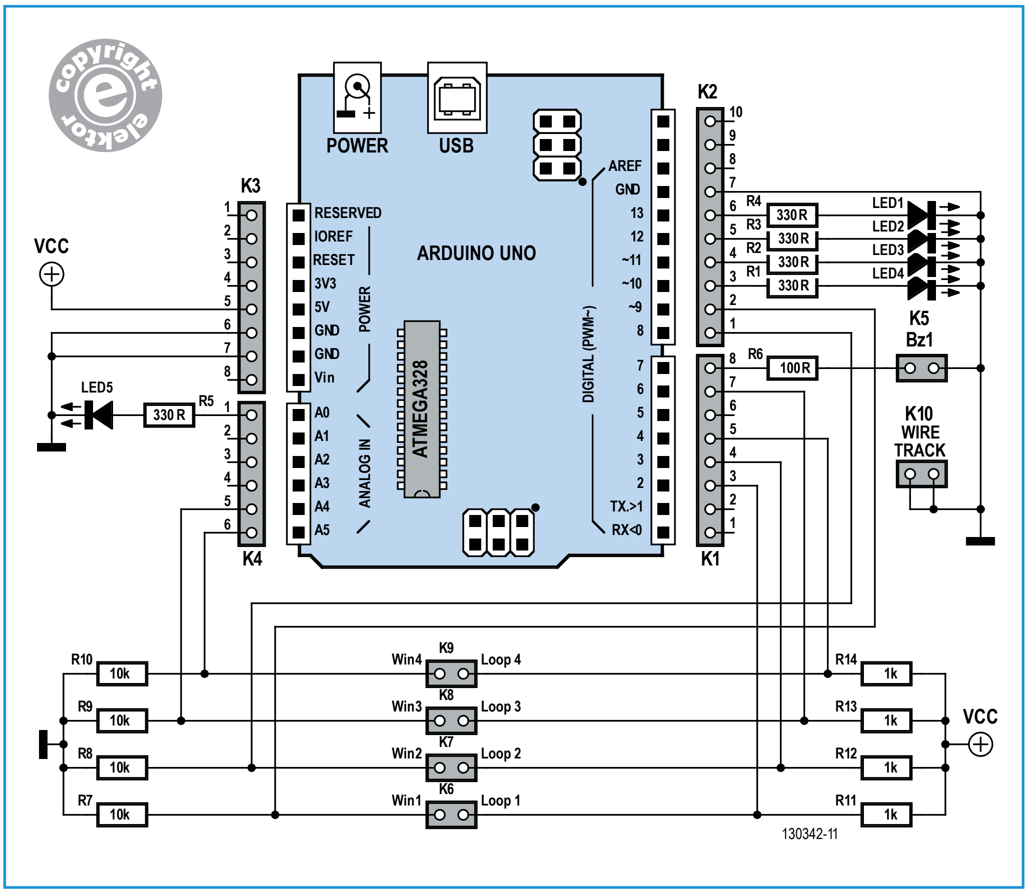


Figure 1.
Schematic of the Wire Loop Game, a unique shield for the Arduino Uno

Building

The double-sided through-plated circuit board designed for the game is shown in **Figure 2**. The PCB artwork if needed can be downloaded from [1]. Words do not do justice to the simplicity of the "shield" we're building, which contains through-hole parts only. The resistors and the four pinheaders are inserted at the underside of the board, noting that the pinheader pins can only be soldered at the PCB top side. The wires to the player loops, and the track wire, are connected on sturdy PCB screw terminal blocks. BTW a shield is a board that gets plugged onto the Arduino's extension connectors. Just as in medieval Italy, a shield kind of helps Arduino deal with things out there in the real world.

Software

The Wire Loop Game for up to four players got developed using an Arduino microcontroller Board and the associated Arduino develop-

Component List

Resistors
R1–R5 = 330Ω
R6 = 100Ω
R7–R10 = 10kΩ
R11–R14 = 1kΩ

Semiconductors
LED1–LED4 = LED, red, 5mm, e.g. Newark/Farnell # 1780754
LED5 = LED, green, 5mm, e.g. Newark/Farnell # 2112108

Miscellaneous
K1,K3 = 8-pin pinheader, 0.1'' pitch
K2 = 10-pin pinheader, 0.1'' pitch
K4 = 6-pin pinheader, 0.1'' pitch
K6–K10 = 2-way PCB screw terminal block, 5mm pitch
Buzzer, e.g. Newark/Farnell #1022400
PCB # 130342

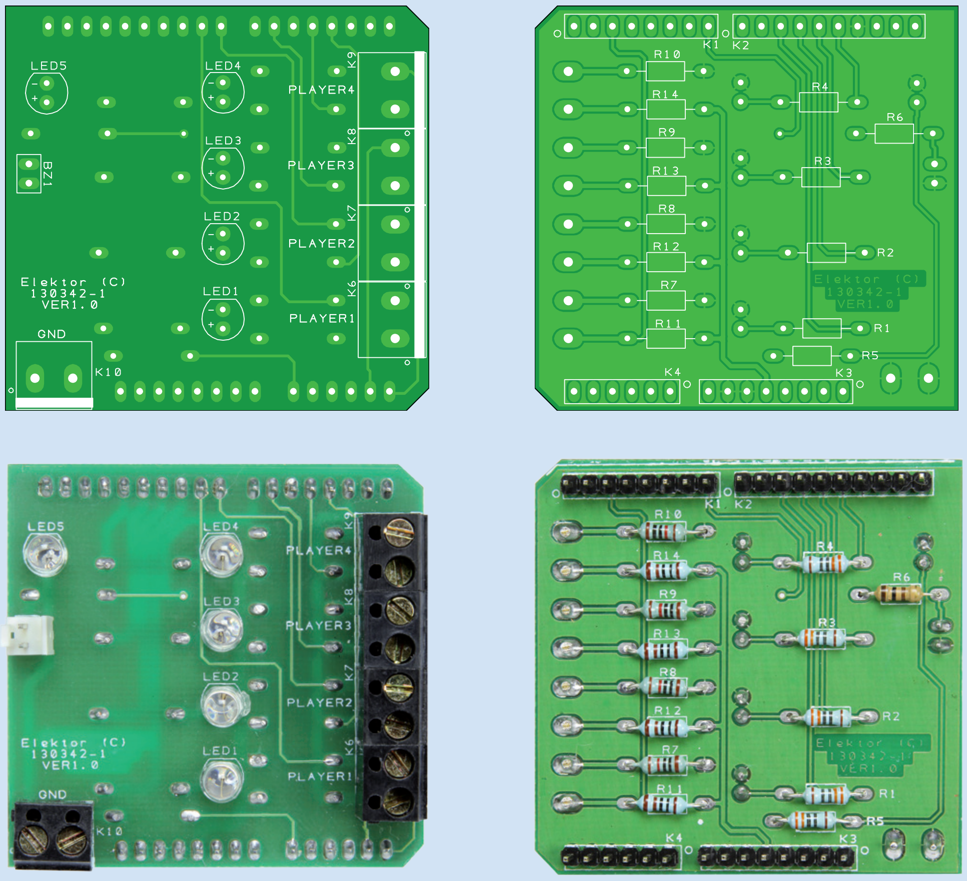


Figure 2.
Circuit board design for the Wire Loop Game, which has taken the form of a shield for Arduino.

ment IDE. The Arduino code can be downloaded free of charge from [1].
The firmware configuration is governed by the number of players. The individual player's winning Finish touch point, fault indicator LED, and the buzzer to loudly reveal "FAULT" all are configured with Normal GPIO Pins of the Arduino microcontroller.
A function identified as `tone ()` enables Arduino to play a tone through a small buzzer, and with some extension, play music too. The function call is as follows:

```
Tone (pin, frequency, duration)
```

or

```
Tone (pin, frequency)
```

Example:
```
Tone (3,440)
```

where
`pin` is the GPIO pin number the speaker/buzzer is hooked up to;
`frequency` is the frequency of tone in hertz;
`duration` is the time in milliseconds the tone lasts.

To stop the sound, simply code:

```
noTone (pin)
```

The tone function generates a square wave of the specified frequency (and 50% duty cycle) on a pin. If the duration of tone is not specified then the wave continues until a call to `noTone()`occurs. The pin can be connected to a piezo buzzer or other speaker to play tones. Only one tone can be generated at a time. If a tone is already playing on a different pin, the call to `tone ()` will have no effect. If the tone is playing on the same pin, the call to `tone ()` will reset the frequency.
To be able to step tones and so make a basic melody, we have to include the "pitches.h" file. This file just gives you a variable defined as a macro for every tuned note in hertz, so we can work out basic notes. This makes writing "songs" much easier. All the tone notes and notation conventions can be found in the "pitches.h" header file.

Creating musical notes

Simple notes work like a musical tone. Pitch and duration of the notes (octaves) can be adjusted by coding in the "pitches.h" file. The tone note can be downloaded from [2]; the tone creation and frequency tutorials from [3], and Arduino-specific stuff, from [4].
In the firmware some musical notes are available based on the "pitches.h" file and created using the above links. A few commonly used ones are listed in **Table 1**.

Table 1. Musical Notes

Constant Name	Frequency (Hz)
NOTE_B2	123
NOTE_C3	131
NOTE_CS3	139
NOTE_D3	147

Web Links

[1] www.elektor-magazine.com/post
[2] http://code.google.com/p/rogue-code/wiki/ToneLibraryDocumentation
[3] http://en.wikipedia.org/wiki/Note
[4] http://arduino.cc/en/Tutorial/Tone

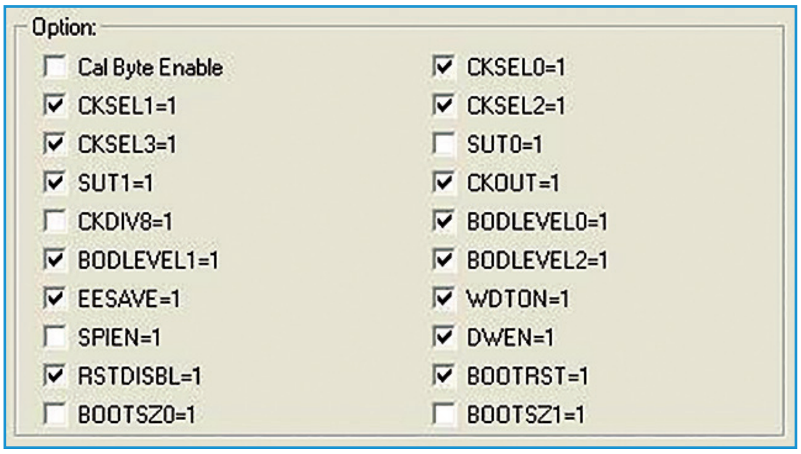
Arduino programming

The Arduino Uno board can be programmed by the Arduino IDE itself using its internal bootloader. Here we have programmed the Arduino Uno using an external programmer. When using an external programmer you have to set *Them Notorious Fuse Bits*—the configuration should follow **Figure 3**.

Playtime!

Once the assembly is done, the Arduino is powered up with the USB/DC 12 V power adaptor. Initially LED5 will blink continuously, indicating Game On! The player reaching the end of his/her track without touching is declared Winner and honored by the corresponding LEDs and his/her musette from the buzzer. A new game is started by pressing the Reset button on the Arduino Board.

(130342)


Option:

☐ Cal Byte Enable	✓ CKSEL0=1
✓ CKSEL1=1	✓ CKSEL2=1
✓ CKSEL3=1	☐ SUT0=1
✓ SUT1=1	✓ CKOUT=1
☐ CKDIV8=1	✓ BODLEVEL0=1
✓ BODLEVEL1=1	✓ BODLEVEL2=1
✓ EESAVE=1	✓ WDTON=1
☐ SPIEN=1	✓ DWEN=1
✓ RSTDISBL=1	✓ BOOTRST=1
☐ BOOTSZ0=1	☐ BOOTSZ1=1

Figure 3.
If you use and external programmer, be advised to adhere to these Fuse Bit settings.