An AVR microcontroller based electronic dice

Abstract:

Travel to outer space sounds very exciting but now we are here in our space ship and we have about 10 square meters for 5 people. It will take another week until we reach the first space station. I took my mp3 player with me and after so many days of travel I have heard every song at least a dozen times.

James was much more clever: He bought an electronic dice from shop.tuxgraphics.org because normal dice don't work in the absence of gravity....

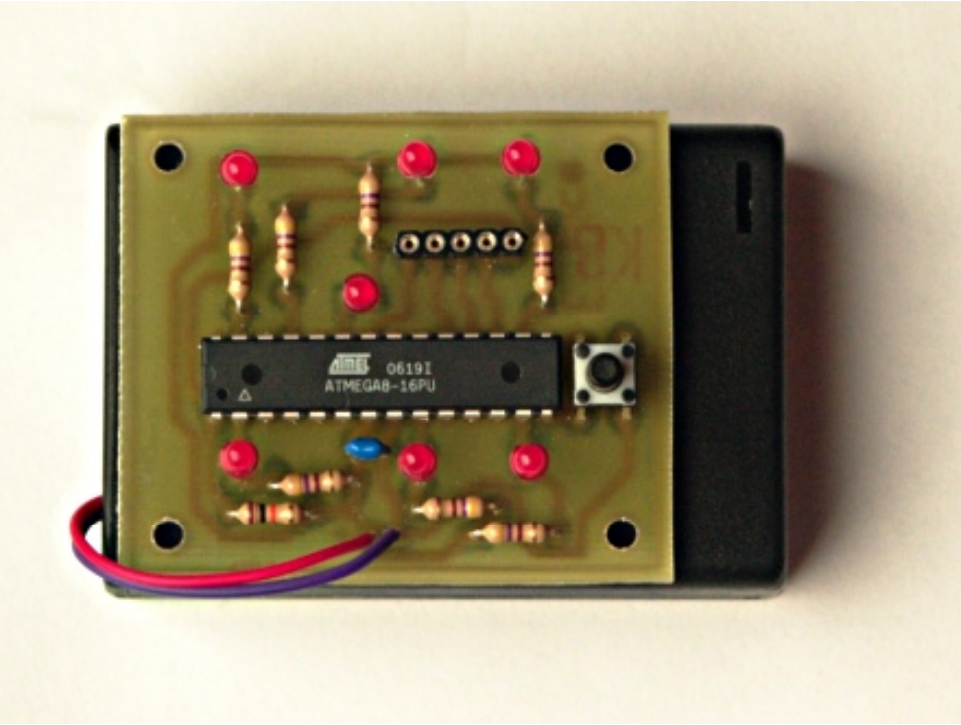
Let's play ;-).

The electronic dice

A dice is essentially just a special display for numbers from 1 to 6.

Why is a real dice a random number generator? If your hand would have perfect precision mechanics and you could control speed and position exactly then a real dice would probably be deterministic. In other words the one tossing the dice is the one introducing the randomness.

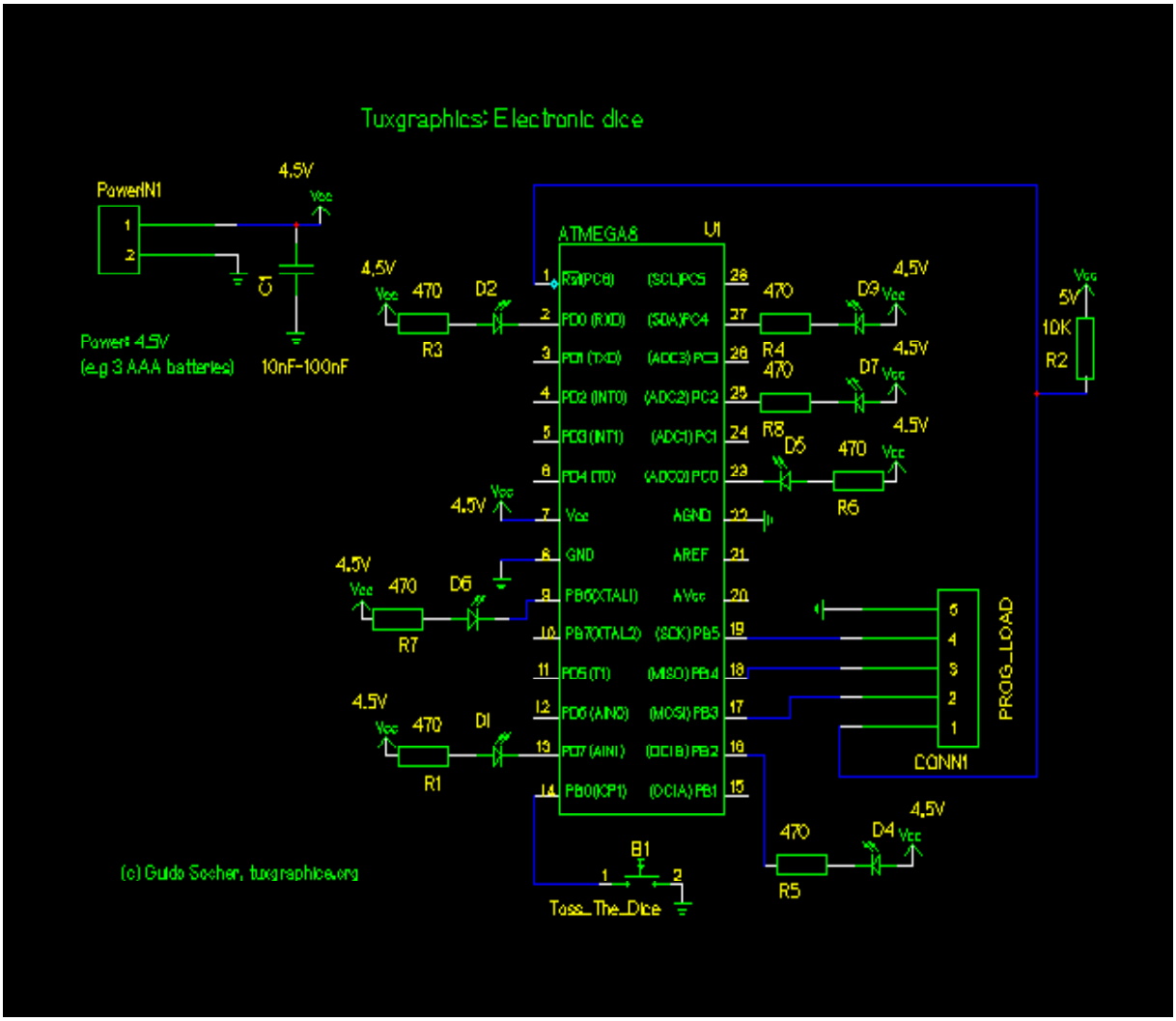
If you look at the code for our electronic dice then you will see that it is just a counter. It counts up while you press a button. Why is this a prefect dice? It's a dice because microcontrollers do operations in µs and humans are unable to control pushbuttons and time with the precision of µ-seconds. Even if you hit the pushbutton just for a moment the microcontroller's counter will loop around 56 times, or 80 times, or 102 times,.... It's impossible to control it precisely. It's random.



7 LEDs with resistors, a microcontroller, a push button and a battery box.

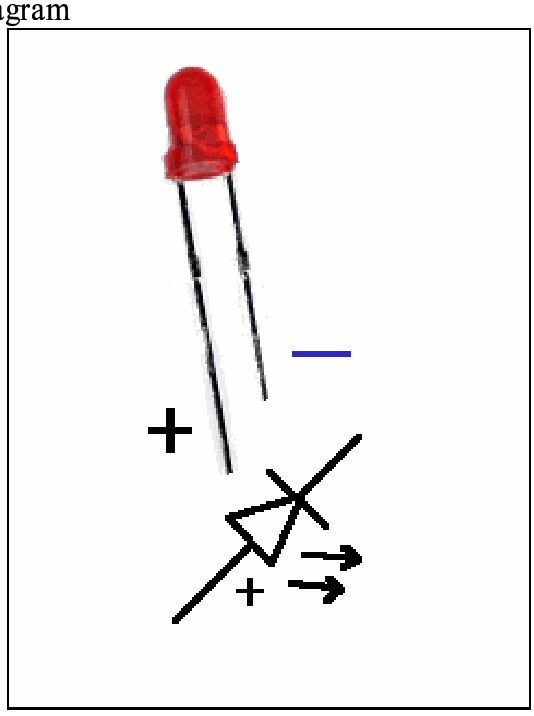
The E-dice as a nice AVR project

The E-dice is a simple but useful AVR starter project.



The circuit diagram

The only polar components in this circuit are the atmega8 and the LEDs. The atmega8 has mark on the side where pin 1 is. The LEDs are always oriented such then minus pin is connected to the wire that leads finally to one of the atmega8 pins.



Which pin on the LED is minus and which is plus? By convention the longer wire is always the plus wire and the shorter wire the minus wire as shown in the picture on the right.

Let's take a look at the code. This is the main part of the code.

The complete source code is available for download at the end of this article:

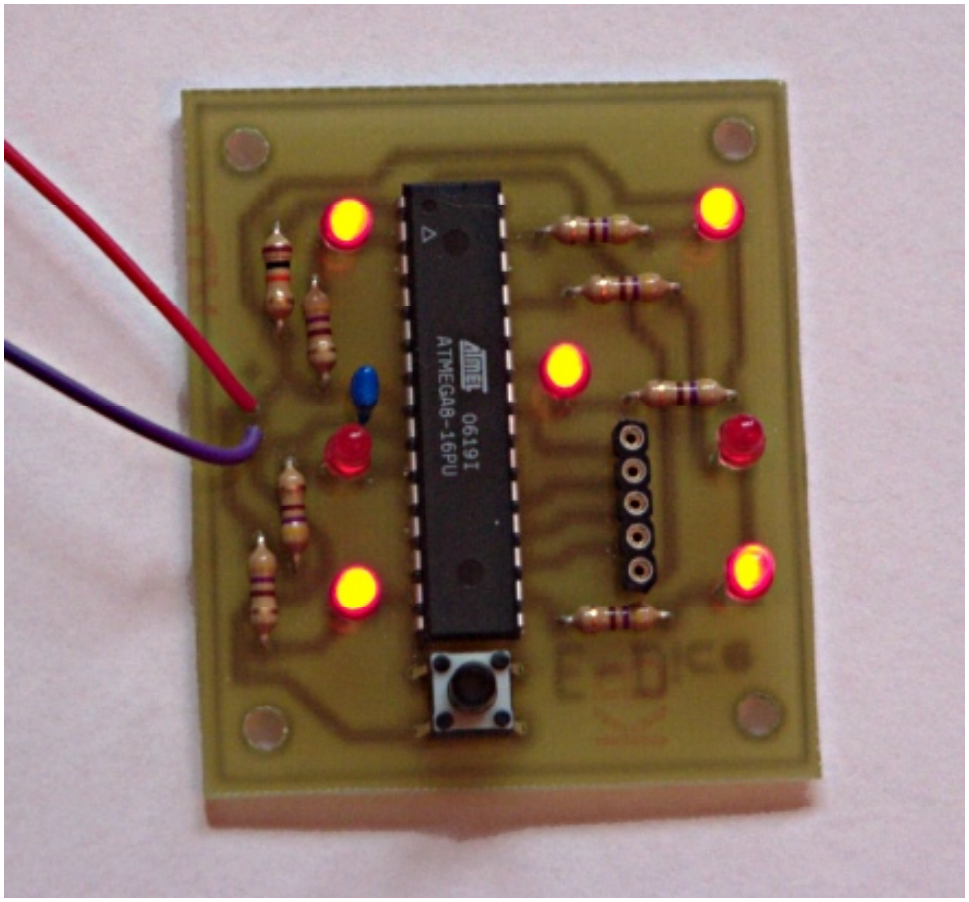
```
unsigned char i=0;
initLEDports();
DDRB &= ~(1<<PINB0); // input line
PORTB|= (1<<PINB0); // internal pullup resistor on

while (1) {
        if (bit_is_clear(PINB,PINB0)){
                // button pressed, switch off the LEDs and count
                allLEDsOff();
                i++;
                i %= 6; // numbers from 0 to 5
        }else{
                // no button press. display result:
                displayNumber(i);
        }
}
```

What you can learn here is how to check if a button was pressed. The button is connected between pin PB0 and GND. After initializing the LED connections as output (initLEDports) we write a zero to the data direction register on the position responsible for PB0. This causes the pin to be a digital input line. There is also the possibility to switch on an internal pull-up resistor such that the input line has a defined state when nothing is connected. Now we enter an endless loop (while(1)) and check the state of the button all the time. bit_is_clear (PINB,PINB0) reads the input line and checks if zero volts are detected. This happens when the button is pressed.

i++; and i %= 6; is the counter which counts up to 5 and then starts again at zero. The rest of the code is just about the special dice like LED display.

Simple but it works very well. .... and it works even in the absence of gravity.



First tests, I got a 5!

Have fun and happy soldering!

References

- Software and future updates: **Download page for this article**
- A complete kit to build this programmer including documentation and the circuit diagram is available from shop.tuxgraphics.org