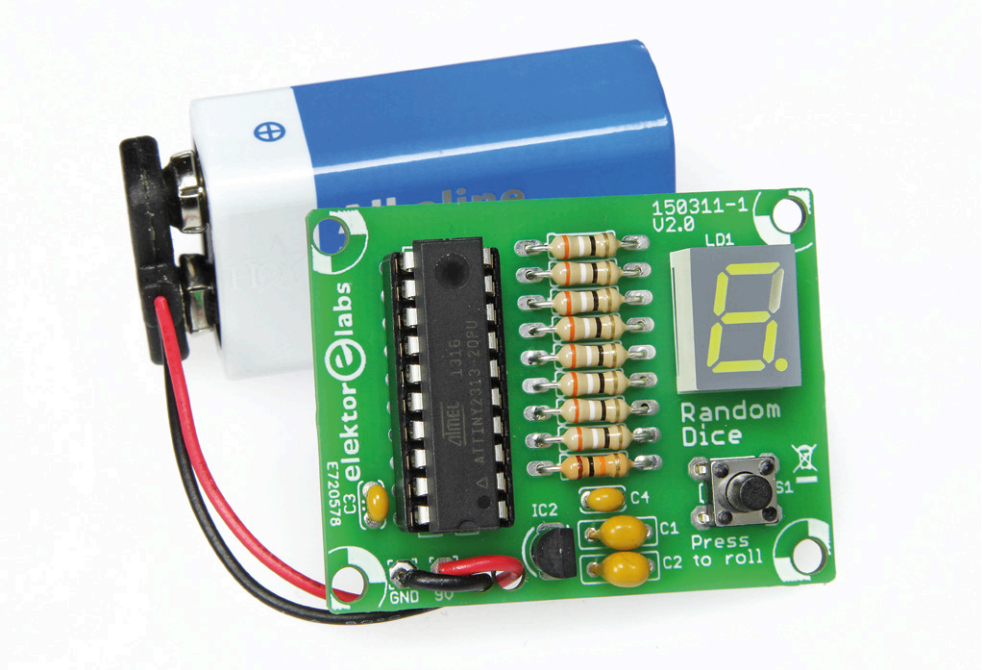
Tiny-Würfel

Elektronischer Würfel mit ATtiny2313



Von **Florian Schäffer** (D)

Dieser einfache elektronische Würfel ist ein ideales Einsteiger-Projekt für Jugendliche oder Junggebliebene, die zum ersten Mal ein heißes Eisen in Form eines Lötkolbens anfassen. Man kann sich bei diesem Projekt relativ gefahrlos darin üben, wie man eine Platine bestückt. Lohn der Mühe ist ein elektronischer Würfel, bei dem man nicht einmal die Augen zusammenzählen muss, da die geworfene Zahl gleich dezimal angezeigt wird.
Bequemer geht's nicht!

Unsere Schaltung simuliert einen Würfel. Auf Tastendruck startet das Würfeln mit der Anzeige verschiedener zufälliger Zahlen zwischen 1 und 6; nach einer Weile bleibt eine der Zahlen auf dem LED-Display stehen. Zwecks Anfängerkompatibilität sind SMDs komplett verbannt und es kommen ausschließlich bedrahtete Bauteile vor.

Würfelschaltung

Wie in **Bild 1** zu sehen, sind ein Taster und ein Siebensegment-LED-Display mit den I/O-Pins eines Mikrocontrollers des Herstellers Atmel verbunden. Mit R1...R8 begrenzen acht Widerstände den Strom für die LEDs. Ein einfacher Festspannungsregler erzeugt aus der 9-V-Batteriespannung stabile 3 V für die Schaltung. C4 dient der Tastenentprellung. Mit dem Taster wird gewürfelt. Nutzt man die Schaltung nicht, dann wird das Display nach kurzer Zeit dunkel, was die Batterie schont. Die LED des Dezimalpunktes dient als Betriebsanzeige.
Der Würfel ist als Bausatz im Elektor-Shop erhältlich [1]. Hier liegt bereits ein programmierter Controller bei. Als Taktgeber des Controllers wird der integrierte R/C-Oszillator mit 8 MHz verwendet. Der reale Takt beträgt durch die interne Teilung 1:8 nur 1 MHz. Ein externer Quarz als Taktgeber ist also überflüssig. Der Pull-up-Widerstand R9 sorgt dafür, dass der hochohmige Reset-Eingang des Mikrocontrollers zuverlässig auf „high" (=inaktiv) liegt.

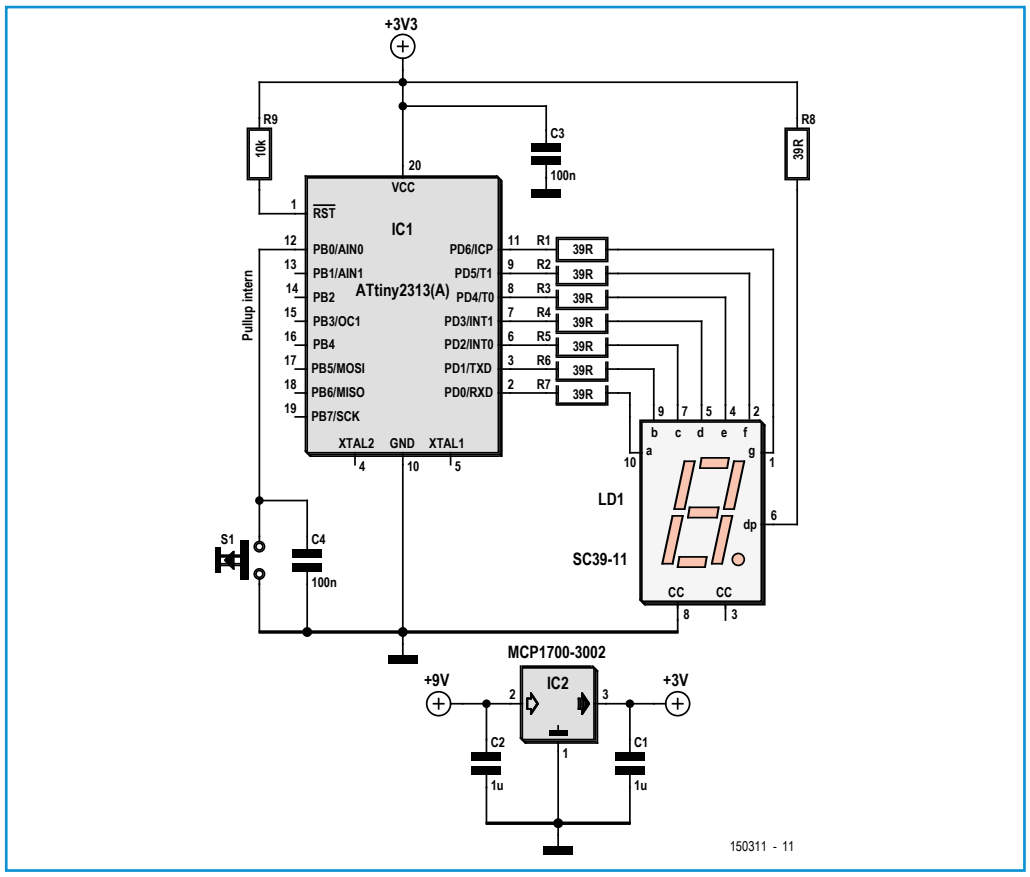


Bild 1. Schaltung des elektronischen Würfels mit Atmel-Mikrocontroller.

Quellcode

Das Würfel-Programm in **Listing 1** wurde mit der kostenlosen Toolchain WinAVR [2] bzw. dem dort enthaltenen Compiler GCC erstellt und für den Controller-Typ ATtiny2313A kompiliert. Die Software funktioniert so: Da ein Mikrocontroller keinen echten Zufallszahlengenerator beinhaltet, wird ein schneller, permanent durchlaufender Zähler (8-bit-Timer) benutzt, um Pseudo-Zufallszahlen zu erzeugen.

Jeder Tastendruck löst im Programm zwei Interrupts aus (Drücken und Loslassen). Bei jedem zweiten Interrupt starten wir eine Schleife, die (in immer größer werdenden Abständen) Zahlen von 1...6 anzeigt und so das Würfeln simuliert. Nach Durchlaufen der Schleife wird der Stand des ständig im Hintergrund laufenden Zählers abgefragt und in eine Zahl von 1...6 umgerechnet (Rest beim Teilen durch 6 ermitteln, plus 1). Das Ergebnis wird dauerhaft angezeigt – dies ist der gewürfelte Wert. Da man nicht weiß, wo der Zähler gerade beim Betätigen des Tasters steht und einen bestimmten Wert bei dem schnelllaufenden Zähler auch gar nicht treffen würde, ist ein Schummeln unmöglich.

Im Code sind die LEDs des Displays den einzelnen Zahlensymbolen zugeordnet (0…9, nur 1…6 werden benötigt). Durch Änderungen im Programm könnten aber auch andere Zahlen oder gar Buchstaben dargestellt werden.

Programmieren

Wie schon erwähnt, liegt dem Bausatz ein programmierter Controller bei. Doch natürlich kann man diesen neu programmieren – etwa mit einem geänderten Programm. Nachdem der Code mit einem Editor erstellt wurde, wird er in Maschinensprache übersetzt (kompiliert). Die resultierende Hex-Datei wird mit einem geeigneten Programm wie etwa AVRDUDE [3] in den Flash-Speicher des Mikrocontrollers geschrieben. Hierzu braucht man aber noch etwas Hardware. Neben einem geeigneten Programmer wie zum Beispiel dem Atmel AVR-ISP MK2 wird noch ein kleiner Programmier-Adapter mit IC-Sockel benötigt. Diesen kann man sich schnell auf einem Stückchen Lochraster-Platine aufbauen (**Bild 2**). Mit dieser Hardware kann man den Mikrocontroller dann immer wieder neu überschreiben. Ein Thema beim Programmieren von

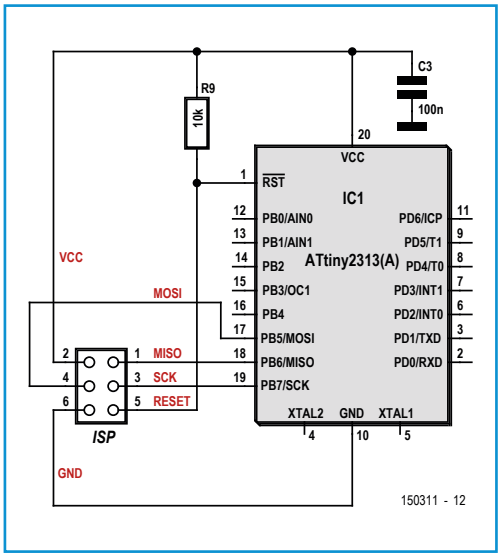


Bild 2. Kleiner Programmier-Adapter mit sechspoligem ISP-Anschluss.

Mikrocontrollern sind auch die Fuses, mit denen man den Controller konfiguriert (zum Beispiel, dass wie erwähnt der interne Oszillator verwendet wird). Für dieses Projekt nutzt man aber die Standard-Einstellungen und muss sich deshalb nicht darum kümmern.

Aufbau

Löten: Falls die Temperatur des Lötkolbens einstellbar ist, stellt man sie auf rund 350...370 °C bei klassischem bleihaltigem Lot (das für solche DIY-Projekte zuverlässig ist) oder bei bleifreiem Lötzinn auf etwa 380...400 °C ein.

1. Bauteil auf Bestückungsseite der Platine einsetzen. Siehe hier den Bestückungsplan bei der Stückliste.
2. Mit der Lötspitze ca. ½ Sekunde das Lötpad aufheizen und den Anschlussdraht erhitzen.
3. Lötzinn von der anderen Seite zuführen.
4. Wenn das Lötzinn geschmolzen ist (ca. 1 Sekunde), Lötzinn entfernen und anschließend erst den Lötkolben.
5. Lötstelle prüfen.
6. Überstehenden Anschlussdraht abschneiden.

Bestückungsreihenfolge: Die Bauteilanschlüsse werden mit der Flachzange passend gebogen – den Draht nicht zu dicht am Gehäuse biegen.

1. Widerstände (Ausrichtung der Farbringe von links nach rechts).
2. Kondensatoren (Werte beachten).
3. Taster.
4. IC-Sockel (Ausrichtung der Markierung Pin 1 beachten).
5. Siebensegment-Display (Orientierung des Dezimalpunktes beachten).
6. IC 2 (Ausrichtung beachten). Das Bauteil nicht zu weit auf die Platine drücken, sondern etwa 5 mm Luft zwischen Platine und Bauteilunterseite belassen. Ansonsten treten große Biegekräfte auf, die das Bauteil intern beschädigen können.
7. Batterie-Clip (Kabel vorher durch die Löcher zur Zugentlastung führen; Polung: Schwarz = GND, Rot = 9 V).

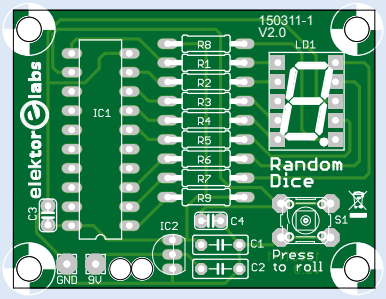
Stückliste

Widerstände:
R1..R8 = 39 Ω, 5 %, ¼ W
R9 = 10 k

Kondensatoren:
C1,C2 = 1 µ, Folie, RM 5 mm
C3,C4 = 100 n, keramisch, RM 1/10“

Halbleiter:
IC1= ATtiny2313A, programmiert
IC2 = MCP1700
LD1 = 7-Segment-LED SC39

Außerdem:
S1 = Schließer, Kurzhubtaster
Batterie-Clip für 9-V-Batterie
IC-Fassung für IC1, 20-pol.
Platine 150311-1 v1.0



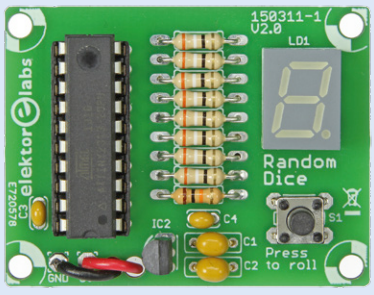


Bild 3. Die fertig bestückte Würfelplatine.

Endmontage: Nach der Bestückung prüft man alle Lötstellen. Wurden welche vergessen oder gibt es gar kalte Lötstellen – oder noch schlimmer: Kurzschlüsse durch Lötzinnreste oder Drähte? Anschließend lohnt ein Kontrollblick auf die Polung von Batterie-Clip und IC2. Die fertige Platine sollte so aussehen wie in **Bild 3**.
Nun schließt man eine Batterie an, ohne dass IC1 in der Fassung steckt. Jetzt muss der Dezimalpunkt leuchten oder die Lieblingsbeschäftigung des Elektronikers startet: die Fehlersuche. Ist alles soweit okay, geht es weiter: Entfernen Sie die Batterie.
Setzen Sie das programmierte IC1 vorsichtig und richtig herum auf den Sockel. Pin 1 (Kerbe/Markierung) zeigt zu den Batteriekabeln.
Prüfen Sie, ob alle Pins sauber über den Löchern des Sockels sitzen. Wenn nicht, korrigieren Sie die Ausrichtung etwas.
Drücken Sie das IC vorsichtig in den Sockel, gleichmäßig mit zwei Fingern und etwas Kraft, bis es bündig sitzt.
Schließen Sie die Batterie wieder an.
Drücken Sie auf den Taster – es geht los!

Viel Spaß mit diesem Projekt!

(150311)

Weblinks

[1] Webseite zu diesem Projekt: www.elektormagazine.de/articles
[2] WinAVR: http://sourceforge.net/projects/winavr/
[3] AVRDUDE: www.nongnu.org/avrdude/

Listing 1.

```
#include <avr/io.h>
#include <util/delay.h>       // definiert _delay_ms()
#include <avr/interrupt.h>    // IRQ Behandlung

int main (void);

volatile uint8_t roll=0;
volatile uint16_t zeit=0;
const int8_t numbers [10] =   // 0..9: Zuweisung der Ports binär zu den Segmenten. High-Aktiv
{
/*      A
      F   B
        G
      E   C
        D         */
   0b00111111,    // 0
   0b00000110,    // 1
   0b01011011,    // 2
   0b01001111,    // 3
   0b01100110,    // 4
   0b01101101,    // 5
   0b01111101,    // 6
   0b00000111,    // 7
   0b01111111,    // 8
   0b01100111,    // 9
};

/**
```

```
   @brief   IRQ Routine wird bei IRQ an PCINT aufgerufen. Lässt den Würfel rollen
*/
ISR (PCINT_B_vect)
{
   uint8_t i=0;
   roll++;        // wie oft wurde IRQ bereits ausgelöst? Da PCINT nur Toggle erkennen kann,
                  // wird pro Tastendruck zweimal IRQ ausgelöst.
                  // Wir wollen aber nur einmal, also jedes 2. Mal würfeln.
   if (roll == 2)
   {
      roll=0;                           // wieder neu zählen
      for (i=1; i < 40; i++)            // simuliert Würfeln
      {
            PORTD = numbers[(i % 6)+1];   // Ausgabe. Counter Modulo 6 = 0-5  => +1 = 1-6
            _delay_ms(i*3);
      }
      PORTD = numbers[(TCNT0 % 6)+1];     // Ausgabe. Timerwert Modulo 6 = 0-5  => +1 = 1-6
      zeit=0;                           // Zähler bis LEDs aus zurücksetzen
   }
}

/**
   @brief   Hauptroutine
   @param   none
   @return  Ende-Status
*/
int main(void)
{
   PORTD = 0;              // PORTD komplett aus
   DDRD = 0xFF;            // PORTD komplett als Ausgang
   DDRB &= ~(1 << DDB0);   // B0 Eingang
   PORTB |= (1 << PB0);    // Pull Up aktiv

   TCCR0B = (1 << CS02) | (1 << CS00); // 8 Bit Timer, Prescaler CLK/1024 => 1.000.000/256 = 3,9 kHz
                                       //  => 0,000256 s/Impuls => x256 > alle 0,065 s Überlauf
   GIMSK |= (1 << PCIE);      // PCIE IRQs zulassen
   PCMSK |= (1 << PCINT0);    // IRQ an PB0 zulassen
   sei();                     // IRQs ein

   while (1)  // endlos
   {
      // Schleife, die in x Sekunden das Display ausschaltet
      for (zeit=0; zeit <=300; zeit++) // 300x100=30.000 ms = 30 Sekunden
         _delay_ms(100);

      PORTD = 0;  // Alle Segmente aus.
   }
   return 1;      // nie
}
```