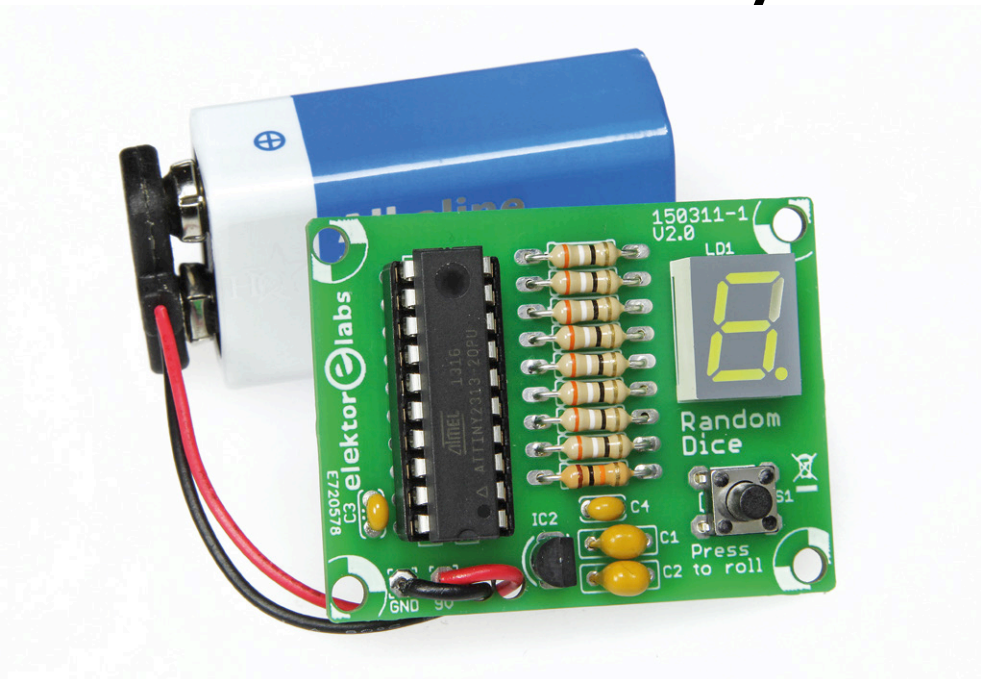
dé-Tiny

simulateur de dé avec ATtiny2313



Florian Schäffer (D)

Ce simulateur de dé électronique est un projet idéal pour jeunes et moins jeunes qui doivent recevoir le baptême du feu munis d'un simple fer à souder. En effet, dans ce projet, les risques liés au montage des composants électroniques sont presque nuls. Le jeu en vaut la chandelle.
Il n'est même pas nécessaire de compter les points puisque le résultat est affiché directement sous forme de chiffre. Que voulez-vous de plus ?

Notre circuit permet de simuler le lancer d'un dé. Il suffit d'appuyer sur le bouton-poussoir pour faire défiler des chiffres aléatoires entre 1 et 6, avant qu'un chiffre reste affiché.

Les composants montés en surface (CMS) ont été proscrits et remplacés exclusivement par des composants traversants pour faciliter le travail aux débutants.

Circuit du dé

La **figure 1** montre un bouton-poussoir et un afficheur à sept segments reliés aux broches d'E/S d'un microcontrôleur Atmel. Les résistances R1 à R8 limitent le courant qui traverse les LED. Un régulateur de tension fixe délivre au circuit une tension stable de 3 V, à partir d'une pile de 9 V. Le condensateur C4 constitue un filtre anti-rebonds. Le bouton-poussoir permet de « lancer » le dé. L'afficheur s'assombrit dès qu'on ne se sert plus du circuit afin de préserver la pile. La LED du point décimal sera un témoin de fonctionnement.

Ce dé est disponible dans la boutique Elektor sous forme d'un kit [1]. On y trouve également le contrôleur programmé. On utilise l'oscillateur R/C intégré à 8 MHz pour cadencer le micro. Le diviseur interne (1:8) réduit la cadence effective à 1 MHz, ce qui rend superflue l'adjonction d'un quartz externe. La résistance de rappel vers le haut R9 force à l'état haut (= inactif) l'entrée d'initialisation (haute impédance) du microcontrôleur de manière fiable.

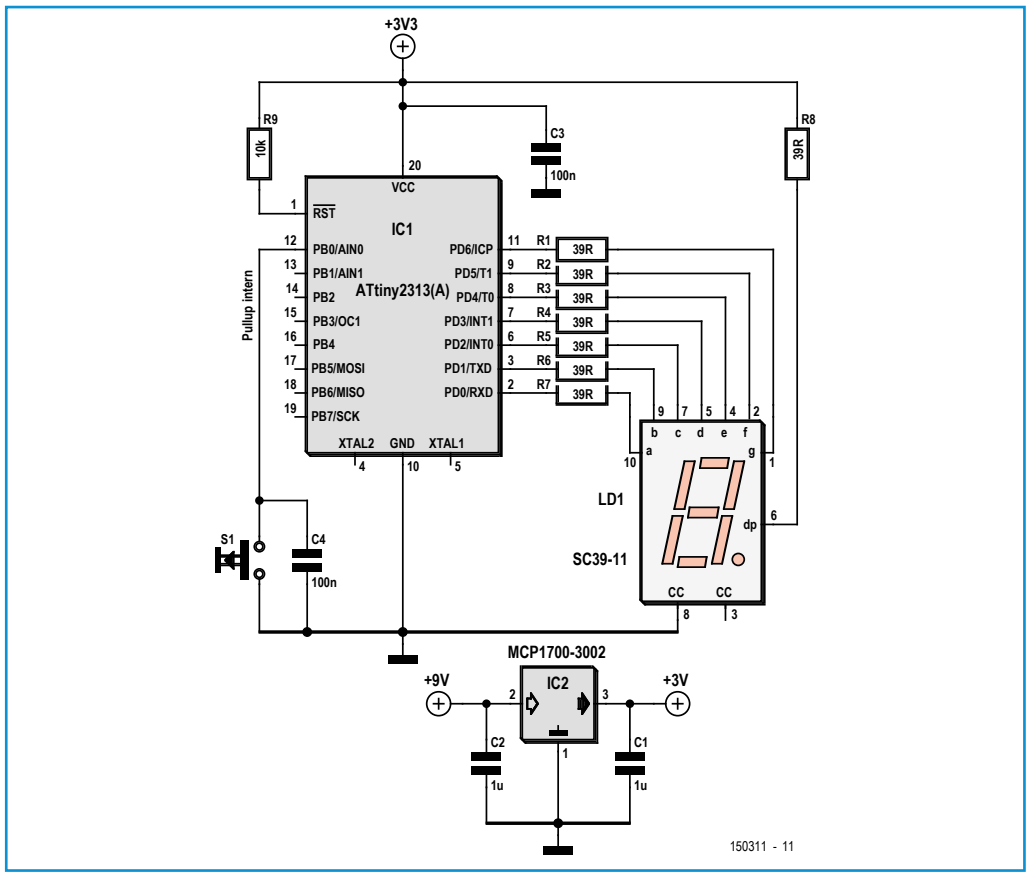


Fig. 1 Circuit du simulateur de dé avec microcontrôleur Atmel.

Code source

Le programme du dé (**listage 1**) a été créé avec la suite d'outils gratuite WinAVR [2] et le compilateur GCC (inclus dans WinAVR), puis compilé pour le contrôleur de type ATtiny2313A (modèle basse consommation). Le logiciel fonctionne comme suit : à vrai dire, un microcontrôleur ne fournit aucun nombre aléatoire, c'est un compteur ultrarapide (*timer* sur 8 bits) qui tourne en permanence pour délivrer des nombres pseudo-aléatoires.

À chaque fois que vous appuyez sur le bouton-poussoir, vous déclenchez deux interruptions dans le programme (pression et relâchement). Au bout de deux interruptions, une boucle est démarrée, elle affiche des chiffres compris entre 1 et 6 à des intervalles toujours plus grands, ce qui permet de simuler le lancer d'un dé. Une fois que la boucle est terminée, le système consulte l'état du compteur qui fonctionne en permanence en tâche de fond et convertit le résultat en un chiffre compris entre 1 et 6 (reste de la division par 6, plus 1). Le résultat qui reste affiché correspond à la valeur du dé. Impossible de tricher puisqu'on ne sait pas où se trouve le compteur au moment d'appuyer sur le bouton, d'autant plus que le compteur défile à grande vitesse.

Dans le code, les LED de l'afficheur sont affectées aux différents symboles numériques (de 0 à 9, toutefois seuls les chiffres compris entre 1 et 6 sont nécessaires). Pour obtenir d'autres chiffres voire même des lettres, il suffirait de modifier le programme.

Programmation

Le kit est doté d'un contrôleur déjà programmé, mais vous pouvez le reprogrammer, si vous le souhaitez. Une fois modifié dans un éditeur, le code est compilé, c'est-à-dire traduit en « langage machine ». Le fichier hexadécimal qui en résulte est écrit dans la mémoire flash du microcontrôleur à l'aide d'un programme adéquat comme AVRDUDE [3]. Cette écriture nécessite néanmoins du matériel supplémentaire. En effet, en plus d'un programmateur adéquat du genre Atmel AVR-ISP MK2, vous aurez besoin d'un adaptateur doté d'un support de circuit intégré. Cet adaptateur se monte rapidement sur un bout de plaque d'essai (**fig. 2**). Ce matériel

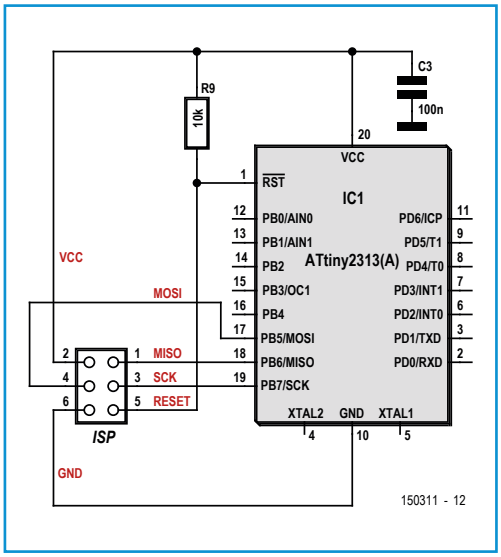


Fig. 2 Petit adaptateur de programmation doté d'un connecteur ISP à six broches.

permet de reprogrammer le microcontrôleur à tout moment. Quand on programme des microcontrôleurs, la gestion des fusibles pour configurer le contrôleur (p. ex. pour exploiter l'oscillateur interne mentionné plus haut) est parfois complexe. Rassurez-vous, dans ce projet, il est inutile de vous en occuper puisqu'on se sert de la configuration standard.

Assemblage

Soudage : si votre fer le permet, réglez la température entre 350 et 370 °C pour une soudure au plomb classique (ce qui est autorisé pour un projet maison) ou entre 380 et 400 °C pour une soudure sans plomb.

1.Insérez un composant dans le circuit imprimé (cf. schéma d'implantation à côté de la liste des composants).

2. Avec la panne du fer, préchauffez le trou pendant une demi-seconde environ, puis chauffez la patte du composant.

3. Appliquez la soudure de l'autre côté du circuit imprimé.

4. Une fois que la soudure a fondu (au bout d'une seconde environ), retirez le fil à souder avant d'enlever le fer à souder.

5. Vérifiez le point de soudage.

6. Coupez la broche si elle dépasse, une fois soudée.

Ordre d'implantation : pliez les broches des composants à l'aide d'une pince plate, mais ne les ramenez pas trop près du boîtier.

1. Résistances (anneaux, de gauche à droite).
2. Condensateurs (attention aux valeurs).
3. Bouton-poussoir.
4. Support du circuit intégré (attention à la position de la broche 1).
5. Afficheur à sept segments (attention à l'orientation du point décimal).
6. Circuit intégré IC2 (respectez le sens d'insertion). Évitez de mettre le boîtier contre le circuit imprimé, car ce composant a tendance à chauffer, un espace de 5 mm environ entre le circuit intégré et le circuit imprimé suffit. L'échauffement excessif du circuit inté-

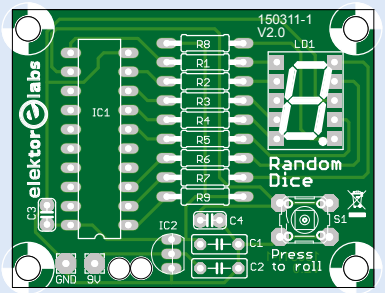
Liste de composants

Résistances :
R1 à R8 = 39 Ω, 5 %, ¼ W
R9 = 10 kΩ

Condensateurs :
C1, C2 = 1 µ, film, au pas de 5 mm
C3, C4 = 100 n, céramique, au pas de 2,54 mm

Semi-conducteurs :
IC1 = ATtiny2313A programmé*
IC2 = MCP1700
LD1 = LED à 7 segments SC39

Divers :
S1 = contact NO, bouton-poussoir à faible course
Connecteur pour pile de 9 V
Support de circuit intégré pour IC1, 20 broches
Circuit imprimé (150311-1)*
*Circuits imprimés, modules assemblés et composants programmés : www.elektor.fr



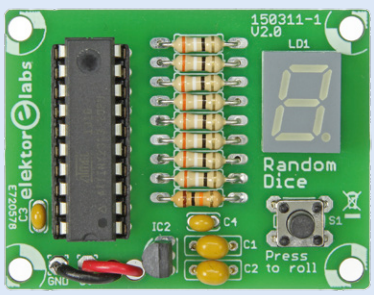


Fig. 3 Le dé terminé.

gré pourrait le déformer et en perturber le fonctionnement.
7. Connecteur pour la pile (passez tout d'abord les fils dans les trous pour réaliser un système d'anti-traction ; polarité : noir = GND, rouge = 9 V).

Assemblage final : une fois le câblage terminé, vérifiez que vous n'avez oublié aucun point de soudage, qu'il n'y a pas non plus de points de soudage froids, voire pire des restes de soudure ou de fil sur le circuit imprimé qui pourraient causer des courts-circuits. Pour finir, vérifiez la polarité du connecteur de la pile et du circuit intégré IC2. Une fois terminé, le circuit imprimé devra ressembler à celui de la **figure 3**.
Raccordez maintenant une pile, sans que le circuit intégré IC1 soit enfiché. Le point décimal devrait s'afficher, sinon c'est le passe-temps favori de l'électronicien qui commence : trouver l'erreur. Si tout va bien, procédez comme suit :

1. Continuez en retirant tout d'abord la pile.
2. Placez le circuit intégré IC1 sur son support, dans le bon sens et avec délicatesse. La broche 1 (encoche/marquage) doit pointer vers les fils de la pile.
3. Vérifiez la position des broches dans le support. Corrigez si nécessaire.
4. Enfoncez délicatement le circuit intégré dans le support en vous servant de deux doigts pour bien répartir la force de pression, mais sans trop forcer.
5. Reliez maintenant à la pile.
6. Appuyez sur le bouton-poussoir, et c'est parti !

Amusez-vous bien !

(150311 – version française : Pascal Duchesnes)

Liens

[1]www.elektormagazine.fr/150311

[2] WinAVR :
http://sourceforge.net/projects/winavr/

[3] AVRDUDE : www.nongnu.org/avrdude/

Listage 1.

```
#include <avr/io.h>
#include <util/delay.h>       // définit _delay_ms()

int main (void);

volatile uint8_t roll=0;
volatile uint16_t zeit=0;
const int8_t numbers [10] =   // 0 à 9 : affectation en binaire des ports aux segments
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  Routine IRQ appelée si demande d'interruption de PCINT. « Lancement du dé »
*/
ISR (PCINT_B_vect)
{
   uint8_t i=0;
   roll++;        // IRQ a été déclenchée combien de fois ? Comme PCINT ne reconnaît
                  // que les basculements, deux IRQ sont déclenchées à chaque
                  // pression du bouton-poussoir.
                  // En revanche, nous ne lancerons le dé qu'une fois sur deux.
   if (roll == 2)
```

```
    {
       roll=0;                              // compter à nouveau
       for (i=1; i < 40; i++)               // simule le lancement du dé
       {
             PORTD = numbers[(i % 6)+1];    // Affichage. Compteur modulo 6 = 0-5
                                            // => +1 = 1-6
             _delay_ms(i*3);
       }
       PORTD = numbers[(TCNT0 % 6)+1];      // Affichage. Timer modulo 6 = 0-5
                                            // => +1 = 1-6
       zeit=0;                              // RAZ compteur -> LED éteintes
    }
}

/**
   @brief   routine principale
   @param   aucun
   @return  état final
*/
int main(void)
{
   PORTD = 0;              // PORTD entièrement inactif
   DDRD = 0xFF;            // PORTD entièrement en sortie
   DDRB &= ~(1 << DDB0);   // B0 en entrée
   |= (1 << PB0);          // Rappel vers le haut actif

   TCCR0B = (1 << CS02) | (1 << CS00); // Timer 8 bits, prédiviseur CLK/1024
                                       // => 1.000.000/256 = 3,9 kHz
                                       // => 0,000256 s/Impulsion => x256
                                       // > débordement toutes les 0,065 s
   GIMSK |= (1 << PCIE);               // Autoriser IRQ PCIE
   PCMSK |= (1 << PCINT0);             // Autoriser IRQ sur PB0
   sei();                              // IRQ activées

   while (1)                           // sans fin
   {
      // Boucle qui éteint l'affichage au bout de x secondes
      for (temps=0; temps <=300; temps++)       // 300x100 = 30.000 ms = 30 s
         _delay_ms(100);

      PORTD = 0;              // Tous les segments sont éteints.
   }
   return 1;                  // jamais
}
```