42 testeur de servos

Stefano Purchiaroni (Italie)

Voici un circuit utile et peu coûteux pour tester tout type de servomoteur, avec une consommation maximale de 1 A.

Le schéma du circuit de ce projet (voir **figure 1**) est minimaliste - la plupart du travail est effectué par le logiciel. Le circuit utilise un microcontrôleur PIC16LF1554 de Microchip fonctionnant avec une horloge interne de 16 MHz, et un régulateur de tension 7805. Pour éviter une surchauffe, il est conseillé de ne pas l'alimenter avec une tension trop élevée. Le réglage des positions des axes de quatre servomoteurs est possible grâce à quatre potentiomètres de 10 kΩ (RV1...RV4).

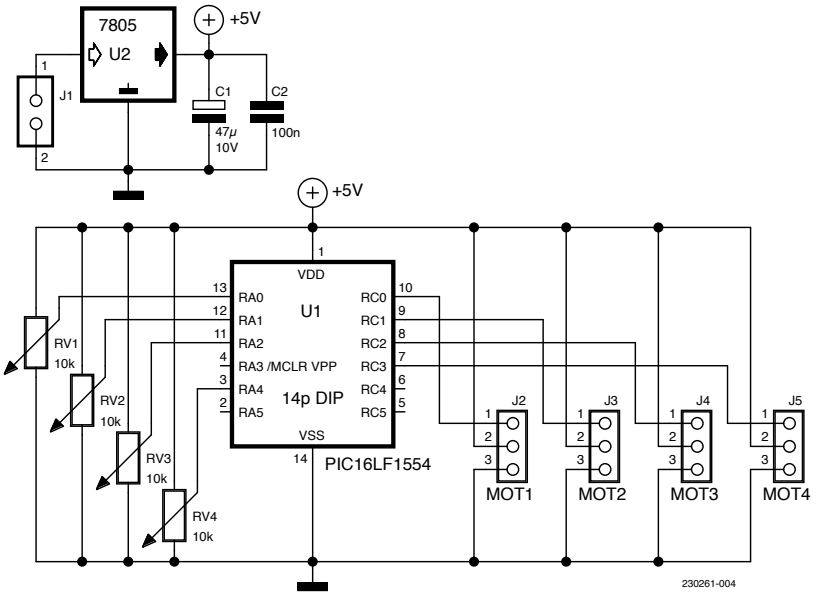

Figure 1. Schéma de principe du testeur de servo.

Logiciel

La logique de commande est assurée par un programme écrit avec le compilateur mikroC PRO pour PIC (version v5.6 ou supérieure). Vous pouvez voir le code dans le **listage 1** ; il est également disponible en téléchargement [1]. Le code est facilement portable sous d'autres compilateurs, mais la valeur que j'ai trouvée expérimentalement pour la fonction `delay_us()` nécessite une révision. Les valeurs du potentiomètre, mesurées par le CA/N, sont attribuées aux valeurs internes 0...99, ce qui détermine la position du servo. Le tableau `img` contient la durée des impulsions à envoyer aux servos. Il est rempli par une chaîne de valeurs de " 1 " d'une longueur de 0 à 99. En fait, il y a quatre chaînes individuelles codées à différentes positions de bits – chacune représente une broche/canal/servo.
Nous envoyons d'abord une impulsion fixe d'environ 0,5 ms au servo, puis nous continuons avec des impulsions de durées variables en 100 étapes, déterminées par les valeurs " 1 " dans le tableau `img` :

```
// Set all outputs to High
PORTC = 0b11111111;

// Fixed first pulse at High
delay_us(500);

 // Playback of prepared sequence
for (i = 0; i < 100; i++) {
  PORTC = img[i];

  // Value from experimentation
  delay_us(14);
 }

// Turn off all motor pulses
// after playback
PORTC = 0;
```

La durée maximale de l'impulsion est d'environ 2,5 ms. Il est vrai que les fiches techniques des servos indiquent un intervalle différent pour couvrir toutes les positions du servo : 1 à 2 ms, mais dans la pratique, on constate que l'extension maximale du servo est obtenue en élargissant cet intervalle de fonctionnement. Il est toujours possible, cependant, d'utiliser les potentiomètres dans une plage plus limitée si nécessaire.

J'ai publié une vidéo de démonstration sur YouTube [2].

Quelques remarques

Il est possible d'adapter le code à d'autres microcontrôleurs dotés de plus de broches, afin de commander davantage de servomoteurs.

Il est possible de simplifier le schéma de câblage (**figure 1**) en utilisant un circuit imprimé (disponible dans les téléchargements). Veillez à adapter le circuit imprimé aux mesures données dans la **figure 2** avant de passer à l'étape de gravure.

230261-04

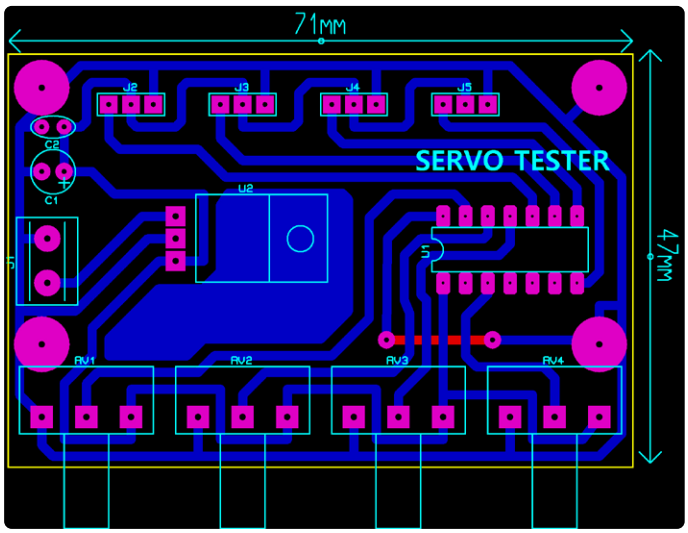

Figure 2. Mettez à l'échelle le tracé en cuivre du circuit imprimé ([1]) aux dimensions indiquées ici.

Produits

- **FeeTech FS90 Micro Servo avec accessoires**
  https://elektor.fr/19788

LIENS

[1] Téléchargements pour cet article : https://elektormagazine.fr/230261-04
[2] Vidéo de démonstration : https://youtu.be/LFJw72H-8GI

Listage 1.

```
//=====================================================================
//
//  Servo Tester: drive up to 4 Servo Motors              SPU 12.2022
//
//  It generates a 50 Hz pulse sized 0.5-2.5 ms by manual trimpot
//  on four independent channels.
//
//  By SPU (info@purchiaroni.com) - Rome (IT)
//
//     MCU Model:       PIC 16LF1554
//     Oscillator:      Internal 20 MHz
//     Compiler:        MikroC Pro 5.61 or higer
//
//  Email me for schematic and documents: info@purchiaroni.com
//  Homepage: www.purchiaroni.com
//
//  Changes log
//  14.12.2022 - Code creation
//=====================================================================
// Timer1 preload value calculated for interrupt each 20ms (50 Hz pulses)
#define TMR1H_INI   0x63
#define TMR1L_INI   0xC0
// Various
#define OFF         0
#define ON          1
#define FALSE       0
#define TRUE        1
unsigned short img[100];    // Data for channels playback (max 8 channels)
int t1, t2, t3, t4;         // Times for pulses width
```

```
void TrimPots_Read() {
  // Map 0..1023 to 0..100 steps
  // (101, 102 values truncated by "for" cycles during playback)
  t1 = ADC1_Read(0) / 10;
  t2 = ADC1_Read(1) / 10;
  t3 = ADC1_Read(2) / 10;
  t4 = ADC1_Read(10) / 10;
 }

//*********************************************************************
// Interrupt management procedure
//*********************************************************************
void interrupt() {
  // Interrupt Service Routine. Called on any interrupt
  int i;       // Generic local variable
  // ---TMR1---: Manage Timer1 overflow (each 0.1s)
  // to update remaining time
  if (PIR1.TMR1IF == TRUE) {
    // Keep Timer1 running
    TMR1H         = TMR1H_INI;   // Reload counter high byte
    TMR1L         = TMR1L_INI;   // Reload counter low byte
    PIR1.TMR1IF   = FALSE;       // Clear TMR1 Interrupt flag

    PORTC = 0b11111111;          // Set all outputs to High
    delay_us(500);               // Fixed initial pulse at High level
    for (i = 0; i < 100; i++) {  // Play back the prepared sequence
      PORTC = img[i];
      delay_us(14);              // Value derived by experimentation...
    }

    PORTC = 0;         // Turn off all motor pulses after playback
    TrimPots_Read();   // Update the potentiometers' readings
    // Prepare a new sequence on the basis of last readings
    for (i = 0; i < 100; i++) {
      img[i] = 0;
      if (i <= t1) img[i] |= 0b0001;
      if (i <= t2) img[i] |= 0b0010;
      if (i <= t3) img[i] |= 0b0100;
      if (i <= t4) img[i] |= 0b1000;
    }
  }
}

void StartTimer() {
  // Timer1 Registers: Prescaler=1:2; TMR1 Preset=25536;
  // Freq=50,00Hz; Period=20,00 ms
  T1CON = 0b00010101;
  T1GCON = 0;
  // Settings for interrupt management
  INTCON.GIE    = TRUE;                // Global Interrupt Enable
  INTCON.PEIE   = TRUE;                // Peripheral Interrupt Enable
  // Start the Timer1 and then the Countdown
  TMR1L = TMR1L_INI;                   // preset for timer1 LSB register
  TMR1H = TMR1H_INI;                   // preset for timer1 MSB register
  PIE1.TMR1IE  = TRUE;                 // Timer1 Interrupt enable
  PIR1.TMR1IF  = FALSE;                // Clear TMR1 Interrupt flag
}

//*********************************************************************
// MAIN
//*********************************************************************
void main() {
  int i;                              // Generic local variable
  // Oscillator and ports settings
  OSCCON = 0b01111000;                // Internal clock 16 MHz
  ANSELA = 0b11111111;                // PORTA analog
  TRISA  = 0b11111111;                // PORTA input
  TRISC  = 0b00000000;                // PORTC output
  ADC1_Init();                        // Initialize ADC #1
  StartTimer();                       // Start TMR1
  for (i = 0; i < 100; i++) img[i] = 0; // Clear the data image
  do { delay_ms(1000); } while (1);     // Idle. Operations managed by ISR
}
```