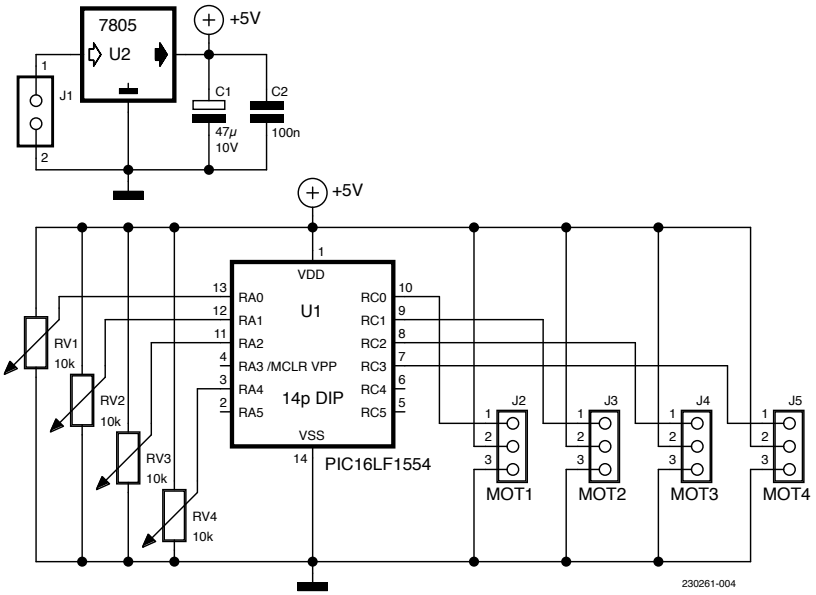
42 Servotester

Stefano Purchiaroni (Italië)

Dit is een praktische, goedkope schakeling voor het testen van servomotoren met een stroomverbruik van maximaal 1 A.

Het schema van dit project (zie **figuur 1**) is minimalistisch: het meeste werk wordt gedaan door de software. De schakeling werkt met een Microchip PIC16LF1554-microcontroller met een interne klok van 16 MHz en een 7805-spanningsregelaar. Gebruik geen te hoge voedingsspanning, om te voorkomen dat de regelaar te heet wordt. Er zijn vier 10kΩ-potmeters (RV1…RV4) waarmee we de stand van vier servo's kunnen regelen.



Figuur 1. Het schema van de servotester.

Software

De besturing wordt verzorgd door een programma in mikroC PRO voor PIC (versie v5.6 of hoger). De code is te zien in **listing 1**; deze kan ook worden gedownload [1]. De code is gemakkelijk te porten naar andere compilers, maar de waarde die ik experimenteel heb bepaald voor functie `delay_us()` moet dan misschien worden aangepast.

De stand van de potmeters worden ingelezen door de ADC en geschaald naar interne waarden 0…99 die de servopositie bepalen. Het array `img` bevat de tijdsduur van de pulsen die naar de servo's worden gestuurd. Het array wordt gevuld met een reeks van '1'-waarden met een lengte van 0 tot 99. Om precies te zijn: er zijn vier reeksen van enen, elk in een andere bitpositie. Elke reeks is bedoeld voor één van de pinnen/kanalen/servo's.

We sturen eerst een vaste puls van ongeveer 0,5 ms naar de servo, daarna gaan we door met een variabele pulslengte in 100 stappen, bepaald door het aantal enen in het array `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;
```

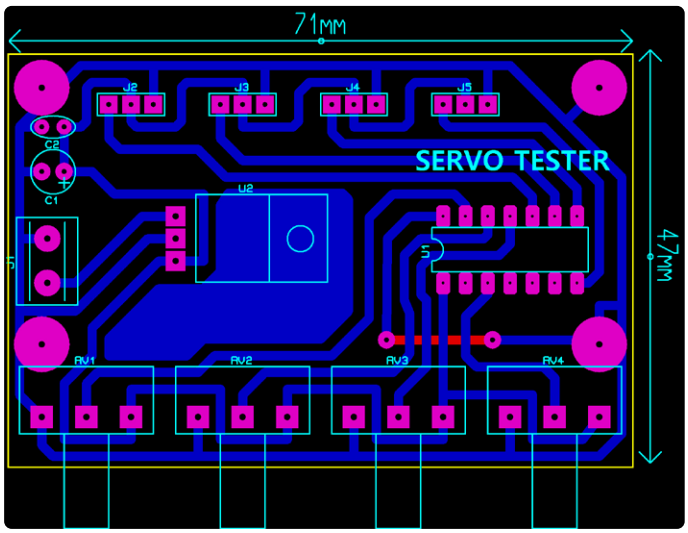
De maximale pulslengte bedraagt ongeveer 2,5 ms. Het klopt dat in de datasheets van servo's een andere pulslengte gespecificeerd wordt: 1 tot 2 ms. Maar in de praktijk blijkt toch vaak dat servo's een iets breder pulsbereik op prijs stellen. Natuurlijk is het altijd mogelijk om de potmeters niet helemaal tot de aanslag te draaien, als dat nodig blijkt.

Ik heb een demovideo op YouTube geplaatst [2].

Enkele opmerkingen

De code kan worden aangepast voor andere microcontrollers met meer pinnen, zodat nog meer servomotoren kunnen worden aangestuurd.
Het schema in figuur 1 kan worden gebouwd met behulp van een print die te vinden is in de downloads. Pas de afmetingen van de print aan tot wat in **figuur 2** te zien is voordat je gaat etsen.

230261-03 (vertaling: Evelien Snel)



Figuur 2. Schaal de print-layout [1] op naar de hier gegeven afmetingen.

Gerelateerd product

- **FeeTech FS90 Micro Servo with Accessories (SKU 19788)**
  www.elektor.nl/19788

WEBLINKS

[1] Projectpagina bij dit artikel: http://www.elektormagazine.nl/230261-03
[2] Demovideo: https://youtu.be/LFJw72H-8GI

Listing 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
}
```