Programmieren mit der BASIC Stamp II

Teil 5: Fernsteuerung

Diesen Monat lernt der BoE-Bot, auf die Signale einer Infrarot-Fernsteuerung zu reagieren.

Entwurf von Al Williams

1

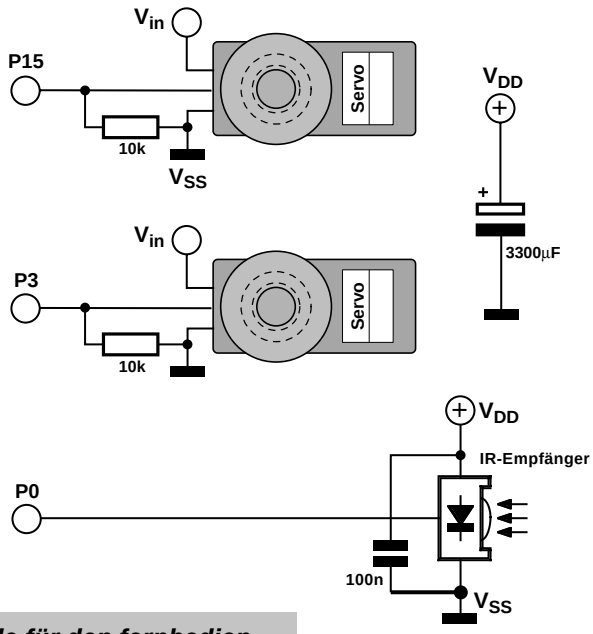

Bild 1. Bauteile für den fernbedienbaren BoE-Bot. Nur ein IR-Empfänger wird benötigt (Panasonic 4602).

Heutzutage gibt es kaum noch ein elektronisches Gerät, zu dem nicht eine Fernbedienung gehört. Was liegt also näher, als auch den BoE-Bot mit einer Infrarot-Fernbedienung auszustatten, zumal im vergangenen Kursteil der BoE-Bot mit IR-Sensoren ausgestattet ist, das Handling mit dem unsichtbaren Licht also aus dem ff beherrscht. Da die Hardware schon vorhanden ist, betrifft die Modifikation ausschließlich die Software. Dabei erfahren wir, wie die BASIC-Stamp Impulslängen misst und mit Arrays umgeht.

DETAILS

Die Roboter-Peripherie in **Bild 1** ist simpel und entspricht im Wesentlichen der des letzten Monats. Allerdings ist nur ein IR-Empfänger und überhaupt keine LED notwendig. Die Sende-LEDs befinden sich ja in der Fernbedienung. Zur Steuerung des BoE-Bots soll eine Fernbedienung mit einem gebräuchlichen Protokoll verwendet werden. Im Gegensatz zu den Elektor-Gepflogenheiten haben wir uns für den von Sony eingesetzten *Serial Infra Red Control Signal (SIRCS* oder *Control-S*) entschieden, da dies ein amerikanischer Kursus (Philips´ RC5 ist dort nahezu unbekannt) und das SIRCS-Protokoll im Internet gut dokumentiert ist (http://www.hut.fi/Misc/Electronics/docs/ir/ircodes.html). Wer über keine Sony-Fernbedienung (oder kompatible) verfügt, kann eine preisgünstige vorprogrammierte Fernbedienung einsetzen, wenn diese auf Sony-TV eingestellt ist.

Es gibt einige Gemeinsamkeiten mit anderen Fernbedienungsprotokollen. Die meisten nutzen eine Art von Impulsweitenmodulation. Beim Sony-Protokoll sendet die Fernbedienung ein Startbit (oft AGC-Impuls genannt), der relativ lang ist (über 2 ms). Dadurch kann der IR-Empfänger synchronisieren und seine interne Verstärkung automatisch einstellen.

Nach dem Startbit sendet die Fernbedienung eine Reihe von Impulsen. Ein 600 µs kurzer Impuls bedeutet eine logische null, ein 1200 µs langer Impuls eine logische eins. Nach einem Impuls folgt unabhängig vom Datum eine 600 µs lange Pause. Natürlich werden diese Zeiten nicht bis auf die Mikrosekunde eingehalten, sondern weichen in der Regel ein wenig von den Sollwerten ab.

IMPULSMESSUNG

Der Befehl PULSIN ermöglicht es der BASIC-stamp, Impulsbreiten zu messen. PULSIN benötigt drei Argumente. Das erste spezifiziert den Pin, an dem der Impuls erscheint. Die BASIC-stamp setzt den Pin automatisch als Eingang, wenn dies nicht schon geschehen ist. Das zweite Argument entscheidet, ob die Messung mit einer negativen (1) oder einer positiven Flanke (0) beginnt, also ob die Länge eines Low- oder eines High-Impulses ermittelt wird. Das letzte Argument ist eine Variable in Wort-Länge, die die ermittelte Impulslänge aufnimmt. Da die BASIC-stamp eine Zyklusbasis von 2 µs verwendet, muss der Variablenwert mit 2 µs multipliziert werden, um die Impulsweite zu erhalten. Nach 131 ms ($2^{16} \cdot 2$ µs) läuft der Zähler über und damit die Wartezeit ab . Erscheint innerhalt dieser Zeit keine relevante Flanke, setzt PULSIN die Variable auf null.
Der PULSIN-Befehl misst nur Impulse, wenn er die spezifizierte Flanke erkennt. Zum beispiel soll bestimmt werden, wie lange der Anwender einen Taster drückt, der in gedrückter Position eine null auf den Eingang der BASIC-stamp legt. Wenn der PULSIN-Befehl erst zu einem Zeitpunkt erfolgt, zu dem der Knopf schon gedrückt ist, ermittelt PULSIN keine Impulslänge.Denken sie bei der Anwendung von PULSIN immer daran, dass der Befehl vor der Flanke erfolgen muss, ansonsten erhalten sie falsche Ergebnisse.
Erwartet man ausschließlich kürzere Impulse als 512 µs ($2^8 \cdot 2$ µs), so kann man statt einer Wort- auch RAM-sparend eine Byte-Variable als drittes Argument einsetzen. Doch ob man nun eine Wort- oder eine Byte-Variable verwendet, der Zähler darf niemals überlaufen, da dies (ohne Vorwarnung!) ein falsches Ergebnis zur Folge hat. Da die Impulslängen des Fernbedienungsprotokolls 2 ms überschreiten, ist eine Wort-Variable angemessen. Angenommen, der IR-Sensor/Empfänger ist an Pin 0 angeschlossen, lässt sich die Länge eines IR-Impulses mit einem winzigen Programm ermitteln.

```
IRREAD:
   PULSIN 0,0,W1
   IF W1=0 THEN IRREAD
        'kein Impuls
   DEBUG ?W1
   GOTO IRREAD
```

Sie können mit diesem einfachen Programm feststellen, dass verschiedene Fernbedienungstasten auch unterschiedliche Impulsbreiten produzieren. Natürlich erfasst man damit nicht jedes Bit und das Resultat ist wenig aussagekräftig.
Das SIRCS-Protokoll umfasst (nach dem Startbit) 12 Informationsbits, die sich aus fünf Bits für den Gerätekode und anschließend sieben Bits für die Funktion (LSB zuerst) zusammensetzen. Wenn man sich ausschließlich für die Funktion interessiert, kann man auf die Auswertung des Gerätekodes verzichten. Im Prinzip lässt sich der Funktionskode durch ein kleines Programm ermitteln.

```
IRREAD:
   B0=0    ' Byte lesen
   B1=1    ' Bit-Maske
   PULSIN 0,0,W5
   IF W5<1200 then IRREAD
          ' kein Startbit
   FOR B2 = 1 to 7
     PULSIN 0,0,W5
     IF W5<400 THEN READZERO
     B0=B0+B1  ' ein Bit setzen
READZERO:
     B1=B1*2   ' Maske erhöhen
   NEXT
```

Dies sieht zwar aus wie ein recht ordentlicher Programmkode, funktioniert aber leider nicht.

Der erste PULSIN-Befehl liest das Startbit und ignoriert jeden Impuls, der nicht die richtige Länge besitzt. Trifft der Startimpuls ein, betritt das Programm eine Schleife, die jedes folgende Bit erfasst und ein korrespondierendes Bit in B0 auf 1 setzt, wenn die Impulslänge 800 µs überschreitet. Dies liegt deutlich über den 600 µs, die eine Null ausmachen, während eine Eins nominell 1200 µs entspricht, also deutlich mehr als 800 µs.
Das einzige Problem ist die nur 600 µs breite Lücke zwischen den Bits. Dies ist nicht genug Zeit für die BASIC-stamp, nach Verarbeitung des vorherigen Bits wieder bereit zu sein, um das nächste Bit zu lesen. Zum Vergleich: Die BASIC-stamp benötigt etwa 470 µs, um einen IF-Befehl zu bearbeiten. Manche Befehle benötigen mehr Zeit, manche weniger - gemittelt schlägt jede Befehlsausführung mit etwa 330 µs zu Buche. Selbst beim schnellsten Befehl reichen 100 µs zur Bearbeitung nicht aus. Muss die Impulslängenmessung beginnen, bevor im Programm der PULSIN-Befehl erreicht ist, ist der Messwert falsch.
Eine Option, mit dieser Problematik fertig zu werden, ist der Einsatz der BASIC-stamp IISX, die deutlich schneller ist als die ordinäre BASIC-stamp. Aber das gleiche Ziel lässt sich auch mit geschickter Programmierung erreichen.

DIE LÖSUNG

Um einen sauberen Programmablauf zu gewährleisten, muss man die Anzahl der Befehle zwischen den PULSIN-Befehlen reduzieren. Dies ist am einfachsten,. wenn man lediglich die PULSIN-Werte in unbearbeiteter, gewissermaßen roher Form speichert und später in aller Ruhe bearbeitet.
Für die Zwischenspeicherung der Rohdaten könnte man acht Wort-Variablen (eine für das Startbit und sieben für Datenbits) verwenden. Dies wäre allerdings sehr unübersichtlich und hätte einen unschönen Programmkode zur Folge. Wenn sie sich schon ein wenig mit Programmieren in Hochsprachen auskennen, dürfte ihnen als beste Lösung ein Array erscheinen. Glücklicherweise unterstützt die BASIC-stamp solche Arrays.

ARRAYS

Ein Array ist eine Möglichkeit, eine Gruppe ähnlicher Variablen unter einer Bezeichnung zusammenzufassen, die man mit *Index-Nummern* versieht, um sie voneinander unterscheiden zu können. Möchte man beispielsweise ein (eindimensionales) Array mit ungeraden Zahlen anlegen, schreibt man

```
oddnums var byte(5)
oddnums(0) = 1
oddnums(1) = 3
oddnums(2) = 5
oddnums(3) = 7
oddnums(4) = 9
```

Zu beachten ist dabei, dass das erste Feld eines Arrays immer mit dem Index 0 korrespondiert. Soll das Array fünf Felder besitzen, erhalten diese die Index-Nummern 0...4. Verwendet man andere Nummern, so riskiert man, für andere Variablen reservierten Speicherplatz zu überschreiben.
Arrays werden am besten in Schleifen behandelt. Um den Inhalt des obigen Arrays zu drucken, kann man schreiben:

```
I var byte
for I = 0 to 4
  Debug ?oddnums(I)
next
```

Natürlich benötigen Arrays wie andere Variablen Speicherplatz. Arrays schaffen keinen zusätzlichen Speicher, sondern sorgen nur für einen übersichtlichen Programmkode.

LESEN DER IR-DATEN

Um den Strom der IR-Daten der Fernbedienung zu lesen und korrekt auszuwerten, bedarf es einer Reihe von 13 PULSIN-Befehlen (oder acht, wenn man die Extra-Bits ignoriert). Da nur acht der PULSIN-Werte gespeichert werden müssen, könnte ein geeigneter Programmkode wie folgt aussehen:

```
irsense con 0
irstartlow con 1100
   ' minimale Startbit-Länge
```

Listing 8. Die Software für eine IR-Steuerung

```
' Remote Rover von Al Williams
irsense con 0
irinput var in0
irthreshold con 450
irstartlow con 1100
irstarthi con 1300

value var byte  ' Ergebnis

raw var word(7)
start var word
dummy var word


right_servo con 3 ' rechter Servo
left_servo con 15 ' linker Servo
delay var byte    ' Motor Umdrehungszeit
center con 750
speed var word
i var byte


delay=10
speed=100

top:
  gosub read_ir
  if value=1 then forward
  if value=3 then left
  if value=5 then right
  if value=7 then back
  goto top


forward:

  for i=1 to delay*2
    pulsout left_servo,center-speed
    pulsout right_servo,center+speed
    pause 20
  next
  goto top

back:
 for i=1 to delay
    pulsout left_servo,center+speed
    pulsout right_servo,center-speed
    pause 20
  next
  goto top

left:
  for i=1 to delay
    pulsout left_servo,center-speed
    pulsout right_servo,center-speed
    pause 20
  next
  goto top

right:
 for i=1 to delay
    pulsout left_servo,center+speed
    pulsout right_servo,center+speed
    pause 20
  next
  goto top


read_ir:
' Das Problem ist die 500 us lange Pause
' zwischen den Bits.
' Die Stamp liest Bits fehlerhaft,
' wenn nicht alles in einem
' Rutsch gelesen wird. Deshalb lässt sich eine
' Auswertung oder ein Test des Startbits erst
' durchführen, wenn alles gelesen ist.

  if irinput=0 then noir  ' schon in der
Impulsmitte, deshalb weglassen
  pulsin irsense,0,start
  pulsin irsense,0,raw(0)
  pulsin irsense,0,raw(1)
  pulsin irsense,0,raw(2)
  pulsin irsense,0,raw(3)
  pulsin irsense,0,raw(4)
  pulsin irsense,0,raw(5)
  pulsin irsense,0,raw(6)
' Diese Zeilen könnten auskommentiert werden:
  pulsin irsense,0,dummy
  pulsin irsense,0,dummy
  pulsin irsense,0,dummy
  pulsin irsense,0,dummy
  pulsin irsense,0,dummy
' Test für ein gutes Startbit
  if (start<irstartlow) or (start>irstarthi)
then noir
  value=0
  for dummy=6 to 0
    value=value*2
    if raw(dummy)<irthreshold then ir0
    value=value+1
ir0:
  next

  return
noir:
  value=-1
  return
```

```
  irstarthi con 1300
     ' maximale Startbit-Länge
  raw var word(7)
  dummy var word
  start var word

read_ir:
  pulsin irsense,0,start
 ' Lesen des potentiellen Startbit
  pulsin irsense,0,raw(0)
  pulsin irsense,0,raw(1)
  pulsin irsense,0,raw(2)
  pulsin irsense,0,raw(3)
  pulsin irsense,0,raw(4)
  pulsin irsense,0,raw(5)
  pulsin irsense,0,raw(6)
  pulsin irsense,0,dummy
          'Geräte-ID ignoriert
  pulsin irsense,0,dummy
          'Geräte-ID ignoriert
  pulsin irsense,0,dummy
          'Geräte-ID ignoriert
  pulsin irsense,0,dummy
          'Geräte-ID ignoriert
  pulsin irsense,0,dummy
          'Geräte-ID ignoriert
```

Nun ist es nur noch nötig, die rohen Daten zu bearbeiten. Es ist denkbar, dass das erste vom Programmkode gelesene Bit nicht das Startbit war. Ein einfacher Test schützt aber davor, ein Datenpaket anders als am Anfang zu beginnen.

```
if start<irstartlow or
  start>irstarthi then noir
```

Unglücklicherweise lässt sich dieser Test nicht direkt nach dem ersten empfangenen Bit durchführen; stattdessen muss man das gesamte Paket lesen und erst dann entscheiden, ob es korrekt war oder nicht.
Die nächste Aufgabe ist es, die Rohdaten in eine binäre Zahl umzuwandeln. Hier gibt es aber keine Zeitprobleme.

```
  value var byte
  value=0
  for dummy=6 to 0
    value=value * 2
```

```
    if raw(dummy)<irthreshold
then ir0
    value = value +1
ir0:
  next
  return
```

Dieser Programmausschnitt untersucht jedes Rohdatum (in umgekehrter Reihenfolge), ob es größer ist als der Schwellwert und addiert in diesem Fall eine 1 zur Variablen **value**. In jedem Schleifendurchlauf wird der Variablenwert mit 2 multipliziert, die binäre Stelle der Variable also nach links verschoben. Die Multiplikation lässt sich bei der BASIC-stamp durch einen Shift-left-Operator ersetzten:

```
value = value << 1
```

Mit diesen Mitteln lässt sich die Eingabe einer Fernbedienung erfassen. Das Programm sollte berücksichtigen, dass die Fernbedienung den Befehl wiederholt, solange die entsprechende Taste gedrückt ist.

BoE-Bot Remote Controlled

Ausgestattet mit diesen IR-Sensor-Routinen ist es kein so großes Problem mehr, die Fernsteuerung des BoE-Bots zu realisieren. Alles, was man jetzt noch wissen muss, ist der Wert, den die Fernbedienung auf den Druck einer bestimmten Taste hin sendet. Dies lässt sich ganz einfach herausfinden, indem man in die Routine *ir_read* einen Debug-Befehl einbaut und die Ausgabe überprüft.

Bei einer Sony-Fernbedienung hat die Taste 1 eine 0 zur Folge, Taste 2 gibt eine 1 aus und so weiter. Bei einem Druck auf Taste 0 sendet die Fernbedienung den Wert 10. Die Steuerung des BoE-Bots geschieht (logisch!) über die Tasten 2 (vorwärts), 8 (rückwärts), 4 (links) und 6 (rechts) der Fernbedienung. Das endgültige Programm für den infrarot ferngesteuerten BoE-Bot ist in **Listing 1** abgedruckt.
Es gibt allerdings noch ein kleines Problem: Wenn man unmittelbar die Richtung des fahrenden Roboters ändert, also direkt von einer auf eine andere Taste wechselt, reagiert der BoE-Bot nicht korrekt. Der Grund: Alle PULSIN-Befehle müssen abgelaufen sein, bevor das Programm wieder in die Hauptschleife springt. Wenn jeder der 13 Befehle eine "Auszeit" von 131 ms benötigt, ergibt dies eine Totzeit von nahezu 2 s zwischen zwei Bewegungsbefehlen. Glücklicherweise gilt dies nur für unterschiedliche, nicht aber für die Befehlswiederholung der Fernbedienung, ansonsten würde der BoE-Bot sich nur ruckweise fortbewegen können. Bei der Wiederholung braucht nicht auf das Ende des PULSIN-Befehls gewartet werden.
Natürlich könnte man diesen Mangel einfach dadurch mildern, indem man den ID-Kode nicht auswertet, also die Anzahl der Auszeiten reduziert. Zum Ausgleich erhöht sich aber leider die Häufigkeit der "verlorenen" Startbits, so dass das Programm neu mit der Fernbedienung synchronisiert werden muss. Wie man´s macht, ist es falsch ...
Schließlich bleibt noch die Variante, den Sensor auf Bereitschaft zu überprüfen, bevor nach dem Startbit gesucht wird. Wenn der Sensor eine null liest, wird gerade ein Datenpaket übertragen, so dass es sich ohnehin nicht mehr lesen lässt. Also braucht man sich nicht darum zu kümmern.
Das Listing 1 enthält einen solchen Test.

Mehr Möglichkeiten

Es gibt viele einfache Möglichkeiten, das Listing zu modifizieren. So lassen sich die Fernbedienungstasten für Lautstärke und Kanalwahl einsetzen, um Geschwindigkeits- und Verzögerungsvariablen zu ändern. Sehr interessant ist die Variante, bestimmte Tasten mit einer Art Makro-Funktion zu versehen, also ganze Bewegungsabläufe per Tastendruck abzurufen. Diese Bewegungssequenzen können gut im EEPROM abgelegt werden. Interessant ist es auch, eine Serie von Fernbedienungsbefehlen (beispielsweise für ein Fernsehgerät) im BoE-Bot zwischenzuspeichern und an ein anderes Gerät (in einem anderen Raum) weiterzugeben. Oder eine Kommunikation zwischen zwei BoE-Bots über eine weitere Distanz zu realisieren ...
Obwohl eine Infrarot-Fernbedienung sehr schnelle Impulse verwendet, kann die BASIC-stamp diese bei einer geeigneten Software lesen und verarbeiten. Der PULSIN-Befehl ermöglicht es, Impulslängen einfach und genau zu messen. Obwohl nicht unbedingt erforderlich, gestalten Arrays die Aufgabe übersichtlicher.

(990050-5)rg