Die Ruwido Merlin Tastatur

Diese Tastatur, die es immer noch bei Pollin gibt (Stand: November 2011), ist eigentlich eine nette Infrarot-Tastatur. Aber sie hat ein eher ekliges Protokoll und sie stellt erhebliche Ansprüche an den IR-Empfänger. Hier gibt's Hinweise zur Benutzung dieser Tastataur.

1. Die Hardware

Die Merlin sendet recht schnell, ihre Puls- und Pausenlängen betragen nur ca. 210 us. Deshalb kann man nicht jeden üblichen IR-Empfänger benutzen, denn da kommt nur Murks heraus. Man benötigt ein schnelles Exemplar wie z.B. den TSOP31256.

2. Das Protokoll

Die Datenpakete bestehen aus einer variablen Anzahl von Taktzeiten mit Pulsen (IR-Ausgang low) und Pausen (IR-Ausgang high) von gleicher Breite. Diese Taktzeiten betragen ca. 210 us. Es kommen in zwei aufeinander folgenden Taktzeiten maximal 2 gleiche Zustände (high oder low) vor. Das Ganze ist also so eine Art Manchestercode. Am Anfang eines jeden Datenpaketes kommt immer das gleiche Muster (siehe Bild), also eine Präambel nach folgendem Schema: 1 Takt L (Startbit), dann 2 Takte H, dann 7x LH dann 2 Takte L. Danach beginnt der eigentliche Datenbereich.

3. Meine Interpretation

Man kann den Kram, den die Merlin sendet, natürlich auf unterschiedlichste Weise interpretieren. Die Linuxer messen mit ihrem LIRC kurzum alle Zeiten und vergleichen diese dann mit einer entsprechend riesigen Tabelle. Wenn man megabyteweise Platz und CPU-Zeit im Überfluß hat, kann man das so machen.

Ich hingegen habe für 2 gleiche aufeinanderfolgende Zustände eine 1 zugeordnet und für zwei unterschiedliche die 0. Demnach wäre die Prämbel so zu lesen:

"100000001"

Dumm bei der ganzen Geschichte ist, daß die Anzahl der Pulse und Pausen eben **nicht** immer gleich ist. Die Tastatur hat mehrere Modifizier-Tasten (Shift, Ctrl, Alt, AltGr) und diese sind nicht etwa irgendwelchen Bitpositionen zugeordnet, sondern es wird so verfahren, daß bei mehreren gleichzeitig gedrückten Tasten die Protokoll-Länge erhöht wird und die Codes der Tasten nacheinander gesendet werden.

Kurzum, man hat bei diesem Protokoll kein berechenbares Ende, sondern nur ein Anfangsmuster. Allerdings beträgt die maximal übertragene Bitanzahl weniger als 32 Bit, so daß man das Ergebnis im uC in einem DWORD aufbauen kann. Ich bin deshalb bei meinem Dekodieralgorithmus so verfahren, daß ich mit dem Startbit in das Ergebnis eine 1 einschiebe und nach dem Ende des Datenpaketes das Ergebnis soweit nach links verschiebe, bis diese 1 an einer festen Stelle erscheint. Damit sind dann die Präambel und die 1..2 Nutz-Tastencodes immer gleich ausgerichtet.

4. Der Dekoder

Auch wenn der IR-Sender quarzgesteuert ist, kann man sich nicht auf präzise Zeiten beim Empfängerausgang verlassen. Die scheinbar einfachste Variante, ab Startbit einfach alle 210 us den IR-Empfänger abzutasten, ist nicht ratsam. Mein Dekoder arbeitet deshalb flankengesteuert - bis auf die Ende-Erkennung, die wird aus einer zu großen Pausenzeit gewonnen. Der Dekoder zählt die Taktzeiten: bei kurzen Zeiten +1 und bei langen Zeiten +2 und schiebt die entsprechenden Bits in das Ergebnisregister.

5. ein Pseudo-Code

Ich benutze hier mal einen pascalähnlichen Pseudo-Code. Der Dekoder besteht aus 2 Teilen: einem Interrupt-Programm, das auf jede Flanke des IR-Empfängers gestartet wird und einem gewöhnlichen Programmstück, das von Zeit zu Zeit aufgerufen wird. Obendrein benötigt das Interruptprogramm einen Counter als Zeitzähler, der die seit seinem letzten Start verflossenen us (Mikrosekunden) liefert.

```
{ Interruptprogramm }
var
  Zustand  : (warte, lade, fertig);
  TaktCont : integer;
  Ergebnis : DWORD;

interrupt procedure Ruwido:
var
  Zeit : integer;
begin
  { household hier, z.B. Int löschen usw.}

  if Zustand=warte
  begin
    Starte_Timer;
    TaktCount := 0;
    Ergebnis  := 1;  { Pilot-Bit }
    Zustand := lade;
    exit;
  end;

  if Zustand=lade
  begin
    Zeit := Hole_Zeit_vom_Timer;
    if Zeit<150us exit; { Spikes ignorieren }
    if Zeit>800us       { Timeout }
    begin
      Zustand:= fertig;
      exit;
    end;

    Starte_Timer;  { für die nächste Runde..}
    if Zeit>330us
    begin
      Ergebnis := Ergebnis * 2 + 1;
      inc(TaktCount,2); {lang = 2 Taktzeiten}
    end
    else
    begin
      inc(TaktCount,1); {kurz = 1 Taktzeit}
      { nicht zwischendurch schieben! }
      if (TaktCount and 1) = 1
         Ergebnis := Ergebnis * 2;
    end;
  end;  { Ende der Interruptroutine }

procedure Ruwido_Abfragen:
var
  Zeit : integer;
begin
  Zeit := Hole_Zeit_vom_Timer;
  if (Zeit>2000us) and (TaktCount>16
  begin
    Stoppe_Timer;
    TaktCount := 0;
    if Ergebnis <> 0
    begin
    while Ergebnis < 0xF8000000
        Ergebnis := Ergebnis shl 1;
    if (Ergebnis and 0x0A020000) =
                    0x0A020000
    begin
      Ergebnis := Ergebnis - 0x0A020000;
      Zustand := fertig;
    end;
end;
```

So, wenn Zustand = fertig, dann kann man das gelieferte Ergebnis abholen und bei Bedarf den Dekoder wieder starten, indem man in 'warte' in Zustand schreibt.

6. Ergebnisse

Bei dem Ergebnis des o.g. Dekoders sieht man folgendes:

a) bei gültigen Tastendrücken ist das Ergebnis immer >= 0x10001, also z.B. bei 'a' ergibt sich 0x10D00, bei 'A' (Shift und 'a') ergibt sich 0x10D23, und wenn man z.B. nur Shift drückt, dann ergibt das 0x02300.

b) bei ungültigen Tastendrücken (z.B. zuerst 'a' und dann Shift gedrückt) ergibt sich 0x0230D anstelle 0x10D23.

c) beim Loslassen der Tasten wird 0x00000 gesendet.

Modifikatoren (haben keine führende 1:

Shift: 0x23

Ctrl: 0x21

Alt: 0x27

AltGr: 0x2B

Die Ziffern 1,2,3..0 (hex):

23, 21, 61, 63, 67, 65, 6D, 6F, 6B, 69

Die Buchstaben a..z (hex):

0D,0F,0B,09,19,1B,1F,1D,15,17,13,11,31, 33,37,35,3D,3F,3B,39,29,2B,2F,2D,25,27

Enter 79, BkSpace 7F, Tab 7D, OnOff 7B,...

IR-Datenpaket der Ruwido-Merlin-Tastatur. gelb: 10 us Raster, rot: Bit-Raster