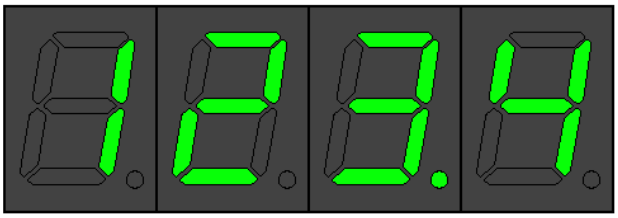
R. Seelig

LED 7-Segment 4-Digit Anzeigetreiber mit Padauk *PFS154-S16*



LED 7-Segment 4-Digit Anzeigetreiber mit Padauk **PFS154-S16**

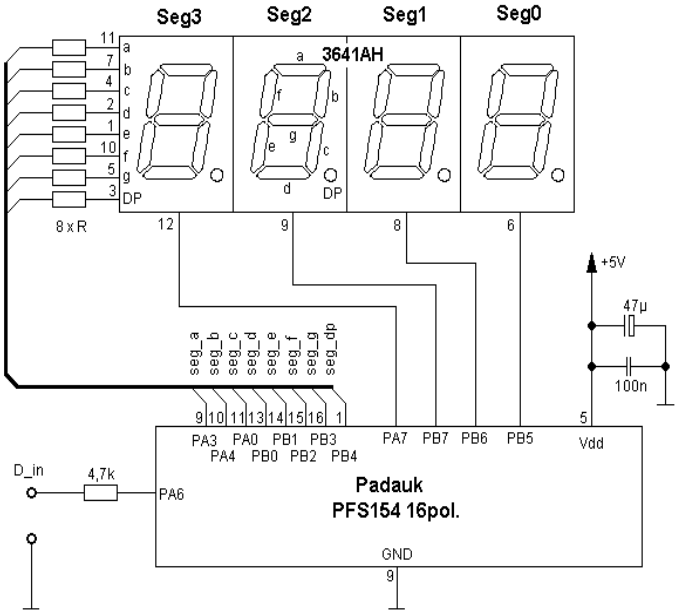
Features des Anzeigetreibers:

- multiplexen einer 4-stelligen Anzeige mit Dezimalpunkt
- Anzeigetreiber in Software für gemeinsame Kathode / gemeinsame Anode umschaltbar
- Datentransfer mit nur einer einzigen Datenleitung vom Host zum Anzeigetreibers

In Anlehnung an fertig existierende Bausteine zum Betrieb von LED 7-Segmentanzeigen, wurde mit den extrem preiswerten (billigen) Padauk PFS154-S16 Mikrocontrollern (MCU) eine Software entwickelt, die eben einen solchigen Anzeigentreiber darstellt.

Die Kommunikation von einem Host (in diesem Text hier am Beispiel eines AVR Atmega Controller) geschieht mit nur einer einzigen Datenleitung.

Die Verwendung nur einer Datenleitung und der Preis des Controllers sind dann auch die Vorteile eines mit PFS154 realisierten Treiber gegenüber existierenden Treibern TM16xx.



Nebenstehender Schaltplan zeigt eine 4-stellige LED 7-Segmenteinheit mit gemeinsamer Kathode. Die Beschaltung mit einer Anzeige mit gemeinsamer Anode ist dieselbe, im Programm des Anzeigetreibers muß hier dann die Auswahl des gemeinsamen Anschlusses gewählt werden.

Dem Schaltplan anzumerken ist, dass die Verwendung von gemeinsamen Kathoden deutlich bessere (hellere) Ergebnisse liefert als die Verwendung mit gemeinsamer Anode.

Desweiteren, um eventuellen Unmut und Äußerungen in einem bestimmten Forum vorzubeugen wurden Serienwiederstände in die Segmentleitungen eingefügt. Der Widerstandswert hier ist der verwendeten Anzeige anzupassen.

Grundsätzlich ist der Autor jedoch der Meinung, dass diese Widerstände überflüssig sind, u.a. auch deshalb, weil der PFS Mikrocontroller den Ausgangsstrom (lt. Datenblatt) je nach verwendetem Anschlusspin auf 5mA oder 10mA begrenzt.

Nach einem Versuchslauf von ca. 1 Woche Dauerbetrieb funktionierten Controller und Anzeige immer noch einwandfrei.

Softwarebeschreibung

Das Programm des Anzeigetreibers besteht im Prinzip aus 2 Teilen:

- Funktion der gemultiplexten Anzeigentreiber
- Kommunikation mit einem Hostprogramm

Multiplexing

Das Multiplexen der Anzeige geschieht sehr konventionell und ist sehr einfach aufgebaut. Die Bitmaps der einzelnen Digits werden in einem 4 Byte großen Pufferarray ***„uint8_t seg7puffer[4]“*** gespeichert.

Zu Beginn des Programms wird der 8-Bit Timer2 mit der Funktion „***void seg7_mpx_init(void)“*** initialisiert. Nach dem Initialisieren wird ca. jede Millisekunde einen Interrupt ausgelöst.

Der PFS154 besitzt nur einen Interruptvektor, jedoch mehrere Interruptquellen, sodass innerhalb des Vektors über die Interruptquellen gepollt werden muß.

Innerhalb dieses Interruptvektors wird nun jedesmal die Multiplexfunktion „***void seg7_mpx(void)“*** aufgerufen, innerhalb der, je nach Inhalt des Pufferarray, ein Bitmuster an die GPIO-Pins angelegt wird, sowie die Digitleitungen der Anzeige sequentiell aktiviert / deaktiviert werden.

Kommunikation

Die Kommunikation mit einem Host, der den Anzeigentreiber ansteuern soll gestaltete sich etwas schwieriger als erwartet. Ursprünglich wollte der Autor lediglich mittels Bitbanging etwas ähnliches wie eine serielle Schnittstelle nur in eine Richtung realisieren. Anstelle eines für eine RS-232 Schnittstelle typische 8N1 Konfiguration hätte es eben eine 32N1 Konfiguration sein sollen.

Hier ergaben sich gleich mehrere Schwierigkeiten. Zum einen hat der PFS154 nur einen einzigen Interruptvektor, eine eingehende Information auf hätte hier den Multiplexbetrieb der Anzeige gestört (Flackern), schlimmer jedoch ist, dass eingehene Informationen verloren gehen können, wenn das Programm gerade in der Multiplexfunktion arbeitet.

Eine weitere Schwierigkeit ist, dass der PFS mangels genügender Anschlüsse ohne Quarz betrieben werden muss und das Timing einer seriellen Schnittstelle, die 32 Bits ohne Synchronisation übertragen soll schwierig ist. Vor allen Dingen dann, wenn der Host evtl. ebenfalls ohne Quarz für ein Timing auskommen muß.

Für die Kommunikation wurde somit etwas proprietäres entwickelt, das am besten als Pulslängenmodulation (PLM oder PWM) bezeichnet werden kann.

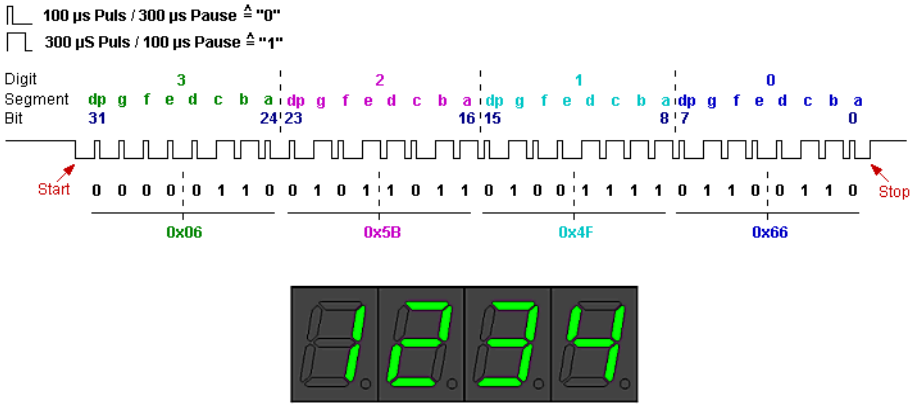
Die Information, ob ein zu übertragendes Bit eine logische 0 oder 1 enthält, ist abhängig von der Pulsdauer. Eine Dauer zwischen 150 und 2000 µs wird als eine logische 1 gewertet, eine Dauer von 50 bis 150 µs wird als logische 0 gewertet.

Überschreitet die Puls- oder Pausedauer einen Zeitrahmen von 2000 µs wird die bestehende Übertragung abgebrochen und auf eine neue Übertragung gewartet.

Im Standby sendet der Host eine „1“ und der Anzeigentreiber wartet. Geht der Host auf eine „0“ ist das für den Treiber ein Startsignal und er wartet nun seinerseits so lange, bis der Host das erste Bit sendet. Der Anzeigentreiber misst nun die Pulsdauer und interpretiert diesen Puls entsprechend seiner Länge als „1“ oder „0“.

Der Anzeigentreiber erwartet bei der Übertragung das höchstwertigste Bit des höchstwertigsten Digit (hier Digit 3) zuerst.

Das nachfolgende Impulsdiagramm zeigt die Übertragung von 32 Bits mit dem Wert 0x065B4F66, was einer Anzeige von „1234“ entspricht.



Hostprogramm für einen Atmega-Mikrocontroller

Um mittels eines AVR Controller den Anzeigetreiber steuern zu können, müssen die Dateien ***pfs_7seg.h*** inkludiert und ***pfs_7seg.c*** hinzugelinkt werden.

pfs_7seg definiert 2 Datenarrays:

```
uint8_t seg7buffer[] = { 0x3f, 0x3f, 0x3f, 0x3f };

const uint8_t seg7bmp[] =                \
  {  0x3f,  0x06,  0x5b,  0x4f, 0x66,  0x6d,  0x7d,  0x07,     \
     0x7f,  0x6f,  0x77,  0x7c, 0x39,  0x5e,  0x79,  0x71,
     0x00,  0x63 };
```

seg7buffer nimmt hierbei die Bitmuster auf, die der Abbildung auf der Anzeige entsprechen. Ein Sendevorgang ***pfs7seg_sendbmp*** wird diesen Buffer an den Anzeigetreiber senden und ist somit das zentrale Element von pfs_7seg.

Seg7bmp definiert sämtliche Bitmuster von „0“ bis „9“, „A“ bis „F“ sowie alle Segmente aus (Arrayelement 16) sowie ein hochgestelltes „o“ (Arrayelement 17) um evtl. eine Temperaturanzeige zu ermöglichen.

Folgende Funktionen sind vorhanden:

```
void pfs7seg_init(void);
void pfs7seg_sendbmp(void);
void pfs7seg_set4dez(int16_t val, uint8_t komma_pos);
void pfs7seg_set2dez(int8_t val, uint8_t pos);
void pfs7seg_sethex(uint16_t val);
```

void pfs7seg_init(void);

Initialisiert den zur Datenübertragung vorgesehenen GPIO Pin als Ausgang (standartmäßig PD1) und setzt diesen auf logisch „1“.

void pfs7seg_sendbmp(void);

Sendet die im Puffer **seg7buffer** enthaltene Bitmuster der Digits an den PFS154 LED-Anzeigecontroller.

void pfs7seg_set4dez(int16_t val, uint8_t komma_pos);

Beschreibt den Puffer **seg7buffer** mit den Bitmaps der in ***int16_t val*** übergebenen dezimalen Zahl sowie ggf. einem Dezimalpunkt und sendet den Puffer.

Übergabe:

val	: zu sendende dez. Zahl (-999 .. 9999)
komma_pos	: Position an der ein Dezimalpunkt angezeigt werden soll (1 = rechts, 3 = links, 0 = kein Dezimalpunkt

Beispiel:

```
pfs7seg_set4dez(1234,1);
```

lässt auf der Anzeige „123.4“ erscheinen.

void pfs7seg_set2dez(int8_t val, uint8_t pos);

Beschreibt den Puffer **seg7buffer** mit den Bitmaps der in ***uint8_t val*** übergebenen 2 stelligen dezimalen Zahl an der angegebenen Anzeigeposition ***uint8_t pos*** und sendet den Puffer.

Übergabe:
- val : zu sendende dez. Zahl
- pos : Position an der die zweistellige 8-Bit Zahl ausgegeben wird, 3 = links, 2 = Mitte, 1 = rechts

Beispiel:

```
pfs7seg_set2dez(83,2);
```

lässt auf der Anzeige „_83_“ erscheinen. Ein vorheriger Inhalt von Digit 3 und Digit 0 bleibt jedoch erhalten. Sollen Digit 3 und Digit 0 bei der Ausgabe gelöscht werden so ist der Puffer **seg7buffer** zuvor explit zu beschreiben:

```
seg7buffer[3]= 0;
seg7buffer[0]= 0;
pfs7seg_set2dez(83,2);
```

void pfs7seg_sethex(uint16_t val);

Beschreibt den Puffer **seg7buffer** mit den Bitmaps der in ***uint16_t val*** übergebenen hexadezimalen Zahl und sendet den Puffer.

Übergabe:
- val : zu sendende hexadezimale Zahl

Beispiel:

```
pfs7seg_sethex(0x3a4f);          // auf der Anzeige erscheint: „3A4F“

pfs7seg_sethex(7866);            // auf der Anzeige erscheint: „1EbA“
```

Anhang

Pinbelegung des PFS154-S16