Proto-Counter Software

Die Software des ProtoCounters wurde in C++ entwickelt und kann als Library in die Arduino-IDE eingebunden werden. Dazu den Ordner „ProtoCounter“ aus dem Proto-Counter Github-Repository herunterladen und in Arduino unter „Sketch → Include Library → Add .ZIP Library…“ installieren.

Die Klasse „ProtoCounter“ kümmert sich um alle für den Betrieb notwendigen Aufgaben:

- Multiplexen der Anzeige
- Darstellung von Zahlen und Text
- Entprellen und Auswerten der Taster
- Ansteuerung externer Schieberegister für zusätzliche I/Os (optional)
- Auslesen eines analogen Drehknopfs (optional)

Die Software belegt weniger als die Hälfte des Speichers im Mikrocontroller, so dass noch ausreichend Platz für eigene Anwendungen bleibt.

Programmbeispiele

Im Ordner „examples“ befinden sich einige Beispiele.

EventCounter

Dieses Beispiel implementiert einen einfachen Ereigniszähler mit zwei Eingängen. Gezählt werden High-Low-Übergänge an den Eingängen. Eingang A erhöht den Zähler um eins. Eingang B kann den Zählerstand wahlweise um eins erhöhen oder erniedrigen. Erreicht der Zähler einen einstellbaren Grenzwert, wird der Ausgang für eine gewisse Zeit aktiv. Für die beiden Eingänge ist im Programm eine Totzeit definiert, mit der z. B. verhindert werden kann, dass das Prellen von mechanischen Kontakten zu Fehlzählungen führt.

TeaTimer

Der TeaTimer illustriert, wie ein Timer mit einstellbarer Zeit realisiert werden kann. Die Bedienung ist im Programm beschrieben. Der Code ist bereits vorbereitet, um z. B. ein Gerät zu aktivieren, solange der Timer läuft, bzw. einen Alarm auszulösen, wenn der Timer abgelaufen ist.

ProtoCounter Test

Das Programm „ProtoCounter_Test.ino“ zeigt, wie die ProtoCounter-Funktionen benutzt werden. Es kann als Funktionstest nach dem Zusammenbau des Bausatzes dienen. Eine Beschreibung findet sich im Programmcode.

Konfiguration

Im Headerfile `ProtoCounter.h` können einige Einstellungen konfiguriert werden, was jedoch nur in besonderen Fällen sinnvoll sein dürfte. Die Bedeutung der Parameter ist in der Datei beschrieben. Folgendes lässt sich einstellen:

- Pinzuordnung
- Anzahl der über externe Schieberegister einzulesenden Bits
- Anzahl der über externe Schieberegister auszugebenden Bits
- Abschalten des analogen Drehknopfs
- Abtastrate der Taster (Entprellzeit)
- Dauer eines langen Tastendrucks

Zum Ausschalten des Drehknopfs wird die entsprechende Zeile auskommentiert.

```
//#define ANALOG_ENABLE
```

Die externen Schieberegister lassen sich abschalten, indem man die Bitbreite auf Null setzt.

```
#define SH_REG_IN_BITCOUNT    0
#define SH_REG_OUT_BITCOUNT   0
```

Interrupt-Steuerung

Unter Arduino nutzt die ProtoCounter-Library den CompareB-Interrupt von Timer0. Durch diesen wird die Methode `update()` regelmäßig aufgerufen. Die gesamte Steuerung des ProtoCounters geschieht somit automatisch und ganz bequem im Hintergrund. Der 16-Bit-Timer (Timer1) steht ohne Einschränkungen für eigene Anwendungen zur Verfügung.

Die Häufigkeit, mit der `update()` aufgerufen wird, hängt vom Prozessortakt ab und beträgt F_CPU / 16384.

Programmierung ohne Arduino-Umgebung

Die Methode `update()` ist die grundlegende Methode, welche sich um den Betrieb des ProtoCounters kümmert. Wer ohne die Arduino-Umgebung programmiert, muss die Methode `update()` regelmäßig aufrufen. Hierzu eignet sich ein Interrupt, der von einem Timer z. B. jede Millisekunde ausgelöst wird.

```
/* Main.cpp */
#include <avr/interrupt.h>
#include "ProtoCounter.h"

// system timing
#define SYS_TIMER_FREQ    1000    // system timer frequency (Hz)
#define SYS_CYCLE (uint8_t)(0.5 + F_CPU / (64.0 * (double)SYS_TIMER_FREQ))
...
```

```
int main(void)
{
    // setup timer0 which is used as system time base
    OCR0A  = SYS_CYCLE;       // set dot matrix refresh time
    TCCR0A = 0;           // normal mode
    TCCR0B = (3 << CS00);   // set prescaler
    TIMSK  = (1 << OCIE0A); // enable timer interrupt
    sei();           // enable interrupts to start the ProtoCounter engine
    ...
}

ISR(TIMER0_COMPA_vect)
// system timer interrupt
{
    OCR0A += SYS_CYCLE;       // setup next interrupt cycle
    sei();           // re-enable interrupts
    ProtoCounter::update();    // run the ProtoCounter engine
}
```

ProtoCounter-Methoden

Im folgenden wird davon ausgegangen, dass ein statisches Objekt der Klasse ProtoCounter angelegt wurde.

```
#include "ProtoCounter.h"

ProtoCounter pc;
```

init()

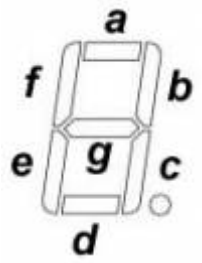
Setzt die verwendeten I/O-Pins und löscht die Anzeige. Die Member-Variablen `decimal_places` und `dimming` werden auf ihre Standardwerte gesetzt. Man kann sie anschließend jedoch auch auf eigene Werte einstellen.

```
pc.decimal_places = 2;
pc.dimming = 0;
```

clearDisplay()

Löscht die 7-Segment-Anzeige (alle Segmente aus).

getDisplay(pos)



Bezeichnung der Display-Segmente

Gibt den Inhalt der angegebenen Stelle der Anzeige zurück. pos = 0 ist die niederwertigste Stelle (ganz rechts). Die Bits 0 - 6 entsprechen den Segmenten a - g.

setDisplay(led_pattern, pos)

Setzt die angegebene Stelle der Anzeige auf das gewünschte Led-Muster. Die Bits 0 - 6 entsprechen den Segmenten a - g; pos = 0 ist die niederwertigste Stelle (ganz rechts).

writeChar(ascii_code, pos)

Schreibt ein ASCII-Zeichen in die angegebene Stelle der Anzeige. Siehe auch die Erläuterungen zum verwendeten Zeichensatz.

writeString_P(str_pointer)

Schreibt einen String in die Anzeige unter Verwendung des Zeichensatzes. Der String muss im Flash-Speicher abgelegt sein und wird durch eine Null beendet. Besonders einfach ist die Verwendung mit PSTR.

```
/* Main.cpp */
#include <avr/pgmspace.h>
...

int main(void)
{
    ...
    pc.writeString_P(PSTR("ABC"));
    ...
}
```

writeInt(val)

Stellt einen vorzeichenbehafteten Integer-Wert (16 bit) als Dezimalzahl auf dem Display dar. Der darstellbare Wertebereich ist -99 bis +999 (MIN_DECIMAL bis MAX_DECIMAL). Überschreitet der Wert diese Grenzen, erscheint „UFL“ bzw. „OFL“ im Display, um einen Unter- bzw. Überlauf anzuzeigen.

writeHex(val)

Stellt ein Byte als Hexadezimalzahl dar. Der Zahl wird ein 'h' angehängt.

setDimming(dim)

Hiermit kann die Displayhelligkeit reduziert werden. Dies verringert auch die Stromaufnahme des ProtoCounters. Sinnvolle Werte für das Dimming liegen zwischen 0 (maximale Helligkeit) und 8.

setDecimalPlaces(decimals)

Legt die Zahl der Nachkommastellen fest, falls über den Jumper J1 das Komma eingeschaltet wurde.

setAnalogResolution(ana_res)

Um das „Flackern“ der Werte des Drehknopfs zu verhindern, kann die Auflösung reduziert werden. Folgende Konstanten sind vordefiniert:

```
#define ANALOG_MAX_RESOLUTION 0
#define ANALOG_129_DETENT_STEPS   1
#define ANALOG_86_DETENT_STEPS    2
#define ANALOG_65_DETENT_STEPS    3
#define ANALOG_43_DETENT_STEPS    4
#define ANALOG_33_DETENT_STEPS    5
#define ANALOG_22_DETENT_STEPS    6
#define ANALOG_OFF        7
```

getButton()

Diese Methode liefert den Zustand der beiden Drucktasten zurück. Mit Hilfe der in ProtoCounter.h vordefinierten Events kann leicht auf einen Tastendruck reagiert werden.

```
/* ProtoCounter.h */
// push button events (do not change)
#define BTN1_PRESSED      (BUTTON1 | PB_PRESS)
#define BTN1_RELEASED     (BUTTON1 | PB_RELEASE)
#define BTN1_LONGPRESSED  (BUTTON1 | PB_LONGPRESS)
#define BTN2_PRESSED      (BUTTON2 | PB_PRESS)
#define BTN2_RELEASED     (BUTTON2 | PB_RELEASE)
#define BTN2_LONGPRESSED  (BUTTON2 | PB_LONGPRESS)


/* Main.cpp */
int main(void)
{
```

```
    ...
    if (pc.getButton() == BTN1_RELEASED) {
        pc.buttonAck();
        /* code to react on push button */
    }
    ...
}
```

buttonAck()

Hierdurch wird die Bearbeitung des aktuellen Button-Events bestätigt und das anstehende Event gelöscht.

readShiftRegister()

Eventuell angeschlossene externe Schieberegister werden mit jedem Aufruf von `update()`, also z. B. jede Millisekunde, automatisch aktualisiert. Mit readShiftRegister erhält man den Zustand der Eingänge beim letzten Update. Der zurückgelieferte Datentyp (sr_in_data_t) hängt von der eingestellten Bit-Anzahl (Parameter SH_REG_IN_BITCOUNT) ab und kann Byte, Word oder Long sein.

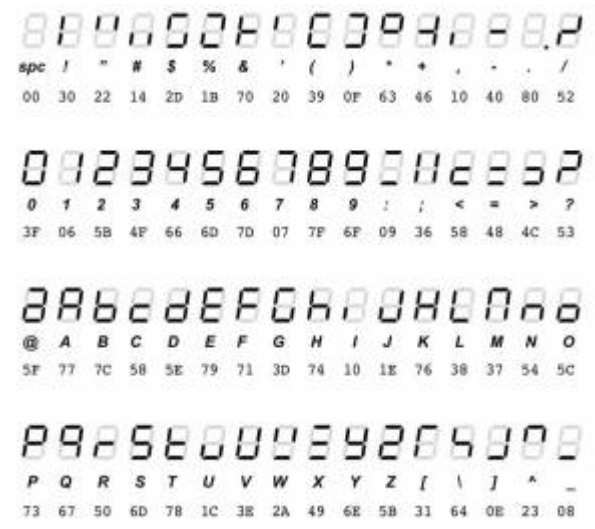
writeShiftRegister(data)

Die Daten werden beim nächsten Update-Zyklus über die externen Schieberegister ausgegeben. Der Datentyp von data (sr_out_data_t) hängt von der eingestellten Bit-Anzahl (Parameter SH_REG_OUT_BITCOUNT) ab und kann Byte, Word oder Long sein.

getAnalog()

Bei angeschlossenem Drehknopf wird ein Wert (uint8) zurückgeliefert, der der Stellung des Knopfes entspricht.

Zeichensatz



7-Segment-Zeichensatz

Zur Ausgabe von Text wird ein eigener 7-Segment-Zeichensatz verwendet. Er enthält die ASCII-Zeichen chr(32) bis chr(96). Die Codes 0 bis 15 werden auf die Hexadezimalziffern '0' bis 'F' abgebildet. Kleinbuchstaben und Großbuchstaben haben dasselbe Aussehen. Codes 16 bis 31 und Codes oberhalb von 127 werden durch ein Leerzeichen (alle Segmente aus) ersetzt.

Wer den Zeichensatz anpassen und eigene Zeichen entwerfen möchte, wird vielleicht dieses Online-Tool nützlich finden (http://www.uize.com/examples/seven-segment-display.html).

ProtoCounter, Elektronik, Bausatz, Programmierung