Mobiles Hardware-Praktikum 2023

Marc Pfeifer, Dr. Philipp Scholl, Professur für Rechnerarchitektur,
Institut für Informatik, Albert-Ludwigs-Universität Freiburg, Freiburg im Breisgau

Übungsblatt 3 (26 Punkte)

Ziel:

Back to the Arduino Pro Mini! Die Funktionen des ATmega328 Microcontrollers werden durch hardwarenahe Befehle gesteuert. Das Ziel ist monophone Klingeltöne in der *Ring Tone Text Transfer Language* (RTTTL) im Hintergrund abspielen zu können.

Vorbereitung:

In Zeiten von MicroPython[1], eLua[2] und Espruino[3] vergisst man schnell, das Microcontroller nur begrenzte Resourcen besitzen. Beispielsweise reicht der Speicher des Arduino Pro Mini gerade einmal aus um das nachfolgende Bild (31 KB 200x200 Pixel als *PNG*) abzuspeichern.

Abbildung 1: Nokia3310 (Jahr 2000) basierend auf einem ARM7 32 bit Prozessor mit 13MHz.

Die ersten Mobiltelefone, die Ende der 90er Jahre entwickelt wurden, kamen ebenfalls mit begrenzten Ressourcen aus. Beispielsweise das in der Abbildung gezeigte Nokia3310 enthielt einen single-core Prozessor der mit 13MHz getaktet wurde - also vergleichbar mit unserem Arduino Pro Mini. Damit lief sich telefonieren, SMS schreiben und einfache Spiele wie “Snake” spielen.
Klingeltöne wurden bei Nokia (und auch anderen Herstellern) im RTTTL Format gespeichert und erlaubten einfache monophone Melodien abzuspielen, die man auch selbst kreieren konnte.
Ihr habt nun die Aufgabe einen Teil der Entwicklung des Nokia Teams aus dem Jahr 1999 zu übernehmen. Dazu sollt ihr einen RTTTL-Player entwickeln, der einen Klingelton im Hintergrund abspielen kann - man will ja zumindest noch gleichzeitig durch das Menü navigieren können, während der Klingelton ertönt.
Die Klingeltöne bestehen aus einer Sequenz von Musik-Noten. Eine Note besteht aus Tonhöhe und Dauer. Außerdem können zwischen Noten noch Pausen entsprechender Länge sein. Zu jedem Zeitpunkt kann nur eine Note abgespielt werden (daher monophon). Zum Spielen einer Note, müssen Schallwellen mit der entsprechenden Trägerfrequenz erzeugt werden. In damaligen Mobiltelefonen und auch auf unserem Roboter ist dafür ein Piepser verbaut. Die Frequenz von Musiknoten können Sie zB `https://pages.mtu.edu/~suits/notefreqs.html` entnehmen.
Die Aufgaben sind so strukturiert, dass Sie nach und nach die für die Gesamtaufgabe erforderlichen Funktionen implementieren. Da es im Jahr 1999 u.A. das Arduino Projekt noch nicht gab, musste hardwarenah programmiert werden - zum Teil wahrscheinlich sogar in Assembler. Wir beschränken uns in diesem Blatt allerdings auf eine hardwarenahe Programmierung in *C*.

[1] https://micropython.org [2] http://www.eluaproject.net [3] http://www.espruino.com

Hinweise:

Beachten Sie die Folien zur hardwarenahen Programmierung des Arduino und die Code-Beispiele auf der Website. Diese enthalten viele Tipps und Beispiele. Beachten Sie außerdem, dass ihr Code ausreichend kommentiert sein sollte. Dies gilt insbesondere, wenn Register geschrieben oder gelesen werden. Sie benötigen außerdem das Datenblatt des ATmega328p. Dies ist im Ilias hinterlegt.

Aufgabe 1 (2 Punkte):

Um nun erstmal zu verstehen, wie wir einen Pin auf 0 und 1 setzen können, ohne die Arduino Funktion *digitalWrite()* zu verwenden, sollen Sie eine Funktion *setPin13(boolean high)* **nach dem Vorbild aus der Vorlesung** entwickeln. Diese soll direkt in das entsprechende Register schreiben um Pin 13 auf 1 ($high = true$) oder 0 ($high = false$) zu setzen.

Nutzen Sie die erstellte Funktion und togglen Sie den Pin mit einer Frequenz von 1 Hz. Die LEDs auf dem Arduino Board sollte entsprechend leuchten.

Hinweis: Arduino Pin 13 entspricht dem Register PORTB, Bit 5.

Aufgabe 2 (4 Punkte):

Um mit dem Piepser z. B. die Note C_6 zu spielen, muss er Schallwellen mit 1046 Hz erzeugen. Dazu muss am entsprechenden Microcontroller Pin eine Frequenz von 1046 Hz anliegen. Um eine solche Frequenz zu erzeugen, verbinden Sie zunächst Pin 12 mit dem Piepser (Buchse “Sound”). Entwerfen Sie nun eine Funktion *setTimer1Freq* die mit Hilfe von Timer 1 und eines Interrupts, an Pin 12 ein Rechtecksignal (ein Sinus ist nur mit zusätzlicher Hardware möglich) mit einer Frequenz von 1046 Hz erzeugt (1046 Hz sind hörbar). Verwenden Sie, wie in der Vorlesung gezeigt, den Timer im *CTC* Modus[4].

Hinweise:

- Achten Sie dabei darauf, in welchem Register die entsprechenden Bits stehen (siehe Datenblatt - z. B. Bit *WGM13* ist nicht im selben Register wie Bit *WGM12*).
- Sie können die Frequenz mit einer entsprechenden Smartphone App analysieren oder einfach mit einem entsprechenden Ton vergleichen (siehe zum Beispiel dazu den Online Tone Gernator unter `https://www.szynalski.com/tone-generator/` - hier kann man ebenfalls ein Rechtecksignal einstellen).
- Da Sie keine perfekte Sinuswelle erzeugen, sondern ein Rechtecksignal, enthält das Freqzenzspektrum ebenfalls Obertöne $n \cdot f_0$ der Basisfrequenz ($f_0 = 1046$ Hz). Wenn sie ein Smartphone zur Frequenzanalyse benutzen kann es sein dass für einen der Obertöne ein höherer Frequenzanteil detektiert wird (siehe Abbildung 2). Ein gewisser Anteil sollte allerding auch beim Grundton (siehe Cursor in Abbildung 2) vorhanden sein.

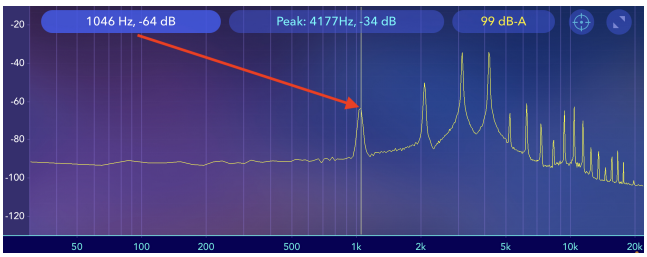

Abbildung 2: Spektrum eines 1046 Hz Tons generiert mit dem Piepser. (iOS App “Spectrum Analyzer”)

Aufgabe 3 (1 Punkte):

Wir möchten nun auf die eben implementierte ISR verzichten. Einerseits um CPU Zyklen zu sparen, und andererseits um Unterbrechungen im Haupprogramm zu unterbinden. Dazu ändern Sie die

[4] Achtung: Wählen Sie den *CTC* Modus mit *TOP* = *OCR1A*, nicht den mit *TOP* = *ICR1*!

Konfiguration von Timer 1 (in *setTimer1Freq*) so dass die eingestellte Frequenz an Pin 10 anliegt ohne diesen Pin in einer ISR explizit auf 0 oder 1 zu setzen. Kommentieren Sie dazu die ISR aus. Testen Sie Ihre Implementierung, indem Sie nun stattdessen Pin 10 mit dem Piepser verbinden.

Hinweise:

- Das *TCCR1A* Register kann entsprechend gesetzt werden, damit Pin 10 toggelt, wenn *OCR1A* erreicht wird.
- Wenn ihr Controller nun abstürzt ist der Timer evtl. noch so konfiguriert, dass er die ISR erwartet (dies sollten Sie ändern).

Aufgabe 4 (4 Punkte):

Um neben C_6 =1046 Hz auch noch andere Töne abspielen zu können, erweitern Sie *setTimer1Freq* um einen Parameter für die gewünschte Frequenz an Pin 10. Diese soll zwischen 100 Hz und 3 kHz liegen können. Die Funktion soll das *Output Compare Register* und möglicherweise den *Prescaler* ändern, aber nicht den gesamten Timer verändern. Beachten Sie, dass es sich um einen 16 Bit Timer handelt, *OCR1A* und *OCR1B* können also maximal einen Wert von 2^{16} enthalten. Bei Frequenzen außerhalb des genannten Bereichs soll der Pin auf *low* gesetzt und der Timer deaktiviert werden.

Aufgabe 5 (5 Punkte):

Spielen eine Tonfolge ab indem sie die Funktion *setTimer1Freq* mit unterschiedlichen Frequenzen nacheinander aufrufen und zB. ein entsprechendes Delay dazwischen einfügen.

Leider verhindert ein aktives Delay dass der Controller in der Zeit andere Aufgaben übernehmen kann. Um dies nebenläufig zu gestalten und nur nach Ablauf der Verzögerung den Timer umzukonfigurieren können wir Interrupts verwenden.

Schreiben Sie eine Funktion *setTimer2* die den Timer 2 so konfiguriert, dass er jede 1 ms einen Interrupt auslöst. Implementieren Sie ebenfalls die entsprechende ISR. Über einen Parameter soll *setTimer2* den Timer aktivieren bzw. deaktivieren können. Führen Sie außerdem eine neue globale Variable *volatile uint32_t tCount*[5] ein, die bei jedem Interrupt (in der ISR) inkrementiert wird. Testen Sie Ihre Implementierung, indem Sie in der *loop* regelmäßig (z.B. 1x pro Sekunde) den aktuellen Wert von *tCount* über die serielle Schnittstelle ausgeben.
Initialisieren Sie die serielle Schnittstelle dazu einmalig in der *setup* mit 38400 baud.

Aufgabe 6 (4 Punkte):

Führen Sie zwei neue globale Arrays *uint16_t durations[10]* und *uint16_t notes[10]* ein. Dabei entspricht der Wert *durations[0]* die Dauer (in Millisekunden) und *notes[0]* die Frequenz (in Hz) des ersten zu spielenden Tons der Melodie. Füllen Sie beide Arrays mit 10 Werten zwischen 200 und 1000. Wenn Sie sich mit *structs* auskennen, können Sie stattdessen auch einen entsprechenden Typ (zB *struct Note*) definieren.
Führen Sie außerdem die neue globale Variable *volatile uint8_t melodyIdx* ein (initialisiert mit 0). *durations[melodyIdx]* und *notess[melodyIdx]* sollte der gerade zu spielende Note und Dauer der Melodie entsprechen.
Erweitern Sie die ISR aus Aufgabe 5 um folgende Funktionalität:
Wenn $tCount \geqslant durations[melodyIdx]$ wird $tCount$ zurück auf 0 gesetzt, $melodyIdx$ inkrementiert und die nächste Frequenz der zu spielenden Melodie angelegt (mittels $notes$ und *setTimer1Freq*). Beachten Sie dabei, dass $melodyIdx$ maximal den Wert 9 haben darf. Ist der Wert 10 erreicht, soll der Timer (über *setTimer2*) deaktiviert werden. Implementieren Sie eine weitere Funktion *playMelody*, die die entsprechenden initialen Werte für den *index* setzt, und *setTimer1Freq* bereits mit dem ersten zu spielenden Ton aufruft. Zudem sollte diese Funktion die in Aufgabe 5 implementierte *setTimer2* entsprechend aufrufen. Ihre Implementierung sollte nun einmalig eine kurze Melodie abspielen können, wenn Sie *playMelody* am Ende der *setup* ausführen.

[5] Das Keyword $volatile$ wird benötigt um dem Compiler mitzuteilen, dass sich der Wert von *tCount* auch außerhalb des "normalen" Programmablaufs ändern kann.

Aufgabe 7 (6 Punkte)

Nun haben wir alle benötigten Funktionen zusammen um Klingeltöne abzuspielen. Allerdings müssen wir diese noch aus dem entsprechenden Format parsen und in die entsprechenden Arrays *durations* und *notes* speichern.
Monophone Klingeltöne sind in der Ring Tone Text Transfer Language (RTTTL)[6] kodiert.
Als Beispiel für die Erklärung von RTTTL dient folglich der Klingelton:

Bgirl:d=4,o=5,b=125:8g#,8e,8g#,8c#6,a,p,8f#,8d#,8f#,8b,g#,8f#,8e,p,8e,8c#,f#,c#,p,8f#,8e,g#,f#

Jeder RTTTL Klingelton beginnt mit einem Namen (der auch leer sein kann, im Beispiel: "Bgirl") gefolgt von einem ":".

Danach werden default Parameter für diesen Klingelton gesetzt:

- d steht für *duration* und beschreibt den default Notenwert (ganze, halbe, viertel, ...). $d = 4$ gibt im Beispiel an, dass alle Noten, für die es nicht explizit anders angegeben ist, Viertelnoten sind. Die möglichen Werte sind: $d \in [1, 2, 4, 8, 16, 32]$.
- o steht für *octave* und beschreibt die default Oktave. $o = 5$ gibt in diesem Beispiel an, dass alle Noten, für die es nicht explizit anders angegeben ist, aus der 5. Oktave zu wählen sind. Die möglichen Werte sind: $o \in [4, 5, 6, 7]$.
- b steht für *beats per minute* und beschreibt die default Dauer einer Viertelnote. $b = 125$ gibt in diesem Beispiel an, dass alle **Viertel**-Noten eine Dauer von $60 \cdot 1000/125 = 480\,ms$ haben sollen. Die möglichen Werte sind: $b \in [25, \ldots, 900]$.

Die Reihenfolge der Parameter ist fest (erst d, dann o und schließlich b), die Parameter sind aber jeweils optional. Die Standardwerte sind $d = 4, o = 6, b = 63$. Die Parameter werden wieder mit ":" von den tatsächlichen Noten getrennt.

Einzelne Noten werden selbst mit "," separiert. Jede Note kann aus den folgenden Komponenten bestehen: <*duration*><*note*><*octave*><*dot*>. Alles außer <*note*> ist dabei optional.

- <*note*> beschreibt den zu spielenden Ton ($\in [c, c\#, d, d\#, e, f, f\#, g, g\#, a, a\#, b, p]$). b kann auch in manchen RTTTL Melodien als h kodiert sein, sie müssen diesen Fall aber nicht unbedingt abfangen. p ist dabei ein Spezialfall für eine Pause Note (Frequenz = $0\,Hz$).
- <*duration*> ist identisch mit dem d Parameter. Ist <*duration*> leer, gilt der default Parameter.
- <*octave*> ist identisch mit dem o Parameter. Ist <*octave*> leer, gilt der default Parameter.
- Ist ein <*dot*> = *"."* angehängt, wird der Ton um die Hälfte seiner Dauer verlängert. Beispiel: "$4g$." ist eine Viertelnote g verlängert um eine Achtelnote g.

Speichern Sie zum Testen ihrer Implementierung zunächst den *Bgirl* Klingelton in ein globales array *char buffer[] = "Bgirl:...";*. Parsen Sie die einzelnen Teile des Strings und übersetzen Sie die Noten in die entsprechenden Frequenzen und Längen und speichern sie diese entsprechend in die Arrays *durations* und *notes* (oder ggf. *struct*).
Achtung: Gegebenfall muss die Array-Größe entsprechend angepasst werden. Nutzen Sie dazu eine maximale Größe (z.B. 100) und eine weitere globale Variable *melodyLen*, die auf die Anzahl der zu spielenden Töne gesetzt wird. Nach dem Parsen des Klingeltons, soll dieser nun einmalig im Hintergrund ausgegeben werden. Verwenden Sie hierzu die *playMelody* Funktion aus Aufgabe 6.

Lösungsskizze:
Ihnen werden bei der Implementierung jegliche Freiheiten gelassen. Nachfolgend ist aber eine mögliche Vorgehensweise skizziert:

1. Ermitteln Sie zunächst die Gesamtlänge des *buffer* z.B. mittels der *strlen* Funktion und laufen Sie mit einer *index* Variable über den *buffer*.

[6] Weitere Information zu RTTTL finden Sie hier unter:
http://www.mobilefish.com/tutorials/rtttl/rtttl_quickguide_specification.html

2. Sie können den Namen überspringen, indem sie bis zum nächsten ':' laufen.
3. Parsen sie die Standardwerte zuerst. Ein Beispiel wie das gehen kann, ist in Zeile 24 von Listing 1 gezeigt.
4. Beim Parsen der einzelnen Noten empfiehlt sich, eine *for* oder *while* loop zu verwenden und einen Index für die Länge der Melodie mitzuführen.
   (a) Parsen Sie zunächst die <*duration*> (falls angegeben - sonst gilt die standard duration).
   (b) Parsen Sie nun die <*note*> (einzelner character). Sie können zB. durch ein *case*-Konstrukt die <note> in die entsprechende Frequenz einer fixen Oktave umwandeln.
   (c) Ist das nächste Zeichen '#' müssen sie diese noch um einen Halbton erhöhen. (Dieser Schritt kann auch in das *case*-Konstrukt mitaufgenommen werden).
   (d) Parsen sie danach noch die <*octave*> (falls angegeben - sonst gilt die standard octave).
   (e) Verändern Sie die Notenfrequenz abhängig der *octave* (siehe Hinweise).
   (f) Ist das nächste Zeichen '.', müssen sie <*duration*> um die Hälfte derselben erhöhen.
   (g) Entsprechend der *bpm* können Sie nun die Dauer der Note in Millisekunden berechnen.
   (h) Die Werte entsprechend dem Loop-Index in das *durations* und *notes* Array speichern.
   (i) Ist das nächste Zeichen kein ',', haben Sie das Ende der Melodie erreicht.
5. Schlussendlich können Sie nun die Melody abspielen, indem Sie *playMelody* aufrufen.

Hinweise:

- Die Frequenz ($f(N_{oct})$) einer Note (N) hat einen exponentiellen Zusammenhang mit der Oktave (oct): $f(N_{oct}) = f(N_0) \cdot 2^{oct}$. So müssen Sie nur die Frequenzen einer Oktave speichern (zB der kleinst möglichen ($o = 4$)) und können die anderen entsprechend berechnen.
- Für Pausen (<*note*>$= p$) können Sie einfach eine Frequenz von 0 Hz wählen. Den Spezialfall sollten Sie bereits in der *setTimer1Freq* Funktion abgefangen haben.
- Um ein character array zB "16" in ein integer 16 umzuwandeln, können Sie z.B. wie in Listing 1 gezeigt vorgehen. Im Code von *parseRTTTL* ab Zeile 24 is die Verwendung der Funktionen *charIsDigit* und *str2uint* gezeigt. Hier wird die *standard duration* nach dem Namen des Klingelton ausgelesen. Alle anderen Werte wie die *standard octave* und *bpm*, sowie <*duration*> und <*octave*> einer jeden Note können analog ausgelesen werden.

```
// Check if the given character is a valid digit
bool charIsDigit(char c) {
  // '0'-'9' are ascii encoded
  if (c < '0' || c > '9') return false;
  return true;
}
// Convert the digits at the current and following
// buffer position idx into an unsigned integer
uint16_t str2uint(char * buffer, uint16_t * idx) {
  uint16_t val = 0;
  // While character is a valid digit
  while (charIsDigit(buffer[*idx])) {
    // '0' is 48 ascii encoded, '1' is 49. '2' is 50 etc.
    val = (val * 10) + (buffer[*idx] - '0');
    // Increment index (* needed to address pointer)
    *idx = *idx + 1;
  }
  return val;
}

void parseRTTTL() {
  ...
  // Example of parsing optional standard duration
  if (buffer[idx] == 'd') {
    idx += 2; // skip 'd'+'='
    if (charIsDigit(buffer[idx])) {
        // That's how you call the above function and pass parameters as pointers
        standardDuration = str2uint(buffer, &idx);
    }
    idx++; // Skip comma
  }
  ...
}
```

Listing 1: Beispiel um *characters* in *integers* umzuwandeln und die *standard duration* zu parsen.

Abgabe:

Archivieren Sie Ihr Programm in einer ZIP-Datei und laden Sie diese im Übungsportal hoch. Ihre Abgabe sollte ein Programm mit einer Lösung für jede Aufgaben enthalten. Code von Aufgaben die verändert oder nicht mehr benötigt werden, sollte auskommentiert werden. Fügen Sie einen entsprechenden Kommentar hinzu (Bsp: *// Ex03: Configuration of Timer 1 for 1046Hz ...*). Überprüfen Sie die Archiv-Datei auf Vollständigkeit.