BERGISCHE UNIVERSITÄT WUPPERTAL
FACHBEREICH C - PHYSIK
ELEKTRONIKPRAKTIKUM

Version 1.19, TEX: 25. Januar 2019

Versuch EP 12
Aufbau einer Meßkette, Rauschen

I. Zielsetzung des Versuches

Zum Abschluss des Elektronikpraktikums werden Sie verschiedene Kenntnisse, die Sie während des Praktiums erworben haben, anwenden. Die Praxis eines Experimentalphysikers ist die Aufnahme und Verarbeitung von Messwerten. Daher wollen wir eine komplette Messkette realisieren und ein System zur Aufnahme des Pulsschlags aufbauen.

- Das Eingangssignal kommt von einem Sensor. Wir nutzen den aus Versuch EP6 bekannten Fingerpulssensor.
- Das Signal ist von verschiedenen Störungen (Rauschen, Störsignale durch Störlicht und Störimpulse) überlagert. Wir müssen es daher filtern und verstärken.
- Zur Weiterverarbeitung wird das Signal mit einem Analog-Digitalkonverter digitalisiert.
- Die weitere Auswertung des Signals erfolgt ausschließlich auf einem Mikrocontroller. Sie kennen den Arduino bereits aus den vergangenen zwei Versuchen EP10 und EP11.

II. Vorkenntnisse

Die bisherigen EP-Versuche, insbesondere EP10 und EP11 mit dem Arduino.

Sensortechnik. Funktionsweise von Photodiode, Phototransistor, Leuchtdiode. Aufbau und Funktionsweise des Fingerpulssensors.

Filtertechniken. RC-Filter, RL-Filter, RCL-Filter. Aufbau von Tiefpaß, Hochpaß, Bandpaß, Integrierer, Differenzierer.

Schaltungen mit Operationsverstärkern, speziell: Verstärkerschaltungen, Diskriminator/Komparator und Strom-nach-Spannung-Wandlung (Transimpedanzverstärker, vgl. Abschnitt *Strommessungen mit dem Op-Amp* in Skript EP4).

Aufbereitung von Signalen für digitale Messung. Funktionsweise eines ADCs.

Mittelwerte (einfach, gleitend), RMS, Berechnungsmethoden.

III. Theorie zum Versuch

1. Mittelwertbildung zur Pulsbestimmung

Die Pulsfrequenz liegt in der Größenordnung von 1 Hz. Will man die Pulszahl (in Schlägen pro Minute) genau messen, muß man eigentlich eine Torzeit von 1 Minute haben. So bekommt man im Abstand von 1 Minute eine Anzeige der jeweiligen Pulszahl.

Dieses Verfahren ist offensichtlich sehr träge und für medizinische Anwendungen fast unbrauchbar.

Kommerzielle Pulsmeßgeräte liefern im Abstand weniger Sekunden die aktuelle Pulszahl. Man kann die Messung erheblich beschleunigen, indem man nicht die Frequenz f, sondern die Periodendauer $T = 1/f$ mißt, denn die Periodendauer ist ja der Abstand zwischen zwei Pulsschlägen.

Die simpelste Methode besteht also darin, ständig den Zeitabstand T_P zwischen den Pulssignalen zu messen und in die Pulszahl N umzurechnen gemäß $N = (1\text{ Minute})/T_P$.

Da T_P aber von Schlag zu Schlag etwas schwankt, werden sich auch deutlich schwankende Pulszahlen ergeben.

Man ist in der Praxis aber an einem Mittelwert interessiert, der aus mehreren Messungen gebildet wird. Jetzt kommt man scheinbar wieder in das ursprüngliche Problem, erst über einen längeren Zeitraum messen zu müssen, bevor ein (mittlerer) Wert angezeigt werden kann.

Die Lösung besteht in der Bildung eines *gleitenden Mittelwertes*. Man bildet *nach jedem Pulsschlag* den Mittelwert *über die letzten (z.B. 10) Messungen*. Auf diese Weise läßt sich nach jedem neuen Pulsschlag ein Meßwert anzeigen, der aber über eine größere Zahl von Messungen mittelt.

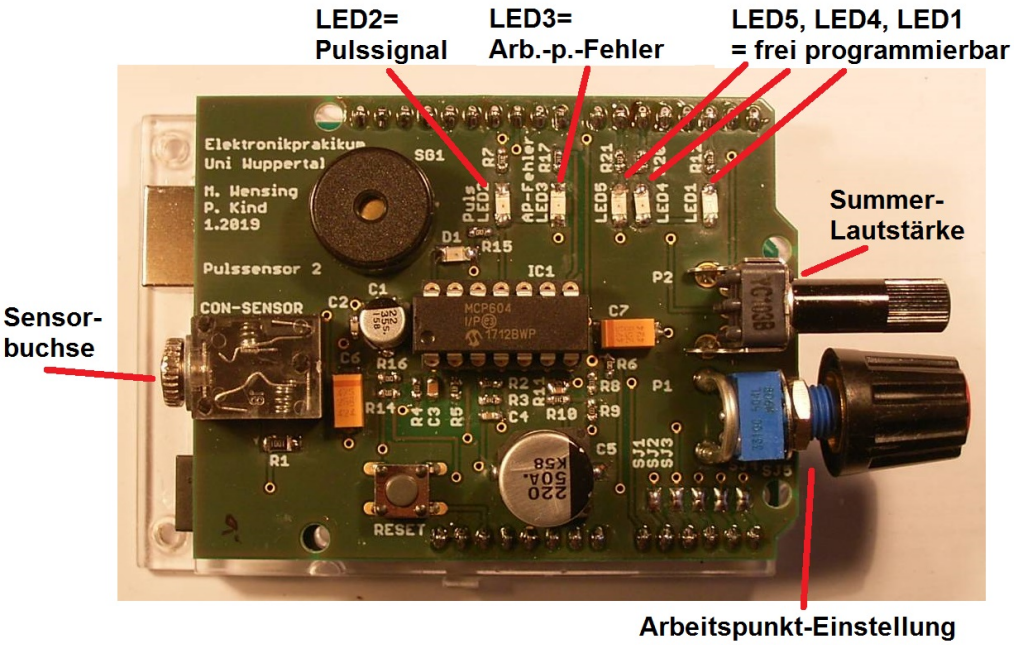
IV. Hinweise zum Versuchsaufbau

1. Fingerpulssensor

Wie aus Versuch EP6 bekannt, befindet sich in einem Fingerpulsaufnehmer eine Lichtschranke aus IR-LED und Phototransistor. Der pulsierende Blutstrom schwächt das Infrarotlicht periodisch ab. Am Oszilloskop können Sie dann Ihren Puls sehen. Für eine störungsfreie Weiterverarbeitung muß dieses Signal noch gefiltert und verstärkt werden.

2. Arduino-Shield

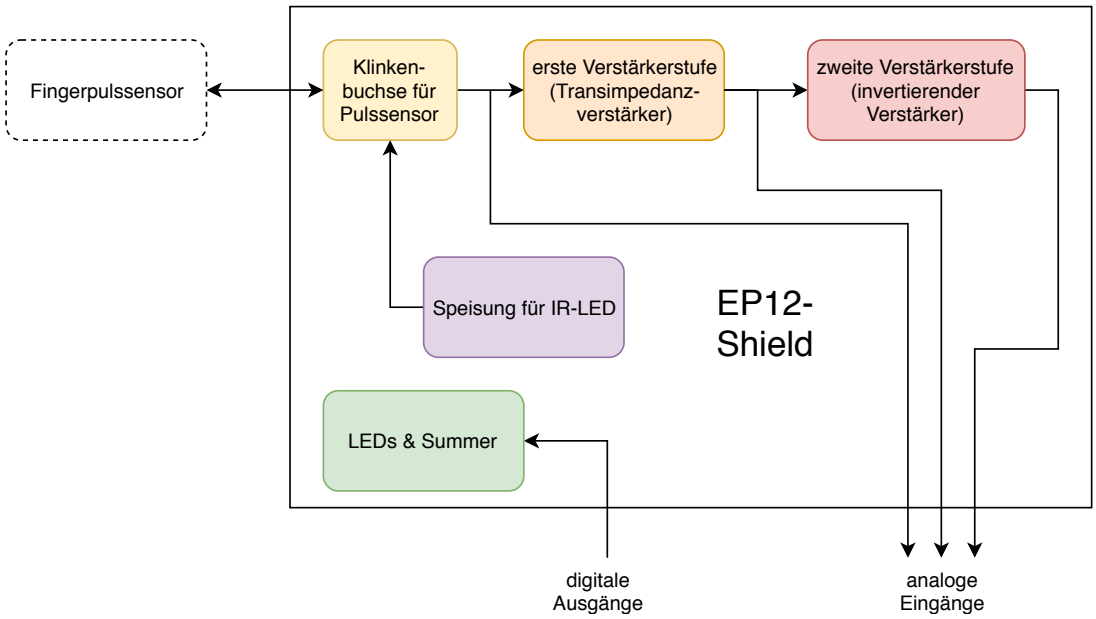
Wie auch in den vorangegangenen Versuchen nutzen wir für diesen Versuch einen Arduino-Shield. Dieser Shield erlaubt es den aus EP6 bekannten Fingerpulssensor anzuschließen. Die folgende Abbildung zeigt den Shield.



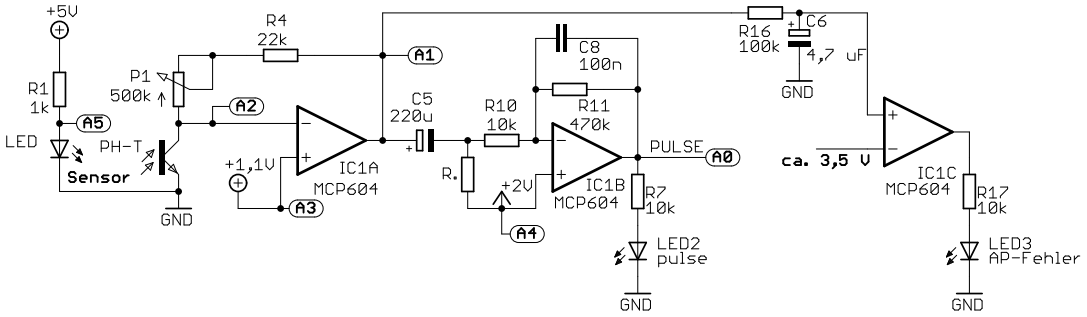
An der linken Seite ist eine Klinkenbuchse, wo Sie den Pulssensor anschließen. Mit dem Potentiometer P1 (an der rechten Seite unten) stellen Sie die Verstärkung und damit den Arbeitspunkt der ersten Verstärkerstufe ein. Drehung des Knopfes im Uhrzeigersinn erhöht die Verstärkung. Falls dadurch der Arbeitspunkt in einen unbrauchbaren Bereich gerät, leuchtet LED3 auf. Eine zweite Verstärkerstufe vergrößert das Signal nochmals und das entsprechende Signal ist an LED2 als schwankende Helligkeit sichtbar. Dieses Signal können Sie über den ADC-Kanal A0 in den Arduino einlesen. Über weitere ADC-Kanäle kann auch das Signal der ersten Verstärkerstufe sowie einige Referenzspannungen gemessen werden.

Außerdem enthält der Shield noch drei weitere Leuchtdioden LED5 (rot), LED4 (gelb) und LED1 (grün) und einen Summer für eine Tonausgabe. Diese sind durch die Software im Arduino steuerbar (siehe Tabelle unten) und können damit frei mit einer Funktion belegt werden. Die Lautstärke des Summers ist mit dem Potentiometer P2 (an der rechten Seite oben) einstellbar, Drehung des Knopfes im Uhrzeigersinn erhöht die Lautstärke.

Das folgende Blockschaltbild zeigt den vereinfachten Aufbau des Shields.



Der folgende vereinfachte Schaltplan zeigt den Aufbau der beiden Verstärkerstufen genauer. (Die eingekreisten Marker A0 .. A5 bedeuten die ADC-Kanäle).



Der Pulssensor besteht aus einer Infrarot-LED (LED) und einem Phototransistor (PH-T). Fällt Licht auf den Phototransistor, so fließt ein Photostrom, der durch einen Widerstand (P1 und R4) in eine Spannung umgewandelt wird. Der Operationsverstärker IC1A ist also als Transimpedanzverstärker geschaltet und gibt eine Spannung aus (zu messen an ADC-Kanal A1), die das Produkt aus Photostrom und Widerstand ist. Aus technischen Gründen ist diese Spannung um etwa 1,1 Volt erhöht. Diese 1,1 Volt sollten an den ADC-Eingängen A3 und A2 zu messen sein.

Die Ausgangsspannung (A1) kann nicht größer als die Versorgungsspannung des Operationsverstärkers (+5 V) werden. Fällt nun viel Licht auf den Phototransistor und wird damit der Photostrom groß, darf das Produkt mit dem Widerstand (P1 und R4) diese 5 Volt nicht erreichen, sonst geht IC1A in die Begrenzung und es ist kein Pulssignal vorhanden. P1 ist dann zu verkleinern. Zur Kontrolle, ob die Ausgangsspannung noch weit genug von +5 V entfernt ist, vergleicht daher der Komparator IC1C die gemittelte Ausgangsspannung (Filter R16, C6) mit etwa 3,5 Volt und gibt bei Überschreitung mit LED3 eine Warnung (Arbeitspunkt-Fehler) aus.

Das Signal der ersten Verstärkerstufe (A1) ist noch recht klein. Deshalb ist mit IC1B eine zweite Verstärkerstufe nachgeschaltet. Offensichtlich handelt es sich um einen inverstierenden Verstärker mit Verstärkung 47fach (Faktor R11/R10). Zur Unterdrückung hochfrequenter Störsignale ist zu R11 der Filterkondensator C8 parallelgeschaltet. Der Arbeitspunkt dieser zweiten Verstärkerstufe ist auf etwa 2 Volt (zu messen mit ADC A4) fest eingestellt. Das endgültige Pulssignal (PULSE) ist wie erwähnt an ADC A0 vorhanden.

Folgende Funktionen sind im Arduino verfügbar:

Name	**Pin**	**Standard pinMode**	Anmerkungen
LED1	D2	OUTPUT	grün, frei programmierbar
LED4	D5	OUTPUT	gelb, frei programmierbar
LED5	D6	OUTPUT	rot, frei programmierbar
SUMMER	D10	OUTPUT	frei programmierbar
Pulssignal	A0	INPUT	zwischen 0 V und 5 V (0 .. 1023 ADC-cnt)
TIA Ausgang	A1	INPUT	zwischen ca. 1,1 V und 5 Volt (ca. 225 .. 1023 ADC-cnt)
Fototransistor	A2	INPUT	ca. 1,1 V (ca. 225 ADC-cnt)
TIA Ref	A3	INPUT	ca. 1,1 V (ca. 225 ADC-cnt)
Verstärker Ref	A4	INPUT	ca. 2,0 V (ca. 410 ADC-cnt)
IR LED Monitor	A5	INPUT	ca. 1,1 V (ca. 225 ADC-cnt)

V. Versuchsdurchführung: Aufbau einer Meßkette

1. Gleitender Mittelwert mit Arduino

Bevor Sie mit der Inbetriebnahme des Fingerpulssensors beginnen, sollten Sie lernen, wie sich ein gleitender Mittelwert (moving average) in einem Mikrocontroller realisieren lässt. Dazu sollen Sie in folgendem Sketch die Funktion *movingAverage()* implementieren:

```
#define MITTELWERT_LAENGE     4

unsigned int buf[MITTELWERT_LAENGE];

unsigned int movingAverage(unsigned int newValue,
                           unsigned int *valueBuffer,
                           const unsigned int N) {
  unsigned int mean = 0;

  // !!!!!!!!!
  // Implementieren Sie hier die Mittelwertbildung
  // Der berechnete Mittelwert soll in der lokalen Variablen mean stehen!
  // !!!!!!!!!

  return mean;
}


// ab hier Testroutine für Mittelwertbildung
void setup() {
  // serielle Schnittstelle mit 115200 Baud initialisieren
  Serial.begin(115200);
}

void loop() {
  // prüfe, ob Zeichen von der seriellen Schnittstelle
  // empfangen wurden
  if(Serial.available() > 0) {
    // empfange eine Zahl und speichere sie als unsigned int
    unsigned int eingabe = Serial.parseInt();

    // wenn auch ein Zeilenvorschub (NewLine == ENTER == '\n')
    // dann verarbeite die Eingabe
    if(Serial.read() == '\n') {
      Serial.print("letzte Eingabe: ");
      Serial.print(eingabe);
      Serial.print(", Mittelwert: ");
      Serial.println(movingAverage(eingabe, buf, MITTELWERT_LAENGE));
    }
  }
}
```

Damit die Funktion möglichst generisch und damit wiederverwendbar bleibt, soll sie folgende Parameter erhalten:

- *unsigned int newValue*: nächster Wert, der für die Mittelwertbildung berücksichtigt werden soll.
- *unsigned int *valueBuffer*: Zwischenspeicher (Array) für die vorherigen Werte, die noch im Mittelwert berücksichtigt werden sollen.
- *const unsigned int N*: Konstante, die angibt, über wie viele Werte gemittelt werden soll.

Der Rückgabewert der Funktion ist dann der gleitende Mittelwert aus den letzten N Zahlen.

Hinweise:

- Bei der Mittelwertbildung müssen Sie Werte summieren. Denken Sie daran, dass die Summe aus (vielen) unsigned int-Werten den Wertebereich eines unsigned int verlassen kann. Nutzen Sie daher für die Summenbildung eine Variable vom Typ unsigned long.
- Natürlich kann es passieren, dass bei der Mittelwertbildung Nachkommastellen auftreten und damit eigentlich float als Datentyp für den Mittelwert genutzt werden müsste. Unser Mikrocontroller benötigt jedoch für Berechnungen mit Nachkommastellen deutlich länger als für reine Ganzzahlberechnungen. Wir beschränken uns daher auf ganze Zahlen und nehmen den Fehler durch Rundung hin.

Prüfen Sie die Funktionalität Ihres Programms, indem Sie folgende Zahlen nacheinander durch ENTER getrennt im Terminal-Fenster (Werkzeuge - Serieller Monitor) an den Mikrocontroller senden: 0, 0, 0, 27500, 35000, 16000, 7555, 1000, 2790, 6571

Hinweis: Achten Sie darauf, dass im Terminal-Fenster unten links neben der Baudrate *Neue Zeile* ausgewählt ist. Sonst arbeitet Ihr Programm möglicherweise nicht korrekt.

Sie sollten als Ausgabe folgende Werte für den Mittelwert bekommen:

```
letzte Eingabe: 0, Mittelwert: 0
letzte Eingabe: 0, Mittelwert: 0
letzte Eingabe: 0, Mittelwert: 0
letzte Eingabe: 27500, Mittelwert: 6875
letzte Eingabe: 35000, Mittelwert: 15625
letzte Eingabe: 16000, Mittelwert: 19625
letzte Eingabe: 7555, Mittelwert: 21513
letzte Eingabe: 1000, Mittelwert: 14888
letzte Eingabe: 2790, Mittelwert: 6836
letzte Eingabe: 6571, Mittelwert: 4479
```

2. Inbetriebnahme des Fingerpulssensors & des Arduino-Shields

Zur Inbetriebnahme des Arduino-Shield stecken Sie zunächst den Pulssensor in die dafür vorgesehene Klinkenbuchse und verbinden Sie Ihren Arduino über ein USB-Kabel mit dem PC.

Wir nutzen anschließend den in die Arduino-Entwicklungsumgebung integrierten Plotter. Sie finden ihn unter Werkzeuge - Serieller Plotter. Dieser Plotter kann Werte, die über die serielle Schnittstelle gesendet werden, visualisieren. Jede Zeile, die auf der seriellen Schnittstelle gesendet wird, entspricht einem Sample auf der x-Achse des Plotters. Es können auch mehrere komma-getrennte Werte gesendet werden. Diese werden dann in verschiedenen Farben im Plotter dargestellt.

Zunächst möchten wir uns das Ausgangssignal der beiden Verstärkerstufen anschauen. Dazu können Sie einen Sketch verwenden, der in regelmäßigen Abständen die beiden Ausgangssignale wandelt und auf der seriellen Schnittstelle ausgibt. Die regelmäßigen Abstände werden hier mithilfe der Funktion[1] millis() erzeugt. Mit dem seriellen Plotter können diese Ausgangssignale dann visualisiert werden.

```
const unsigned long sampling_period = 2; // Abtastung alle 2 ms = 500 Hz
unsigned long next_sampling = 0; // Zeitpunkt der nächsten Abtastung (in ms)

void setup() {
  // initialisiere die serielle Schnittstelle mit 115200 Baud
  Serial.begin(115200);
}

void loop() {
  // nächste Abtastung fällig?
  if(millis() >= next_sampling) {
    // lese Spannungen mit dem ADC ein
    unsigned int verst2 = analogRead(A0);
    unsigned int verst1 = analogRead(A1);

    // Ausgabe auf serielle Schnittstelle
    Serial.print(verst2);
    Serial.print(',');
    Serial.println(verst1);

    // nächste Abtastung "planen"
    next_sampling = next_sampling + sampling_period;
  }
}
```

Stecken Sie nun den Finger in den Pulssensor und prüfen Sie, wie sich das Ausgangssignal verhält. Wenn die rote LED (LEDxx, AP prüfen) leuchtet, so verändern Sie bitte die Verstärkung der ersten Verstärkerstufe mit dem Potentiometer P1. **Achtung:** Es dauert ein paar Sekunden, bis Sie das Ergebnis im seriellen Plotter sehen. Sorgen Sie dafür, dass das Signal den verfügbaren Wertebereich des ADC (0-1023) gut ausnutzt, aber nicht am oberen oder unteren Anschlag ist.

[1] In der Arduino-Bibliothek können Sie jederzeit mit der Funktion millis() den aktuellen Wert der Systemzeit in Millisekunden als *unsigned long* ermitteln. Damit haben Sie präzise Zeitmarken und können durch Vergleich mit vorgegebenen anderen Zeitmarken genaue Zeitpunkte für Aktionen definieren.

3. Filterung des Eingangssignals

Im vorherigen Versuchsteil haben Sie gesehen, dass das Signal nach der zweiten Verstärkerstufe (an Port A0) schon sehr deutlich zu erkennen ist. Die weitere Verarbeitung dieses Signals passiert ab sofort in Software. Zunächst möchten wir noch verbliebenes Rauschen unterdrücken. Das wollen wir durch zwei Maßnahmen erreichen:

- Verringerung der Abtastrate auf 100 Hz
- Filterung des Eingangssignals mit gleitendem Mittelwert

Nutzen Sie dann folgenden Sketch, um das Signal zu erfassen:

```
const unsigned long sampling_period = 10; // Abtastung alle 10 ms = 100 Hz
unsigned long next_sampling = 0; // Zeitpunkt der nächsten Abtastung (in ms)

void setup() {
  // initialisiere die serielle Schnittstelle mit 115200 Baud
  Serial.begin(115200);
}

void loop() {
  // nächste Abtastung fällig?
  if(millis() >= next_sampling) {
    // lese Spannungen mit dem ADC ein
    unsigned int PulseValue = analogRead(A0);

// Hier gehört Ihre Mittelwertbildung  hin ...
        // (nutzen Sie die Funktion aus dem ersten Versuchsteil!)
unsigned int PulseMean = 0;

// Pulssignal und Mittelwert auf Plotter ausgeben
Serial.print(PulseValue);
Serial.print(",");
Serial.println(PulseMean);

    // nächste Abtastung "planen"
    next_sampling = next_sampling + sampling_period;
  }
}
```

Die Abtastrate wurde durch Änderung der Konstanten *sampling_period* angepasst. Nutzen Sie nun Ihre Funktion movingAverage() aus dem ersten Versuchsteil, um den gleitenden Mittelwert zu berechnen. Probieren Sie verschiedene Werte für die Anzahl an Messwerten N, über die gemittelt wird, aus. **Tipp:** Beginnen Sie z.B. mit N=4.

4. Eliminierung des Gleichspannungsanteils

Sie werden feststellen, dass das Signal noch einen Gleichanteil besitzt, der sich je nach Arbeitspunkt und Position des Fingers im Pulssensor und weiteren Einflüssen ändern kann. Es gilt nun, diesen Anteil des Signals zu eliminieren, damit sich das Signal besser verabeiten lässt.

Dafür soll ein zweiter gleitender Mittelwert gebildet werden, der als Eingangssignal das gefilterte Signal erhält und über einen deutlich längeren Zeitraum mittelt. Dieser Mittelwert kann dann vom gefilterten Signal abgezogen werden. Es sollte dann das reine Pulssignal übrig bleiben. **Hinweis:** Bisher haben Sie in der Regel mit *unsigned int* gerechnet. Wenn Sie den Mittelwert vom Pulssignal abziehen, können jedoch auch negative Zahlen auftreten. Rechnen Sie daher mit *int* weiter.

5. Diskriminierung des Signals

Bisher haben Sie das Pulssignal noch weitestgehend analog betrachtet und nur minimal gefiltert. Zur Bestimmung der Pulsfrequenz müssen Sie jedoch detektieren, wann ein Pulsschlag stattgefunden hat. Am einfachsten ist es, das Signal dazu mit einer Schwelle zu vergleichen. Der Puls wird dann erkannt, wenn das Signal diese Schwelle überschreitet.

Programmieren Sie also einen Diskriminator, der einen Pulsschlag zuverlässig erkennt, indem er das mittelwertfreie Pulssignal mit einer festen Schwelle vergleicht. Abhängig vom Ergebnis des Vergleichs soll eine LED auf dem Board (LED1, LED4 oder LED5) aufleuchten und damit den Puls visualisieren. **Tipp:** Da das Signal mittelwertfrei ist, bietet sich als Detektionspunkt der Nulldurchgang des Signals an. Falls Sie Probleme mit Doppeldetektion oder ähnlich haben, können Sie auch andere Vergleichsschwellen testen.

6. Tonausgabe auf dem Summer

Mit dem auf dem Shield vorhandenen Summer können Sie auch den Pulsschlag hörbar machen. Dazu können Sie mit der Arduino-Funktion tone(pin, freq, dauer) einen Ton mit einer bestimmten Frequenz und Dauer auf einem Pin ausgeben. Wenn Sie den Parameter dauer weglassen, so wird der Ton abgespielt, bis Sie die Funktion erneut aufrufen oder aber die Funktion noTone() aufrufen.

Erweitern Sie Ihr Programm also, dass mit jedem detektierten Pulsschlag ein Ton in der Tonhöhe 1000 Hz für 100 ms ausgegeben wird.

7. Messung der Pulsfrequenz und Mittelung

In der Arduino-Bibliothek können Sie jederzeit mit der Funktion millis() den aktuellen Wert der Systemzeit in Millisekunden als *unsigned long* ermitteln. Damit haben Sie auch präzise Zeitstempel des Zeitpunkts, an dem Sie den Puls erkennen. Um nun die Pulsfrequenz zu ermitteln, ist die Dauer zwischen dem aktuellen Pulsschlag und dem vorangegangenen Pulsschlag zu messen. Das können Sie tun, indem Sie einfach den aktuellen Zeitstempel vom letzten Zeitstempel abziehen. Dann erhalten Sie die Pulsperiode in ms. Was müssen Sie tun, um die Pulsfrequenz in Schlägen pro Minute zu berechnen?

Erweitern Sie Ihr Programm also so, dass Sie nach jedem Pulsschlag die aktuelle Pulsfrequenz in Schlägen pro Minute angeben. Außerdem sollte noch ein gleitender Mittelwert über die letzten 16 Schläge erfolgen. Dazu können Sie wieder die movingAverage()-Funktion aus dem ersten Versuchsteil nutzen. Geben Sie beide Werte kommagetrennt auf der seriellen Schnittstelle aus und lassen Sie sich die Pulsfrequenz im Plotter anzeigen.

8. Optionale Aufgaben

Die folgenden Aufgaben sollten Sie bearbeiten, wenn noch Zeit ist. Sie sollen Ihnen einige Ideen liefern, wie das Programm verbessert werden kann.

8.1. Tonhöhe in Abhängigkeit der Herzfrequenz

Modulieren Sie die Tonhöhe Ihres Summers in Abhängigkeit der Herzfrequenz. Ein langsamer Herzschlag soll niedrigere Frequenzen als ein schneller erzeugen.

8.2. „Asystolie“

Sollte einige Zeit lang kein Pulssignal erkannt worden sein, so geben Sie statt des Piepsens einen Dauerton aus. (Bei manchen Shields gibt es noch winzige Instabilitäten. Sie stören überhaupt nicht die bisherigen Versuchsteile, aber es kann sein, daß ohne jedes Pulssignal (kein Finger im Sensor) alle paar Sekunden doch ein Impuls detektiert wird. Die Zeitspanne, nach der die Asystolie detektiert wird, sollte also nicht zu lang sein. Ein Ruhepuls von weniger als 30 Schlägen pro Minute ist sehr selten.

8.3. Automatische Schwellenbestimmung

Während des Versuchs haben Sie vielleicht gemerkt, dass der Nulldurchgang nicht die optimale Schwelle ist und auch eine andere feste Schwelle nicht immer erfolgreich ist. Das liegt an der Dynamik des Signals. Es ändert sich nämlich neben dem Mittelwert auch die Signalamplitude. Bauen Sie also in Ihr Programm eine Spitzenwerterkennung ein, um auch die (positive) Signalamplitude erfassen zu können. Sie können dann den Schwellenwert in Abhängigkeit der Signalamplitude festlegen (z.B. halbe Signalamplitude).