Abgabe

Dieses Übungsblatt ist bis Freitag, 16.05. um 12:00 Uhr per Email an den eigenen Tutoren abzugeben. Benennen Sie die Abgabe bitte eindeutig, z.B. "*Matrikelnummer*_Abgabe_*Blattnummer.Format*".

1 Anschließen

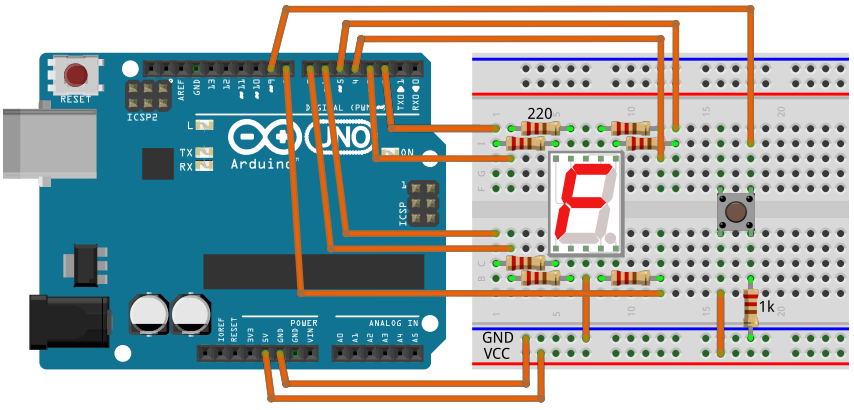
Beim Anschließen der Siebensegmentanzeige müssen Sie unbedingt auf die Verwendung von Vorwiderständen mit einem gewissen Mindestwiderstand achten. Laut Datenblatt sind für den maximal zulässigen Strom von 20mA eine Spannung zwischen 2.2 und 2.5 Volt nötig. Um im sicheren Bereich zu bleiben, dimensionieren wir die Vorwiderstände so, dass schon bei 15mA 2.8V am Widerstand abfallen, also 2.2V für die LEDs in der Anzeige bleiben. Der Widerstand muss also mindestens folgenden Wert haben:

$$R = \frac{U}{I} = \frac{2.8\,V}{0.015\,A} = 186.\overline{6}\,\Omega$$

In unserem Sortiment befindet sich ein 220Ω Widerstand, den wir hierfür gut verwenden können.

Beachten Sie außerdem, dass alle Segmente der Anzeige eine gemeinsame Kathode verwenden, die jeweils an den mittleren Pins anliegt. Es genügt also, einen dieser beiden Pins mit GND zu verbinden, und die jeweiligen Anoden über den Vorwiderstand mit einem Pin des Arduino-Boards.

Das folgende Bild zeigt den Aufbau inklusive eines Push-Buttons mit Pull-Down-Widerstand.



2 Lauflicht

Als ersten Test der Anzeige, und um die Zuordnung der Segmente auf die Pins 2-8 zu bestimmen (falls Sie das nicht bereits beim Anschließen beachtet haben) werden Sie als erstes ein Lauflicht implementieren, das im Uhrzeigersinn über die sechs äußeren Segmente laufen soll. Implementieren Sie hierfür eine *dot* Funktion analog zu der in der Vorlesung gezeigten. Bei Eingabe 0 soll ausschließlich das Segment unten links leuchten, bei 1 oben links, bei 2 dann oben und so weiter.

Nutzen Sie diese Funktion, um ein kontinuierliches Lauflicht mit einem Delay von 200 Millisekunden zu realisieren.

Die Zuordnung der Pinnummern auf die entsprechenden Segmente speichern sie am einfachsten in einem globalen Array.

Beispiellösung.

```
int seg[] = {
  2, /* unten links */
  3, /* oben links */
  4, /* oben */
  5, /* oben rechts */
  6, /* unten rechts */
  7, /* unten */
  8  /* mitte */
};

void setup() {
  /* Definiere alle Segmente als Output */
  for (int i = 0; i < 7; i++) {
    pinMode(seg[i], OUTPUT);
  }
}

void dot(int d) {
  for (int i = 0; i < 7; i++) {
    if (i == d) {
      digitalWrite(seg[i], HIGH);
    } else {
      digitalWrite(seg[i], LOW);
    }
  }
}

void loop() {
  for (int i = 0; i < 6 ; i++) {
    dot(i);
    delay(200);
  }
}
```

3 Digitaler Würfel

Nun zu dem eigentlichen Projekt: Implementieren Sie einen digitalen 10-seitigen Würfel (Wertebereich 0 bis 9). Jedes Mal, wenn der Button gedrückt wird, werden solange zufällige Zahlen in diesem Bereich erzeugt und angezeigt, bis der Button erneut gedrückt wird. Jede erzeugte Zufallszahl ist für 50 Millisekunden gültig, dann wird die nächste generiert.

Strukturieren Sie das Programm wie folgt:

1. Implementieren Sie eine Funktion *digit*, die als Parameter eine Zahl zwischen 0 und 9 entgegen nimmt und diese auf der Siebensegmentanzeige darstellt. Um den Code übersichtlicher zu halten, sollten Sie die Kodierung der Darstellung der Ziffern auf der Anzeige vom Code getrennt halten. Legen Sie hierfür ein globales, statisch initialisiertes Feld an, das diese Darstellung kodiert. Für jede Ziffer hinterlegen Sie, welche der sieben Segmente aktiv sind.

   Wenn wir die Nummerierung von unten links beginnend im Uhrzeigersinn vornehmen, dann würde das Feld z.B. so beginnen:

   ```
   // Kodierung der Darstellung der Ziffern 0-9:
   int zifferSegmente[70] = {
       HIGH, HIGH, HIGH, HIGH, HIGH, HIGH, LOW,  // Ziffer '0'
       LOW,  LOW,  LOW,  HIGH, HIGH, LOW,  LOW,  // Ziffer '1'
       ...
   };
   ```

   Nutzen Sie dieses Feld für die Implementierung von *digit*, die hierdurch sehr kompakt ausfallen sollte.

2. Implementieren Sie nun die Schleife, die alle 50 Millisekunden eine neue Zufallszahl generiert und ausgibt. Hierfür können Sie die Funktion *random*() nutzen, die auf dem Galileo-Board entgegen der Dokumentation keine Parameter entgegen nimmt, und eine zufällige positive Zahl zurückliefert. Um eine Zahl im Bereich 0 bis n (jeweils inklusive) zu generieren, berechnen Sie *random*() % $(n + 1)$.

3. Als letzten Schritt integrieren Sie den Button. Sobald er gedrückt wird, beginnt die kontinuierliche Ausgabe von Zufallszahlen. Beim nächsten Druck bleibt die gerade angezeigte Zahl auf der Anzeige stehen.

4. **Bonusaufgabe:** Ändern Sie ihr Programm so, dass bei jedem Tastendruck die Generierung von Zufallszahlen beginnt, doch die Verzögerung bis zur Generierung der nächsten Zahl immer weiter zunimmt, bis schließlich eine Zahl stehen bleibt – wie bei einem Glücksrad. Beginnen Sie wie zuvor mit einer Verzögerung von 50 Millisekunden, und stoppen Sie bei 500 Millisekunden. Der gesamte Vorgang soll etwa 5 Sekunden dauern. Zeigen Sie das Ende dadurch an, dass der Punkt in der Siebensegmentanzeige dauerhaft leuchtet (erweitern Sie die Schaltung entsprechend).

Beispiellösung.

1. + 2. Implementierung von digit und Anzeigen einer Zufallszahl alle 50 ms.

```
int seg[] = {
  2, /* unten links */
  3, /* oben links */
  4, /* oben */
  5, /* oben rechts */
  6, /* unten rechts */
  7, /* unten */
  8  /* mitte */
};

int numbers[70] = {
  HIGH, HIGH, HIGH, HIGH, HIGH, HIGH, LOW,   /* Ziffer 0 */
  LOW,  LOW,  LOW,  HIGH, HIGH, LOW,  LOW,   /* Ziffer 1 */
  HIGH, LOW,  HIGH, HIGH, LOW,  HIGH, HIGH,  /* Ziffer 2 */
  LOW,  LOW,  HIGH, HIGH, HIGH, HIGH, HIGH,  /* Ziffer 3 */
  LOW,  HIGH, LOW,  HIGH, HIGH, LOW,  HIGH,  /* Ziffer 4 */
  LOW,  HIGH, HIGH, LOW,  HIGH, HIGH, HIGH,  /* Ziffer 5 */
  HIGH, HIGH, HIGH, LOW,  HIGH, HIGH, HIGH,  /* Ziffer 6 */
  LOW,  LOW,  HIGH, HIGH, HIGH, LOW,  LOW,   /* Ziffer 7 */
  HIGH, HIGH, HIGH, HIGH, HIGH, HIGH, HIGH,  /* Ziffer 8 */
  LOW,  HIGH, HIGH, HIGH, HIGH, HIGH, HIGH,  /* Ziffer 9 */
};

void digit(int n){
  for (int i = 0; i < 7; i++) {
    digitalWrite(seg[i], numbers[n*7 + i]);
  }
}

void setup() {
  for (int i = 0; i < 7; i++) {
    pinMode(seg[i], OUTPUT);
  }
}

void loop() {
  digit(random() % 10);
  delay(50);
}
```

3. Integration des Buttons

```
int buttonPin = 9;
int generateNumber;
int displayNext;
int previousPush;

int seg[] = {
  2, /* unten links */
  3, /* oben links */
  4, /* oben */
  5, /* oben rechts */
  6, /* unten rechts */
  7, /* unten */
  8  /* mitte */
};

int numbers[70] = {
  HIGH, HIGH, HIGH, HIGH, HIGH, HIGH, LOW,   /* Ziffer 0 */
  LOW,  LOW,  LOW,  HIGH, HIGH, LOW,  LOW,   /* Ziffer 1 */
  HIGH, LOW,  HIGH, HIGH, LOW,  HIGH, HIGH,  /* Ziffer 2 */
  LOW,  LOW,  HIGH, HIGH, HIGH, HIGH, HIGH,  /* Ziffer 3 */
  LOW,  HIGH, LOW,  HIGH, HIGH, LOW,  HIGH,  /* Ziffer 4 */
  LOW,  HIGH, HIGH, LOW,  HIGH, HIGH, HIGH,  /* Ziffer 5 */
  HIGH, HIGH, HIGH, LOW,  HIGH, HIGH, HIGH,  /* Ziffer 6 */
  LOW,  LOW,  HIGH, HIGH, HIGH, LOW,  LOW,   /* Ziffer 7 */
  HIGH, HIGH, HIGH, HIGH, HIGH, HIGH, HIGH,  /* Ziffer 8 */
  LOW,  HIGH, HIGH, HIGH, HIGH, HIGH, HIGH,  /* Ziffer 9 */
};

void digit(int n){
  for (int i = 0; i < 7; i++) {
    digitalWrite(seg[i], numbers[n*7 + i]);
  }
}

void setup() {
  for (int i = 0; i < 7; i++) { pinMode(seg[i], OUTPUT); }

  pinMode(buttonPin, INPUT);
  generateNumber = 0;
  displayNext = 0;
}

void loop() {
  if (digitalRead(buttonPin) == HIGH && millis() - previousPush >= 100) {
    generateNumber = !generateNumber;
    previousPush = millis();
  }
  if (millis() > displayNext && generateNumber) {
    digit(random() % 10);
    displayNext = millis() + 50;
  }
}
```

4. Bonusaufgabe

```
/* Nur Ergaenzungen */
int dot = 10;
int delayNextNumber = 50;

void setup() {
  for (int i = 0; i < 7; i++) {
    pinMode(seg[i], OUTPUT);
  }

  pinMode(buttonPin, INPUT);
  generateNumber = 0;
  displayNext = 0;

  pinMode(dot, OUTPUT);
}

void loop() {
  if (digitalRead(buttonPin) == HIGH && millis() - previousPush >= 100) {
    generateNumber = 1;
    previousPush = millis();
    digitalWrite(dot, LOW);
  }
  if (delayNextNumber >= 500) {
    generateNumber = 0;
    delayNextNumber = 50;
    digitalWrite(dot, HIGH);
  }
  if (millis() > displayNext && generateNumber) {
    digit(random() % 10);
    displayNext = millis() + delayNextNumber;
    delayNextNumber = delayNextNumber * 1.1;
  }
}
```

4 Robustheit der Implementierung

Wie Sie in der Vorlesung gelernt haben, ist bei Feldern insbesondere darauf zu achten, immer nur mit validen Indices darauf zuzugreifen.

Stellen Sie sich zum Beispiel vor, ihr Programm wird später von einer anderen Entwicklerin erweitert. Die bedenkt jedoch nicht, dass man auf einer Siebensegmentanzeige nur einstellige Werte anzeigen kann.

Was würde bei Ihrer Implementierung der *digit* Funktion passieren, wenn sie mit Werten größer als 9 aufgerufen wird? Was passiert bei negativen Werten?

Erweitern Sie die Funktion so, dass in diesem Fall ein "E" (für "error") auf der Segmentanzeige ausgegeben wird.

Beispiellösung.

```
/* Nur Ergaenzungen */
int error[] = {
  HIGH, HIGH, HIGH, LOW,  LOW,  HIGH, HIGH
};

void digit(int n) {
  if (n < 0 || n > 9) {
    for (int i = 0; i < 7; i++){
      digitalWrite(seg[i], error[i]);
    }
  } else {
    for (int i = 0; i < 7; i++) {
      digitalWrite(seg[i], numbers[n*7 + i]);
    }
  }
}
```