Abgabe

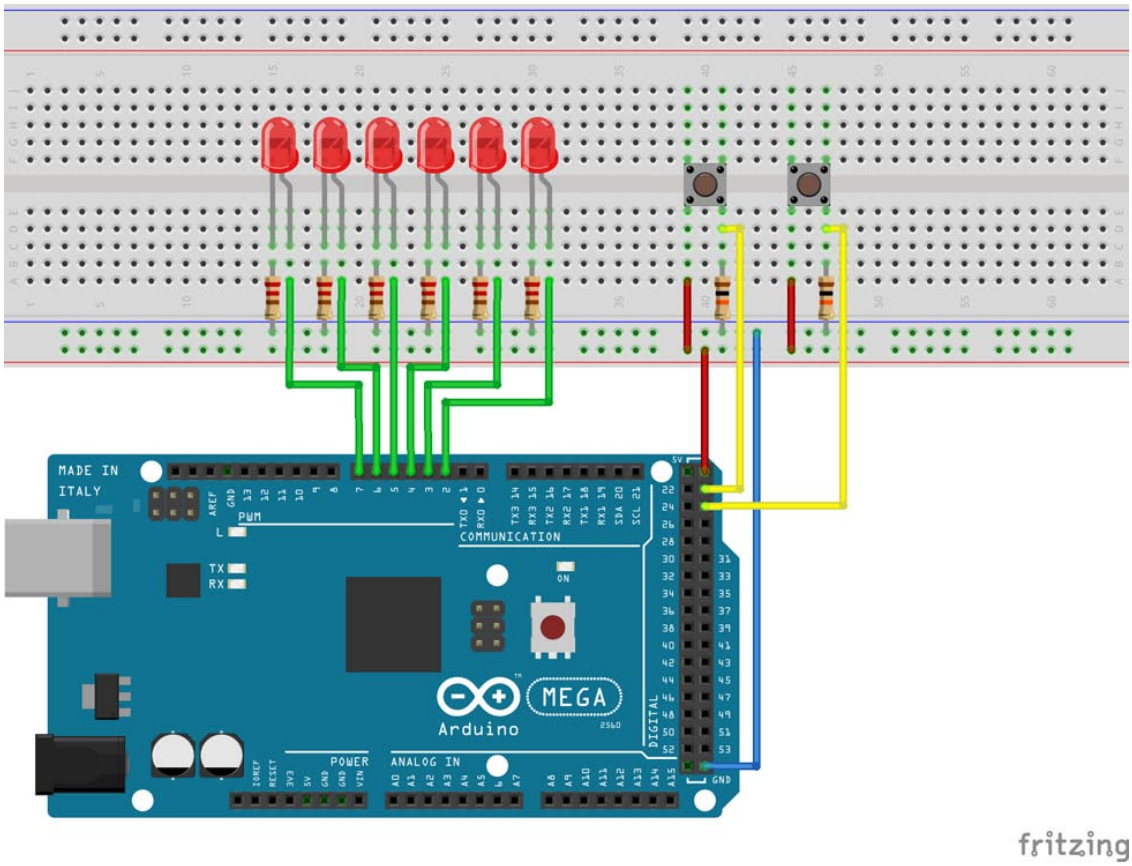
Geben Sie die Lösung bis **Montag, 12. Juni** um 23:59 als PDF-Dokument in einem ZIP, inklusive aller Quelldateien, per E-Mail an Ihren Tutor ab. Die Abgabeformalitäten sind die Gleichen wie auf Blatt 1.

1 Binärtaschenrechner (7 Punkte + 2 Bonus)

1.1 Aufbau des Taschenrechners

Ziel dieser Übung ist es einen Taschenrechner, der mit Binärzahlen rechnet, zu bauen. Hierbei dienen die sechs LEDs der Ein- und Ausgabe einer maximal 6 Bit (also 0 bis 63) großen Zahl. Hierbei handelt es sich um eine `unsigned` Zahl, also eine Zahl ohne Vorzeichen. Somit gibt es keine negativen Zahlen.

Bauen Sie zuerst die Schaltung wie in der Abbildung dargestellt auf.



1.2 Programmablauf

1. Zum Programmstart sind alle 6 LEDs aus. Durch Drücken des linken Tasters wird ein Counter hochgezählt, die aktuelle Zahl wird binär auf den LEDs ausgegeben. Hierbei steht die linke LED für 2^5 (Most Significant Bit), die rechte für 2^0 (Least Significant Bit). Wenn man die Zahl 63 erreicht hat und erneut auf den Taster drückt, dann springt der Zähler erneut auf 0
2. Nach jeder Änderung der Zahl wird dies auch auf der Anzeige dargestellt.
3. Durch Drücken des rechten Tasters wird die Zahl bestätigt und eine weitere Eingabe wird erwartet, zunächst sind alle LEDs erneut aus
4. mit dem gleichen System wie eben wird auch die zweite Zahl erstellt, ein Drücken des rechten Tasters bestätigt auch diese und addiert die beiden Zahlen. Das Ergebnis wird auf den LEDs ausgegeben
5. Wenn das Ergebnis größer als 63 ist und ein Overflow auftritt, dann blinken die LEDs zwei Sekunden lang 4 mal die Sekunde und die Eingabe wird zurück gesetzt. Das Programm befindet sich nun wieder im Ausgangszustand und eine neue Rechnung kann gestartet werden.
6. Die Addition erfolgt binär, das heißt, Sie können nicht einfach integer Zahlen aufeinander addieren, Sie müssen die Bits vergleichen und entscheiden, wie die Ergebnisbits gesetzt werden.
7. Ein erneutes Drücken des rechten Tasters führt wieder in den Ausgangszustand zu Punkt 1

1.3 Programmaufbau

- Es gibt 3 Arrays, in denen die Zahlen binär gespeichert werden, jedes Array hat 6 Einträge
- Bei jedem Aufruf von `add()` wird überprüft, ob der Output `<= 63` ist. Je nachdem gibt die Funktion dann `true` oder `false` zurück
- Auch der Counter funktioniert binär, bei jedem Drücken wird die `add()` Funktion aufgerufen und 1 dazu addiert.
- Die `clear()` Funktion dient dazu alle Arrays auf 0 zu setzen, auch wichtig bei jedem neuen Programmstart
- Entprellen der Taster ist hier sehr wichtig!
- Jedes Drücken des Linken Tasters zählt als **eine** Eingabe, kein schnelles Hochzählen

```
const int SIZE = 6;
int input_1[SIZE];
int input_2[SIZE];
int output[SIZE];

void clear(int arr[], int size) {
  //setzt alle Eintraege des uebergebenen Arrays auf 0
}

bool add(int dest[], int src_1[], int src_2[], int size) {
  //addiert die beiden binaeren Arrays src_1, src_2 in dest und ueberprueft ob
       ein Overflow auftritt, gibt true zurueck, wenn Zahl <= 63 ist
}

void setup() { ... }

void loop() { ... }
```

1.4 Pointer als Parameter

Hier ein Beispiel, wie man eine solche Funktion mit Pointern aufruft. Hier ist nichts weiter zu beachten, da Arrays immer als Pointer gespeichert werden.

```
const int SIZE = 6;
int input_1[SIZE];
int input_2[SIZE];
int output[SIZE];

//loop:
add(output, input_1, input_2, SIZE);
```

Dies hat den Vorteil, dass eine Adresse von den Variablen übergeben und somit keine Kopie erstellt wird. Wenn nun eine der Variablen in der `add()`-Funktion verändert wird, dann ist auch die globale Variable, die der Funktion übergeben wurde, verändert.

Jedoch funktionieren Funktionen wie `sizeof()` nicht bei Pointern, weshalb hier die Größe als Parameter mit übergeben wird.

1.5 Binäre Addition

Die Binäre Addition funktioniert prinzipiell genauso, wie schriftliche Addition, nur zur Basis 2.

Man beginnt mit dem letzten beiden Bits der Zahlen und addiert diese. 0 und 0 ergibt 0, 0 und 1 ergibt 1, 1 und 1 ergibt 10. Letzteres ergibt also 0 mit Übertrag 1.

Beispiel:
Addition von **001111** und **101010**, wir beginnen rechts

1, da $0 + 1 = 1$
0, da $1 + 1 = 10$, also 0 mit 1 im Übertrag
0, da $0 + 1 + 1$ (Übertrag) $= 10$, also 0 mit 1 im Übertrag
1, da $1 + 1 + 1$ (Übertrag) $= 11$, also 1 mit 1 im Übertrag
1, da $0 + 0 + 1$ (Übertrag) $= 1$
1, da $1 + 0 = 1$

Die Zahl lautet also: **111001**

Dezimal wäre dies (von rechts gelesen): $1 \cdot 2^0 + 0 \cdot 2^1 + 0 \cdot 2^2 + 1 \cdot 2^3 + 1 \cdot 2^4 + 1 \cdot 2^5 = 57$

1.6 Bonus: Schneller Zähler

Damit man schneller große Zahlen eingeben kann, soll der Code so erweitert werden, dass - sobald der Knopf länger als 500ms gedrückt ist - der Zähler 10 Mal die Sekunde erhöht wird, bis der Knopf losgelassen wird. Die Anzeige muss nun auch 10 Mal pro Sekunde aktualisiert werden.

2 Scope - Lokale und Globale Variablen (5 Punkte)

Welcher Wert wird von `Serial.print()` ausgegeben? Und warum?

Listing 1: Ein kleines Programm

```
int x = 7;

void print_x() {
  int x = 12;
  Serial.print(x);
}

//loop:
print_x();
```

Listing 2: Ein kleines Programm

```
int x = 7;

void print_x() {
  x = 12;
  Serial.print(x);
}

//loop:
print_x();
```

Listing 3: Ein kleines Programm

```
int x = 7;

void print_x() {
  { int x = 12; }
  Serial.print(x);
}

//loop:
print_x();
```

Listing 4: Ein kleines Programm

```
int x = 7;

void alternate(int newX) {
  int x = newX;
}

void print_x() {
  Serial.print(x);
}

//loop:
alternate(12);
print_x();
```

Listing 5: Ein kleines Programm

```
int x = 7;

void alternate(int newX) {
  x = newX;
}

void print_x() {
  Serial.print(x);
}

//loop:
alternate(12);
print_x();
```