Übung 10: Zustandsautomaten

Aufgabe 1: Schaltung, Vorbereitung

b) Der Code erkannt, ob der Mikrofonsensor gerade ein Geräusch erkennt oder nicht. Nur wenn er gerade ein Geräusch erkennt, leuchtet die LED. Bei normalem Hintergrundgeräuschen, sollte die LED also nicht brennen (sondern nur bei Klatschen). Deshalb dreht man so lange am Potentiometer (bei normalem Hintergrundgeräusch) bis die LED ausgeht.

Aufgabe 2: Implementierung über Case-Struktur

a) Siehe Quelltext.

```
int microPin = 24;
int ledPin = 21;

typedef enum {
  START,
  CLAP1_DETECTED,
  CLAP2_PENDING
} state_t;

state_t state;                  // current state
unsigned long timer100 = 0;     // start time of timer100
unsigned long timer300 = 0;     // start time of timer300

void setup() {
  pinMode(microPin, INPUT);
  pinMode(ledPin, OUTPUT);
  digitalWrite(ledPin, HIGH);
  state = START;   // initial state
}

void loop() {
  state_machine();
}

void state_machine() {
  switch (state) {
    case START:
      if (digitalRead(microPin) == HIGH) {
        state = CLAP1_DETECTED;
        timer100 = millis();  // start timer for 100 ms
      }
      break;
    case CLAP1_DETECTED:
        if (millis() - timer100 > 100) {     // timer 100 ms expired
          state = CLAP2_PENDING;
          timer300 = millis(); // start timer for 300 ms
        }
      break;
    case CLAP2_PENDING:
        if (millis() - timer300 > 300) {  // timer 300 ms expired
```

```
          state = START;
        }
        else if (digitalRead(microPin) == HIGH) {
          digitalWrite(ledPin, !digitalRead(ledPin)); // toggle
          state = START;
        }
      break;
  }
}
```

b) Das hochgeladene Programm hat eine Größe von 1750 Byte.

Aufgabe 3: Implementierung über Tabelle

a) Siehe Quelltext

```
int microPin = 24;
int ledPin = 21;

typedef enum {
  START,
  CLAP1_DETECTED,
  CLAP2_PENDING,
} state_t;

// events
typedef enum {
  NONE,
  MICRO_PIN_HIGH,
  TIMER100_EXPIRED,
  TIMER300_EXPIRED
} event_t;

// global variables
state_t state;                  // current state
unsigned long timer100 = 0;     // start time of timer100
unsigned long timer300 = 0;     // start time of timer300

// transition functions
void idle() {
}
void toClap1Detected() {
  state = CLAP1_DETECTED;
  timer100 = millis();  // start timer for 100 ms
}
void toClap2Pending() {
  state = CLAP2_PENDING;
  timer300 = millis(); // start timer for 300 ms
}
void toStart() {
   state = START;
}
void toStartToggle() {
   state = START;
   digitalWrite(ledPin, !digitalRead(ledPin)); // toggle LED
}
```

```
// transition table
state_t (*state_table[3][4]) (void) = {
  //                 NONE     MICRO_PIN_HIGH    TIMER100_EXPIRED  TIMER300_EXPIRED
  /*START*/          {idle,   toClap1Detected,  idle,               idle},
  /*CLAP1_DETECTED*/ {idle,   idle,             toClap2Pending,     idle},
  /*CLAP2_PENDING*/  {idle,   toStartToggle,   idle,               toStart}
};

void setup() {
  pinMode(microPin, INPUT);
  pinMode(ledPin, OUTPUT);
  state = START;   // initial state
}

void loop() {
  state_machine();
}

void state_machine() {
  // detect events
  event_t event = NONE;
  if (digitalRead(microPin) == HIGH) {
    event = MICRO_PIN_HIGH;
  }
  else if (timer100 && millis() - timer100 > 100) {  // 100 ms expired
    event = TIMER100_EXPIRED;
    timer100 = 0;     // deactivate timer
  }
  else if (timer300 && millis() - timer300 > 300) {  // 300 ms expired
    event = TIMER300_EXPIRED;
    timer300 = 0;     // deactivate timer
  }

  // use transition table to switch state
  state_table[state][event]();
}
```

Hinweis: Beachten Sie, dass die Variablen `timer100` bzw. `timer300` nach dem Auslaufen der Timer immer auf 0 zurückgesetzt werden. Die Zahl 0 signalisiert dabei, dass der Timer aktuell deaktiviert ist. So werden nicht fälschlicherweise `timer100` Ereignisse erkannt, obwohl der `timer100` aktuell gar nicht läuft.

b) 1896 Byte. Es ist leicht größer als mit Switch-Case. Ein Grund ist sicherlich, dass für bestimmte Zustände nur wenige Transitionen möglich sind.

c) Allgemein ist eine Implementeirung mit switch/case unter Umständen kompakter, wenn die Übergangstabelle nur dünn besetzt ist, also Ereignisse nur für wenige Zustände relevant sind. Dafür ist bei einer Tabelle mit Funktionszeigern direkt die Übergangstabelle ablesbar.