Arduino-Ampel

Dokumentation

Heye Hamadmad

7. März 2018, Bremen

Diese Dokumentation wurde entnommen von `http://github.com/hamadmad/arduino-ampel/`

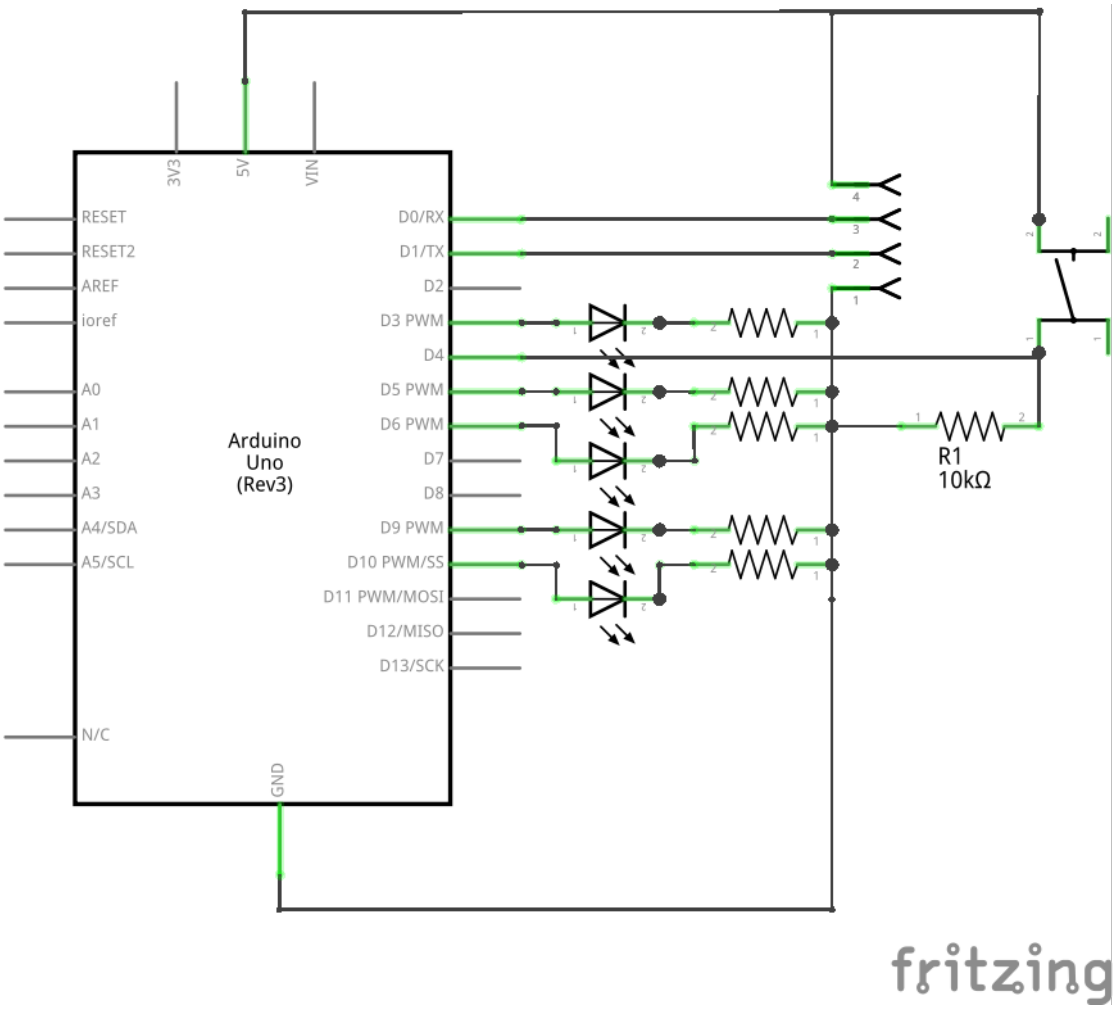
Das Projekt dient zur Demonstration einer kleinen Ampelschaltung und ist ggf. zur Realisierung eines Modellbauprojekts geeignet.

Die Hardware entspricht einem Shield für einen Arduino UNO und ist dementsprechend auch mit einem Arduino Duemilanove kompatibel. Eingesetzt werden 5 LEDs um die Lampen der Ampel zu symbolisieren, sowie ein Taster zum Betätigen der Fußgängerampel. Außerdem existiert ein Port zur Seriellen Kommunikation über z.B. ein Bluetooth-Modul mit einer zentralen Steuereinheit oder zur Meshartigen Vernetzung und Selbstorganisation mit anderen Ampeln.

Die Software basiert ausschließlich auf einer rudimentären state-machine. Funktionen zur Kommunikation mit anderen Plattformen sind **NICHT** implementiert. Da unter Umständen die PWM-Pins eingesetzt werden können, die nicht auf dem Arduino Duemilanove vorhanden sind, ist dieser softwareseitig **NICHT** unbedingt unterstützt.

1 Hardware

Die Hardware orientiert sich am folgenden Schaltplan:



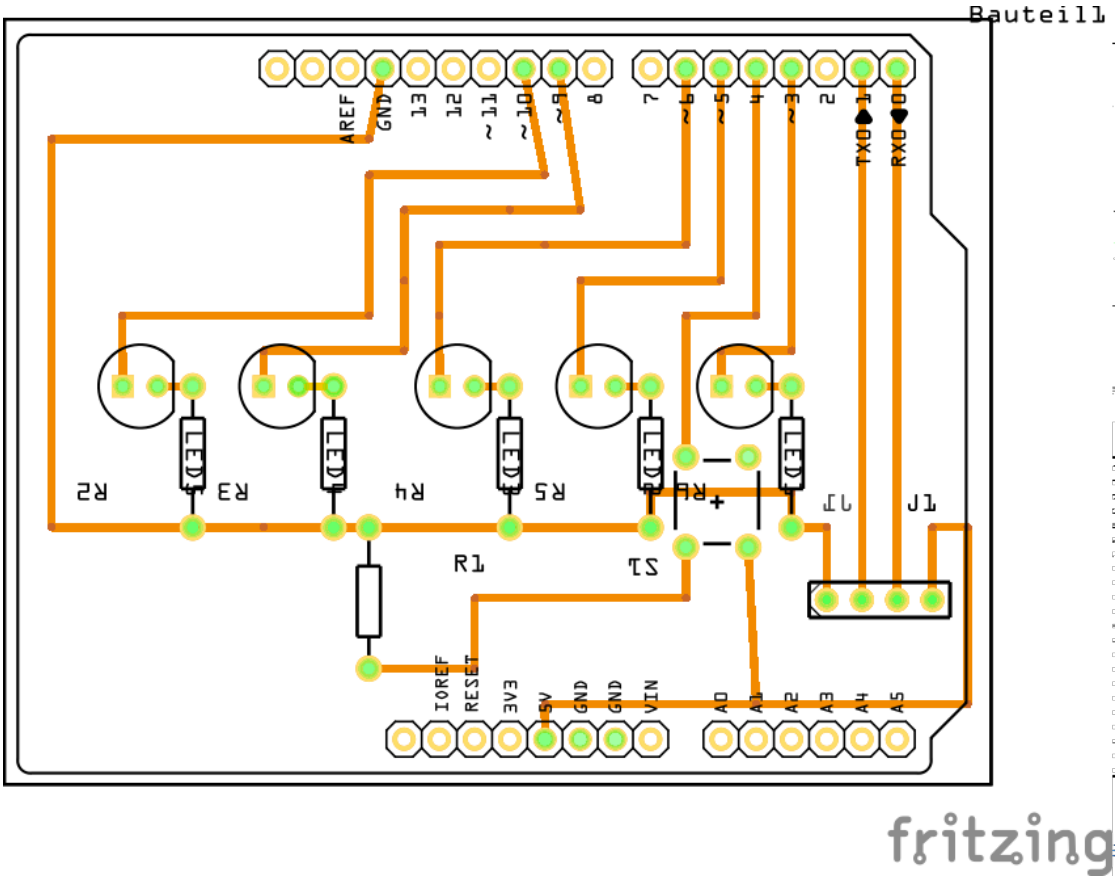
Die LEDs werden jeweils mit einem 150Ohm Widerstand an Masse geführt. Der Taster wird mit einem 10kOhm Pull-Down-Widerstand ebenfalls an Masse geführt. Der Steckverbinder wird außer an die beiden Pins zur Seriellen Kommunikation auch mit den Stromleitungen verbunden um ggf. ein Erweiterungsboard mit Strom versorgen zu können. Bei Zukünftigen Versionen könnte außerdem über eine I2C oder OneWire Verbindung nachgedacht werden.

Die folgenden Teile werden benötigt:

- 1 Arduino Uno Rev3 / Duemilanove
- 4 [OPTIONAL] Buchsenleiste (J1)
- 2 LED Rot (LED1, LED4)
- 2 LED Grün (LED2, LED5)
- 1 LED Gelb (LED 3)

- 1 10kΩ Widerstand (R1)
- 5 150Ω Widerstand (R2-6)
- 1 Taster (S1)

Das Shield kann nach dem folgenden Layout aufgebaut werden:



Die Dateien zum Anfertigen von Platinen finden sich im Unterordner hardware/board_pdf

Es ist darauf zu achten, dass statt der vorgeschlagenen LEDs und Taster auch Verlängerungskabel angebracht werden können um die eigentliche Ampelanlage klein zu halten. Des Weiteren ist die Buchsenleiste optional.

2 Software

Der Quellcode findet sich hier und außerdem im Unterverzeichnis code/code

```
#include <TimerOne.h>

// Defines for traffic light Pins

// Car traffic lights
#define CRPin 3
#define CYPin 5
#define CGPin 6
//Pedestrian traffic lights
#define PRPin 9
#define PGPin 10


// Defines for timings (In ms)
#define DURATION_RESTART 5000
#define DURATION_PREPARE_DRIVE 1000
#define DURATION_DRIVE 50000 // Don't exceed one minute
#define DURATION_PREPARE_HALT 2000
#define DURATION_WALK 15000

#define DURATION_DRIVE_AFTER_REQUEST 10000

//Misc Defines
#define BTNPin 4
#define UseRY true
// Red and yellow at the same time after red
#define AutomaticPedestrianPhase false
// Let the pedestrians go after the Duration of DURATION_DRIVE

// Variable definitions:
unsigned int mainState = 0;
unsigned int mainTimer = 0;

bool requested = false;

void setLamps(bool cr, bool cy, bool cg, bool pr, bool pg) {
  // Car traffic lights
  digitalWrite(CRPin, cr);
  digitalWrite(CYPin, cy);
  digitalWrite(CGPin, cg);
  // Pedestrian traffic lights
  digitalWrite(PRPin, pr);
  digitalWrite(PGPin, pg);
}
```

```
unsigned int mainStateMachine(unsigned int state) {
    int returnState = state;
    switch (state) {
      case 0: // Restarted state
        setLamps(1, 0, 0, 1, 0); // Both Lights red, nobody
            should go.
        if(mainTimer >= DURATION_RESTART) {
          mainTimer = 0;
          if(UseRY) {
            returnState = 1; // Let the cars prepare to drive
          }
          else {
            returnState = 2; // Let the cars drive
          }
        }
        break;
      case 1: // Cars preparing to drive
        setLamps(1, 1, 0, 1, 0);
        if(mainTimer >= DURATION_PREPARE_DRIVE) {
          mainTimer = 0;
          returnState = 2; // Let the cars drive
        }
        break;
      case 2: // Cars driving
        setLamps(0, 0, 1, 1, 0);
        if(mainTimer >= DURATION_DRIVE &&
            AutomaticPedestrianPhase) {
          mainTimer = 0;
          returnState = 3; // Let the cars prepare to halt
        }
        else if(requested && mainTimer >=
            DURATION_DRIVE_AFTER_REQUEST) {
          requested = false;
          mainTimer = 0;
          returnState = 3; // Let the cars prepare to halt
        }
        break;
      case 3: // Cars preparing to halt
        setLamps(0, 1, 0, 1, 0);
        if(mainTimer >= DURATION_PREPARE_HALT) {
          mainTimer = 0;
          returnState = 4; // Let the cars halt
```

```
        }
        break;
      case 4: // Cars halting
        setLamps(1, 0, 0, 1, 0);
        if(mainTimer >= DURATION_WALK) {
          mainTimer = 0;
          returnState = 5; // Let the cars pedestrians walk
        }
        break;
      case 5: // Pedestrians walking
        setLamps(0, 1, 0, 1, 0);
        if(mainTimer >= DURATION_WALK) {
          mainTimer = 0;
          returnState = 0; // Reset to let everyone halt
        }
        break;
      default:
        returnState = 0;
        // Something went wrong, better restart and let nobody
           go.
  }
  return returnState;
}

void incrementTimers() {
  /*
   * Unsigned ints overflow about every 65 seconds.
   * Don't set timings larger than a minute.
   * Use multiple states instead
   */
  ++mainTimer;
}

void setup() {
  Timer1.initialize(1000); //Interrupt every millisecond
  Timer1.attachInterrupt(incrementTimers);

  // Set pin modes
  pinMode(CRPin, OUTPUT);
  pinMode(CYPin, OUTPUT);
  pinMode(CGPin, OUTPUT);
  pinMode(PRPin, OUTPUT);
  pinMode(PGPin, OUTPUT);
```

```
  pinMode(BTNPin, INPUT);
}

void loop() {
  mainState = mainStateMachine(mainState);
  if(digitalRead(BTNPin) && mainState == 2) {
    //Just allow request while in phase 2
    requested = true;
  }
}
```

Der Quellcode besteht hauptsächlich aus einer state-machine in der der Ablauf der Ampelphasen definiert ist. Der Timer benötigt die Bibliothek TimerOne. Es existieren die folgenden optionalen Konfigurationsmöglichkeiten:

- UseRY Entscheidet darüber, ob die Gelbe Leuchte am Ende der Rotphase signalisieren soll, dass es gleich weitergeht. Standardmäßig aktiviert.
- AutomaticPedestrianPhase ermöglicht die automatische Unterbrechung der Grünphase für die Autofahrer ohne die Anfrage eines Fußgängers. Standardmäßig deaktiviert.

Die Zeiten für die Phasen können ebenfalls durch das Anpassen der dafür zuständigen defines geändert werden.

3 Inbetriebnahme

Ggf. müssen die Zeiten für die Ampelphasen noch im Code angepasst werden. Sobald eine Stromzufuhr vorhanden ist, läuft die Ampel autonom. Die Funktion zur Kommunikation unter Benutzung der seriellen Schnittstelle ist noch nicht implementiert.

4 Weiterhin zu beachten:

Der gesamte Quellcode wurde nicht getestet. Er sollte sich zwar kompilieren lassen, aber die Funktionsfähigkeit kann nicht garantiert werden.