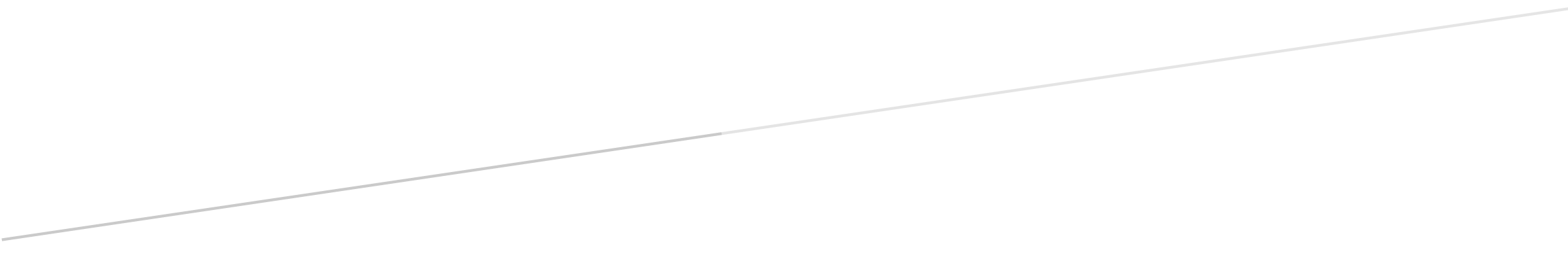
Smart Traffic Light

Bushra Yasin Ahmed Mohamed Fitim Faiku



Inhalt

[Inhalt 1](#_Toc30435150)

[Projektbeschreibung 2](#_Toc30435151)

[Blockschaltbild 2](#_Toc30435152)

[Module 2](#_Toc30435153)

[Ultraschallsensor 2](#_Toc30435154)

[1.Modus 2](#_Toc30435155)

[Berechnung der Entfernung 3](#_Toc30435156)

[Buzzer 3](#_Toc30435157)

[Clock Modul 4](#_Toc30435158)

[RTC real time clock (DS1302) 4](#_Toc30435159)

[LCD Display 6](#_Toc30435160)

[Intro 6](#_Toc30435161)

[Menü 6](#_Toc30435162)

[Anzeige der Uhrzeit 6](#_Toc30435163)

[Probleme 6](#_Toc30435164)

[Master Slave Kommunikation 7](#_Toc30435165)

[State Machine 7](#_Toc30435166)

[LED Streifen 8](#_Toc30435167)

[9](#_Toc30435168)

[Quellen 9](#_Toc30435169)

# Projektbeschreibung

Das Ziel dieses Projekts ist eine smarte Ampelsteurung zu entwickeln. Dabei werden drei Mikrokontroller verwendet. An zwei dieser Mikrokontroller werden die Ampeln angehängt und mit dem Dritten werden diese gesteuert. Des Weiteren werden Ultraschallsensoren verwendet um zu testen ob eine Person zum oder ein Auto an der jeweiligen Ampel steht. Bei der Fußgängerampel soll auch ein Buzzer eingesetzt werden der in der Grünphase angehen soll. Ebenfalls soll ein Display und ein Clock Modul verbaut werden. Das Display soll die Aktuelle Uhrzeit anzeigen und wie lange es noch dauert bis die Ampel umschaltet. Die Verbindung zwischen den drei Mikrokontrollern erfolgt über SPI.

# Blockschaltbild

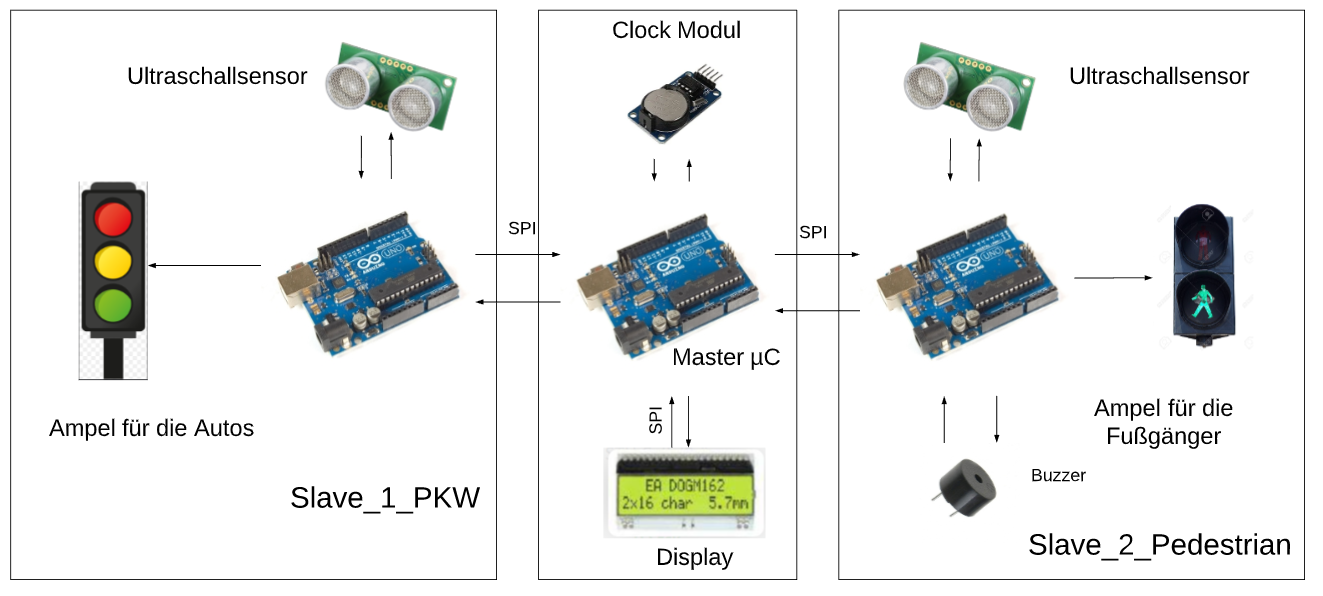


Abbildung 1 Blockschaltbild

# Module

## Ultraschallsensor

Als Ultraschallsensor wurde der SRF05 verwendet. Dieser hat eine Reichweite von 4 Meter. Der SRF05 hat zwei Modi mit denen er in Betrieb genommen werden kann. Bei der Verwendung des Sensors wurde der erste Modus benutzt.

### 1.Modus

Bei diesem Modus wird eine Messung über den Trigger Input Pin des Sensors ausgelöst, und anschließend wird auf dem Echo Output Pin das Echo der Ultraschallwelle gemessen. [1]

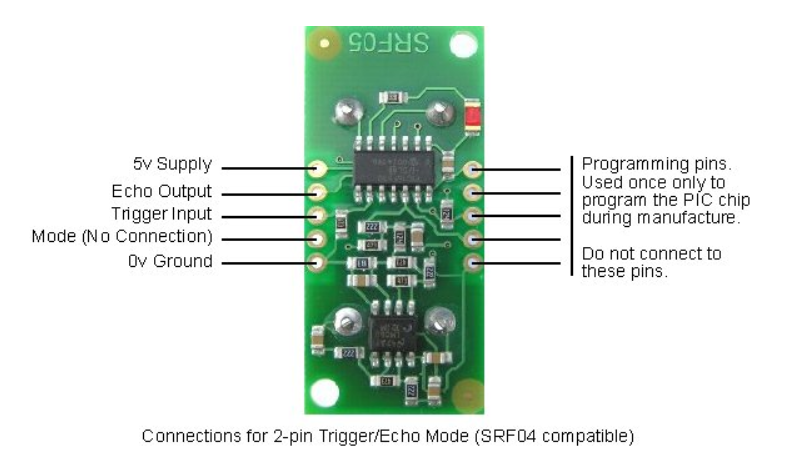


Abbildung 2 Anschlüsse des SRF05 für den 1.Modus [1]



Abbildung 3 Timing Diagramm SRF05 [1]

## Berechnung der Entfernung

Für die Erfassung der Entfernung wird die Zeit, die der Echo Output Pin des SRF05 High ist, gemessen. Um die gemessene Zeit, die in Millisekunden angegeben wird, in Zentimeter umzuwandeln wurde folgende Formel verwendet:

Distanz in cm = Zeit in Millisekunden /29/2 [2]

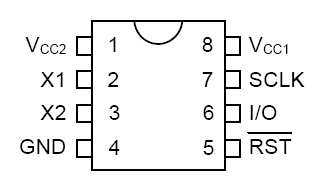
## Buzzer

Als Buzzer wurde der RS 7800712 verwendet. Diese wird mit einer Frequenz angesteuert um einen Ton zu erzeugen. Je höher die Frequenz ist, desto höher ist der Ton der erzeugt wird.

## Clock Modul

## RTC real time clock (DS1302)

Dieses Modul funktioniert wie eine physikalische Uhr. Es misst die Zeit. Wenn die real time clock nicht mit Vcc verbunden ist, läuft die RTC trotzdem weiter, sodass immer die aktuelle Uhrzeit angezeigt wird. Denn die RTC hat eine Batterie. d.h. wenn ein Stromausfall passiert, kann die Zeit weiter ohne Hindernisse erfasst werden.



Die DS1302 hat ein eigenes Protokoll (ähnlich, aber anders als I²C), das per Software auf beliebigen Pins realisiert wird. Sie hat ein sogenannte Simple Serial Interface. Man braucht dafür 3 PINS: CLK (Clock Leitung), DATA (Input/Output) und RST (CE). [3]

Jedes Mal wenn man den Register beschreiben: ein Byte (Hour, minutes, oder secondes...) auf den Register schreiben, muss das Modul resetet werden.

Bei Reset wird die Clock Leitung Low sein und CE wird Low dann High sein.

void DS13xx\_Reset(void) {

PORTD &= ~(1<<DS13xx\_SCLK);

PORTD &= ~(1<<DS13xx\_CE);

PORTD |= (1<<DS13xx\_CE);

}

Die zweite Möglichkeit den Register zu beschreiben ist Burst Mode. Hier werden alle Bytes gelesen oder geschrieben, dann wird die RTC resetet:

void DS13xx\_Write\_CLK\_Registers(void) {

DS13xx\_Reset();

DS13xx\_WriteByte(0xbe); /\* clock burst write (eight registers) \*/

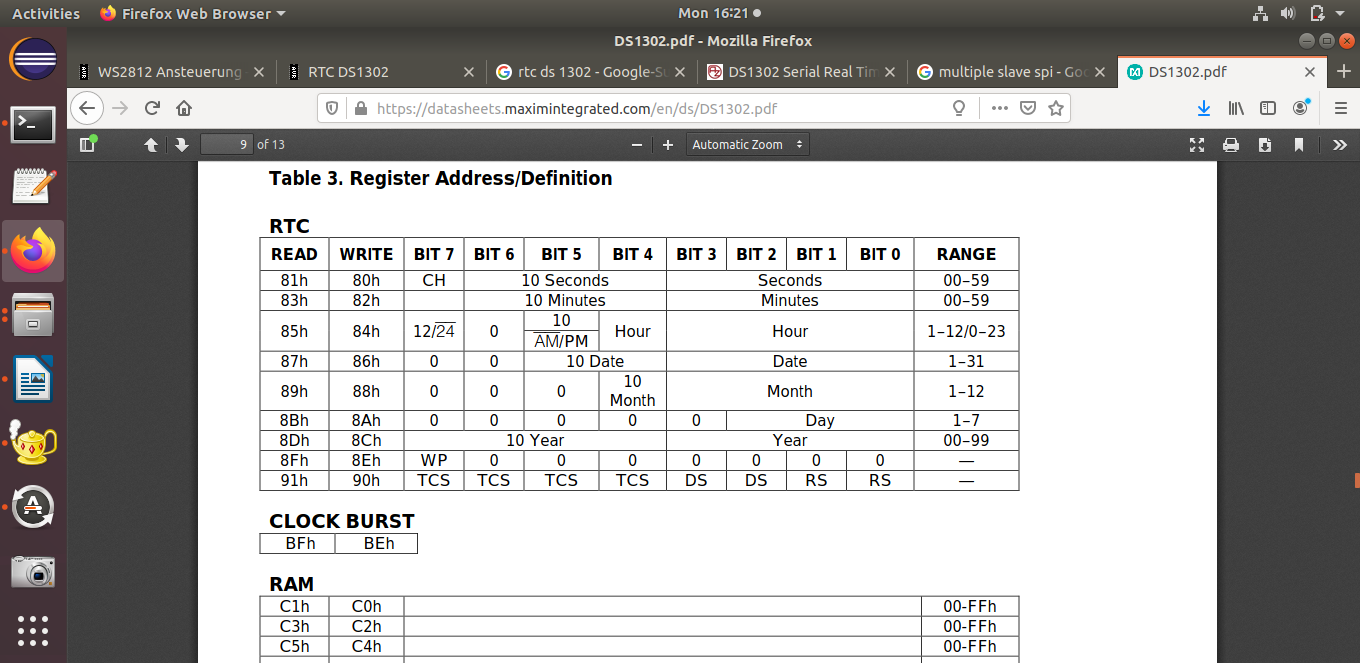
DS13xx\_WriteByte(ClockSecond);

DS13xx\_WriteByte(ClockMinute);

DS13xx\_WriteByte(ClockHour);

DS13xx\_Reset();

}



Wichtig auch bei diesem Modul ist, dass man die Data Leitung nicht anschließt, wenn man minicom startet, weil der Code von oben nach unten ausgeführt wird. D.h. DS13xx\_Init() in der main wird auch ausgeführt → die Zeit im Register wird überschrieben und das Modul beginnt die Zeit von Anfang an zu messen.

void DS13xx\_Init(void) {

// Initialise PORTA for the DS13xx clock

DDRD |= (1<<DS13xx\_CE) | (1<<DS13xx\_IO) | (1<<DS13xx\_SCLK);

DS13xx\_WriteByte(0x8E);

DS13xx\_WriteByte(0x00);

DS13xx\_Reset();

DS13xx\_WriteByte(0x80);

DS13xx\_WriteByte(0x00);

DS13xx\_Reset();

DS13xx\_WriteByte(0x81);

DS13xx\_Reset();

}

## LCD Display

### Intro

Für die Anzeige der aktuellen Uhrzeit haben wir ein 16x2 LC Display verwendet. Tatsächlich wurde das Display-Shield genommen welches auch über mehrere Buttons verfügt und auch 3 LEDs besitzt. Die Kommunikation zum LC Display erfolgt mittels SPI-Schnittstelle.



In Abbildung wird das Display-Shield dargestellt. Die Buttons können mittels den Port D2-5 angesteuert werden. Die Ports müssen nicht für die Buttons verwendet werden. Durch entfernen des Jumpers können diese anderweitig verwendet werden.

### Menü

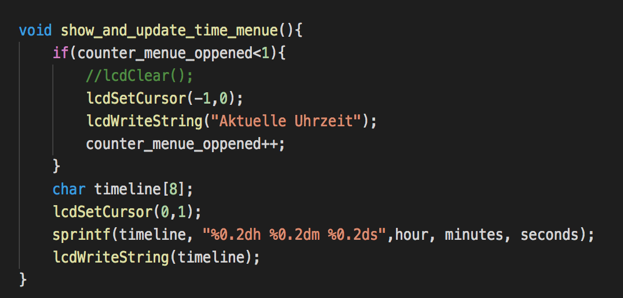
Um zwischen den Menüs wechseln zu können wurde eine Funktion erstellt welche erstmal das Display entleert und danach zwischen den Menüpunkten wechselt.

### Anzeige der Uhrzeit

Um die aktuelle Uhrzeit anzuzeigen werde mehrere volatile Variablen erstellt und im Master nach der Initialisierung der RTC mittels setter funcions gesetzt. Eine display and show menue Funktion wird im Sekundentakt aufgerufen und aktualisiert die Anzeige des Displays. In der Funktion wird der Cursor an einer Position gesetzt und die Funktion lcd\_writeString wird mit der aktuellen Uhrzeit aufgerufen.

### Probleme

Die Probleme beim LC Display waren, dass wenn mehrmals die Funktion lcd\_writeString verwendet wurde, dass die Ausgabe an den falschen Stellen erfolgt ist. Das wurde versucht mittels eines Menü Aufruf Counters zu unterbinden jedoch ist nicht gegangen.





## Master Slave Kommunikation

Der Datenaustausch bei SPI benötigt 5 Verbindungen (Bus-Leitungen):

GND

MOSI (master out, slave in)

MISO (master in, slave out)

SCK (system clock)

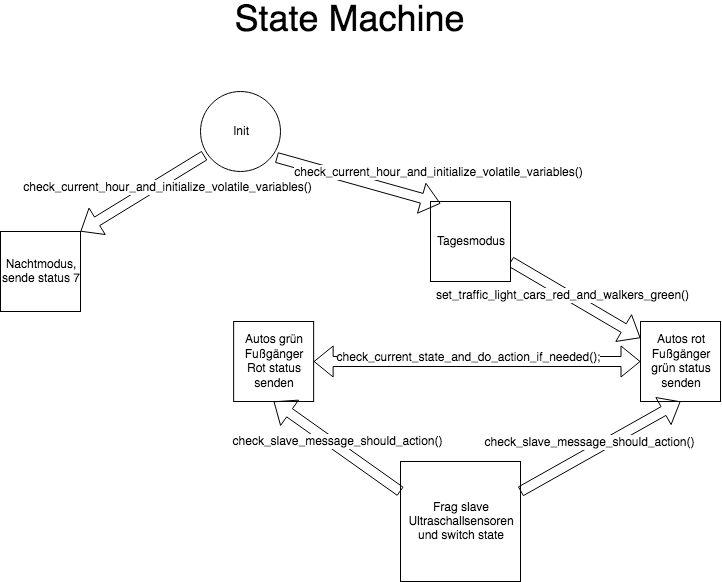
/SS (slave select)

Um zwischen den drei Arduinos zu kommunizieren haben wir SPI verwendet, wobei der Master drei verschiedene Slave Leitungen hat um die Arduinos spezifisch ansprechen zu können. Die Slaves können in den Register SPDR genauso Daten hinterlegen, welche in der Folge von dem Master interpretiert werden konnten.

Die SS Leitung für den PKW Micro Controller wurde PORTD Pin 3 verwendet und wird im Code mittels SS\_SELECT\_SLAVE\_1 selektiert. Die SS Leitung für den Fußgänger Micro Controller wurde am PORTD Pin 4 gelegt und wird im Code mittels SS\_SELECT\_SLAVE\_1 selektiert.

### State Machine

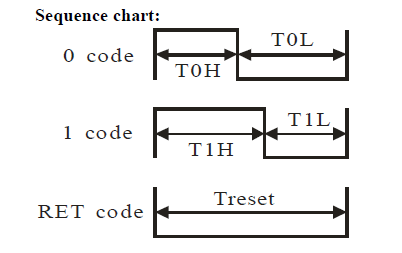
Damit die Kommunikation zwischen den Micro Controllern reibungslos funktioniert haben wir uns eine State Machine überlegt, damit die einzelnen States klar ersichtlich sind, sowie der Übergang zwischen den States auch für jeden verständlich ist.



## LED Streifen

Die Ansteuerung der WS2812 erfolgt über eine einzelne Datenleitung mit einem asynchronen seriellen Protokoll. Eine "0" wird dabei über einen kurzen, eine "1" über einen langen High-Puls definiert. Jede LED benötigt 24 Datenbits (G8:R8:B8). Die Daten aller LEDs werden seriell direkt hintereinander übertragen. Wenn die Datenleitung für mehr als 50µs auf "low" gehalten wird ("reset code"), werden die Daten in die PWM-Register der LEDs übernommen.

Die Herstellerangaben für das genaue Timing des Protokolls weichen teilweise zwischen unterschiedlichen Datenblattrevisionen ab. In der Praxis hat sich allerdings gezeigt, dass die Bauteile relativ tolerant gegenüber kleinen Timingfehlern sind. Pausen von mehr als 5 µs zwischen den Datenbits sind zu vermeiden, da diese als Reset interpretiert werden können, was eine gravierende Abweichung von dem im Datenblatt hinterlegten Wert von 50µs ist. [4]



Für dieses Projekt wurden zwei Funktionen verwendet für WS2812:

1). void ws2812\_setleds (struct cRGB \*ledarray, uint16\_t number\_of\_leds);

2). void ws2812\_setleds\_pin (struct cRGB \*ledarray, uint16\_t number\_of\_leds,uint8\_t pinmask);

Für die erste Funktion muss man die Anzahl der Leds und rgb Struct übergeben

struct cRGB rgb[unit8\_t Anzahl\_der\_PINS]; // 24 led

for (uint8\_t i = 0; i<24; i++){ // 0; i<11;

rgb[i].r = Rot; // rot ist ein Hex Wert = 0xFF

rgb[i].g = green\_0; // 0x00

rgb[i].b = blue\_0;

}

ws2812\_setleds(rgb,24);

Mit der zweiten Funktion kann man noch einen bestimmten Pin auswählen.

Ws2812 hat eine relativ große Toleranz mit der Zeit

## 

# Quellen

|  |  |
| --- | --- |
|  | <https://www.robot-electronics.co.uk/htm/srf05tech.htm> |
| [2] | <https://create.arduino.cc/projecthub/Nicholas_N/distance-measurement-with-an-ultrasonic-sensor-hy-srf05-64554e> |
| [3] | <https://datasheets.maximintegrated.com/en/ds/DS1302.pdf> |
| [4] | <https://www.mikrocontroller.net/articles/WS2812_Ansteuerung> |
|  |  |