3.3.4 Slave(Gäret von Wächter)

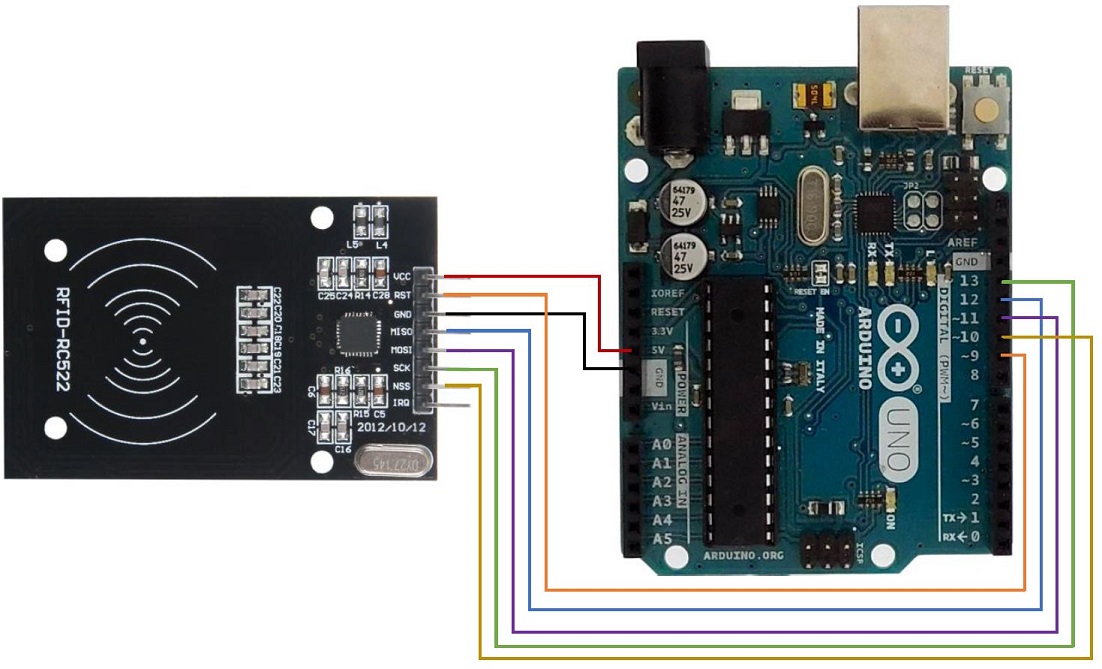
Fuer jedes Check-Point wird eine RFID-Karte implementiert. Der Wächter wird mit dem Slave die Check-Points kontroliert. Der Slave wurde gemacht ganz einfach damit der Wächter keine Problemme von der Verwendung bekommt.

Der Slave besteht aus:

* **Arduino Uno**
* **Ethernet Shield 2**
* **RFID-Reader**
* **Real-Time Clock**
* **RGB-Led**
* **LCD-Display**

Der LCD-Display mit Hilfe von RGB-Led wird den Wächter informiert ob er seinen Arbeit gemacht hat oder nicht. Er kann aber auch andere Informationen sehen wie z.B.: die Datum, das Uhr und den Tag der realen Zeit. So kann er sehen ob er zu spät ist oder noch Zeit fuer den Kontroll hat. Wenn der Blaue Led leuchten wird, wird in der LCD-Display gezeigt, dass das Gäret wartet bis auf einem Kontakt mit der Check-Point. Wenn das passiert wird, wird der Gruenne Led leuchten und die Daten in SD-Karte von Ethernet Shield speichern. Wenn der Wächter fertig mit der Rund wird, muss er das Gäret mit den Ethernet verbinden um die Daten zu Server schicken. Dann ise er fertig mit den ersten Rund.

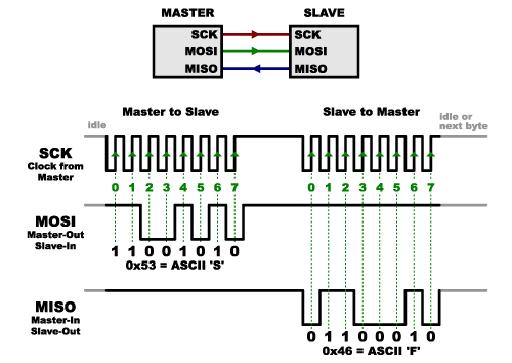
3.3.5 RFID-Karte Erkennung



Es wurde verwendet ein RFID-RC522 um die RFID-Karte zu erkennen. RFID-Reader wird komuniziert durch Serial Peripheral Interface (SPI).

**Spi** funktioniert etwas anders. Es ist ein "synchroner" Datenbus, was bedeutet, dass es getrennte Leitungen für Daten und eine "Uhr" verwendet, die beide Seiten in perfekter Synchronisation hält. Die Uhr ist ein oszillierendes Signal, das dem Empfänger genau sagt, wann die Bits auf der Datenleitung abgetastet werden müssen. Dies könnte die ansteigende (niedrig zu hoch) oder fallende (hoch zu niedrig) Flanke des Taktsignals sein; Das Datenblatt gibt an, welches zu verwenden ist. Wenn der Empfänger diese Flanke erkennt, wird er sofort auf die Datenzeile schauen, um das nächste Bit zu lesen. Da die Uhr mit den Daten gesendet wird, ist die Angabe der Geschwindigkeit nicht wichtig, obwohl die Geräte eine Höchstgeschwindigkeit haben, bei der sie arbeiten können.

Ein Grund, warum SPI so beliebt ist, ist, dass die Empfangshardware einfach sein kann [Schieberegister](https://translate.google.com/translate?hl=en&prev=_t&sl=en&tl=de&u=https://www.sparkfun.com/products/733) . Dies ist ein viel einfacheres Stück Hardware als die   satt   UART (Universal Asynchronous Receiver / Transmitter), der asynchrone serielle Schnittstelle benötigt.



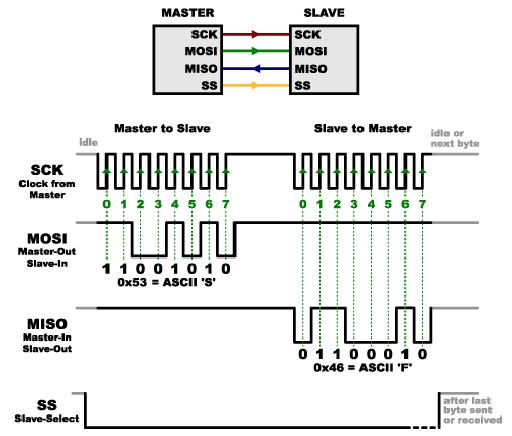
**Daten empfangen**

In SPI erzeugt nur eine Seite das Taktsignal (üblicherweise CLK oder SCK für Serial ClocK genannt ). Die Seite, die die Uhr erzeugt, wird "Master" genannt, und die andere Seite wird "Slave" genannt. Es gibt immer nur einen Master, aber es kann mehrere Slaves geben.

Wenn Daten vom Master an einen Slave gesendet werden, werden sie auf einer Datenleitung namens MOSI für "Master Out / Slave In" gesendet. Wenn der Slave eine Antwort zurück an den Master senden muss, generiert der Master weiterhin eine vorher festgelegte Anzahl von Taktzyklen, und der Slave legt die Daten auf eine dritte Datenleitung mit der Bezeichnung MISO für "Master In / Slave Out".

Da der Master immer das Taktsignal generiert, muss es im Voraus wissen, wann ein Slave Daten zurückgeben muss und wie viele Daten zurückgegeben werden. Dies unterscheidet sich erheblich von der asynchronen seriellen Schnittstelle, bei der zufällige Datenmengen jederzeit in beide Richtungen gesendet werden können. In der Praxis ist dies kein Problem, da SPI im Allgemeinen verwendet wird, um mit Sensoren zu sprechen, die eine sehr spezifische Befehlsstruktur haben.

SPI ist "Vollduplex" (hat separate Sende- und Empfangsleitungen), und daher können wir in bestimmten Situationen Daten gleichzeitig senden und empfangen . Das Datenblatt des Geräts wird zeigen, ob dies der Fallist möglich.



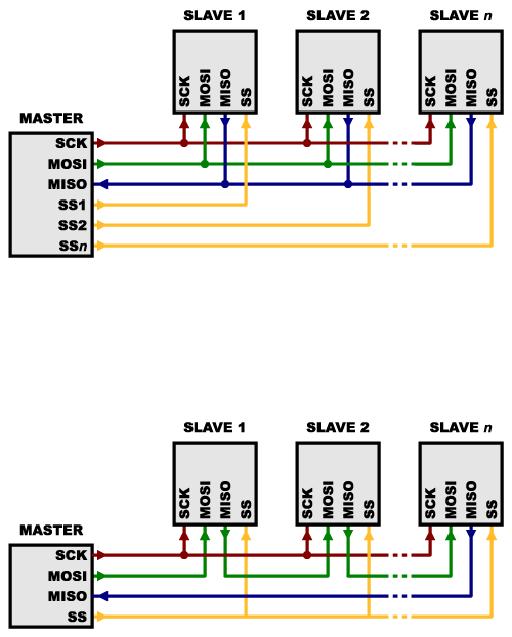
**Slave auswählen (SS)**

Slave Select (SS) teilt dem Slave mit, dass er aufwachen und Daten empfangen / senden soll. Er wird auch verwendet, wenn mehrere Slaves vorhanden sind, um denjenigen auszuwählen, mit dem wir sprechen möchten.

Die SS-Leitung wird normalerweise hoch gehalten, wodurch der Slave vom SPI-Bus getrennt wird. Kurz bevor Daten an den Slave gesendet werden, wird die Leitung auf Low geschaltet, wodurch der Slave aktiviert wird. Wenn wir mit dem Slave fertig sind, wird die Leitung wieder auf High gestellt. In einem [Schieberegister](https://translate.google.com/translate?hl=en&prev=_t&sl=en&tl=de&u=https://www.sparkfun.com/products/733) entspricht dies dem "Latch" -Eingang, der die empfangenen Daten an die Ausgangsleitungen überträgt.

**Mehrere Slaves**

Es gibt zwei Möglichkeiten, mehrere Slaves an einen SPI-Bus anzuschließen:

1. Im Allgemeinen benötigt jeder Slave eine separate SS-Leitung. Um mit einem bestimmten Sklaven zu sprechen, werden wir die SS-Linie dieses Sklaven niedrig machen und den Rest von ihnen hoch halten.
2. Auf der anderen Seite, bevorzugen einige Teile zu sein   verkettet   zusammen mit dem MISO (Ausgang) von einem, der zum MOSI (Eingang) des nächsten geht. In diesem Fall wird eine einzige SS-Leitung an alle Slaves übertragen. Sobald alle Daten gesendet sind, wird die SS-Leitung angehoben, wodurch alle Chips aktiviert werden   [gleichzeitig.](https://translate.google.com/translate?hl=en&prev=_t&sl=en&tl=de&u=https://www.sparkfun.com/products/10444) [Dies wird oft für verwendet](https://translate.google.com/translate?hl=en&prev=_t&sl=en&tl=de&u=https://www.sparkfun.com/products/10444)  [verkettet](https://translate.google.com/translate?hl=en&prev=_t&sl=en&tl=de&u=https://www.sparkfun.com/products/10444)  [Schieberegister und](https://translate.google.com/translate?hl=en&prev=_t&sl=en&tl=de&u=https://www.sparkfun.com/products/10444) [adressierbare LED-Treiber](https://translate.google.com/translate?hl=en&prev=_t&sl=en&tl=de&u=https://www.sparkfun.com/products/10444).

Bei diesem Layout laufen Daten von einem Slave zum nächsten über. Um also Daten an jeden Slave zu senden, müssen wir genug Daten übertragen, um alle zu erreichen. D er Das erste Stück Daten, das wirübertragen, endet im letzten Slave.

Diese Art von Layout wird normalerweise in verwendet   Nur Ausgabe   Situationen, wie das Ansteuern von LEDs, wo wir keine Daten zurück erhalten müssen. In diesen Fällen können wir die MISO-Leitung des Masters getrennt lassen. Wenn jedoch Daten an den Master zurückgeschickt werden muss, können wir dies tun , indem das Schließen   Daisy- Chainloop (blauer Draht im obigen Diagramm). Ich f wir dies tun, 1 die Rückkehrdaten vom Slave müssen durch alle Sklaven vergehen , bis an den Master immer wieder.

**Vorteile von SPI:**

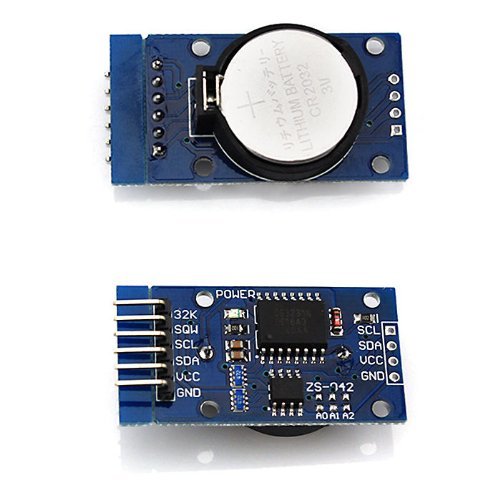
* Es ist schneller als asynchrone serielle
* Die Empfangshardware kann ein einfaches Schieberegister sein
* Es unterstützt mehrere Slaves

**Nachteile von SPI:**

* Es erfordert mehr Signalleitungen (Drähte) als andere Kommunikationsmethoden
* Die Kommunikation muss im Voraus gut definiert sein (Sie können keine zufälligen Beträge von senden Datum wann immer du willst)
* Der Master muss die gesamte Kommunikation steuern (Slaves können nicht direkt miteinander kommunizieren)
* Es erfordert normalerweise separate SS-Leitungen zu jedem Slave, was bei zahlreichen Problemen problematisch sein kann

Sklaven werden benötigt.

3.3.6 Real-Time Clock & LCD-Display



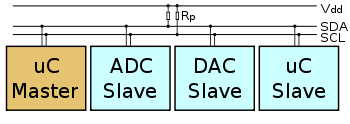
1. **Real-Time Clock**



1. **LCD-Display**

Fuer Real-Time Clock wurde die D3231 Modul und fuer das LCD-Display wurde die PCF8574AT verwendet. Real-Time Clock und LCD-Display verwenden die I2C-Protokoll.

**I2C** ist ein serielles Protokoll für Zweidraht-Schnittstellen zum Anschluss von Low-Speed- Geräten wie Mikrocontrollern, EEPROMs, A / D- und D / A-Wandlern, I / O-Schnittstellen und anderen ähnlichen Peripheriegeräten in Embedded-Systemen. Es wurde von Philips erfunden und wird jetzt von fast allen großen IC-Herstellern verwendet. Jedes I2C-Slave-Gerät benötigt eine Adresse - sie müssen noch von erhalten werden   *[NXP](https://translate.google.com/translate?hl=en&prev=_t&sl=sq&tl=de&u=http://www.nxp.com/campaigns/i2c-bus/" \t "_blank)* .



I2C-Bus ist beliebt, weil es einfach zu bedienen ist, es gibt mehr als einen Master, nur die obere Busgeschwindigkeit ist definiert und nur zwei Drähte mit Pull-up-Widerständen sind erforderlich, um eine fast unbegrenzte Anzahl von I2C-Geräten zu verbinden. I2C kann sogar langsamere Mikrocontroller mit universellen I / O-Pins verwenden, da diese nur korrekt generieren müssen   [Start- und Stoppbedingungen](https://translate.google.com/translate?hl=en&prev=_t&sl=sq&tl=de&u=http://i2c.info/i2c-bus-specification%23start-stop" \l "start-stop)  zusätzlich zu Funktionen zum Lesen und Schreiben eines Bytes.

Jedes Slave-Gerät hat eine eindeutige Adresse. Die Übertragung vom und zum Master-Gerät ist seriell und wird in 8-Bit-Pakete aufgeteilt. All diese einfachen Anforderungen machen es sehr einfach, die I2C-Schnittstelle auch mit billigen Mikrocontrollern zu implementieren , die keinen speziellen I2C-Hardware-Controller haben. Sie benötigen nur 2 freie I / O-Pins und ein paar einfache i2C-Routinen zum Senden und Empfangen von Befehlen.

Die ursprüngliche I2C-Spezifikation ist definiert als die maximale Taktfrequenz von 100 kHz. Dieser wurde später auf 400 kHz erhöht   [Fast](https://translate.google.com/translate?hl=en&prev=_t&sl=sq&tl=de&u=http://i2c.info/i2c-bus-specification%23fast-mode" \l "fast-mode) - [Modus](https://translate.google.com/translate?hl=en&prev=_t&sl=sq&tl=de&u=http://i2c.info/i2c-bus-specification%23fast-mode" \l "fast-mode) . Es gibt auch einen Hochgeschwindigkeitsmodus, der bis zu 3,4 MHz gehen kann, und es gibt auch einen 5 MHz Ultraschnellmodus.

**I2C-Schnittstelle**

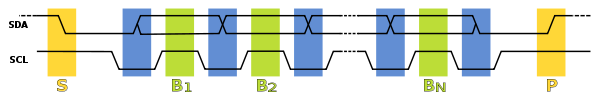
I2C verwendet nur zwei Drähte: SCL (serielle Uhr) und SDA (serielle Daten). Beide müssen mit einem Widerstand an + Vdd hochgezogen werden . Es gibt auch I2C-Level-Shifter, die zum Anschluss an zwei I2C-Busse mit unterschiedlichen Spannungen verwendet werden können.

**I2C-Adressen**

Basic I2C-Kommunikation verwendet Übertragungen von 8 Bits oder Bytes. Jedes I2C-Slave-Gerät hat a 7-Bit-Adresse Das muss im Bus einzigartig sein. Einige Geräte haben eine feste I2C-Adresse, während andere wenige Adresszeilen haben, die die unteren Bits der I2C-Adresse bestimmen. Dies macht es sehr einfach, alle I2C-Geräte mit einer eindeutigen I2C-Adresse am Bus zu haben. Es gibt auch Geräte, die haben 10-Bit-Adresse wie von der Spezifikation erlaubt.

Die 7-Bit-Adresse repräsentiert die Bits 7 bis 1, während das Bit 0 verwendet wird, um das Lesen von oder das Schreiben in das Gerät zu signalisieren. Wenn Bit 0 (im Adressbyte) auf 1 gesetzt ist, liest das Master-Gerät vom Slave-I2C-Gerät. Das Master-Gerät benötigt keine Adresse, da es die Uhr (über SCL) generiert und einzelne I2C-Slave-Geräte adressiert.

**I2C Protocoll**



Im Normalzustand sind beide Linien (SCL und SDA) hoch. Die Kommunikation wird vom Master-Gerät initiiert. Es erzeugt das   [Startbedingung](https://translate.google.com/translate?hl=en&prev=_t&sl=sq&tl=de&u=http://i2c.info/i2c-bus-specification%23start-stop" \l "start-stop)   (S) gefolgt von der Adresse der Slave-Vorrichtung (B1). Wenn das Bit 0 des Adressbytes auf 0 gesetzt wurde, schreibt das Master-Gerät in das Slave-Gerät (B2). Andernfalls wird das nächste Byte vom Slave-Gerät gelesen. Sobald alle Bytes gelesen oder geschrieben sind ( Bn ), erzeugt das Master-Gerät   [Stoppbedingung](https://translate.google.com/translate?hl=en&prev=_t&sl=sq&tl=de&u=http://i2c.info/i2c-bus-specification%23start-stop" \l "start-stop)   (P). Dies signalisiert anderen Geräten auf dem Bus, dass die Kommunikation beendet wurde, und ein anderes Gerät kann den Bus verwenden.

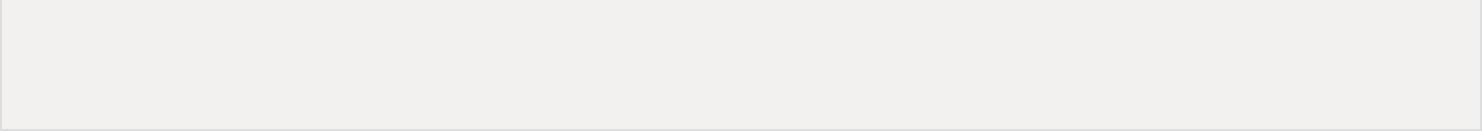
Jetzt unterstützen I2C-Geräte die Startwiederholungsbedingung. Das heißt, bevor die Kommunikation mit einer Stoppbedingung endet, kann das Master-Gerät die Startbedingung mit dem Adressbyte wiederholen und den Modus von Schreiben in Lesen ändern.

**FAZIT**

I2C-Bus wird von vielen integrierten Schaltungen verwendet und ist einfach zu implementieren. Jeder Mikrocontroller kann mit I2C-Geräten kommunizieren, auch wenn er keine spezielle I2C-Schnittstelle hat.  [I2C-Spezifikationen](https://translate.google.com/translate?hl=en&prev=_t&sl=sq&tl=de&u=http://i2c.info/i2c-bus-specification)   sind flexibel -   I2C-Bus kann mit langsamen Geräten kommunizieren und kann auch Hochgeschwindigkeitsmodi verwenden, um große Datenmengen zu übertragen. Wegen der vielen Vorteile wird der I2C-Bus eine der beliebtesten seriellen Schnittstellen bleiben, um integrierte Schaltkreise auf der Platine zu verbinden.

3.3.7 Die Programmierung von das Arduino

Am Anfang wenn das Gaeret starten wird, wird in das LCD-Display „Welcome to Guard Watch!“ geschrieben und der Code wird in void Setup geschrieben. Zuerst wird „Welcome“ in die zweite Zeile geschrieben und nach 0.75 Sekunden wird „to Guard Watch“ in die dritte Zeile dann nach 2 Sekunden kommt void loop.

*lcd.setCursor(6,1); //Start at character 7 on line 2*

*lcd.print("Welcome");*

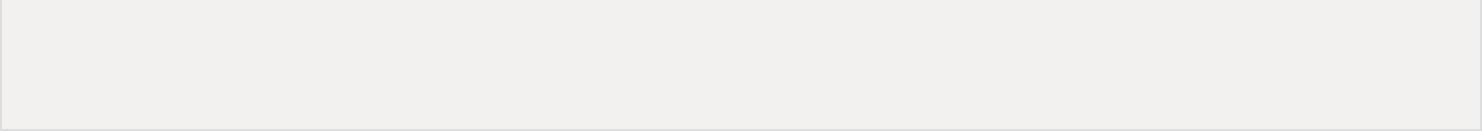
*delay(750);*

*lcd.setCursor(2,2); //Start at character 3 on line 3*

*lcd.print("to Guard Watch!");*

*delay(2000);*

In void loop, wird zuerst gesagt, dass der Waechter in die Check-Points gehen muss. Der RFID-Reader wird fuer ein Kontakt mit die RFID-Karte gewartet. Im diesen Moment wird die blaue LED-Diode leuchten. Unten wird der Tag, die Datum und die Uhr in der realen Zeit gezeigt. Es wird jede Sekunde die Uhr gezeigt.

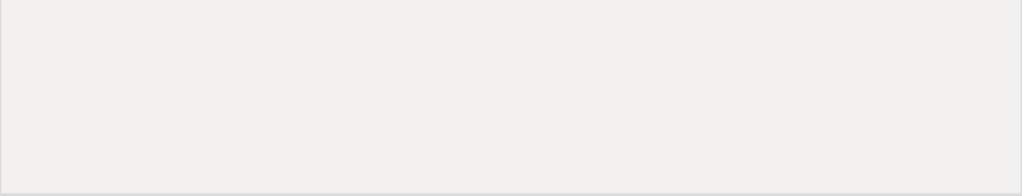
**

*lcd.setCursor(3,0); //Start at character 4 on line 1*

*lcd.print("Put the device");*

*lcd.setCursor(1,1); //Start at character 2 on line 2*

*lcd.print("on the Checkpoint!");*

**

*lcd.setCursor(0,2); //Start at character 1 on line 3*

*lcd.print(rtc.getDOWStr());*

*lcd.setCursor(8,2); //Start at character 9 on line 3*

*lcd.setCursor(10,2); //Start at character 11 on line 3*

*lcd.print(rtc.getDateStr());*

*lcd.setCursor(6,3); //Start at character 7 on line 4*

*lcd.print(rtc.getTimeStr());*

*delay(1000);*

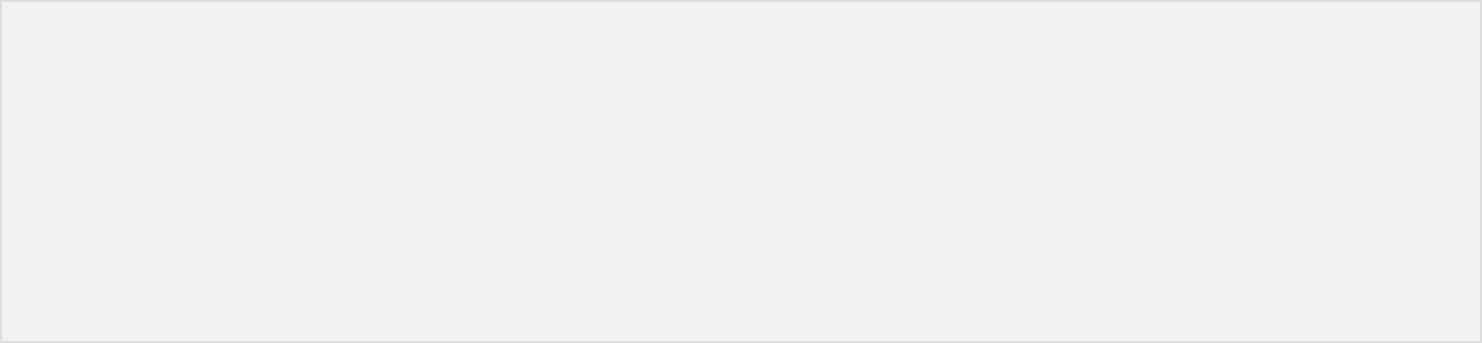
*lcd.clear();*

*digitalWrite(b\_pin,HIGH); // Blue light on*

*digitalWrite(g\_pin,LOW); // Green light off*

*digitalWrite(r\_pin,LOW); // Red light off*

Wenn ein Kontakt mit der Check-Point gegeben wird, wird dann die Gruenne LED-Diode leuchten und in LCD-Display wird geschrieben „Card Detected“.



*// Look for new cards*

*if ( ! rfid.PICC\_IsNewCardPresent()){*

*return;*

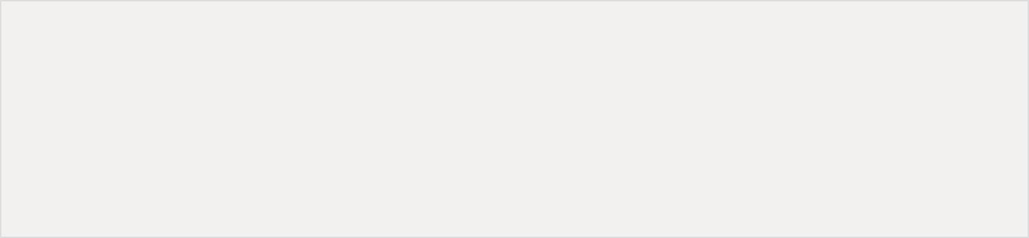
*}*

*// Verify if the NUID has been readed*

*if ( ! rfid.PICC\_ReadCardSerial()){*

*return;*

*}*

**

*lcd.clear();*

*lcd.setCursor(8,1); //Start at character 3 on line 1*

*lcd.print("Card");*

*lcd.setCursor(6,2); //Start at character 3 on line 2*

*lcd.print("Detected!");*

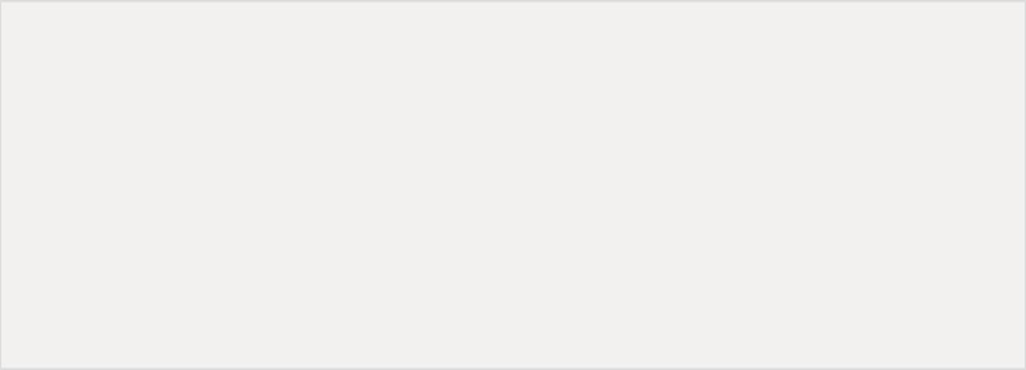
*// Green light on*

*digitalWrite(g\_pin,HIGH);*

*digitalWrite(r\_pin,LOW);*

*digitalWrite(b\_pin,LOW);*

Nach der Kontakt wird der Code von die RFID-Karte in ein byte Array gespeichert. Mithilfe von eine for Schleife, werden die Bytes vier mal in das Array gespeichern und so wurde vier zweier HEX blocke als die RFID-Code sein. Wenn der RFID-Code lesen wird, wird der Code, die Datum und die Uhr gezeigt.

**

*for (byte i = 0; i < 4; i++) {*

*nuidPICC[i] = rfid.uid.uidByte[i];*

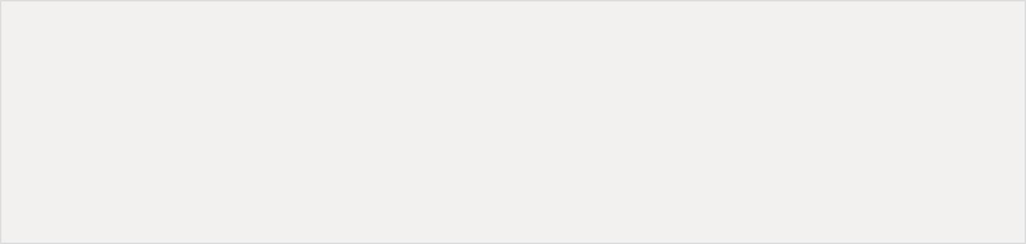
*}*

*printHex(rfid.uid.uidByte, rfid.uid.size);*

*Serial.println();*

*rfid.PICC\_HaltA();*

*rfid.PCD\_StopCrypto1();*

**

*Serial.print(" ");*

*Serial.print(rtc.getDateStr());*

*Serial.print(" ");*

*Serial.print(rtc.getTimeStr());*