
Smart Lock



MARTEDÌ GAETANO M63/1226

SALZILLO BIAGIO M63/1227



Smart Lock

1. Problema
2. Soluzione
3. Implementazione
4. Demo

Problema



Quante volte restiamo bloccati fuori casa perché abbiamo dimenticato le chiavi all'interno?

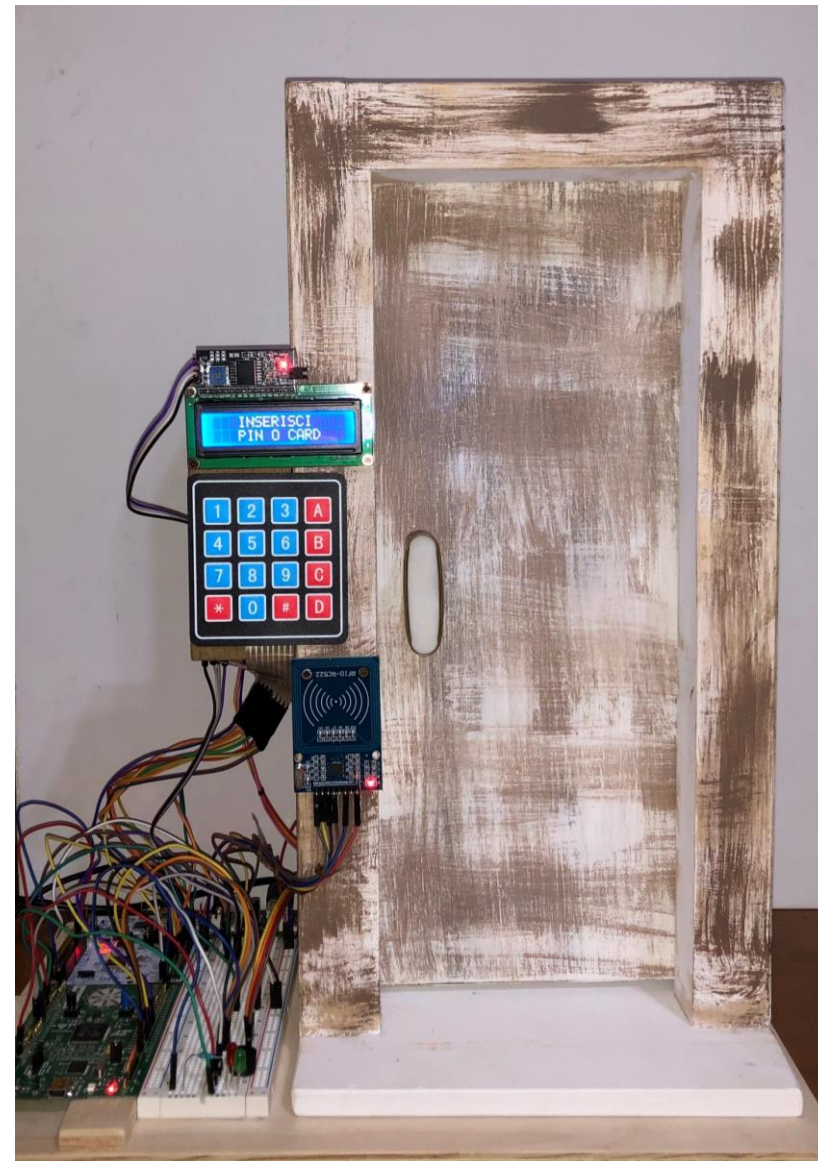
Quante volte non sappiamo dove mettere le chiavi per via di tasche o borse troppo piene?

Quante volte cercare un mazzo di chiavi dentro una borsa si rivela essere un'impresa estenuante?



Soluzione

Smart Lock

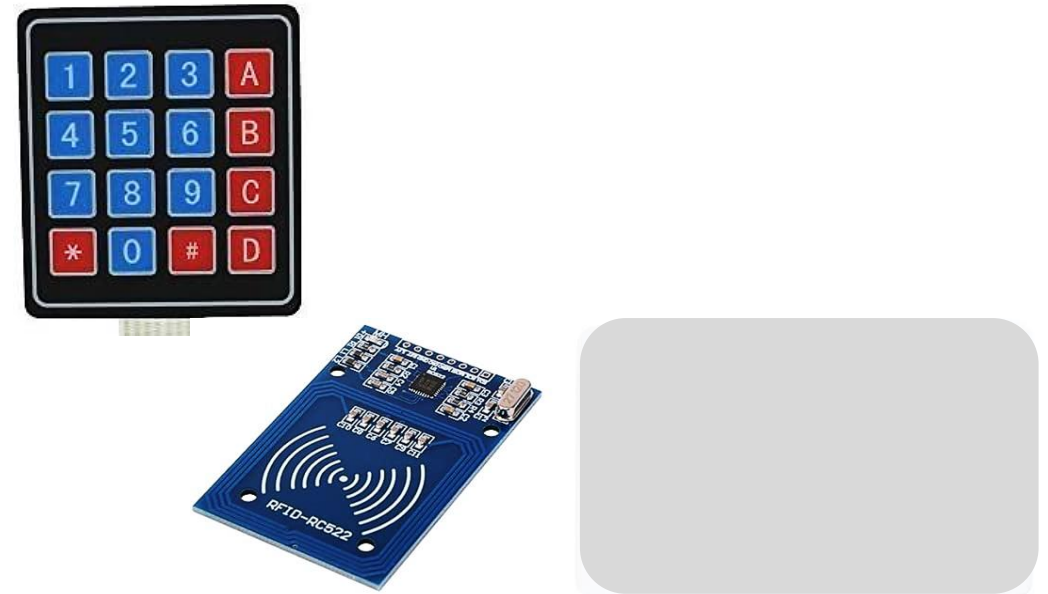


Soluzione

Smart Lock è la serratura elettronica che consente di aprire e chiudere la porta di casa senza la necessità di avere con sé le obsolete chiavi in metallo.

È possibile aprire Smart Lock scegliendo tra due metodi differenti:

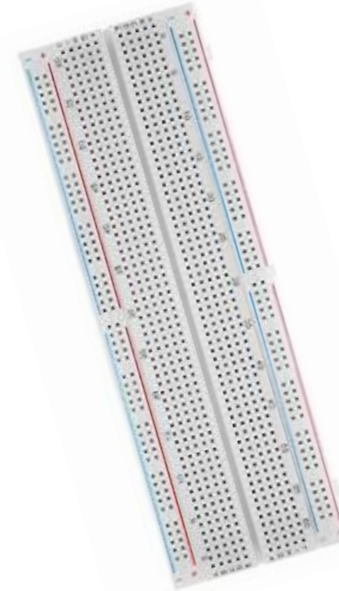
- Inserendo un PIN segreto
- Avvicinando la propria Smart Card al sensore RFID



Implementazione

Componenti:

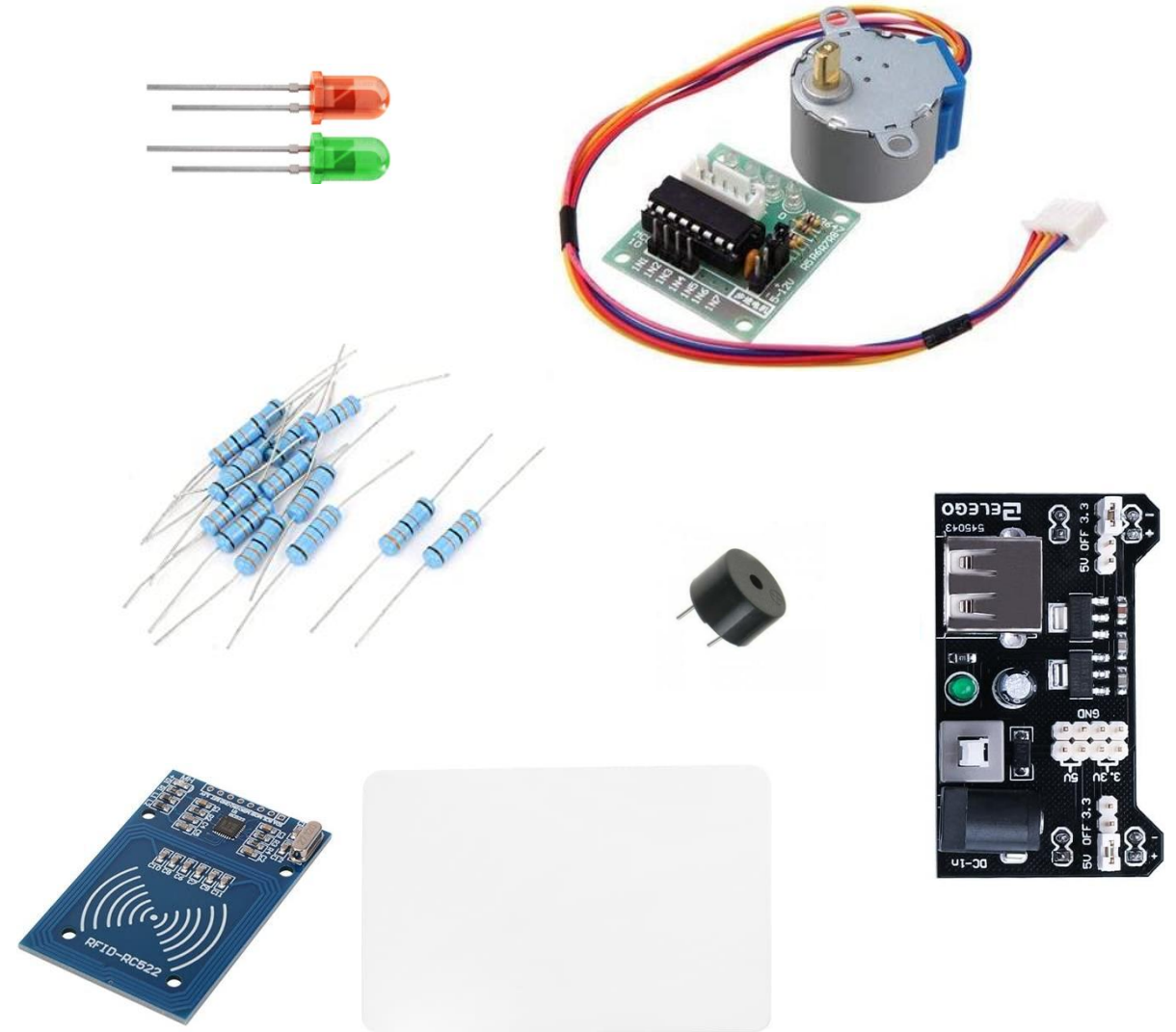
- Nucleo-64 STM32F401 (x1)
- Discovery Kit STM32F3 (x1)
- Keypad (x1)
- Breadboard (x1)
- Display LCD HD44780U (x1)
- Board I²C PCF8574 (x1)



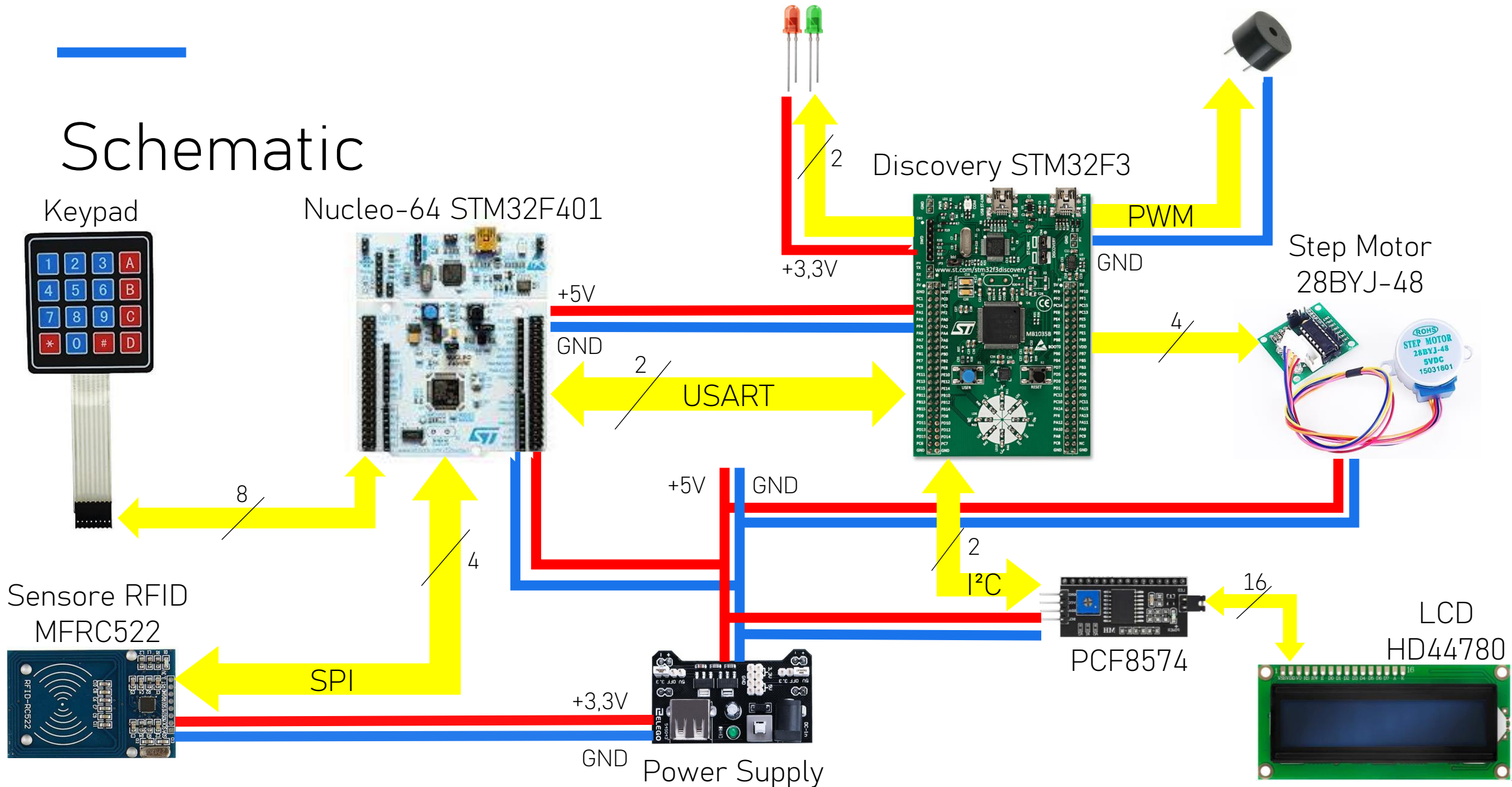
Implementazione

Componenti:

- Sensore RFID MFRC522 (x1)
- Smart Card MF1S50 (x1)
- Stepper Motor 28BYJ-48 (x1)
- Led (x2)
- Resistori (x2)
- Power Supply Module (x1)
- Passive buzzer (x1)



Schematic



Alla scheda Nucleo-64 STM32F401 sono stati collegati tutti i dispositivi di input, mentre alla scheda Discovery STM32F3 tutti quelli di output.

Le due schede comunicano tra loro tramite protocollo seriale.

INPUT

Nucleo-64 STM32F401



OUTPUT

Discovery STM32F3

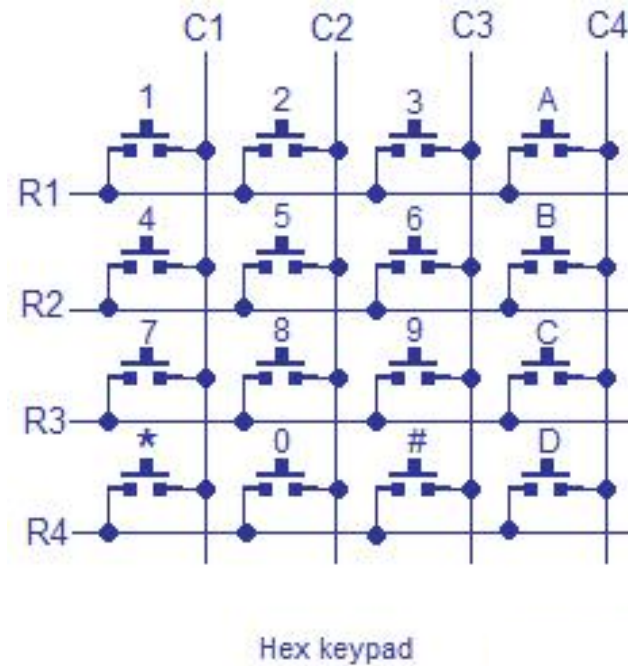


Keypad

Il keypad utilizzato nel progetto è composto da un tastierino 4x4.

Esso presenta 8 linee (pin), 4 per le righe e 4 per le colonne.

Ogni volta che un tasto (i,j) viene premuto, si crea un collegamento tra la riga i-esima e la colonna j-esima.



Keypad

GPIO Mode and Configuration

Configuration

Group By Peripherals

SPI SYS USART NVIC
GPIO Single Mapped Signals RCC

Search Signals
Search (Ctrl+F) ☐ Show only Modified Pins

Pin N...	Signal on ...	GPIO out...	GPIO mode	GPIO Pull...	Maximum...	User Label	Modified
PC7	n/a	Low	Output P...	No pull-up...	Low	r2_keypad	✓
PC8	n/a	n/a	External I...	Pull-up	n/a	c1_keypad	✓
PC9	n/a	n/a	External I...	Pull-up	n/a	c0_keypad	✓
PC10	n/a	Low	Output P...	No pull-up...	Low	red_led	✓
PC11	n/a	Low	Output P...	No pull-up...	Low	green_led	✓
PC13-AN...	n/a	n/a	External I...	No pull-up...	n/a	B1 fBlue ...	✓

GPIO mode: External Interrupt Mode with Falling edge trigger detection

GPIO Pull-up/Pull-down: Pull-up

User Label: c0_keypad

Per utilizzare il keypad sulla scheda Nucleo-64 sfruttando il meccanismo delle interruzioni sono stati effettuati i seguenti collegamenti:

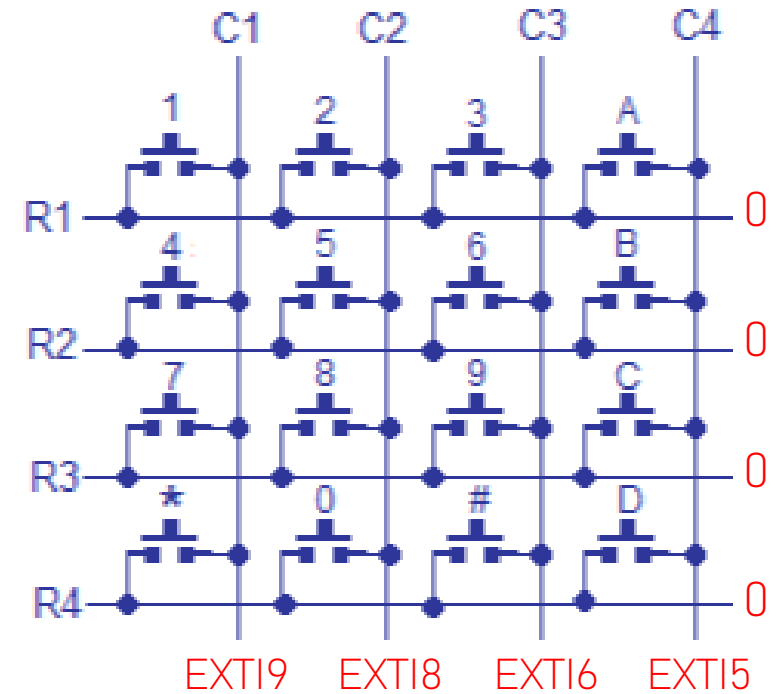
- Le linee relative alle righe sono state collegate a dei GPIO configurati in output con livello in uscita Low.
- Le linee relative alle colonne sono state collegate a dei GPIO configurati in modalità External Interrupt (EXTI) con il trigger sul fronte di discesa del segnale in ingresso.

I GPIO relativi alle colonne, nel momento in cui nessun tasto viene premuto, hanno un valore di tensione indefinito che potrebbe causare la generazione di interrupt indesiderati. Per evitare ciò, e in generale per evitare di avere valori di tensione indefiniti nel progetto, i GPIO relativi alle colonne del keypad sono dotati di un resistore di pull-up che mantiene tali linee al valore logico High, finché su di esse non verrà collocato un segnale esterno.

Keypad

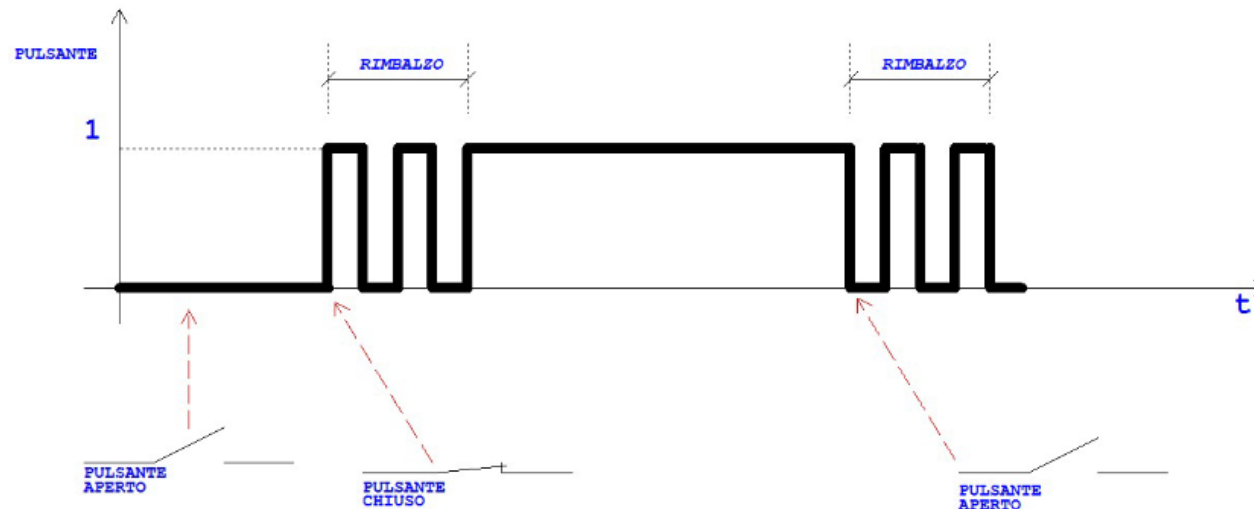
Funzionamento:

1. Viene premuto il tasto (i,j)
2. Si crea un collegamento tra la riga i e la colonna j
3. La linea relativa alla colonna j si abbassa
4. Viene generato un interrupt verso il processore
5. Viene eseguita l'ISR relativa alle interruzioni del keypad che esegue:
 1. Una funzione per individuare la riga a cui appartiene il tasto premuto
 2. Una funzione per ottenere il carattere associato al tasto premuto
 3. Una funzione che invia il suddetto carattere verso la scheda di output



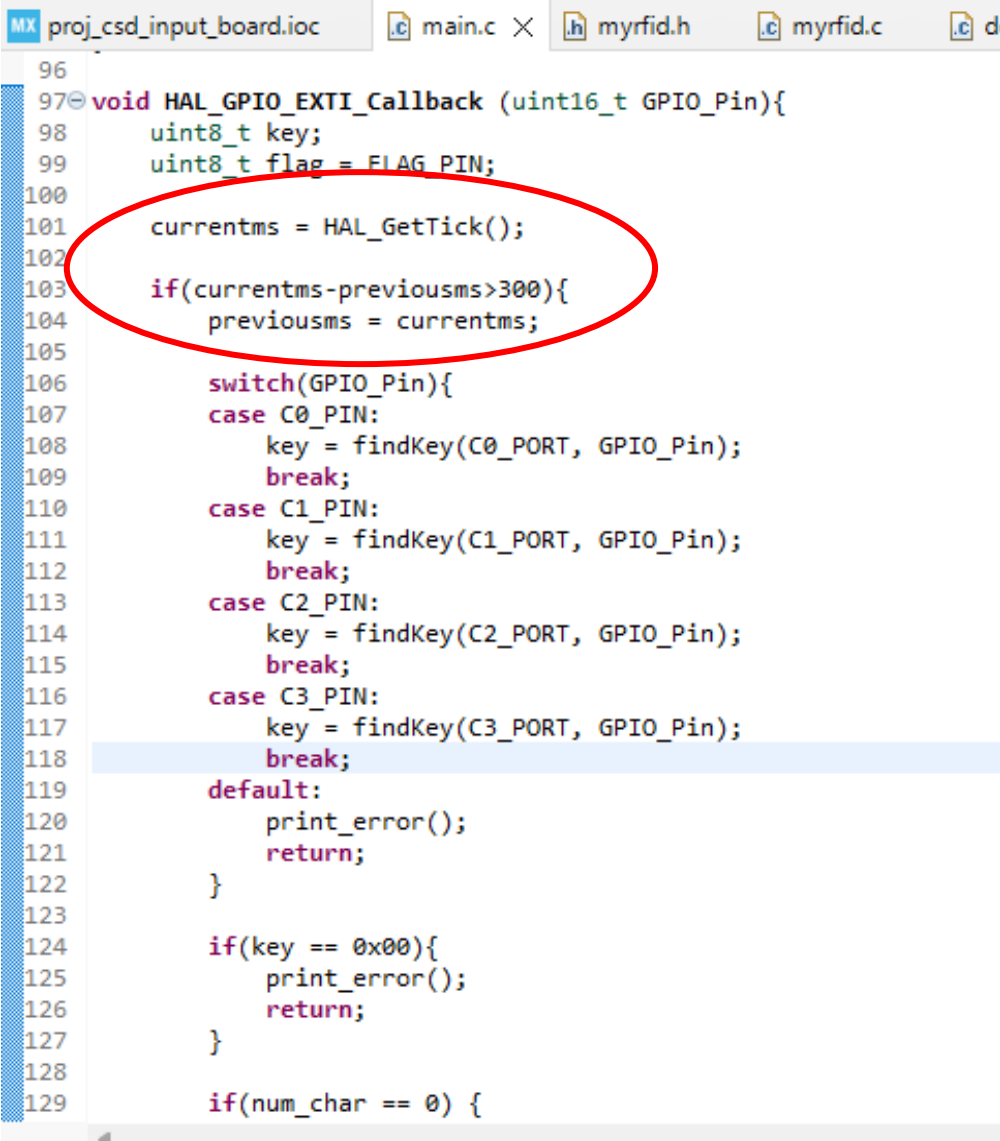
Keypad

Per utilizzare il keypad non ci resta che risolvere il problema elettro-meccanico del **flickering** o **rimbalzo** del segnale elettrico. In pratica, nel momento in cui viene chiuso o aperto un contatto fisico, come un bottone, si generano delle transizioni spurie nel segnale elettrico che attraversa il circuito, che nel nostro caso possono causare la generazione di numerosi interrupt verso il processore per ogni singolo tasto premuto.



Keypad

Per risolvere tale problematica si è adottata una tecnica di **debouncing** software, che consiste nel servire solo interruzioni che siano intervallate tra loro per almeno 300ms (valore trovato con approccio euristico).



```
96
97 void HAL_GPIO_EXTI_Callback (uint16_t GPIO_Pin){
98     uint8_t key;
99     uint8_t flag = FLAG_PIN;
100
101     currentms = HAL_GetTick();
102
103     if(currentms-previousms>300){
104         previousms = currentms;
105
106         switch(GPIO_Pin){
107             case C0_PIN:
108                 key = findKey(C0_PORT, GPIO_Pin);
109                 break;
110             case C1_PIN:
111                 key = findKey(C1_PORT, GPIO_Pin);
112                 break;
113             case C2_PIN:
114                 key = findKey(C2_PORT, GPIO_Pin);
115                 break;
116             case C3_PIN:
117                 key = findKey(C3_PORT, GPIO_Pin);
118                 break;
119             default:
120                 print_error();
121                 return;
122         }
123
124         if(key == 0x00){
125             print_error();
126             return;
127         }
128
129         if(num_char == 0) {
```

Sensore RFID MFRC522

Il sensore MFRC522 è un circuito integrato di lettura/scrittura per la comunicazione contactless a 13,56 MHz. Tale sensore viene utilizzato per implementare un sistema di comunicazione RFID.

L'**identificazione a radiofrequenza** (in inglese **Radio-Frequency IDentification**, acronimo **RFID**) è una tecnologia di riconoscimento e validazione e/o memorizzazione di informazioni a distanza. Essa si basa sulla memorizzazione di dati in particolari dispositivi elettronici passivi (o attivi), capaci di rispondere a chiamate di prossimità da parte di dispositivi attivi chiamati reader o lettori.



RFID

Nello specifico un sistema RFID è costituito da tre elementi fondamentali:

- uno o più etichette RFID (o **tag** o **transponder**, **PICC**) → Smart Card
- un apparecchio di lettura e/o scrittura (**lettore** o **PCD**) → Sensore MFRC522
- un sistema informativo per il trasferimento dei dati da e verso il lettore → Nucleo-64 STM32F4

Tag RFID



Il **tag RFID** o **etichetta** è un dispositivo elettronico composto da un chip e un'antenna RF montati su di un substrato. Il chip è la parte «smart» del dispositivo costituita da una memoria non volatile (tipicamente EEPROM), e contenente un codice univoco di identificazione (UID), il quale viene trasmesso tramite l'antenna RF all'apparato lettore, che leggerà i dati ricevuti e/o li aggiornerà. L'antenna del tag riceve un segnale emesso dal lettore, che tramite il principio di induzione è convertito in energia elettrica per alimentare il microchip. Il tag così attivato trasmette i dati in esso contenuti, attraverso l'antenna RF, al reader.



Sensore RFID MFRC522

Caratteristiche:

- Distanza operativa in modalità read/write fino a 50 mm
- Interfacce host supportate:
 - SPI fino a 10 Mbit/s
 - I²C-bus fino a 400 kBd in Fast mode, fino a 3400 kBd in High-speed mode
 - RS232 Serial UART fino a 1228,8 kBd
- Buffer FIFO contenente fino a 64 byte in ricezione e trasmissione
- Power-down by software mode
- Timer programmabile
- 2.5 V to 3.3 V power supply
- Coprocessore CRC

Sensore RFID MFRC522

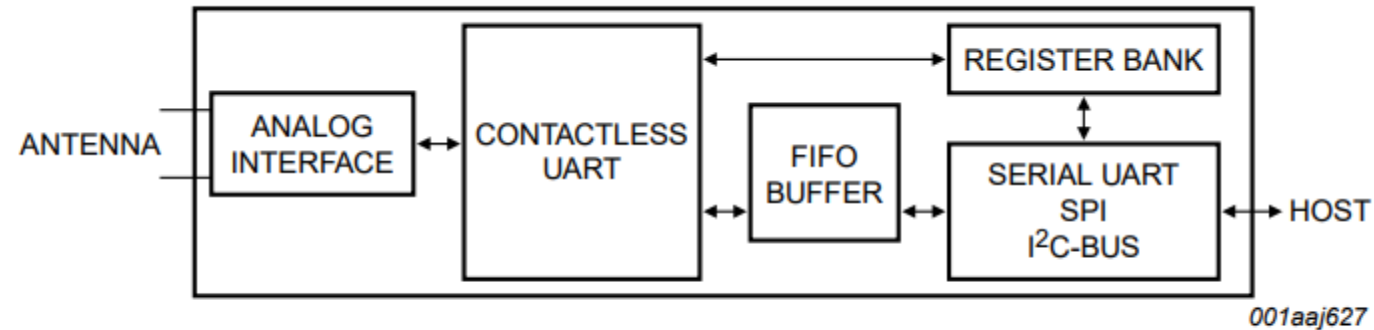


Fig 1. Simplified block diagram of the MFRC522

Il sensore presenta 2 interfacce differenti: una digitale rivolta verso il microcontrollore (SPI, UART o I²C) e un'altra analogica rivolta verso i tag RFID. Quest'ultima implementa un protocollo di comunicazione seriale attraverso una UART «senza fili».



SPI

L'interfaccia SPI (Serial Peripheral Interface), venne originariamente ideata dalla Motorola e, a differenza dello standard I²C, non è mai stata standardizzata.

Il bus SPI si definisce:

- di tipo **seriale**
- **sincrono**, con una linea dedicata per trasmettere il segnale di clock
- **full-duplex**

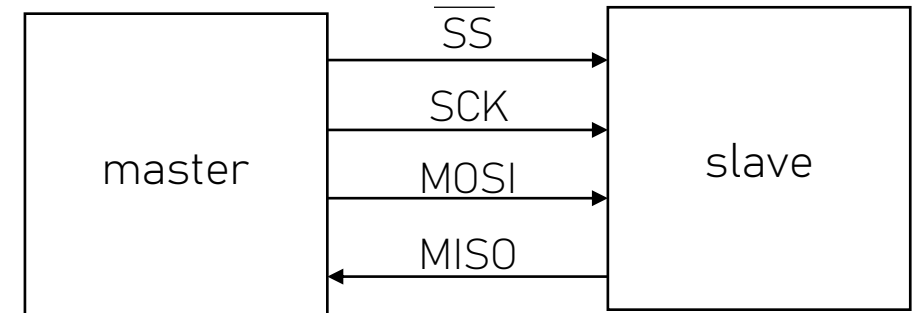
La comunicazione avviene tra un dispositivo **master**, che si occupa di iniziare la comunicazione e generare il segnale di clock, ed uno **slave**.

SPI

L'interfaccia presenta 4 linee di collegamento (esclusa la massa comunque necessaria), per cui lo standard SPI è anche noto come **4 Wire Interface**.

Le linee sono normalmente denominate come segue:

- **SS (CS, nSS)**: slave Select o Chip Select
- **SCK**: Serial Clock
- **MOSI**: master Output slave Input
- **MISO**: master Input slave Output



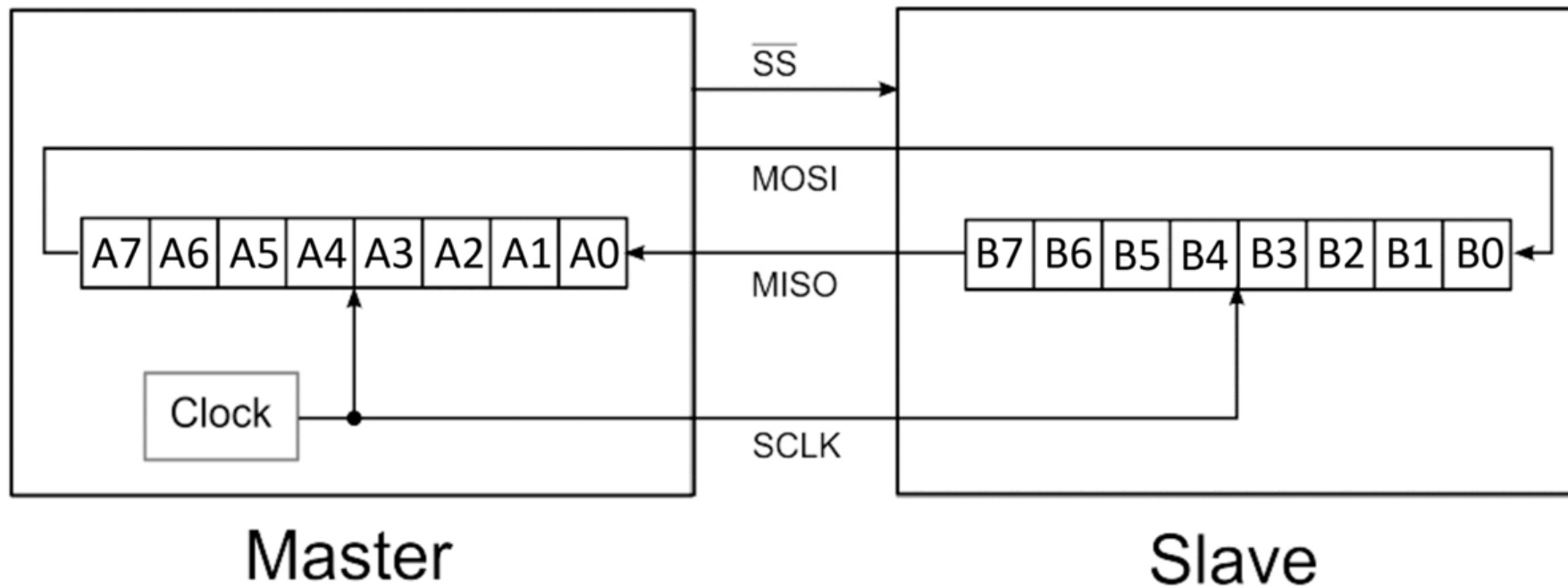


SPI

Un modulo di un microcontrollore che implementi l'interfaccia SPI si presenta come un registro a scorrimento (**Shift Register**) sia in una periferica master che slave. La dimensione del registro a scorrimento non è specificata, ma spesso è di un solo byte.

Prima di avviare una comunicazione, il master attiva la linea SS relativa allo slave con cui vuole effettuare la comunicazione e successivamente fornisce il clock alla frequenza con cui avverrà la trasmissione.

SPI



Nel momento in cui il master voglia leggere un dato dallo slave deve comunque inviargli un dato fittizio e, allo stesso modo, se il master volesse inviare dati verso lo slave, riceverà comunque dei dati fittizi da quest'ultimo.

SPI

L'interfaccia SPI può essere impostata per trasmettere o ricevere in 4 modalità differenti. La modalità selezionata deve essere la stessa sia per il master che per lo slave. Dal momento che lo slave spesso non ha modo di cambiare la modalità di comunicazione, è il master a doversi adeguare.

Le 4 modalità sono normalmente impostate per mezzo di due parametri (spesso implementati con due bit):

- **CPOL (Clock Polarity)**: indica il livello logico assunto dalla linea SCK a riposo
 1. CPOL = 0, livello di riposo Low
 2. CPOL = 1, livello di riposo High
- **CPHA (Clock Phase)**: indica su quale fronte del clock devono essere campionati i dati sulle linee MISO e MOSI
 1. CPHA = 0, sul primo fronte di clock incontrato dopo l'attivazione della linea SS
 2. CPHA = 1, sul secondo fronte di clock incontrato dopo l'attivazione della linea SS

SPI

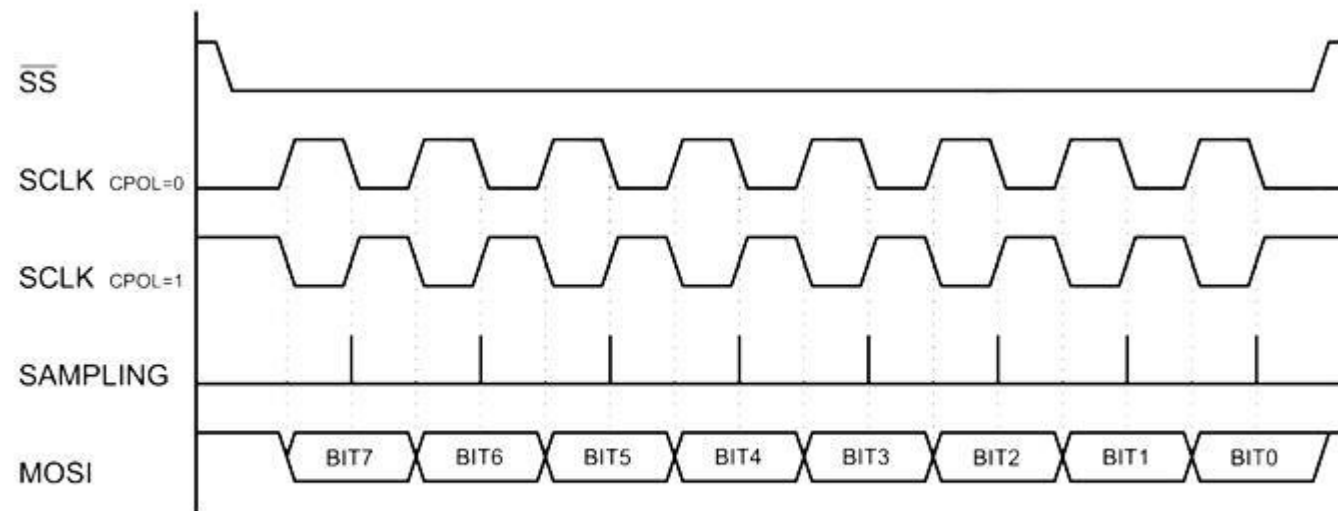


Figura 141: Diagramma temporale con il parametro $CPHA=1$.

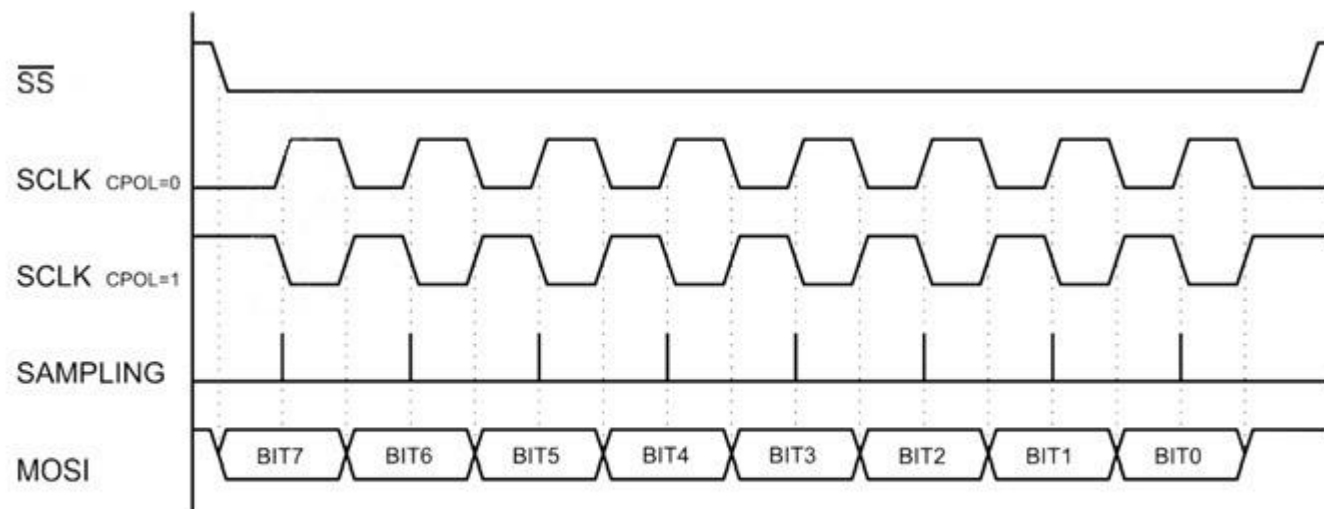


Figura 5: Diagramma temporale con il parametro $CPHA=0$.

© 2006 The Authors
Journal compilation © 2006 Blackwell Publishing Ltd

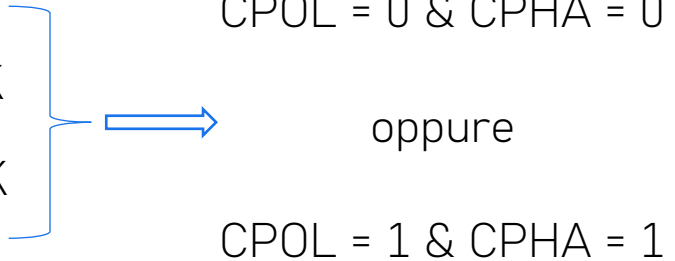
The diagram illustrates the internal architecture of the SPI module. It shows the connection between the external **Address and data bus** and the internal components:

- Address and data bus:** The primary interface for data and address signals.
- Shift register:** The core component for data transfer, operating in **LSB first** mode. It is connected to the bus via **MOSI** (Master Out Slave In) and **MISO** (Master In Slave Out) pins.
- Rx buffer (Receive buffer):** Receives data from the shift register and outputs it to the bus via a **Read** operation.
- Tx buffer (Transmit buffer):** Receives data from the bus via a **Write** operation and outputs it to the shift register.
- Communication control:** Manages the communication process, including baud rate generation and master/slave selection. It is controlled by the **SS** (Slave Select) pin and the **BR[2:0]** (Baud Rate Register bits 2, 1, and 0).
- Baud rate generator:** Generates the clock signal for the SPI module, outputting to the **SCK** (Serial Clock) pin.
- Master control logic:** Controls the master's operation, including the **SS** pin and the **BR[2:0]** register.
- SPI Registers:**
 - SPI_CR1 (Control Register 1):** Contains fields for **LSB FIRST**, **SPE** (SPI Enable), **BR2**, **BR1**, **BR0**, **MSTR** (Master/Slave Select), **CPOL** (Clock Polarity), **CPHA** (Clock Phase), **BIDI MODE**, **BIDI OE** (Bidirectional Output Enable), **CRCEN** (CRC Enable), **CRC Next**, **DFF** (Data Frame Format), **RX ONLY**, **SSM** (Slave Select Mode), and **SSI** (Slave Select Inversion).
 - SPI_CR2 (Control Register 2):** Contains fields for **TXE IE** (Transmit Event Interrupt Enable), **RXNE IE** (Receive Not Empty Interrupt Enable), **ERR IE** (Error Interrupt Enable), **SSOE** (Slave Select Output Enable), **TXDM AEN** (Transmit DMA Enable), and **RXDM AEN** (Receive DMA Enable).
 - SPI_SR (Status Register):** Contains fields for **BSY** (Busy Flag), **OVR** (Overrun Flag), **MOD F** (Mode Fault Flag), **CRC ERR** (CRC Error Flag), **TXE** (Transmit Event Flag), and **RXNE** (Receive Not Empty Flag).

Periferica SPI per i SoC della famiglia STM32F401

Sensore RFID MFRC522

Per scrivere o leggere sui registri del sensore MFRC522 tramite interfaccia SPI occorre seguire un protocollo, che prevede:

- Inviare i dati a partire dal MSB
 - I dati sono modulati sulla linea MISO durante il fronte di discesa del segnale SCK
 - I dati sono campionati sulla linea MOSI durante il fronte di salita del segnale SCK
 - La linea NSS va abbassata all'inizio di ogni trasmissione e rialzata alla fine
- 
- CPOL = 0 & CPHA = 0
oppure
CPOL = 1 & CPHA = 1

Sensore RFID MFRC522

8.1.2.1 SPI read data

Reading data using SPI requires the byte order shown in [Table 6](#) to be used. It is possible to read out up to n-data bytes.

The first byte sent defines both the mode and the address.

Table 6. MOSI and MISO byte order

Line	Byte 0	Byte 1	Byte 2	To	Byte n	Byte n + 1
MOSI	address 0	address 1	address 2	...	address n	00
MISO	X ^[1]	data 0	data 1	...	data n – 1	data n

[1] X = Do not care.

Remark: The MSB must be sent first.

Sensore RFID MFRC522

8.1.2.2 SPI write data

To write data to the MFRC522 using SPI requires the byte order shown in [Table 7](#). It is possible to write up to n data bytes by only sending one address byte.

The first send byte defines both the mode and the address byte.

Table 7. MOSI and MISO byte order

Line	Byte 0	Byte 1	Byte 2	To	Byte n	Byte n + 1
MOSI	address 0	data 0	data 1	...	data n – 1	data n
MISO	X ^[1]	X ^[1]	X ^[1]	...	X ^[1]	X ^[1]

[1] X = Do not care.

Remark: The MSB must be sent first.

Sensore RFID MFRC522

Il byte address inviato durante le operazioni di read/write deve essere rappresentato nel seguente formato:

Table 8. Address byte 0 register; address MOSI

7 (MSB)	6	5	4	3	2	1	0 (LSB)
1 = read 0 = write	address						0



Sensore RFID MFRC522

Il sensore contiene un buffer FIFO di input/output da 64 byte, che contiene i dati da scrivere verso i tag oppure quelli letti dai tag.

Inoltre il sensore è composto da 64 registri da 8 bit, indirizzati con 6 bit e suddivisi in 4 categorie:

1. Registri di controllo
2. Registri di stato
3. Un registro dato, che serve a leggere/scrivere un byte alla volta da/verso il buffer FIFO
4. Registri riservati ad un utilizzo futuro

Sensore RFID MFRC522

I registri principali sono:

Indirizzo (hex)	Nome	Descrizione
01h	CommandReg	Salva i comandi da impartire al sensore
02h	ComlEnReg	Abilita/disabilita i bit per le interrupt request
04h	ComIrqReg	Bit per le interrupt request
06h	ErrorReg	Bit di errore relativi all'ultimo comando eseguito
09h	FIFODataReg	Input/output verso il buffer FIFO
0Ah	FIFOLevelReg	Numero di byte memorizzati nel buffer FIFO
0Dh	BitFramingReg	Impostazioni per i frame bit-oriented

Sensore RFID MFRC522

I registri principali sono:

Indirizzo (hex)	Nome	Descrizione
11h	ModeReg	Definisce le modalità di trasmissione e ricezione
14h	TxControlReg	Controlla il funzionamento dei pin driver dell'antenna TX1 e TX2
2Ah	TModeReg	Impostazioni per il timer interno al sensore
2Bh	TPrescalerReg	Imposta il valore del prescaler del timer
2Ch	TReloadReg_Hi	Gli 8 bit alti del valore di ricarica del timer
2Dh	TReloadReg_Lo	Gli 8 bit bassi del valore di ricarica del timer
2Eh	TCounterValReg_Hi	Contiene gli 8 bit alti del valore attuale del timer
2Fh	TCounterValReg_Lo	Contiene gli 8 bit bassi del valore attuale del timer

Sensore RFID MFRC522

I comandi impartibili al sensore sono:

Table 149. Command overview

Command	Command code	Action
Idle	0000	no action, cancels current command execution
Mem	0001	stores 25 bytes into the internal buffer
Generate RandomID	0010	generates a 10-byte random ID number
CalcCRC	0011	activates the CRC coprocessor or performs a self test
Transmit	0100	transmits data from the FIFO buffer
NoCmdChange	0111	no command change, can be used to modify the CommandReg register bits without affecting the command, for example, the PowerDown bit
Receive	1000	activates the receiver circuits
Transceive	1100	transmits data from FIFO buffer to antenna and automatically activates the receiver after transmission
-	1101	reserved for future use
MFAuthent	1110	performs the MIFARE standard authentication as a reader
SoftReset	1111	resets the MFRC522

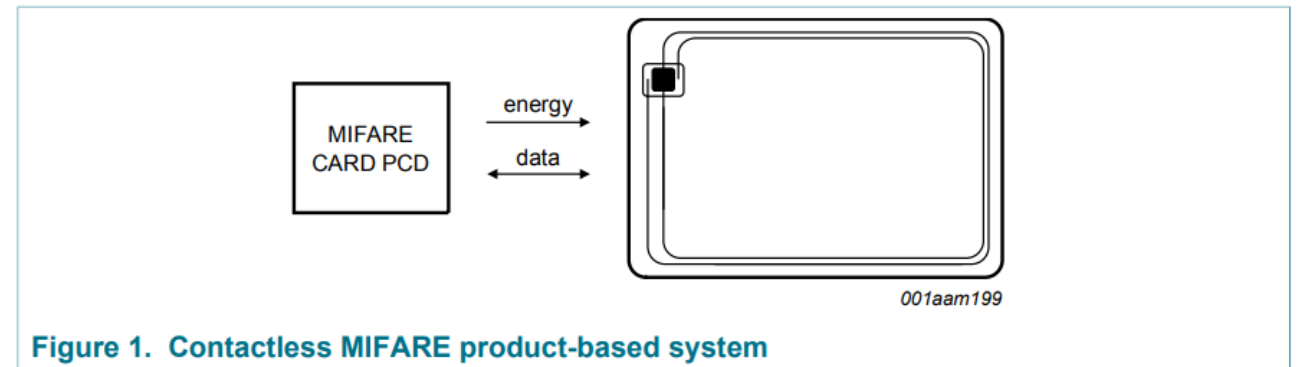


Smart Card MF1S50

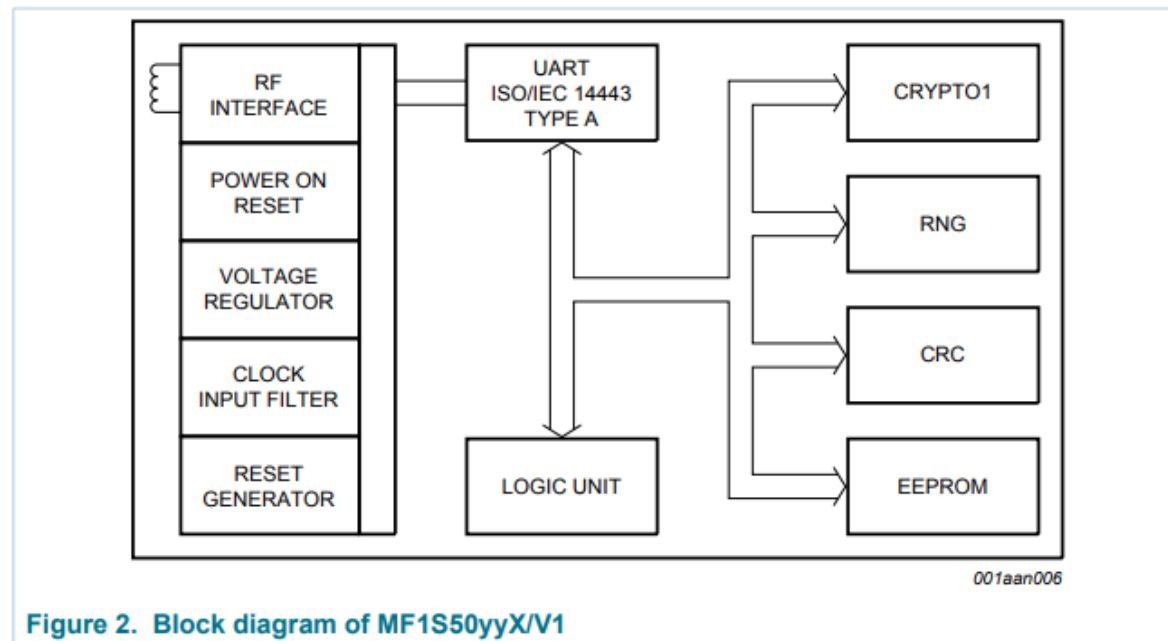
La MF1S50 è una smart card della Mifare, composta da un chip collegato ad un'antenna, realizzata tramite una bobina con un piccolo numero di spire, il tutto immerso in un wafer plastico.

Caratteristiche:

- Trasmissione wireless di dati ed energia
- Distanza operative fino ad un massimo di 100mm
- Frequenza operativa di 13.56 MHz
- Trasferimento dati a 106 kbit/s
- Integrità dei dati tramite 16-bit CRC, parity, bit coding, bit counting
- Anticollisione
- 4 Byte NUID



Smart Card MF1S50



EEPROM: 1 kB organizzato in 16 settori da 4 blocchi. Ogni blocco contiene 16 byte. L'ultimo blocco di ogni settore è chiamato «trailer», e contiene due chiavi segrete e condizioni di accesso programmabili per ogni blocco del settore.

Smart Card MF1S50

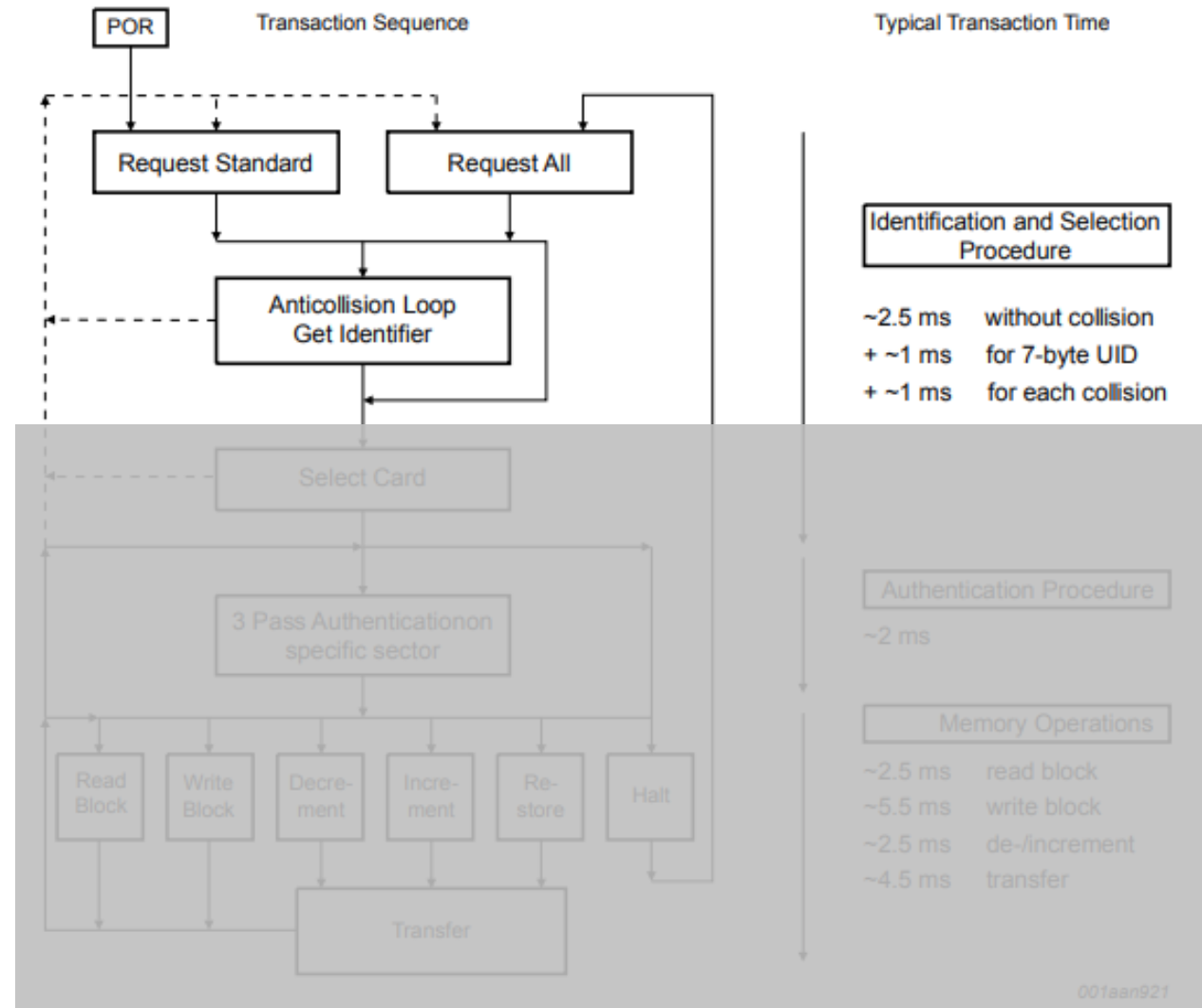
Request standard / all

Dopo un Power-On Reset (POR) la card risponde ad un comando di request REQA o di wakeup WUPA con un codice ATQA (Answer To reQest code), che identifica il tipo di card

Anticollision loop

Nel ciclo di anticollisione viene letto il codice identificativo della card. Se ci sono più card nel campo operativo del lettore, esse possono essere distinte in base al loro ID, e una di esse può essere selezionata per le prossime transazioni. Le card non selezionate tornano in uno stato inattivo, in attesa di un prossimo comando di request.

MIFARE Classic command flow diagram



Smart Card MF1S50

Tutti i comandi eseguiti dalla card sono impartiti dal reader e controllati dalla Digital Control Unit della MF1S50.

Table 9. Command overview

Command	ISO/IEC 14443	Command code (hexadecimal)
Request	REQA	26h (7 bit)
Wake-up	WUPA	52h (7 bit)
Anticollision CL1	Anticollision CL1	93h 20h
Select CL1	Select CL1	93h 70h
Anticollision CL2	Anticollision CL2	95h 20h
Select CL2	Select CL2	95h 70h
Halt	Halt	50h 00h
Authentication with Key A	-	60h
Authentication with Key B	-	61h
Personalize UID Usage	-	40h
SET_MOD_TYPE	-	43h
MIFARE Read	-	30h

© 2006 The Authors
Journal compilation © 2006 Blackwell Publishing Ltd

Vediamo un po' di codice:

```

179 MX_SPI2_Init();
180 MX_USART1_UART_Init();
181 MX_TIM2_Init();
182 /* USER CODE BEGIN 2 */
183 MFRC522_Init(&hspi2, &htim2);
184 /* USER CODE END 2 */
185
186 /* Infinite loop */
187 /* USER CODE BEGIN WHILE */
188 while (1)
189 {
190     /* USER CODE END WHILE */
191
192     /* USER CODE BEGIN 3 */
193
194
195     if(!disable_rfid_sensor){
196         if(!MFRC522_Check(str)){
197             uint8_t flag = FLAG_CARD;
198
199             send_byte_to_output_board(&flag, 1);
200
201             for(int i = 0; i < 4 ; i++) {
202                 send_byte_to_output_board(&str[i], 1);
203                 print_byte_to_hex(str[i]);
204             }
205             print_lf();
206
207             enter_wait_state();
208         }
209     }
210
211
212

```

Sensore RFID MFRC522

```
51
52 void MFRC522_Init(SPI_HandleTypeDef* handle_spi, TIM_HandleTypeDef* handle_timer) {
53     hspi = handle_spi;
54     htim = handle_timer;
55
56
57     MFRC522_Reset();
58     MFRC522_WriteRegister(MFRC522_REG_T_MODE, 0x8D);
59     MFRC522_WriteRegister(MFRC522_REG_T_PRESCALER, 0x3E);
60     MFRC522_WriteRegister(MFRC522_REG_T_RELOAD_L, 30);
61     MFRC522_WriteRegister(MFRC522_REG_T_RELOAD_H, 0);
62     MFRC522_WriteRegister(MFRC522_REG_TX_AUTO, 0x40);
63     MFRC522_AntennaOn(); // Open the antenna
64 }

//
65 uint8_t MFRC522_Check(uint8_t * id) {
66     uint8_t status;
67     status = MFRC522_Request(PICC_REQIDL, id); // Find cards, return card type
68     if (status == MI_OK) status = MFRC522_Anticoll(id); // Card detected. Anti-collision, return card
69     return status;
70 }
71
72
73
74
```

Sensore RFID MFRC522

```
119
120 uint8_t MFRC522_Request(uint8_t reqMode, uint8_t * TagType) {
121     uint8_t status;
122     uint16_t backBits;
123
124     MFRC522_WriteRegister(MFRC522_REG_BIT_FRAMING, 0x07);
125     TagType[0] = reqMode;
126     status = MFRC522_ToCard(PCD_TRANSCEIVE, TagType, 1, TagType, &backBits);
127     if ((status != MI_OK) || (backBits != 0x10)) status = MI_ERR;
128
129     return status;
130 }
```

```
199 uint8_t MFRC522_Anticoll(uint8_t * serNum) {
200     uint8_t status;
201     uint8_t i;
202     uint8_t serNumCheck = 0;
203     uint16_t unLen;
204
205     MFRC522_WriteRegister(MFRC522_REG_BIT_FRAMING, 0x00);
206     serNum[0] = PICC_ANTICOLL;
207     serNum[1] = 0x20;
208     status = MFRC522_ToCard(PCD_TRANSCEIVE, serNum, 2, serNum, &unLen);
209     if (status == MI_OK) {
210         // Check card serial number
211         for (i = 0; i < 4; i++) serNumCheck ^= serNum[i];
212         if (serNumCheck != serNum[i]) status = MI_ERR;
213     }
214     return status;
215 }
```

Passive Buzzer



Il Buzzer è uno dispositivo elettronico che permette di emettere un suono se correttamente alimentato.

I buzzer si suddividono in:

- Buzzer Attivi: alimentati con una tensione opportuna riproducono un tono ad una frequenza pre-impostata.
- Buzzer Passivi: non emettono alcun suono pre-impostato, ma necessitano di una forma d'onda specifica per fare vibrare la membrana interna. Possono produrre toni differenti in funzione del segnale di alimentazione utilizzato.

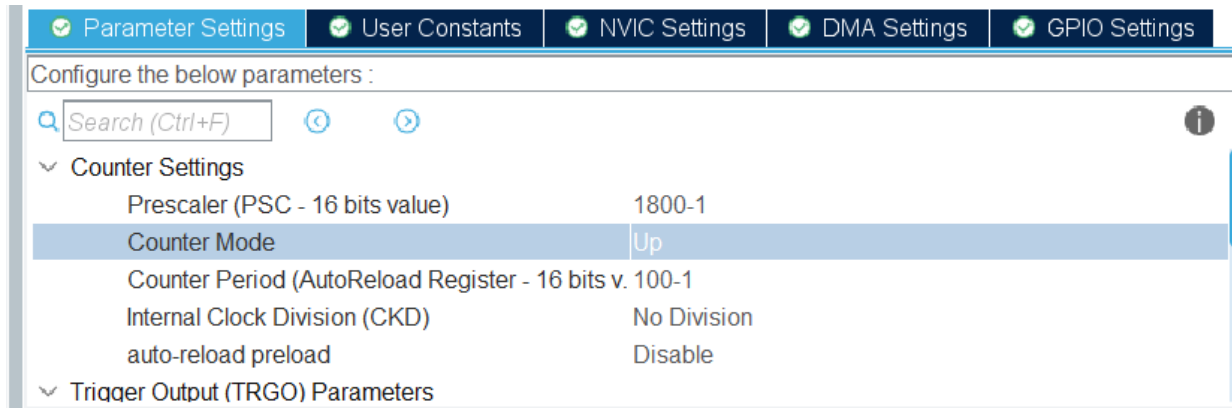
Per questo progetto utilizzeremo un buzzer passivo, per produrre dei suoni diversi a seconda di se il pin o la card siano corretti o meno.

Passive Buzzer

La scheda STM32F3 fungerà da sorgente per il buzzer, generando un segnale PWM. Tale segnale verrà trasmesso in output dal GPIO PC6 (channel 1 del timer 3) al PIN+ del buzzer.

Dunque per poter generare un segnale PWM abilitiamo dapprima un timer generico (TIM3), selezionando come sorgente del clock il clock interno. Anche il TIM3 è collegato al clock APB1 che ha sempre frequenza di clock pari a 72 MHz.

Inoltre, abilitiamo il channel 1 per la generazione del segnale PWM e settiamo i parametri del timer nel seguente modo:



In questo modo il segnale PWM generato dal TIM3, avrà una frequenza pari a 400 Hz.



Passive Buzzer

I parametri della slide precedente sono stati settati osservando tali formule:

$$TIM\ CLOCK = \frac{APB\ TIM\ CLOCK}{PRESCALER}$$

$$FREQUENCY = \frac{TIM\ CLOCK}{ARR}$$

$$DUTY\ (\%) = \frac{CCRx}{ARR} \times 100$$

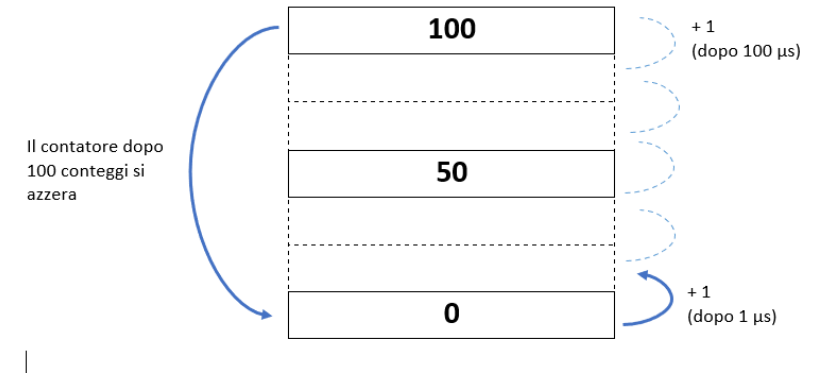
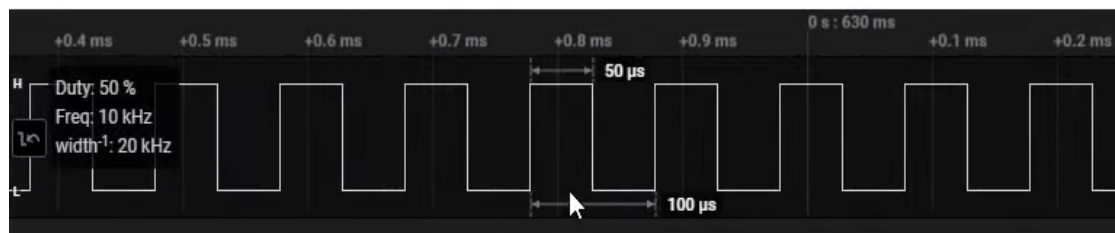
Il duty cycle è il rapporto che c'è tra il periodo del segnale quando è alto e il periodo totale del segnale ed è espresso in percentuale. Un duty cycle del 100% equivale ad un segnale alto continuo, un duty cycle del 0% equivale ad un segnale basso continuo mentre un duty cycle del 50% significa che il segnale alto dura quanto quello basso. Vediamo un esempio esplicativo nella slide successiva.

Passive Buzzer

$$TIM\ CLOCK = \frac{APB\ TIM\ CLOCK}{PRESCALER} = \frac{72\ MHz}{72} = 1\ MHz$$

$$FREQUENCY = \frac{TIM\ CLOCK}{ARR} = \frac{1\ MHz}{100} = 10\ KHz$$

$$DUTY\ (\%) = \frac{CCRx}{ARR} \times 100 = \frac{50}{100} \times 100 = 50\%$$



Il registro di conteggio del timer si incrementerà ogni 1 μs (1 / 1MHz) e una volta raggiunto 100 (valore del registro ARR), ovvero dopo 100 μs, il contatore si azzerà e riparte da 0 (vedi figura in alto).

Il registro CCR fa in modo che finché il conteggio non supera il valore 50, il valore logico del segnale è ALTO altrimenti il valore logico del segnale è BASSO. Dunque in questo esempio dopo ogni 50 μs il valore logico del segnale passa da alto a basso (come mostrato in figura).

Passive Buzzer

```
107 void buzzer_on(){
108
109     htim3.Instance->CCR1 = 60; // ON
110     delay(4000); // wait for 400 ms
111     htim3.Instance->CCR1 = 0; // OFF
112 }
113
114 void buzzer_err(){
115
116     htim3.Instance->CCR1 = 60; // ON
117     delay(1000); //wait for 100 ms
118     htim3.Instance->CCR1 = 0; // OFF
119     delay(1000); // wait for 100 ms
120     htim3.Instance->CCR1 = 60; // ON
121     delay(1000); // wait for 100 ms
122     htim3.Instance->CCR1 = 0;
123     delay(1000); // wait for 100 ms
124     htim3.Instance->CCR1 = 60;
125     delay(1000); // wait for 100 ms
126     htim3.Instance->CCR1 = 0;
127 }
```



Se il pin o la card inseriti sono validi allora il buzzer emetterà un breve suono continuo per circa 400 ms, accompagnato anche dall'accensione del led verde.



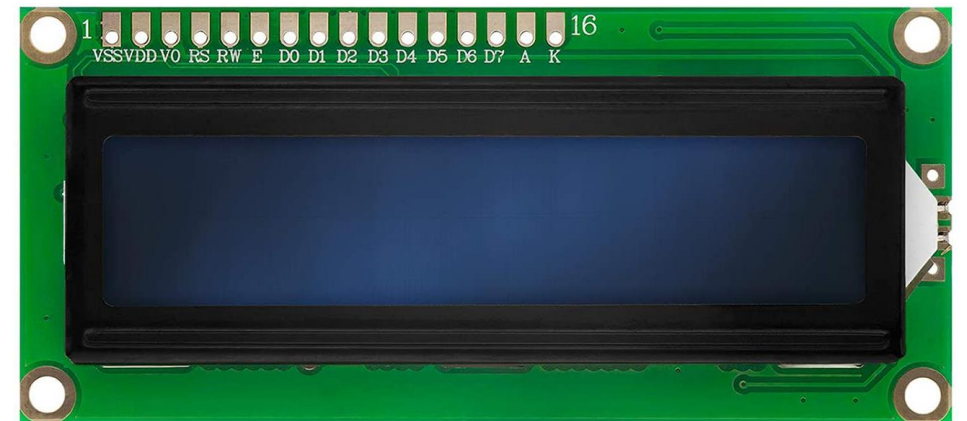
Se il pin o la card inseriti non sono validi allora il buzzer emetterà dei suoni ad intermittenza ogni 100 ms per tre volte, accompagnato dall'accensione del led rosso.

Display LCD HD44780U

Il display LCD è un dot-matrix liquid crystal display con una risoluzione di 16 caratteri per 2 linee, controllato dal chip di controllo HD44780U della Hitachi.

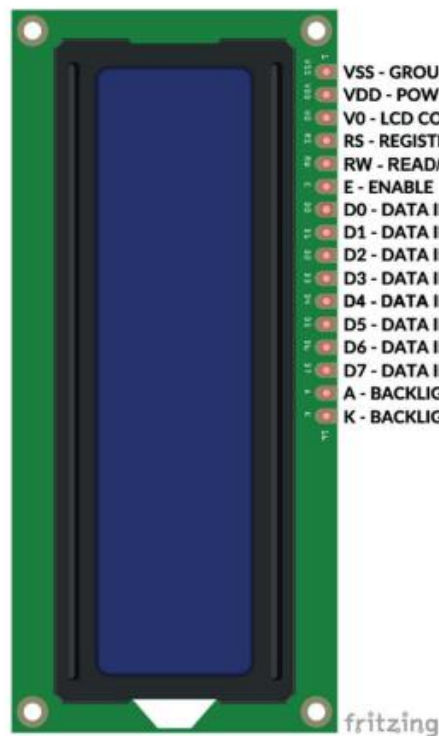
Caratteristiche:

- View area: 12 mm x 56 mm
- Resolution: 16 Characters x 2 Lines
- Backlight: ED, White
- View Angle: 180 degrees
- Operating Voltage: 5V



Display LCD HD44780U

Pinout:



16 pin

4 pin

I2C SERIAL CLOCK LINE - SCL
I2C SERIAL DATA LINE - SDA
POWER SUPPLY - VCC
GROUND - GND

JUMPER PADS (A0 - A2)



Potenziometro



I²C



Il protocollo I²C è stato creato dalla Philips Semiconductors nel 1982; la sigla sta per Inter-Integrated Circuit. Il protocollo permette la comunicazione di dati tra due o più dispositivi I²C utilizzando un bus a due fili. Il bus I²C è un bus seriale che necessita di sole due linee nominate SDA (Serial Data) e SCL (Serial Clock) più la linea di massa (GND), comune a tutti i dispositivi. La prima è utilizzata per il transito dei dati che sono in formato ad 8 bit, mentre la seconda è utilizzata per trasmettere il segnale di clock necessario per la sincronizzazione della trasmissione. Il bus I²C permette la connessione di più periferiche su uno stesso bus ma permette la comunicazione tra due soli dispositivi alla volta.

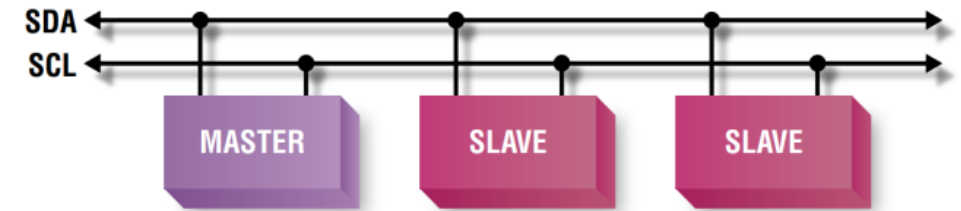
I²C

In generale ci sono 4 distinti modi di operare:

- un master trasmette – controlla il clock e invia dati agli slave
- un master riceve – controlla il clock e riceve dati dallo slave
- uno slave trasmette – il dispositivo non controlla il clock e invia dati al master
- uno slave riceve – il dispositivo non controlla il clock e riceve dati dal master.

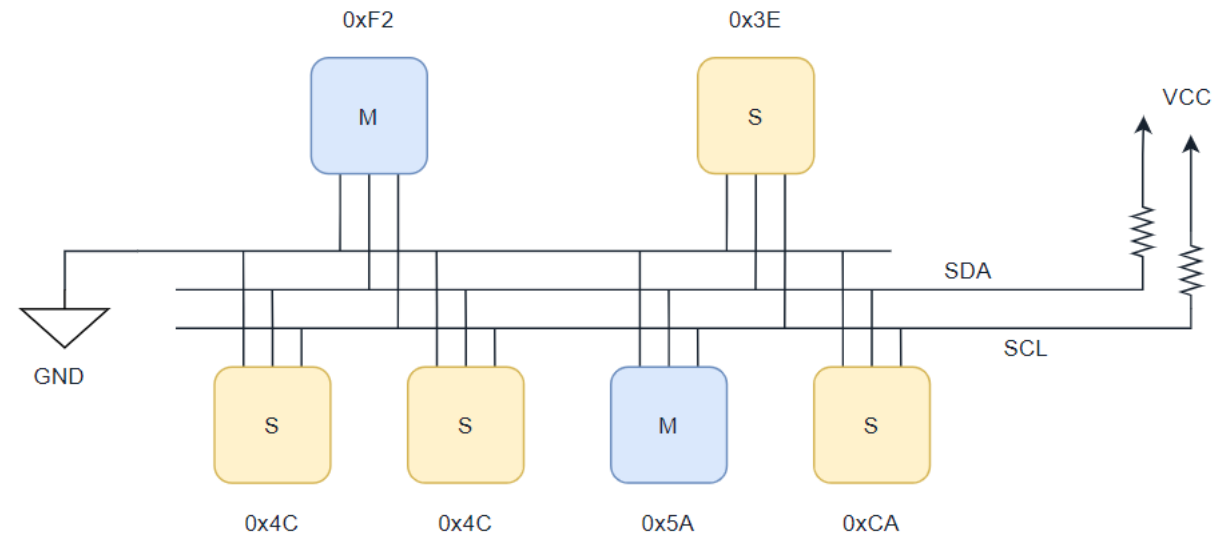
Il dispositivo master è semplicemente il dispositivo che controlla il bus in un certo istante; tale dispositivo controlla il segnale di Clock e genera i segnali di START e di STOP. I dispositivi slave semplicemente “ascoltano” il bus ricevendo dati dal master o inviandone qualora questo ne faccia loro richiesta.

Su uno stesso bus è inoltre possibile la presenza di più master, ma solo uno alla volta ricoprirà questo ruolo (modalità multi-master).



I²C

Ogni device connesso sul bus è identificato da un indirizzo a 7 bit. L'indirizzo è assegnato ai device in fase di produzione e quasi sempre non è modificabile, ma nonostante ciò possono coesistere device aventi indirizzo identico.





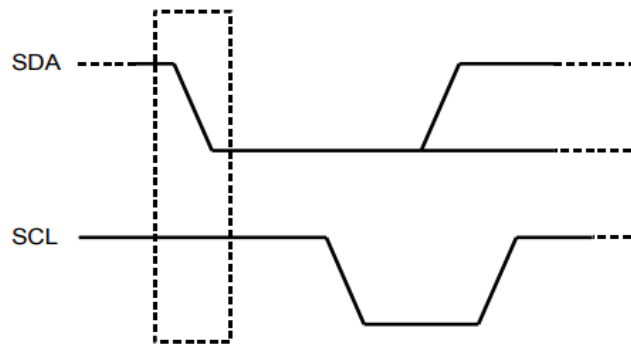
Comunicazioni sul bus I²C

Vediamo ora come avviene una comunicazione su di un bus I²C. Come detto solo le periferiche master possono avviare una comunicazione. Le fasi che devono essere seguite sono riportate di lato.

1. Il master controlla se le linee SDA e SCL non siano attive, ovvero siano poste ambedue a livello alto.
2. Se il bus è libero invia il bit di start che fa capire alle altre periferiche che il bus è ora occupato. Le altre periferiche si metteranno allora in ascolto per comprendere con chi il master ha intenzione di comunicare.
3. Il master provvede all'invio del segnale di sincronizzazione sulla linea SCL, che sarà rappresentato da un onda quadra, non necessariamente periodica.
4. Il master invia l'indirizzo della periferica con la quale vuole parlare.
5. Il master segnala poi se la comunicazione che vuole intraprendere verso la periferica è di lettura o scrittura.
6. Il master attende la risposta da parte della periferica che nella chiamata ha riconosciuto il suo indirizzo. Se nessuna periferica risponde il master libera il bus.
7. Dopo l'avvenuto riconoscimento della periferica, il master inizia lo scambio dei dati. Lo scambio avviene inviando pacchetti di 8 bit. Ad ogni pacchetto si deve attendere il segnale che avvisa dell'avvenuta ricezione (ACK).
8. Quando la trasmissione è terminata il master libera il bus inviando il bit di stop.

Fase 1 e 2

L'hardware del microcontrollore, dedicato alla gestione dell'interfaccia I²C, controlla le linee SDA e SCL per un tempo superiore al massimo periodo di trasmissione consentito dall'hardware. Se il bus risulta libero, il master invia la sequenza di Start, che consiste nel portare la linea SDA a livello basso quando SCL è a livello alto. Dopo l'invio della sequenza di Start, il bus è considerato occupato.

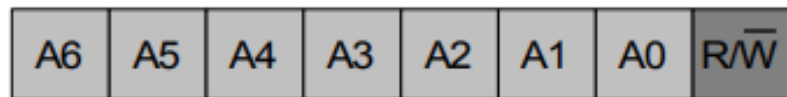


Fase 3

Dopo la transizione della linea SDA da alto a basso, il master invia il segnale di sincronizzazione per le altre periferiche. A differenza della sequenza di Start e di Stop la linea SDA assume un valore valido solo se la linea SCL è a livello basso. Questo vuol dire che non sono ammesse transizioni di livello della linea SDA durante il livello alto della linea SCL, se non da parte del master per inviare un nuovo Start o uno Stop.

Fase 4 e 5

I 7 bit dell'indirizzo vengono inviati dal bit più significativo al bit meno significativo. In coda a questo indirizzo viene aggiunto un bit per segnalare se il master vuole intraprendere, con la periferica individuata da tale indirizzo, una comunicazione di scrittura o di lettura. In particolare se tale bit è 0 vuol dire che il master vuole scrivere sulla periferica, se il bit è 1 vuol dire che il master vuole leggere della periferica. La figura in basso mostra il formato del byte.



Fase 6

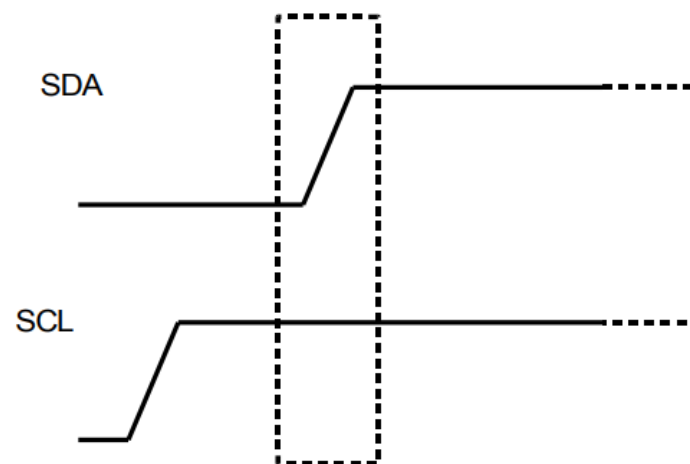
L'invio dell'indirizzo a 7 bit e della modalità del colloquio (lettura/scrittura), avviene grazie ad otto transizioni, da livello alto basso, della linea SCL. Al nono impulso della linea SCL il master si aspetta una risposta di un bit da parte della periferica che ha chiamato. La risposta della periferica chiamata consiste nel mantenere a livello basso la linea SDA, per la durata di un ciclo SCL. Ovvero il master attende l'Acknowledge da parte della periferica chiamata. Una sola periferica risponderà alla chiamata del master. Qualora la periferica non sia presente il master libera il bus permettendo ad eventuali altri master di prenderne il controllo.

Fase 7

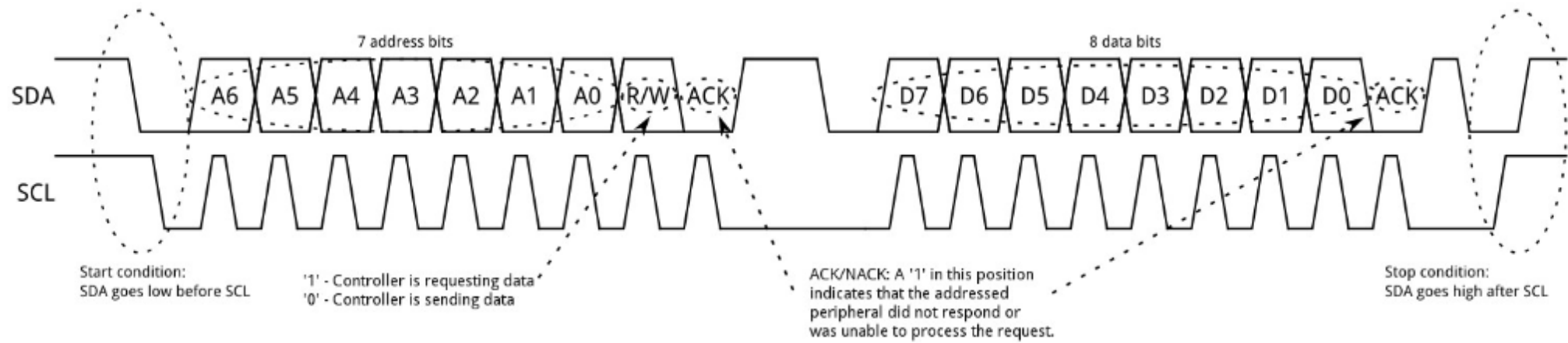
Dopo l'avvenuto riconoscimento, avviene lo scambio dei dati verso la periferica, nel caso di scrittura, o dalla periferica al master, in caso di lettura. In una comunicazione si possono avere sia fasi di scrittura che di lettura. Ad ogni invio di un byte sarà necessario l'Acknowledge del byte inviato o ricevuto. In particolare se il master invia un byte allo slave si aspetta, dopo l'ottavo bit, un bit basso sulla linea SDA. Se lo slave sta inviando un byte al master si aspetta che quest'ultimo invii un bit alto dopo aver ricevuto il byte. La mancanza dell'Acknowledge determina un errore di comunicazione.

Fase 8

Quando la comunicazione è terminata, il master libera il bus, inviando la sequenza di Stop. Questa consiste nella transizione dal livello basso ad alto della linea SDA, quando la linea SCL è alta. Il vecchio slave comprenderà che la comunicazione con lui è terminata.

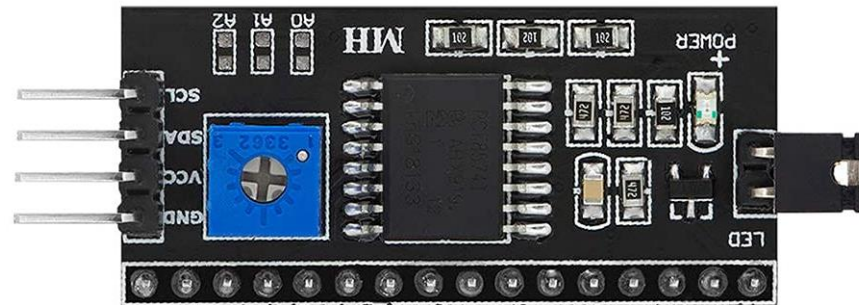


I²C



Display LCD HD44780U

La driver board I²C monta un chip PCF8574 in grado di implementare il protocollo I²C e di trasferire i dati scambiati tramite tale protocollo su di un porto parallelo collegato al display. Tale board, inoltre, presenta un potenziometro per regolare il contrasto dello schermo, e si occupa anche dare alimentazione al display.



Display LCD HD44780U

Il chip PCF8574 consente di implementare un dispositivo slave I2C, con un indirizzo selezionabile tra 8 disponibili, variando i livelli logici dei pin di input del chip A2, A1 e A0.

Table 4. PCF8574 address map

Pin connectivity			Address of PCF8574								Address byte value		7-bit hexadecimal address without R/W
A2	A1	A0	A6	A5	A4	A3	A2	A1	A0	R/W	Write	Read	
V _{SS}	V _{SS}	V _{SS}	0	1	0	0	0	0	0	-	40h	41h	20h
V _{SS}	V _{SS}	V _{DD}	0	1	0	0	0	0	1	-	42h	43h	21h
V _{SS}	V _{DD}	V _{SS}	0	1	0	0	0	1	0	-	44h	45h	22h
V _{SS}	V _{DD}	V _{DD}	0	1	0	0	0	1	1	-	46h	47h	23h
V _{DD}	V _{SS}	V _{SS}	0	1	0	0	1	0	0	-	48h	49h	24h
V _{DD}	V _{SS}	V _{DD}	0	1	0	0	1	0	1	-	4Ah	4Bh	25h
V _{DD}	V _{DD}	V _{SS}	0	1	0	0	1	1	0	-	4Ch	4Dh	26h
V _{DD}	V _{DD}	V _{DD}	0	1	0	0	1	1	1	-	4Eh	4Fh	27h



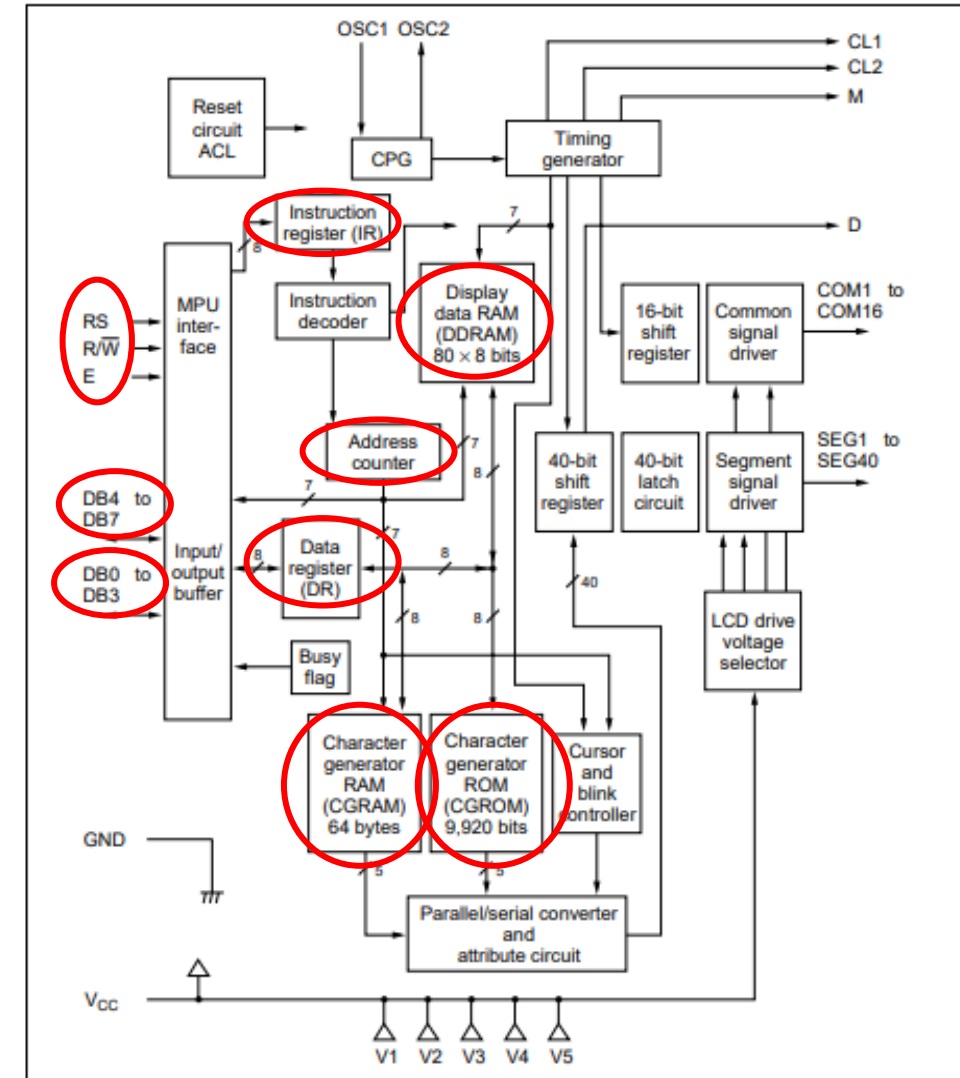
HD44780U

Il display è controllato dal chip HD44780U della Hitachi.

Caratteristiche:

- 5×8 and 5×10 dot matrix possible
- Low power operation support: 2.7 to 5.5V
- 4-bit or 8-bit MPU (MicroProcessor Unit) interface enabled
- 80×8 -bit display RAM (80 characters max.)
- 9,920-bit character generator ROM for a total of 240 character fonts
- 64×8 -bit character generator RAM
- Wide range of instruction functions
- Automatic reset circuit that initializes the controller after power on

HD44780U Block Diagram



HD44780U

Instruction Register (IR, 8 bit): contiene i codici delle istruzioni, e può solo essere scritto dall' MPU.

Data Register (DR, 8 bit): contiene temporaneamente i dati da scrivere nella DDRAM (o CGRAM), oppure i dati da leggere dalla DDRAM (o CGRAM).

Address Counter (AC): contiene l'indirizzo della locazione di memoria della DDRAM su cui verrà eseguita la prossima operazione di lettura/scrittura. In pratica indica la posizione del **cursore** all'interno della DDRAM. Dopo aver scritto (letto) nella DDRAM, il valore di AC viene automaticamente incrementato (decrementato) di 1.

Table 1 Register Selection

RS	R/ \overline{W}	Operation
0	0	IR write as an internal operation (display clear, etc.)
0	1	Read busy flag (DB7) and address counter (DB0 to DB6)
1	0	DR write as an internal operation (DR to DDRAM or CGRAM)
1	1	DR read as an internal operation (DDRAM or CGRAM to DR)

HD44780U

Data Display RAM (DDRAM): contiene i caratteri da visualizzare sul display codificati su 8 bit. La sua capacità è di 80 x 8bit, cioè 80 caratteri. Non tutti gli 80 caratteri sono visualizzati sul display nello stesso momento, ovviamente, ma solo una parte, che dipende dalla modalità di funzionamento selezionata:

- **1-line display (N = 0):** la DDRAM è vista come un vettore lineare di 80 caratteri di cui solo i primi 8 sono visualizzati a schermo. È possibile shiftare a sinistra o a destra i valori nella DDRAM per cambiare i caratteri visualizzati.

Display position (digit)	1	2	3	4	5	...	79	80
DDRAM address (hexadecimal)	00	01	02	03	04	4E	4F

Display position	1	2	3	4	5	6	7	8
DDRAM address	00	01	02	03	04	05	06	07

For shift left	01	02	03	04	05	06	07	08
-------------------	----	----	----	----	----	----	----	----

For shift right	4F	00	01	02	03	04	05	06
--------------------	----	----	----	----	----	----	----	----

HD44780U

- **2-line display (N = 1):** la DDRAM viene suddivisa logicamente in 2 parti uguali: le prime 40 locazioni di memoria contengono i caratteri della prima linea, mentre le successive 40 contengono i caratteri della seconda linea. Ovviamente non tutti i caratteri sono visualizzati, ma solo i primi 8 della prima linea (00h-07h) e i primi 8 della seconda linea (28h-2Fh).

Display position	1	2	3	4	5	39	40
DDRAM address (hexadecimal)	00	01	02	03	04	26	27
	28	29	30	31	32	4E	4F

Display position	1	2	3	4	5	6	7	8
DDRAM address	00	01	02	03	04	05	06	07
	28	29	30	31	32	33	34	35

For shift left	01	02	03	04	05	06	07	08
	29	30	31	32	33	34	35	36


For shift right	27	00	01	02	03	04	05	06
	4F	28	29	30	31	32	33	34

HD44780U

Codici delle istruzioni
eseguibili dal controller, che
l'MPU scrive nell'IR.

Table 6 Instructions

Instruction	Code										Description	Execution Time (max) (when f_{cp} or f_{osc} is 270 kHz)
	RS	R/W	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0		
Clear display	0	0	0	0	0	0	0	0	0	1	Clears entire display and sets DDRAM address 0 in address counter.	
Return home	0	0	0	0	0	0	0	0	1	—	Sets DDRAM address 0 in address counter. Also returns display from being shifted to original position. DDRAM contents remain unchanged.	1.52 ms
Entry mode set	0	0	0	0	0	0	0	1	I/D	S	Sets cursor move direction and specifies display shift. These operations are performed during data write and read.	37 μ s
Display on/off control	0	0	0	0	0	0	1	D	C	B	Sets entire display (D) on/off, cursor on/off (C), and blinking of cursor position character (B).	37 μ s
Cursor or display shift	0	0	0	0	0	1	S/C	R/L	—	—	Moves cursor and shifts display without changing DDRAM contents.	37 μ s
Function set	0	0	0	0	1	DL	N	F	—	—	Sets interface data length (DL), number of display lines (N), and character font (F).	37 μ s
Set CGRAM address	0	0	0	1	ACG	ACG	ACG	ACG	ACG	ACG	Sets CGRAM address. CGRAM data is sent and received after this setting.	37 μ s
Set DDRAM address	0	0	1	ADD	ADD	ADD	ADD	ADD	ADD	ADD	Sets DDRAM address. DDRAM data is sent and received after this setting.	37 μ s
Read busy flag & address	0	1	BF	AC	AC	AC	AC	AC	AC	AC	Reads busy flag (BF) indicating internal operation is being performed and reads address counter contents.	0 μ s



HD44780U

Il controller HD44780U è in grado di interfacciarsi con l'MPU in 2 modalità distinte:

- **4-bit interface data:** solo le 4 linee dati DB4-DB7 sono attive, mentre le linee DB0-DB3 sono disabilitate. Lo scambio dati avviene sempre su parole di 8 bit, dunque uno scambio dati tra HD44780U e MPU è completo solo dopo 2 trasferimenti consecutivi sulle linee DB4-DB7. Sono inviati prima i 4 bit più significativi, e poi i 4 meno significativi.
- **8-bit interface data:** tutte e 8 le linee dati (DB0-DB7) sono abilitate ed utilizzate.

Il driver I²C PCF8574 comunica con il controller HD44780U tramite interfaccia a 4-bit, dunque lo scambio dati tra la board STM32 e il display controller richiede l'invio di 2 frame I²C.

HD44780U

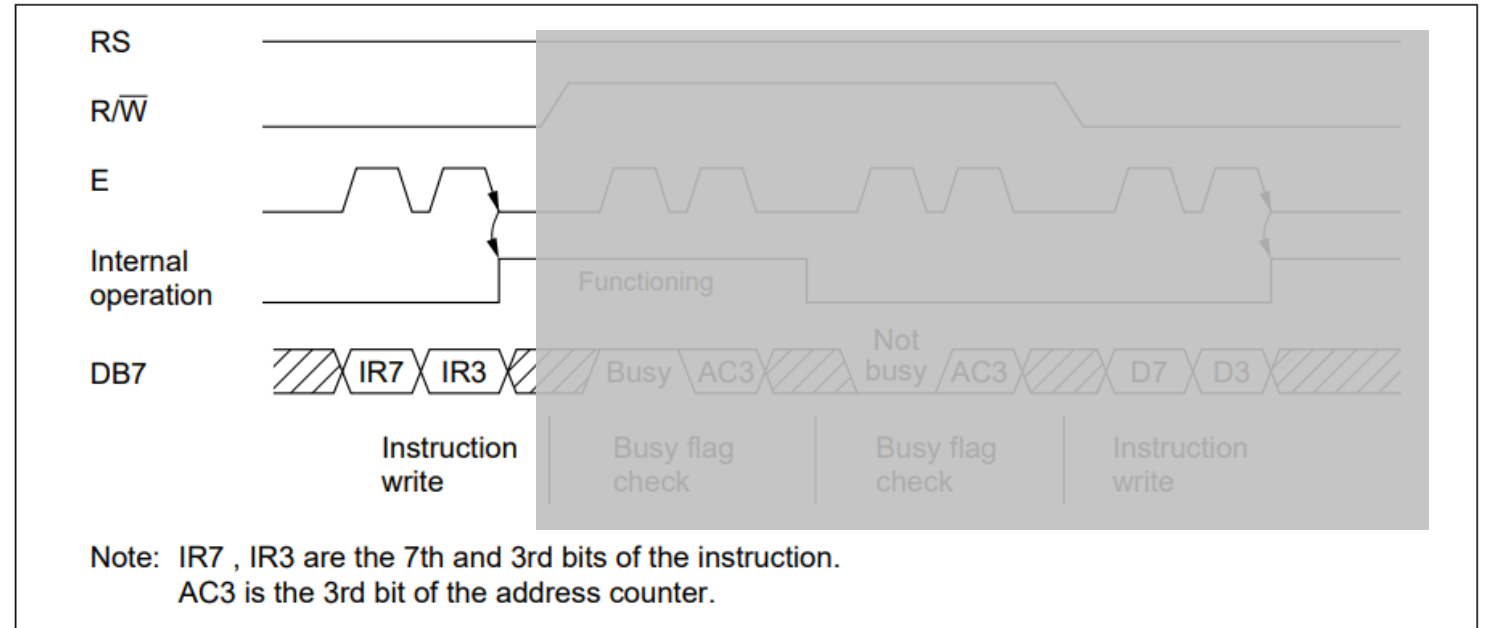
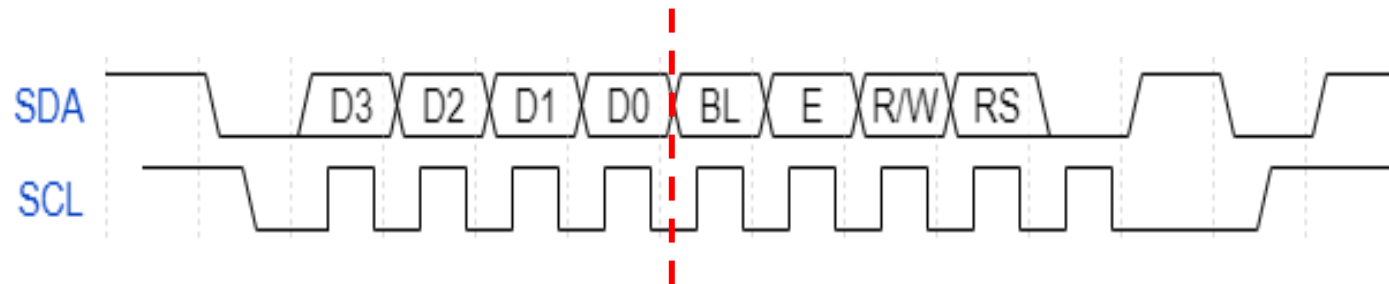


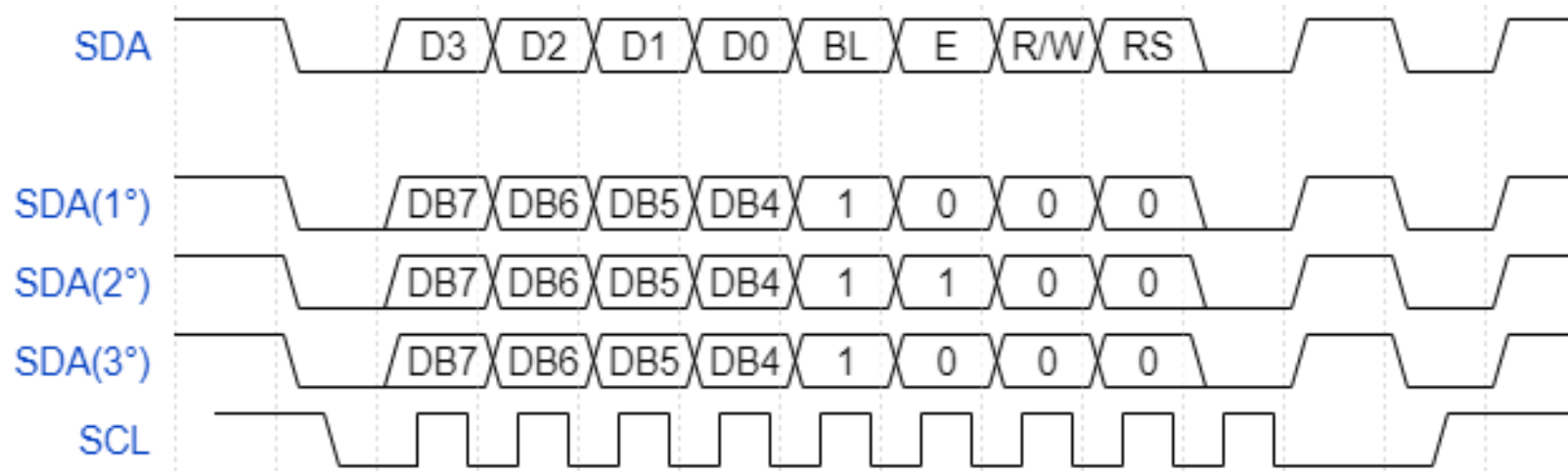
Figure 17 Example of 4-Bit Data Transfer Timing Sequence



BL(BackLight) è il bit per attivare/disattivare la retroilluminazione del display

HD44780U

Supponiamo di voler scrivere un'istruzione generica nell'Instruction Register:



Ripetiamo poi gli stessi passaggi per scrivere i 4 bit meno significativi DB3-DB0.

HD44780U

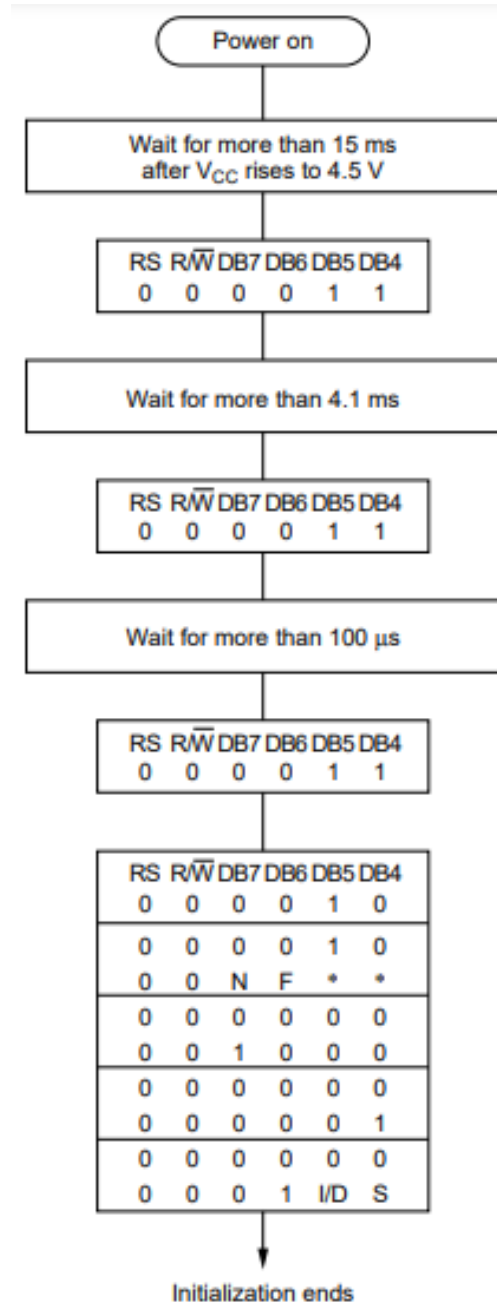
Osserviamo il codice utilizzato per inviare dati verso il controller HD44780U del display:

```
30 void lcd_send(uint8_t d, uint8_t mask){
31     uint8_t d_h = d & 0xf0;
32     uint8_t d_l = (d << 4) & 0xf0;
33
34     lcd_send4(d_h,mask);
35     lcd_send4(d_l,mask);
36
37 }
38
39 void lcd_send4(uint8_t d, uint8_t mask){
40     uint8_t data[2];
41     uint8_t mask_en = mask | En;
42
43     data[0] = d | mask;
44     data[1] = d | mask_en;
45
46     HAL_I2C_Master_Transmit(hi2c, LCD_I2C_SLAVE_ADDR, data, 2, HAL_MAX_DELAY);
47     HAL_I2C_Master_Transmit(hi2c, LCD_I2C_SLAVE_ADDR, data, 1, HAL_MAX_DELAY);
48 }
```

HD44780U

lcd_i2c.c

```
74
75 void lcd_init(I2C_HandleTypeDef* handle){
76     hi2c = handle;
77
78     HAL_Delay(100);
79
80     lcd_send4(0x30,0x08);
81     HAL_Delay(5);
82     lcd_send4(0x30,0x08);
83     HAL_Delay(1);
84     lcd_send4(0x30,0x08);
85     HAL_Delay(10);
86     lcd_send4(0x20,0x08);
87     HAL_Delay(10);
88
89     lcd_send_cmd(LCD_FUNCTIONSET | LCD_4BITMODE | LCD_2LINE | LCD_5x8DOTS);
90     HAL_Delay(2);
91     lcd_send_cmd(LCD_DISPLAYCONTROL | LCD_DISPLAYOFF | LCD_CURSOROFF | LCD_BLINKOFF);
92     HAL_Delay(2);
93     lcd_send_cmd(LCD_CLEARDISPLAY);
94     HAL_Delay(2);
95     lcd_send_cmd(LCD_ENTRYMODESET | LCD_ENTRYRIGHT | LCD_ENTRYSHIFTDISABLE);
96     HAL_Delay(2);
97     lcd_send_cmd(LCD_DISPLAYCONTROL | LCD_DISPLAYON | LCD_CURSOROFF | LCD_BLINKOFF);
98     HAL_Delay(2);
99
100 }
```

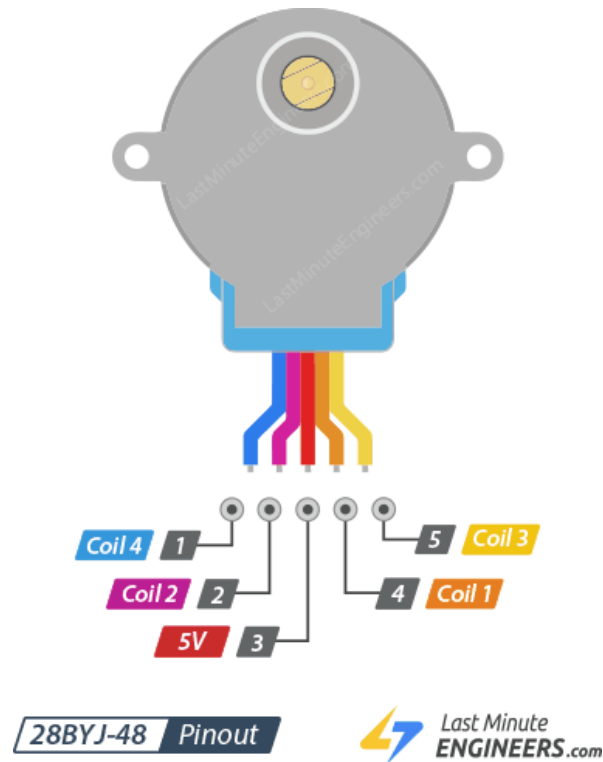


HD44780U

Infine, per controllare il display tramite la board STM32 è stata implementata una libreria software che offre poche e semplici funzioni di base, necessarie per poter utilizzare il dispositivo.

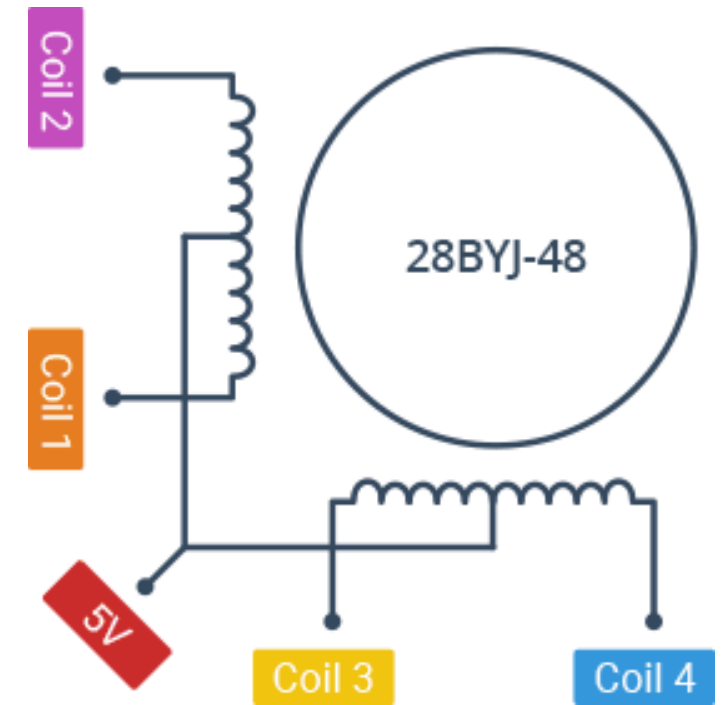
```
lcd_i2c.c  lcd_i2c.h X
36 #define LCD_BLINKON 0x01
37 #define LCD_BLINKOFF 0x00
38
39 // flags for display/cursor shift
40 #define LCD_DISPLAYMOVE 0x08
41 #define LCD_CURSORMOVE 0x00
42 #define LCD_MOVERIGHT 0x04
43 #define LCD_MOVELEFT 0x00
44
45 // flags for function set
46 #define LCD_8BITMODE 0x10
47 #define LCD_4BITMODE 0x00
48 #define LCD_2LINE 0x08
49 #define LCD_1LINE 0x00
50 #define LCD_5x10DOTS 0x04
51 #define LCD_5x8DOTS 0x00
52
53 // flags for backlight control
54 #define LCD_BACKLIGHT 0x08
55 #define LCD_NOBACKLIGHT 0x00
56
57 #define En 0x04 // Enable bit
58 #define Rw 0x02 // Read/Write bit
59 #define Rs 0x01 // Register select bit
60
61
62 //Prototipi
63 void lcd_send_cmd(uint8_t cmd);
64 void lcd_write(const char* s);
65 void lcd_init(I2C_HandleTypeDef* handle);
66 void lcd_clear();
67 int lcd_set_cursor(uint8_t row, uint8_t col);
68
69
```

Step Motor



Il 28BYJ-48 è un motore passo-passo unipolare a 5 fili che funziona a 5V. La particolarità di questo motore è che può essere posizionato con precisione un "passo" alla volta.

All'interno del motore sono presenti due bobine, al centro di ogni bobine abbiamo la connessione alla tensione di alimentazione (5V – filo di colore rosso), mentre gli altri connettori (arancione, rosa, giallo, blu) saranno connessi ciclicamente a massa in modo che ci sia passaggio di corrente tra questi e il positivo (rosso). Ogni volta che vi sarà il passaggio di corrente in queste bobine, l'albero del motore compirà un passo.





Step Motor

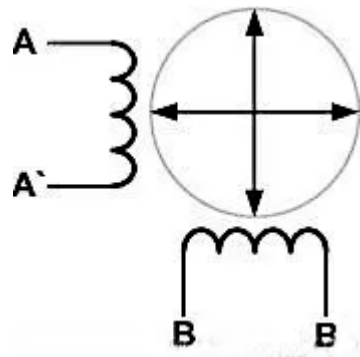
Tali motori vengono forniti con un IC ULN2003. Questo circuito integrato viene utilizzato per azionare il motore, poiché i pin del micro controllore non sono in grado di fornire corrente sufficiente. Dunque collegheremo la scheda STM32F3 a questo IC e i fili saranno collegati sul micro controllore a dei GPIO configurati in output con livello in uscita Low, in particolare:

- IN1 → PA1
- IN2 → PA2
- IN3 → PA3
- IN4 → PA4

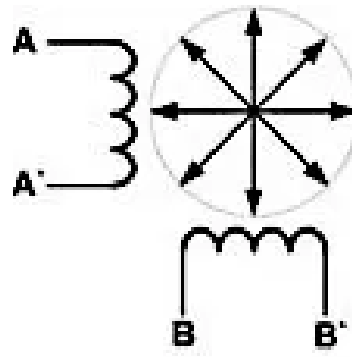
Sfrutteremo, per il nostro progetto, tale motore per aprire e chiudere la serratura della porta.

Step Motor

Half Drive Stepping Sequence				
Step	A	B	C	D
1	1	0	0	0
2	1	1	0	0
3	0	1	0	0
4	0	1	1	0
5	0	0	1	0
6	0	0	1	1
7	0	0	0	1
8	1	0	0	1



Full Drive



Half Drive

Esistono tre diversi tipi di modalità passo-passo utilizzate per questi motori:

- Wave Drive
- Full Drive
- Half Drive

Sceghieremo per questo progetto la modalità Half Drive poiché, rispetto alla modalità Full Drive, presenta i seguenti vantaggi: raddoppiare la precisione e fornire meno vibrazioni durante il funzionamento a bassa velocità.

In tale modalità, ogni passo corrisponderà ad una rotazione di $5,625^\circ$. Ciò significa che verranno effettuati 64 passi per giro ($360^\circ/5,625^\circ = 64$), però il motore possiede al suo interno una riduzione di $1/63,68395$ che viene approssimato ad $1/64$ ciò implica che ci saranno in realtà 64×64 passi per giro = 4096 passi.

Step Motor

La function «stepper_half_drive» permette di far funzionare il motore in modalità Half Drive. Il motore compie 8 step per completare 1 sequenza in questa modalità.

Infatti per poter far compiere al motore una rivoluzione di 360° bisogna abilitare i pin seguendo la sequenza, come mostrata in figura, 512 volte ($4096 / 8 = 512$).

```
16 void stepper_half_drive (int step)
17 {
18     switch (step){
19         case 0:
20             HAL_GPIO_WritePin(GPIOA, GPIO_PIN_1, GPIO_PIN_SET); // IN1
21             HAL_GPIO_WritePin(GPIOA, GPIO_PIN_2, GPIO_PIN_RESET); // IN2
22             HAL_GPIO_WritePin(GPIOA, GPIO_PIN_3, GPIO_PIN_RESET); // IN3
23             HAL_GPIO_WritePin(GPIOA, GPIO_PIN_4, GPIO_PIN_RESET); // IN4
24             break;
25
26         case 1:
27             HAL_GPIO_WritePin(GPIOA, GPIO_PIN_1, GPIO_PIN_SET); // IN1
28             HAL_GPIO_WritePin(GPIOA, GPIO_PIN_2, GPIO_PIN_SET); // IN2
29             HAL_GPIO_WritePin(GPIOA, GPIO_PIN_3, GPIO_PIN_RESET); // IN3
30             HAL_GPIO_WritePin(GPIOA, GPIO_PIN_4, GPIO_PIN_RESET); // IN4
31             break;
32
33         case 2:
34             HAL_GPIO_WritePin(GPIOA, GPIO_PIN_1, GPIO_PIN_RESET); // IN1
35             HAL_GPIO_WritePin(GPIOA, GPIO_PIN_2, GPIO_PIN_SET); // IN2
36             HAL_GPIO_WritePin(GPIOA, GPIO_PIN_3, GPIO_PIN_RESET); // IN3
37             HAL_GPIO_WritePin(GPIOA, GPIO_PIN_4, GPIO_PIN_RESET); // IN4
38             break;
39
40         case 3:
41             HAL_GPIO_WritePin(GPIOA, GPIO_PIN_1, GPIO_PIN_RESET); // IN1
42             HAL_GPIO_WritePin(GPIOA, GPIO_PIN_2, GPIO_PIN_SET); // IN2
43             HAL_GPIO_WritePin(GPIOA, GPIO_PIN_3, GPIO_PIN_SET); // IN3
44             HAL_GPIO_WritePin(GPIOA, GPIO_PIN_4, GPIO_PIN_RESET); // IN4
45             break;
46
47         case 4:
48             HAL_GPIO_WritePin(GPIOA, GPIO_PIN_1, GPIO_PIN_RESET); // IN1
49             HAL_GPIO_WritePin(GPIOA, GPIO_PIN_2, GPIO_PIN_RESET); // IN2
50             HAL_GPIO_WritePin(GPIOA, GPIO_PIN_3, GPIO_PIN_SET); // IN3
51             HAL_GPIO_WritePin(GPIOA, GPIO_PIN_4, GPIO_PIN_RESET); // IN4
52             break;
53
54         case 5:
55             HAL_GPIO_WritePin(GPIOA, GPIO_PIN_1, GPIO_PIN_RESET); // IN1
56             HAL_GPIO_WritePin(GPIOA, GPIO_PIN_2, GPIO_PIN_RESET); // IN2
57             HAL_GPIO_WritePin(GPIOA, GPIO_PIN_3, GPIO_PIN_SET); // IN3
58             HAL_GPIO_WritePin(GPIOA, GPIO_PIN_4, GPIO_PIN_SET); // IN4
59             break;
60
61         case 6:
62             HAL_GPIO_WritePin(GPIOA, GPIO_PIN_1, GPIO_PIN_RESET); // IN1
63             HAL_GPIO_WritePin(GPIOA, GPIO_PIN_2, GPIO_PIN_RESET); // IN2
64             HAL_GPIO_WritePin(GPIOA, GPIO_PIN_3, GPIO_PIN_RESET); // IN3
65             HAL_GPIO_WritePin(GPIOA, GPIO_PIN_4, GPIO_PIN_SET); // IN4
66             break;
67
68         case 7:
69             HAL_GPIO_WritePin(GPIOA, GPIO_PIN_1, GPIO_PIN_SET); // IN1
70             HAL_GPIO_WritePin(GPIOA, GPIO_PIN_2, GPIO_PIN_RESET); // IN2
71             HAL_GPIO_WritePin(GPIOA, GPIO_PIN_3, GPIO_PIN_RESET); // IN3
72             HAL_GPIO_WritePin(GPIOA, GPIO_PIN_4, GPIO_PIN_SET); // IN4
73             break;
74     }
75 }
76 }
```


Step Motor

Attraverso questa function è possibile scegliere di quanti gradi il motore dovrà girare. Possiamo inoltre scegliere il verso della rotazione (orario o antiorario) e la velocità (RPM, numero di giri al minuto).

```
78 void stepper_step_angle (float angle, int direction, int rpm)
79 {
80     int numberofsequences = (int) ((angle*512)/360); // (512 : 360 = x : angle)
81
82     for (int seq=0; seq < numberofsequences; seq++)
83     {
84         if (direction == 0) // ruota in senso antiorario
85         {
86             for (int step=7; step >= 0; step--)
87             {
88                 stepper_half_drive(step);
89                 stepper_set_rpm(rpm);
90             }
91         }
92
93         else if (direction == 1) //ruota in senso orario
94         {
95             for (int step=0; step < 8; step++)
96             {
97                 stepper_half_drive(step);
98                 stepper_set_rpm(rpm);
99             }
100         }
101     }
102 }
103
104 }
```

Questa function permette di generare un ritardo tra uno step e l'altro inversamente proporzionale al valore RPM dato in ingresso. In questo modo si riesce ad avere un controllo della velocità del motore.

```
11 void stepper_set_rpm (int rpm)
12 {
13     delay(600000/stepsperrev/rpm); // rpm = n_giri(60") = (steps_angle/360) * n_step(1") * 60" = stepsperrev * 60 * 1/t_s --> t_s = 60s / stepsperrev / rpm
14 }
15
```

stepsperrev = 4096

Step Motor

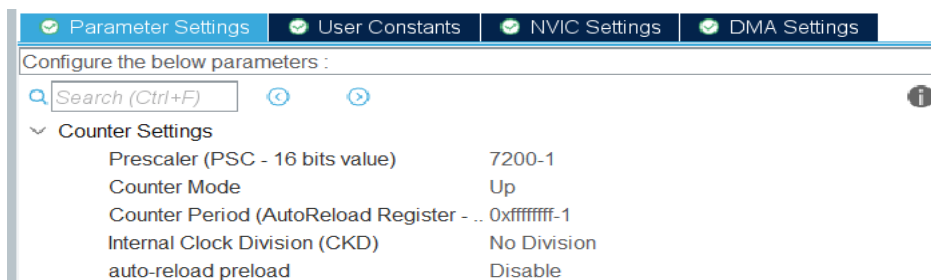
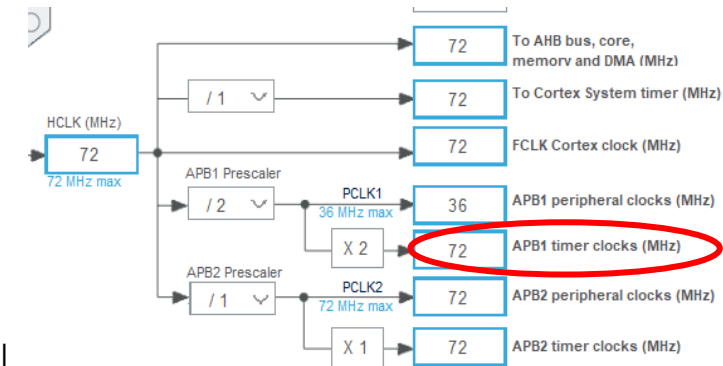
Abbiamo dunque bisogno di un timer, che permette di generare un ritardo tra uno step e l'altro. Dunque attiviamo un timer generico, selezionando come sorgente del clock il clock interno. In questo progetto utilizzeremo il TIM2 che è collegato al clock APB1 (informazione fornita dal datasheet).

Dalla figura di lato si evince che il clock APB1 è a 72MHz, questo significa che anche il timer 2 lavora a questa frequenza.

Ora riduciamo questa frequenza, settando i parametri nella sezione di configurazione del timer, impostando il prescaler a 7200-1 in modo tale che il nostro timer riesca a contare ogni 10^{-4} s ($72000000 \text{ Hz} * 10^{-4} \text{ s} = 7200$). Invece settiamo il counter period al suo valore massimo 0xffffffff (registro a 16 bits).

N.B.

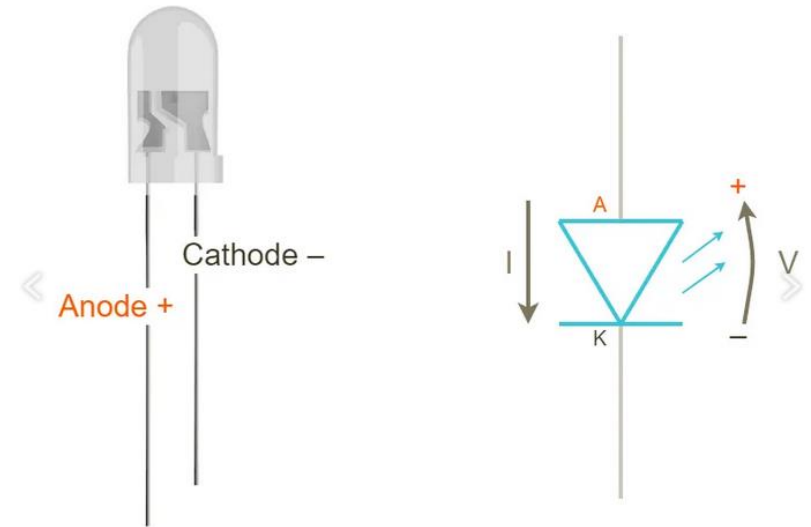
7200-1 poiché il registro inizia a contare da 0 anziché 1.



```
9 void delay(double t) // 1 s --> t = 10000
10 {
11     __HAL_TIM_SET_COUNTER(&htim2, 0);
12     while (__HAL_TIM_GET_COUNTER(&htim2) < t);
13 }
```

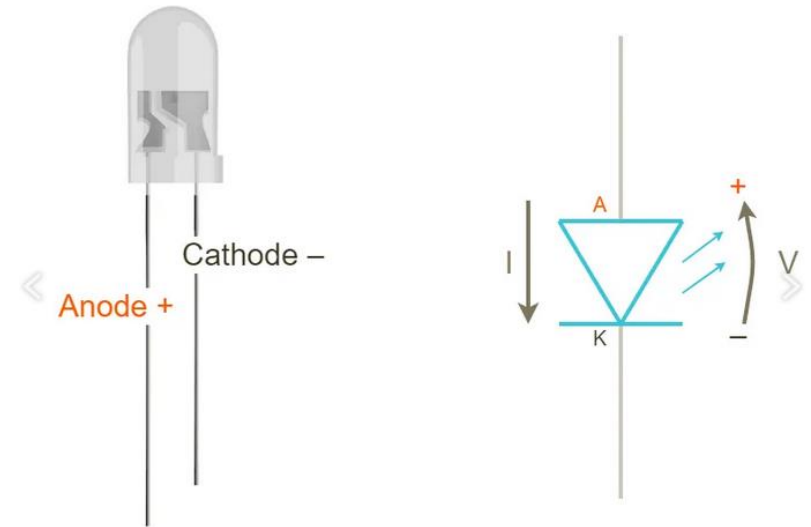
Led

Un led è un diodo che emette luce quando attraversato da corrente. Come ogni diodo, i led hanno un anodo e un catodo. Affinché scorra corrente attraverso il led è necessario applicare una differenza di potenziale positiva tra l'anodo e il catodo, il cui valore superi la tensione di soglia del diodo. Inoltre, l'intensità luminosa del led è proporzionale alla corrente che lo attraversa, e non alla tensione applicata ai suoi estremi.



Led

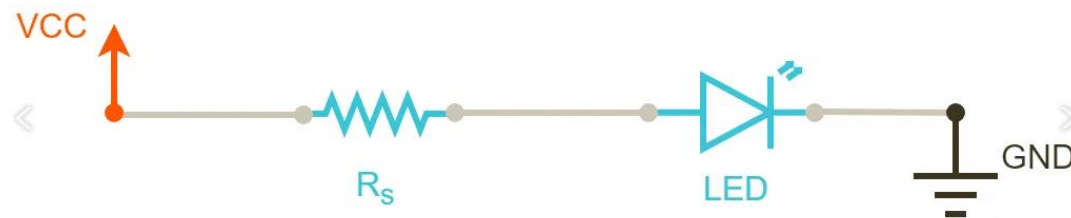
Pilotare un led applicando direttamente una tensione ai suoi estremi è sconsigliato dato che i led soffrono il problema del **thermal drift**. Applicare una tensione ai capi del diodo provoca il passaggio di una corrente attraverso di esso, che a sua volta comporta un aumento della temperatura del componente. Tale aumento di temperatura causa un aumento dei portatori di carica nel diodo, che induce un incremento dell'intensità di corrente, che implica un continuo aumento di temperatura, e così via, fino a che la corrente non sarà così elevata da bruciare il componente.



Led

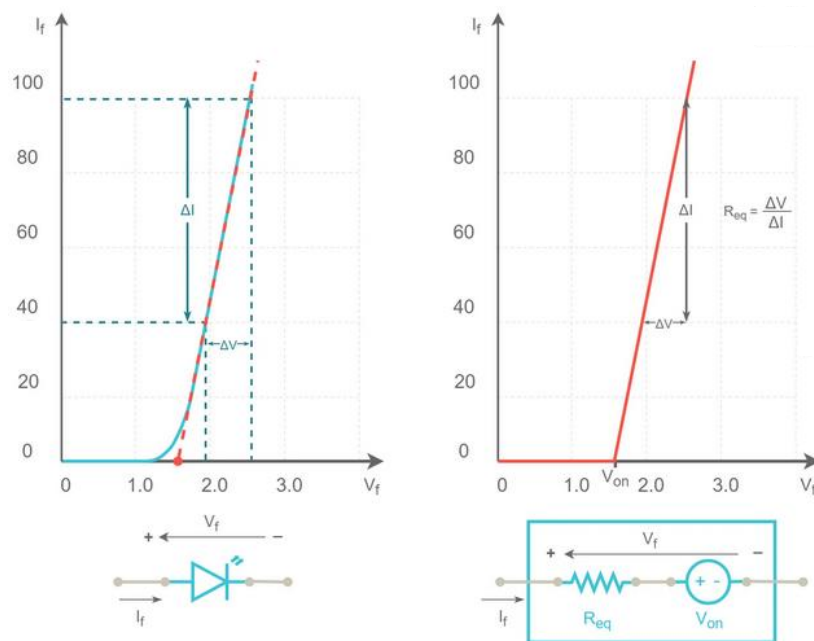
Pilotare un led applicando direttamente una tensione ai suoi estremi è sconsigliato dato che i led soffrono il problema del **thermal drift**. Applicare una tensione ai capi del diodo provoca il passaggio di una corrente attraverso di esso, che a sua volta comporta un aumento della temperatura del componente. Tale aumento di temperatura causa un aumento dei portatori di carica nel diodo, che induce un incremento dell'intensità di corrente, che implica un continuo aumento di temperatura, e così via, fino a che la corrente non sarà così elevata da bruciare il componente.

Il modo più semplice per risolvere tale problema è inserire un resistore in serie al led, in modo tale che, all'aumentare della corrente, aumenti la tensione ai capi del resistore, diminuendo di conseguenza quella ai capi del led, ed evitando così che si generino correnti troppo elevate nel circuito.



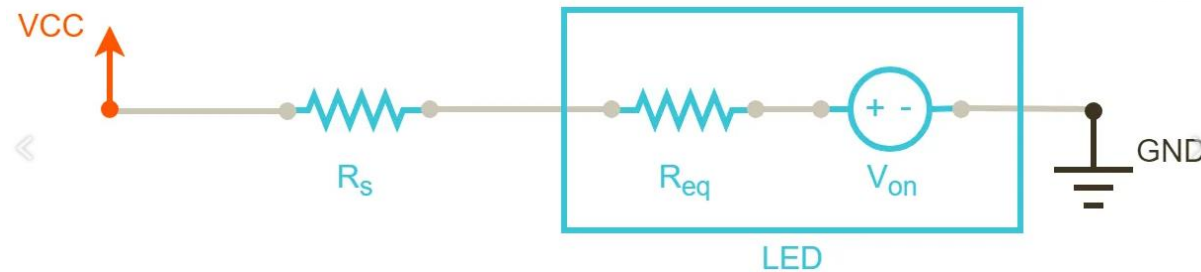
Led

Ciò che ci resta da fare ora è dimensionare il resistore da inserire in serie al led nel circuito per ottenere l'intensità luminosa desiderata. Per fare ciò è utile costruire un modello equivalente del diodo, composto da un resistore ed un generatore di tensione in serie, in modo da poter fare dei conti approssimativi.



Led

Dalla curva caratteristica del led si ricavano i valori di R_{eq} e V_{on} , si fissa la corrente I che deve scorrere nel circuito a seconda dell'intensità luminosa desiderata, e così è possibile risolvere il circuito sottostante per calcolare il valore della resistenza R_s .



$$V_{cc} - V_{on} = (R_s + R_{eq}) \cdot I$$

$$R_s = \frac{(V_{cc} - V_{on})}{I} - R_{eq}$$

Led

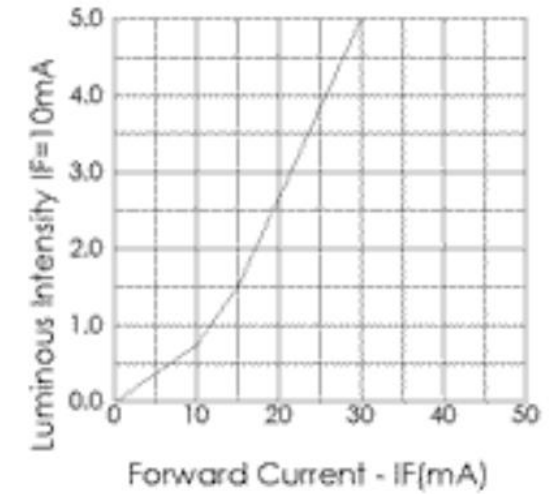
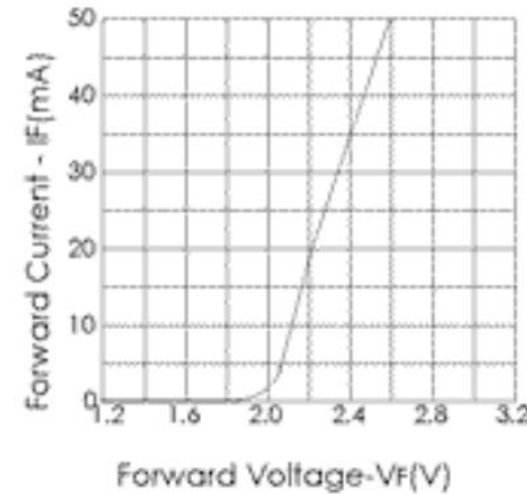
Esempio: Led Rosso

$$R_{eq} = \frac{2,4 \, \Omega - 2,2 \, \Omega}{(35 - 20) \cdot 10^{-3} \, A} \cong 13,3 \, \Omega$$

$$V_{on} = 2 \, V$$

$$I = 50 \, mA$$

$$V_{cc} = 3,3 \, V$$



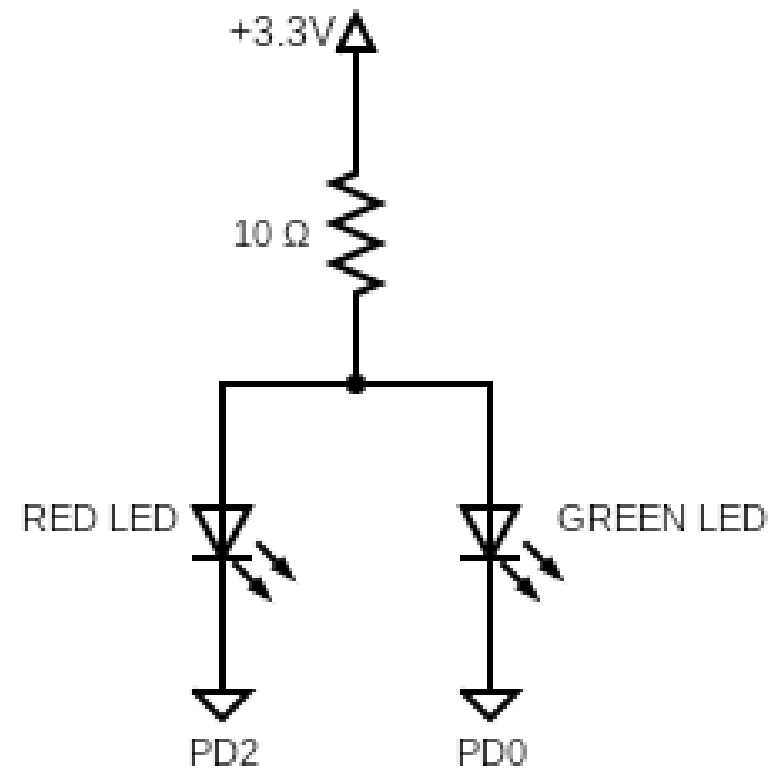
$$R_s = \frac{(V_{cc} - V_{on})}{I} - R_{eq} = \frac{3,3 \, V - 2 \, V}{50 \cdot 10^{-3} \, A} - 13,3 \, \Omega = 12,7 \, \Omega$$

Conti simili si effettuano per il led verde.

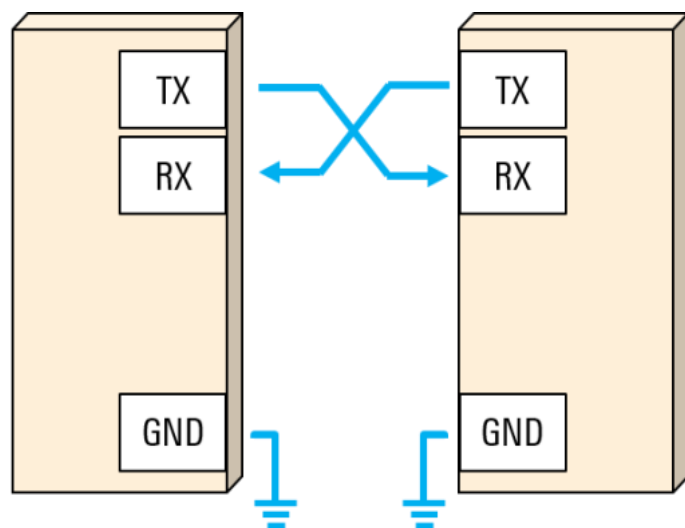
Led

Il led sono collegati ai GPIO della board STM32-Discovery nella configurazione a qui a destra.

- **GPIO-High:** led spenti
- **GPIO-Low:** led accesi



Comunicazione seriale tra le schede STM32



Lo UART o Universal Asynchronous Receiver-Transmitter (ricevitore-trasmittitore asincrono universale) è un dispositivo hardware, di uso generale o dedicato, che converte flussi di bit di dati da un formato parallelo a un formato seriale asincrono o viceversa. Gli USART (Universal Synchronous-Asynchronous Receiver/Transmitter) costituiscono un'evoluzione degli UART, come si evince dal nome stesso, in grado di gestire anche trasmissioni seriali sincrone.

Per la comunicazione tramite USART, utilizziamo soltanto due fili fra il trasmettitore e il ricevitore per trasmettere e ricevere in entrambe le direzioni. Entrambe le estremità hanno anche una connessione a massa (nel nostro caso le due schede saranno collegate allo stesso GND).

USART

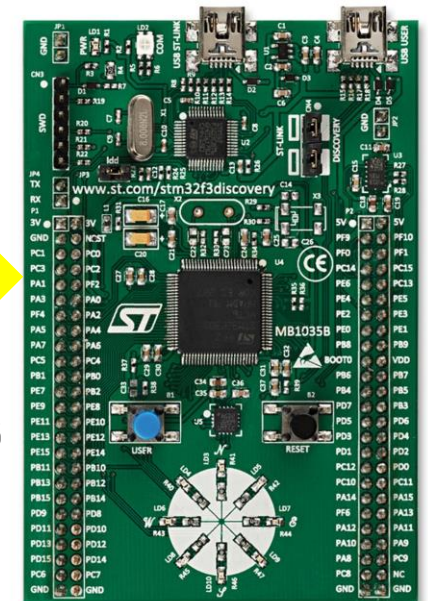
Per il nostro progetto, le due schede comunicheranno attraverso lo USART 1, collegando rispettivamente i pin mostrati in figura. La scheda di input (STM32 Nucleo) invierà all'altra scheda il PIN, inserito tramite la keyboard, o lo UID delle card, lette attraverso il sensore RFID. La scheda di output (STM32F3), una volta ricevuta questa informazione, validerà o meno l'apertura della serratura. Di seguito verranno mostrati i passaggi per realizzare tale comunicazione.

Nucleo-64 STM32F401



RX → PC7
TX → PC6

Discovery STM32F3



RX → PC5
TX → PC4

USART

Per questo progetto, utilizzeremo lo USART1 sia per la scheda STM32F3 che la scheda Nucleo-64, entrambe abilitate in modalità asincrona. Per ricevere dati, utilizzeremo il metodo delle interruzioni, in questo modo il processore verrà interrotto solo quando il messaggio è stato completamente ricevuto, lasciandolo così libero per l'elaborazione di altri processi. Setteremo inoltre entrambe le USART con i parametri mostrati nella figura in basso.

Parameter Settings

User Constants

NVIC Settings

DMA Settings

GPIO Settings

Configure the below parameters :

Search (Ctrl+F)

Basic Parameters

Baud Rate

Word Length

Parity

Stop Bits

9600 Bits/s

8 Bits (including Parity)

None

1

Advanced Parameters

Advanced Features

Mode

Asynchronous

Hardware Flow Control (RS232)

Disable

Hardware Flow Control (RS485)

Configuration

Reset Configuration

User Constants

NVIC Settings

DMA Settings

GPIO Settings

Parameter Settings

NVIC Interrupt Table

Enabled

Preemption Priority

Sub Priority

USART1

through EXTI line 25

☒

0

0



USART

Per abilitare il device alla ricezione, seguendo il metodo delle interruzioni, richiameremo la seguente function (fuori dal programma principale), indicando l'indirizzo del buffer ricevente e il numero di byte che deve ricevere:

```
HAL_UART_Receive_IT(&huart1, recvbuffer, 1);
```

Una volta fatto ciò, il dispositivo è pronto a ricevere dal trasmettitore e verrà eseguita una ISR ogni volta che la ricezione è completata. La relativa ISR è riportata di seguito:

```
void HAL_UART_RxCpltCallback(UART_HandleTypeDef *huart) { ... HAL_UART_Receive_IT(&huart1, recvbuffer, 1); }
```

N.B.

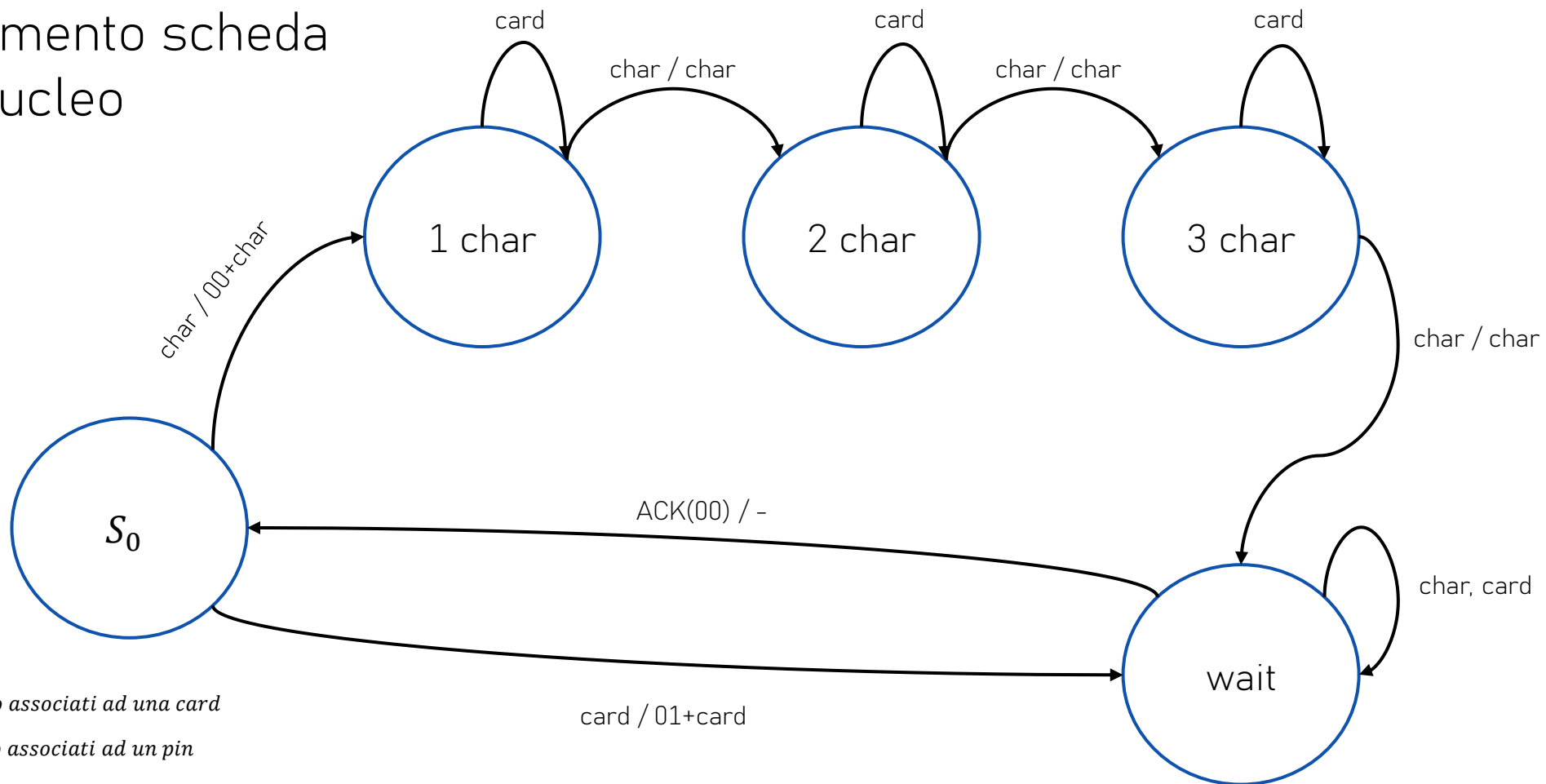
Bisogna riavviare la ricezione (all'interno della funzione HAL_UART_RxCpltCallback), altrimenti la ricezione avverrà solo una volta.

ISR di funzionamento

```
190 void HAL_UART_RxCpltCallback(UART_HandleTypeDef * huart){
191
192     cont = cont + 1;
193
194     //WRITE ON LCD "PIN: ****"
195     if(recvbuffer[0] == 0x00){
196         switch(cont){
197             case 1:
198                 lcd_clear();
199                 delay(1000); // wait 100 ms
200                 lcd_write("PIN: ");
201                 break;
202
203             default:
204                 lcd_write("");
205                 break;
206         }
207     }
208
209     if(cont == 5) {
210
211         switch(recvbuffer[0]){
212             case 0x00:
213                 flag_err = flag_compare_pin(recvbuffer);
214                 break;
215
216             case 0x01:
217                 flag_err = flag_compare_card(recvbuffer);
218                 break;
219
220             default:
221                 break;
222         }
223     }
```

```
222         if(flag_err == 0x00){ //correct pin or card
223             open_door();
224             lcd_clear();
225             delay(5000); // wait for 5 s
226             close_door();
227             delay(1000); //wait for 100 ms
228             reset_lcd();
229
230         }
231         else { // wrong pin or card
232
233             card_pin_error();
234             reset_lcd();
235         }
236
237         HAL_UART_Transmit(&huart1, &send, 1, HAL_MAX_DELAY);
238         cont = 0;
239         flag_err = 0x00;
240     }
241     HAL_UART_Receive_IT(&huart1, &recvbuffer[cont], 1); //rimettiamo il device in ricezione
242 }
243
244
245
246
247
248
249
250
251
252 /* USER CODE BEGIN PV */
253 uint8_t recvbuffer[5]; //buffer di ricezione, conterrà il pin o la card utilizzato
254 uint8_t card[4] = {0x83, 0x16, 0x04, 0x1E}; //UID card abilitata allo sblocco della serratura
255 uint8_t pass[4] = "1234"; //Pin abilitato allo sblocco della serratura
256 int cont = 0; //contatore di caratteri ricevuti
257 uint8_t flag_err; //riconosce se la sequenza trasmessa è valida oppure no (0x01 -> ERRORE, 0x00 -> OK)
258 uint8_t send = 0x00; //ACK da trasmettere per sbloccare il trasmettitore
259
```

Funzionamento scheda STM32 Nucleo



01 → i prossimi byte inviati sono associati ad una card

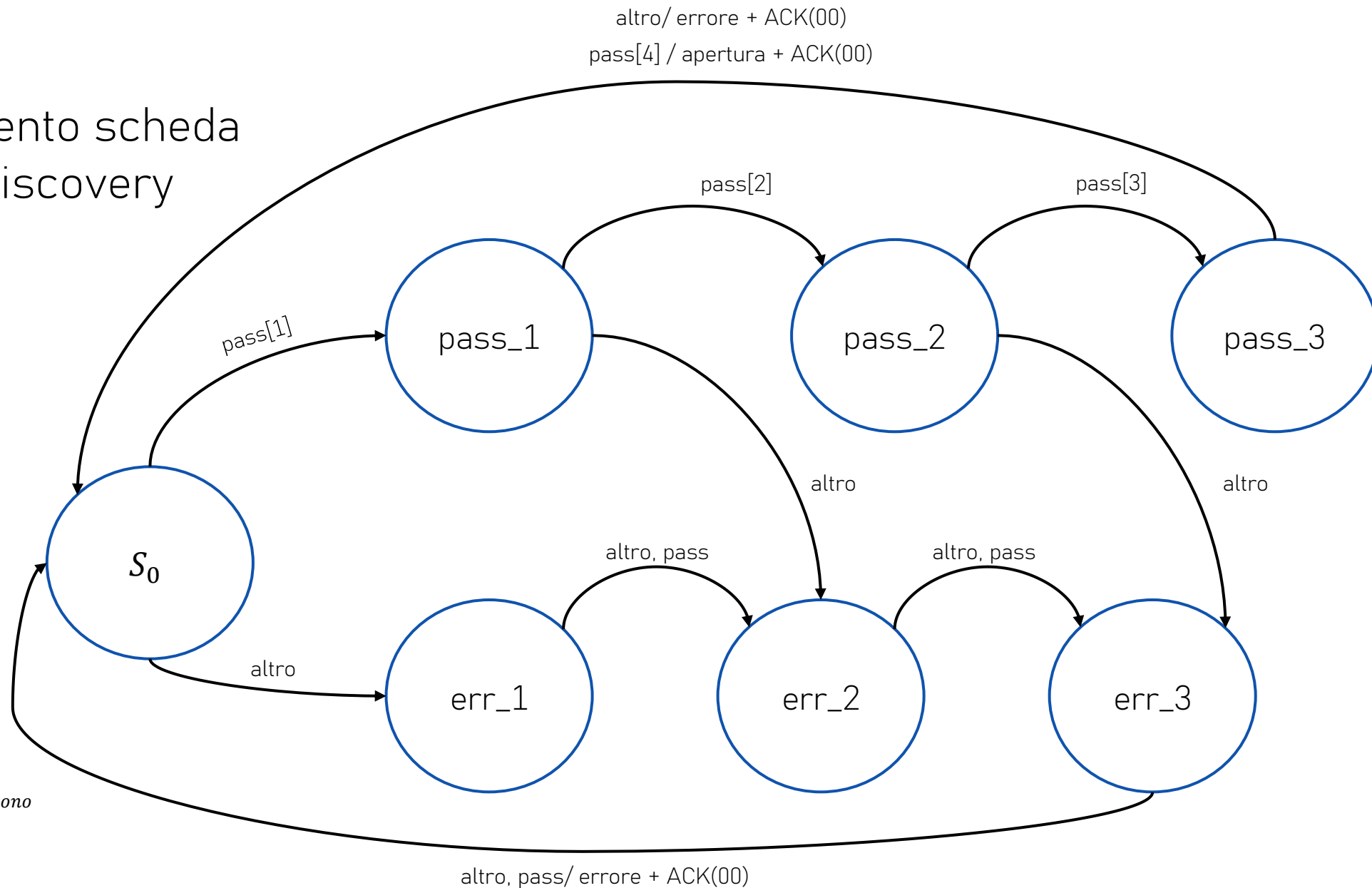
00 → i prossimi byte inviati sono associati ad un pin

card → UID della card (4 byte)

char → carattere (1 byte) inserito sulla keyboard

ACK(00) → resta in uno stato di attesa finchè non arriva l'ACK (byte 0x00)

Funzionamento scheda STM32F3 Discovery



altro ∈ ASCII / {pass}

pass → sequenza di 4 caratteri, riconoscono
se il pin o la card sono esatti

apertura → sequenza riconosciuta

errore → sequenza non riconosciuta

Demo

