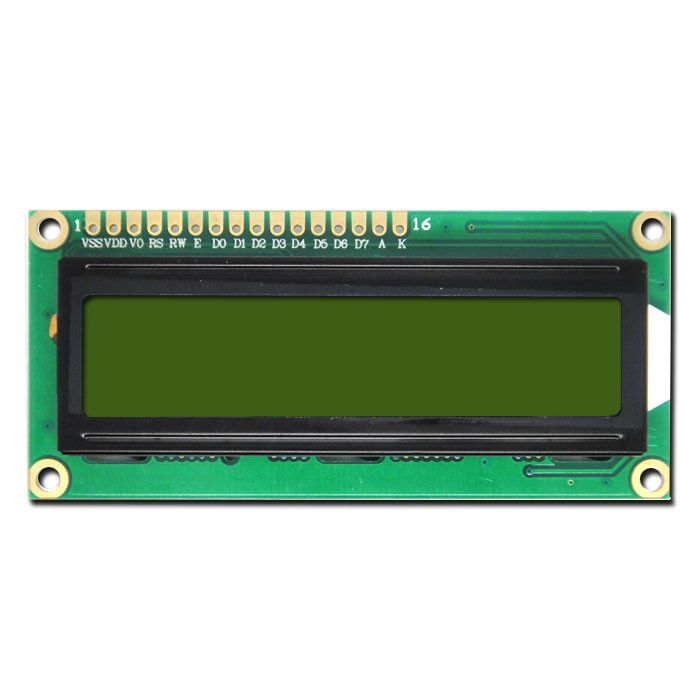
**Descrizione dei componenti**

I componenti utilizzati sono:

* *Fotoresistenza*: sensore la cui resistenza varia in base all’intensità luminosa. In particolare la resistenza del fotoresistore è inversamente proporzionale alla quantità di luce che lo colpisce.
* *Emettitore laser*: attuatore che emette luce attraverso un processo di amplificazione ottica basata sull’emissione stimolata di una radiazione elettromagnetica. A distinguerlo da altre fonti di luce è la coerenza: quella spaziale permette al raggio di essere focalizzato su un punto stretto, mentre quella temporale di avere uno spettro molto sottile (emissione di luce di un solo colore).
* *Led:* dispositivo optoelettronico che sfrutta la capacità di alcuni materiali semiconduttori di produrre fotoni attraverso un fenomeno di emissione spontanea. Essenzialmente è un diodo a giunzione p-n.
* *Sistema RFID (lettore + tag):*  con l'acronimo **RFID** (dall'inglese **Radio-Frequency IDentification**, in italiano **identificazione a radiofrequenza**) si intende una tecnologia per identificazione e/o memorizzazione automatica basata sulla capacità di memorizzazione di dati da parte di particolari etichette elettroniche, chiamate *tag* e sulla capacità di queste di rispondere all'interrogazione a distanza da parte di appositi apparati fissi o portatili, chiamati *reader*. Questa identificazione avviene mediante radiofrequenza. In particolare si è utilizzato un sistema RFID denominato MIFARE RC522, basato sull’omonimo chip Philips.
* *Servo motore:* un servomotore è un servosistema a ciclo chiuso che usa un feedback sulla posizione per controllare il suo movimento. L’ingresso di controllo a tale motore è un segnale, analogico o digitale, che rappresenta la posizione finale che esso deve raggiungere. In particolare è stato utilizzato un SG 90 micro-servo. Esso è controllato in PWM con un periodo di 20 ms e un duty cicle variabile che codifica la posizione desiderata.
* *Display LCD:* è stato utilizzato un display LCD a matrice di punti. In particolare è stato utilizzato un lcd dotato del comune controller HD44780 prodotto da Hitachi.

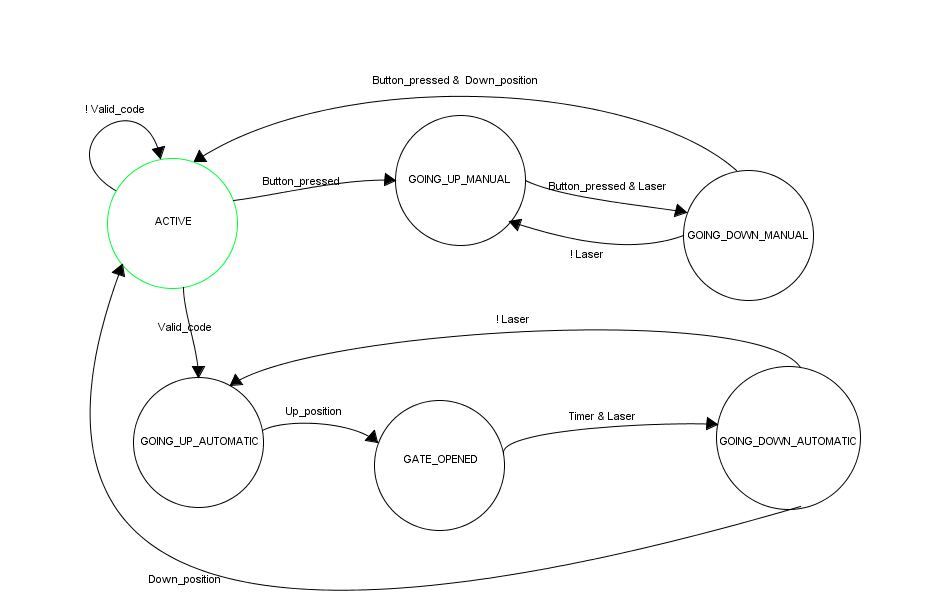


* *Altri componenti:* una resistenza da 2KΩ, jumper wires (M-M , M-F ,F-F) e un potenziometro.

**Descrizione ed obiettivi del sistema**

Si intende realizzare un sistema automatizzato per la gestione degli accessi. Nella versione base del sistema si permette solo a chi ha un lasciapassare di poter entrare all’interno (facendo uso della tecnologia RFID). Se il lasciapassare risulta valido, allora la barra metallica per il transito si alza per poi abbassarsi dopo un dato lasso di tempo a meno che il raggio laser, che è puntato sulla fotoresistenza, non sia interrotto. Analogamente, se questo fascio viene interrotto mentre la sbarra si abbassa essa viene rialzata. Vicino al lettore di schede, posto all’entrata, è presente anche uno schermo LCD per fornire un minimo di interfaccia utente utile a conoscere l’esito della lettura ed, eventualmente, altre informazioni. È possibile inoltre alzare o abbassare la sbarra tramite un pulsante; questa opzione può essere utile, ad esempio, per operazioni di carico/scarico o per l’accesso saltuario di persone sprovviste del tag di accesso. È inoltre fornita all’operatore/custode, una shell che da terminale permette di visualizzare informazioni sullo stato del sistema (come lista degli utenti autorizzati) e di interagire con esso rimuovendo o aggiungendo codici autorizzati sempre tramite un lettore RFID.

**Descrizione della macchina a stati**



Stati:

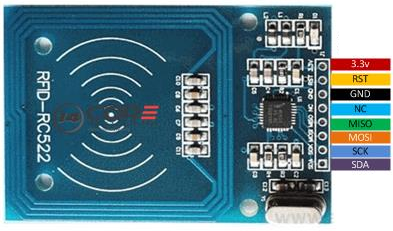
* ACTIVE: stato in cui il sistema è attivo (stato iniziale). In questo stato la sbarra è abbassata e il sistema è pronto a rilevare i tag RFID o il comando manuale per alzare la sbarra.
* GOING\_UP\_MANUAL: stato in cui sta avvenendo il sollevamento della sbarra in seguito alla pressione dell’opportuno pulsante.
* GOING\_DOWN\_MANUAL: stato in cui si sta abbassando la sbarra, in seguito alla pressione dell’opportuno pulsante, se non è interrotto il fascio di luce della fotocellula.
* GOING\_UP\_AUTOMATIC: stato in cui sta avvenendo il sollevamento della sbarra in seguito al passaggio di un tag autorizzato sul lettore.
* GATE\_OPENED: stato in cui si transita quando viene ultimato il sollevamento automatico della sbarra dovuto al riconoscimento di un tag autorizzato.
* GOING\_DOWN\_AUTOMATIC: stato in cui si sta abbassando la sbarra se sono passati alcuni secondi dal passaggio del tag autorizzato e se il fascio di luce del laser non è interrotto.

**Ingressi:**

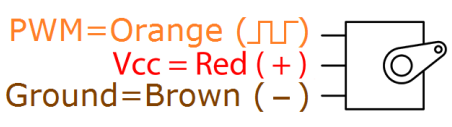
* Valid\_code: ingresso che segnala la lettura di un tag autorizzato da parte dell’RFID reader.
* Button\_pressed: ingresso che segnala la pressione del pulsante per l’utilizzo manuale.
* Down\_position: è vero se la sbarra è in posizione bassa.
* Up\_position: è vero se la sbarra è in posizione alta.
* Laser: è vero se il fascio laser non è interrotto.
* Timer: è vero quando la permanenza nello stato GATE\_OPENED raggiunge un tempo prestabilito.

|  |  |
| --- | --- |
| **Modulo RFID** | **STM32F3** |
| 3.3V | 3.3.V |
| RST | Non connesso |
| GND | Ground |
| IRQ | Non connesso |
| MISO | PB14 |
| MOSI | PB15 |
| SCK | PB13 |
| SDA | PD11 |

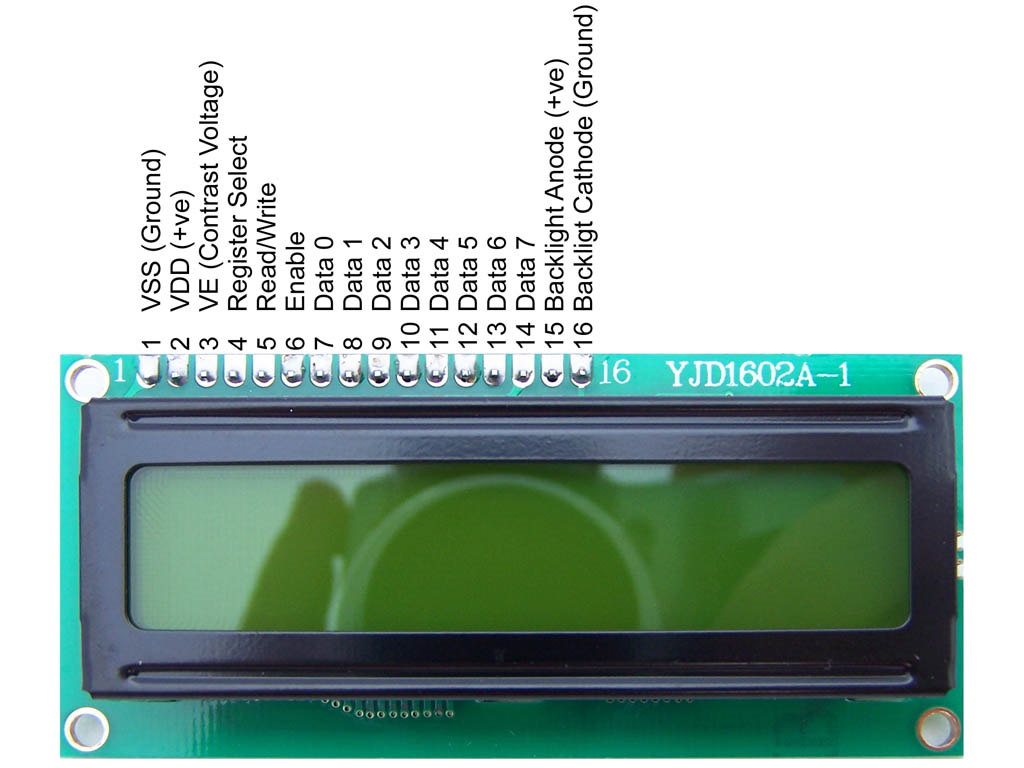
**Schematico delle connessioni**



|  |  |
| --- | --- |
| **Servo motore** | **STM32F3** |
| PWM | PB11 |
| Vcc | 5V |
| Ground | Ground |



|  |  |
| --- | --- |
| **LCD** | **STM32F3** |
| VSS | Ground |
| VDD | 5 V |
| VE | Collegato al terminale di pilotaggio di un potenziometro (per regolare il contrasto) |
| RS | PA4 |
| RW | PA1 |
| ENABLE | PA3 |
| DATA 0 | PC0 |
| DATA 1 | PC1 |
| DATA 2 | PC2 |
| DATA 3 | PC3 |
| DATA 4 | PB0 |
| DATA 5 | PB10 |
| DATA 6 | PC6 |
| DATA 7 | PC7 |
| A | 3 V |
| K | Ground |



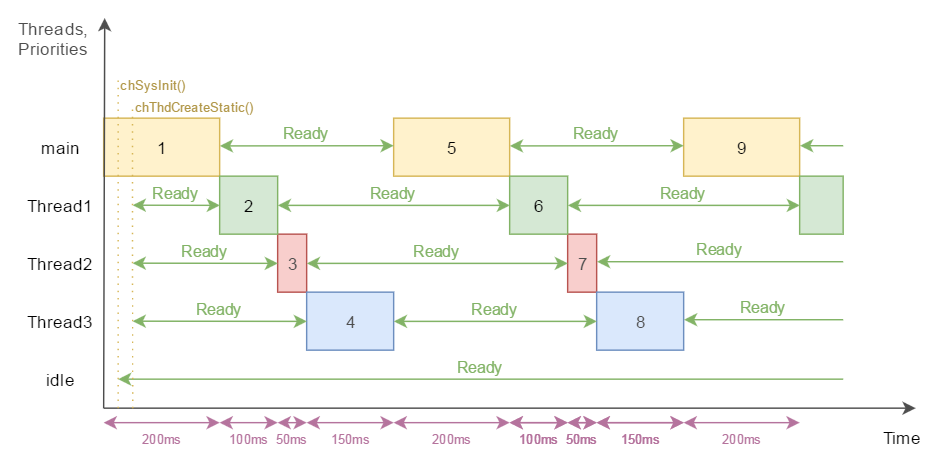
**Note:**

****L’ingresso “!Laser” è ottenuto attraverso la circuiteria mostrata nella figura accanto, e in particolare quella con il parallelo fra il led e il fotoresistore. Il segnale d’ingresso è prelevato prima del parallelo e dopo la resistenza di pull-up. Inoltre quest’ultima è di 2KΩ.

**Descrizione del software**

**Architettura**

Per la realizzazione del software è stato impiegato ChibiOS, un ambiente di sviluppo completo per applicazioni embedded che include un RTOS, un HAL e driver per le periferiche. L’architettura del software è di tipo cooperative multi-threading: tutti i thread hanno la stessa priorità ed ognuno di essi deve rilasciare la cpu attraverso l’uso di opportune API. Nella figura sottostante è riportato lo scenario di esecuzione di un programma con architettura cooperative multi-threading costituito da 3 thread oltre al main.



Il software per il controllo del sistema di accessi è costituito da 6 thread oltre al main. Un thread si occupa dell’attuazione (controllo del motore per il sollevamento della sbarra), un altro thread accende e spegne un led facendo da watchdog per l’esecuzione del programma, mentre gli altri 4 monitorano costantemente gli ingressi quali: RFID reader, pulsante per l’uso manuale, raggio laser, la posizione della sbarra e aggiornano gli stati di conseguenza. Vi è anche un altro thread, il quale procede in maniera indipendente dalla macchina a stati, che ha il compito di eseguire i comandi lanciati tramite la shell.

**Moduli software**

All’interno del progetto software sono presenti, oltre ai moduli software necessari a controllare i dispositivi utilizzati, altri moduli specifici per la configurazione di ChibiOS tra i quali:

* halconf.h: viene utilizzato per specificare i servizi forniti da HAL che vengono effettivamente impiegati, in modo da ottimizzare le prestazioni.
* chconf.h: contiene parametri di configurazione di ChibiOS, è possibile ad esempio scegliere di abilitare uno scheduling preemptive etc.
* mcuconf.h: viene utilizzato per indicare le specifiche periferiche effettivamente utilizzate.

I moduli elencati sono quelli di cui è prevista la modifica da parte dell’utente di ChibiOS; ovviamente vengono inclusi e compilati anche tutti gli altri moduli necessari alla corretta esecuzione del RTOS.

Nella cartella principale del progetto si trova anche il file “main.c” in cui è contenuta tutta la logica del programma orientata all’utilizzo dei thread.

Nella cartella userlib sono contenuti tutti i moduli software relativi alla gestione dei sensori e attuatori presenti nel progetto. È presente anche un file denominato userconf.h che svolge funzioni analoghe ad halconf.h, ovvero abilita o meno la compilazione dei vari moduli software. Si presenta l’elenco dei moduli software presenti in userlib e delle relative interfacce. Verranno elencate e commentate soltanto le funzioni più significative, trascurando quelle che sono utilizzate internamente dalle librerie.

* lcd.c/lcd.h: modulo software per la gestione dell’lcd.

**void** **lcdInit**(**void**)

Inizializza il driver per l’lcd

**void** **lcdStart**(LCDDriver \*lcdp, **const** LCDConfig \*config)

Configura e attiva il driver per l’lcd.

**void** **lcdWriteString**(LCDDriver \*lcdp, **char**\* string, uint8\_t pos)

Scrive la stringa specificata a partire da una certa casella, indicata dal parametro pos, dell’lcd.

* mfrc522.c/ mfrc522.h: modulo software per la gestione del RFID reader.

**void** **MFRC522ObjectInit**(MFRC522Driver\* mfrc522p)

Inizializza il driver per il RFID reader.

**void** **MFRC522Start**(MFRC522Driver\* mfrc522p, **const** MFRC522Config\* config)

Configura e attiva il driver per il RFID reader.

MIFARE\_Status\_t **MifareCheck**(MFRC522Driver\* mfrc522p, **struct** MifareUID\* id);

Controlla la presenza di un tag nelle vicinanze del lettore e ne ritorna il codice attraverso la struttura di tipo MifareUID, passata per riferimento.

**extern** **void** **MFRC522WriteRegister**(MFRC522Driver\* mfrc522p, uint8\_t addr, uint8\_t val);

**extern** uint8\_t **MFRC522ReadRegister**(MFRC522Driver\* mfrc522p, uint8\_t addr);

Due funzioni, rispettivamente per scrivere e leggere i registri del RFID reader, da implementare in base alle esigenze. In questo caso esse sono state implemementate nel main e scrivono/leggono i registri tramite la periferica SPI2.

* motor.c/motor.h: modulo software per la gestione del motore.

**void** **motorObjectInit**(MotorDriver\* motorp);

Inizializza il driver per il motore.

**void** **motorStart**(MotorDriver\* motorp, **const** MotorConfig \*config);

Configura e attiva il driver per il motore.

**void** **motorUp**(MotorDriver\* motorp);

Effettua lo spostamento della staffa di circa un grado verso l’alto.

**void** **motorDown**(MotorDriver\* motorp);

Effettua lo spostamento della staffa di circa un grado verso il basso.

* rfidcodes.c/rfidcodes.h: modulo per la gestione di un insieme di codici di tag RFID contenuti in un array.

int8\_t **codeIndex**(RfidCodes \*validCodes, MifareUID \*code);

Nel caso in cui il codice del tag è presente all’interno della struttura validCodes ritorna il suo indice, altrimenti ritorna -1.

bool **equalCodes**(MifareUID \*code1, MifareUID \*code2);

Ritorna true se i due codici sono uguali.