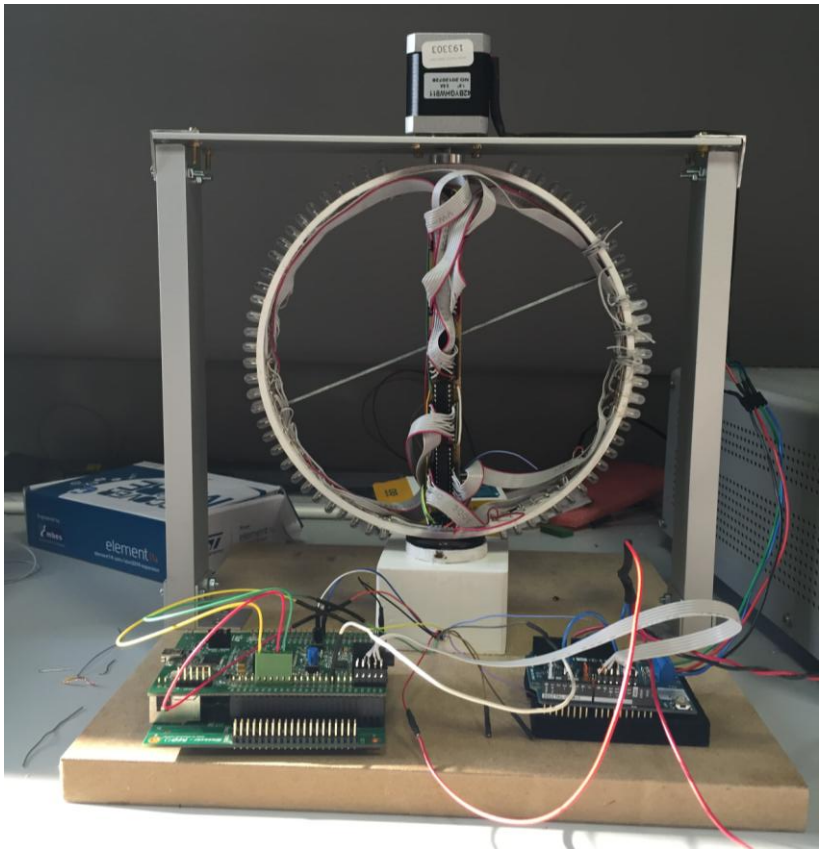
CORSO DI LAUREA MAGISTRALE IN EMBEDDED COMPUTING SYSTEMS

DESIGN OF EMBEDDED SYSTEMS



LED GLOBE

STUDENTI: Silvia Panicacci, Marco Marini

EMAIL: s.panicacci@live.it, marcomarini93@gmail.com

ANNO ACCADEMICO 2015/2016

DATA DI CONSEGNA: 8/02/2016

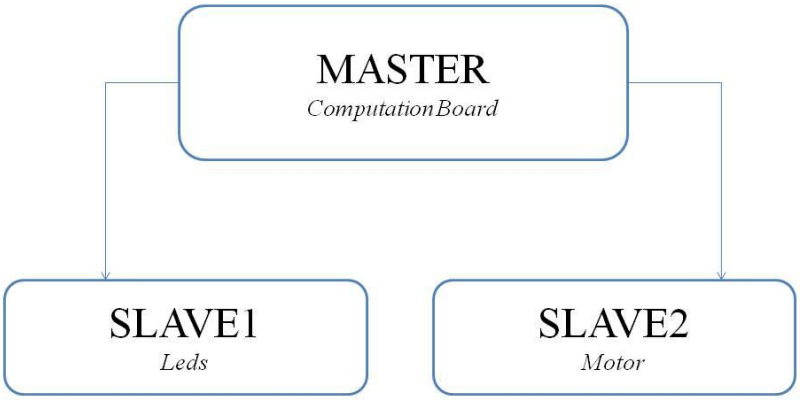
Indice

1. Requisiti generali

Il sistema che si vuole realizzare consiste nell'immagine di un globo terrestre, generato da una striscia di led, ruotata da un motore.

2. Analisi dei requisiti

Il sistema può essere descritto con un pattern *master-slave*, in cui il *master* è una scheda computazionale, che invia dati per pilotare il motore e per accendere o spegnere i led, in modo sincrono. Gli *slaves*, invece, sono i led e il motore.



2.1 MASTER: *Computation Board*

La *computation board* scelta è la STM32F4 Discovery (http://www.st.com/st-web-ui/static/active/cn/resource/technical/document/user_manual/DM00039084.pdf), con il microcontrollore STM32F407VGT (ARM Cortex-M4 32-bit). Compito della *board* è di pilotare i

Figura 1. STM32F4 Discovery

due *slaves*. A tal fine, vengono utilizzati 4 degli 80 pin GPIO e 3 delle 6 porte SPI.

2.2 SLAVE 1: *Leds*

I *leds* utilizzati sono 80. Essi sono montati su una circonferenza in plastica e sono così disposti: 64 (32 da una parte e 32 dall'altra), di colore bianco, si trovano lungo tutta la circonferenza, mentre gli altri 16, di colore rosso, sono posizionati nella zona centrale, 8 per lato.

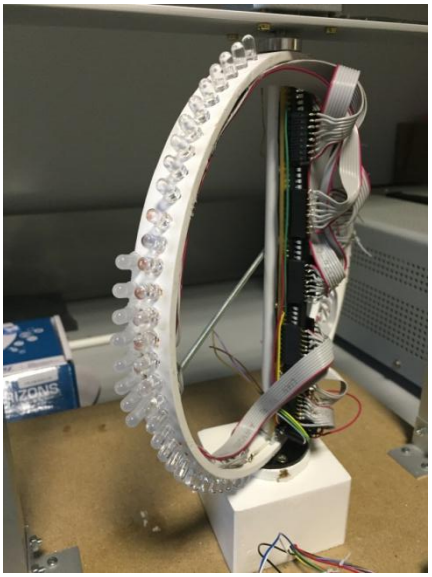


Figura 2. Disposizione dei led

I *leds* vengono pilotati dalla *board* tramite 5 *shift registers* da 16 bit (MAX6969, http://www.mouser.com/ds/2/256/MAX6969-90903.pdf), 4 per i led bianchi, 1 per i led rossi.

2.3 SLAVE 2: *Motor*

Il motore scelto è 42BYGHW811 NEMA-17 Bipolar 48mm Stepper (http://www.phidgets.com/documentation/Phidgets/3302_0_Datasheet.pdf), un motore passo-passo

Figura 3. Motor Stepper NEMA-17

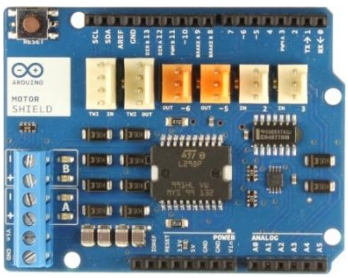
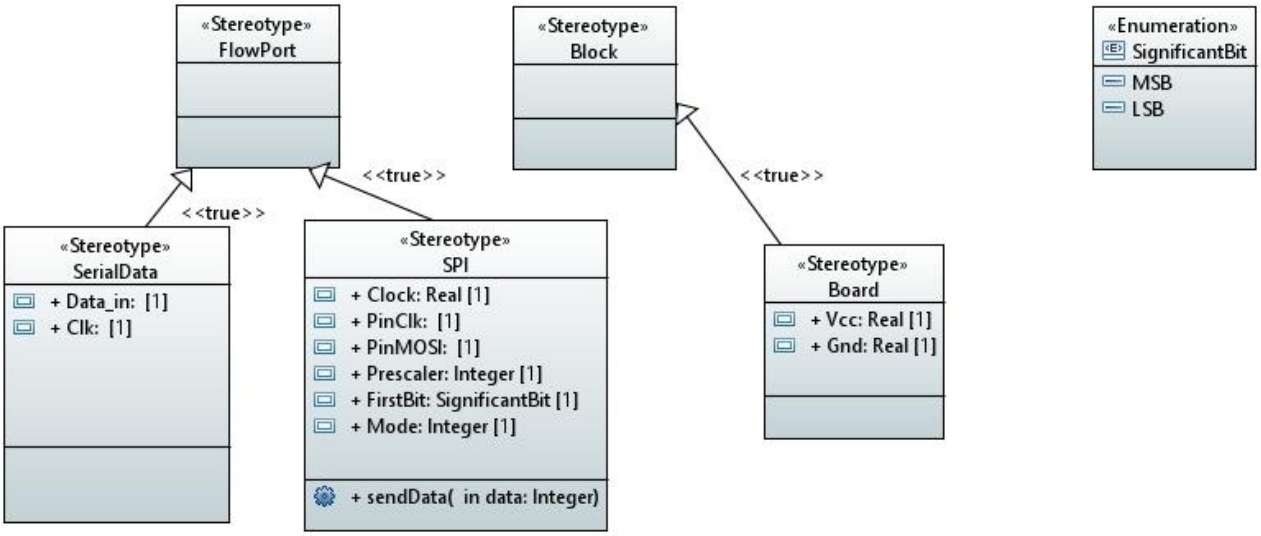


Figura 4. Arduino Motor Shield

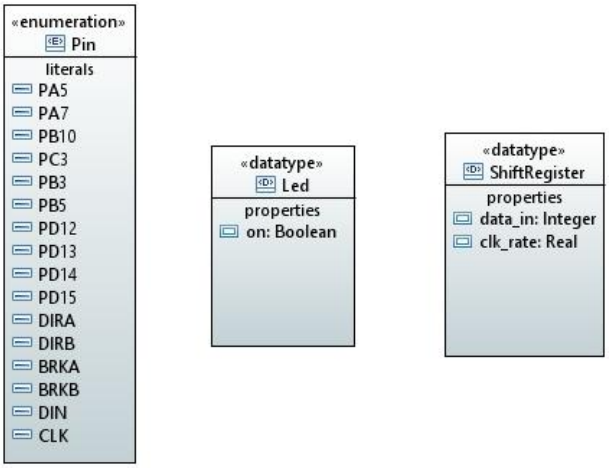
con un angolo di 1.8° e tensione di alimentazione 12V. Il dispositivo che si frappone tra la *board* e il motore è un *Arduino Motor Shield*, che monta un driver L298P (https://www.arduino.cc/en/Main/ArduinoMotorShieldR3). Esso presenta 4 pin in ingresso, tensione di alimentazione per il motore e 4 pin in uscita per il motore passo-passo.

2.4 Modello *SysML*

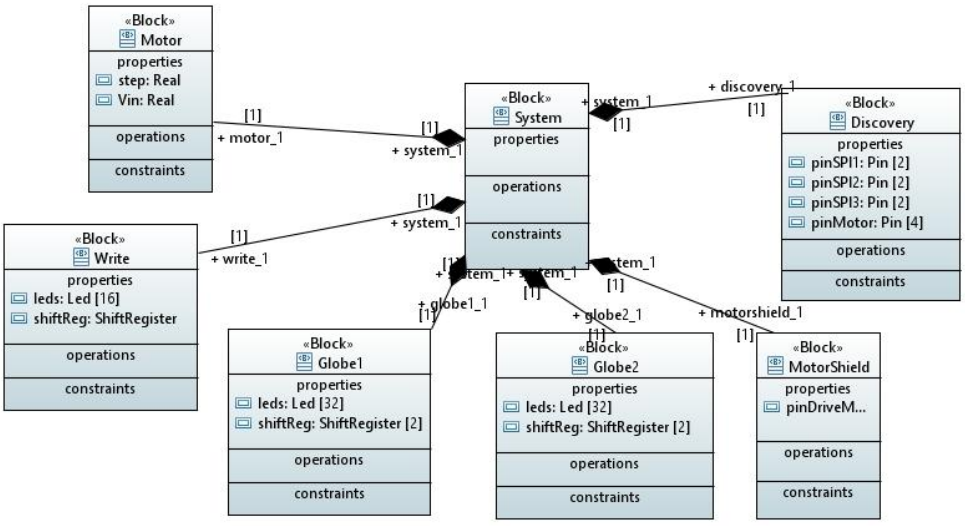
- *Profile Diagram*, applicato al diagramma *SysML*:



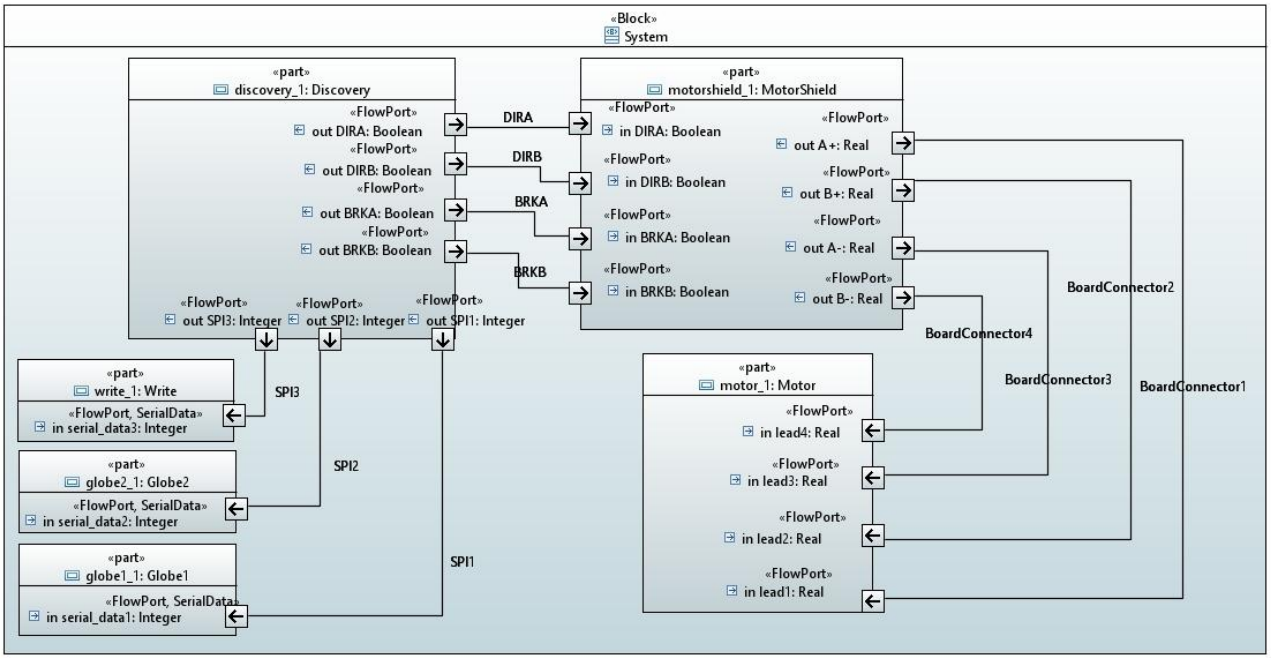
- *Type Block Definition Diagram*, dove sono definiti i tipi utilizzati nel modello:



- *System Block Definition Diagram*, che descrive le componenti del sistema:



- *System Internal Block Diagram*, che descrive le relazioni tra i vari componenti:



2.5 Cablaggio

- Pilotaggio SPI:

  La *Discovery* dialoga con gli *shift registers* tramite 3 porte SPI (SPI1, SPI2, SPI3). Ogni SPI è composta da 3 pin, CLK, MOSI e MISO. Poiché la comunicazione è unilaterale, il pin MISO non viene collegato. Nella tabella seguente, sono mappati i pin della *board* in base alle funzioni delle relative SPI.

	CLOCK	MOSI
SPI1	PA5	PA7
SPI2	PB10	PC3
SPI3	PB3	PB5

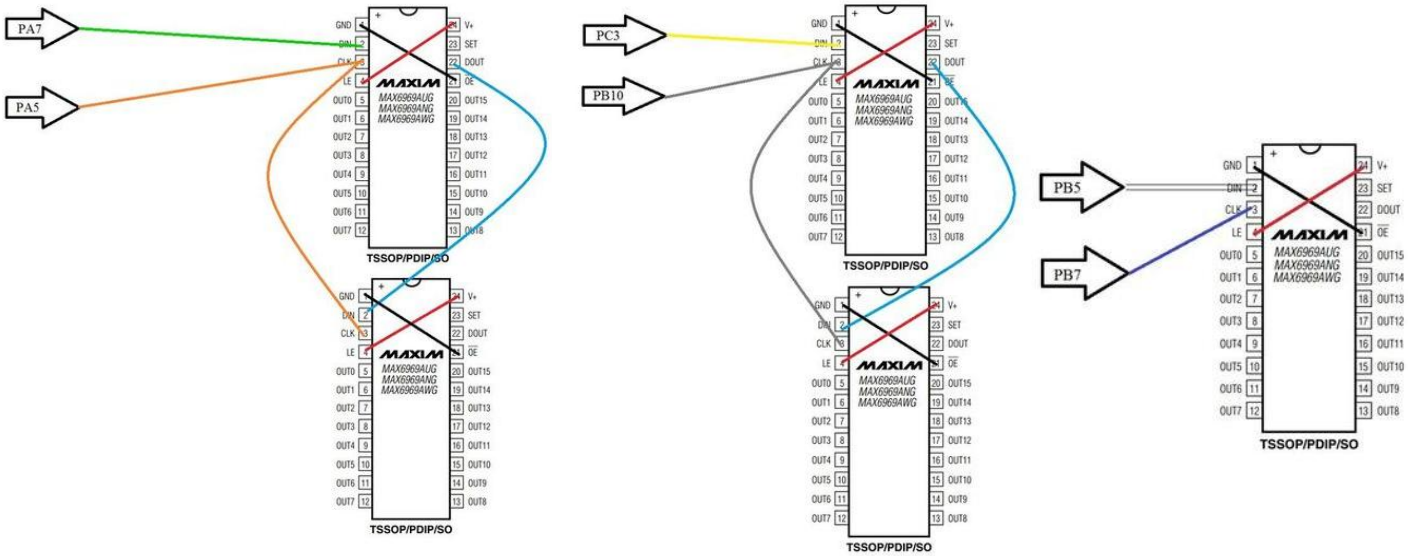


Figura 5. Cablaggio pin SPI e *shift registers*

L'SPI1 e l'SPI2 comandano, ognuna, due *shift registers* in cascata e ognuna è responsabile del pilotaggio di una semicirconferenza di led. Il quinto *shift register*, invece, è pilotato dall'SPI3 e gestisce i 16 led rossi, 8 per lato.

- Pilotaggio motore:

  La *Discovery*, inoltre, utilizza 4 pin del GPIO, per pilotare il motore. Vengono, infatti, messi a 0 o a 1 i *bits*, a seconda della giusta combinazione di valori di DIRA, DIRB, BRKA e BRKB del *Motor Shield*, collegati rispettivamente a PD13, PD15, PD14 e PD12. A seconda dei valori ricevuti in ingresso, esso produce 4 valori in uscita, che comandano, appunto, il motore e gli permettono di ruotare l'asse di 1.8°, ossia di compiere un passo.
  Il motore è alimentato a una tensione di 12V, mentre il circuito logico del *Motor Shield* a 3.3V. Per avere sempre potenza massima sul motore, il controllo del PWM è stato bloccato a Vcc.

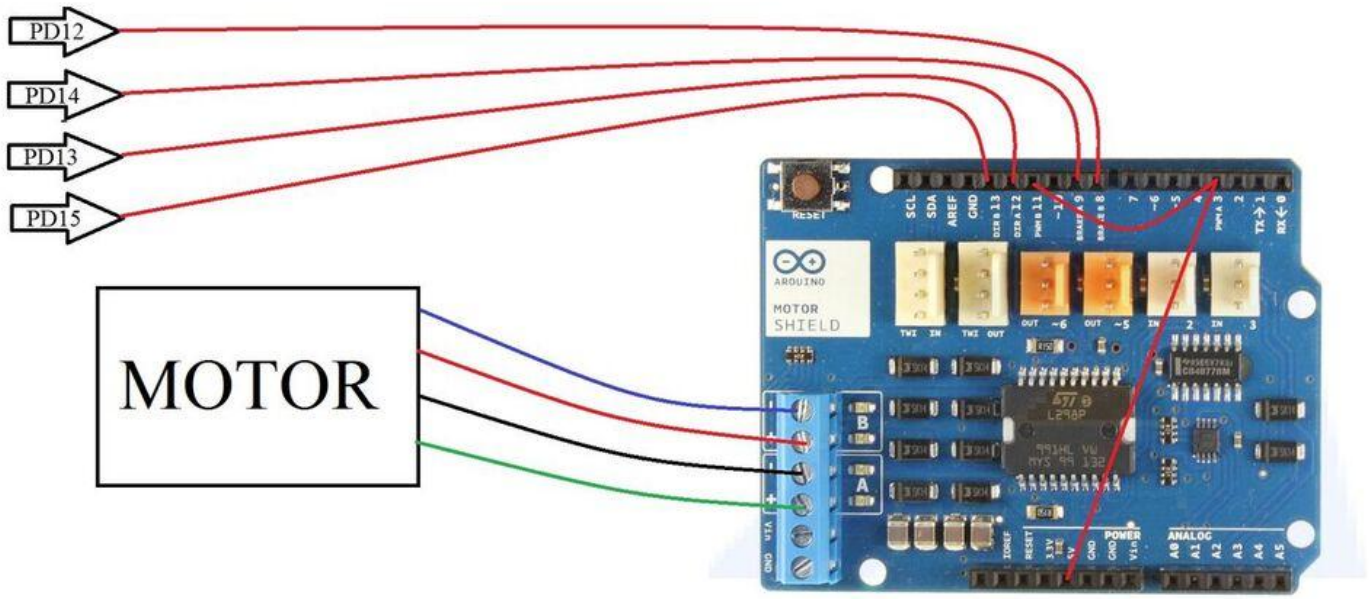


Figura 6. Cablaggio pin GPIO, *Motor Shield* e motore

3. Implementazione

3.1 *Main*

Nel *Main* è gestita l'inizializzazione delle strutture e dei parametri, in particolare vengono inizializzate le periferiche GPIO e SPI. Il periodo del *tick*, inoltre, è settato a 10 µs, per garantire al task che pilota le periferiche un periodo di attivazione dell'ordine dei microsecondi, condizione necessaria per far girare il motore ad una velocità appropriata.

```
EE_systick_set_period(MICROSECONDS_TO_TICKS(1, SystemCoreClock)*10);
```

Sperimentalmente è stato testato che il fattore moltiplicativo 10 è necessario per la corretta gestione delle interruzioni.

Per la gestione delle 3 SPI, si è fatto uso delle apposite librerie. Le SPI scelte sono SPI1, SPI2 e SPI3. La prima lavora ad una frequenza di 90MHz, mentre per le altre due la frequenza è di 45MHz. Poiché il periodo minimo di lavoro degli *shift registers* è di 40ns (25MHz), nell'inizializzazione viene modificata la frequenza di lavoro delle SPI, tramite l'utilizzo di un *prescaler*, uguale a 4 per l'SPI1, 2 per le altre.

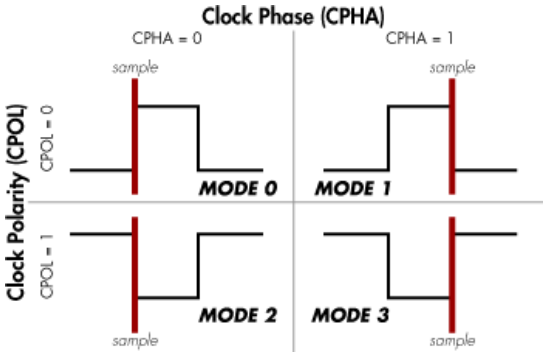


Figura 7. SPI *modes*

La modalità di pilotaggio scelta è la 0, in accordo con le specifiche degli *shift registers*. L'ordine di trasmissione dei bit, infine, è dal meno al più significativo (LSB, *Least Significant Bit*).

```
TM_SPI_InitFull(SPI1, TM_SPI_PinsPack_1, SPI_BaudRatePrescaler_4, TM_SPI_Mode_0,
                SPI_Mode_Master, SPI_FirstBit_LSB);
TM_SPI_InitFull(SPI2, TM_SPI_PinsPack_1, SPI_BaudRatePrescaler_2, TM_SPI_Mode_0,
                SPI_Mode_Master, SPI_FirstBit_LSB);
TM_SPI_InitFull(SPI3, TM_SPI_PinsPack_1, SPI_BaudRatePrescaler_2, TM_SPI_Mode_0,
                SPI_Mode_Master, SPI_FirstBit_LSB);
```

Nel *Main* viene, inoltre, settato il periodo del task che si occupa del pilotaggio del motore e dei led. Il periodo minimo per compiere un passo (1.8°) e avere una risoluzione ottimale è 650µs. In questo modo, il motore compie 461.54 giri/minuto (sono 200 i passi per ogni giro) e la frequenza di *refresh* dell'immagine è 15Hz, poiché avendo due strisce di led esso avviene ogni 100 passi. Il problema di far andare il motore a questa velocità è la partenza; esso, infatti, non riesce a partire da solo, ma ha bisogno di un aiuto umano. Aumentando il periodo fino a 850µs, l'effetto ottenuto rimane buono, nonostante l'immagine sia ricaricata completamente con una frequenza di circa 11.8Hz e il motore giri a una velocità di 352.94 giri/minuto.

3.2 *Task Globe*

Il *task Globe* si occupa del pilotaggio del motore e dei led. Per prima cosa spegne tutti i led, per evitare l'effetto scia. Successivamente, fa eseguire il passo al motore, pilotandolo a seconda dei valori contenuti nell'array *pin*. Infine, accende i led in base alla configurazione contenuta negli array *globe1*, *globe2* e *we* (o *wf*), che rappresentano rispettivamente l'emisfero boreale, australe e la scritta rossa su sfondo bianco (o bianca su sfondo rosso).

Il tempo che viene impiegato per caricare il dato su 32 led è 1.28µs. Il periodo del task *Globe* scelto è, quindi, consono al tempo computazionale del task stesso.

3.3 Immagini

Le immagini relative al globo terrestre sono state realizzate discretizzando un planisfero, in 200 colonne e 32 righe. Il risultato è un array di interi da 16 bit per ogni emisfero, in cui ogni elemento indica lo stato di un striscia di led. In formato binario, ogni bit rappresenta lo stato di ogni led (1 acceso, 0 spento).

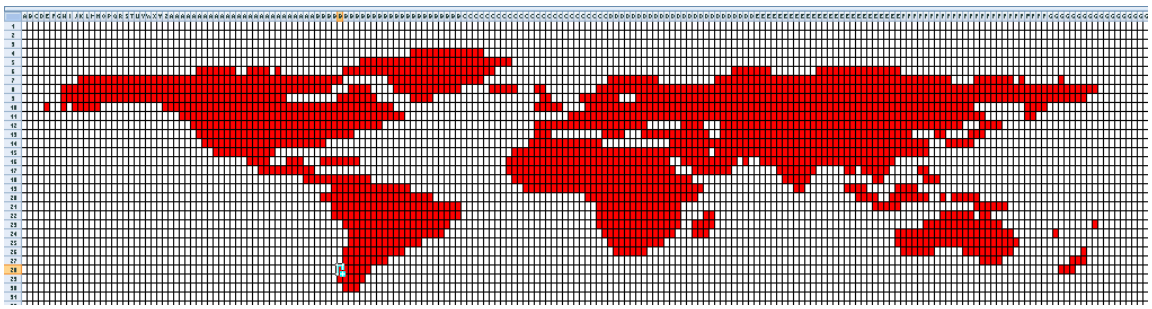


Figura 8. Globo terrestre discretizzato

La scritta è stata ottenuta nel medesimo modo, con la differenza che le righe sono 16.

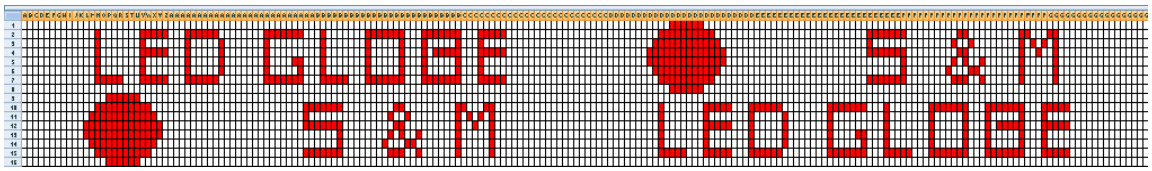


Figura 9. Scritta discretizzata

4. Test

La fase di testing si è svolta con l'ausilio delle librerie *CUnit*.

Le funzioni da testare (nel file "globe.c") sono tre:

- *int motor(int j)*: dato l'indice *j*, restituisce un intero che rappresenta il valore decimale dei 4 bit contenuti nell'array *pin* a partire da *j* (quello di indice *j* è il bit meno significativo);
- *int led_globe(int i)*: dato l'indice *i*, restituisce la somma tra l'elemento i-esimo di *globe1* e di *globe2*;
- *int led_write(int i)*: dato l'indice *i*, restituisce l'elemento i-esimo dell'array *w*, corrispondente all'array *we*.

È stata utilizzata una singola *suite* con due test:

- *void testMotor(void)*, che controlla che, per ogni *j* compreso tra 0 e 15, la funzione *motor()* restituisca il valore aspettato;
- *void testLed(void)*, che testa le due funzioni relative agli array di led (*led_globe()* e *led_write()*), controllando che il risultato ottenuto sia quello desiderato, per ogni elemento degli array.

L'interfaccia utilizzata è *Basic*, che produce il seguente risultato:

```
marco@Diciotto:~/Scrivania/CUnit-2.1-0/globe$ make
gcc -c progs//test_globe.c -Iinc/ -o obj/test_globe.o
gcc -c testedCode//globe.c -Iinc/ -o obj/globe.o
gcc --static obj/test_globe.o obj/globe.o -L /usr/local/lib/ -lcunit  -o test_globe
marco@Diciotto:~/Scrivania/CUnit-2.1-0/globe$ ./test_globe


     CUnit - A Unit testing framework for C - Version 2.1-0
     http://cunit.sourceforge.net/


Initializing suite1

Suite: Suite_1
  Test: test of motor(int) ... passed
  Test: test of led(int) ... passed
Completing suite1


--Run Summary: Type      Total     Ran  Passed  Failed
               suites        1       1     n/a       0
               tests         2       2       2       0
               asserts     616     616     616       0
```

Figura 10. Risultati dell'esecuzione del test con interfaccia *Basic*

È anche possibile eseguire il test con l'interfaccia *Automated*, decommentando le righe di codice 123 e 124 del file "test_globe.c".