

Timer programmabile per esposizioni su materiale fotosensibile in camera oscura

Gabriele Esposito, 964431

November 9, 2021

Contents

1 INTRODUZIONE	1
2 Architettura	2
2.1 Hardware	2
2.2 Software	4
2.2.1 main	5
2.2.2 Timer.h	5
2.2.3 Keypad.h	9
2.2.4 7seg.h/.cpp	10
3 Valutazione della performance	10
4 Consumi	10

1. INTRODUZIONE

La stampa di pellicole negative su supporti fotosensibili e' tradizionalmente realizzata utilizzando un **ingranditore fotografico**. Questo strumento emette luce mediante una lampada, che passando attraverso il negativo e la lente dell'ingranditore imprime l'immagine positiva sul supporto. L'ingranditore e' generalmente dotato di un interruttore per accendere la lampada che viene operato a mano, in modo da gestire l'esposizione del fotogramma. Più il supporto viene esposto alla luce più questo dopo i processi chimici di rivelazione e fissaggio diviene scuro nelle aree "trasparenti" del negativo, viene generata così un'immagine positiva.

Il tempo di esposizione del fotogramma varia a seconda del tipo di pellicola negativa utilizzata e delle preferenze dell'utilizzatore, questo viene normalmente contato utilizzando un orologio dall'operatore oppure viene gestito in maniera più precisa da un **timer programmabile** collegato all'ingranditore.

Sono presenti molti timer per la camera oscura sul mercato. La maggior parte di questi, ad eccezione di quelli industriali integrati negli ingranditori, sono vecchi dispositivi programmabili utilizzando una manopola e privi di schermo, quindi scomodi da utilizzare in condizioni di scarsa illuminazione. Oppure dotati di pochi tasti che devono essere premuti ripetutamente per

indicare il tempo desiderato. I piu' sofisticati possono arrivare a costare anche qualche centinaio di euro.

Questo progetto si propone di realizzare un timer programmabile economico e facile da usare, collegabile ad un ingranditore mediante una presa di corrente, migliorando la facilita' di lettura e di inserimento del tempo, oltre ad aggiungere alcune *funzioni* utili per la stampa in camera oscura.

2. ARCHITETTURA

2.1 Hardware

Il timer e' realizzato utilizzando un microcontrollore STM32F407-DISCOVERY ed e' dotato di un **tastierino alfanumerico** e di un **display** a 7 segmenti per l'interazione con l'utente. Quest'ultimo puo' inserire tramite il tastierino il periodo di tempo durante il quale il carico dovrà rimanere acceso, che verrà visualizzato sul display. Una volta confermato il tempo da parte dell'utente il timer si avvia, viene chiuso il circuito e la corrente passa attraverso il carico fino allo scadere del tempo, al termine del quale il circuito viene riaperto. Il timer è pensato per esposizioni su *materiale fotografico fotosensibile*, i led del display a 7 segmenti emettono quindi luce ad una lunghezza d'onda alla quale i supporti non sono sensibili ($> 550\text{nm}$).

Le componenti sono qui elencate:

- I. Microcontrollore STM32F407-DISCOVERY
- II. Display 7 segmenti 4 cifre (led rosso).
- III. Scheda Relè (5V DC fino a 240V AC con optocoupler).
- IV. Schede millefori.
- V. Tastierino alfanumerico (matrice 4x4). 10 numeri, 2 simboli e 4 lettere
- VI. 2 Lastre plexiglass (Case).
- VII. 1 Fogli acetato trasparente rosso (Case).
- VIII. 2 Contenitori batterie AA 1.5 V LR6.
- IX. Batterie AA 1.5V LR6.
- X. 1 Buzzer attivo.
- XI. Resistori.
- XII. Viti con dadi (Case).
- XIII. Morsetti per cavi elettrici.
- XIV. Jumper dupont.
- XV. Prolunga.
- XVI. Header femmina.

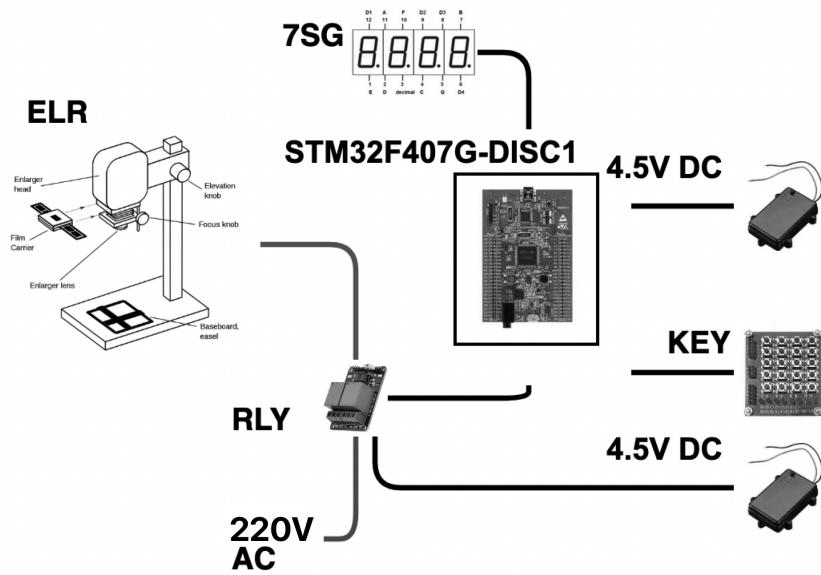


Figure 1: Schema ad alto livello delle componenti

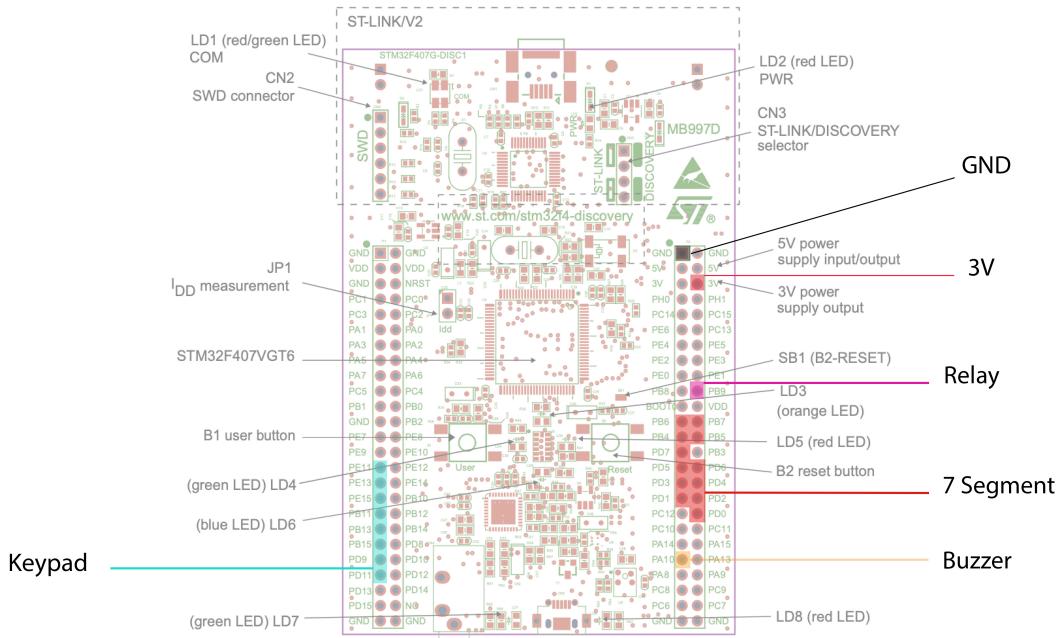


Figure 2: GPIO

La figura 1 mostra l'architettura del dispositivo ad alto livello. Le componenti hardware sono saldate o - nel caso del microcontrollore - montate utilizzando degli header femmina su una scheda millefori, in seguito chiamata solo *scheda*. I GPIO utilizzati sono visibili in figura 2.

La tastiera e il rele' si interfacciano con il microcontrollore utilizzando dei dupont collegati a degli header maschio posizionati sulla scheda. Il display e' saldato direttamente. Tutte le connessioni sulla scheda sono realizzate con cavetti elettrici o direttamente a stagno.

Il dispositivo e' alimentato da due set da 3 batterie 1.5V LR6, un gruppo alimenta il microcontrollore attraverso il pin 3V e l'altro gruppo alimenta il rele'.

Ciascuno dei due contenitori per le batterie presenta un interruttore che puo' essere utilizzato per staccare l'alimentazione quando il dispositivo non viene utilizzato. Si e' optato per una fonte di alimentazione integrata per ridurre il piu' possibile il numero di prese/cavi utilizzati oltre quelli necessari per l'ingranditore.

L'ingranditore viene collegato al dispositivo utilizzando una prolunga i cui terminali sono connessi al rele'. Il rele' e' dotato di un optoaccoppiatore che isola il circuito ad alta tensione da quello a bassa tensione del microcontrollore, inoltre dispone di un alimentazione separata. Quindi il dispositivo e' composto da un gruppo principale formato dalla scheda contenente schermo e MCU STM32F407. Le restanti componenti si interfacciano con la scheda tramite jumper dupont (rele', batterie, keypad). Entrambi i gruppi sono contenuti in un case realizzato in plexiglass visibile in figura 3.

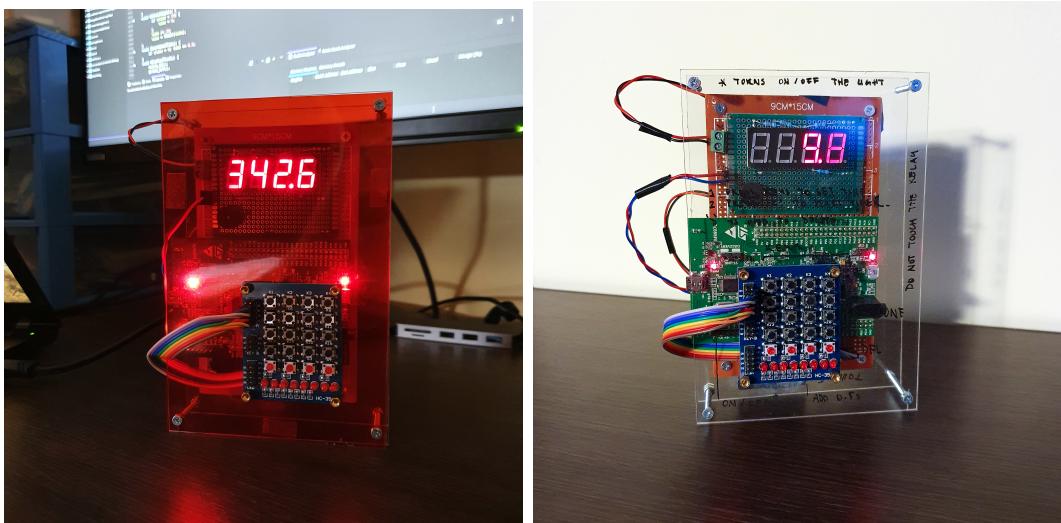


Figure 3: Timer programmabile

2.2 Software

Il software e' strutturato in 5 files:

- I. `main.cpp`
- II. `Timer.h`
- III. `Keypad.h`
- IV. `7seg.h`

V. 7seg.cpp

Ad alto livello l'applicazione utilizza una **ISR** per ricevere input dall'esterno utilizzando il *keypad*, la **ISR** modifica lo stato di un oggetto **Timer** che implementa la logica dell'applicazione e scrive su *display* e *rele*'.

2.2.1 main

Il file main contiene la **configurazione** e l'**inizializzazione** delle periferiche (due *Basic Timer*: TIM6, TIM7), e dei GPIO. Sono definite nel main anche le **callback** collegate agli *Update Event* dei timer hardware. E la callback degli interrupt provenienti dai GPIO collegati al keypad. Assieme ad una funzione **stopMode()** che mette il microcontrollore in **STOP** (sec 4). Tutte le funzioni di callback definite nel main chiamano le funzioni **statiche** dell'oggetto Timer (sec 2.2.2).

```
void HAL_GPIO_EXTI_Callback(uint16_t GPIO_Pin) {
    timer::getIn(GPIO_Pin);
}

void HAL_TIM_PeriodElapsedCallback(TIM_HandleTypeDef *htim) {
    if (htim == &htim7) {
        if(timer::inactivityTimeUpdate()) {
            stopMode();
        }
    } else if (htim == &htim6) {
        timer::updateTime();
    }
}
```

Il **while** loop chiama la funzione statica **run()** dell'oggetto Timer.

```
typedef Timer<&htim6, &htim7> timer;
int main(void) {
    while(1) {
        timer::run();
    }
}
```

2.2.2 Timer.h

L'intera logica applicativa e' definita nella classe **Timer.h**, questa e' una classe **singleton**, **templatata** che ha lo scopo di definire il comportamento dell'oggetto Timer. In altre parole la classe Timer funge da *namespace* per i dati e le funzioni dell' applicazione.

Il pattern singleton e' comodo in quanto non deve esistere nell'applicazione piu' di un oggetto

Timer contemporaneamente, ed il template permette di creare l'oggetto utilizzando degli argomenti quando viene definito, impossibile altrimenti a causa dell'assenza del costruttore. L'oggetto Timer fa uso di due periferiche hardware: i due *Basic Timer* **TIM6** e **TIM7**.

Il primo viene utilizzato per aggiornare il tempo visualizzato sul display ogni millisecondo una volta avviato il conto alla rovescia.

Il secondo misura il tempo di inattività del device, se questo non viene resettato da un'azione dell'utente (usando il keypad) al raggiungimento del 15 secondo il microcontrollore va in **Stop-Mode**. Gli indirizzi dei due **TIM_HandleTypeDef** vengono passati come *template arguments* alla classe Timer in modo da poter essere utilizzati dall'oggetto.

```
template <TIM_HandleTypeDef * htim, TIM_HandleTypeDef * htim_sleep>
class Timer {
    //class definition
};
```

Tutte le funzioni della classe Timer sono *statiche*. Restituiscono l'istanza dell'oggetto singleton, e su quella chiamano la **funzione privata** di implementazione richiesta. Ad esempio la funzione statica **run** è così definita:

```
static void run() {
    getTimer().runImpl();
}
```

La funzione **getTimer()** restituisce l'istanza dell'oggetto timer. Nello stesso tempo si assicura che esista una sola versione dell'oggetto, che viene infatti dichiarato **static** dentro questa funzione.

```
static Timer& getTimer() {
    static Timer timer_instance;
    return timer_instance;
}
```

Mentre la funzione privata **runImpl()** contiene l'implementazione della funzione **run()**. Così facendo è possibile utilizzare direttamente le funzioni statiche della classe senza ogni volta farsi restituire l'oggetto dalla funzione **getTimer()**.

Di seguito sono descritte le funzioni principali della classe timer, queste sono le funzioni di interfaccia che vengono utilizzate nel **main**.

Il flusso di esecuzione principale è contenuto nella funzione **runImpl()** dell'oggetto Timer (fig 4). Questa consiste essenzialmente in due fasi. La prima rappresentata dalla funzione **execute()**, aggiorna lo stato del timer sulla base degli input ricevuti, e la seconda rappresentata dalle funzioni **displayTime()** e **displayWait()** è incaricata della visualizzazione dell'input numerico dell'utente (se presente).

Timer Run Loop

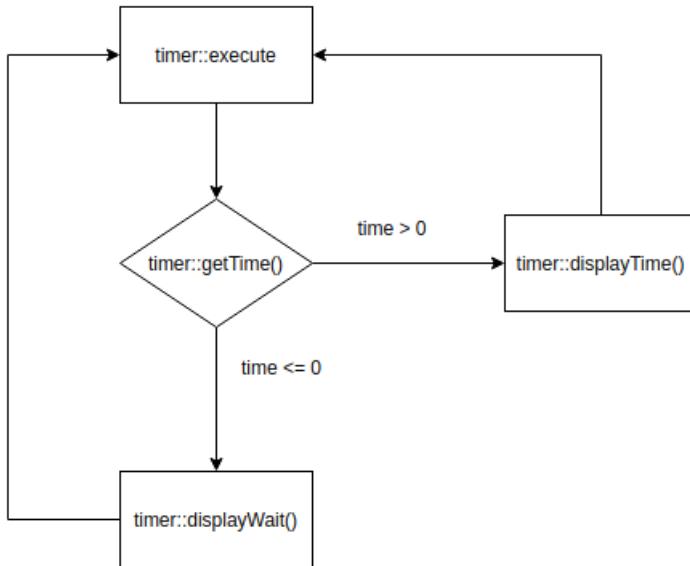


Figure 4: flowchart loop principale

Execute execute() effettua un controllo dello stato interno del timer, se e' presente un input aggiorna lo stato dell'oggetto ed esegue la funzione associata al tasto premuto dall'utente.

L'input si divide essenzialmente in due tipologie: *tasto funzione* o *numero*. Se l'input e' un numero viene aggiornato il valore della variabile **time**. (La variabile **time** rappresenta il tempo durante il quale la lampada dell'ingranditore deve rimanere accesa.)

Se l'input corrisponde ad un *tasto funzione* viene eseguita la funzione associata.

Le funzioni sono le seguenti:

BREAK

Al tasto **B** del keypad visibile in figura 5 e' associata la funzione di **reset** del device, questa funzione riporta lo stato del timer a quello iniziale di default, se un'esposizione e' in corso questa viene interrotta. Il valore della variabile **time** viene riportato a 0.

CONFIRM

Il tasto **C** e' un tasto multifunzione, se e' stato inserito un input numerico precedentemente questo avvia il timer dell'esposizione. Chiude il circuito accendendo la lampada dell'ingranditore per il tempo che e' stato inserito, al termine del lasso di tempo indicato riapre il circuito. Se il tasto **C** viene premuto in assenza di input questo *ripete* il tempo inserito per l'esposizione precedente. Il tasto deve essere premuto nuovamente per confermare ed avviare la nuova esposizione.

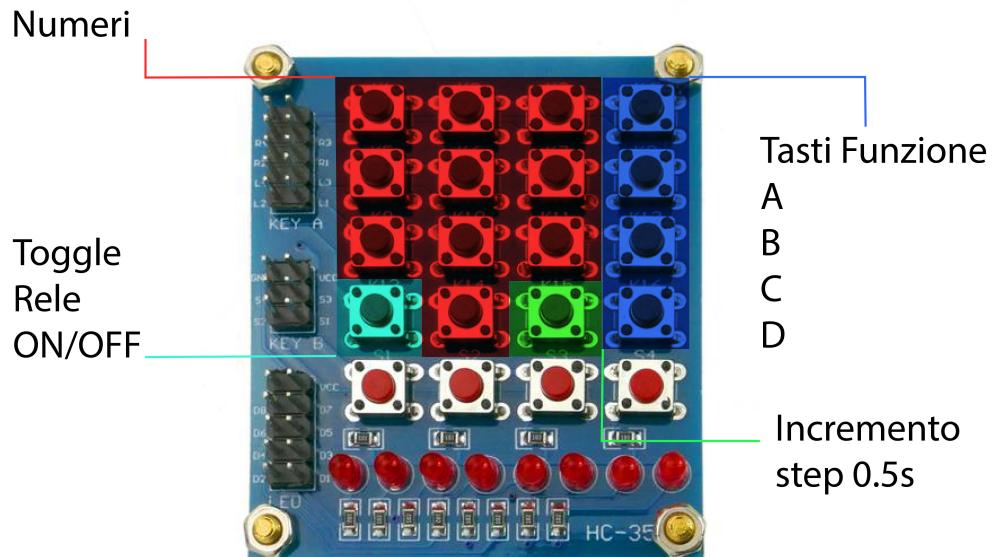


Figure 5: Schema input Keypad

DELETE

Il tasto **D** del keypad ha la funzione di *eliminare* l'ultima cifra inserita, se e' stata inserita una sola cifra il tempo torna a zero.

INCREMENT

Il tasto **incremento** del keypad ha la funzione di incrementare il tempo inserito di $0.5s$, se non e' presente nessun input il comando viene ignorato, non e' necessario infatti effettuare esposizioni piu' brevi di $1s$.

TOGGLE

Il tasto funzione **toggle** accende/spegne la lampada dell'ingranditore per un tempo indefinito. Questo e' necessario per effettuare il posizionamento del frame da impressionare sul supporto fotosensibile.

DisplayTime Nella seconda fase di `runImpl()` la funzione `displayTime()` scrive il valore corrente della variabile `time` sul display, se questo e' diverso da zero. In caso contrario la funzione `displayWait()` visualizza dei *dashes* (fig 6).

Lo stato dell'oggetto puo' essere aggiornato da degli **eventi asincroni** come l'input da keypad o lo scadere di un *Basic Timer*:

- I. Interrupt keypad.

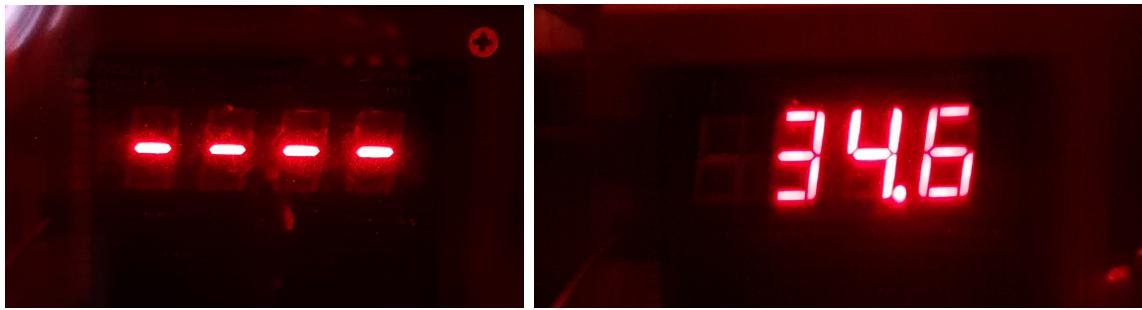


Figure 6: Input visualizzato su display 7 segmenti

II. Interrupt TIM6 (Tempo di esposizione)

III. Interrupt TIM7 (Tempo di inattività')

Questi eventi asincroni sono gestiti dalle funzioni dello stesso oggetto Timer e sono analizzate nei paragrafi che seguono.

IgetIn La funzione di callback per l'interrupt da keypad (`HAL_GPIO_EXTI_Callback()`) fa uso della funzione `IgetIn()`. Questa funzione decodifica l'input da keypad per restituire un carattere corrispondente alla combinazione riga/colonna del tasto premuto. La funzione `execute` eseguirà la richiesta dell'utente associata al carattere decodificato.

IupdateTime Il *Basic Timer* TIM6 è configurato per generare un *Update Event* ogni millisecondo. La funzione di callback `HAL_TIM_PeriodElapsedCallback()` - se si tratta del TIM6 ad aver generato l'*Update Event* - fa uso della funzione `IupdateTime()`. Questa funzione aggiorna lo stato della variabile `time` membro dell'oggetto timer e al raggiungimento dello zero ferma TIM6 e stacca il rele' (il tempo di esposizione è scaduto). La variabile `time` viene aggiornata ogni millisecondo per visualizzare lo scorrimento del tempo in maniera fine sul display.

IinactivityTimeUpdate() Il *Basic Timer* TIM7 è configurato per generare un *Update Event* ogni secondo. La funzione di callback `HAL_TIM_PeriodElapsedCallback()` - se si tratta del TIM7 ad aver generato l'*Update Event* - fa uso della funzione `IinactivityTimeUpdate()`. Questa aggiorna lo stato della variabile `inactivity_t` dell'oggetto Timer. Se `inactivity_t` raggiunge il valore 15 (dopo 15 secondi) il microcontrollore va in **StopMode**. Ogni qualvolta che viene premuto un tasto il valore della variabile `inactivity_t` viene resettato e torna ad essere zero, in modo che siano necessari almeno 15 secondi di inattività per passare alla stop mode.

2.2.3 Keypad.h

Il file `keypad.h` contiene delle definizioni utili alla configurazione del keypad, come i GPIO utilizzati per la connessione al microcontrollore ed una matrice contenente i caratteri corrispondenti a ciascuno dei tasti.

2.2.4 7seg.h/.cpp

I due file `7seg.h` e `7seg.cpp` contengono delle definizioni utili per la configurazione del display e il suo funzionamento, come i GPIO utilizzati per la connessione al microcontrollore e le funzioni che disegnano numeri e lettere sul display.

3. VALUTAZIONE DELLA PERFORMANCE

Il dispositivo non ha particolari requisiti di performance eccetto i tempi di risposta agli input dell'utente ed i tempi di visualizzazione degli output sul display. Di seguito vengono riportati i tempi di esecuzione delle funzioni dell'interfaccia utilizzate nel main.

Run La funzione `run()` (come descritto nella sezione [2.2.2](#)) costituisce il flusso principale dell'applicazione. Il tempo impiegato da questa funzione in ogni esecuzione del ciclo `while` nel `main` riflette il tempo impiegato dal dispositivo ad elaborare gli input dell'utente ed aggiornare lo stato dell'applicazione.

La funzione ha un tempo di esecuzione di circa $12ms$. Questo significa che la funzione `run()` e' in grado di scrivere sul display circa 83 volte in un secondo. Una frequenza di aggiornamento piu' che sufficiente per evitare lo sfarfallio del display.

getIn Di maggiore importanza e' invece il tempo impiegato dalla funzione `getIn()` di decodifica dell'input dal tastierino quando viene generato un interrupt. Infatti interrompendo la funzione `run()` questo potrebbe causare un freeze del display se l'esecuzione non fosse sufficientemente veloce.

Il tempo misurato in questo caso corrisponde a $130ms$. E' da considerare che $75ms$ dei $130ms$ totali corrispondono al tempo di *debouncing* dei bottoni del tastierino. (La classe Timer dispone di una funzione privata `debounce` per evitare che input multipli vengano rilevati con una sola pressione di un bottone). Anche in questo caso il tempo di risposta e' sufficientemente rapido considerato lo scopo del dispositivo.

Callback Le due funzioni `inactivityTimeUpdate()` e `updateTime()` nella callback `HAL_TIM_PeriodElapsedCallback()` hanno entrambe un tempo di esecuzione di meno di $1ms$. Questo tempo di latenza e' ugualmente di particolare importanza perche' potrebbe portare ad uno stato delle variabili dell'oggetto timer non corrispondente al tempo effettivamente trascorso.

4. CONSUMI

Per ridurre i consumi al minimo il dispositivo fa uso delle modalita' risparmio energetico fornite dal microcontrollore. Il dispositivo si avvia in modalita' di default **Run**, in questa modalita' viene utilizzata la funzione di voltage scaling `scale2`. Vengono cosi' ridotti i consumi riducendo la frequenza massima del core.

Dopo 15 secondi di inattività il microcontrollore passa nella modalita' **Stop** nella quale i consumi sono notevolmente ridotti. Nella modalita' **Stop** tutti i clock del dominio 1.8V sono

Total power = $I_{average} * V$, with:

$$I_{average} = (I_{Run} * Time_{Run} + I_{Standby} * Time_{Standby} + I_{Sleep} * Time_{Sleep} + I_{Stop} * Time_{Stop}) / Total\ time$$

Figure 7: Calcolo potenza dissipata

spenti (Core, Memorie, Periferiche digitali), gli oscillatori HSI e HSE sono spenti, mentre il contenuto dei registri e della SRAM sono mantenuti. E' mantenuto anche lo stato dei GPIO, eccetto quelli del display che viene spento. Il microcontrollore entra nella *low power mode* mediante l'istruzione WFI quindi qualsiasi EXTI-line configurata come interrupt forza l'uscita dalla modalita' a risparmio energetico per ritornare a quella di default (Run).

Nello specifico quando viene gestito un interrupt e il microcontrollore esce dalla **StopMode** la funzione `execute()` dell'oggetto Timer verifica se il timer sta uscendo dalla suddetta modalita' e ignora l'input inserito. In questo modo puo' essere utilizzato qualsiasi bottone del tastierino per risvegliare il dispositivo prima di riprenderne l'utilizzo.

La misurazione dei consumi e' stata fatta utilizzando un amperometro collegato ai due header IDD sul microcontrollore.

$$\text{RUN (scale2)} \quad 27mA \cdot 4.5V = 121.5mW$$

$$\text{STOP} \quad 9mA \cdot 4.5V = 40.5mW$$

Nel caso peggiore il dispositivo viene utilizzato per meta' del tempo in **RunMode** e per la restante meta' in **StopMode**. Questo corrisponde ad un consumo medio di:

$$18mA \cdot 4.5V = 81mW \tag{1}$$

considerando che le batterie utilizzate hanno una capacita' di 1862 mAh queste avrebbero (nel caso peggiore) una durata media di

$$\frac{1862mAh}{18mA} = 103.4h \tag{2}$$

Possiamo considerare un caso d'uso piu' accurato, con un tempo di esposizione totale di circa 8s e un tempo di preparazione del frame da impressionare di 180s. La corrente media corrisponde a:

$$I_{avg} = \frac{27mA \cdot 8s + 9mA \cdot 180s}{188s} \tag{3}$$

con un consumo medio di:

$$Power = I_{avg} \cdot V = 9.76 \cdot 4.5 = 43.92mW \tag{4}$$

In questo caso la durata delle batterie corrisponderebbe a circa 190 ore di utilizzo.