StuFW Documentazione

Autore: Carlo Dormeletti

Versione di riferimento del firmware: **0.0.6**
Versione della guida: **0.01**
Data di stampa: **7 dicembre 2020**

Indice

Prefazione

Questo documento è in fase sperimentale.

Al momento non vuole essere nulla in particolare, solo alcuni spunti per documentare il firmware.

Quindi prendetelo come qualcosa da leggere, senza nessuna garanzia di correttezza.

Principalmente serve per tenere traccia delle scoperte che si fanno sul firmware, mentre lo si modifica o lo si elabora.

Introduzione

Genesi del firmware

Questo firmware è nato per rendere semplice il firmware MK4duo, si è evoluto dopo la "morte" dello stesso e il suo passaggio nel limbo dei progetti abbandonati.

Nel compiere queste affermazioni, rendiamo il dovuto omaggio sia a MK4duo stesso, che al suo autore MagoKimbra Alberto Cotronei.

Nello spirito dell'antico motto: *"Il Re è morto. Viva il Re"* forkiamo e sviluppiamo secondo alcune linee guida, il paradigma che abbiamo in mente è quello di un firmware per studenti delle superiori per definizione squattrinati e con poche risorse, per cui:

- due "schede madri" tutte a 8 bit l'accoppiata ds:amr e la scheda all in one MKS Gen 1.2 - 1.4
- due display tra i più diffusi:
  - RepRapDiscount Full Graphics Controller (grafico a matrice di punti 128x64)
  - RepRapDiscount Smart Controller
- driver per stepper A4988 e DRV8825 diffusi ed economici
- poche funzioni con un occhio ai costi
- cercheremo di rendere semplici le operazioni di taratura e di messa in moto della stampante

La scelta di Mk4duo ci facilita le cose, in quanto è un ottimo firmware e permette di lavorarci sopra, ultima considerazione è stato forkato da Marlin per cui non compiamo nessun peccato di lesa maestà, continuiamo sulla strada dell'OpenSource.

Gli Autori

Capitolo 1

Terminologia

Non possiamo esimerci dal cominciare la spiegazione chiarendo alcune precisazioni terminologiche:

Estrusore La parte meccanica che fa avanzare il filo.

Hot End Abbreviato spesso con **HE**, la parte elettrica che scalda il filamento.

Questa accoppiata a volte è definita semplicemente come ”estrusore”.

Questo è generato dal fatto che nella documentazione originale di RepRap l'estrusore era inteso come l'accoppiata di HE e ”Cold End” cioè l'insieme estrusore e HE che era unito in un unico gruppo elettromeccanico.

Dato che nel corso del tempo, questa parte si è evoluta e non è infrequente trovare le due parti separate, usiamo i due concetti esposti sopra.

Altri termini che necessariamente incontreremo e che saranno dati come acquisiti per cui non spiegati nel corso della trattazione sono:

Hot Bed Abbreviato spesso con **HB** detto più propriamente piano riscaldato (letteralmente è ”letto caldo”)

MOSFET Un tipo di Transistor che riesce a gestire grossi carichi di corrente, in genere è collegato alle uscite di potenza della scheda e serve per accendere e spegnere carichi importanti come corrente. Comunemente è montato sulla scheda male ma non è raro che sia una scheda esterna più adatta a maneggiare grossi cariche come quelli di HB particolarmente grandi.

Stepper Con il significato di ”motore passo passo”, verrà illustrato in dettaglio più avanti, genericamente quando si parla di ”motori” in ambito stampa 3D si intende un motore ”passo passo” se non diversamente indicato.

Driver Cioè il circuito elettronico che controlla gli stepper.

Display Anche LCD (Liquid Crystal Display) il piccolo schermo (se c'è) dove si leggono le informazioni.

Encoder La manopolina che controlla l'avanzamento dei menu sul Display.

Pin Parola inglese che significa spillo, in pratica il ”piedino” (la connessione) del circuito elettronico a cui è collegato qualcosa. Si usa anche per riferirsi anche al piedino

di uscita del chip elettronico, ad esempio il microcontrollore Arduino Mega che è il cuore della stampante.

Capitolo 2

Configurazione

Cercheremo di analizzare in questo capitolo la configurazione del firmware.

Il firmware StuFW è derivato da MK4duo e MK4duo usava uno speciale meccanismo di configurazione.

Sul sito del firmware esisteva un configuratore online che produceva un file zip che conteneva oltre alla versione scelta del firmware un file speciale, **Configuration_Overall.h** dove si trovavano tutte le informazioni necessarie per configurare il firmware, ed in coda ad esso c'era una sezione speciale che permetteva di ricaricare le opzioni simili al cambio di versione.

Configuration_Overall.h è stato mantenuto in StuFW ma non cono lo stesso funzionamento e nemmeno con la sezione speciale in coda al file, è diventato un metodo molto veloce per unire in un unico file tutte le opzioni di configurazione, rendendo comodo trasferire un solo file al posto di molti file.

Pertanto un file **Configuration_Overall.h** prodotto da MK4duo NON DEVE essere usato in sostituzione di un file **Configuration_Overall.h** di StuFW.

Può essere usato al massimo come traccia per ottenere un file **Configuration_Overall.h** funzionante per StuFW.

La configurazione è fatta usando vari file di configurazione nominati nel disorso per comodità **Configuration_xxxxx.h** li elenchiamo:

- **Configuration_Basic.h**
- **Configuration_Mechanics.h**
- **Configuration_Temperature.h**
- **Configuration_Feature.h**

Tutti i settaggi contenuti in questi file possono essere raggruppati in un corretto **Configuration_Overall.h**, creando una sorta di file centralizzato che li raccoglie tutti.

Esistono dunque vari modi per configurare StuFW:

- Creare un file **Configuration_Overall.h** vuoto e modificare le opzioni ne vari file **Configuration_xxxx.h** elencati sopra.
- Creare un corretto **Configuration_Overall.h** che include tutte le informazio-

ni importanti, cioè i vari #define necessari per configurare la propria stampante copiando e incollando le opzioni non commentate presenti in quei file di configurazione.

- Usare un file **Configuration_Overall.h** precompilato o fornito da qualche sviluppatore, alcuni di questi file saranno presenti in futuro nel repository principale e potrebbero essere resi disponibili da qualche sviluppatore di progetti di stampanti 3D.

Sono stati compiuti alcuni sforzi per rendere chiaro il funzionamento delle varie opzioni presenti nei file di configurazione.

Esiste un altro file che non abbiamo citato sopra:

- **Configuration_Pins.h**

In questo file si possono ridefinire i vari pin che sono definiti nelle impostazioni delle varie schede madri, ridefinendo ad esempio il pin per l'HB in modo da usare un MOSFET esterno.

Il file **Configuration_Pins.h** fornito contiene moltissimi valori chiamati ORIG_xxxx che possono essere ridefiniti per fare in modo che un pin sia assegnato correttamente alla funzione desiderata, questi ORIG_xxxx sono definiti appunto nelle varie configurazioni delle schede madri.

2.1 Come configurare il firmware

Descriveremo di seguito come configurare il firmware, cercheremo di spiegare le cose più importanti, rimandando a capitoli di approfondimento le cose più complicate.

Si configurerà il firmware usando il file **Configuration_Overall.h** perché risulterà più comodo da maneggiare essendo primo di commenti e renderà più chiara la spiegazione.

Come accennato prima si è cercato di rendere i singoli file di configurazione abbastanza descrittivi, per cui rimandiamo ai commenti in essi presenti oppure a capitoli specifici per le cose più approfondite.

2.1.1 Configurazione di Base

Presentiamo e spieghiamo le righe del file contenute nella sezione di **Configuration_Overall.h** sotto questa intestazione:

```
/************************
 * Configuration_Basic *
 ************************/
```

Non tutte le righe ci interesseranno, questa sezione è simile se non identica a quella del firmware originale, per cui alcune cose sono state volutamente lasciate e potrebbero sparire in futuro.

Questa sezione del file di configurazione si occupa principalmente di configurare le cose di base, alcune cose hanno una certa importanza.

Comunicazione

Le righe che si occupano della comunicazione sono:

```
#define SERIAL_PORT_1 0
#define BAUDRATE_1 250000
```

La voce **SERIAL_PORT_1** stabilisce la porta seriale di comunicazione, lasciamola al suo valore di default.

La voce **BAUDRATE_1** normalmente va lasciata con quel valore, si potrebbe pensare di aumentarlo a 500000, 1000000 per velocizzare il trasferimento dei file, ma va considerato una miglioria da fare eventualmente in un secondo momento dopo il collaudo e la messa a punto.

Si potrebbe diminuirlo a 115200 se si riscontrassero problemi di comunicazione.

Altro da fare non c'è, la porta seriale n° 2 verrà usata raramente in quanto su **Arduino Mega + RAMPS** i pin sono D17 che è mappato su AUX4 mentre D18 è mappato sull'endstop Z-.

Usarla non è impossibile, però diventa macchinoso perché bisogna spostare molte cose basilari su altri pin.

Identificazione Macchina

```
#define STRING_CONFIG_H_AUTHOR "(none, default config)"
#define MACHINE_UUID "
                ↪ 00000000-0000-0000-0000-000000000000"
```

Si occupano del nome della versione dei firmware e dell'identificazione della macchina tramite UUID, possiamo lasciarli ai valori di default, oppure personalizzarli, servono principalmente negli utilizzi commerciali del firmware dove è necessario identificare diverse macchine con lo stesso nome (una sorta di numero seriale) e quando si compila il firmware per differenziare una configurazione personalizzata dalle altre.

Rimandiamo alle note nel file **Configuration_Basic.h** per ulteriori informazioni.

Tipo di Scheda e Meccanica

```
// see boards.h for the board names
#define MOTHERBOARD BOARD_RAMPS_13_HFB  //BOARD_MKS_13
#define MECHANISM MECH_CARTESIAN
```

Nella Tabella 2.1 nella pagina seguente presentiamo i valori possibili per il valore di **MOTHERBOARD** che sono volutamente limitati a solo due tipi di scheda madre.

La scheda **BOARD_MKS_13** presenta un MOSFET aggiuntivo rispetto ad **Arduino Mega + RAMPS** sul pin **D7** che è assegnato per difetto al secondo HE anche se non presente.

Tabella 2.1: Tipi di scheda madre

Identificativo	Descrizione	Uscite di Potenza			
		D10	D9	D8	D7
BOARD_RAMPS_13_HFB	**Arduino Mega + RAMPS** 1.3 / 1.4	HE0	Fan	Bed	–
BOARD_RAMPS_13_HHB	**Arduino Mega + RAMPS** 1.3 / 1.4	HE0	HE1	Bed	–
BOARD_RAMPS_13_HFF	**Arduino Mega + RAMPS** 1.3 / 1.4	HE0	Fan	Fan	–
BOARD_RAMPS_13_HHF	**Arduino Mega + RAMPS** 1.3 / 1.4	HE0	HE1	Fan	–
BOARD_RAMPS_13_HHH	**Arduino Mega + RAMPS** 1.3 / 1.4	HE0	HE1	HE2	–
BOARD_MKS_13	**Scheda MKS GEN dalla v1.2 alla 1.4**	HE0	Fan	Bed	HE1

Alimentazione

Ci occupiamo delle impostazioni relative al tipo di alimentatore.

In genere ci sentiamo di consigliare un alimentatore cosidetto a ”mattonella” cioè non del tipo da computer, più che altro per motivi di potenza, in genere gli alimentatori da computer hanno la maggior parte della potenza sul ramo a +5V mentre sempre in genere una stampante 3D utilizza la parte a +12V.

```
#define POWER_SUPPLY 0
#define PS_DEFAULT_OFF false
#define DELAY_AFTER_POWER_ON 5
#define POWER_TIMEOUT 30
```

Faremo alcune considerazioni sulla ”taglia” dell’alimentatore a (TODO! Inserire rimando pagina)

Estrusori e HotEnd

```
#define EXTRUDERS 1
#define DRIVER_EXTRUDERS 1
```

La voce **EXTRUDERS** indica il numero di estrusori presenti, in genere la configurazione più semplice è 1

La voce **DRIVER_EXTRUDERS** indica invece il numero di driver presenti che pilotano i vari estrusori, in genere è uguale al valore di **EXTRUDERS**.

Questo valore potrebbe essere diverso in presenza di configurazioni particolari, è stata conservata la possibilità di pilotare fino a 6 estrusori con un sistema che usa due driver e 8 relais (Sistema MKR6), ma la configurazione di questo sistema è abbastanza esotica e non certo indicata al principiante, la citiamo solamente per completezza.

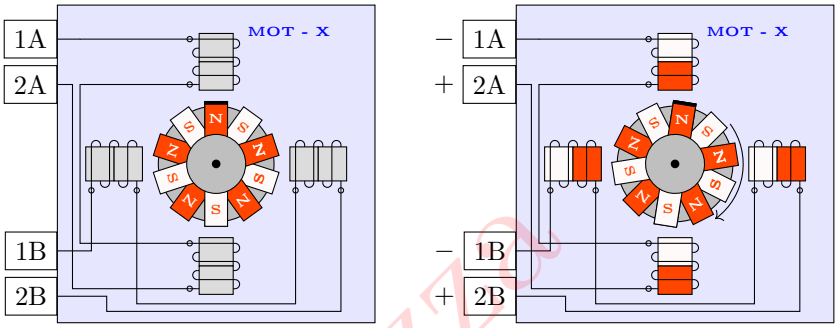
2.1.2 Configurazione di Base

Capitolo 3

Meccanica

3.1 I motori passo passo

Nella figura 3.1 viene rappresentato un motore passo passo.



(a) Schema teorico (b) Motore in half-step

Figura 3.1: Schema di un motore passo passo.

nella figura 3.1a è rappresentato il motore con le fasi non energizzate.

Spieghiamo alcuni particolari del disegno:

- Il motore è rappresentato in modo molto schematico, è un motore a due fasi, quindi con due gruppi di bobine.
- il rotore (cioè la parte che gira collegata all'albero) è rappresentato in modo altrettanto semplice.
- il colore rosso indica un magnete con polarità Nord
- il colore bianco indica un magnete polarità Sud
- Tra le tante possibilità di elencare i nomi dei pin di eccitazione delle bobina, ho usato quello che in genere viene scritto sulla documentazione in RepRap che chiama:
  - 1A e 2A i terminali del gruppo di bobine A, di seguito indicati semplicemente come bobina A.

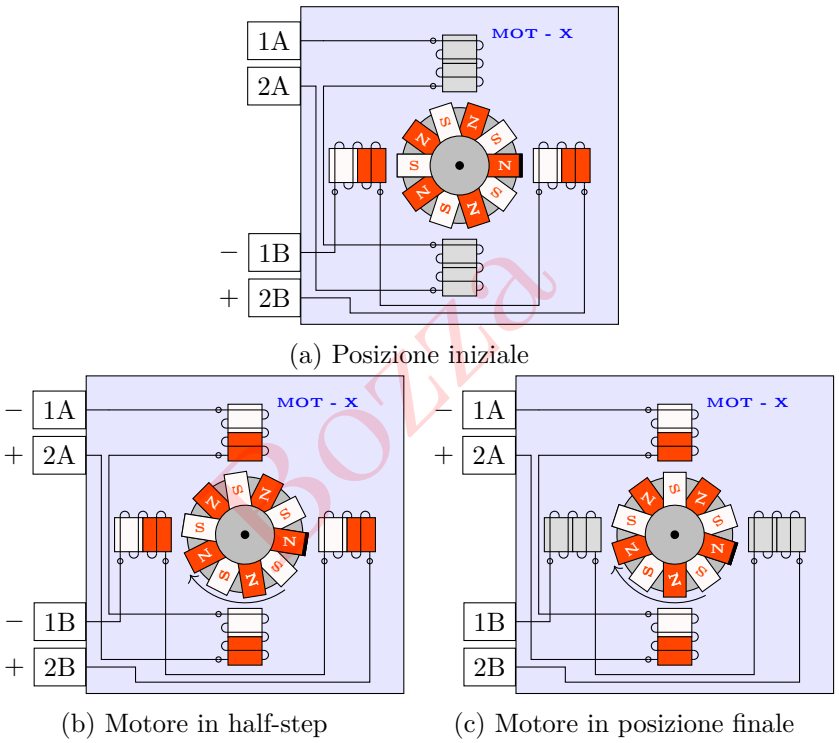
– 1B e 2B i terminali del gruppo di bobine B, di seguito indicati semplicemente come bobina B .

Ha poco senso parlare di terminale positivo e di terminale negativo, perché come vedremo fra poco la tensione può essere data in un senso, ad esempio nella figura 3.1b nella pagina precedente al terminale 1A viene fornito un negativo e quello 2B viene fornito un positivo.

Ricordiamo per inciso che un magnete attira la polarità opposta e respinge la stessa polarità.

Per cui nella figura 3.1b nella pagina precedentevediamo che il rotore sta in una posizione intermedia tra le bobine, perché essendo entrambi polarizzate con la stessa polarità, entrambe attirano il rotore, e lui non sa dove andare.

In somma sintesi, mostriamo nella figura 3.2 l'ordine di eccitazione delle fasi per avere una rotazione oraria.



(a) Posizione iniziale

(b) Motore in half-step

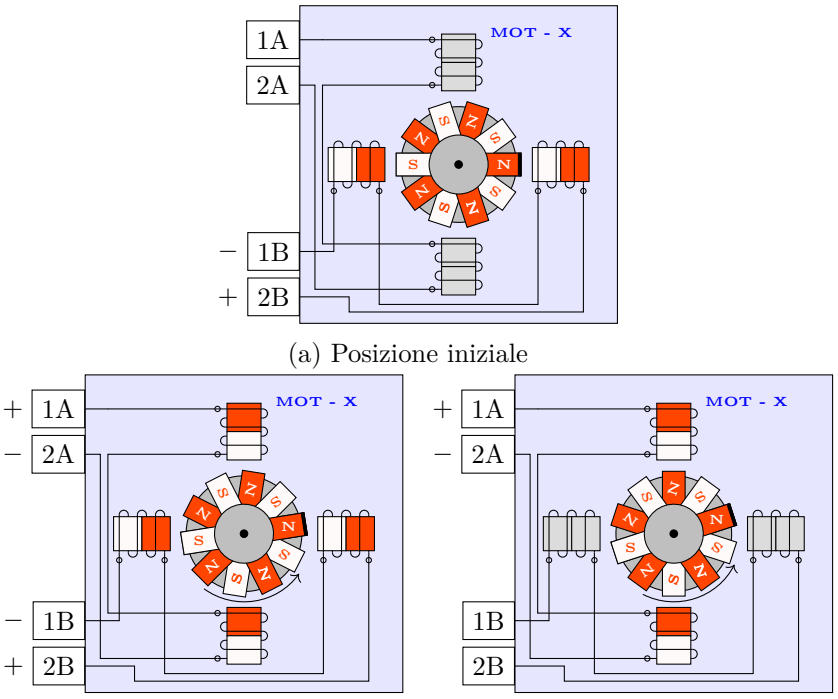
(c) Motore in posizione finale

Figura 3.2: Rotazione Oraria.

Nella figura 3.3 nella pagina seguente mostriamo lo schema per una rotazione antioraria.

Potete notare che l'unica cosa che cambia è l'eccitazione dei terminali **1A** e **2A**

Presentiamo nella Tabella 3.1 nella pagina successiva seguente l'eccitazione delle fasi:



(a) Posizione iniziale

(b) Motore in half-step

(c) Motore in posizione finale

Figura 3.3: Rotazione Antioraria.

Tabella 3.1: Ordine di eccitazione delle fasi

Terminale	1A	2A	1B	2B
Oraria			-	+
	-	+	-	+
	-	+		
Antioraria			-	+
	+	-	-	+
	+	-		

Tutto questo per far capire il concetto importante e a volte non compreso perché per invertire la rotazione di un motore, basta **invertire l'ordine dei fili di una sola fase**.

Questo per spiegare come funziona uno motore stepper.

Ai nostri fini bastano queste informazioni, o poco più.

Il motore stepper ha un secondo dato importante da conoscere per settare il firmware, il numero di step per giro.

Nella rappresentazione data sopra il numero di step per giro è volutamente limitato,

nella pratica i valori più comuni sono 200 e 400 step per giro.

Questi sono valori che interessano in alcuni settaggi della parte meccanica. (TODO! inserire il rimando pagina per la sezione di Configuration_Mechanics.h)

Capitolo 4

Analisi del firmware

Il firmware viene compilato usando queste direttive:

```
#include "StuFW.h"

void setup() {
  printer.setup();
}

void loop() {
  printer.loop();
}
```

potete notare come include un file **StuFw.h** che contiene gli header

All'interno di questo potete trovare una linea

```
#include "src/core/printer/printer.h"
```

che al suo interno definisce come "pubbliche" le due funzioni:

```
111   public: /** Public Function */
112
113     static void setup();  // Main setup
114     static void loop();   // Main loop
```

che poi vengono richiamate.

La parte interessante che poi costituisce il cuore del programma è la funzione **printer.loop()** che è definita nel file **printer.cpp**.

```
213 void Printer::loop() {
214
215   for (;;) {
216
217     printer.keepalive(NotBusy);
218
219     #if HAS_SD_SUPPORT
```

```
220
221       card.checkautostart();
222
223       if (card.isAbortSDprinting()) {
224         card.setAbortSDprinting(false);
225
226         #if HAS_SD_RESTART
227           // Save Job for restart
228           if (IS_SD_PRINTING()) restart.save_job(true);
229         #endif
230
231         // Stop SD printing
232         card.stopSDPrint();
233
234         // Clear all command in quee
235         commands.clear_queue();
236
237         // Stop printer job timer
238         print_job_counter.stop();
239
240         // Auto home
241         #if Z_HOME_DIR > 0
242           commands.enqueue_and_echo_P(PSTR("G28"));
243         #else
244           commands.enqueue_and_echo_P(PSTR("G28 X Y"));
245         #endif
246
247         // Disabled Heaters and Fan
248         thermalManager.disable_all_heaters();
249         zero_fan_speed();
250         setWaitForHeatUp(false);
251
252       }
253
254     #endif // HAS_SD_SUPPORT
255
256     commands.get_available();
257     commands.advance_queue();
258     endstops.report_state();
259     idle();
260
261   }
262 }
```


notate che alla fine richiama la funzione **idle()**, che è definita alla linea 440.

```
440 void Printer::idle(const bool ignore_stepper_queue/*=
                    ↪ false*/) {
441
442   lcdui.update();
```

In questo modo il firmware continua eseguendo le funzioni.

Una piccola nota sui nomi:

ad esempio nella linea:

```
printer.keepalive(NotBusy);
```

viene chiamata la funzione **printer.keepalive** che però non troviamo in giro con quel nome.

Il nome più vicino è quello che troviamo in:

```
816 #if ENABLED(HOST_KEEPALIVE_FEATURE)
817
818   /**
819    * Output a "busy" message at regular intervals
820    * while the machine is not accepting
821    */
822   void Printer::keepalive(const BusyStateEnum state) {
823     if (!isSuspendAutoreport() && host_keepalive_watch.
                        ↪ stopwatch && host_keepalive_watch
                        ↪ .elapsed()) {
824       switch (state) {
```

Che però è nominata come **Printer::keepalive**.

Non è un mistero è semplicemente un artifizio che permette di nominare in modo diverso le cose, questo artifizio è fatto nel file **printer.h** con la linea

```
extern Printer printer;
```

piazzata proprio in fondo al file, questa permette di rinominare lo "spazio dei nomi" **Printer** come **printer** e quindi di utilizzare le funzioni chiamandole **printer.nomefunzione**.

I più scaltri di voi noteranno che però la funzione **keepalive** è si definita attorno alla linea 8228 che però è messa sotto la direttiva del preprocessore:

```
#if ENABLED(HOST_KEEPALIVE_FEATURE)
```

mentre viene richiamata senza particolari cautele all'inzio della funzione **loop**.

Il gioco viene fatto sempre nel file **printer.h**, attraverso le linee:

```
    #if ENABLED(HOST_KEEPALIVE_FEATURE)
      static void keepalive(const BusyStateEnum state);
    #else
```

```
    FORCE_INLINE static void keepalive(const
                        ↪ BusyStateEnum state) { UNUSED(
                        ↪ state); }
#endif
```

che dichiarano la funzione **keepalive** ma usando l'else la dichiarano come FORCE_INLINE per cui questo riferimento è presente sia che sia che **HOST_KEEPALIVE_FEATURE** sia definito sia che non lo sia e il programma gira in modo corretto in entrambi i casi.

Un esempio ulteriore di questo meccanismo lo potere trovare in **src/lcd/lcdui.h**, dove vengono definite alcune cose sia per una stampante con LCD (grafico o a righe di caratteri) ma anche nel caso in cui non sia definito un LCD.

```
250 #else // NO LCD
251
252   static inline void init() {}
253   static inline void update() {}
254   static inline void set_alert_status_P(PGM_P message
                        ↪ ) { UNUSED(message); }
255   static inline void refresh() {}
256   static inline void set_status(const char* const
                        ↪ message, const bool persist=
                        ↪ false) { UNUSED(message);
                        ↪ UNUSED(persist); }
257   static inline void set_status_P(PGM_P const message
                        ↪ , const int8_t level=0) {
                        ↪ UNUSED(message); UNUSED(level);
                        ↪ }
258   static inline void status_printf_P(const uint8_t
                        ↪ level, PGM_P const fmt, ...) {
                        ↪ UNUSED(level); UNUSED(fmt); }
259   static inline void return_to_status() {}
260   static inline void reset_status() {}
261   static inline void reset_alert_level() {}
262   static constexpr bool has_status() { return false;
                        ↪ }
```

Ovviamente vengono definite come funzioni "vuote" oppure come funzioni che non compiono nulla, ma sono definite per poter essere richiamate liberamente senza ricorrere ad una marea di **#IF ENABLED** nel codice.