

**ALMA MATER STUDIORUM - UNIVERSITÀ DI BOLOGNA**  
**SCUOLA DI INGEGNERIA E ARCHITETTURA (SEDE DI BOLOGNA)**  
**Anno Accademico 2020/2021**

---

**RELAZIONE DI FINE TIROCINIO CURRICULARE**

svolto dallo studente:

*Gian Paolo Currà (matricola nr. 0000789852)*

iscritto al Corso di Studio in

*Ingegneria dell'Automazione (codice CdS: 0920)*

presso:

*Alma Mater Studiorum - DIN*

sul seguente argomento:

*Utilizzo materiali SMA in realizzazione e controllo di un sistema*

**Studente:**

\_\_\_\_\_  
(firma)


**Tutor Accademico:**

Prof. (inserire Gregorio Pisaneschi)

\_\_\_\_\_  
(firma per approvazione della relazione finale)

**Referente Struttura Ospitante:**

(inserire Andrea Zucchelli, referente struttura)

  
\_\_\_\_\_  
(firma per approvazione della relazione finale)

# Sommario

|                                                                                            |    |
|--------------------------------------------------------------------------------------------|----|
| Introduzione .....                                                                         | 3  |
| Capitolo 1 – Shape Memory Alloy .....                                                      | 4  |
| <i>Gli Smart Materials:</i> .....                                                          | 4  |
| <i>Martensite (Temperature basse):</i> .....                                               | 5  |
| <i>Austenite (Temperature alte):</i> .....                                                 | 5  |
| <i>Comportamento SMA: Shape Memory Effect (SME)</i> .....                                  | 6  |
| <i>Comportamento SMA: Super-elasticità (SE)</i> .....                                      | 7  |
| Capitolo 2 – Controllo.....                                                                | 8  |
| <i>Obbiettivi del sistema</i> .....                                                        | 8  |
| <i>Apparato circuitale</i> .....                                                           | 10 |
| <i>Unità di governo</i> .....                                                              | 13 |
| <i>L' STM32F407G-DISC1</i> .....                                                           | 13 |
| <i>Task preliminare: accendere e spegnere un led con temporizzazione controllata</i> ..... | 13 |
| <i>Task finale: pilotaggio PWM di MOSFET</i> .....                                         | 14 |
| <i>Utilizzo dei timer integrati sulla scheda STM</i> .....                                 | 15 |
| <i>STM32 Timers In PWM Mode</i> .....                                                      | 16 |
| <i>STM32 PWM Frequenza, Duty Cycle e Risoluzione</i> .....                                 | 16 |
| <i>Codice di programmazione</i> .....                                                      | 17 |

## Introduzione

La finalità alla base di questo progetto di tirocinio è l'utilizzo di materiali Shape Memory Alloy (SMA) nella ricostruzione di un sistema di masse su pista già esistente.

L'impiego di questi materiali ci permette di usufruire delle particolarità degli Smart Materials in modo tale che il sistema finale risulti molto più compatto e dalla buona controllabilità. Il secondo punto in particolare, in quanto il sistema originario possedeva una forte componente spaziale molto ingombrante. Essa che permetteva un controllo ad alta precisione ma non versatilità se si fosse reso utile uno spostamento dell'impianto e portabilità nulla.

Gli SMA posseggono una serie di proprietà non presenti in altre classi di materiali, questo ne permette l'impiego in particolari ambiti applicativi, semplificando il sistema posto in esame.

In ambito mecatronico sono sempre più utilizzati per via delle loro proprietà chimico-fisiche alterabili in maniera pseudo controllata se sottoposti a stress esterni. Tramite un controllo semplice e diretto uno SMA potrebbe andare a sostituire i moderni attuatori riducendo la complessità in termini di cablaggio e spazio di lavoro.

Mediante speciali trattamenti termici, è possibile imprimere al materiale SMA una forma geometrica iniziale che, una volta deformato, è capace di riacquisire se sottoposto a stress termico senza mostrare plasticizzazione.

Questa capacità permette di effettuare un'attuazione in quanto per tornare alla forma impressa, lo SMA utilizza l'energia data dal riscaldamento del materiale per creare lavoro che gli permetta di tornare alla forma originaria, questo meccanismo è utilizzabile dal nostro sistema per effettuare movimentazioni.

In questo trattato è stata applicata una suddivisione in due argomenti riguardanti gli aspetti del sistema trattato.

Nel primo capitolo verranno descritte le proprietà degli Shape Memory Alloy, facendo uno studio teorico e pratico dei comportamenti osservabili nel nostro progetto.

Successivamente verrà analizzato il lavoro compiuto in modo da avere una gestione ottimale degli attuatori SMA, toccando anche argomenti di progettazione del microcontrollore e del circuito di alimentazione utilizzati. Verrà descritta inoltre l'implementazione di elementi circuitali che permettano un'analisi e un controllo in retroazione sugli attuatori gestito dal microcontrollore già in uso.

In questo modo viene introdotta nel sistema una controllabilità significativa, nonostante la compattezza e versatilità maggiori rispetto al sistema originario e fornite dall'attuazione SMA.

## Capitolo 1 – Shape Memory Alloy

### ***Gli Smart Materials:***

Gli Shape Memory Alloy (SMA) appartengono alla famiglia degli “Smart Materials”, ovvero la classe di materiali le cui proprietà chimico-fisiche possono essere alterate in maniera controllata in seguito a stress esterni.

Si tratta di leghe o polimeri che, a fronte di sollecitazioni di tipo termico, luminoso, elettrico, magnetico o vibrazionale, si adattano alle circostanze in modo automatico.

Il gruppo dei Shape Memory Alloy ha la particolarità di avere “memoria” di una forma impressa termomeccanicamente al momento della produzione, ovvero la capacità di poter tornare ad una forma originaria se sottoposto a stress termico senza mostrare plasticizzazione.

Questi materiali riescono ad avere questo comportamento grazie alla loro struttura cristallina e alla compresenza, in rapporti sempre variabili, di due tipologie di composizione cristallografica differenti: martensite (bassa temperatura) ed austenite (alta temperatura).

La transizione tra queste fasi stabili è chiamata trasformazione martensitica termoelastica ed è ciò che causa lo “shape recovery”, considerando che lo stato energetico della struttura cristallina tende a portarsi al livello più basso. Questo effetto si basa sul riordinamento della struttura cristallina interna del materiale mantenendo invariata la composizione chimica.

Un’ altro comportamento che possono assumere gli SMA è quello di superelasticità; nei paragrafi successivi analizzeremo prima le due fasi che caratterizzano gli SMA e poi i due effetti (superelastico e memoria di forma) in relazione alla struttura del materiale.

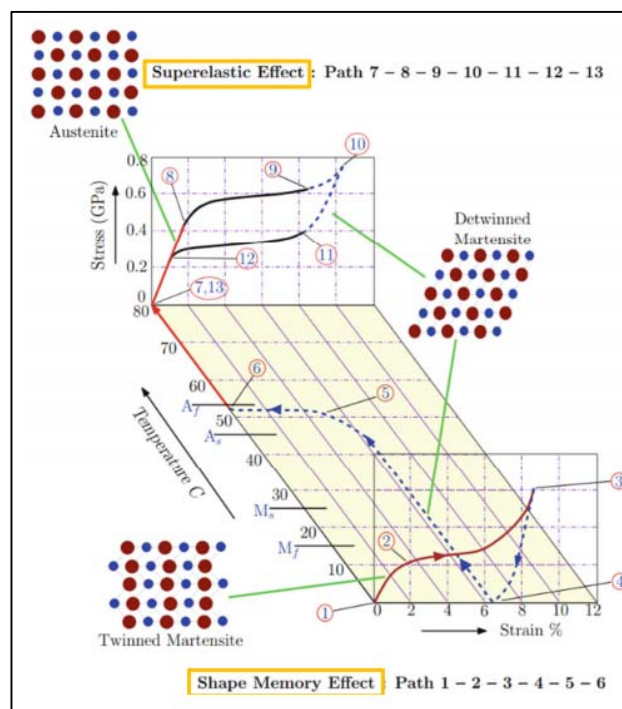


Figura 1

### **Martensite (Temperature basse):**

È caratterizzata da un modulo di Young  $E_M$  basso, questo corrisponde ad una deformabilità maggiore rispetto alla struttura austenitica.

La fase di martensite può essere suddivisa a sua volta in due sottofasi dette martensite twinned e martensite detwinned.

Il passaggio tra una sottofase e l'altra avviene quando il materiale, a bassa temperatura, è sottoposto a carico, provocando un cambiamento strutturale detto "Detwinnig". Questo passaggio è di tipo plastico, anche se il carico non permane, la struttura mantiene la deformazione.

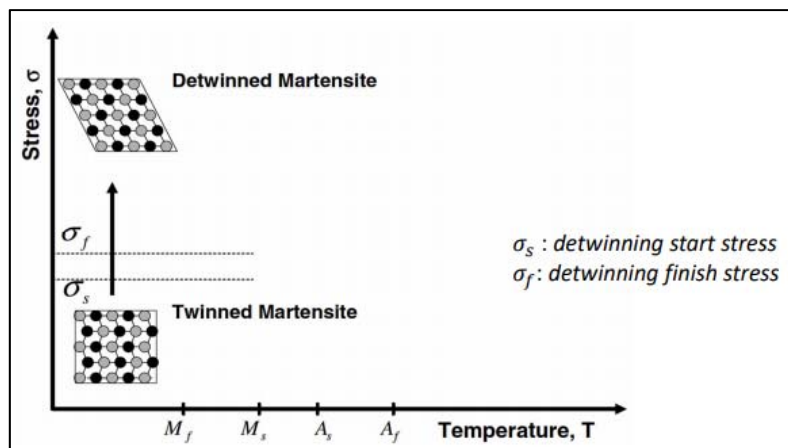


Figura 2 – In questo grafico, tenendo la temperatura costante tale che si rimanga in fase martensitica, possiamo visualizzare il processo di detwinnig in seguito ad uno stress sull'elemento SMA.

### **Austenite (Temperature alte):**

Non possiede sottofasi strutturali, i suoi cristalli si dispongono tutti nella stessa direzione (similmente alla martensite Detwinned) e la struttura è caratterizzata da un modulo elastico  $E_A$  alto, quindi difficilmente deformabile.

È la fase stabile che il materiale raggiunge a temperature elevate e coincide con la struttura termoimpressa al momento della produzione dell'attuatore SMA. Aumentando la temperatura si passa da martensite ad austenite utilizzando il passaggio di fase per generare lavoro.

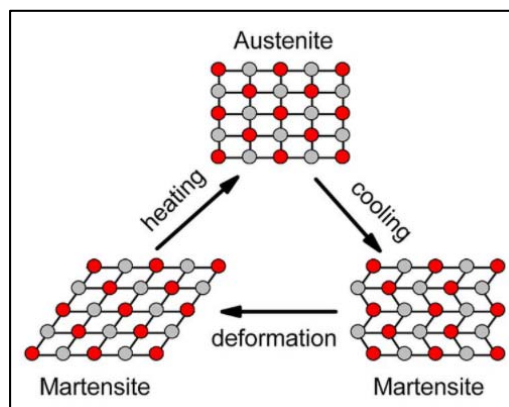


Figura 3

### **Comportamento SMA: Shape Memory Effect (SME)**

Questo effetto si basa sulla transizione graduale da martensite ad austenite. Come accennato precedentemente, questo comportamento può esistere solo grazie alla struttura termoimpressa al momento della produzione del nostro provino SMA.

I provini vengono portati a temperature molto alte (maggiori di quella che sarà la temperatura di austenite del prodotto finale) e caricate opportunamente in modo che durante il processo venga mantenuta una forma voluta, ad esempio molle o filamenti.

Una volta finito questo processo, l'attuatore SMA sarà in grado, tramite un aumento di temperatura, di passare da fase martensitica (in genere temperatura ambiente) a fase austenitica, ritornando alla forma originaria impressa durante la produzione.

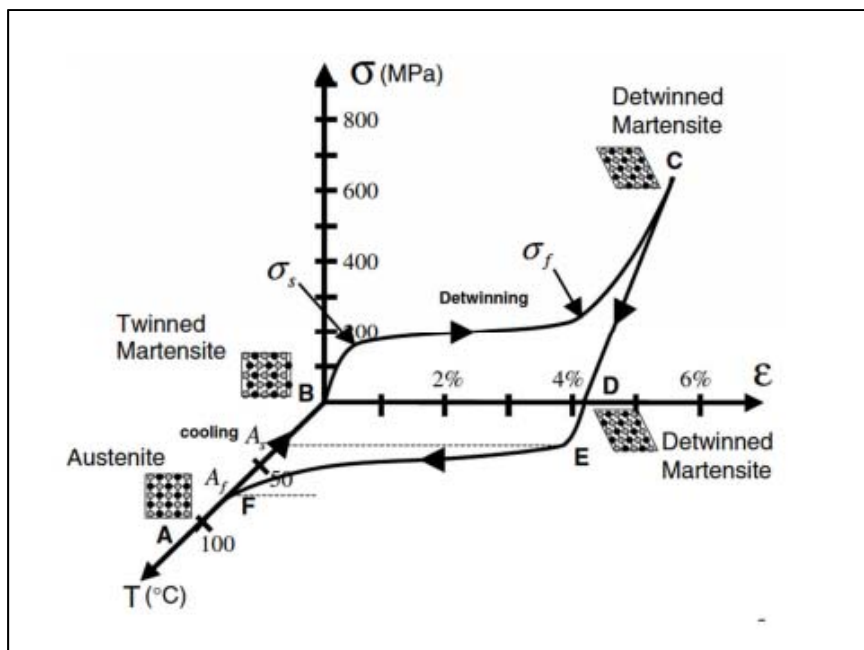


Figura 4:  $A_s < T < A_f$ : intervallo in cui ho la trasformazione graduale della martensite in austenite

Descrizione a step di trasformazione martensitica termoelastica (SME) in Fig.2:

1. Punto B: l'origine degli assi non significa che le grandezze siano nulle, temperatura ambiente e in fase martensite twinned.
2. Tratto BC: sto caricando il provino SMA attuando un cambiamento graduale da martensite twinned a detwinned.
3. Tratto CD: diminuisco fino ad azzerare il carico, idealmente il materiale mantiene la deformazione in modo plastico ma possiamo notare che un piccolo errore (dato da fattori di produzione) può portare ad una diminuzione di  $\varepsilon$ , caso non ideale.
4. Tratto DE: aumento la temperatura del provino, arrivando al valore  $A_s$ .
5. Tratto EF: T arriva al valore  $A_f$ , lo SMA possiede solo struttura cristallina austenitica, tornando alla forma originaria, azzerando la deformazione  $\varepsilon$ .
6. Tratto FB: raffreddamento, mantenendo la stessa forma ( $\varepsilon = 0$ ), il provino SMA torna alla sola struttura cristallina martensitica.

### Comportamento SMA: Super-elasticità (SE)

Al contrario del caso precedente, dove a temperatura ambiente l'attuatore SMA è in fase martensitica, in questo caso è l'austenite ad essere "settata" sulla temperatura ambiente. Se un materiale SMA in questa fase venisse sottoposto all'azione di un carico superiore ad un certo valore critico, sarebbe indotta una trasformazione in martensite. In questo caso, la struttura cristallina viene chiamata martensite indotta da sforzo (Stress Induced Martensite - SIM). Una volta rimosso il carico, si ha completo recupero della deformazione indotta da esso.

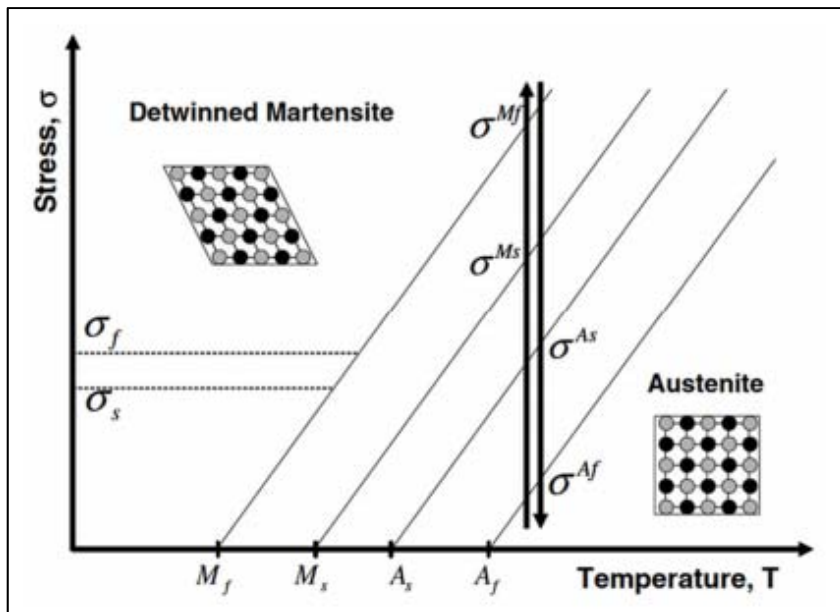


Figura 5

## Capitolo 2 – Controllo

### Obbiettivi del sistema

Dato il sistema iniziale composto da:

- Due fili SMA che fungono da freni, il design particolare gli permette, una volta scaldati, che possano spingere la massa a cui sono legati contro la pista in modo che l'attrito possa mantenere la massa in posizione.
- Complesso molle, in particolare, una passiva in trazione e una SMA in compressione.
- Masse  $M_1$ ,  $M_2$ .

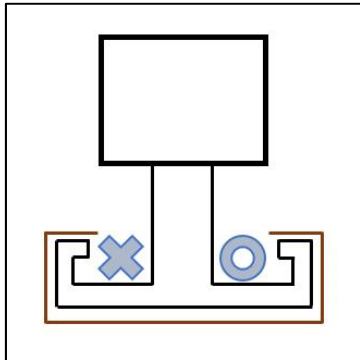


Figura 6: Freno

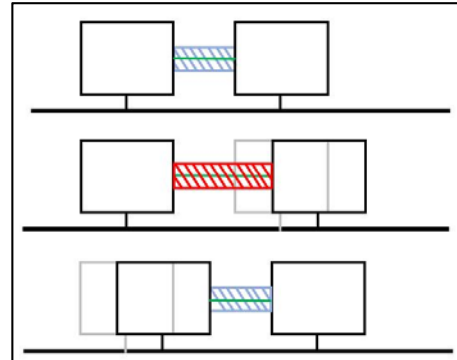


Figura 7: Impianto

Si vuole definire un controllo temporizzato atto al movimento su pista delle due masse (sormontate dai freni a filo SMA). A tale scopo abbiamo definito sette step che riassumono il moto:

1.  $F_2$  ON.
2. Compressione della molla centrale => SMA ON.
3.  $F_1$  ON.
4.  $F_2$  OFF.
5. Trazione della molla centrale => SMA OFF.
6.  $F_2$  ON.
7.  $F_1$  OFF.

Con:

$F_2$  = freno anteriore

$F_1$  = freno posteriore

Al fine di definire i delta temporali delle varie fasi di attivazione e raffreddamento degli SMA è necessario usufruire di dati sperimentali.

Per quanto riguarda invece il tempo richiesto per lo spostamento, abbiamo effettuato delle valutazioni riguardo all'equilibrio energetico.



### Ipotesi di studio

- Tempo iniziale  $t_0 = 0$  .
- $M_1$  si trova in  $s_0 = 0$  all'istante  $t_0$ .
- $M_2$  si trova in  $s_0 + l_0$  all'istante  $t_0$  ,dove  $l_0$  è la lunghezza a riposo delle molle in parallelo.
- All'istante  $t_0$  le molle in parallelo sono scariche.

### Procedimento

Dopo l'attivazione del  $F_2$  e della molla SMA di compressione si ha uno spostamento dovuto alla differenza di forze elastiche (SMA – passiva).

È noto che la  $F_{SMA} = K_{SMA} * \Delta L_{compressione} (= M_1 * a)$ .

Essendo incognita unicamente l'accelerazione, in quanto la rigidità viene calcolata sperimentalmente, è possibile ricavare un valor medio (root-mean-square) di tale grandezza risolvendo l'equazione.

Considerando quindi un'accelerazione costante, è possibile ricavare le velocità e il tempo di spostamento attraverso il sistema:

$$\begin{cases} v_f = v_i + a_{rms} * t \\ s_f = s_i + vt + \frac{1}{2}at^2 \end{cases}$$

E quindi trovare il valore temporale totale dello stato ON di  $F_2$  (attivazione + spostamento).

Al termine di questa fase si ha la disattivazione di  $F_2$  e della molla di compressione (SMA) contemporaneamente con l'attivazione del filo  $F_1$  .

Per temporizzare lo stato ON di questo freno è necessario considerare il minimo tra i valori di cooldown degli attuatori e il tempo necessario alla completa estensione del complesso molle. Analogamente al caso precedente è possibile fare una considerazione energetica per definire l'accelerazione riscontrata in tale moto, e quindi il tempo.

### Apparato circuitale (Primo Prototipo)

In questo progetto useremo le proprietà di materiale a memoria di forma degli SMA, infatti bisogna che sia i fili che le molle vengano alimentati con una corrente abbastanza elevata da ottenere una variazione di temperatura. Per fare ciò è stato impostato un circuito di alimentazione pilotato da un microcontrollore tramite un segnale PWM e MOSFET.

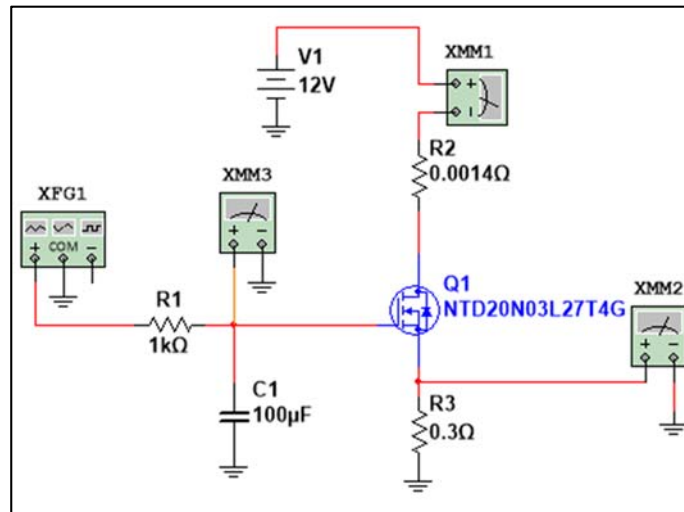


Figura 8: esempio del prototipo del circuito necessario all'alimentazione di un solo attuttore SMA

Il circuito raffigurato in Fig.6 è il prototipo di un singolo attuttore SMA. Per comodità abbiamo deciso di controllare tutte e tre i circuiti di alimentazioni collegati alla rete 220V con lo stesso microcontrollore.

Componenti:

- XMM1/2/3: voltmetri necessari alla misurazione della tensione nei punti evidenziati.
- XFG1: generatore di onde PWM (onda quadra) coincidente con la STM32.
- R1 e C1: fungono da filtro passa-basso con frequenza di taglio pari a  $\frac{1}{2\pi RC}$ , questo filtro è necessario a ricavare l'armonica zero (al valor medio di tensione) per alimentare il MOSFET in tensione continua, poiché il MOSFET non può essere alimentato con tensione alternata (nel caso PWM quadra).
- R2: resistenza che riassume la R variabile del nostro carico SMA, è stato scelto un valor medio ideale tra la fase martensitica e austenitica.
- R3: resistenza utilizzata per ricavare la corrente nel circuito, siamo in grado di ricavare una corrente variabile in funzione della variabilità di R2 reale. Ricavando questa informazione e portandola dentro la scheda STM32 è possibile introdurre una retroazione, alimentando il carico SMA in base a quanto varia la R2.
- V1: alimentatore da banco da 12V.

Calcolo delle resistenze:

$$R = \rho \frac{1}{A}$$

Con:

$$\rho_A = 82 \cdot 10^{-6} \Omega \cdot cm$$

$$\rho_M = 76 \cdot 10^{-6} \Omega \cdot cm$$

Dalla formula della resistenza ottengo che:

$$R_A = 82 \cdot 10^{-6} \cdot \frac{2}{\pi \cdot 0.01^2} = 0.52 \Omega$$

$$R_M = 76 \cdot 10^{-6} \cdot \frac{(2-5\% \cdot 2)}{\pi \cdot 0.01^2} = 0.46 \Omega$$

In fase martensitica la lunghezza corrisponde al 95% della lunghezza in fase austenitica (5% è la massima deformazione attuabile dal filo SMA).

È stato definito un valore ideale di  $R_{filo}$  pari al valor medio tra le due resistenze  $R_A$  e  $R_M$ :

$$R_{filo} = \frac{0.46 + 0.52}{2} = 0.49 \Omega$$

Applicando la legge di Kirchoff al circuito di alimentazione del filo:

$$v_{R3_{filo}} = 12 - R_{filo} \cdot i_3, \text{ con } i_3 \text{ gestita da utente}$$

$$R_3 = \frac{v_{R3_{filo}}}{i_3}$$

Allo stesso modo si calcola la  $R_{molla}$ :

$$R_A = 82 \cdot 10^{-6} \cdot \frac{3.2}{\pi \cdot 0.075^2} = 0.015 \Omega$$

$$R_M = 76 \cdot 10^{-6} \cdot \frac{(3.2-5\% \cdot 3.2)}{\pi \cdot 0.075^2} = 0.013 \Omega$$

Di conseguenza:

$$R_{molla} = \frac{0.015 + 0.013}{2} = 0.014 \Omega$$

Applicando la legge di Kirchoff al circuito di alimentazione della molla

$$v_{R3_{molla}} = 12 - R_{filo} \cdot i_3, \text{ con } i_3 \text{ gestita da utente}$$

$$R_3 = \frac{v_{R3_{molla}}}{i_3}$$

### Apparato circuitale (Secondo Prototipo)

La progettazione del secondo prototipo dell'apparato circuitale è derivata dall'alta alimentazione necessaria per poter innalzare, in tempistiche relativamente basse, l'attuatore SMA. Con correnti di quella portata, il primo prototipo non risulta robusto e sicuro a fronte di guasti di tipo energetico quali, dissipazione in calore e/o rottura dei componenti usati.

Il secondo prototipo è caratterizzato dalla seguente struttura mostrata in due varianti:

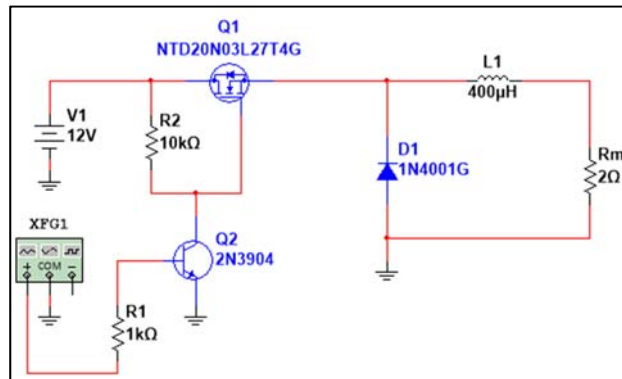


Figura 9 Convertitore Buck

#### 1. Convertitore buck:

Come mostrato in Fig. 9 l'interruttore viene aperto e chiuso con una frequenza controllata dall'unità di comando (STM32DISCOVERY) tramite onda quadra PWM.

- Interruttore Chiuso: l'induttore inizia a caricarsi della corrente fornita dall'alimentazione in ingresso in quanto il diodo risulta interdetto.
- Interruttore Aperto: l'induttore si scarica sul comparto RC. Il diodo e l'interruttore aperto non permettono all'alimentazione di caricare l'induttore ulteriormente.

La tensione sull'induttore risulta fortemente legata al Duty Cycle dell'onda PWM ( $V_o = V_i \cdot Duty_{cycle}$ ), per questo è necessario un componente C (condensatore) in parallelo al carico resistivo (SMA) per poter estrarre la componente continua dagli impulsi di carica e scarica dell'induttore in modo da ottenere un passaggio di corrente costante sul carico.

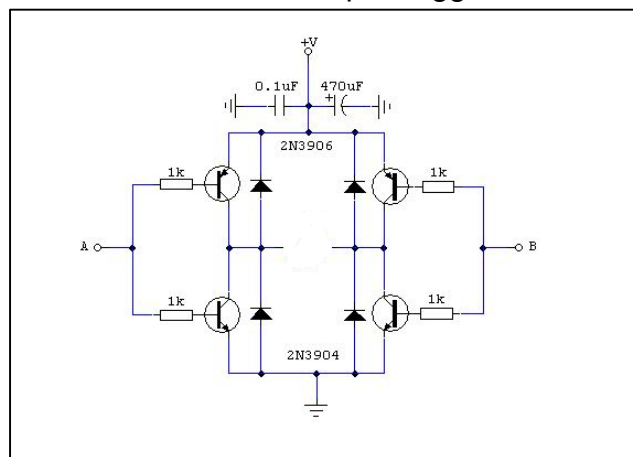


Figura 10:Ponte H

#### 2. Ponte H (Integrated Circuit):

Questa possibile variante è mostrata nella Fig. 10, il funzionamento è simile al Buck Converter ma permette un controllo della corrente erogata al carico completo, ovvero in entrambi i versi. In genere viene utilizzato per controllare carichi come motori elettrici in modo da controllare la coppia del carico in entrambi i sensi (4 quadranti).

## **Unità di governo**

Sono stati intrapresi due percorsi differenti riguardo alla scelta dell'unità di controllo:

1. Arduino.
2. STM32F407G-DISC1

Quello che ho scelto di approfondire come percorso di tirocinio è l'approccio con il microcontrollore STM32.

### **L' STM32F407G-DISC1**

È un microcontrollore programmabile in linguaggio C, viene utilizzato molto spesso per la programmazione embedded la cui funzione di base sta nel venire integrato nel sistema che si vuole controllare (approccio Special Purpose).

In questo progetto è stato utilizzato un firmware di tipo StdPeripheral e considerato che si sta cercando una temporizzazione di attuazioni, è stato scelto un approccio con i timer integrati sulla scheda.

### **Task preliminare: accendere e spegnere un led con temporizzazione controllata**

Per questo primo task che ci permette di configurare i pin come output per led (anch'essi integrati sulla scheda per comodità), è stata scelta una funzione ad interrupt particolare chiamata SysTick.

Questa funzione funge da "metronomo" del nostro sistema, ovvero un timer periodico per porre un ritardo o controllare un determinato comando. Si basa su particolari costrutti logici chiamati "interrupt" che sono eventi straordinari (ossia non previsti nella normale esecuzione del codice) che provocano una deviazione nel normale flusso del programma. In genere sono generati da periferiche esterne alla CPU, ma esistono anche degli interrupt interni. Il punto chiave è che sono eventi asincroni e quindi non avvengono in modo sincrono con l'esecuzione del codice. Il SysTick non è una periferica in senso stretto (come GPIO, seriali ecc), esso è un timer che è parte del CORTEX, ossia del CORE stesso.

La SysTick in particolare può riassumere il comportamento di un semplice timer, ovvero esegue un interrupt periodico ogniqualvolta viene raggiunto il valore di un certo contatore. Quando l'interrupt è abilitato, l'esecuzione principale del main viene fermata, salvata e poi spostata ad eseguire l'Interrupt Service Routine (ISR ovvero parte di codice) contenuta nel SysTick\_handler.

Esempio di funzione SysTick dove un led effettua un blinking :



Figura 11

1. Main: funzione principale che funge da corpo del programma e da cui vengono “chiamate” le funzioni accessorie. Le particolarità sono tre:
  - “while(1);” : permette alla STM32 di eseguire in loop il programma creato dal programmatore.
  - “SysTick\_Config”: effettua un controllo sul corretto avvenimento dell’interrupt.
  - SystemCoreClock: corrisponde al numero di tick che produce il clock della scheda in un secondo, dividendo per 1000 si ottiene l’effetto di entrare nella funzione SysTick\_Handler ogni millisecondo
2. Routine di interrupt: in questa funzione effettuo l’azione di “toggle” del led una volta che il contatore risulta pari o maggiore di 500 millisecondi azzerando il contatore.

### **Task finale: pilotaggio PWM di MOSFET**

Una volta definito il pilotaggio PWM, la funzione SysTick è risultata inadeguata in quanto non permette un controllo su MOSFET.

Considerando le altre peculiarità della scheda STM32, è stato deciso di utilizzare il comparto timer della stessa. I timer rappresentano uno degli organi più complessi del microcontrollore in quanto possono essere programmati in modo molto versatile permettendo così una precisione elevata nel controllo. Nel caso particolare dei PWM abbiamo il settaggio dei pin di output come GPIO PWM che permette di riconoscere il timer (opportunamente associato ai pin voluti) come generatore di onde di tipo PWM.

## Utilizzo dei timer integrati sulla scheda STM

I timer presenti sulla scheda STM32 sono suddivisi nelle seguenti tipologie:

- 5 timer a 16 bit;
- 1 timer a 16 bit per il controllo del motore PWM sincronizzato con il converter (AC);
- 2 timer da 32 bit.

Di cui la maggior parte sono detti general purpose, ovvero programmabili per molti impieghi diversi.

Nel progetto è stato utilizzato il timer 4 (compreso tra quelli general purpose da TIM2 a TIM5). Le caratteristiche fondamentali sono le seguenti:

- Molteplicità a 16 bit: ossia il registro del contatore principale è un registro a 16 bit che può contare in alto (ovvero essere incrementato per ogni ciclo di clock), può essere decrementato, oppure entrambi
- Un prescaler programmabile a 16 bit: si applica a monte di un clock principale e permette, a seconda del rapporto di scaling, di frazionare l'onda del clock (anche "on the fly") il contatore della frequenza del clock con qualsiasi fattore compreso tra 1 e 65535 (ovvero si può scalare la frequenza con un fattore compreso tra questi due valori).
- Ogni timer presenta quattro canali indipendenti (questi 4 canali indipendenti sono collegati allo stesso contatore principale, che è l'elemento principale del timer) ognuno di questi quattro canali può essere configurato in una delle 4 modalità:
  - Input capture.
  - Output compare.
  - Generazione del PWM (Edge-aligned o Center-aligned modes)
  - Modalità output one-pulse (un singolo impulso in risposta ad un singolo evento).

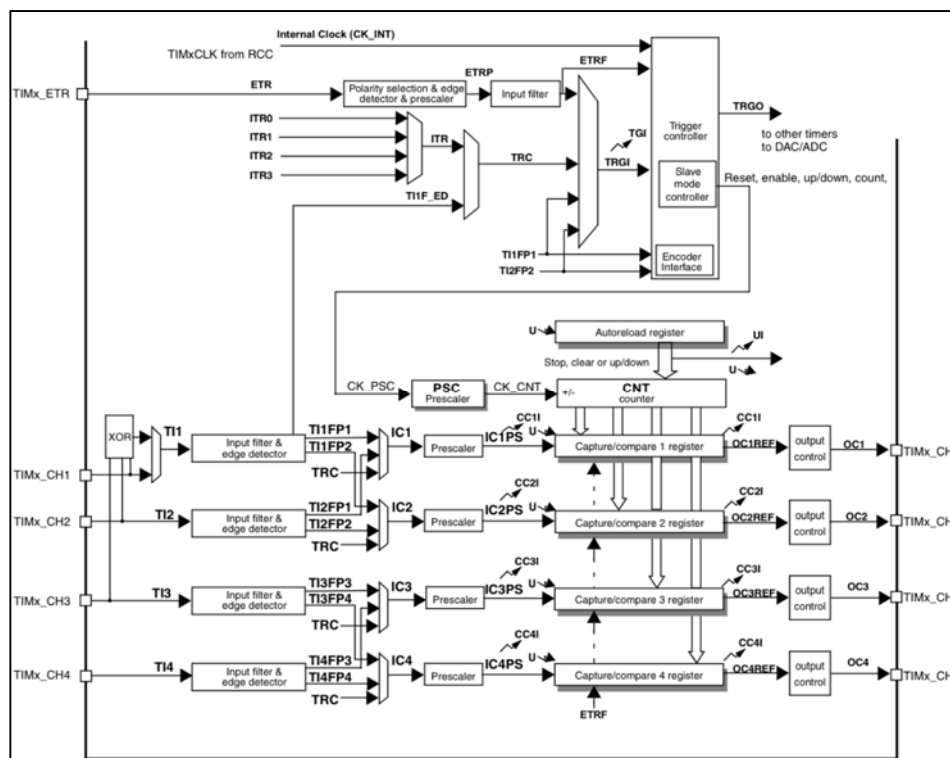


Figura 12: Schema a blocchi tipico del timer general-purpose

### STM32 Timers In PWM Mode

La modalità PWM (Pulse Width Modulation) permette di generare un segnale con una frequenza e un duty cycle determinati rispettivamente tramite il contenuto dei registri TIMx\_ARR e TIMx\_CCRx.

- TIMx\_ARR: contiene l'auto reload value, una volta raggiunto questo valore, se siamo in modalità upcounting, il timer riparte a contare da zero.
- TIMx\_CCRx: registro che manda un interrupt appena viene raggiunto il valore contenuto.

La modalità PWM può essere selezionata indipendentemente su ogni canale scrivendo '110' (PWM mode 1) o '111' (PWM mode 2) nei bits di OCxM nel registro TIMx\_CCMRx.

#### 1. PWM mode 1:

- In upcounting, il canale è attivo fintanto che contatore timer < capture compare value, altrimenti inattivo.
- In downcounting, il canale è inattivo fintanto che contatore timer > capture compare value, altrimenti attivo.

#### 2. PWM mode 2:

- in upcounting, il canale è inattivo fintanto che contatore timer < capture compare value, altrimenti attivo.
- in downcounting, il canale è attivo fintanto che contatore timer > capture compare value altrimenti inattivo.

### STM32 PWM Frequenza, Duty Cycle e Risoluzione

La  $F_{PWM}$  si ottiene tramite la formula:

$$F_{PWM} = \frac{F_{CLK}}{(ARR + 1) * (PSC + 1)}$$

Il Duty Cycle percentuale è controllato dal cambiamento del valore del registro CCR, di conseguenza il Duty Cycle equivale a:

$$DutyCycle_{PWM}[\%] = \frac{CCR_x}{ARR_x}[\%]$$

La risoluzione è il numero di livelli discreti del duty cycle che si possono settare. Questo numero determina quanti step il timer deve contare prima di raggiungere il massimo valore. La grandezza dello step o il numero degli step sono strettamente legati alla precisione che l'onda PWM può raggiungere, da qui la formula della risoluzione:

$$Resolution_{PWM} = \frac{\log(\frac{F_{CLK}}{F_{PWM}})}{\log(2)} [Bits]$$

Per completezza è qui inserita la formula che lega il valore di ARR o il Prescaler:

$$Resolution_{PWM} = \frac{\log((ARR + 1) * (PSC + 1))}{\log(2)} [Bits]$$

Per introdurre la possibilità di un algoritmo "User Friendly", considerato che il registro ARR coincide con il numero di step totali che il timer deve percorrere e il Prescaler con il divisore



di frequenza del clock. Definendo SystemCoreClk come il numero di tick che il clock della scheda STM32 produce in un secondo ottenendo le seguenti formule:

$$f_{timer} = step_{pwm} \cdot f_{pwm}$$

$$Prescaler = \frac{\frac{SystemCoreClk}{2}}{f_{timer}}$$

### Codice di programmazione

```

15 /***** Defines *****/
16 #define PWM_Steps 1000
17 #define PWM_Frequency 40
18
19 /*----- Function prototypes -----*/
20 void TimerConfiguration(void);
21 void GPIOConfiguration(void);
22 void TIM4Ch1Configuration(void);
23 void TIM4Ch2Configuration(void);
24
25 /***** Main *****/
26 int main(void)
27 {
28     /* GPIO Configuration */
29     GPIOConfiguration();
30
31     /* TIM4 Configuration */
32     TimerConfiguration();
33
34     /* TIM4_Ch1Configuration */
35     TIM4Ch1Configuration();
36
37     /* TIM4_Ch2Configuration */
38     TIM4Ch2Configuration();
39
40     while (1);
41 }

```

Figura 13: Qui raffigurato il main del programma di generazione di onde PWM

È possibile notare le dichiarazioni di 4 funzioni poi richiamate nel main:

1. GPIOConfiguration()
2. TimerConfiguration()
3. TIM4Ch1Configuration()
4. TIM4Ch2Configuration()

```

47 void GPIOConfiguration(void)
48 {
49     // LED initialization
50     GPIO_InitTypeDef GPIO_InitStructure; /* Enable the GPIO_LED Clock */
51
52     /*Always initialise local variables before use*/
53     GPIO_StructInit (&GPIO_InitStructure);
54
55     /* Configure the GPIO_LED pin and enable AHB1 BUS Port D */
56     RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_GPIOD, ENABLE);
57
58     /*Configure for Pin 12 and Pin 13 Port D as output*/
59     GPIO_InitStructure.GPIO_Pin = GPIO_Pin_12 | GPIO_Pin_13;
60     GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AF;
61     GPIO_InitStructure.GPIO_OType = GPIO_OType_PP;
62     GPIO_InitStructure.GPIO_PuPd = GPIO_PuPd_UP;
63     GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
64     GPIO_Init(GPIOD, &GPIO_InitStructure);
65
66     GPIO_ResetBits(GPIOD,GPIO_Pin_12 | GPIO_Pin_13);
67
68 }

```

Figura 14: Funzione GPIOConfiguration()

La funzione GPIOConfiguration è necessaria alla configurazione di tutti i General Purpose pin (GPIO):

- GPIO\_InitTypeDef GPIO\_InitStructure: creazione della struttura dati che permette la gestione dei pin.
- GPIO\_StructInit (&GPIO\_InitStructure): inizializzazione della suddetta struttura.
- RCC\_AHB1PeriphClockCmd(RCC\_AHB1Periph\_GPIOD, ENABLE): abilitazione del clock del bus AHB1, che permette l'accensione e l'utilizzo della porta D (pin 12,13), dove sono contenuti i pin utilizzati.
- GPIO\_InitStructure.GPIO\_Mode = GPIO\_Mode\_AF: i pin vengono inizializzati come AF (alternate function), questo permette al pin configurato in output di accogliere un'onda proveniente dalla scheda e riconoscerla come tale
- GPIO\_InitStructure.GPIO\_OType = GPIO\_OType\_PP: specifica il tipo di output che devono fornire i pin, ovvero PushPull.
- GPIO\_InitStructure.GPIO\_PuPd = GPIO\_PuPd\_UP: specifica la tipologia di pull (up or down) per l'output.
- GPIO\_InitStructure.GPIO\_Speed = GPIO\_Speed\_50MHz: specifica la velocità di commutazione del pin.

```

71 void TimerConfiguration(void)
72 {
73     uint16_t ARR_Value = PWM_Steps-1;
74     uint32_t Counter_Freq = PWM_Steps*PWM_Frequency;
75     uint32_t PSC_Value = ((SystemCoreClock/2) /Counter_Freq) - 1;
76
77     /* TIM4 clock enable */
78     RCC_APB1PeriphClockCmd(RCC_APB1Periph_TIM4, ENABLE);
79
80     TIM_TimeBaseInitTypeDef TIM_TimeBaseStructure;
81
82     TIM_TimeBaseStructure.TIM_Prescaler = PSC_Value;
83     TIM_TimeBaseStructure.TIM_CounterMode = TIM_CounterMode_Up;
84     TIM_TimeBaseStructure.TIM_Period = ARR_Value;
85     TIM_TimeBaseStructure.TIM_ClockDivision = TIM_CKD_DIV1;
86     TIM_TimeBaseStructure.TIM_RepetitionCounter = 0;
87     TIM_TimeBaseInit(TIM4, &TIM_TimeBaseStructure);
88
89     TIM_Cmd(TIM4, ENABLE);
90
91 }

```

Figura 15: Funzione TimerConfiguration()

È possibile notare nelle dichiarazioni all'inizio della funzione che corrispondono alle funzioni descritte nella teoria dei timer configurati in PWM. La particolarità sta nel prescaler, il "-1" posto a destra è presente in quanto la scheda considera PSC=0 e non PSC=1 come divisione nulla, quindi il SystemCoreClock rimane invariato nel timer solo se il valore del prescaler è uguale a zero.

- uint16\_t ARR\_Value = PWM\_Steps-1: qui è stato dichiarato il valore ARR (autoreload register) pari a un numero a scelta dell'utente (ad esempio 1000 se si volesse un segnale ogni 1000 us).
- uint32\_t Counter\_Freq = PWM\_Steps\*PWM\_Frequency: è stata definita la frequenza voluta dall'utente, essa è composta dal numero di step percorribili dal timer moltiplicato per la frequenza che si vuole dare al PWM
- uint32\_t PSC\_Value = ((SystemCoreClock/2) /Counter\_Freq) - 1: formula necessaria al valore di prescaler, è stata introdotta nel capitolo precedente.
- RCC\_APB1PeriphClockCmd(RCC\_APB1Periph\_TIM4, ENABLE): inizializzazione del clock del bus APB1, abilitando così il timer 4.
- TIM\_TimeBaseInitTypeDef TIM\_TimeBaseStructure: creazione della struttura dati che permette la gestione del timer.
- TIM\_TimeBaseStructure.TIM\_CounterMode = TIM\_CounterMode\_Up: gestione del timer in upcounting.
- TIM\_TimeBaseStructure.TIM\_Period = ARR\_Value: valore di fondoscala del periodo e definizione della risoluzione del PWM.
- TIM\_TimeBaseStructure.TIM\_ClockDivision = TIM\_CKD\_DIV1: suddivisione del clock non necessaria, di conseguenza settaggio con DIV1 di default.
- TIM\_TimeBaseStructure.TIM\_RepetitionCounter = 0: contatore di interrupt scattati a fronte dell'azzeramento del timer, non necessario in quanto si cerca la forma d'onda e non il settaggio di un contatore.

- TIM\_TimeBaseInit(TIM4, &TIM\_TimeBaseStructure); inizializzazione della struttura dati sul timer 4.
- TIM\_Cmd(TIM4, ENABLE); abilitazione del timer 4.

```

93 void TIM4Ch1Configuration(void)
94 {
95     TIM_OCInitTypeDef  TIM_OCInitStructure_1;
96     /* Output Compare Timing Mode configuration: Channel1 */
97     TIM_OCInitStructure_1.TIM_OCMode = TIM_OCMode_PWM1;
98     TIM_OCInitStructure_1.TIM_OutputState = TIM_OutputState_Enable;
99     TIM_OCInitStructure_1.TIM_Pulse = 400;
100    TIM_OCInitStructure_1.TIM_OCPolarity = TIM_OCPolarity_High;
101
102    TIM_OC1Init(TIM4, &TIM_OCInitStructure_1);
103    TIM_OC1PreloadConfig(TIM4, TIM_OCPreload_Enable);
104    GPIO_PinAFConfig(GPIOD, GPIO_PinSource12, GPIO_AF_TIM4);
105
106 }
107 void TIM4Ch2Configuration(void)
108 {
109     TIM_OCInitTypeDef  TIM_OCInitStructure_2;
110     /* Output Compare Timing Mode configuration: Channel2 */
111     TIM_OCInitStructure_2.TIM_OCMode = TIM_OCMode_PWM1;
112     TIM_OCInitStructure_2.TIM_OutputState = TIM_OutputState_Enable;
113     TIM_OCInitStructure_2.TIM_Pulse = 900;
114     TIM_OCInitStructure_2.TIM_OCPolarity = TIM_OCPolarity_High;
115
116     TIM_OC2Init(TIM4, &TIM_OCInitStructure_2);
117     TIM_OC2PreloadConfig(TIM4, TIM_OCPreload_Enable);
118     GPIO_PinAFConfig(GPIOD, GPIO_PinSource13, GPIO_AF_TIM4);
119 }

```

Figura 16: funzioni di configurazione canale TIM4ChXConfiguration, X coincide con il numero del canale

- TIM\_OCInitTypeDef TIM\_OCInitStructure X: definizione struttura dati per configurare il canale corrispondente.
- TIM\_OCInitStructure X.TIM\_OCMode = TIM\_OCMode\_PWM1: gestione del canale con modalità PWM (1 nel nostro caso).
- TIM\_OCInitStructure X.TIM\_OutputState = TIM\_OutputState\_Enable:
- TIM\_OCInitStructure X.TIM\_Pulse = 900: numero corrispondente alla percentuale di duty cycle.
- TIM\_OCInitStructure X.TIM\_OCPolarity = TIM\_OCPolarity\_High: polarità del PWM, usata per creare generatori PWM inversi.
- TIM\_OCXInit(TIM4, &TIM\_OCInitStructure X): inizializzazione struttura dati.
- GPIO\_PinAFConfig(GPIOD, GPIO\_PinSource13(12), GPIO\_AF\_TIM4): configurazione e collegamento del pin di output al timer 4 canale X.

