Appunti IoT Systems

Capitolo 4

I Timer

Simone Coco

I timer

I timer sono necessari per gestire delle esigenze periodiche del microcontrollore che non possono essere trattate dal codice che esegue il microcontrollore.

Ad esempio, il campionamento del segnale non può essere affidato al codice che esegue il microcontrollore, esso è gestito da un'altra parte del microcontrollore, addetta al controllo di eventi che si susseguono periodicamente: **i timer**.

I timer sono dei componenti, interni al microcontrollore, in grado di mandare un segnale in certi output periodicamente, risvegliando l'avvenimento di determinati eventi.

Esempio utilizzo di un timer:

- Ogni qualvolta avvenga un evento aleatorio che deve essere "etichettato" con un determinato tempo (timestamp) il timer può restituire il tempo richiesto
- Campionamento
- Oscillazione del PVM per accensione e spegnimento
- Qualsiasi tipologia di evento periodico da gestire

Contatore

Il timer è semplicemente un contatore e ad ogni ciclo del clock esso può essere incrementato o decrementato.

Alcuni Timer possono andare avanti e indietro, altri possono essere solo incrementati, ad esempio vi sono timer che possono essere solo incrementati e quando arrivano al valore massimo (overflow) lanciano un segnale e ricominciano, altri possono andare verso l'alto e verso il basso in maniera configurabile o fissata.

Essendo un contatore, il timer ha un valore attuale, quindi anche esso ha una propria **risoluzione: range di valori che esso può assumere,** avendo n bit avremo un range che va da 0 a $2^{(n-1)}$valori.

Alimentazione del timer

Un Timer può essere alimentato in diversi modi:

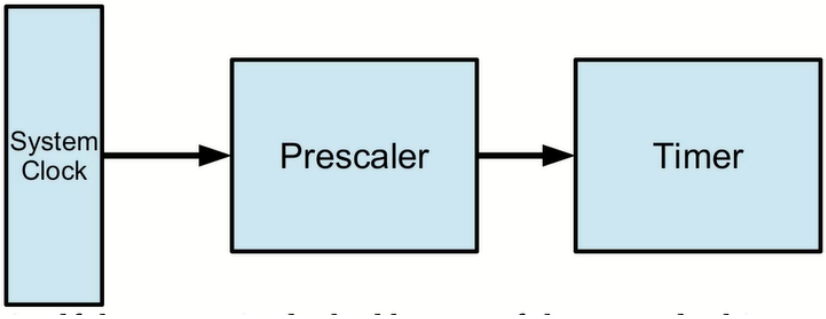
1. Clock di sistema del microcontrollore (detto system clock internal clock)

Il timer incrementa il suo valore per ogni ciclo di clock. Questa è l'implementazione più "ovvia" e semplice;
in base alla frequenza di clock, ad ogni tick si questo viene incrementato il contatore e se quest'ultimo è di 4bit possiamo arrivare a $2^{(4-1)} = 15$bit, ogni 15 giri di clock il timer va in overflow e deve ricominciare da capo. Questo significa che il massimo intervallo misurabile è 15 tick di clock. Se aumenta il numero di bit del timer aumenta anche il massimo intervallo misurabile

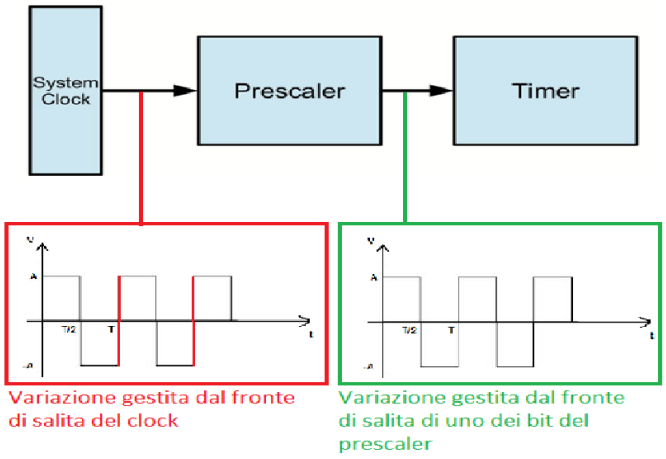
2. Prescaler

Incrementare il numero di bit in un timer non è però sempre la giusta soluzione, poiché potrebbe risultare essere costoso, quindi si potrebbe necessitare di un timer che, a parità di bit, duri più a lungo (ad esempio, con 4 bit non voglio arrivare ad un massimo di 15bit ma a 150, ovvero ben 100 volte il tempo previsto).

In un microcontrollore si trova spesso un **prescaler** tra il timer e il clock; il precaler è un altro contatore che viene alimentato dal clock di sistema e che, a sua volta, alimenta il timer in questione. Esso ha a disposizione una determinata quantità di bit (tipicamente sono 8 o 10) e viene incrementato direttamente dal clock;

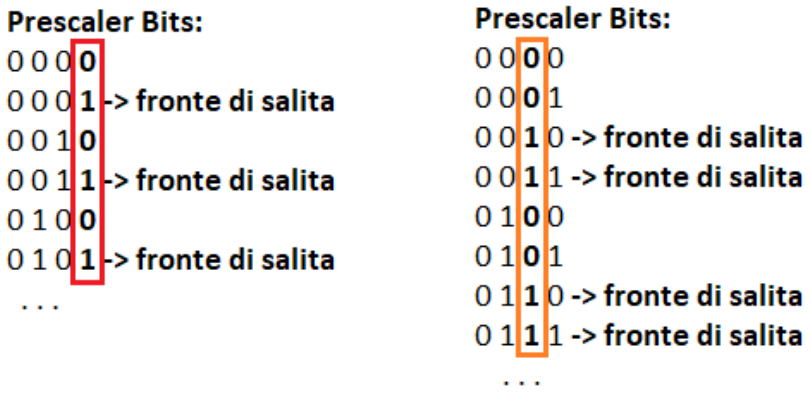


Il timer viene collegato ad uno dei bit del prescaler, questo significa che il timer riceverà le variazioni di uno e zero rispetto ai fronti di salita di uno dei bit a scelta del prescaler, il prescaler a sua volta riceverà le variazioni rispetto ai fronti di salita del clock.



Il prescaler aumenta i suoi bit quando si presenta il fronte di salita del clock, con una certa frequenza: nel primo fronte di salita il bit va da 0000 a 0001, nel secondo va da 0001 a 0010, nel terzo da 0010 a 0011, etc… Quindi **il prescaler si incrementa con la stessa velocità del system clock.**

Se invece il timer si incrementa a seconda di un bit del prescaler scelto riceverà il fronte di salita con una frequenza scalata rispetto a quella del system clock: se si sincronizza con il primo bit del prescaler, il valore non cambia per ogni tempo di clock ma varia con una frequenza che è praticamente alla metà della velocità del system clock:



Più si va verso sinistra e più il prescaler si muove più lentamente, ovvero ad una frequenza sempre più rallentata. Quindi l'idea è quella di connettere il timer ad un bit del prescaler a seconda dell'esigenza: ogni bit darà uno scaling diverso di un fattore 2x.

Analizziamo i vantaggi e gli svantaggi nell'utilizzo del prescaler:
Vantaggio:

- √ **Rallenta il timer facendo raggiungere il tempo massimo più lentamente.** Più il bit del prescaler scelto è alto e più lentamente il timer raggiungerà il tempo massimo misurabile.

Svantaggio:

- ❖ **Diminuisce la granularità (la precisione) del tempo minimo che il timer riesce a percepire.** Più è alto il bit scelto e meno il timer sarà preciso

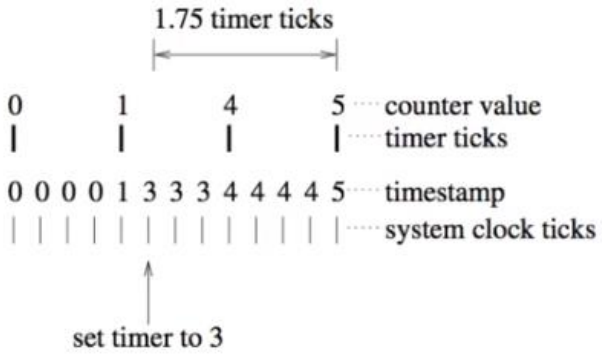
Esempio:
Un timer che lavora a 1MHz a 8 bit raggiunge il massimo valore in 255 microsecondi. Con un prescale con fattore 2 avanziamo ogni due cicli di clock quindi il tempo misurato arriva fino a 511 microsecondi ma la granularità minima che il tempo riesce a percepire è di 2 microsecondi;
NB: il timer resta sempre a 8bit e quindi avrà sempre un range che va da 0 a 255 ma i valori di questo range vengono raggiunti più lentamente: 511 microsecondi invece di 255).
Facciamo un esempio quantitativamente più esteso: con un prescale di 1024 (decimo bit) raggiungeremo il valore massimo in 260milliscondi con una risoluzione minima di 1ms.

In poche parole, **aumenta la durata ma diminuisce la risoluzione;** quindi il bit deve essere selezionato a seconda dello scopo: se interessa una maggiore durata allora si può trascurare la risoluzione e scegliere dei bit del prescaler elevati, se invece l'interesse è proiettato sulla granularità (quindi, in altre parole, interessa quanto sia "preciso" il timer, magari perché si necessita di un timestamp molto accurato) allora si scelgono bit del prescaler molto bassi.

NB: Chiaramente, il prescaler aumenta la frequenza che alimenta il timer, quindi se si sta cercando di diminuirla il prescaler non potrà mai esser d'aiuto!

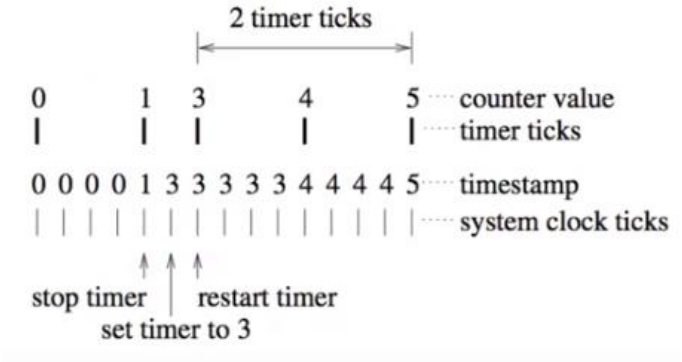
Se si è sincronizzati con un determinato valore scalato del prescaler, quando si cambia un valore del timer (ovvero si cambia una valore nel suo registro ad insaputa di quest'ultimo) e si vuole che esso continui a contare dal valore assegnatogli si hanno delle conseguenze che non sono istantanee

Esempio: ho un fattore di scala 4 e voglio che il timer riprenda il conteggio dal valore 3, per qualsiasi motivo. Quello che accade è che il passaggio da 3 a 4 non avviene con la solita cadenza del timer poiché il valore è stato cambiato durante l'esecuzione del timer.



Come si può osservare dall'immagine, il 3 è stato settato quando il timer era appena passato dal valore 0 al valore1. Come si può vedere, il 3 ha una durata inferiore rispetto allo 0, al 4 e al 5 e ciò è avvenuto perché il cambiamento del valore nel timer è avvenuto durante l'esecuzione e quindi il tempo a disposizione è inferiore.

Non vi è una particolare soluzione a questo problema, la politica che risolve tale problema è il reset del timer, in questo modo il valore assegnatogli dura il numero corretto di tick del clock, tuttavia questa soluzione "butta" il tempo precedentemente accumulato.



Riassumendo:
Il timer può essere visto come un contatore avente un registro che conta il valore attuale e che a seguito di un impulso esegue determinate istruzioni (si avvia, si interrompe, si resetta…); questo può essere sincronizzato al clock o ritardato di un opportuno fattore di scala attraverso il prescaler che aumenta il massimo tempo misurabile dal timer ma diminuisce la precisione di quest'ultimo.

3. External pulse timer

Vi sono delle varianti di timer che non devono per forza essere comandati da un segnale del clock o da un segnale pre-scalato, quindi dà segnali di natura temporali, esistono timer che agiscono a seconda degli impulsi esterni.

Il timer potrebbe avanzare al verificarsi di una condizione esterna: esso si incrementa sul fronte di salita di un segnale esterno piuttosto che sul fronte di salita del clock interno (anche perché non fa distinzioni tra i due segnali, si incrementa a prescindere da quale segnale gli venga dato).

Il segnale esterno che incrementa il timer deve essere più ampio del tempo di clock altrimenti non riusciremmo a campionarlo in maniera appropriata.

4. External Crystal

Un altro metodo per alimentare il timer è attraverso un cristallo di clock, ovvero un clock esterno che produce un segnale oscillando ad una certa frequenza in maniera indipendente rispetto al clock interno del microprocessore. Questa implementazione è utile nei sistemi Real Time in cui il tempo che viene scandito è esterno dal clock di sistema e non viene influenzato da quest'ultimo [implementazione del Real-Time Clock -> RTC)

Utilizzo del timer: input capture e output capture

Il contatore quindi incrementa o decrementa il valore che riceve e salva temporaneamente nel registro. Vi sono due possibili utilizzi del timer (o contatore): **Input Capture** e **Output Capture**

Input capture

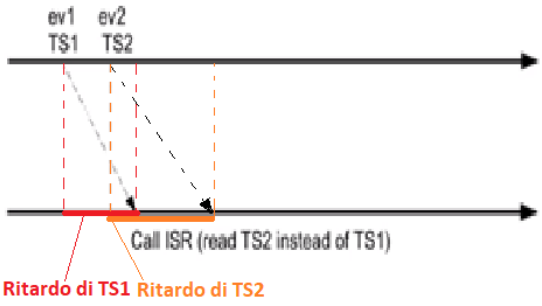
L'utilizzo di un timer in input più comune è quello per la **memorizzazione del timestamp** non appena arriva un determinato segnale aleatorio in entrata al microcontrollore.
Arriva un segnale dal mondo esterno e il timer verrà adoperato per memorizzare l'istante in cui esso arriva.

Funzionamento:
Il funzionamento è semplice: il timer avanza durante l'evoluzione del sistema, arriva un segnale esterno (se viene dal mondo esterno) o interno (se viene dal microcontrollore stesso) e appena esso arriva il timer segna il tempo di arrivo e "marchia" il segnale con il tempo ad esso riferito. Il timestamp viene memorizzato temporaneamente in un registro e, a questo punto, il microcontrollore avanza a seconda del compito da svolgere: tipicamente, viene lanciato un interrupt routine che avvia una routine (una procedura) nella parte del codice addetta alla gestione dell'informazione "marchiata".

Non tutti i pin sono tanto sensibili da segnare con massima accuratezza il timestamp esatto, per questo motivo esistono pin addetti, aventi una determinata precisione.

In questo caso allora si deve scegliere un fattore di scaling del prescaler quando più piccolo possibile poiché minore è il tempo massimo misurabile da parte del timer e maggiore darà la granularità di quest'ultimo (quindi maggiore sarà la precisione per il timestamp)

Problematiche:
La memorizzazione di un timestamp nel registro del timer è pressocché immediata poiché viene memorizzato nell'istante in cui viene captato, tuttavia deve poi essere avviata una **routine** (un frammento di codice), essa dovrà richiamare dei parametri e magari interrompere l'esecuzione di un altro codice in esecuzione e questi passaggi richiedono del tempo: **il tempo di gestione della chiamata della procedura**, ed inducono ad un ritardo.



Se il timestamp TS1 subisce un ritardo e nel frattempo avviene un evento timestamp TS2, anche TS2 subirà un ritardo nella sua gestione;
ciò che praticamente accade è che nel registro in cui è presente il valore TS1 viene sovrascritto il valore TS2 e quindi la routine chiamata da TS1 non andrà più a leggere il valore di TS1, bensì quello di TS2.
Conclusa la routine per TS1, inizierà quella per TS2 (anch'essa in ritardo). Tuttavia, conclusa anche quest'ultima si scoprirà che le routine di TS1 e di TS2 torneranno lo stesso valore timestamp, poiché durante l'esecuzione della routine per TS1 era già stato memorizzato TS2.

Questo problema si può **quantizzare:** il ritardo introdotto dal microcontrollore per processare le varie chiamate è un tempo ben noto e caratteristico di quest'ultimo, quindi si ha sempre un'idea sui ritardi possibili; inoltre, seppur i timestamp vengano catturati in maniera asincrona, in quanto gli eventi possono avvenire in maniera aleatoria, per sommi capi il progettista deve essere in grado di stimare quando dovrebbe avvenire la memorizzazione del timestamp, seppur in maniera grossolana.

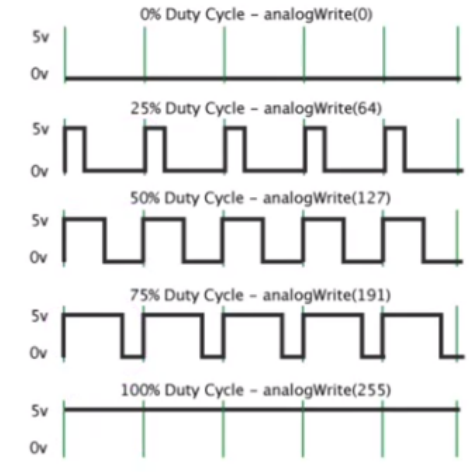
Inoltre, a seconda del problema da risolvere, potrebbe anche essere irrilevante distinguere il timestamp attuale con quello successivo, dunque questo problema non ostacolerebbe il corretto funzionamento del microcontrollore.

Output capture (Output Compare)

La controparte dell'input capture è l'**output compare:** quando il counter del timer raggiunge una certa soglia, si deve dare in output un certo valore. In questo caso viene asserito un segnale verso il mondo esterno quando si raggiunge un certo valore, quindi la cadenza degli avvenimenti non è più aleatoria come per l'utilizzo del timer in input capture: in quest'ultimo utilizzo, infatti, venivano memorizzati i valori del contatore (i timestamp) in funzione dell'arrivo dei segnali dal mondo esterno; utilizzando invece il timer come output compare, i segnali vengono gestiti dal timer stesso che li emette solo dopo aver raggiunto un determinato valore del contatore, in maniera determinata.

Un esempio pratico è il **campionamento**: per campionare un segnale esterno, esso viene scandito grazie ad un timer che ogni periodo temporale scandisce il segnale; in questo caso il timer è di tipo output compare, che scandisce il segnale ricevuto ad un tempo stabilito dal contatore. Altre possibili applicazioni sono le letture temporizzate di determinati valori, l'esecuzione di una routine, l'accensione di una luce con una determinata cadenza, etc…

Quindi **il timer output compare può essere utilizzato sia per comunicare con l'esterno, mandando dei segnali temporizzati dal contatore del timer, oppure per la gestione interna di determinati eventi periodici** (come, appunto, il campionamento).

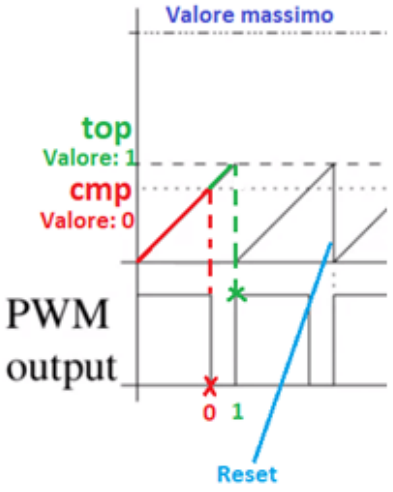


Un utilizzo diffuso di questo timer è quello per la realizzazione del PWM (Pulse Width Modulation), che permetteva la generazione di un determinato valore analogico usando un pin, facendolo switchare tra basso e alto, a seconda del **duty cycle**, convertendo così un segnale da digitale ad analogico. Per implementare questa funzione si necessita di un timer, in particolare di un output compare timer.

La PWM è caratterizzata da:

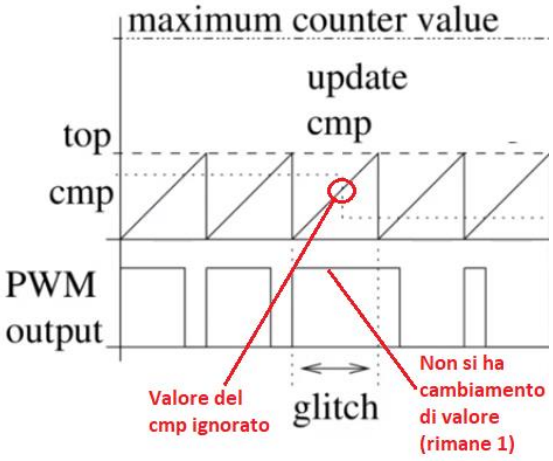
1. **Duty Cycle**: durata del massimo intervallo di tempo (quanto è "largo" l'intervallo temporale)
2. **High-time**: quantità di tempo trascorsa quando il segnale ha valore alto (quanto tempo il pin rimane "acceso" per stabilire il valore alto)

Funzionamento:
Si utilizza un timer configurato secondo le caratteristiche del sistema in cui opera, tale timer è caratterizzato da due **soglie**: **top** e **compare (CMP);**
Il timer può supportare fisicamente un **valore massimo**, oltre ad esso non supporta alcun valore, chiaramente il valore soglia top è inferiore a quest'ultimo.
Quando il segnale raggiunge il valore **top** in uscita viene trasmesso il valore **1**, dopo di che si resetta; quando viene raggiunto **compare** viene trasmesso il valore **0**.

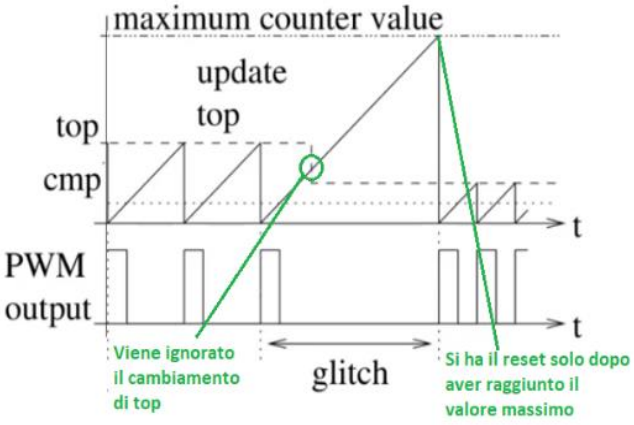


Supponiamo (solo a titolo d'esempio) di partire dal valore 1, successivamente avremo un reset del timer, che ripartirà a zero; sale fino a raggiungere il valore CMP (compare) e in output vi sarà il valore 0, dopo di che sale fino al valore top, in output vi sarà 1 e il timer si resetta.

La scelta sulla grandezza di top e di compare danno praticamente i valori caratteristici del PWM: **la grandezza di top ci da la durata del duty cycle,** sapendo fino a quanti bit si può contare. Questa durata si può incrementare con un prescaler; **scegliendo la soglia CMP si decide quanto tempo il microcontrollore rimane acceso.**



Il valore del cmp può anche essere aggiornato dinamicamente, ciò che avviene è, praticamente, il cambiola "soglia di "accensione".
Dopo aver raggiunto il valore top il timer si resetta, se quando sta per raggiungere il valore di CMP (e quindi l'output sta passando da 1 a 0), quest'ultimo viene cambiato con un valore più basso, si perde l'informazione e non si ha lo switch tra bit;
Come si può notare in figura, dopo aver raggiunto nuovamente la soglia top, il timer si resetta e non perde più il valore cmp, switchando a 0 non appena lo raggiunge.
Quindi l'effetto dell'abbassamento del cmp si può osservare solo dopo che il timer si è resettato, questo evento si chiama glitch.
Si può anche aggiornare dinamicamente il valore top, inserendo un valore più basso, il timer non incontrerà mai il valore top. Per essere resettato allora **il timer dovrà raggiungere il valore massimo del contatore**, anche questo evento prende il nome di glitch.
Il glitch è praticamente un malfunzionamento temporaneo imprescindibile dovuto ad un aggiornamento dinamico.



Watchdog timer

Il Watchdog timer è un particolare tipo di timer che decrementa il contenuto del proprio contatore. Nel momento in cui raggiunge il valore 0, esso **si occupa di resettare il microcontrollore e serve ad evitare che quest'ultimo entri in loop rimanga malfunzionante per tempi indeterminati**. Fintanto che il microcontrollore rimane operativo, senza alcun malfunzionamento, si occuperà, tra le istruzioni da eseguire, anche di incrementare il contatore del Watchdog timer, impostando al valore massimo e facendolo ripartire.

Chiaramente, il Watchdog timer deve avere un clock separato da quello del microcontrollore, deve "vivere" in maniera assestante, così che, ad esempio, se il microcontrollore va in risparmio energetico e ferma il clock, il Watchdog non si blocchi ma continui a lavorare.

Esso è il "punto critico" del microcontrollore: qualora non funzionasse o rimanesse disattivato, il microcontrollore potrebbe rischiare di rimanere bloccato e di non poter essere mai più sbloccato