USRS26
Cours 6 Timers

Matthias Puech

Master 1 SEMS — Cnam

Qu'est-ce qu'un *timer* ?

C'est un compteur asservi à une horloge interne ou externe, qui peut déclencher des évenements selon sa valeur.

Qu'est-ce qu'un *timer* ?

C'est un compteur asservi à une horloge interne ou externe, qui peut déclencher des évenements selon sa valeur.

C'est *la* source de temps pour nos programme.

Qu'est-ce qu'un *timer* ?

Plus précisément

C'est un périphérique qui comprend :

- un registre R de n bits
  (n est la *résolution* du timer)
- une entrée "horloge" externe
  (e.g., l'horloge d'un bus du système)
- ... qui alimente un diviseur d'horloge S
  (appelé *prescaler*)
- ... qui incrémente le registre R
- quand R atteint une valeur P (sa période), il est remis à 0
  (débordement ou *overflow*)
- une sortie "IRQ", qui est 1 quand le registre déborde
  (reliée au contrôleur d'interruption)

Fréquence d'un timer

Quiz

Soit un timer :

- alimenté par une horloge de $F_c = 4\,\text{MHz}$,
- de période $P = 400$
- avec un prescaler de $S = 10$

Quelle est la fréquence à laquelle il débordera ?

Fréquence d'un timer

Quiz

Soit un timer :

- alimenté par une horloge de $F_c = 4\,\text{MHz}$,
- de période $P = 400$
- avec un prescaler de $S = 10$

Quelle est la fréquence à laquelle il débordera ?

$\approx 1kHz$

Fréquence d'un timer

Quiz

Soit un timer :

- alimenté par une horloge de $F_c = 4\,\text{MHz}$,
- de période $P = 400$
- avec un prescaler de $S = 10$

Quelle est la fréquence à laquelle il débordera ?

$\approx 1kHz$

Fréquence F d'un timer

$$F = \frac{F_c}{(P-1)(S-1)}$$

Fréquence d'un timer

Quiz

Quelle est la période maximum d'un timer :

- alimenté par une horloge de $F_c = 4MHz$
- de résolution 16 bits
- avec un prescaler codé sur 16 bits aussi

Fréquence d'un timer

Quiz

Quelle est la période maximum d'un timer :

- alimenté par une horloge de $F_c = 4MHz$
- de résolution 16 bits
- avec un prescaler codé sur 16 bits aussi

$\approx$ 16 minutes

Timers sur STM32

Bien plus complexes que ce que l'on vient de décrire
(le plus complexe de tous les périphériques)

Timers sur STM32

Bien plus complexes que ce que l'on vient de décrire
(le plus complexe de tous les périphériques)

Quelques caractéristiques

- choix de l'entrée horloge
  - bus système APB (dont la fréquence est réglable avec le RCC)
  - pin externe
- choix de l'évenement à déclencher quand *overflow*
  - interruption
  - requête DMA
- calcul de fréquence d'un évenement externe
  (mode *input capture*)
- génération d'un signal PWM sur plusieurs canaux
  (modes *output compare* et *PWM*)
- démarrage contrôlé pour génération de délais (mode *one pulse*)

Timers sur STM32

Trois catégories de timers

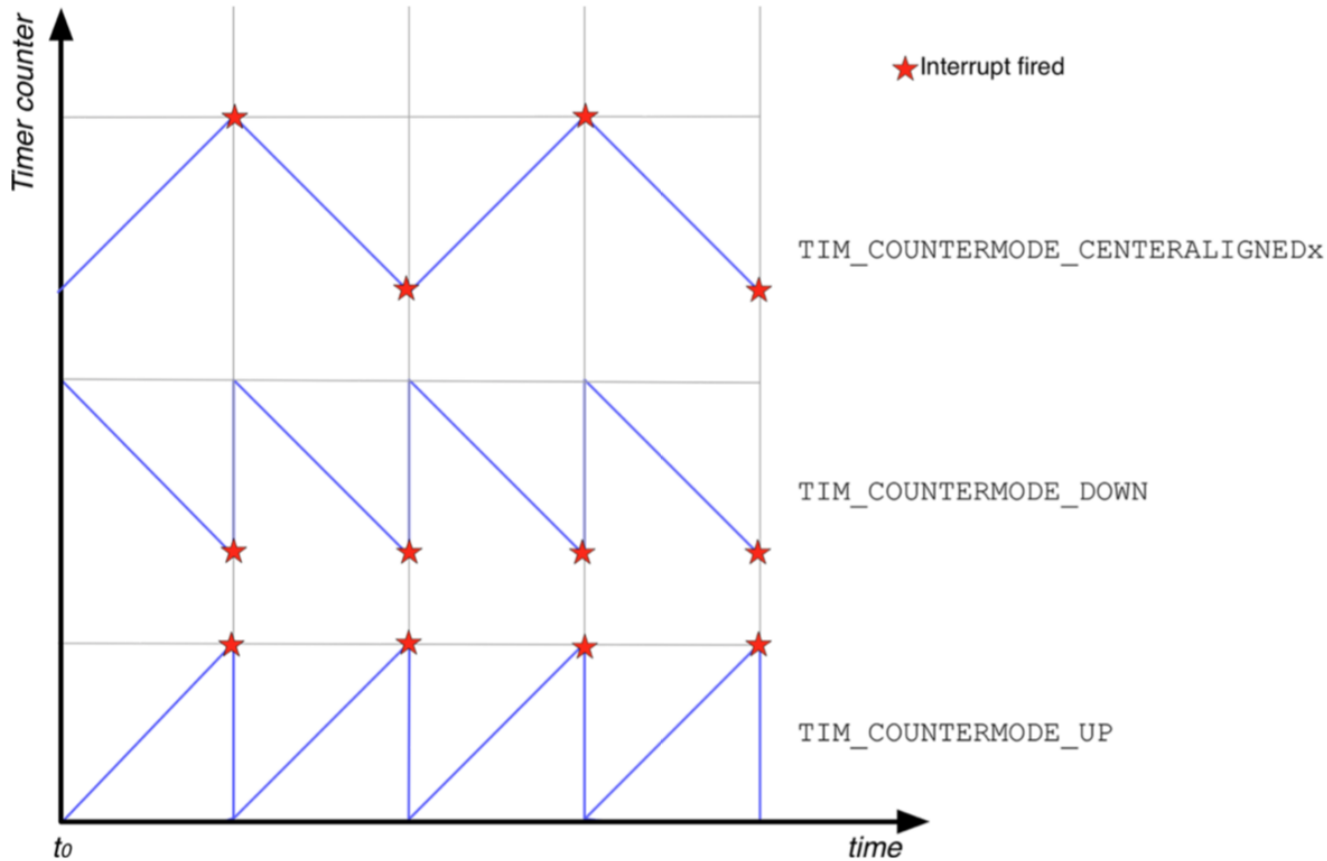
- SysTick
  (commun à tous les Cortex-M)
- Basic timers
  (génère des interruptions ou des requêtes DMA)
- General purpose timers
  (Basic + output compare + input capture + sensor intf)
- Advanced timers
  (pour le contrôle de moteurs et d'alimentations)
- High Resolution timers
  (précision de l'ordre de la picoseconde)

Timers sur STM32

Modes de comptage

Trois modes de comptage différents :

(disponible uniquement sur les timers > General Purpose)



Sur nos cartes

Les STM32F30x ont 13 timers :

SysTick	que l'on connaît déjà
TIM6, TIM7	Basic timer (16 bits)
TIM3, TIM4, TIM15, TIM16, TIM17	General Purpose (16 bits)
TIM2	General Purpose (32 bits)
TIM1, TIM8, TIM20	Advanced
HRTIM1	High Resolution

HAL Configuration des timers basiques

cf. `stm32f3xx_hal_tim.{c|h}`

- Activer l'horloge du périphérique
  `(__HAL_RCC_TIM`*x*`_CLK_ENABLE();)`

HAL Configuration des timers basiques

cf. `stm32f3xx_hal_tim.{c|h}`

- Activer l'horloge du périphérique
  (`__HAL_RCC_TIMx_CLK_ENABLE();`)
- Déclarer une *handle* `TIM_HandleTypeDef` et remplir ses champs de configuration :
  - `Instance` : le timer à utiliser (`TIMx`)
  - `Init.Period` : sa période
  - `Init.Prescaler` : la valeur du diviseur d'horloge
  - `Init.CounterMode` : mode de comptage
    (up, down, ou up+down)

HAL Configuration des timers basiques

cf. `stm32f3xx_hal_tim.{c|h}`

- Activer l'horloge du périphérique
  (`__HAL_RCC_TIMx_CLK_ENABLE();`)
- Déclarer une *handle* `TIM_HandleTypeDef` et remplir ses champs de configuration :
  - `Instance` : le timer à utiliser (`TIMx`)
  - `Init.Period` : sa période
  - `Init.Prescaler` : la valeur du diviseur d'horloge
  - `Init.CounterMode` : mode de comptage (up, down, ou up+down)
- Passer cette *handle* à `HAL_TIM_Base_Init()`

HAL Configuration des timers basiques

cf. `stm32f3xx_hal_tim.{c|h}`

- Activer l'horloge du périphérique
  (`__HAL_RCC_TIMx_CLK_ENABLE();`)
- Déclarer une *handle* `TIM_HandleTypeDef` et remplir ses champs de configuration :
  - `Instance` : le timer à utiliser (`TIMx`)
  - `Init.Period` : sa période
  - `Init.Prescaler` : la valeur du diviseur d'horloge
  - `Init.CounterMode` : mode de comptage
    (up, down, ou up+down)
- Passer cette *handle* à `HAL_TIM_Base_Init()`
- Appeler une des fonctions pour activer le timer :
  - `HAL_TIM_Base_Start()` : démarrage simple
  - `HAL_TIM_Base_Start_IT()` : + armement de l'interruption
    (dans ce cas, ne pas oublier d'activer l'IRQ dans le NVIC :
    `HAL_NVIC_EnableIRQ(TIMx_IRQn)`)

HAL Utilisation des timers basiques

cf. `stm32f3xx_hal_tim.{c|h}`

- on peut lire la valeur du registre d'un timer avec `__HAL_TIM_GET_COUNTER(&htim)`

HAL Utilisation des timers basiques

cf. `stm32f3xx_hal_tim.{c|h}`

- on peut lire la valeur du registre d'un timer avec `__HAL_TIM_GET_COUNTER(&htim)`
- on définit le *handler* d'interruption, qui passe la main au *handler* HAL :

```
void TIM3_IRQHandler() {
  HAL_TIM_IRQHandler(&htim3);
}
```

- ... qui appellera un *callback* que l'on peut définir :

```
void HAL_TIM_PeriodElapsedCallback(TIM_HandleTypeDef *h)
```

quand le timer débordera.

Application Modulation de largeur d'impulsion

Une utilisation courante des timers :
la *modulation de largeur d'impulsion* (PWM)

Application Modulation de largeur d'impulsion

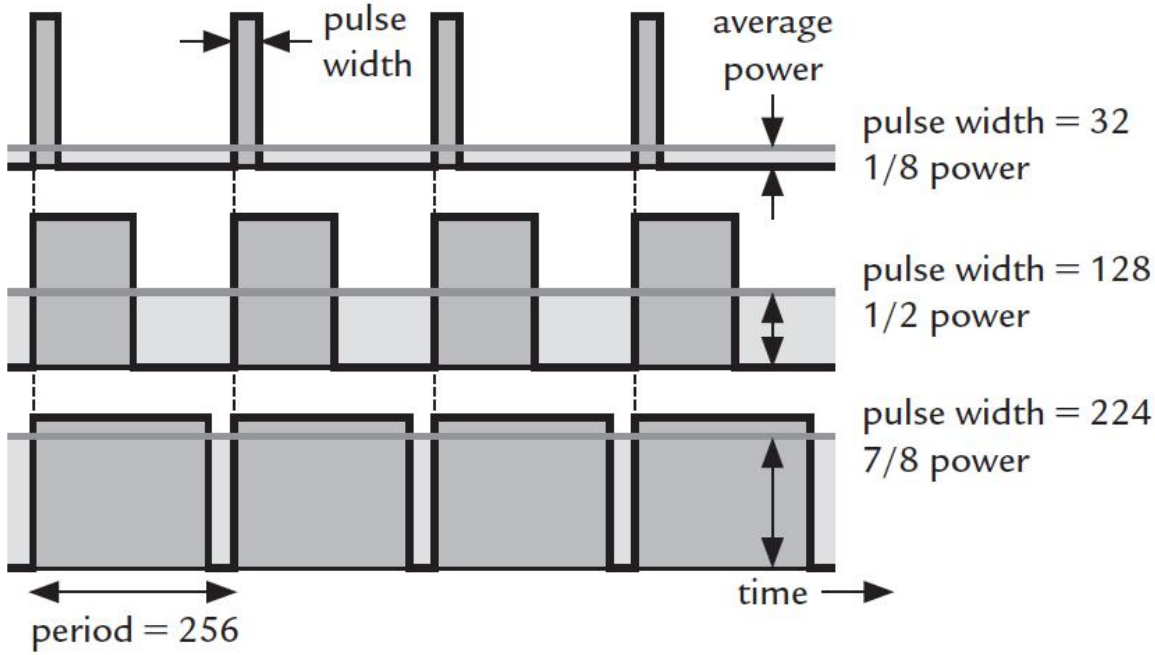
Une utilisation courante des timers :
la *modulation de largeur d'impulsion* (PWM)

Definition (partielle)

Technique d'encodage d'un signal continu en un signal numérique par variation de la largeur d'un signal périodique rectangulaire
(on/off, réalisable par un simple interrupteur)

On reconstruit le signal en prenant la *moyenne* du signal encodé

Vocabulaire



fréquence de porteuse f_C (*switching frequency*) fréquence du signal périodique rectangulaire

largeur d'impulsion (*duty cycle/pulse width PW*) ratio de la période de la porteuse où le signal est haut

résolution nombre de valeurs que peut prendre la largeur d'impulsion

reconstruction action de lisser/moyenner le signal encodé

Propriétés

Utilisations

- alimentations à découpage
  (PW=ratio de conversion, reconstruction par filtres analogiques)
- amplificateurs audio
  (*class D* ; f_C de l'ordre du MHz)
- contrôle de moteurs
- variation de l'intensité d'une LED
  (la reconstruction est faite par le temps de réponse et/ou par la persistence rétinienne)

Propriétés

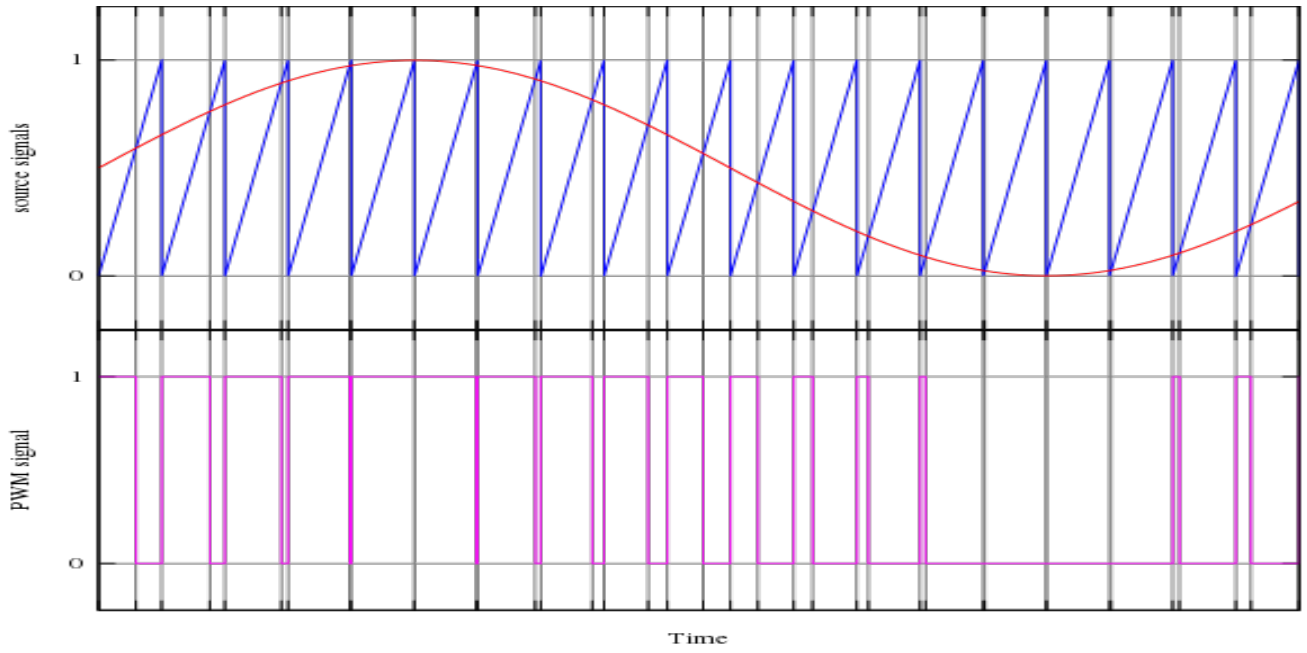
Utilisations

- alimentations à découpage
  (PW=ratio de conversion, reconstruction par filtres analogiques)
- amplificateurs audio
  (*class D* ; f_C de l'ordre du MHz)
- contrôle de moteurs
- variation de l'intensité d'une LED
  (la reconstruction est faite par le temps de réponse et/ou par la persistence rétinienne)

Théorème (échantillonage PWM)

On peut reconstruire parfaitement n'importe quel signal "continu" encodé en PWM si et seulement si la fréquence de porteuse f_c est supérieure à 1.56 × la plus haute fréquence contenue dans le signal continu.

Implémentation PWM par intersection



- générer un signal en dent de scie de fréquence f_C (en bleu) à la fréquence d'échantillonnage $f_s = f_c \times n$
  (n est la résolution)
- comparer le signal à encoder et la dent de scie :
  - si $>$, émettre 1
  - si $<$, émettre 0

Implémentation PWM par intersection

Génération de signal en dent de scie

On peut représenter un signal par un entier non signé 16 bits :

- valeur minimale du signal = 0
- valeur maximale = `INT16_MAX`

et se servir de l'*overflow* des entiers machine pour le retour à 0.

$\rightsquigarrow$ le *phazor*

Implémentation PWM par intersection

Génération de signal en dent de scie

On peut représenter un signal par un entier non signé 16 bits :

- valeur minimale du signal = 0
- valeur maximale = `INT16_MAX`

et se servir de l'*overflow* des entiers machine pour le retour à 0.

$\rightsquigarrow$ le *phazor*

```
uint16_t s = 0,
uint16_t increment = fc * INT16_MAX / fs;
uint16_t phazor() {
  s += increment;
  return s;
}
```