

Les Timers

Timers sur ESP32

Fondamentalement, un Timer (minuterie) est une horloge, qui est utilisée pour mesurer et contrôler les événements temporels. offrant une temporisation précise. La plupart des microcontrôleurs ont des minuterie intégrées. Les minuterie des microcontrôleurs ne sont pas seulement utilisées pour générer des retards, mais sont également utilisées comme compteur. Cette caractéristique du temporisateur est utilisée pour de nombreuses applications. Les minuterie du microcontrôleur sont contrôlées par des registres de fonctions spéciales qui sont affectés aux opérations de minuterie.

Une interruption est un événement externe qui interrompt le programme en cours et exécute une routine de service d'interruption (ISR).

Une fois l'ISR terminé, le programme en cours se poursuit avec l'instruction suivante. Et les interruptions du temporisateur sont les interruptions générées par le temporisateur. Voici l'exemple montrant comment configurer le temporisateur pour générer périodiquement une interruption et comment la gérer. ESP32 a deux groupes de minuterie, chacun avec deux minuterie matérielles à usage général. Tous les temporisateurs sont basés sur des compteurs 64 bits et des prédiviseurs 16 bits. Le prédiviseur est utilisé pour diviser la fréquence du signal de base (généralement 80 MHz), qui est ensuite utilisé pour incrémenter ou décrémenter le compteur de la minuterie. La variable de compteur sera partagée entre la boucle principale et l'ISR, puis elle doit être déclarée avec le mot-clé volatile .

```
volatile int interruptCounter ;
```

Nous aurons un compteur supplémentaire pour suivre le nombre d'interruptions déjà survenues.

```
int totalInterruptCounter ;
```

Afin de configurer le timer, nous aurons besoin d'un pointeur vers une variable de type hw_timer_t .

```
hw_timer_t * timer = NULL;
```

Enfin, nous devons déclarer une variable de type portMUX_TYPE qui nous servira à nous occuper de la synchronisation entre la boucle principale et l'ISR.

```
portMUX_TYPE timerMux = portMUX_INITIALIZER_UNLOCKED;
```

Pour initialiser le timer à l'aide d'une fonction timerbegin , cette fonction reçoit le numéro du timer que l'on souhaite utiliser (de 0 à 3, puisque nous avons 4 timers matériels), la valeur du prescaler et un drapeau indiquant si le compteur doit compter (vrai) ou vers le bas (faux).

```
timer = timerBegin(0, 80, vrai);
```

Pour cet exemple, nous utiliserons la première minuterie et passerons vrai au dernier paramètre, donc le compteur compte la fréquence du signal de base utilisé par les compteurs ESP32 est de 80 MHz . Si nous divisons cette valeur par 80 (en utilisant 80 comme valeur du prescaler), nous obtiendrons un signal avec une fréquence de 1 MHz qui incrémentera le compteur de la minuterie 1

000 000 fois par seconde.

Avant d'activer le temporisateur, nous devons le lier à une fonction de traitement, qui sera exécutée lorsque l'interruption sera générée. Cela se fait avec un appel à la fonction `timerAttachInterrupt` .

```
timerAttachInterrupt(timer, &onTimer, true);
```

Cette fonction reçoit en entrée un pointeur vers le timer initialisé, que nous avons stocké dans notre variable globale, l'adresse de la fonction qui va gérer l'interruption et un drapeau indiquant si l'interruption à générer est front (`true`) ou level (`false`) . Pour cet exemple, nous passerons notre variable de minuterie globale en première entrée, en deuxième l'adresse d'une fonction appelée `onTimer` que nous spécifierons plus tard, et en troisième la valeur `true` , donc l'interruption générée est de type `edge`.

```
timerAlarmWrite(timer, 1000000, true);
```

fonction `timerAlarmWrite` pour spécifier la valeur du compteur dans laquelle l'interruption du temporisateur a été générée. Donc, pour cet exemple, on suppose que l'on veut générer une interruption chaque seconde, et on passe donc la valeur de 1 000 000 microsecondes, qui est égale à 1 seconde. Le troisième argument que nous passerons `true` , donc le compteur se rechargera et donc l'interruption sera périodiquement générée. Pour terminer la fonction de configuration en activant un appel à `timerAlarmEnable(timer)`;

Main loop

La boucle principale (`main loop`) sera l'endroit où nous gérons réellement l'interruption de la minuterie, après qu'elle ait été signalée par l'ISR (routine de service d'interruption également appelée gestionnaire d'interruption) . Pour vérifier la valeur du compteur d'interruptions, nous allons donc vérifier si la variable du compteur d'interruptions est supérieur à zéro et si c'est le cas, nous entrerons le code de gestion des interruptions. Là, la première chose que nous allons faire est de décrémenter ce compteur, signalant que l'interruption a été acquittée et sera traitée.

1.ino

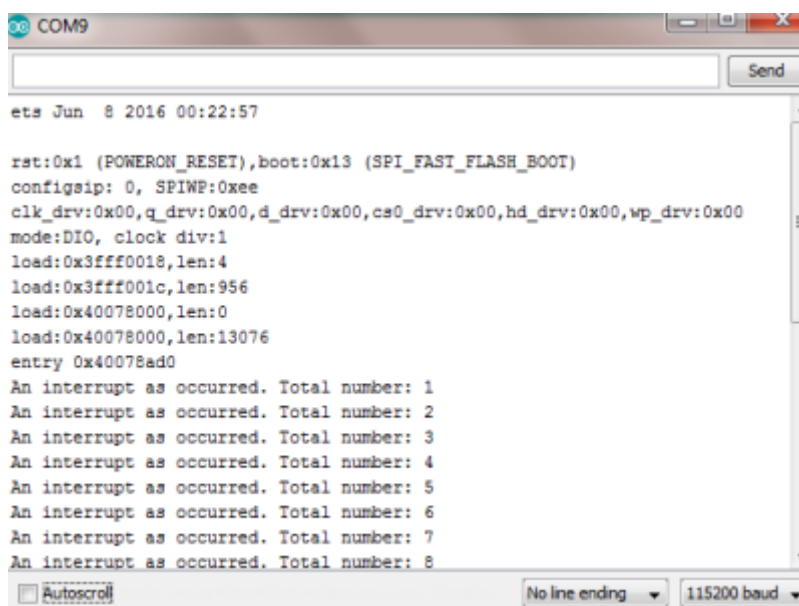
```
if (interruptCounter > 0) {  
  
    portENTER_CRITICAL(&timerMux);  
    interruptCounter--;  
    portEXIT_CRITICAL(&timerMux);  
  
    totalInterruptCounter++;  
  
    Serial.print("An interrupt as occurred. Total number: ");  
    Serial.println(totalInterruptCounter);  
  
}
```

La fonction ISR doit être une fonction qui renvoie void et ne reçoit aucun argument. La routine de gestion des interruptions doit avoir l'attribut IRAM_ATTR, pour que le compilateur place le code dans IRAM. De plus, les routines de gestion des interruptions ne doivent appeler que les fonctions également placées dans l'IRAM.

2.ino

```
void IRAM_ATTR onTimer() {  
  portENTER_CRITICAL_ISR(&timerMux);  
  interruptCounter++;  
  portEXIT_CRITICAL_ISR(&timerMux);  
}
```

Puisque cette variable est partagée avec l'ISR, nous le ferons dans une section critique, que nous spécifions en utilisant une macro portENTER_CRITICAL et une macro portEXIT_CRITICAL. Ces deux appels reçoivent en argument l'adresse de notre variable globale portMUX_TYPE. La gestion réelle des interruptions consistera simplement à incrémenter le compteur avec le nombre total d'interruptions survenues depuis le début du programme et à l'imprimer sur le port série.



```
COM9  
ets Jun 8 2016 00:22:57  
  
rst:0x1 (POWERON_RESET),boot:0x13 (SPI_FAST_FLASH_BOOT)  
config:0, SPIWP:0xee  
clk_drv:0x00,q_drv:0x00,d_drv:0x00,cs0_drv:0x00,hd_drv:0x00,wp_drv:0x00  
mode:DIO, clock div:1  
load:0x3fff0018,len:4  
load:0x3fff001c,len:956  
load:0x40078000,len:0  
load:0x40078000,len:13076  
entry 0x40078ad0  
An interrupt as occurred. Total number: 1  
An interrupt as occurred. Total number: 2  
An interrupt as occurred. Total number: 3  
An interrupt as occurred. Total number: 4  
An interrupt as occurred. Total number: 5  
An interrupt as occurred. Total number: 6  
An interrupt as occurred. Total number: 7  
An interrupt as occurred. Total number: 8
```

ESP32TimerInterrupt Library

ESP32TimerInterrupt Library

From:

<https://magenealogie.chanterie37.fr/www/fablab37110/> - Castel'Lab le Fablab MJC de Château-Renault

Permanent link:

https://magenealogie.chanterie37.fr/www/fablab37110/doku.php?id=start:arduino:esp32:les_timers&rev=1650783432

Last update: 2023/01/27 16:08

