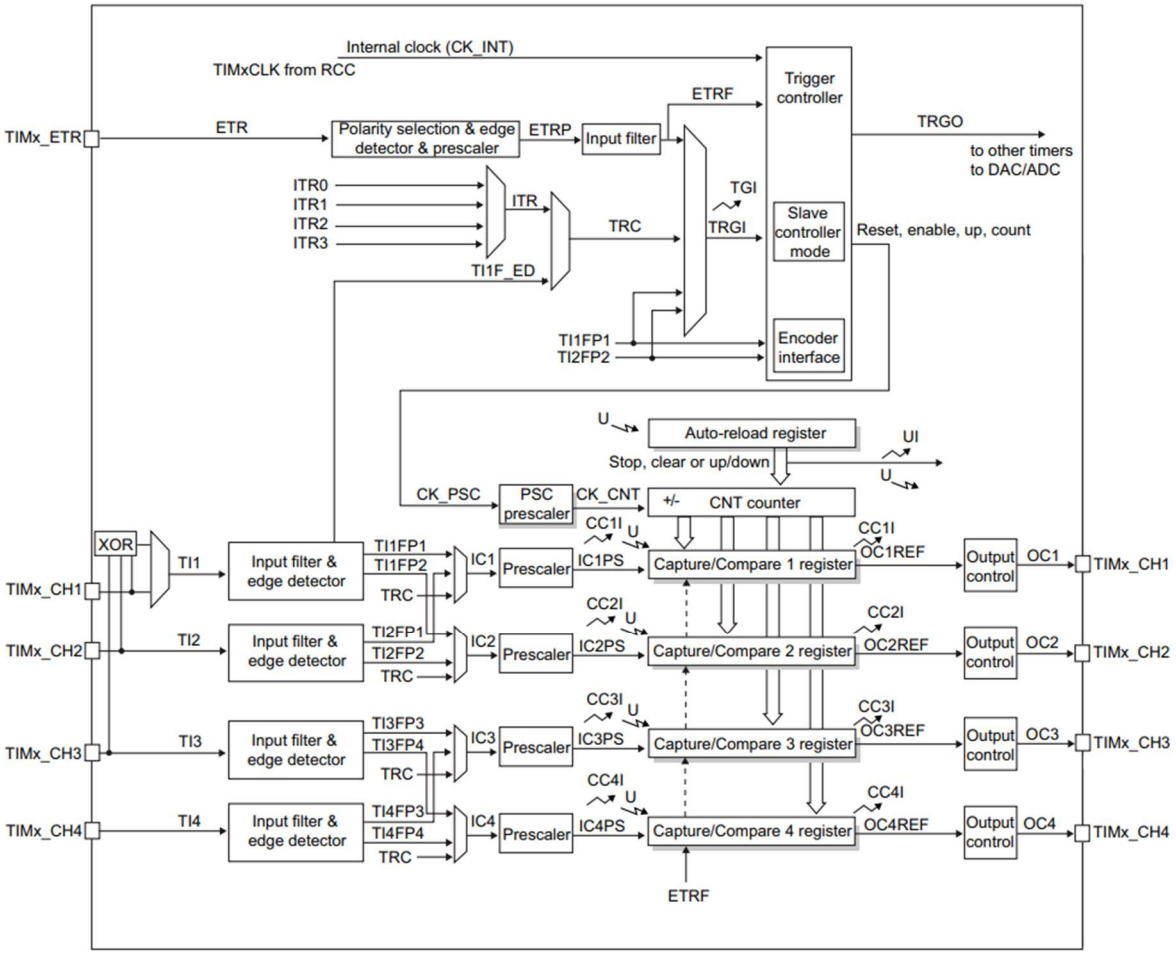
LE DIMNA Mathieu
mathieu.le-dimna@yncrea.fr

Chenillard IT

Objectif

Reprenez le code écrit dans l'exercice précédent, le but est de temporiser l'animation du chenillard par un timer dont le débordement déclenchera une **interruption**.

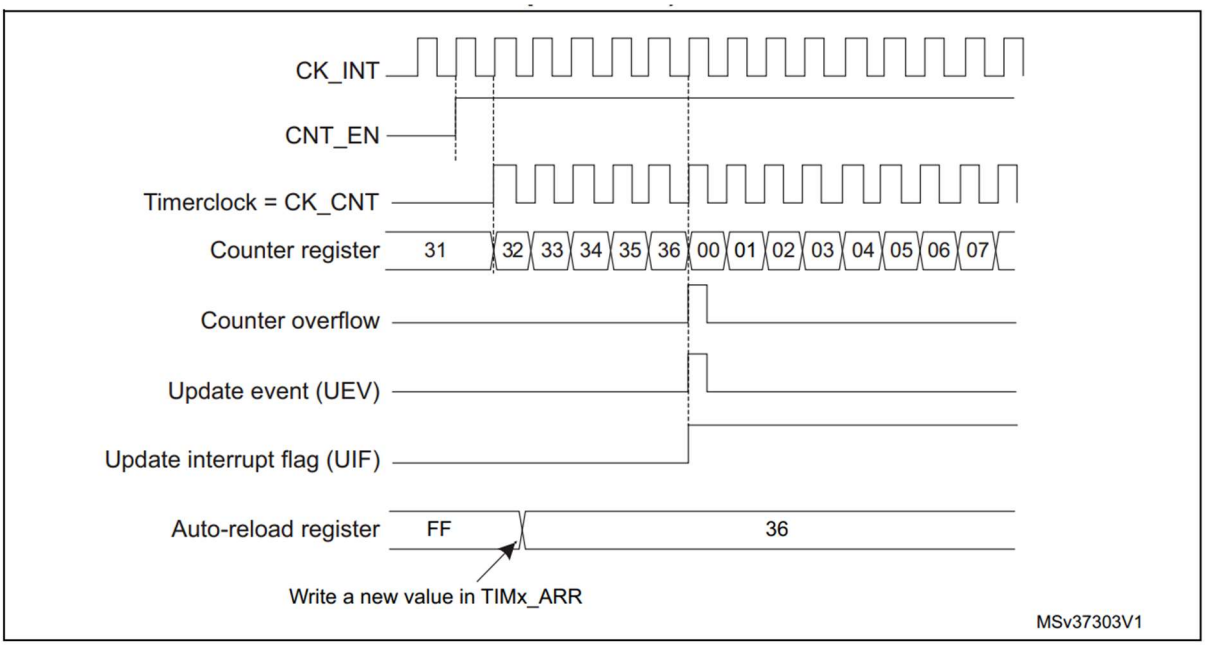
Diagramme de fonctionnement



Le timer est un compteur avancé qui permet de générer divers évènements. Dans notre cas nous avons besoin de générer une interruption à chaque fois que l'on veut allumer une autre LED de notre chenillard. Il est possible de configurer un compteur, qui, lorsqu'il atteint une valeur génère une interruption.

Notez que cette interruption se caractérise par l'exécution d'une fonction dont l'adresse est connue du cortex ARM. Votre code se trouvera dans cette fonction, il vous sera donc possible de dire au processeur ce qu'il doit faire dans ce contexte d'exécution particulier. ***Attention, cette fonction devra être rapide à exécuter.***

Fonctionnement du comptage avec pré-chargement de la valeur dans ARR



Lorsque le compteur est activé (bit *CNT_EN* à 1 dans les registres du timer et horloge active) il compare constamment la valeur de son compteur (*Counter register*) à celle écrite dan le registre *ARR*. Dès lors qu'il y a un dépassement de la valeur dans *ARR* le compteur lève un flag[3] indiquant un débordement.

Fréquence de comptage

La fréquence de comptage est fondamentale pour calculer les différentes valeurs utiles à notre signal PWM. Elle définit le temps que met le timer à incrémenter le registre CNT. *Notez que notre fréquence f_{CK_INT} a été soigneusement définie à 32MHz. La période de comptage (en secondes) est l'inverse de la fréquence de comptage.*

$$f_{CK\_CNT} = \frac{f_{CK\_INT}}{PSC + 1}$$
$$f_{CK\_CNT} : Fréquence\ de\ comptage$$
$$PSC : Prescaler, permet\ de\ diviser\ l'horloge\ d'entrée\ du\ timer$$
$$f_{CK\_INT} : Fréquence\ d'entrée\ du\ timer$$

Fréquence de débordement

Le débordement est la base du timer, la valeur appliquée au registre ARR du timer est comparée à chaque incrément du registre CNT, si celle-ci correspond le registre CNT est remis à zéro. De cette manière le timer exécute son comptage périodiquement.

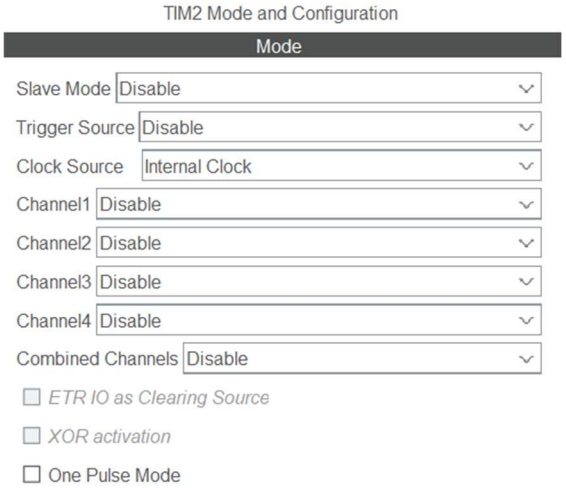
$$t_{ARR} = t_{CK\_CNT} * (ARR + 1)$$
$$t_{ARR} : Période\ de\ rafraichissement\ du\ comptage$$
$$ARR : Registre\ d'auto - rafraichissement$$

[3] Bit dans les registres qui représente l'état d'un périphérique ou un évènement

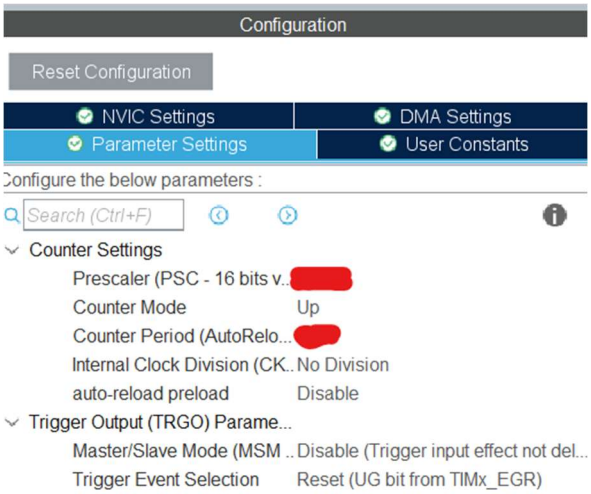
LE DIMNA Mathieu
mathieu.le-dimna@yncrea.fr

Fichier IOC

Allez dans vue du fichier IOC cherchez la configuration des timers et sélectionnez TIM2. Dans les paramètres du timer, paramétrez la source d'horloge comme étant l'horloge interne.



Trouvez dès à présent les valeurs de PSC et ARR à changer pour une période d'animation de **500ms**.



Une fois les valeurs trouvées appelez le professeur pour vérifier la méthode utilisée.

Il faut désormais indiquer au NVIC[4] que l'interruption du timer 2 doit être prise en compte, pour cela allez dans l'onglet ***NVIC Settings*** de la configuration du timer 2. Cochez la case *Enabled*. Maintenant lorsqu'un évènement lié au timer 2 survient le processeur exécutera la fonction associée.

[4] Périphérique du processeur ARM responsable de la gestion des interruptions

Code utilisateur

Dans les fichier d'en-têtes de la librairie HAL (***Drivers > STM32L1xx_HAL_Driver > Inc***) cherchez un fichier en rapport avec les timers. Une fois ouvert observez dans l'*Outline* le contenu de ce fichier.

Fonctions associées aux timers de base

- HAL_TIM_Base_Init(TIM_HandleTypeDef*) : HAL_StatusTypeDef
- HAL_TIM_Base_DeInit(TIM_HandleTypeDef*) : HAL_StatusTypeDef
- HAL_TIM_Base_MspInit(TIM_HandleTypeDef*) : void
- HAL_TIM_Base_MspDeInit(TIM_HandleTypeDef*) : void
- HAL_TIM_Base_Start(TIM_HandleTypeDef*) : HAL_StatusTypeDef
- HAL_TIM_Base_Stop(TIM_HandleTypeDef*) : HAL_StatusTypeDef
- HAL_TIM_Base_Start_IT(TIM_HandleTypeDef*) : HAL_StatusTypeDef
- HAL_TIM_Base_Stop_IT(TIM_HandleTypeDef*) : HAL_StatusTypeDef
- HAL_TIM_Base_Start_DMA(TIM_HandleTypeDef*, uint32_t*, uint16_t) : HAL_StatusTypeDef
- HAL_TIM_Base_Stop_DMA(TIM_HandleTypeDef*) : HAL_StatusTypeDef

Fonctions associées aux évènements des timers de base

- HAL_TIM_PeriodElapsedCallback(TIM_HandleTypeDef*) : void
- HAL_TIM_PeriodElapsedHalfCpltCallback(TIM_HandleTypeDef*) : void
- HAL_TIM_OC_DelayElapsedCallback(TIM_HandleTypeDef*) : void
- HAL_TIM_IC_CaptureCallback(TIM_HandleTypeDef*) : void
- HAL_TIM_IC_CaptureHalfCpltCallback(TIM_HandleTypeDef*) : void
- HAL_TIM_PWM_PulseFinishedCallback(TIM_HandleTypeDef*) : void
- HAL_TIM_PWM_PulseFinishedHalfCpltCallback(TIM_HandleTypeDef*) : void
- HAL_TIM_TriggerCallback(TIM_HandleTypeDef*) : void
- HAL_TIM_TriggerHalfCpltCallback(TIM_HandleTypeDef*) : void
- HAL_TIM_ErrorCallback(TIM_HandleTypeDef*) : void

Observez maintenant le contenu du fichier ***Core > Src > stm32l1xx_it.c***. Vous trouverez la fonction appelée par le processeur une fois qu'un évènement du timer 2 survient. Notez que cette fonction est déjà peuplée d'une fonction de la librairie HAL. Cette fonction va contrôler à notre place les procédures de gestion d'interruption liée au timer, cependant, elle est capable d'exécuter du code depuis votre fichier ***main.c***.

```
201⊖/**
202   * @brief This function handles TIM2 global interrupt.
203   */
204⊖void TIM2_IRQHandler(void)
205 {
206   /* USER CODE BEGIN TIM2_IRQn 0 */
207
208   /* USER CODE END TIM2_IRQn 0 */
209   HAL_TIM_IRQHandler(&htim2);
210   /* USER CODE BEGIN TIM2_IRQn 1 */
211
212   /* USER CODE END TIM2_IRQn 1 */
213 }
```

Pour cela il vous suffit de réécrire dans un fichier une des **fonctions associées aux évènements des timers de base** (vu plus haut). Par exemple, si vous voulez exécuter du code à chaque débordement de timer, réécrivez la fonction *HAL_TIM_PeriodElapsedCallback* en gardant les mêmes paramètres et type de retour.

Dans cette fonction vous écrirez le code qui gère l'animation du chenillard. N'hésitez pas à utiliser l'attribut ***static*** sur une variable dont vous souhaitez garder la valeur entre deux appels de cette fonction par le cortex.

Une fois tout ça préparé, appelez la fonction HAL ***HAL_TIM_Base_Start_IT*** avec l'adresse du handler de timer passé en paramètres.

Pour aller plus loin

Séparez votre *main* et les fonctions/types/defines associées au chenillard. Créez un nouveau dossier sous ***Drivers*** nommé ***Chenillard***. Dans ce dossier créez un fichier source nommé ***chenillard.c*** et un fichier d'en-têtes nommé ***chenillard.h***.

Créez une fonction nommée ***animer_chenillard*** qui contient le code placé précédemment dans la fonction d'interruption. Elle prendra en paramètres votre tableau de LED et la taille du tableau et ne renverra rien. Dans le callback[5] vous appellerez cette nouvelle fonction pour animer votre chenillard.

Note

Le fichier .h contient les inclusions, les types, les prototypes de fonction, les énumérations… Le fichier .c redéfiit les prototypes du .h, il contient le code de ces fonctions et ses variables propres.

[5] Fonction appelée par la librairie HAL que vous avez réécrite plus tôt