Université IBN TOFAIL – Kénitra
ECOLE NATIONALE DES SCIENCES APPLIQUEES

Cours Microcontrôleurs Avancés

CH6: Les Timers

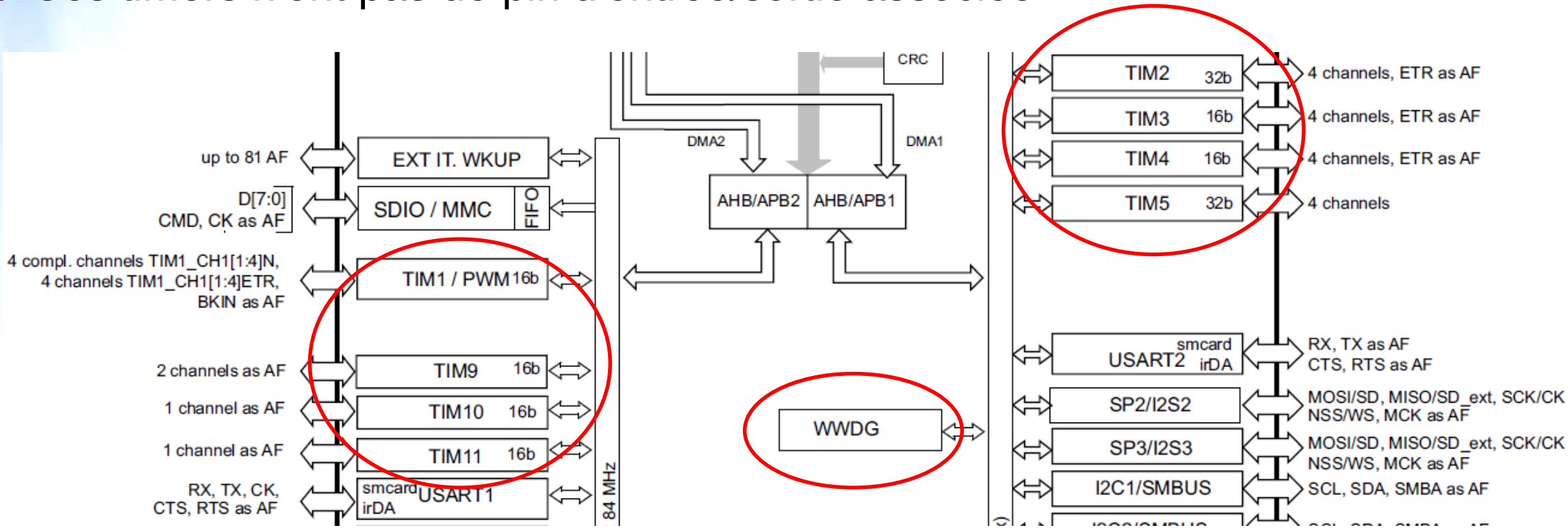
Par:

Hassan EL FADIL

elfadil.h@gmail.com

Rôles des Timers

- Dans le microcontrôleur STM32F401RE, les timers sont des composants matériels très polyvalents qui peuvent être utilisés pour diverses tâches.
- Ces timers n'ont pas de pin d'entrée/sortie associée.



Timers and watchdogs

Les STM32F401RE intègrent:

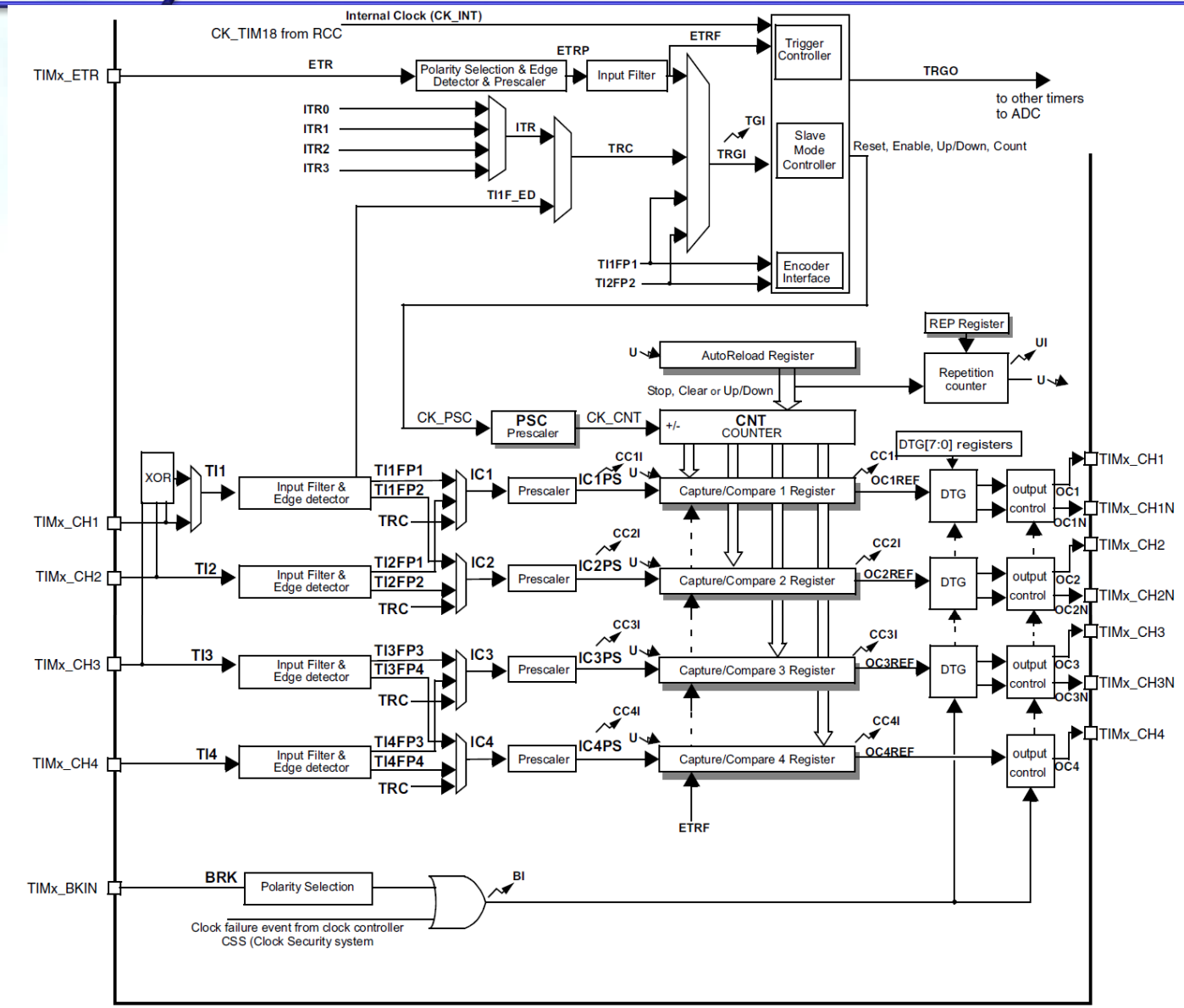
- 1 timer de contrôle avancé (**TIM1**),
- 7 timers à usage general (**TIM2, TIM3, TIM4, TIM5, TIM9, TIM10, TIM11**)
- 2 deux timers de surveillance (**Independent watchdog et Window watchdog**) .

Timer type	Timer	Counter resolution	Counter type	Prescaler factor	DMA request generation	Capture/ compare channels	Complementary output	Max. interface clock (MHz)	Max. timer clock (MHz)
Advanced-control	TIM1	16-bit	Up, Down, Up/down	Any integer between 1 and 65536	Yes	4	Yes	84	84
General purpose	TIM2, TIM5	32-bit	Up, Down, Up/down	Any integer between 1 and 65536	Yes	4	No	42	84
	TIM3, TIM4	16-bit	Up, Down, Up/down	Any integer between 1 and 65536	Yes	4	No	42	84
	TIM9	16-bit	Up	Any integer between 1 and 65536	No	2	No	84	84
	TIM10, TIM11	16-bit	Up	Any integer between 1 and 65536	No	1	No	84	84

Les Cannaux des Timers (Channels)

- Dans le contexte des microcontrôleurs STM32, les "canaux" des timers se réfèrent aux sorties ou aux entrées spécifiques associées à chaque timer.
- Chaque timer STM32 a un certain nombre de canaux qui peuvent être configurés pour différentes fonctions, ce qui offre une grande flexibilité dans la génération de signaux, la mesure de temps et l'interaction avec d'autres périphériques.
- Voici les canaux des timers STM32 qui peuvent être utilisés:
- **Capture de Signaux d'Entrée (Input Capture) :** Les canaux peuvent être configurés pour capturer le temps auquel un signal d'entrée externe change d'état. Cela est utile pour mesurer des durées, des fréquences ou des délais.
- **Comparaison avec une Valeur de Référence (Output Compare) :** Les canaux peuvent être utilisés pour comparer la valeur actuelle du compteur avec une valeur de référence. Cela peut être utilisé pour déclencher des actions spécifiques lorsque le compteur atteint une certaine valeur.
- **Génération de Signaux PWM (Pulse Width Modulation) :** Les canaux des timers peuvent être configurés pour générer des signaux PWM: Modulation de Largeur d'Impulsion (MLI)
- **Un seule impulsion en sortie: (One-Pulse Output) :** fonctionnalité qui permet au timer de générer un seul pulse sur l'une de ses sorties en réponse à un événement de déclenchement spécifique. Cette fonction est extrêmement utile dans les applications nécessitant une précision temporelle, telle que la génération de signaux de déclenchement pour des capteurs ou la production d'impulsions précises dans des systèmes de contrôle.

Schéma Fonctionnel d'un Timer



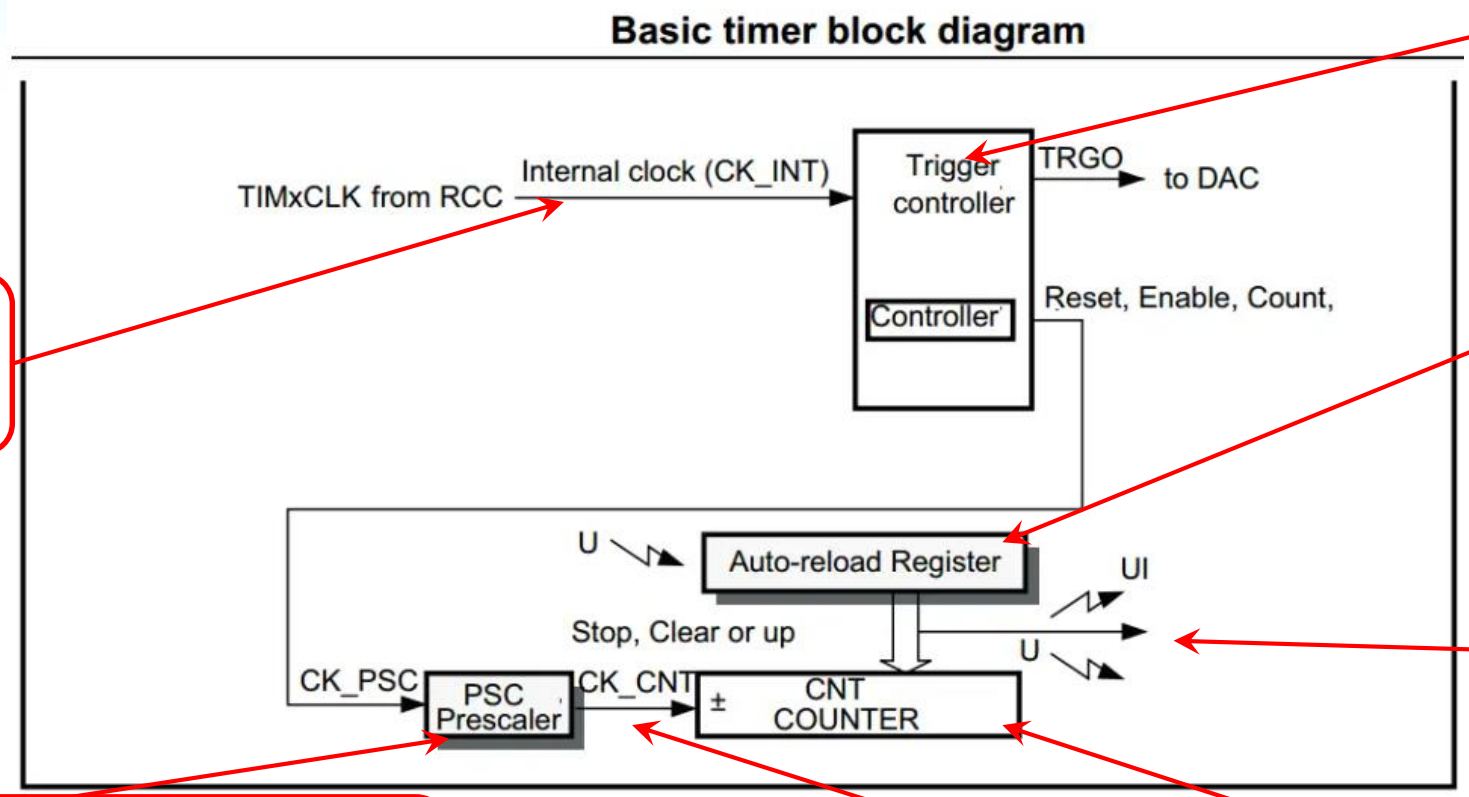
Notes:

Reg — Preload registers transferred to active registers on U event according to control bit

event

interrupt & DMA output

Diagramme Simplifié d'un Timer



Block du contrôle du Timer

Horloge principale du Timer: Issue du RCC

Le compteur se met à zéro, quand il atteint la valeur stockée dans ce registre. Il se remet à compter de nouveau

Cet événement se déclenche à chaque remise à zéro du compteur (Update Event)

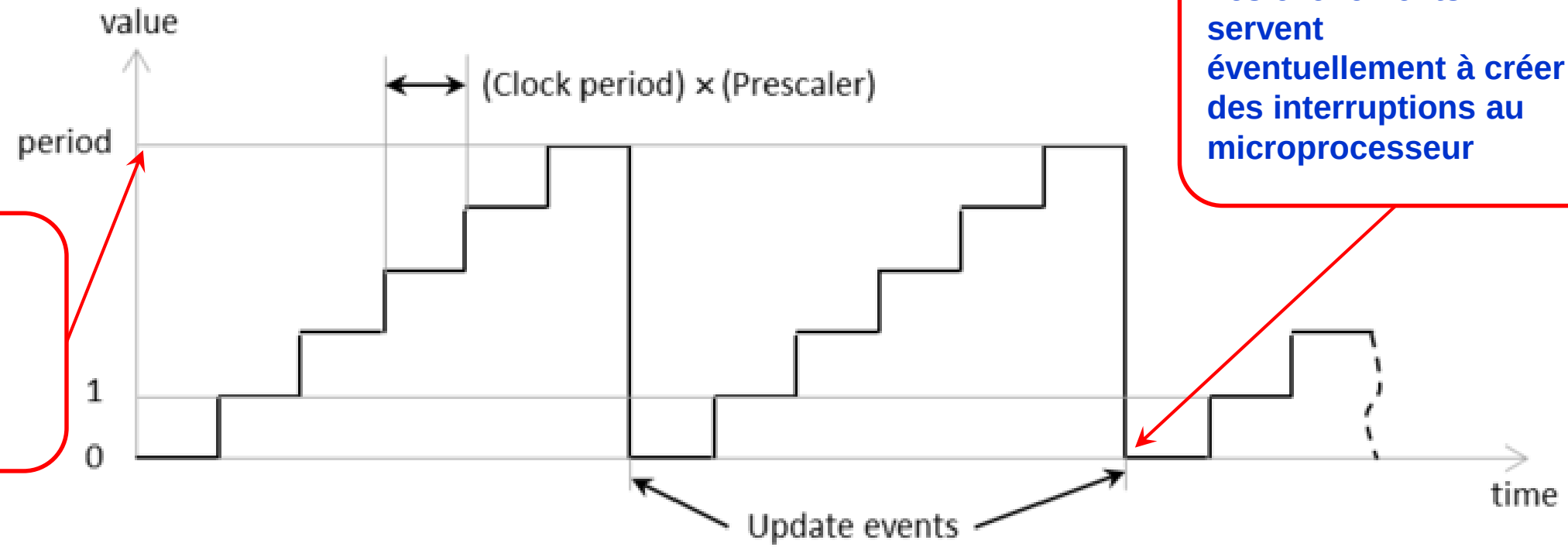
Le Prescaler: utilisé pour diviser la fréquence du CK_PSC pour obtenir celle du compteur

CK_CNT: Horloge du Compteur: La vitesse de comptage dépend de sa fréquence

Compteur 16 bits

La Séquence de Comptage

Un Timer est essentiellement un compteur indépendant qui compte de zéro jusqu'à sa valeur maximale à une vitesse donnée et génère divers événements. Il s'exécute en arrière-plan indépendamment du programme C/C++ et sa valeur suit généralement la séquence décrite ci-dessous :



Valeur préprogrammée: stockée dans le registre « Auto-Reload Register »

Les événements servent éventuellement à créer des interruptions au microprocesseur

Modes de Compteur

Mode de comptage: Upcounting mode

En mode comptage, le compteur compte de 0 jusqu'à la valeur de rechargement automatique (contenu duTIMx_ARR), puis redémarre à 0 et génère un événement de dépassement de compteur.

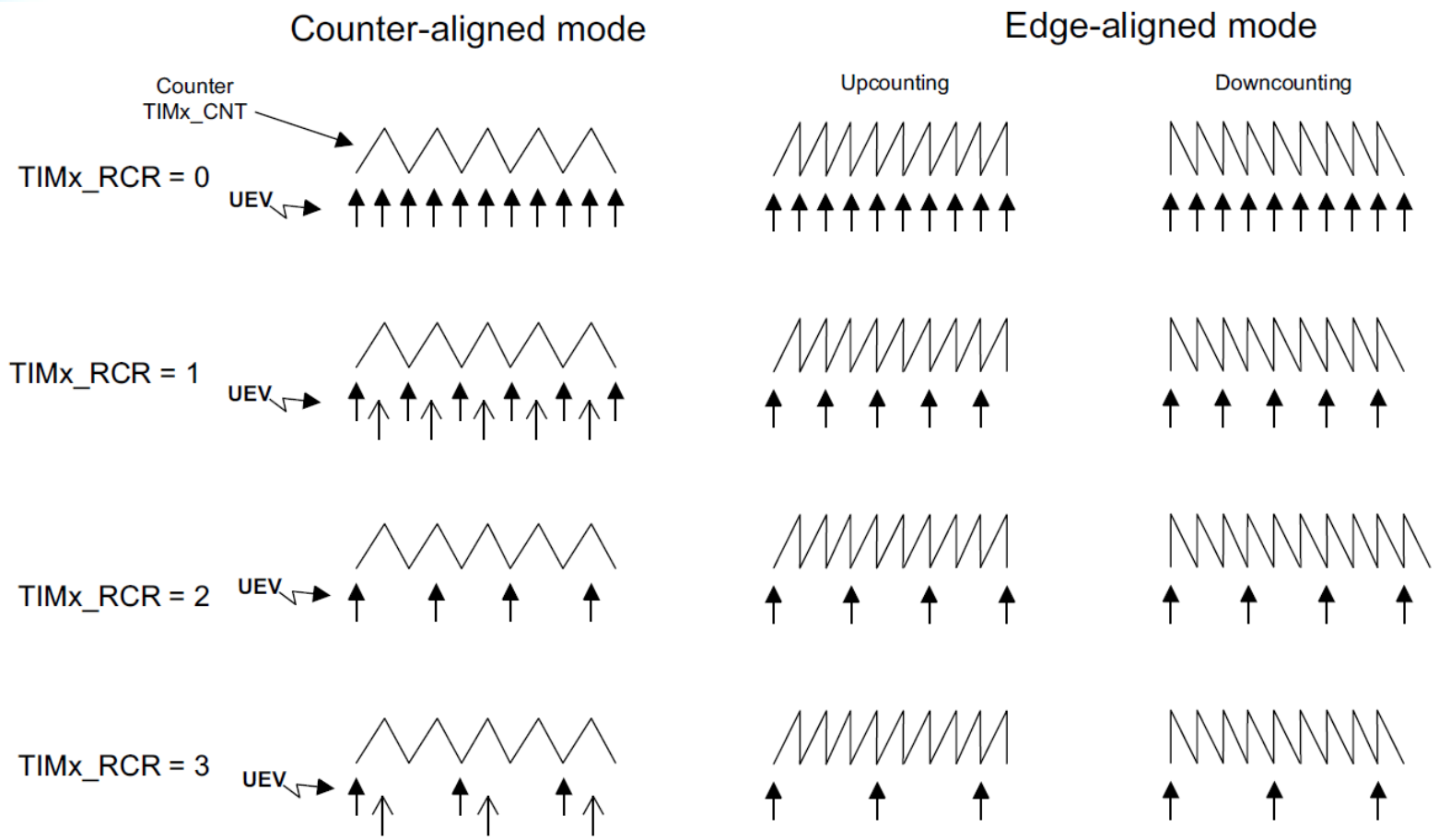
Mode décomptage: Downcounting mode

En mode décomptage, le compteur compte depuis la valeur de rechargement automatique (contenu du registre **TIMx_ARR**) jusqu'à 0, puis redémarre à partir de la valeur de rechargement automatique et génère un événement de dépassement inférieur du compteur.

Mode aligné au centre (comptage/décomptage): Center-aligned mode (up/down counting)

En mode aligné au centre, le compteur compte de 0 à la valeur de rechargement automatique **(contenu du registre TIMx_ARR) – 1**, génère un événement de dépassement de compteur, puis compte à partir de la valeur de rechargement automatique jusqu'à 1 et génère un événement de dépassement de dépassement de compteur. Puis il recommence à compter de 0.

Modes de Compteur



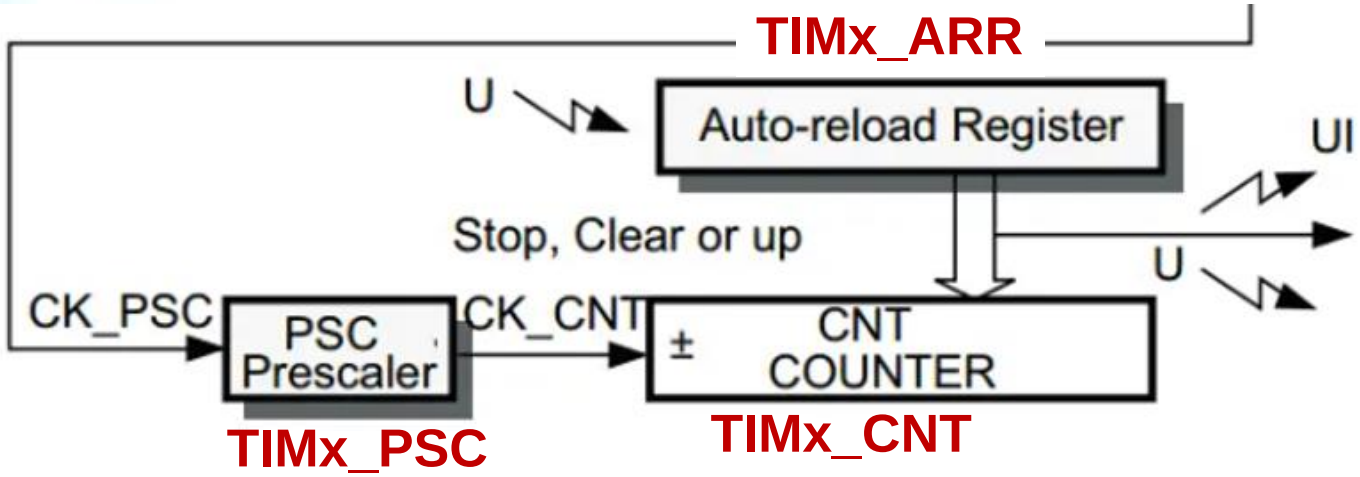
UEV → **Update event**: Preload registers transferred to active registers and update interrupt generated
Update Event if the repetition counter underflow occurs when the counter is equal to the auto-reload value

Unité de base de temps du Timer

L'unité de base de temps du Timer contient:

- Un compteur 16 bits
- Un registre du compteur (Counter Register): **TIMx_CNT**
- Un registre Prescaler (Prescaler Register) : **TIMx_PSC**
- Un registre de rechargement automatique ARR (Auto-Reload Register) : **TIMx_ARR**
- Un registre du compteur de répétitions (Repetition Counter Register): **TIMx_RCR**
- Des registres **CCR** (Capture/Compare Register)

x: remplace la N° du Timer utilisé:
1, 2, 3, 4, 5, 9, 10, 11

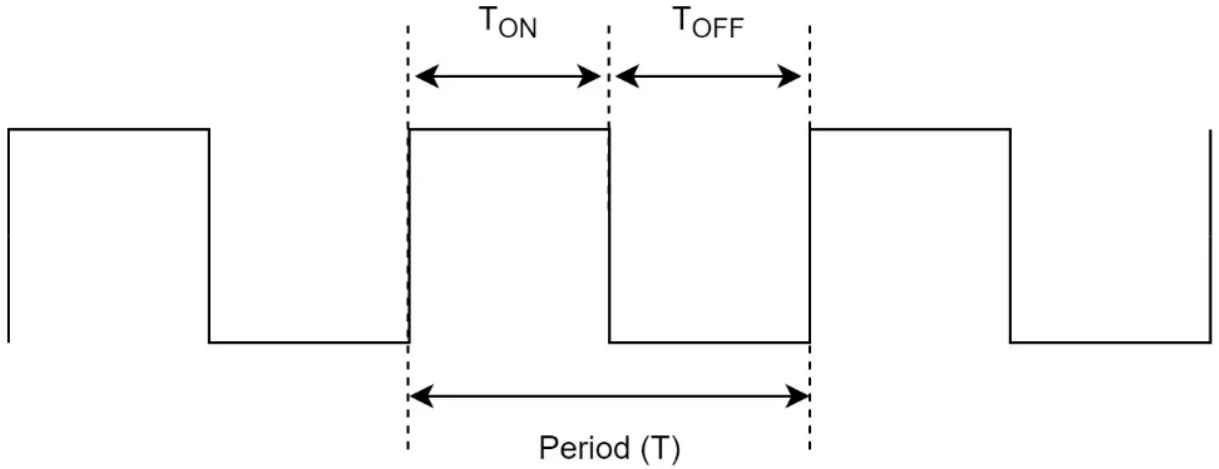


REMARQUE: Vous ne pouvez pas voir ce qui se passe à l'intérieur de « Compteur » directement via le code, vous devez utiliser le registre TIMx_CNT. Lorsque vous lisez le registre TIMx_CNT, il donne un instantané de la valeur du compteur.

Exemple d'Application: Génération d'un Signal PWM

Définition d'un Signal PWM:

- La modulation de largeur d'impulsion (PWM: **P**ulse **W**idth **M**odulation) est une technique permettant de générer un signal numérique et de contrôler par programme sa largeur d'impulsion et sa fréquence.
- Cette technique est largement utilisée dans les systèmes embarqués pour contrôler la luminosité des LED, la vitesse du moteur et d'autres applications.



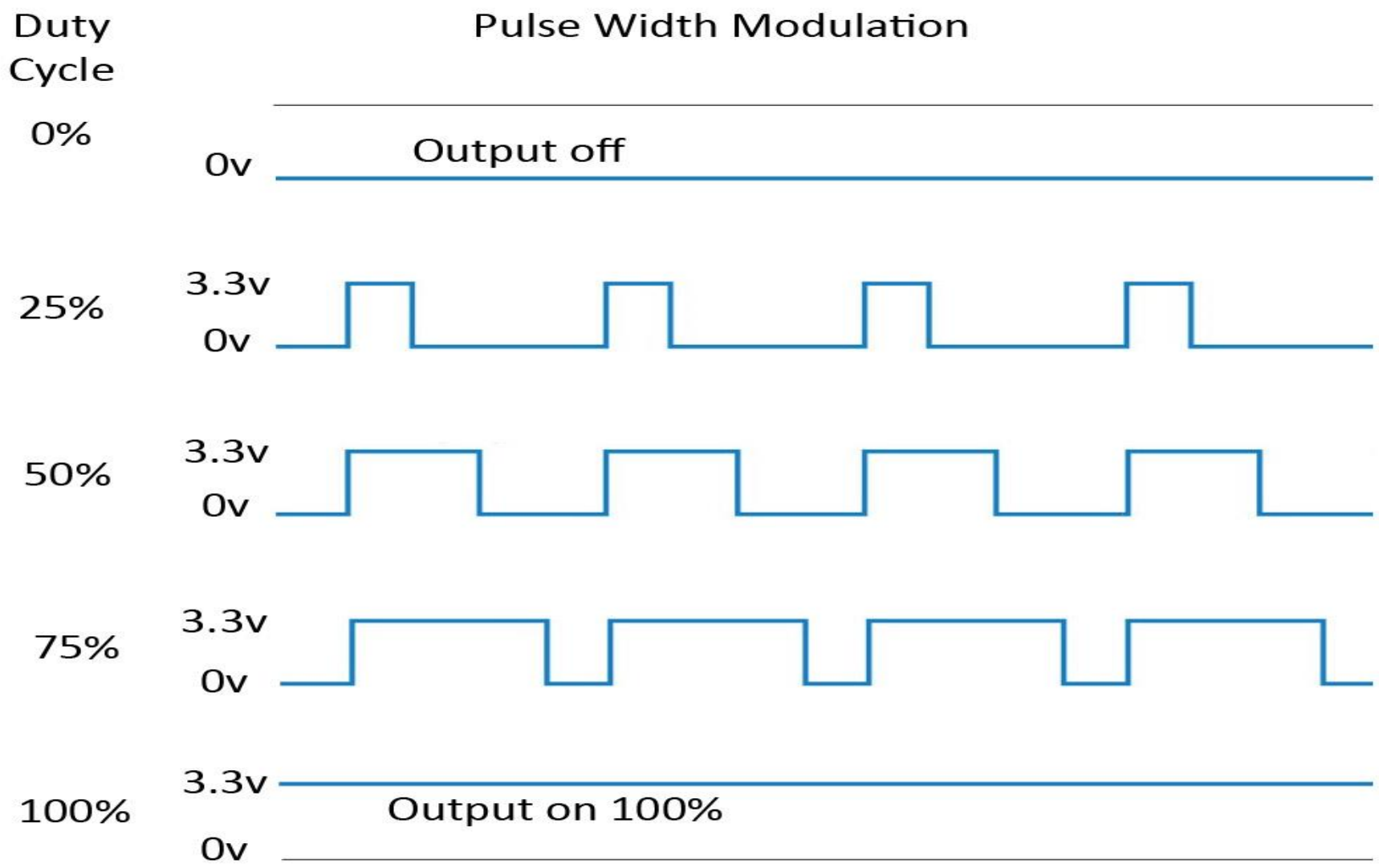
Fréquence:

$$F\,[Hz] = \frac{1}{T}$$

Rappart Cyclique (Duty Cycle):

$$Duty\_Cycle[\%] = \frac{T_{ON}}{T} \times 100$$

Exemple d'Application: Génération d'un Signal PWM



Exemple d'Application: Génération d'un Signal PWM

Résolution d'un Signal PWM:

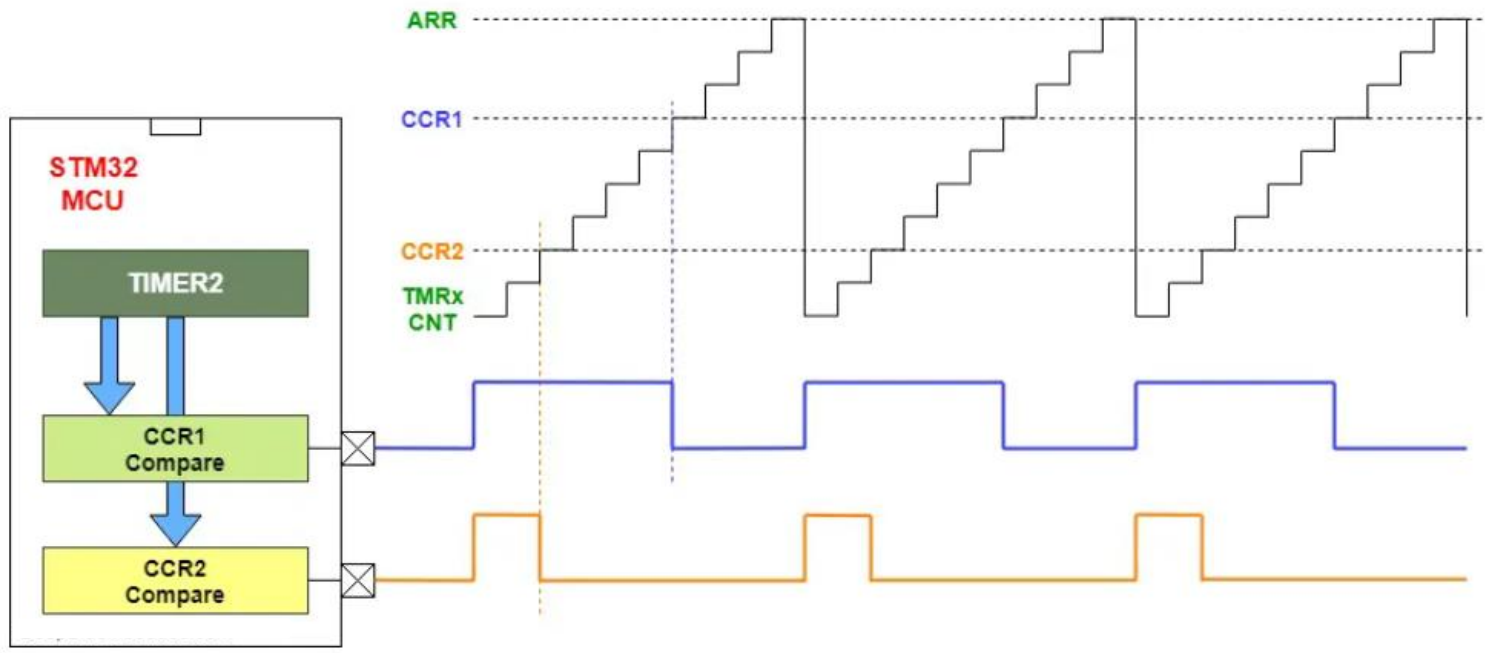
- La résolution PWM est exprimée en (bits).
- C'est le nombre de bits utilisés pour représenter la valeur du rapport cyclique.
- Il peut s'agir de 8 bits, 10, 12 ou même 16 bits.
- La résolution PWM peut correspondre au nombre de niveaux de rapport cyclique discrets compris entre 0 % et 100 %.
- Plus la résolution PWM est élevée, plus le nombre de niveaux discrets est élevé sur toute la plage du cycle de service du PWM.
- Exemple: une résolution PWM de seulement 2 bits signifie qu'il n'y a que 4 niveaux discrets pour le rapport cyclique sur toute la plage (de 0 % à 100 %): **0%; 25%; 50%; 75%; 100%**
- Un PWM avec une résolution de 8 bits aura 256 niveaux discrets pour le rapport cyclique sur toute la plage (de 0 % à 100 %).

Exemple d'Application: Génération d'un Signal PWM

Séquence de Comptage en mode PWM du STM32 :

- Aut début, la broche du canal de sortie est pilotée au niveau **HAUT**.
- Lorsque le compteur du Timer atteigne la valeur du registre **CCRx**, l'événement de correspondance fait passer la broche du canal de sortie au niveau **BAS**.
- Cette broche reste à l'état BAS jusqu'à ce que le compteur atteigne la valeur du registre de rechargement automatique (**ARR**), qui remet à zero le compage
- Le cycle, ainsi, recomence.

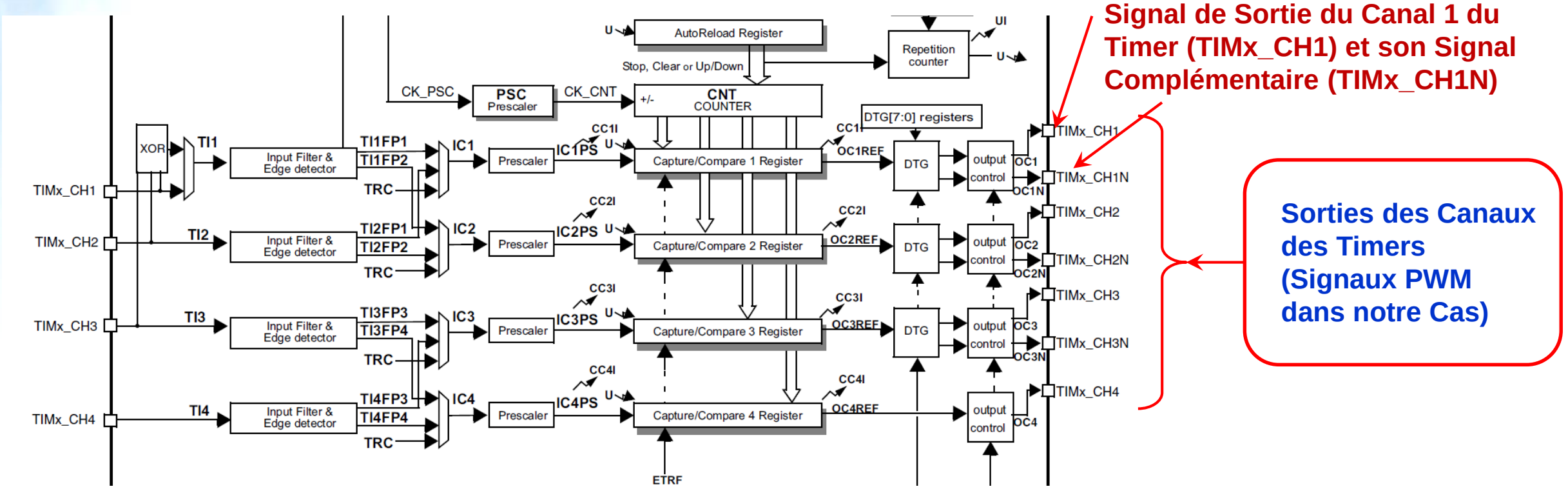
Processus de génération de signal PWM



La valeur ARR affecte la période (fréquence) du signal PWM.
La valeur CCRx affecte le rapport cyclique du signal PWM

Exemple d'Application: Génération d'un Signal PWM

REMARQUE: Un seul Timer STM32 possède généralement plusieurs canaux (2, 4, 6, ou tout ce qui se trouve dans la fiche technique). Par conséquent, en utilisant un seul Timer, vous pouvez générer indépendamment plusieurs signaux PWM avec des rapports cycles différents, mais ils ont la même fréquence, et tous seront synchronisés.



Exemple d'Application: Génération d'un Signal PWM

Fréquence PWM dans le STM32:

Dans diverses applications, vous aurez besoin de générer un signal PWM avec une fréquence spécifique: dans le contrôle des servomoteurs, la commande des LED, les convertisseurs de puissance, etc.

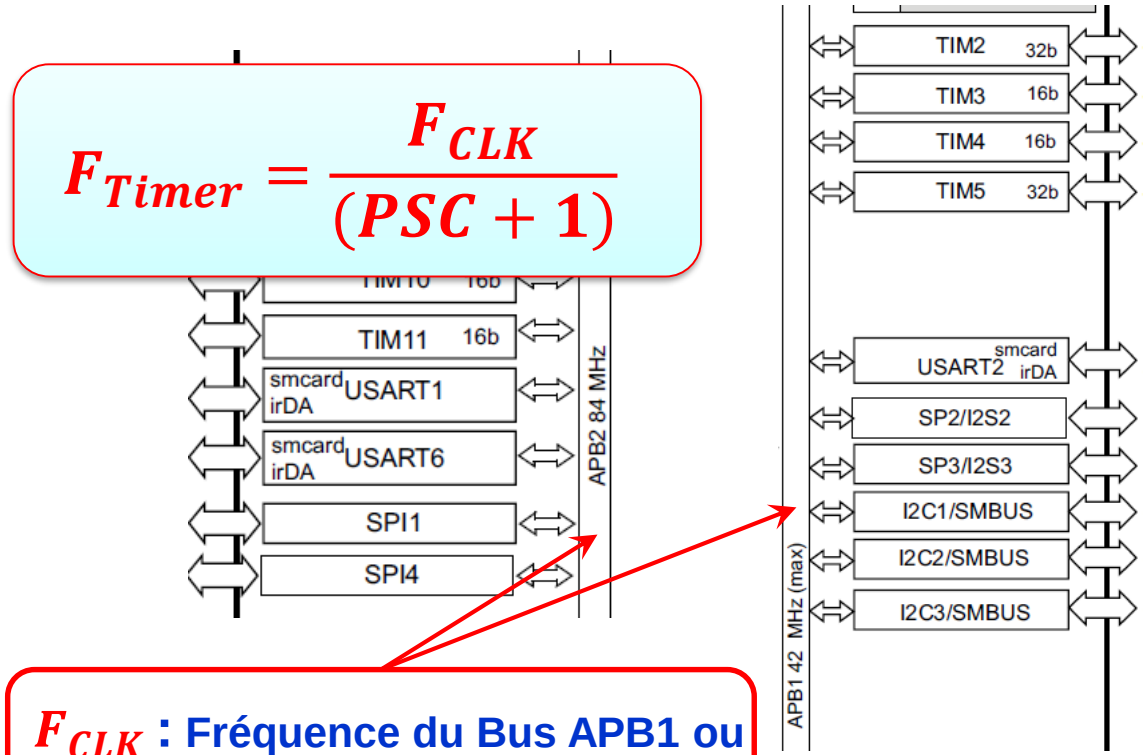
La fréquence PWM (F_{PWM}) est définie par les paramètres suivants :

- la valeur **ARR**,
- la valeur Prescaler (**PSC**)
- et l'horloge interne elle-même qui pilote le module de Timer $\mathbf{F_{CLK}}$.

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

Vous pouvez régler l'horloge que vous utilisez, le Prescaler et résoudre la valeur ARR afin d'obtenir la F_{PWM} désirée.

$$F_{Timer} = \frac{F_{CLK}}{(PSC + 1)}$$



F_{CLK} : **Fréquence du Bus APB1 ou APB2 selon le Timer utilisé**

Exemple d'Application: Génération d'un Signal PWM

Le Rapport Cyclique du PWM dans STM32:

Pour une configuration: "PWM signal in edge-aligned mode with up-counting configuration", Le rapport cyclique est contrôlé en modifiant la valeur du registre CCRx.

$$DutyCycle_{PWM}[\%] = \frac{CCRx}{ARRx} \times 100$$

La Résolution du PWM dans STM32:

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

PWM frequency versus PWM Resolution

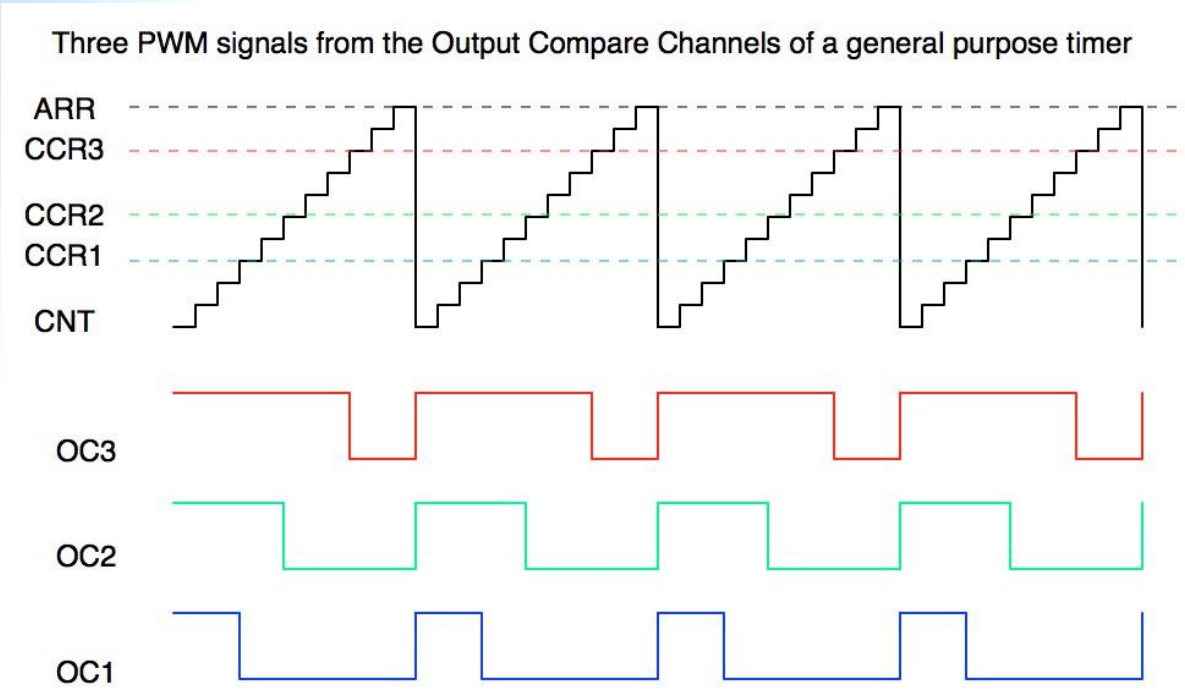
STM32 16-bit timer	PWM resolution	PWM frequency
72 MHz	16 bit	~1.1 kHz
72 MHz	14 bit	~4.4 kHz
72 MHz	12 bit	~17.5 kHz
72 MHz	10 bit	~70 kHz
72 MHz	8 bit	~281 kHz
72 MHz	6 bit	~1.125 MHz
72 MHz	4 bit	~4.5 MHz

Exemple d'Application: Génération d'un Signal PWM

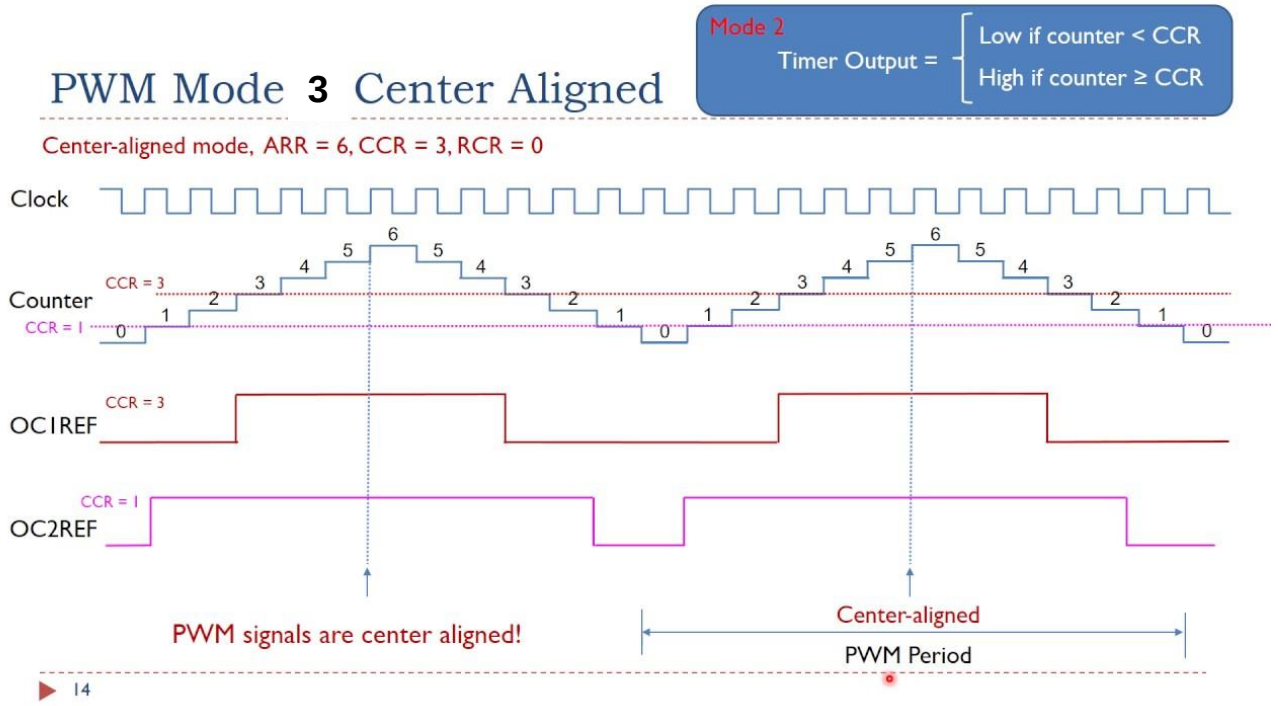
Les Modes PWM du STM32:

Mode1: Comptage
Mode2: Décomptage
Mode 3: Aligné au centre: Comptage/décomptage

Exemple: Upcounting mode



Exemple: Center-aligned mode



Exemple d'Application: Génération d'un Signal PWM

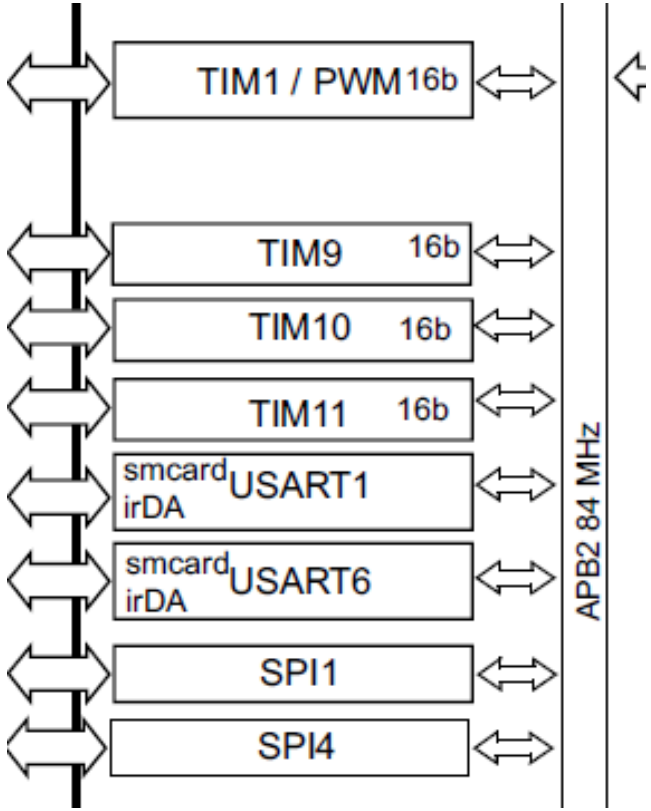
Cahier des Charges:

- Générer un signal PWM de fréquence 1kHz et de rapport cyclique 50%;
- Vérifier d'autres fréquences et d'autres rapports cycliques: 25% et 75%.

Choix de Fclk de PSC et puis determination de ARR:

1) On choist d'utiliser le **TIMER 1**
2) On choisit la fréquence du bus **APB2** qui pilote le TIM1 à **21MHz**
3) On choisit une division de cette fréquence par 21: PSC+1=21
4) Donc **PSC = 20**
5) A partir de la formule $$F_{PWM} = \frac{F_{CLK}}{(ARR + 1) \times (PSC + 1)}$$

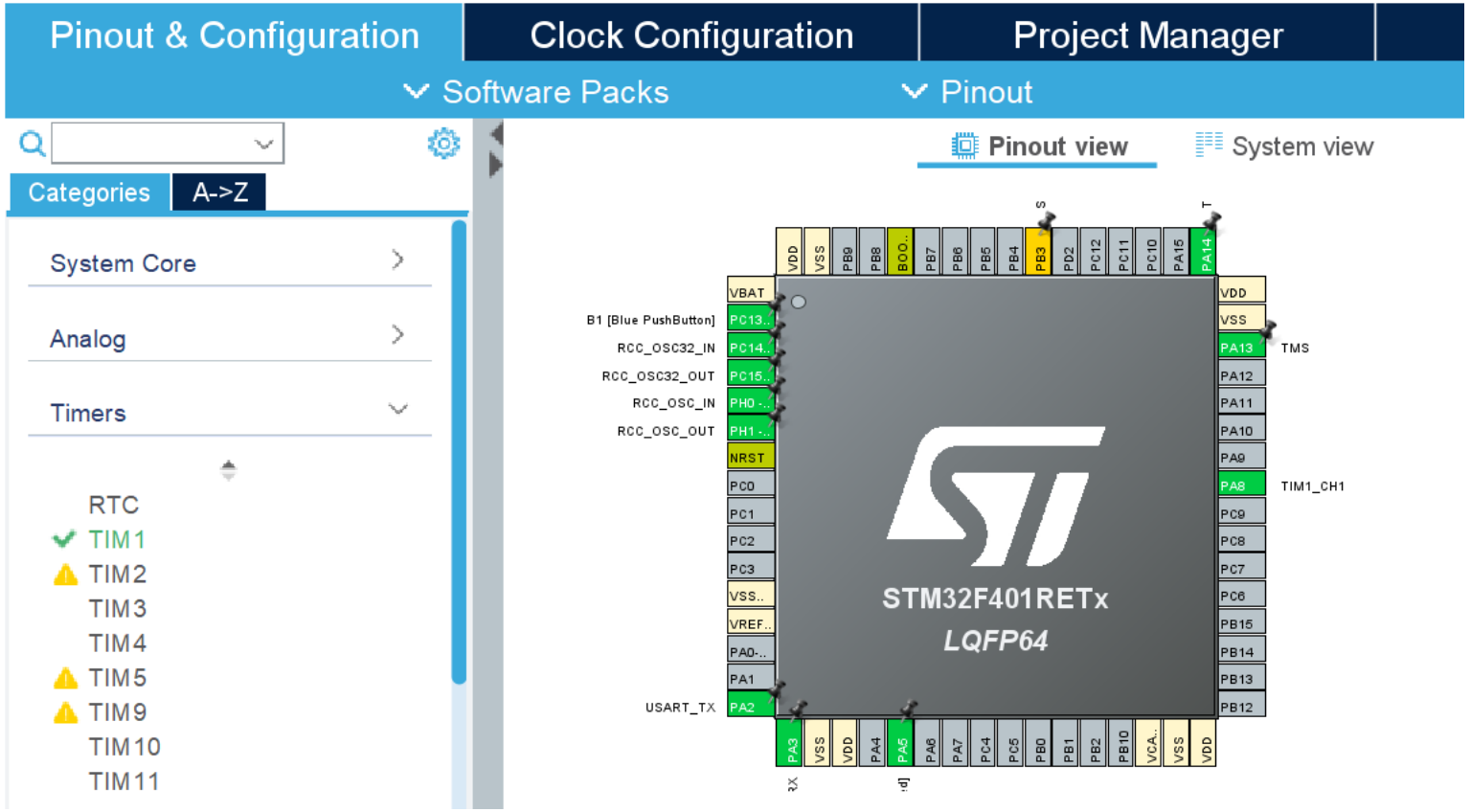
   On détermine **ARR = 999**
6) Pour avoir un rapport cyclique de 50% on fixe un CCR+1 moitié de ARR+1, soit **CCR+1 = 500;** donc **CCR = 499**



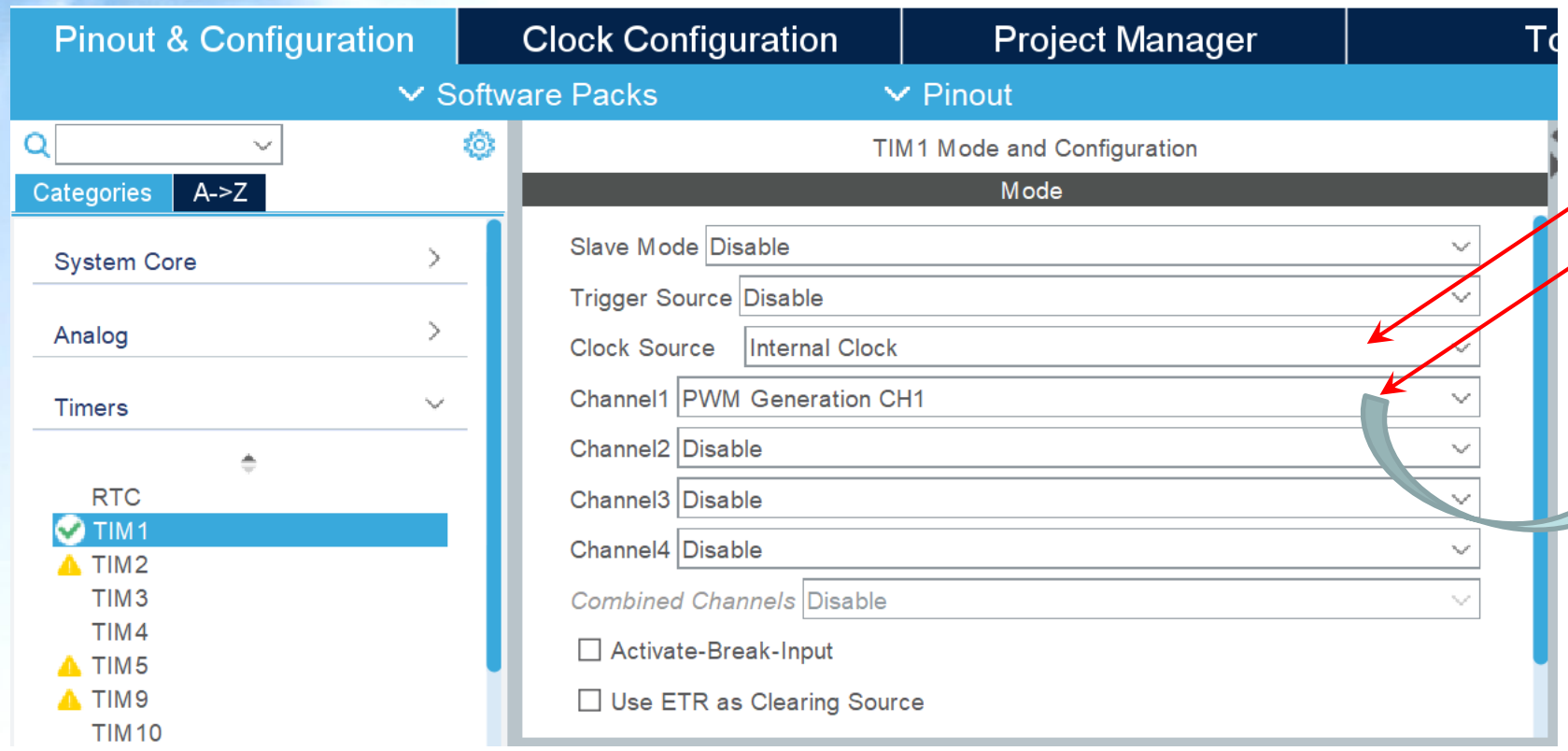
Exemple d'Application: Génération d'un Signal PWM

1. On commence par créer un nouveau projet STM32CubeIDE (**Voir Chapitre 3**)
   Appeler votre projet: **PWM_Generation**

2. **Cliquer sur Timers et choisir TIM1**



Exemple d'Application: Génération d'un Signal PWM



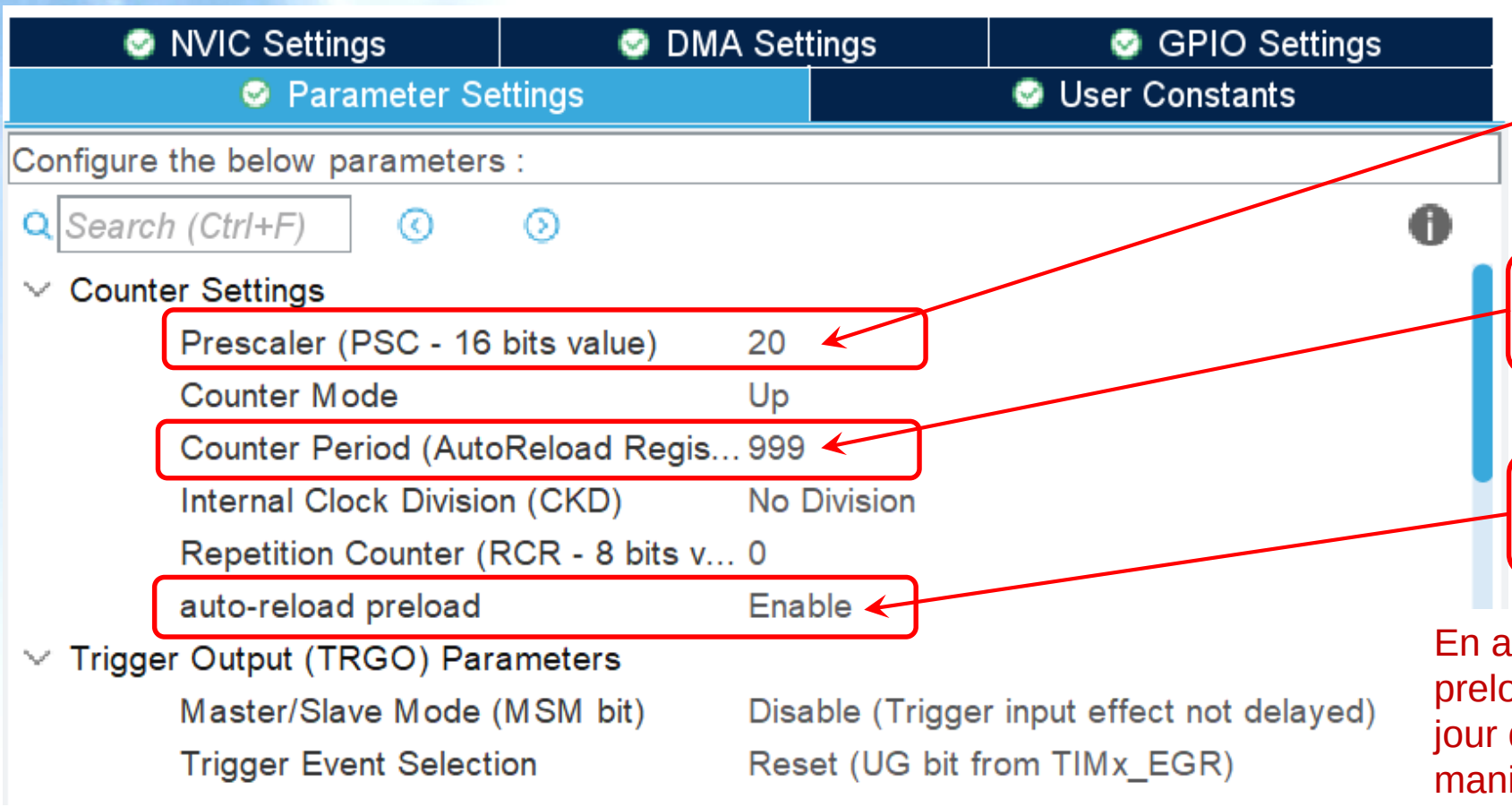
3

Dans le menu de configuration choisir:

- **Internal clock**
- **PWM Generation CH1**

On voit ici la Pin PA8 en vert nommée TIM1_CH1

Exemple d'Application: Génération d'un Signal PWM



4

Configurer le PSC = 20 (PSC+1 = 21)

5

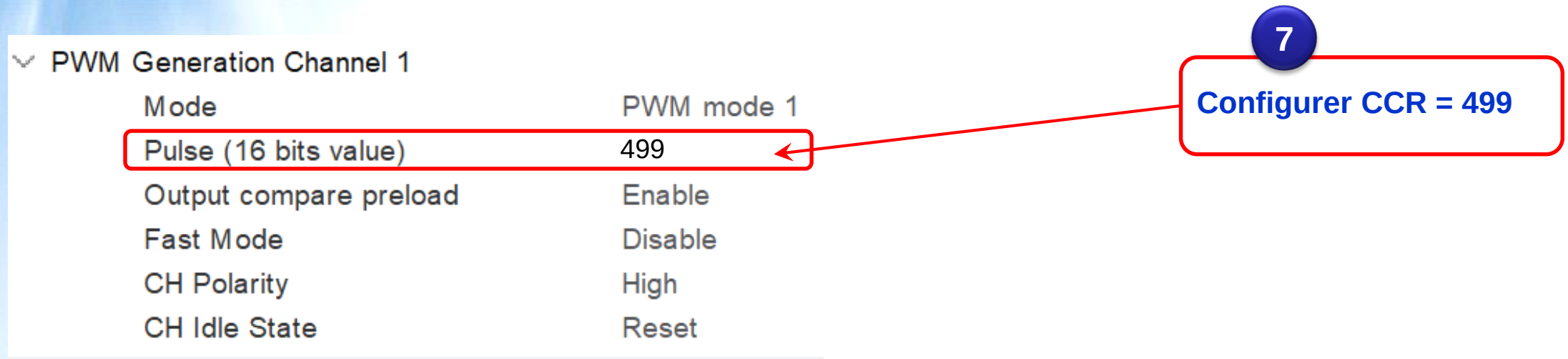
Configurer le ARR = 999 (ARR+1 = 1000)

6

Activer « auto-reload preload »

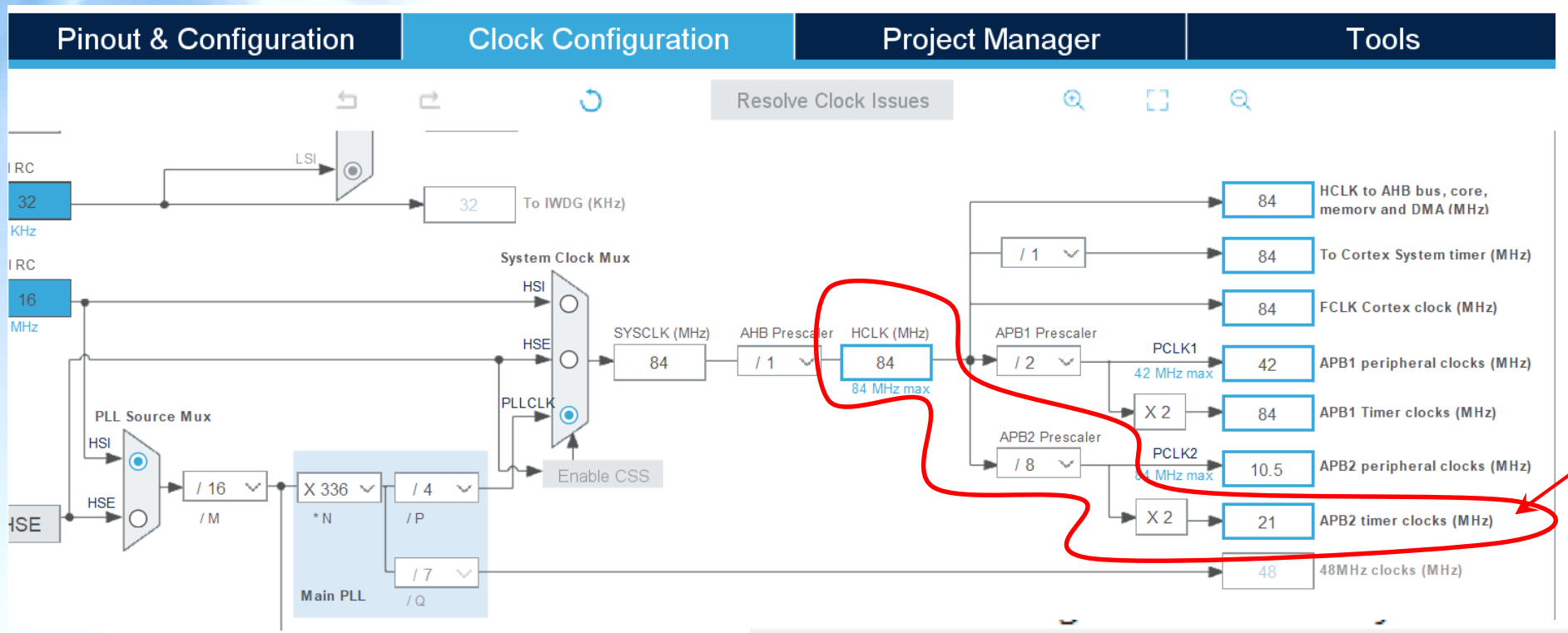
En activant la fonctionnalité auto-reload preload, vous vous assurez que les mises à jour de la valeur ARR se produisent de manière prévisible et synchronisée, évitant ainsi les problèmes liés à des mises à jour de registre intempestives et garantissant un fonctionnement fiable

Exemple d'Application: Génération d'un Signal PWM



REMARQUE: Tous les autres paramètres seront maintenus à leurs configurations par défaut.

Exemple d'Application: Génération d'un Signal PWM

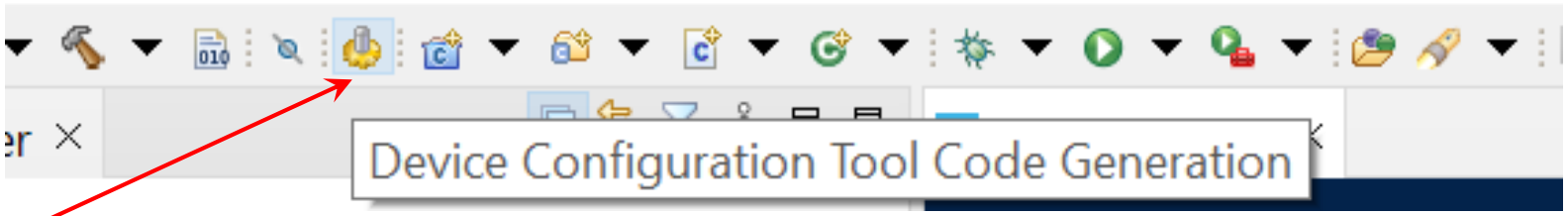


8

Avant de lancer la génération du code, configurer l'horloge pour avoir la fréquence du bus APB2 égale à 21MHz

9

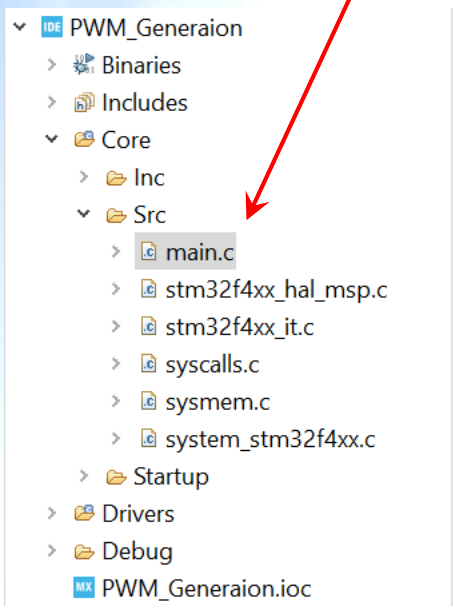
Cliquer ensuite sur « Device Configuration Tool Code Generation »



Exemple d'Application: Génération d'un Signal PWM

10

Ouvrir le fichier principal main.c




```
39
40 /* USER CODE END PM */
41
42 /* Private variables ---------------------
43 TIM_HandleTypeDef htim1;
44
45 UART_HandleTypeDef huart2;
46
47 /* USER CODE BEGIN PV */
48
49 /* USER CODE END PV */
50
51 /* Private function prototypes -----------
52 void SystemClock_Config(void);
53 static void MX_GPIO_Init(void);
54 static void MX_USART2_UART_Init(void);
55 static void MX_TIM1_Init(void);
56 /* USER CODE BEGIN PFP */
57
```

On voit ici la fonction de configuration du Timer 1

```
162 static void MX_TIM1_Init(void)
163 {
164
165   /* USER CODE BEGIN TIM1_Init 0 */
166
167   /* USER CODE END TIM1_Init 0 */
168
169   TIM_ClockConfigTypeDef sClockSourceConfig = {0};
170   TIM_MasterConfigTypeDef sMasterConfig = {0};
171   TIM_OC_InitTypeDef sConfigOC = {0};
172   TIM_BreakDeadTimeConfigTypeDef sBreakDeadTimeConfig = {0};
173
174   /* USER CODE BEGIN TIM1_Init 1 */
175
176   /* USER CODE END TIM1_Init 1 */
177   htim1.Instance = TIM1;
178   htim1.Init.Prescaler = 20;
179   htim1.Init.CounterMode = TIM_COUNTERMODE_UP;
180   htim1.Init.Period = 999;
181   htim1.Init.ClockDivision = TIM_CLOCKDIVISION_DIV1;
182   htim1.Init.RepetitionCounter = 0;
183   htim1.Init.AutoReloadPreload = TIM_AUTORELOAD_PRELOAD_ENABLE;
184   if (HAL_TIM_Base_Init(&htim1) != HAL_OK)
185   {
```

ICI les paramètres configurés

PSC

ARR

```
202   }
203   sConfigOC.OCMode = TIM_OCMODE_PWM1;
204   sConfigOC.Pulse = 499 ;
205   sConfigOC.OCPolarity = TIM_OCPOLARITY_HIGH;
206   sConfigOC.OCNPolarity = TIM_OCNPOLARITY_HIGH;
207   sConfigOC.OCFastMode = TIM_OCFAST_DISABLE;
208   sConfigOC.OCIdleState = TIM_OCIDLESTATE_RESET;
209   sConfigOC.OCNIdleState = TIM_OCNIDLESTATE_RESET;
```

CCR

Exemple d'Application: Génération d'un Signal PWM

11

Faire sortir le signal PWM généré sur une Pin du microcontrôleur: Pour cela, on ajoute la fonction suivante au fichier principal main.c

```
91   /* Initialize all configured peripherals */
92   MX_GPIO_Init();
93   MX_USART2_UART_Init();
94   MX_TIM1_Init();
95   /* USER CODE BEGIN 2 */
96   HAL_TIM_PWM_Start(&htim1, TIM_CHANNEL_1); // Génération du Signal PWM
97
98   /* USER CODE END 2 */
99
100  /* Infinite loop */
101  /* USER CODE BEGIN WHILE */
102  while (1)
103  {
104    /* USER CODE END WHILE */
```

```
HAL_TIM_PWM_Start(&htim1, TIM_CHANNEL_1); // Génération du Signal PWM
```

12

Lancer, enfin, la compilation

Exemple d'Application: Génération d'un Signal PWM

Explication de la fonction: HAL_TIM_PWM_Start(&htim, TIM_CHANNEL_1)

- **htim** : un pointeur vers la **structure** du TIM (Timer). Cette structure contient les informations de configuration du Timer
- **TIM_CHANNEL** : le canal PWM que vous souhaitez démarrer. Ce paramètre peut avoir les valeurs TIM_CHANNEL_1, TIM_CHANNEL_2, TIM_CHANNEL_3 ou TIM_CHANNEL_4, selon le canal sur lequel vous souhaitez démarrer la sortie PWM.

- **HAL** : Cela signifie « **Hardware Abstraction Layer** » (couche d'abstraction matérielle). C'est une interface de haut niveau dans les bibliothèques HAL de STM32 qui fournit une méthode unifiée d'interaction avec le matériel, permettant d'écrire du code portable sur différents microcontrôleurs STM32.
- **TIM** : "Timer"
- **PWM** : « Pulse Width Modulation » "
- **Start** : Cela indique que la fonction est utilisée pour démarrer la sortie PWM. Lorsque vous "démarrez" le PWM, le canal du Timer configuré commence à générer le signal PWM en fonction des paramètres spécifiés (comme le rapport cyclique et la fréquence).

- En résumé, **HAL_TIM_PWM_Start** est une fonction fournie par la couche HAL de STM32 pour démarrer la sortie PWM sur un canal du Timer spécifique. Elle simplifie le processus de configuration des Timers et de leurs canaux pour l'opération PWM, permettant aux développeurs de se concentrer sur la logique de l'application plutôt que sur les détails matériels de bas niveau

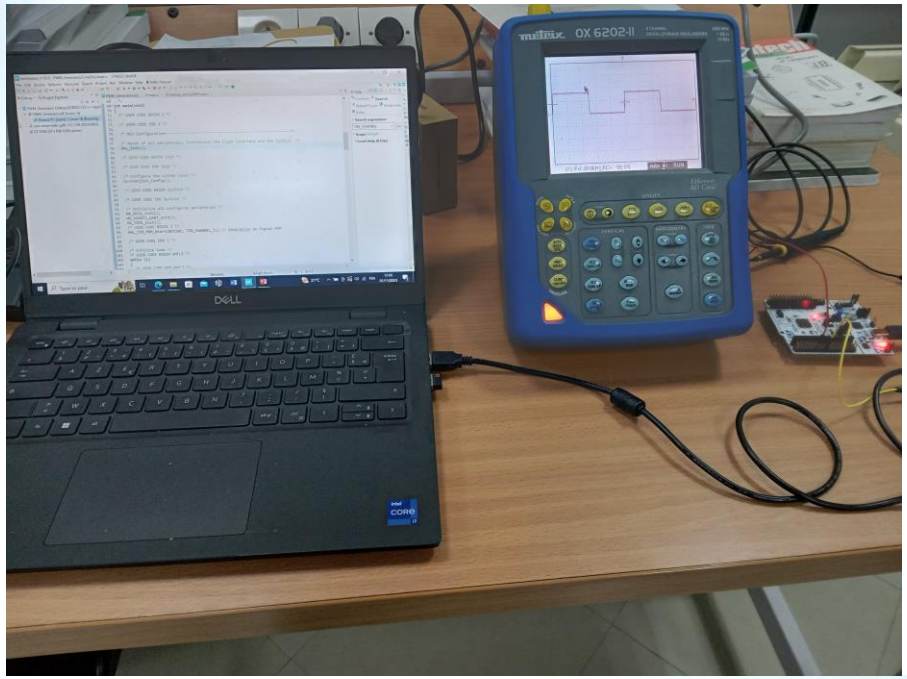
Exemple d'Application: Génération d'un Signal PWM

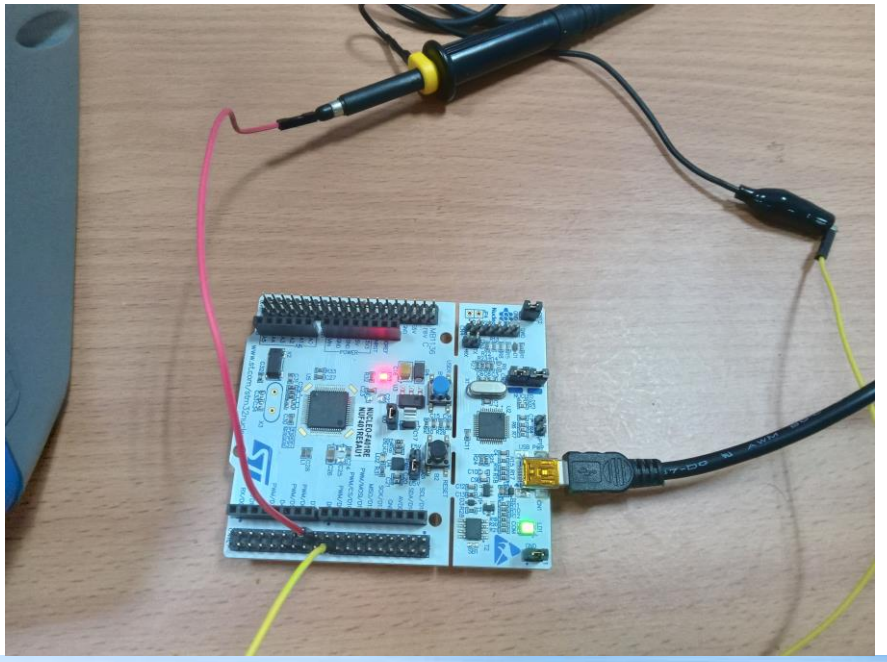
Le banc de Test :

13

Avant de lancer le Débogage, il convient de faire le câblage nécessaire:

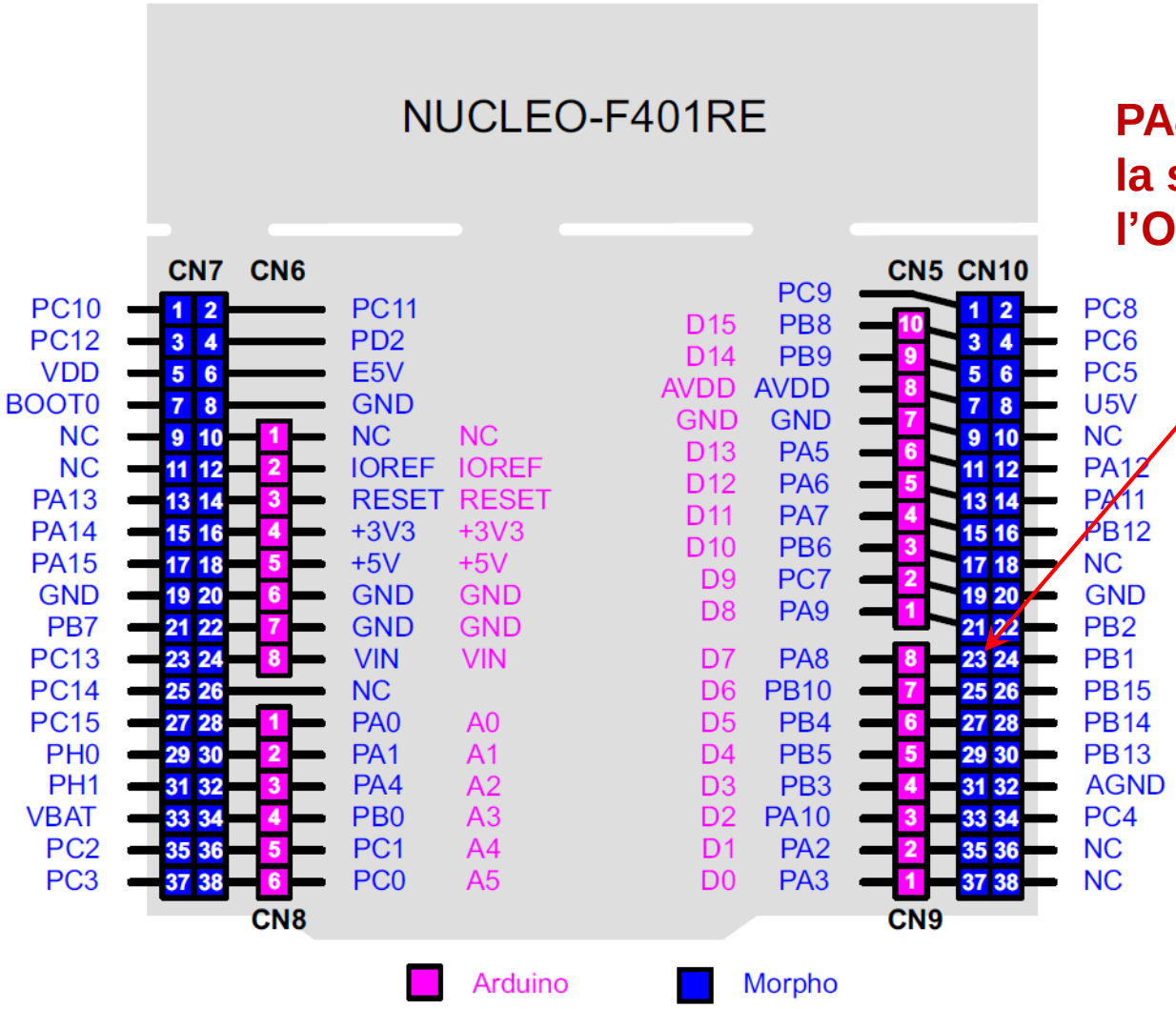
- Lier la Pin PA8 à la sonde l'oscilloscope
- Lier le GND de la carte au GND de l'oscilloscope
- Relier la carte NUCLEO F401RE au port USB



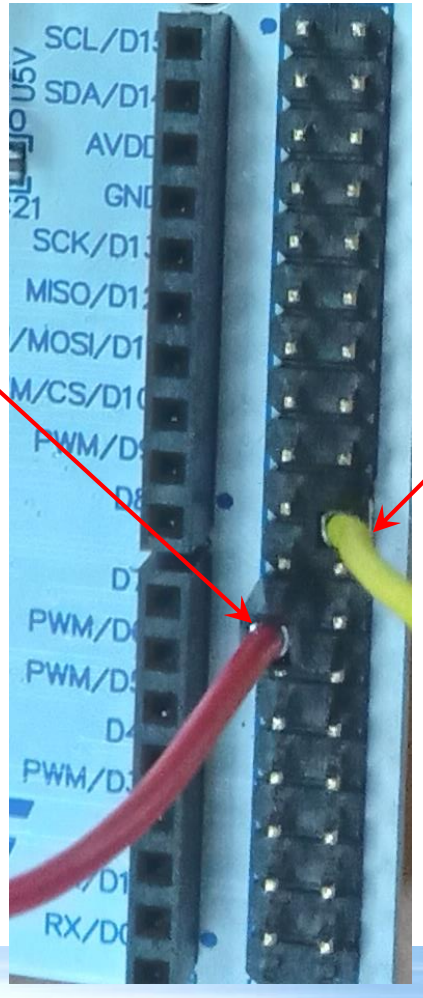


Exemple d'Application: Génération d'un Signal PWM

Le Câblage:



PA8 (Pin 23) vers la sonde de l'Oscilloscope

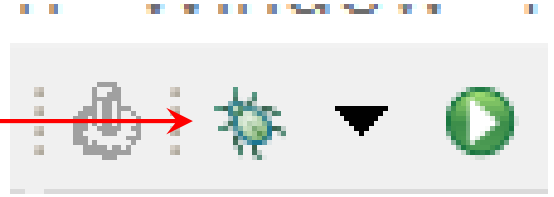


GND (Pin 20) vers le GND de l'Oscilloscope

Exemple d'Application: Génération d'un Signal PWM

14

Lancer le Débogage



15

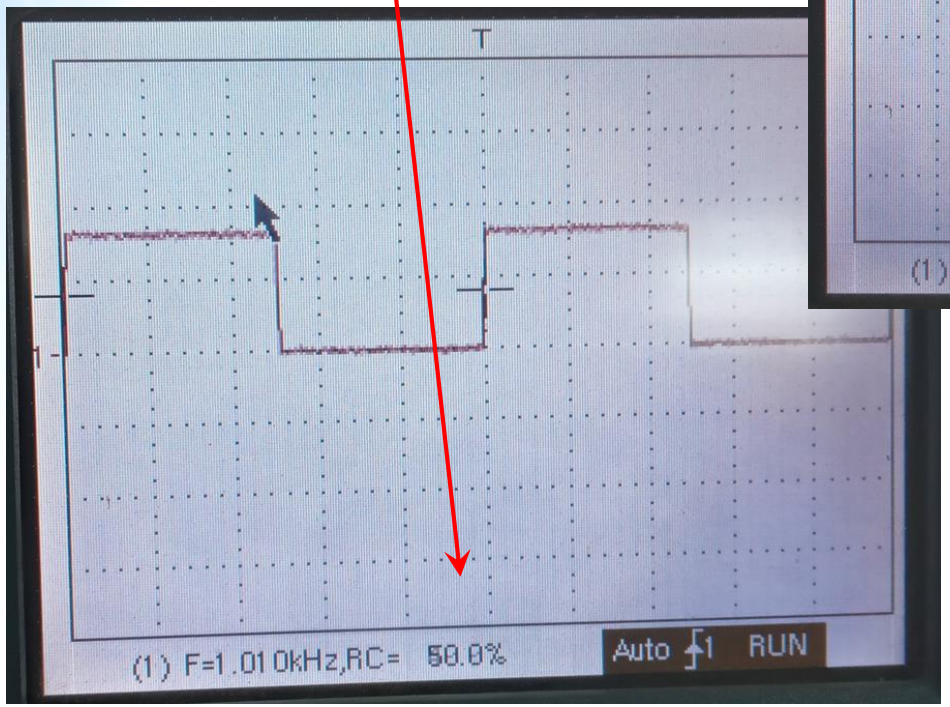
Cliquer enfin sur ce bouton RESUME en VERT
Cela permet d'exécuter la suite du programme.
Le Signal PWM doit normalement s'afficher sur l'écran de l'oscilloscope.

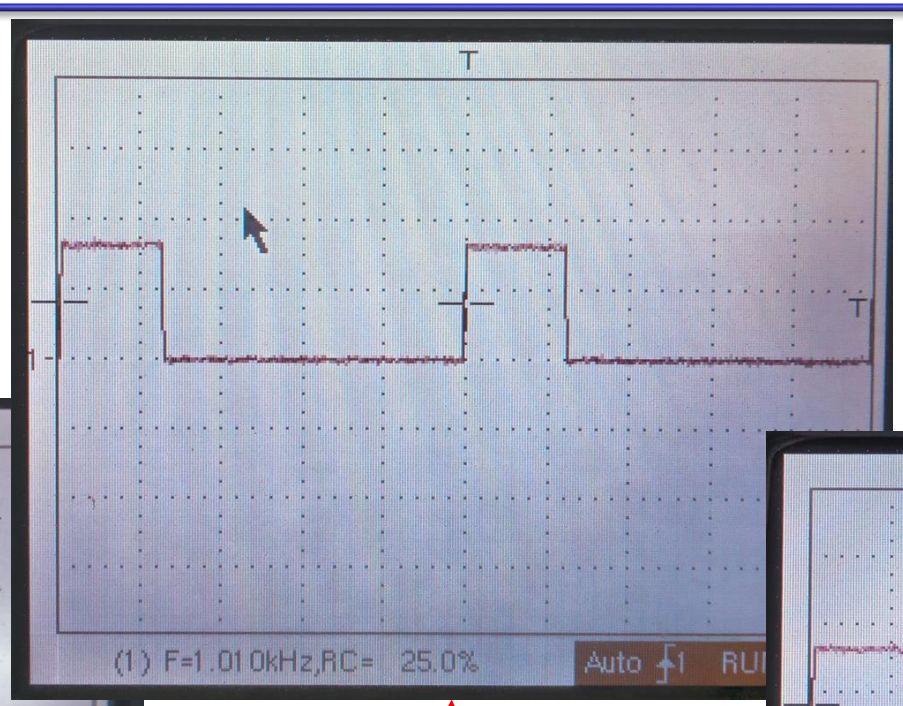
VERIFIER !!!

Exemple d'Application: Génération d'un Signal PWM

Les Résultats de Test:

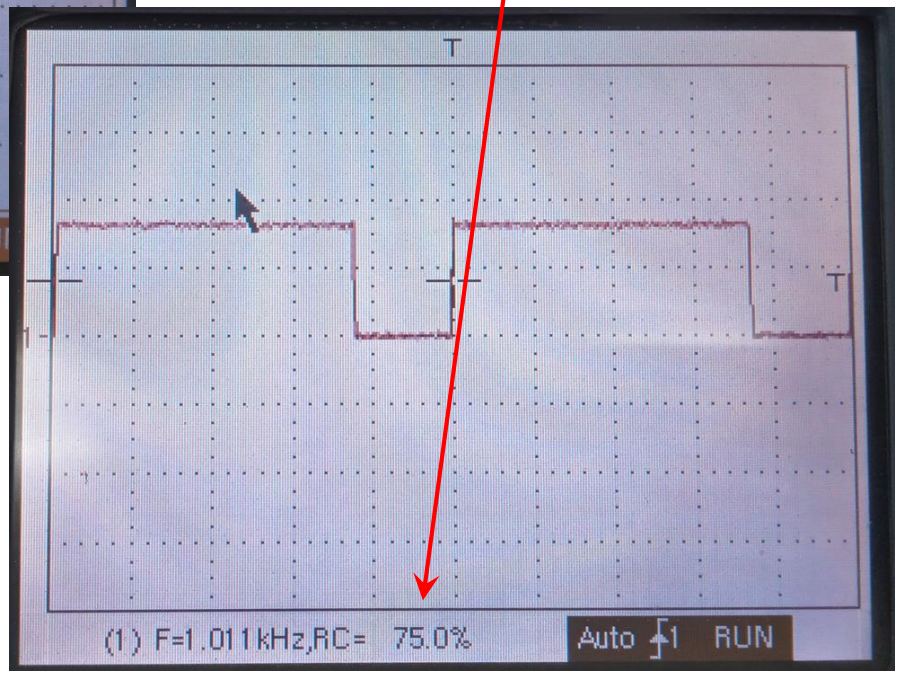
Essai pour F=1kHz et un rapport cyclique de 50%





Essai pour F=1kHz et un rapport cyclique de 25%

Essai pour F=1kHz et un rapport cyclique de 75%

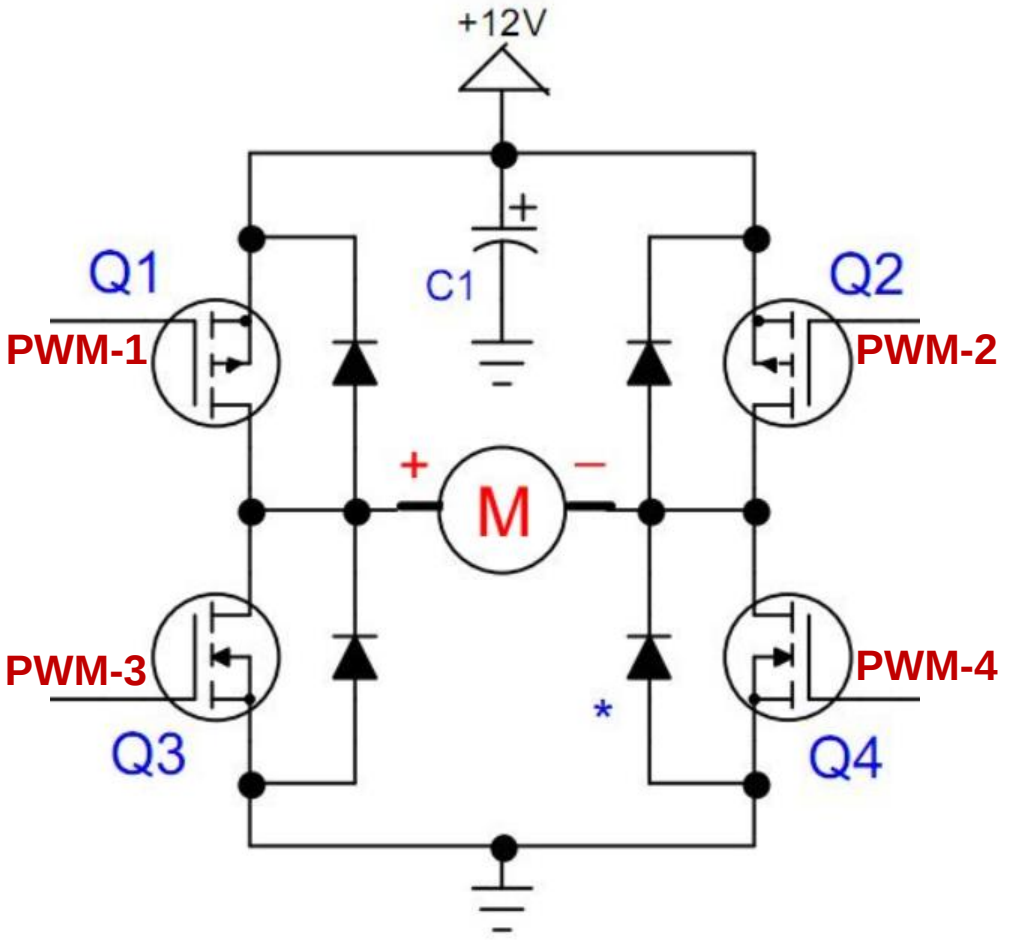


Génération de Signaux PWM Complémentaires

Pourquoi des signaux PWM complémentaires:
Générer des signaux PWM complémentaires offre plusieurs avantages dans de nombreuses applications électroniques, notamment dans les systèmes de commande et d'entraînement de moteurs.
Les signaux PWM complémentaires permettent de contrôler les dispositifs bidirectionnels tels que les ponts en H utilisés dans les systèmes de commande de moteurs.
En utilisant deux signaux PWM complémentaires, on peut contrôler la direction du courant à travers le moteur en activant/désactivant les transistors du pont en H de manière appropriée:

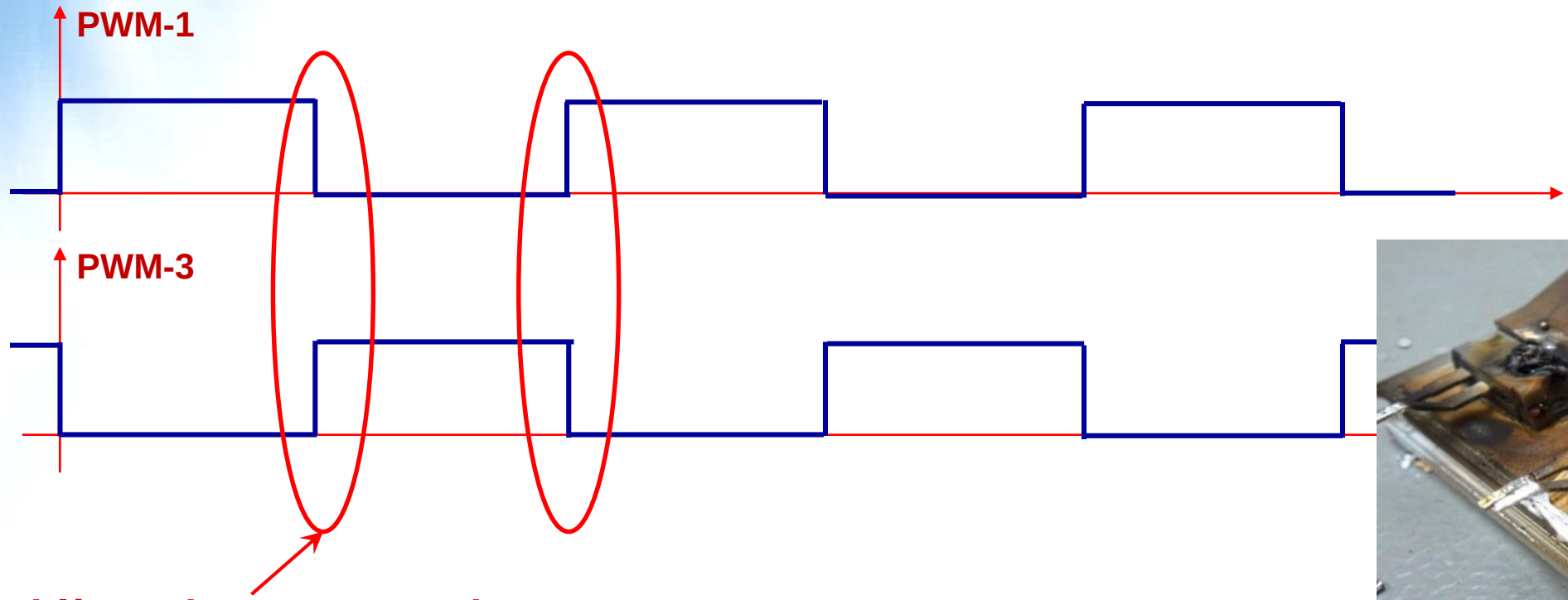
- Q1 et Q4 ON: moteur tourne dans le sens direct
- Q2 et Q3 ON: moteur tourne dans le sens inverse

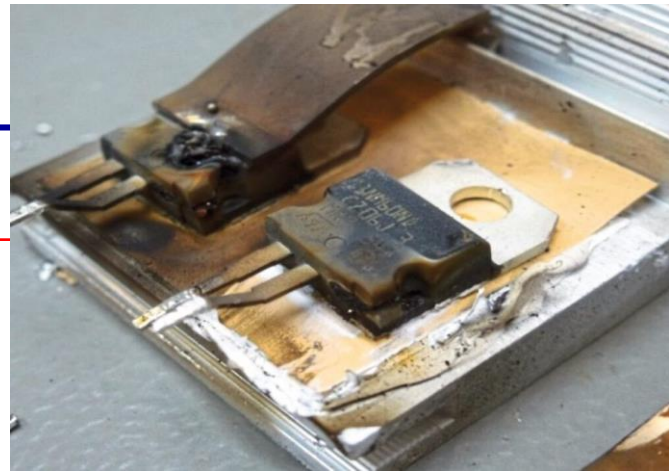
Les signaux de commande de Q1 et Q3 doivent être complémentaires. De même pour ceux de Q2 et Q4.



PONT EN H

Génération de Signaux PWM Complémentaires



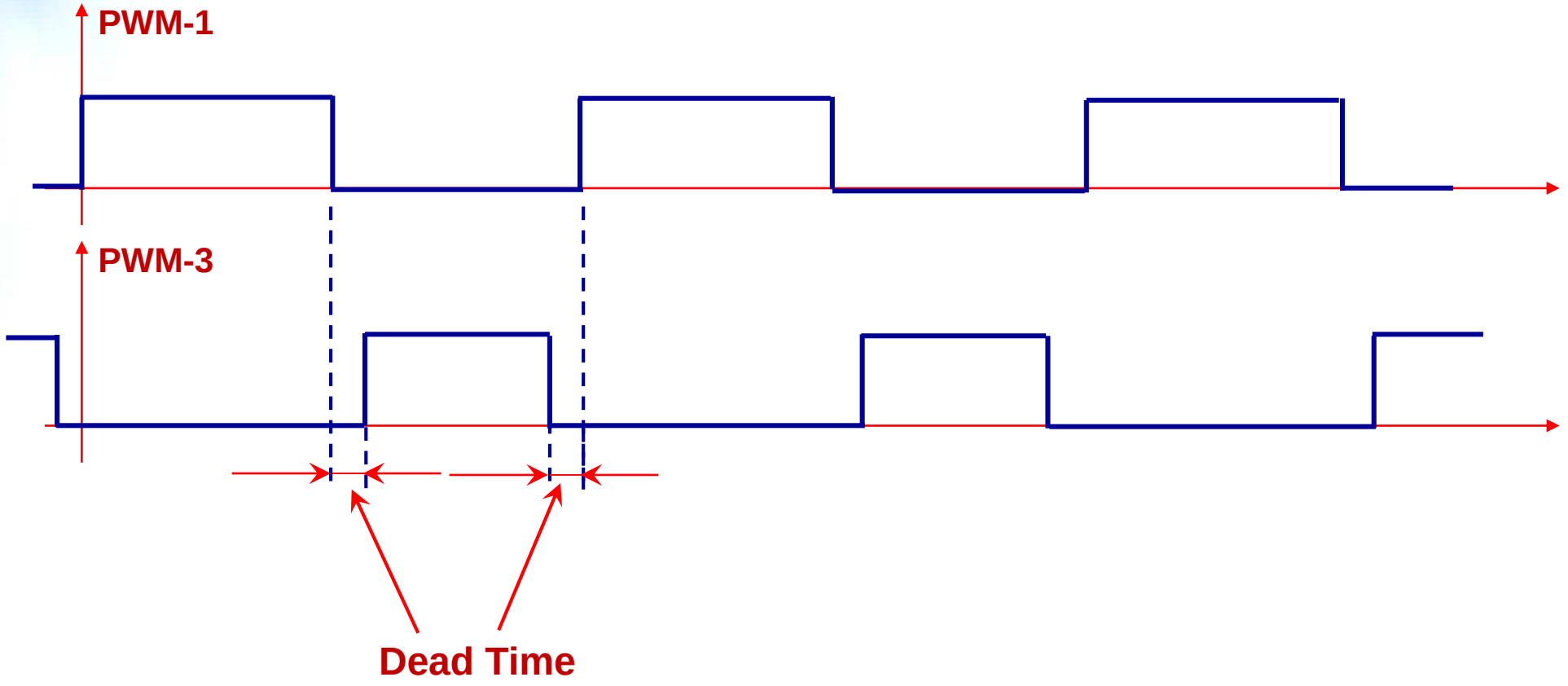


Problème des commutations:

Aux instants des transitions, et à cause des temps de commutations des transistors (temps de montée et temps de descente), les transistors d'un bras (Q1 et Q3, ou Q2 et Q4) peuvent conduire simultanément (ON). Le source est donc court-circuitée, le courant qui traverse les transistors est assez grand. Par conséquent, les transistors seront détruits!!!!!

Génération de Signaux PWM Complémentaires

Solution : Insérer un temps mort « Dead Time » entre les signaux complémentaires.



Génération de Signaux PWM Complémentaires

Détermination du « Dead Time », en nombre de cycles pour STM32 :

$$DT_{Cycles} = T_{DEAD} \times F_{Timer}$$

Le T_{DEAD} peut être imposé ou déterminé à partir de la formule suivante:

$$T_{DEAD} = (DT\%) \times T_{PWM}$$

$$T_{PWM} = \frac{1}{F_{PWM}}$$

$$F_{PWM} = \frac{F_{Timer}}{(ARR + 1)}$$

$$F_{Timer} = \frac{F_{CLK}}{(PSC + 1)}$$

- T_{DEAD} : Temps mort à imposer
- DT_{Cycles} : Nombre de cycles d'horloge du temps mort
- $DT\%$: Pourcentage de temps mort par rapport à la période de la PWM
- T_{PWM} : Période du signal PWM et F_{PWM} sa fréquence
- F_{Timer} : Fréquence d'horloge du TIMER
- F_{CLK} : Fréquence du Bus (APB1 ou APB2) qui pilote le TIMER

REMARQUE: Le pourcentage de temps mort dans un microcontrôleur STM32 dépend des exigences spécifiques de l'application et des caractéristiques des composants de l'étage de puissance (tels que les MOSFET ou les IGBT) utilisés. VOIR le DOC CONSTRUCTEUR des composants.

Génération de Signaux PWM Complémentaires

Exemple d'Application:

Cahier des Charges:

- Générer deux signaux PWM complémentaires de fréquence 1kHz et de rapport cyclique 50%, avec un temps mort représentant 5% de la période de la MLI;

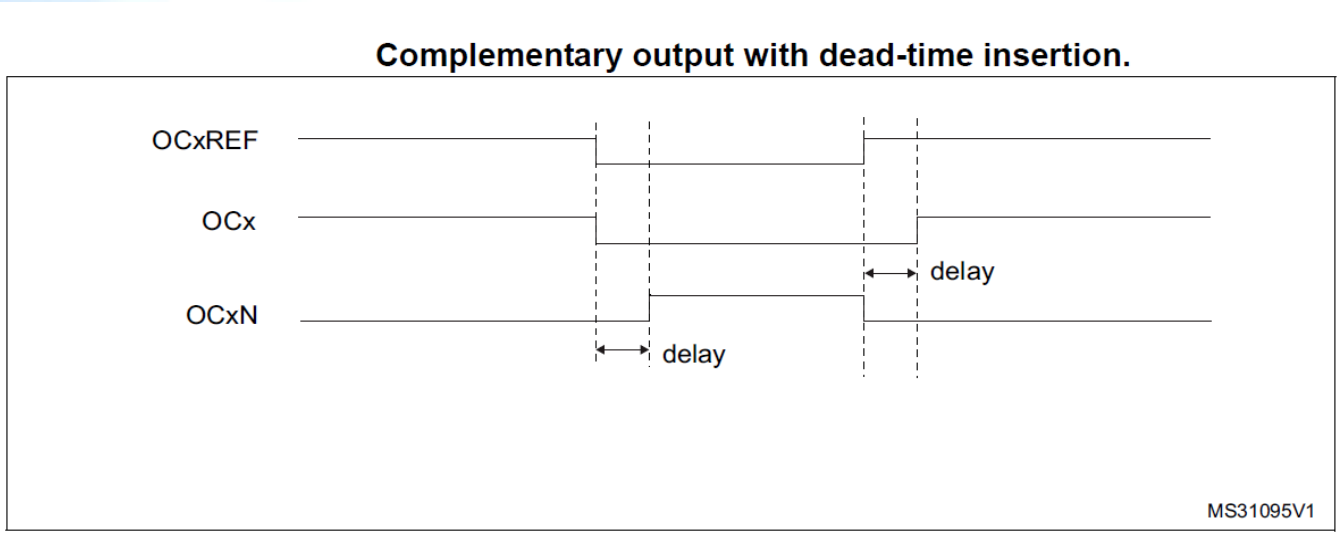
Choix de Fclk de PSC et puis determination de ARR:

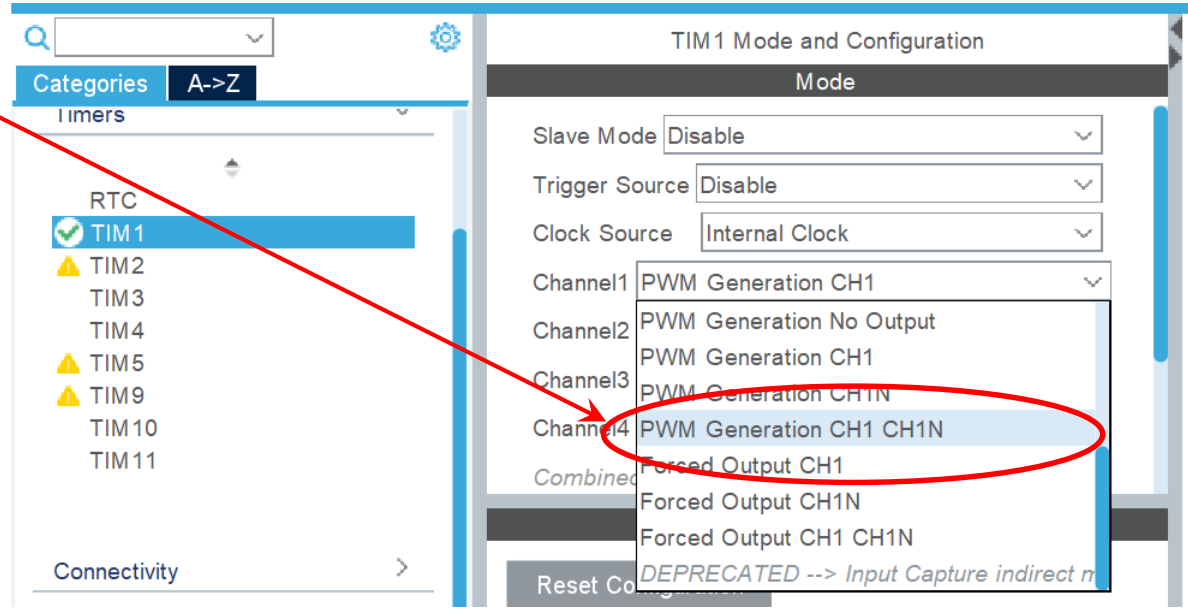
1) On choisit d'utiliser le **TIMER 1**
2) On choisit la fréquence du bus **APB2** qui pilote le TIM1 à $F_{CLK} = 21MHz$
3) On choisit une division de cette fréquence par 21: PSC+1=21; $PSC = 20$
4) Donc : $F_{Timer} = 1MHz$;
5) Fréquence de la PWM 1kHz: $F_{PWM} = 1kHz$; $ARR = 999$; $T_{PWM} = 1ms$
6) Rapport cyclique de 50% : $CCR = 499$;
7) On se fixe un $DT\% = 5\%$; donc : $T_{DEAD} = (DT\%) \times T_{PWM} = 50\mu s$
8) Finalement, on obtient : $DT_{Cycles} = T_{DEAD} \times F_{Timer}$; $DT_{Cycles} = 50$

Génération de Signaux PWM Complémentaires

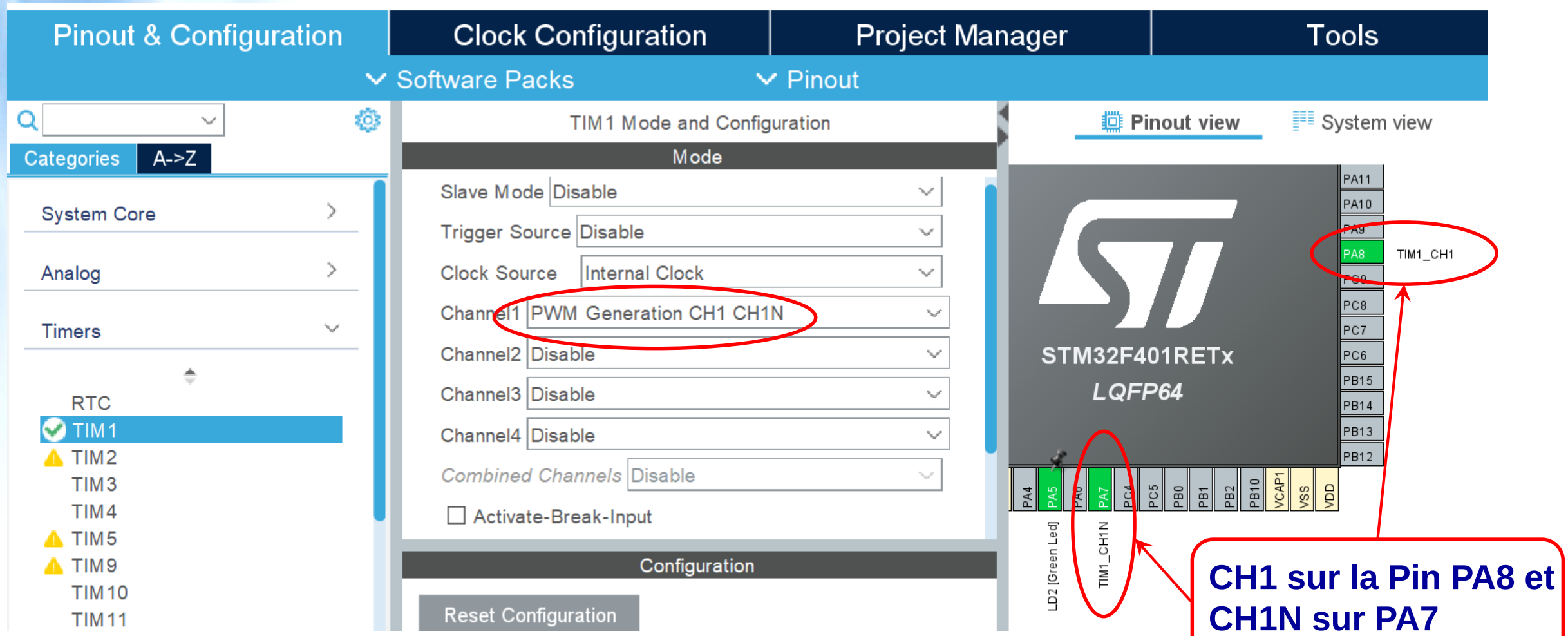
Signaux complémentaires avec temps mort sous STM32:

Sous STM32, les signaux complémentaires sont obtenus en choisissant «PWM Generation CH1 CH1N»





Génération de Signaux PWM Complémentaires



Génération de Signaux PWM Complémentaires

Counter Settings

Prescaler (PSC - 16 bits value)	20
Counter Mode	Up
Counter Period (AutoReload Registe...	999
Internal Clock Division (CKD)	No Division
Repetition Counter (RCR - 8 bits valu...	0
auto-reload preload	Enable

Configurer le PSC = 20

Configurer le ARR = 999

Activer « auto-reload preload »

Break And Dead Time management - Outpu...

Automatic Output State	Disable
Off State Selection for Run Mode (O...	Disable
Off State Selection for Idle Mode (O...	Disable
Lock Configuration	Off
Dead Time	50

Dead Time DT = 50

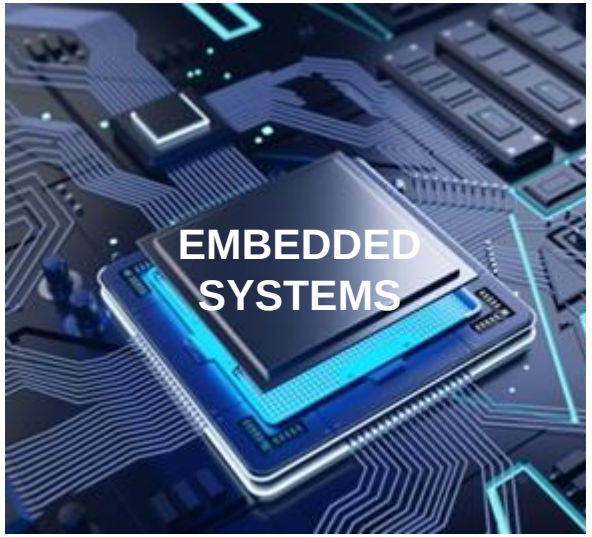
Configurer CCR = 499

PWM Generation Channel 1 and 1N

Mode	PWM mode 1
Pulse (16 bits value)	499
Output compare preload	Enable
Fast Mode	Disable
CH Polarity	High
CHN Polarity	High
CH Idle State	Reset
CHN Idle State	Set

Au repos, CH1 à 0 et CH1N à 1

VERIFIER !!!



FIN !
Questions ?