3 – Modulation en largeur d'impulsion PWM

Objectifs : ***gestion de périphériques (PWM), génération de signaux modulés en largeur d'impulsion, utilisation d'un analyseur logique.***

Préparation: à quelle valeur faut-il régler la période du PWM pour avoir en sortie une fréquence de 10KHz. Expliquez le calcul.

Réalisation : les parties **1.2**, **1.3** et **1.4** doivent être validées par l'encadrant au cours de la séance.

La préparation est obligatoire et vérifiée en début de séance.

1.1 Description

La modulation en largeur d'impulsion (Pulse Width Modulation, PWM) est une solution très utilisée pour permettre le contrôle d'un système électrique à partir d'une commande numérique générée par un processeur. Le principe est de générer un signal logique (valant 0 ou 1), à fréquence fixe mais dont le rapport cyclique est contrôlé numériquement (Figure 1). Une application typique est le contrôle de moteurs électriques par un microcontrôleur.

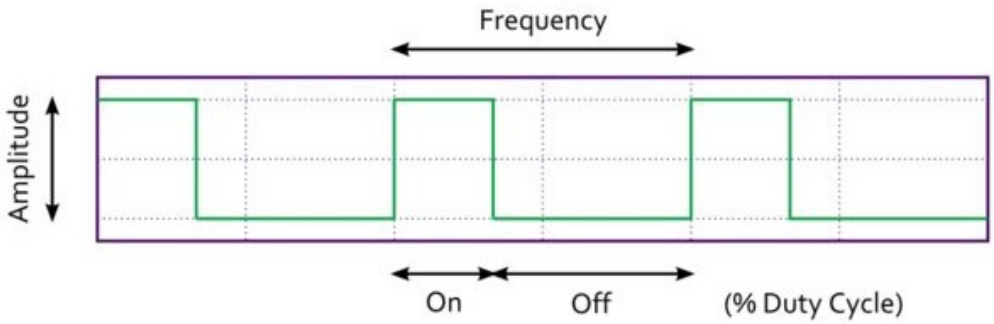


Figure 1. Principe d'un signal PWM et rapport cyclique

Dans le principe, un PWM est un compteur périodique qui commute sur une valeur définie par l'utilisateur. L'exemple de la Figure 2 est programmé pour passer de 1 à 0 sur la valeur 65. La fréquence de fonctionnement du timer est de 84 MHz (autrement dit, le timer s'incrémente tous les 1/84000000 secondes).

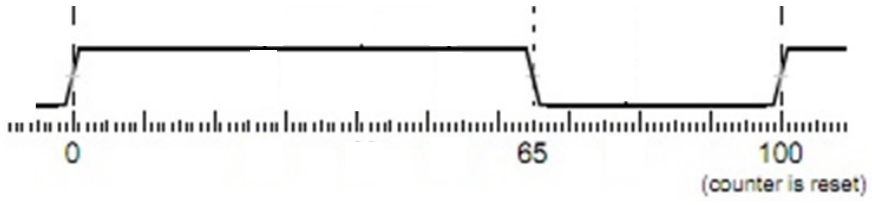


Figure 2. Principe de fonctionnement d'un PWM

La période est réglée à 100, ce qui signifie que lorsque le compteur arrive à 100, il recommence à compter à partir de 0. La fréquence du signal de sortie sera donc de 84 10^6/100 = 840 KHz, avec un rapport cyclique de 65%.

Dans ce TP nous illustrerons les périphériques de génération de signaux PWM pour générer les signaux de commande d'un moteur pas à pas à 10 KHz.

Configuration du projet en C:

- Ouvrir Keil µVision.
- Project → Create new µVision project
  Sélectionner le répertoire `C:\users\elec3\TP_microprocesseurs\TP3` (à créer s'il n'existe pas). **/!\ ne jamais travailler dans Mes Documents, ni sur le Bureau Windows.**
  Nom du fichier : TP3_Timer
- Dans la fenêtre Select Device for Target 'Target 1', cliquer dans STMicroelectronics et sélectionner STM32F4 Series / STM32F407 / STM32F407VG.
- Dans la fenêtre Manage Run-Time Environment, sélectionner: Board Support → STM32F4-Discovery (à sélectionner dans le menu déroulant), CMSIS → CORE, Device → Startup
- Pour pouvoir utiliser les librairies CMSIS pour les timers, cocher Device → StdPeriph Drivers → TIM, Framework et RCC.
- Pour pouvoir utiliser les librairies CMSIS pour les LEDs et le GPIO, cocher aussi Board Support → STM32F4-Discovery → LED et Device → GPIO dans le menu, puis faites OK.
- Dans le menu projet à gauche, clic droit sur Source Group 1, puis Add New Item to Group 'Source Group 1' / C File (.c) / Name: `main_timer.c`, puis Add.
- Essayez de compiler, si vous avez l'erreur « function assert param declared implicitely », clic droit dans Target1 → Options for Target 'Target 1'… → C/C++ → Preprocessor Symbols → Define : `USE_STDPERIPH_DRIVER`
- Remplacer le fichier `system_stm32f4xx.c` par celui qui se trouve sur la page http://users.polytech.unice.fr/~bilavarn/
- Clic droit dans Target1 → Options for Target 'Target 1'… → C/C++ → Preprocessor Symbols → Define et ajouter : `HSE_VALUE=8000000`.

Configuration du debugger ST LINK:

- Dans le menu Options → Debug → ST Link Debugger → Settings → Port: SW.

1.2 Réalisation d'un premier signal par le PWM

Dans un premier temps on va chercher à réaliser un simple signal carré (rapport cyclique 50%) de fréquence 10KHz.

Sur la carte STM32F4-Discovery, un signal PWM est généré par un timer. Nous utiliserons le timer `TIM4` configuré en mode Pulse.
A l'aide des fonctions de la librairie CMSIS (`stm32f4xx_rcc.h`, `stm32f4xx_tim.h`, `GPIO_STM32F4xx.h`), écrire les trois fonctions suivantes qui permettront d'initialiser le timer `TIM4` et de le configurer en mode PWM:

- `void TIM4_Initialize(void)` pour initialiser et démarrer le timer `TIM4`. Les fonctions à utiliser sont:
  - `RCC_APB1PeriphClockCmd` pour activer l'horloge périphérique `RCC APB1` sur lequel est connecté le timer `TIM4`.
  - `TIM_TimeBaseInit` pour configurer le timer (prescaler, mode comptage croissant, période, division d'horloge=1).

- `void TIM4_PWM_Initialize(void)` pour configurer le timer `TIM4` en mode PWM (rapport cyclique 50%, fréquence 10KHz) en utilisant les fonctions :
  - `TIM_OC1Init` pour configurer la sortie PWM Output Compare 1 (mode PWM2, sortie activée, polarité/bas, valeur de commutation dans TIM_PULSE).
  - `TIM_OC1PreloadConfig` pour activer l'utilisation de la valeur de commutation / Capture Compare.

- `void GPIO_AF_Initialize(void)` : il est possible de rediriger la sortie OC1 du timer `TIM4` sur la LED `LED3` (PD.12). Le tableau page 21 du manuel d'utilisation de la carte discovery décrit les possibilités de redirection des sorties des TIMER (ex : TIM4_OC1=TIM4_CH1 sur PD.12). Nous utiliserons cette possibilité pour vérifier que le signal PWM fonctionne en faisant clignoter le LED `LED3`. Les fonctions à utiliser sont:
  - `RCC_AHB1PeriphClockCmd` pour activer l'horloge périphérique `RCC AHB1` sur lequel est connecté le port GPIOD.
  - `GPIO_PinAF` pour activer le mode *alternate function / TIM4* pour la broche concernée.
  - `GPIO_PinConfigure` pour configurer cette broche en mode: Alternate Function, Output Push Pull, Output Speed 100MHz, et no Pull up / down.

On pourra modifier le *prescaler* à 1000 pour observer un clignotement.

Le timer `TIM4` peut ensuite être démarré en utilisant la fonction `TIM_Cmd`.

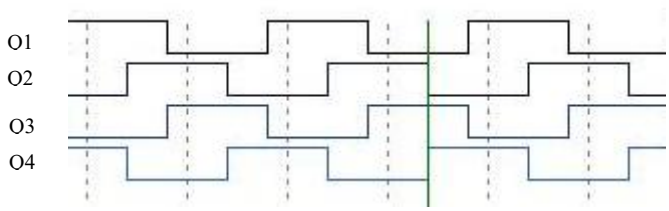
Vérifier à l'aide d'un oscilloscope la génération du signal à 10KHz (remettre le *prescaler* à 0).

1.3 Application à la commande d'un moteur pas à pas

Nous allons maintenant ajouter et écrire le code d'une fonction `pwm_step_motor_control()` qui devra générer les quatre signaux de commande d'un moteur pas à pas. Le moteur pas à pas est étudié en travaux pratiques d'électronique numérique (pour le principe de fonctionnement, se rapporter au TP1). Pour obtenir une rotation du moteur dans le sens horaire, il faut produire la séquence d'excitation suivante :

Q_4	Q_3	Q_2	Q_1
1	0	0	1
0	0	1	1
0	1	1	0
1	1	0	0

Celle-ci correspond aux chronogrammes suivants :



Pour générer les 4 signaux, réalisez les étapes suivantes :

1) le signal `Q3` sera l'inverse du signal `Q1`. En utilisant les modes de fonctionnement de la sortie PWM vous pouvez réaliser l'inversion. Redirigez la sortie OC3 du timer `TIM4` sur la LED `LED5` (PD.14). Avec le *prescaler* à 1000, vérifiez que les 2 LEDS clignotent en opposition.

2) Les signaux `Q2` et `Q4` pourront être générés en utilisant le timer `TIM5` en mode *center-aligned1* afin de décaler Q2 de T/4 par rapport à Q1. Il faudra ajouter deux fonctions `TIM5_Initialize` et `TIM5_PWM_Initialize`. Le signal `Q4` sera l'inverse du signal `Q2`. Utilisez le tableau p21 du manuel de la carte discovery pour rediriger les sorties `OC1` et `OC3` du `TIM5` sur le port A.

Avec le *prescaler* à 0 et à l'aide d'un oscilloscope, vérifiez que la fréquence des signaux `Q1` à `Q4` est bien à 10 kHz puis vérifiez l'opposition entre `Q1` et `Q3` et entre `Q2` et `Q4`.

A l'aide de l'enseignant, testez ensuite le fonctionnement du programme dans son ensemble en vérifiant les quatre chronogrammes à l'analyseur logique.

Annexe : utilisation de l'analyseur logique

L'analyseur logique comme son nom l'indique, est un outil utilisé pour observer des signaux logiques. Pour l'utiliser, il suffit de connecter les signaux à visualiser à l'entrée d'un des boitiers *Logic Pod*, et de lancer une capture par les commandes Go/Stop dans le menu Trigger. Le seul réglage à faire est la fréquence d'échantillonnage qui doit être sensiblement supérieure à la fréquence du signal à observer (Théorème de Shannon). Avant chaque mesure, faites *Clear Data Buffer* pour éviter de saturer votre station de travail à cause du très grand nombre de données échantillonnées.