École affiliée IMT

Université de Lyon

Université Jean Monnet Saint-Étienne

Cycle Initial en Technologies de l'Information de Saint-Étienne

# Commande MLI d'un onduleur en pont monophasé

Lucas Lescure - Eva Maturana

# Table of Content



## Introduction

L'alimentation d'un moteur en courant alternative sinusoïdale est utile dans des application ou l'on souhaite un fonctionnement à vitesse constante. Ces sources sont typiquement fixées et donc pas adaptés pour des fonctionnement à vitesse variable. On préfèrera donc utiliser une commande MLI[1], permettant de controller la puissance fournie au moteur à partir d'une source continue de courant en faisant varier la largeur des impulsions.

Cependant lorsqu'une charge est alimenté par une commande MLI, il s'y présente des harmoniques sur le réseau qui nuisent la qualité de transfert de l'énergie, et interfèrent avec d'autres appareils sur le circuit. C'est pourquoi on cherchera donc comment supprimer les harmoniques multiples de rang 3 et 5 pour améliorer ce transfert d'énergie.

[1] Modulation en Largeur d'Impulsion

# 1. Objectifs du TP

L'objectif de ce TP sera alors d'utiliser une structure en pont monophasé à IGBT[2] en commande MLI précalculée avec suppression des harmoniques 3 et 5.

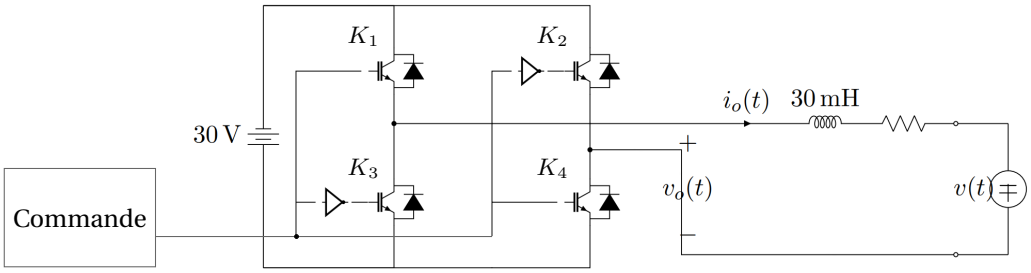



*Figure 1.1. Structure en pont monophasé à IGBT*

On rappelle aussi la forme d'onde de la tension de sortie qui est la suivante::

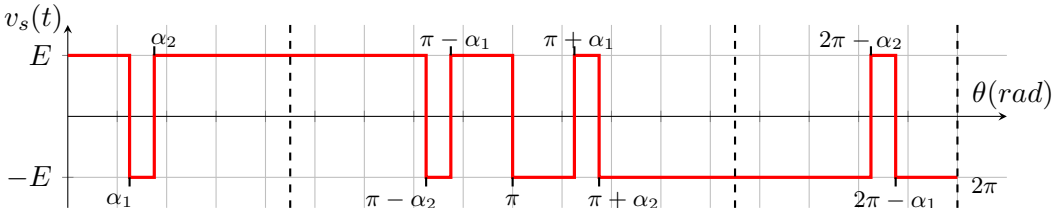



*Figure 1.2. Forme d'onde $v_s(t)$*

On utilisera, comme ci-présent sur la forme d'onde, 2 degrés de liberté $\alpha_1$ et $\alpha_2$ qui permettrons la suppression des harmoniques sur le signal de commande MLI.

## 1.1. Analyse spectrale du signal MLI

D'après la Figure 1.2. on a :

- $v_s(t) = -v_s(t)$: Donc c'est une fonction impaire et n'admet que des sinus dans le développement en série de Fourier de $v_s(t)$.
- $v_s\left(t + \frac{T}{2}\right) = -v_s(t)$: Donc il y a symétrie de de glissement et il n'y aura que des termes impaires dans la série de Fourier.

Ceci nous permet alors de réduire l'intervalle d'étude de la série de Fourier à $\frac{T}{4}$, soit:

$$\widehat{V}_{s_{2k+1}} = \frac{4E}{\pi}\left[\int_0^{\alpha_1} \sin\left((2k+1)\theta\right)\,d\theta - \int_{\alpha_1}^{\alpha_2} \sin\left((2k+1)\theta\right)\,d\theta + \int_{\alpha_2}^{\frac{\pi}{2}} \sin\left((2k+1)\theta\right)\,d\theta\right]$$

$$\widehat{V}_{s_{2k+1}} = \frac{4E}{(2k+1)\pi}\left[1 - 2\cos\left((2k+1)\alpha_1\right) + 2\cos\left((2k+1)\alpha_2\right)\right]$$

On peut alors écrire:

$$v_s(t) = \widehat{V}_{s_{2k+1}} \cdot \sin\left((2k+1)\omega t\right)$$

$$v_s(t) = \frac{4E}{(2k+1)\pi}\left[1 - 2\cos\left((2k+1)\alpha_1\right) + 2\cos\left((2k+1)\alpha_2\right)\right] \cdot \sin\left((2k+1)\omega t\right)$$

Ainsi pour trouver les valeurs de $\alpha_1$ et $\alpha_2$ afin d'éliminer les harmoniques 3 et 5 il suffit de résoudre le système :

$$\begin{cases} \widehat{V}_{s_3} = 0 = 1 - 2\cos(3\alpha_1) + 2\cos(3\alpha_2) \\ \widehat{V}_{s_1} = 0 = 1 - 2\cos(5\alpha_1) + 2\cos(5\alpha_2) \end{cases} \Leftrightarrow \begin{cases} \alpha_1 = 23.6° \\ \alpha_2 = 33.3° \end{cases}$$

[2] Insulated-Gate Bipolar Transistor

### 1.2. Stratégie de commande

En ce référant encore à la Figure 1.2. on peut établir la commande de l'onduleur en pont qui sera:

- Pour $v_s = E$ alors les interrupteurs $K_1$ et $K_4$ doivent êtres ouvert donc $v_c = 10\,V$.
- Pour $v_s = -E$ alors les interrupteurs $K_2$ et $K_3$ doivent êtres ouvert donc $v_c = 0\,V$ (ou $-10\,V$) suivante le type de pont utilisé.

La commande devra donc avoir la forme suivante:

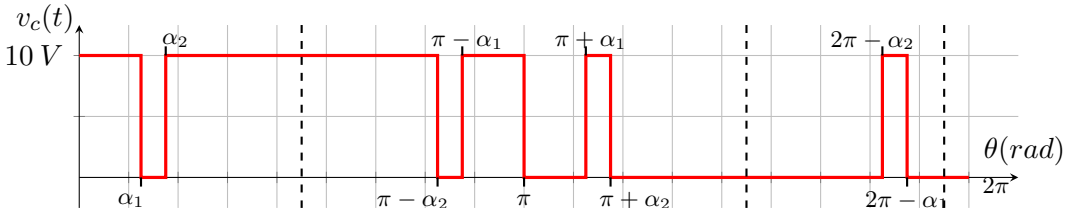
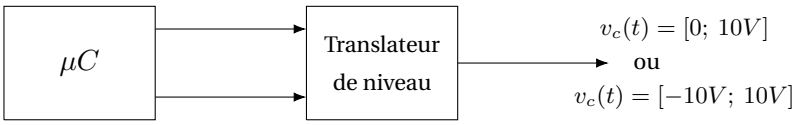



*Figure 1.3. Forme d'onde de la commande $v_c(t)$*

Le schéma Synoptique de l'ensemble de la commande du convertisseur est la suivante:




## 2. Pré-étude

On peut découper le signal $v_s(t)$ comme une succession de signaux créneaux de période et durée à l'état haut variables. Cependant ces signaux ont chacun un $\alpha$ et $T$ différent comme on peut voir ci-dessous:

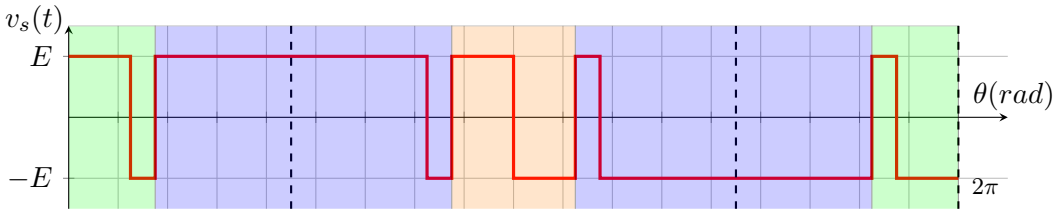



*Figure 2.1. Forme d'onde $v_s(t)$ découpée en 5 créneaux*

On peut relever ces signaux créneaux dans le tableau suivant avec $\Delta_k = \alpha'_k \cdot T_k$:

| $\Delta_k$ | $\alpha_1$ | $\pi - 2\alpha_2$ | $\alpha_1$ | $\alpha_2 - \alpha_1$ | $\alpha_2 - \alpha_1$ |
|---|---|---|---|---|---|
| $T_k$ | $\alpha_2$ | $\pi - \alpha_2 - \alpha_1$ | $2\alpha_2$ | $\pi - \alpha_1 - \alpha_2$ | $\alpha_2$ |

Pour des raison de commodité on choisira une horloge pour le PWM qui aura 36000 période d'horloge pour une période du PWM, soit 36000 $T_{clock} = T_{total}$. Ainsi pour un réseau de période 20 $ms$ (50 $H_z$), on a une période d'horloge à $T_{clock} = 55.5\,ns$ et de fréquence $f_{clock} = 1.8\,MH_z$

## 3. Réalisation

### 3.1. Assemblage

On va réaliser la commande de l'onduleur à l'aide du logiciel PSoC[3] en utilisant la carte microcontrôlleur CY8CKIT-042 PSoC 4 Pioneer Kit du fabricant Inifneon sur laquel nous avons un chip CY8C4225AXI-483.

[3] Programmable System-on-Chip

Sur un nouveau projet vide, on instancie le `PWM(TCPPWM mode)` dans la fenêtre `TopDesign`. On configure alors le composant en cochant `On terminal count` puis on insère dans la case `Period` la valeur de $T_1 = \alpha_1 = 2360$, et dans la case `Compare` la valeur du rapport cyclique $\Delta_1 = \alpha_2 = 3330$.

On ajoute de la même façon le composant `Clock` (renomée "clock"), en lui configurant une fréquence $f_{clock} = 1.8\,MH_z$, le composant `Interrupt`(renomée "isr") et un `Digital Output Pin`(renomé "PWM_OUT"). On relie ensuite ces composants par un fil au PWM avec le composant `Clock` allant vers l'entrée `clock`, le composant `Interrupt` branché à la sortie `interrupt`, et le `Digital Output Pin` à la sortie `line`.

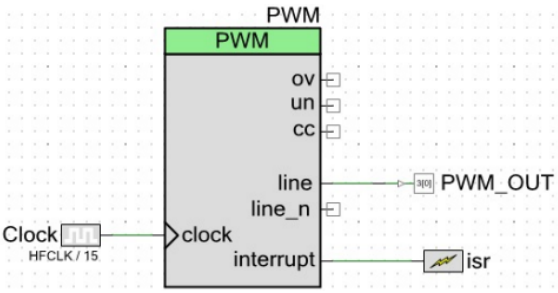



*Figure 3.1. Cablage des composant en `TopDesign` du projet*

## 3.2. Programmation

On utilisera des variable globale dans le fichier `main.c` pour les durées à l'état haut et les périodes.Ainsi on fera intervenir dans des tableau `volatile`, `delta[5]` et `T[5]`, les valeurs précédemment trouvés lors de la pré-étude. Le qualificatif `volatile` est utilisé pour ne pas optimiser l'accès à la variable de façon à ce que le tableau puisse être utilisé par l'interruption même si une tache concurente l'utilisait.

On vas aussi définir des macros `alpha1`,`alpha2` et `demip` pour remplacer la valeur de $\alpha_1$, $\alpha_2$ et celle de la demi période.

```
#include "fonction.h"

#define alpha1 2360
#define alpha2 3330
#define demip 18000

volatile uint16_t delta[5]={alpha1, demip-2*alpha2, alpha1, alpha2-alpha1, alpha2-alph
volatile uint16_t T[5]={alpha2, demip-alpha2-alpha1, 2*alpha1, demip-alpha1-alpha2, al
```

*Figure 3.2. Déclaration et initialisation des variables globales dans le `main.c`*

Pour l'interruption on utilisera la fonction `isr_StartEx()` qui prendra en paramètre le nom de notre interruption, ici `moninter`. On définira cette interruption dans le fichier `fonction.c` pour ceci on créé d'abord le fichier `fonction.h` dans lequel on utilise un API[4] avec la libraire `project.h`. Grâce à celle ci on peut alors déclarer une interruption `moninter` de la façon suivante.

```
#ifndef INTERRUPT_HEADER
        #define INTERRUPT_HEADER
        #include "project.h"
        void startup(void);
        CY_ISR_PROTO(moninter);
#endif
```

*Figure 3.3. Contenu du fichier `fonction.h`*

[4]Application Programming Interface

On en profite aussi pour déclarer la fonction `startup()` qui permettra d'initialiser les périphériques au démarrage.

On code alors dans le fichier `fonction.c` les définitions des fonctions `startup()` et `CY_ISR(moninter)` qui sera déclenchée dans l'évènement d'une interruption. Dans le cas de l'interruption on affecte à une variable globale `source`, la source de l'interruption en veillant à plus tard effacer cette source pour éviter de rentrer dans une boucle infinie.

Le NVIC[5] gère les demandes d'interruption et les envois au processeur. Quand une interruption est déclenchée un signal est envoyé au NVIC pour executer l'interruption correspondante définie dans notre programme, à la fin de l'interruption il faut remettre ce signal à 0 pour eviter que le NVIC ne réexécute l'interruption, c'est pourquoi on fait intervenir la fonction `PWM_ClearInterrupt()` qui éffface la source d'interruption au NVIC.

Pour récupérer la source on utilise une variable globale `source` à laquelle on lui affecte la valeur de l'interruption avec la commande `GetInterruptSourceMasked()` pour qu'elle puisse ensuite être gérée par le NVIC dans la commande `PWM_ClearInterrupt(source)`.

Dans l'interruption on affecte les valeurs de `Compare` et `Period` du MLI en utilisant les tableaux définis dans le fichier `main.c` donc il faut utiliser le mot clé `extern` pour que l'interruption puisse les accéder.

Cette affectation doit être réalisé à chaque coup d'interruption c'est à dire à chaque fois que la période d'un créneaux est finie. Pour passer d'un créneau à l'autre on définis une variable locale `i` à l'aide du mot clé `static` et qui permettra à la variable de réserver en emplacement mémoire qui ne sera pas désallouée quand le compilateur sort de l'interruption. On évite ainsi une erreur de segmentation car sans ce mot clé, la mémoire sera désallouée mais à la prochaine interruption on réutilise cette variable qui a déjà été définie mais qui n'as plus d'emplacement mémoire.

```
#include "fonction.h"
extern uint16_t delta[5];
extern uint16_t T[5];

void startup () {
        Clock_Start();
        PWM_Start();
        isr_StartEx(moninter);
}

CY_ISR(moninter) {
        // Declaration de la variable source d'interruption
        uint32 source;
        //Affectation de la source d'interruption a la variable source
        source = GetInterruptSourceMasked();
        //Code a inserer
        static volatile uint8_t i=0;
        i = (i + 1) % 5;
        //Utilisation des fonctions PWM_WritePeriod() et PWM_WriteCompare()
        PWM_WriteCompare(delta[i]);
        PWM_WritePeriod(T[i]);
        //Effacage de la source d'interruption pour eviter la boucle infinie
        PWM_ClearInterrupt(source);
}
```

*Figure 3.4. Contenu du fichier `fonction.c`*

[5] Nested Vector Interrupt Controller

```
int main() {
        CyGlobalIntEnable //Enable global interrupts

        //Place your initialisation/startup code here
        startup();

        for(;;) {
                //Place your application code here
        }
}
```

*Figure 3.5. Contenu du main dans le fichier* `main.c`

## 3.3. Mise en Pratique

On effectue le teste en utilisant un oscilloscope ayant le module FFT pour obtenir le spectre du signal de sortie du bloc MLI.

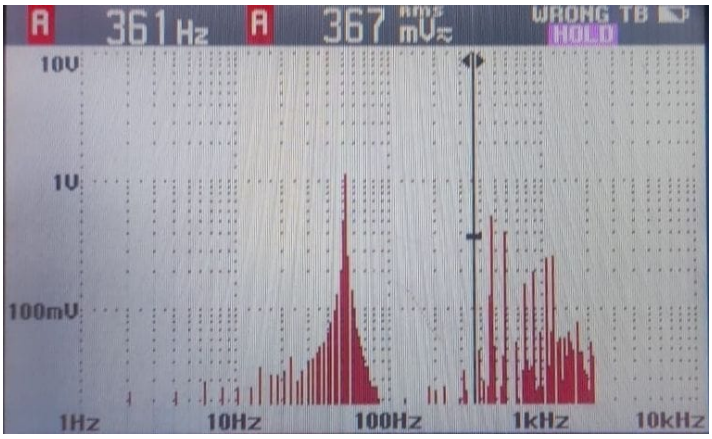



Figure 3.6. Spectre du signal avec harmoniques 3 et 5 supprimées

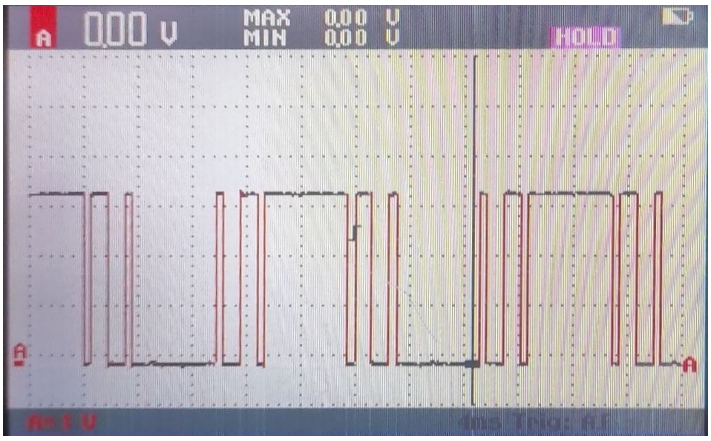



*Figure 3.7. Chronogramme du signal de commande* $v_c(t)$

On obtient comme nous le voulions un spectre dont les harmoniques de rang 3 et 5 ont été supprimés et un signal de commmande conforme à ce qui à été trouvé lors de la pré-étude.