

## Projet de Simulation et Synthèse des Matériels (TP8) : Contrôleur BLDC



**Auteur :**

*PABOEUF Alexandre & PESCAZ Maxime*

**Encadrant :**

*Dr THIEBOLT François*



**9 avril 2025**

## Table des matières

<b>1</b>	<b>Introduction</b>	<b>2</b>
<b>2</b>	<b>Rappels théoriques</b>	<b>2</b>
2.1	Le moteur BLDC . . . . .	2
2.2	Contrôle de vitesse . . . . .	2
2.3	Ramp-up / Ramp-down . . . . .	2
<b>3</b>	<b>Collaboration</b>	<b>2</b>
<b>4</b>	<b>Spécifications du projet</b>	<b>3</b>
4.1	Paramètres de synthèse . . . . .	3
4.2	Architecture globale . . . . .	3
<b>5</b>	<b>Conception VHDL</b>	<b>4</b>
5.1	Création du projet . . . . .	4
<b>6</b>	<b>Détails d'implémentation</b>	<b>6</b>
6.1	Logique de commutation . . . . .	6
6.2	Génération du PWM . . . . .	6
6.3	Gestion du ramp-up/down . . . . .	7
6.4	Interface capteur (Hall / optique) . . . . .	8
<b>7</b>	<b>Simulation et validation fonctionnelle</b>	<b>8</b>
7.1	Environnement de test . . . . .	8
7.2	Résultats de simulation . . . . .	8
<b>8</b>	<b>Implémentation sur FPGA</b>	<b>9</b>
8.1	Synthèse . . . . .	9
<b>9</b>	<b>Conclusion</b>	<b>9</b>

# 1 Introduction

La maîtrise de la simulation et de la synthèse sur FPGA constitue un enjeu majeur dans la conception de systèmes embarqués. Dans le cadre de l'UE «Simulation et synthèse des matériels», nous avons développé et implémenté un contrôleur de moteur BLDC (Brushless DC) sur FPGA. Le présent document retrace l'ensemble du travail, depuis les principes de fonctionnement d'un moteur BLDC, la conception des modules en VHDL, leur simulation, jusqu'à l'implémentation sur une carte FPGA.

L'objectif principal est de piloter un moteur BLDC – par exemple, un moteur de disque dur modifié – en assurant :

- la commutation des 3 phases et le séquencement des signaux de commande,
- la génération d'un signal PWM (modulation de largeur d'impulsion) pour réguler la vitesse,
- une montée et une descente progressives en vitesse (ramp-up/ramp-down),
- la synchronisation à partir d'un capteur de position (capteur Hall ou optique).

## 2 Rappels théoriques

### 2.1 Le moteur BLDC

Un moteur BLDC (Brushless Direct Current) est un moteur synchrone dont le rotor est équipé d'aimants permanents et dont le stator comporte des enroulements. La commutation du courant, qui permet de faire tourner le rotor, est assurée par une électronique de puissance plutôt que par des balais mécaniques. Pour cela, la position du rotor est détectée par des capteurs (effet Hall ou capteurs optiques) ou estimée via une méthode *sensorless*.

### 2.2 Contrôle de vitesse

La vitesse d'un moteur BLDC est déterminée par la tension moyenne appliquée aux enroulements. Dans notre design FPGA, cette tension est réglée numériquement à l'aide d'un signal PWM. Une période de PWM est définie, puis le *duty cycle* (rapport cyclique) est modulé pour délivrer plus ou moins d'énergie au moteur, influençant ainsi la vitesse.

### 2.3 Ramp-up / Ramp-down

Pour éviter des variations brusques et les pics de courant associés, une rampe de montée et une rampe de descente sont mises en place. Cette technique permet d'ajuster progressivement le *duty cycle* du signal PWM vers une consigne finale.

## 3 Collaboration

La réussite de ce projet repose non seulement sur l'implémentation technique, mais également sur une bonne collaboration entre les membres de l'équipe. Pour assurer une

coordination efficace et partager rapidement les avancées, nous avons adopté les pratiques suivantes :

- **Gestion de versions avec Git** : Chaque membre de l'équipe a travaillé sur des branches dédiées et a régulièrement effectué des commits pour suivre l'évolution du code de chaque membre à distance et garder un suivi optimal.

[Lien du Dépôt Git](#)

- **Communication via Discord** : Pour échanger rapidement sur des questions techniques et partager des retours d'expérience, nous avons utilisé Discord. Cette plateforme de communication nous a permis d'organiser des réunions virtuelles, de partager des captures d'écran et de discuter en temps réel des problèmes rencontrés.

Grâce à ces méthodes collaboratives, nous avons pu avancer rapidement et de manière structurée dans le développement du contrôleur BLDC, tout en assurant la qualité et la cohérence du code.

## 4 Spécifications du projet

### 4.1 Paramètres de synthèse

Le cahier des charges impose la définition d'une valeur **MAX-CPT**, qui représente le nombre de coups d'horloge pour parcourir un cycle de phase complet. Par exemple, avec une horloge principale à 1 MHz, pour obtenir un signal de phase à environ 50 Hz, il est courant de fixer **MAX-CPT** à 20000. De plus, la consigne de ramp-up/ramp-down est définie par le paramètre **RAMP\_INC\_PERIOD** (ici 500 cycles).

### 4.2 Architecture globale

Le design se compose de plusieurs modules principaux :

- **Module de commutation** (sequencer.vhd) qui génère les six signaux de commande (A\_H, A\_L, B\_H, B\_L, C\_H, C\_L) en fonction du code rotor.
- **Module PWM** (pwm\_generator.vhd) qui produit un signal PWM en comparant un compteur à la consigne.
- **Module ramp-up/down** (ramp\_controller.vhd) qui ajuste progressivement le *duty cycle* vers la consigne cible.
- **Interface capteur** (sequencer.vhd) qui convertit le signal du capteur de position en information exploitable par la logique de commutation.

## 5 Conception VHDL

### 5.1 Crédation du projet

Nous avons commencé par lancer Vivado et créer un nouveau projet de type RTL. La Figure 1 montre la fenêtre utilisée pour définir le nom du projet, son emplacement et le type de projet.

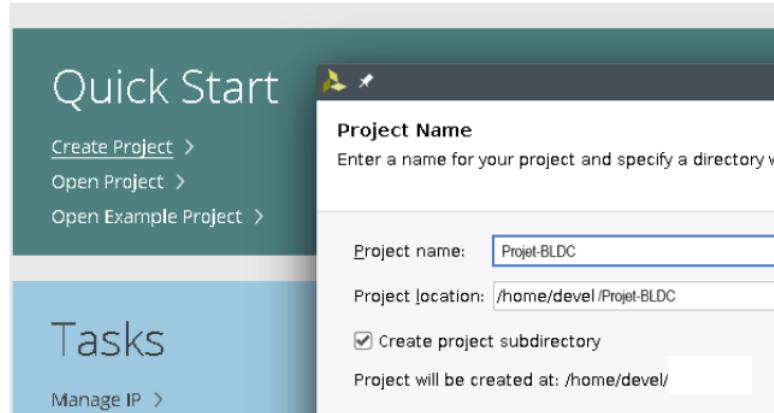


FIGURE 1 – Crédation d'un nouveau projet Vivado

Nous avons ensuite sélectionné la carte cible, ici une Zybo Z7-20 (Figure 2). Enfin, nous avons ajouté les fichiers sources (fichiers .vhd et test bench) dans les sections Design Sources et Simulation Sources. Un répertoire récapitulatif est présenté dans la Figure 6.3.

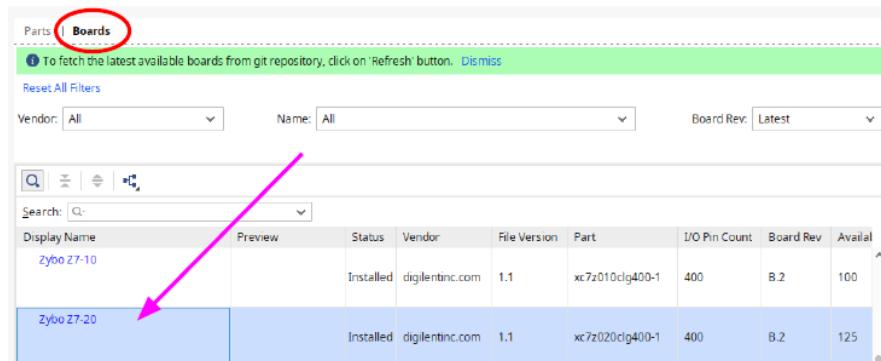


FIGURE 2 – Sélection de la carte Zybo Z7-20 comme cible

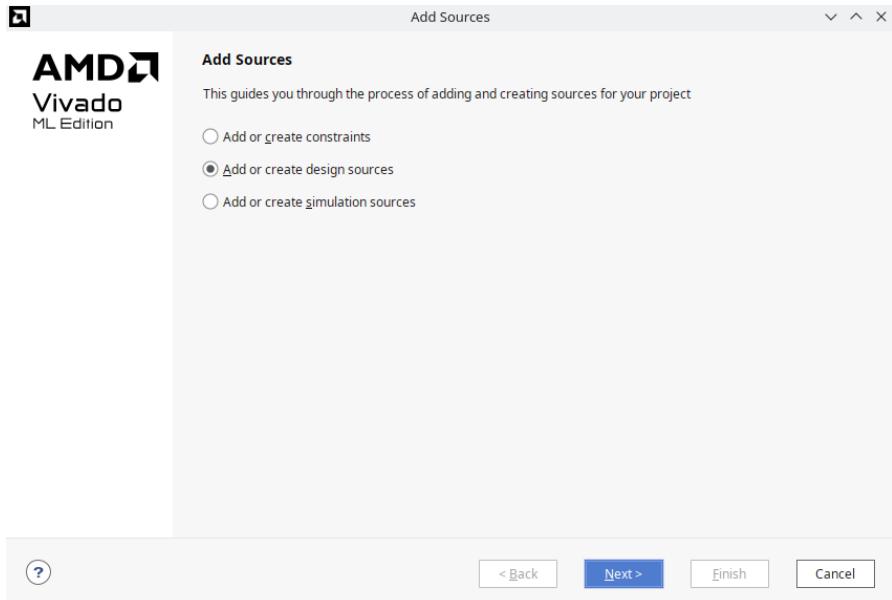


FIGURE 3 – Sélection des fichiers sources VHDL

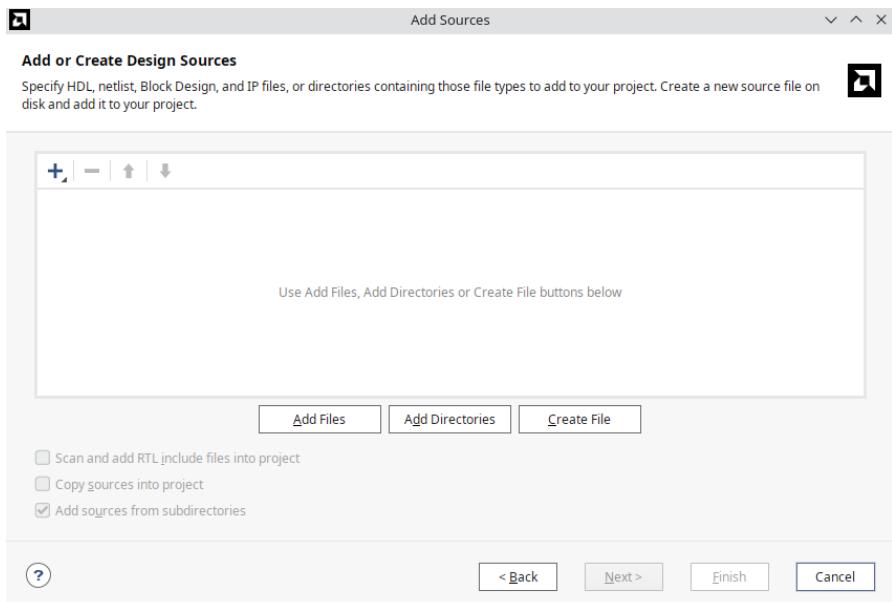


FIGURE 4 – Ajout des fichiers VHDL dans Vivado

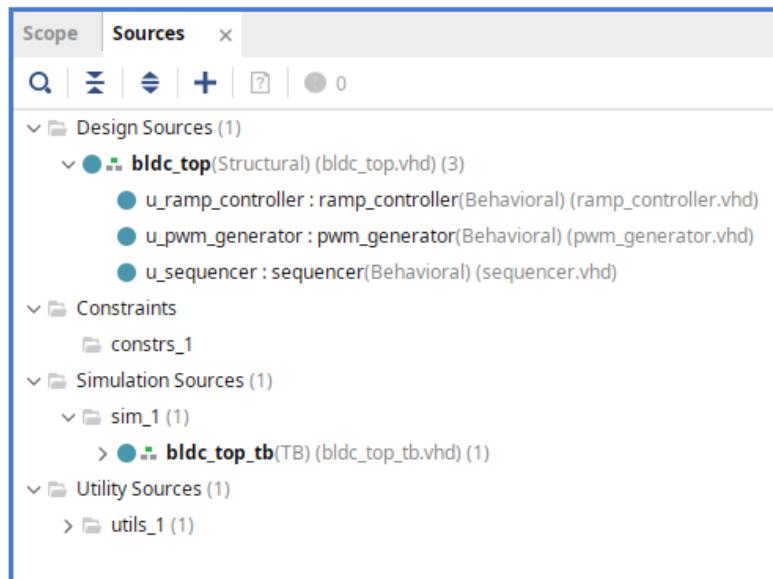


FIGURE 5 – Répertoire des fichiers sources

## 6 Détails d’implémentation

### 6.1 Logique de commutation

Ce module détermine, en fonction du code rotor issu des capteurs, quelles phases activer. Même si dans notre projet le module de commutation est intégré dans le fichier `sequencer.vhd`, nous présentons ici un extrait représentatif :

```

when "001" =>
    -- A+ / B- / C off
    reg_A_H <= pwm_in; -- phase A high reçoit le PWM
    reg_A_L <= '0';
    reg_B_H <= '0';
    reg_B_L <= '1'; -- phase B low au niveau bas
    reg_C_H <= '0';
    reg_C_L <= '0';

```

Ce bloc de code, qui se trouve dans le processus réactif au front montant de l’horloge, gère la commutation pour l’état "001". Pour les autres états ("011", "010", "110", "100" et "101"), une logique similaire est appliquée afin de couvrir les 6 étapes de commutation requises pour un système triphasé.

### 6.2 Génération du PWM

Le module PWM (fichier `pwm_generator.vhd`) génère une onde en comparant un compteur à la consigne de duty cycle :

```

-- PROCESS de comptage
process(clk, reset_n)
begin
    if (reset_n = '0') then
        cpt <= 0;
    elsif rising_edge(clk) then
        if (cpt >= MAX_CPT - 1) then
            cpt <= 0; -- On reboucle
        else
            cpt <= cpt + 1;
        end if;
    end if;
end process;

-- Génération du PWM en comparant cpt et duty_cycle
pwm_out <= '1' when cpt < duty_cycle else '0';

```

Ce mécanisme assure que, pour chaque période, le signal PWM est à "1" tant que le compteur est inférieur à la consigne `duty_cycle` et passe à "0" ensuite.

### 6.3 Gestion du ramp-up/down

Dans le fichier `ramp_controller.vhd`, le module de ramp-up/down ajuste progressivement le *duty cycle* pour limiter les variations brusques :

```

-----  

-- PROCESS : Gère le timer de rampe et ajuste current_duty_cycle vers la consigne  

-----  

process(clk, reset_n)
begin
    if (reset_n = '0') then
        current_duty_cycle <= 0;
        ramp_timer <= 0;
    elsif rising_edge(clk) then
        -- Incrémente le timer
        if (ramp_timer < RAMP_INC_PERIOD) then
            ramp_timer <= ramp_timer + 1;
        else
            -- Quand on atteint RAMP_INC_PERIOD, on remet ramp_timer à 0,
            -- et on fait un pas vers la consigne
            ramp_timer <= 0;
            if (current_duty_cycle < target_duty_cycle) then
                current_duty_cycle <= current_duty_cycle + 1;
            elsif (current_duty_cycle > target_duty_cycle) then
                current_duty_cycle <= current_duty_cycle - 1;
            else
                current_duty_cycle <= current_duty_cycle; -- Déjà égal, on ne bouge pas
            end if;
        end if;
    end if;
end process;

ramped_duty_cycle <= current_duty_cycle;

```

Ici, toutes les `RAMP_INC_PERIOD` périodes d'horloge, le signal `current_duty_cycle` est rapproché de `target_duty_cycle` d'un point, ce qui lisse la transition de commande.s

## 6.4 Interface capteur (Hall / optique)

Dans notre design, le signal de position du rotor est directement capté par l'entrée `hall_code` dans le fichier `sequencer.vhd`. Contrairement à un traitement intermédiaire utilisant une variable distincte, notre implémentation exploite directement `hall_code` dans la structure `case` pour déterminer la séquence de commutation. Dans notre fichier, le signal `hall_code` est utilisé directement pour décider des états de sortie. Cela simplifie le design en réduisant le nombre de signaux intermédiaires.

# 7 Simulation et validation fonctionnelle

## 7.1 Environnement de test

Pour vérifier le comportement du contrôleur, nous avons développé un test bench (`bldc_top_tb.vhd`) qui :

- Génère une horloge simulée à 1 MHz,
- Applique un reset asynchrone actif bas,
- Fait varier les valeurs de `hall_code` pour simuler la rotation du rotor,
- Modifie la consigne `target_duty_in` afin de tester l'effet de la rampe.

## 7.2 Résultats de simulation

Les simulations montrent clairement :

- La séquence de commutation adaptée au `hall_code`,
- La génération du signal PWM selon la consigne,
- Une montée/descente progressive du *duty cycle* grâce au module de ramp-up/down.

La Figure 6 ci-dessous illustre un exemple de chronogramme obtenus en simulation (via Vivado).

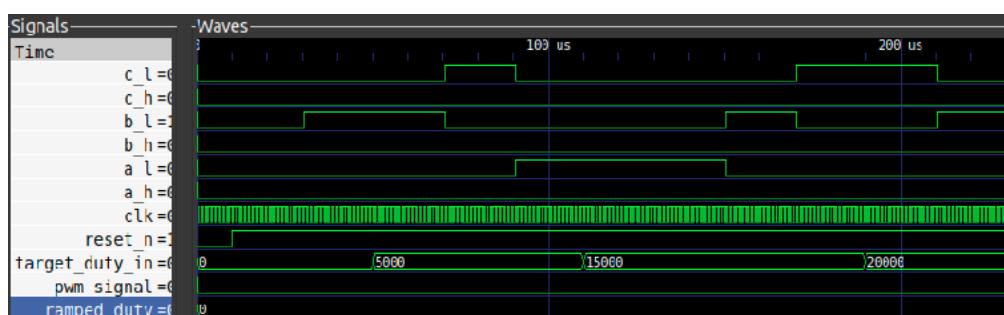


FIGURE 6 – Chronogramme de simulation

## 8 Implémentation sur FPGA

Après avoir validé notre design en simulation, nous avons procédé à l'implémentation sur FPGA.

### 8.1 Synthèse

La synthèse a été réalisée en utilisant le fichier top-level `bldc_top.vhd` et en associant le script `bldc_synth.pre.tcl` afin de fixer automatiquement nos génériques (par exemple `MAX_CPT` et `RAMP_INC_PERIOD`). L'extrait de code suivant illustre le contenu du script TCL, Figure 7), tandis que la Figure 8 présente un aperçu du rapport de synthèse générée.

```
set_property generic {MAX_CPT=20000 RAMP_INC_PERIOD=500} [current_fileset]
```

FIGURE 7 – script TCL

Ce script affecte les paramètres `MAX_CPT` et `RAMP_INC_PERIOD` au fichier de synthèse courant (`[current_fileset]`), garantissant ainsi que ces valeurs soient prises en compte dès que vous lancez la synthèse via Vivado.

Report Instance Areas:			
	Instance	Module	Cells
1	top		204
2	u_pwm_generator	pwm_generator	52
3	u_ramp_controller	ramp_controller	116
4	u_sequencer	sequencer	9

FIGURE 8 – Rapport Synthèse

Dans le rapport de synthèse (Figure 8), on constate que :

- Le **module PWM** (`u_pwm_generator`) utilise environ 52 cellules,
- Le **module de ramp-up/down** (`u_ramp_controller`) en utilise 116,
- Le **module de commutation** (`u_sequencer`) seulement 9.

De manière générale, l'occupation en LUT et FF reste très faible, ce qui indique une implémentation légère, facilement adaptée à la majorité des plateformes FPGA de la gamme Xilinx 7 Series (ou équivalentes).

## 9 Conclusion

Ce projet de simulation et synthèse des matériels (TP8) autour d'un moteur BLDC nous a permis de mettre en pratique plusieurs notions essentielles :

1. La conception VHDL modulaire avec des blocs séparés (commutation, PWM, ramp-up/down),
2. La simulation et la vérification d'un système temps réel complexe,
3. L'implémentation sur FPGA en incluant la synthèse
4. Le pilotage d'un moteur BLDC et l'importance d'un retour de position pour la synchronisation de la commutation,
5. La mise en place d'une gestion progressive (ramp-up/down) pour limiter les variations de couple et les appels de courant.

Le système final satisfait aux exigences de base, en générant une commande triphasée avec modulation de vitesse via PWM et synchronisation par retour capteur. Les perspectives d'amélioration pourront inclure l'intégration de techniques de contrôle plus avancées ou un asservissement complet basé sur le retour de vitesse ou de position.