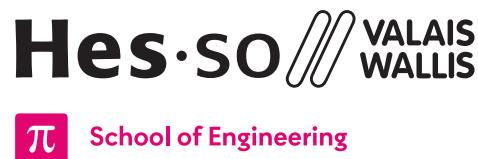


## Cursor (Cur)

Course Conception Numérique (CNum)



**Orientation:** Informatique et systèmes de communication (ISC)

**Spécialisation:** Data Engineering (DE)

**Cours:** Conception Numérique (CNum)

**Auteurs:** Silvan Zahno, Axel Amand, François Corthay, Christophe Bianchi

**Date:** 01.12.2025

**Version:** v3.1



# Contenu

1	Introduction .....	3
2	Spécification .....	4
2.1	Fonctions .....	4
2.2	Circuit .....	4
2.3	Scénario (exemple) .....	6
2.4	Projet HDL-Designer .....	6
3	Composants .....	7
3.1	Chariot .....	7
3.2	Circuit de commande de moteur .....	7
3.2.1	Moteur à courant continu .....	8
3.3	Codeur (encodeur) .....	9
3.4	Reed-Relais .....	10
3.5	Carte FPGA .....	10
3.6	Boutons et LEDs .....	11
4	Implémentation des rampes moteur .....	12
4.1	Concept de base des rampes .....	12
4.2	Profil de rampe naïf .....	13
4.3	Problèmes et améliorations de la solution naïve .....	15
4.4	Rampe finale .....	16
4.5	Vérification des hypothèses .....	17
5	Evaluation .....	18
6	Premières étapes .....	19
6.1	Tips .....	19
	Glossaire .....	20



# 1 | Introduction

L'objectif de ce projet est d'appliquer concrètement les connaissances acquises à travers un exemple pratique en fin de semestre. Il consiste à contrôler un moteur à courant continu afin de déplacer précisément un chariot le long d'une vis vers des positions prédéfinies. Ce système de positionnement peut être observé dans la [Figure 1](#).

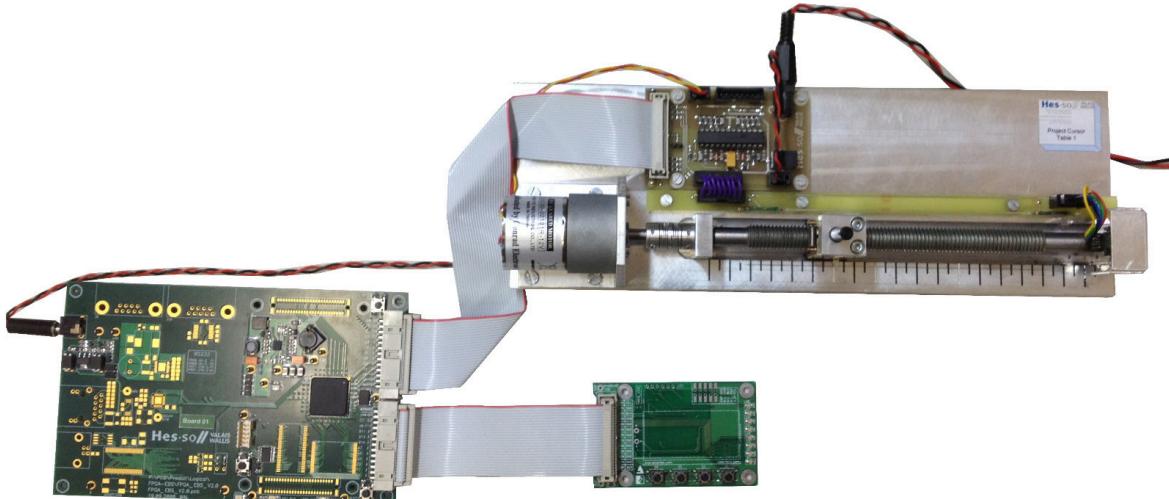


Figure 1 - Équipement du curseur (EBS2)

Les spécifications minimales (voir [Chapitre 2](#)) doivent être respectées, mais les étudiants sont encouragés à implémenter des fonctionnalités supplémentaires. Par exemple, un écran **Liquid Crystal Display (LCD)** peut être utilisé pour afficher diverses informations.



L'implémentation de fonctionnalités supplémentaires permettra d'obtenir des points supplémentaires dans l'évaluation finale.



## 2 | Spécification

### 2.1 Fonctions

Les fonctions de base sont définies comme suit :

- Lorsque la touche **restart** est appuyée, le curseur se déplace vers la position de départ indiquée par un relais reed (Chapitre 3.4) situé à proximité du moteur à courant continu (Chapitre 3.2.1).
- Lorsque l'on appuie sur la touche Position<sub>1</sub>, le curseur doit d'abord accélérer régulièrement vers la position 1 ( $p_1$ ), puis avancer à pleine vitesse et enfin ralentir régulièrement pour s'arrêter à la position 1 ( $p_1$ ). Cela peut se faire à partir de la position de départ ou de la position 2, voir Figure 2.
- Lorsque l'on appuie sur la touche Position<sub>2</sub>, le curseur doit d'abord accélérer régulièrement vers la position 2 ( $p_2$ ), puis avancer à pleine vitesse et enfin ralentir régulièrement pour s'arrêter à la position 2 ( $p_2$ ), voir Figure 2.

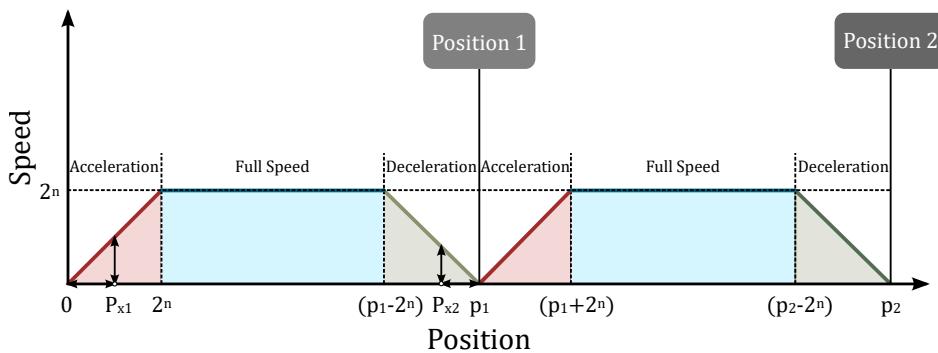


Figure 2 - Diagramme de la vitesse du chariot

Les positions à atteindre sont :

- Position 0 ( $p_0$ , position de réinitialisation) = 3.3 - 3.5 cm suivant les assemblages mécaniques
- Position 1 ( $p_1$ ) = 8 cm
- Position 2 ( $p_2$ ) = 12 cm

### 2.2 Circuit

Pour accomplir la tâche, le circuit suivant est donné pour déplacer le chariot.

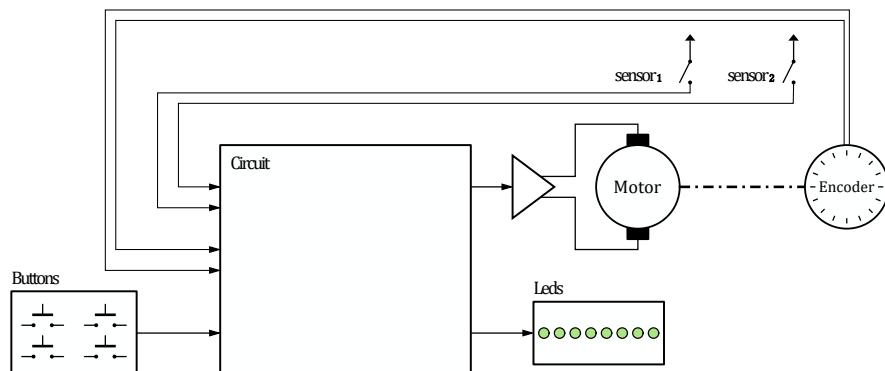


Figure 3 - Circuit du curseur



Le circuit fonctionne comme suit :

- Le moteur à courant continu ([Chapitre 3.2.1](#)) est commandé par les trois signaux  $\text{motor}_{\text{On}}$ ,  $\text{side}_1$ ,  $\text{side}_2$ :
  - Le moteur est activé lorsque le signal  $\text{motor}_{\text{On}}$  est à 1 et doit être remis à 0 lorsque le moteur ne doit pas tourner.
  - Sa vitesse est contrôlée par une modulation **Pulse Width Modulation (PWM)** appliquée soit au signal  $\text{side}_1$ , soit au signal  $\text{side}_2$  selon le sens de rotation désiré.
- Deux relais Reed ([Chapitre 3.4](#))  $\text{sensor}_1$  et  $\text{sensor}_2$  sont placés aux extrémités du rail [1]. Ils détectent la présence du chariot du curseur en indiquant la présence d'un aimant par un 1.
- L'encodeur ([Chapitre 3.3](#)) est utilisé pour suivre, respectivement compter la position du curseur. Ses trois sorties,  $\text{encoder}_A$ ,  $\text{encoder}_B$  et  $\text{encoder}_I$ , permettent de suivre les mouvements de la vis. Les sorties A et B alternent entre 0 et 1 lors des déplacements avec un déphasage de  $90^\circ$  entre elles, ce qui permet de déterminer la direction du mouvement. La sortie I génère une impulsion 1 à chaque tour complet de la vis.
- Trois touches sont utilisées pour contrôler le système:  $\text{restart}$ ,  $\text{go}_1$  et  $\text{go}_2$ . Une touche supplémentaire,  $\text{button}_4$ , peut être utilisée pour des fonctions optionnelles. Lorsque l'une des touches est enfoncée, le signal correspondant passe à 1.
- Les broches  $\text{testOut}$  peuvent être utilisées pour sortir des informations supplémentaires du système, notamment pour les brancher sur les **Light Emitting Diodes (LEDs)** à des fins de debug ou visualisation
- Le signal  $\text{testMode}$  est mis à 1 lors de la simulation. Ce dernier peut être utilisé pour raccourcir les compteurs utilisés afin d'accélérer la génération de signaux lors des tests.

Le toplevel vide du design ([cursor-toplevel-empty.pdf](#)) montre tous les signaux connectés à la platine **Field Programmable Gate Array (FPGA)** Figure 4.



Figure 4 - Circuit Toplevel vide



## 2.3 Scénario (exemple)

Dans le [Figure 5](#), trois scénarios différents sont présentés. Tout d'abord, on appuie sur la touche restart et le chariot se déplace à pleine vitesse vers la position initiale (sensor<sub>1</sub>). Les deux autres scénarios go<sub>2</sub> et go<sub>1</sub> déplacent le chariot vers la position<sub>2</sub> et la position<sub>1</sub> respectivement, un signal PWM variable est appliqué aux signaux side<sub>2</sub> et side<sub>1</sub> pour accélérer et ralentir le chariot.

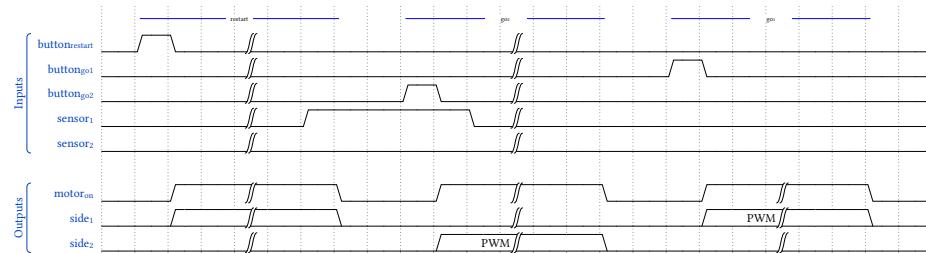


Figure 5 - Scenario curseur



Les scénarios ci-dessus sont des exemples. C'est aux étudiants de les compléter.

## 2.4 Projet HDL-Designer

Un projet HDL-Designer prédéfini peut être téléchargé ou cloné dans [Cyberlearn](#) ou [Github](#). La structure de fichier du projet se présente comme suit:

```
did_cursor
+--Board/           # Project and files for programming the FPGA
|   +-concat/       # Complete VHDL file including PIN-UCF file
|   +-ise/          # Xilinx ISE project
+--Cursor/          # Library for the components of the student solution
+--Cursor_test/     # Library for the simulation testbenches
+--doc/             # Folder with additional documents relevant to the project
|   +-Board/        # All schematics of the hardware boards
|   +-Components/   # All data sheets of hardware components
+--img/              # Pictures
+--Libs/             # External libraries which can be used e.g. gates, io, sequential
+--Prefs/            # HDL-Designer settings
+--Scripts/          # HDL-Designer scripts
+--Simulation/       # Modelsim simulation files
```



Le chemin d'accès au dossier du projet ne doit pas contenir d'espaces.



Le dossier **doc/** contient de nombreuses informations importantes: fiches techniques, évaluation de projet et documents d'aide pour HDL-Designer, pour n'en citer que quelques-unes.



## 3 | Composants

Le système se compose de 3 plaques matérielles différentes, visibles dans la [Figure 1](#).

- Un assemblage de chariot avec une carte électronique **Printed Circuit Board (PCB)** qui commande le moteur et lit les capteurs ([Figure 6](#))
- Une carte de développement **FPGA** ([Figure 14](#) ou [Figure 15](#))
- Une carte de contrôle à 4 boutons et 8 LEDs ([Figure 16](#))

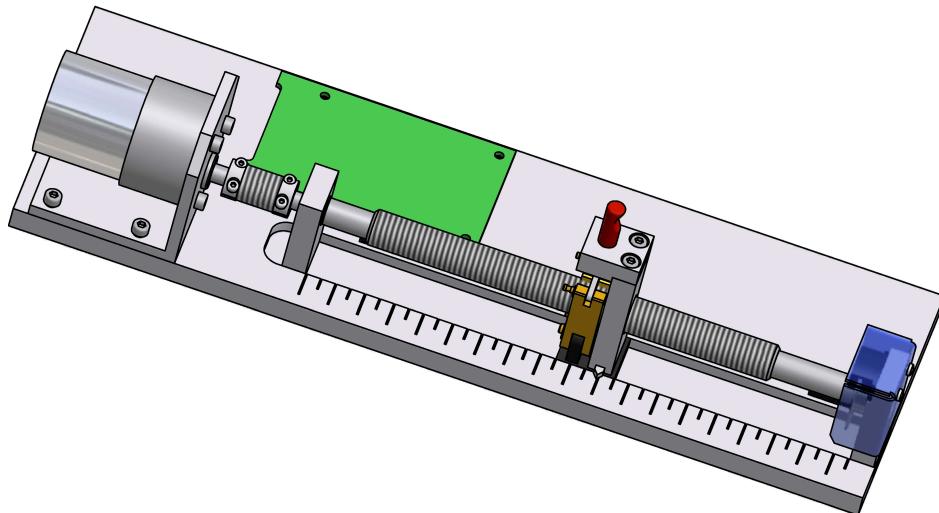


Figure 6 - Assemblage de chariot du curseur

### 3.1 Chariot

La structure du chariot comprend le moteur à courant continu, les deux relais reed [Chapitre 3.4](#) ainsi que le chariot et la vis. Le pas de la vis est de M12x1.75, ce qui signifie qu'une distance de 1.75mm est parcourue par révolution ([Figure 7](#)).

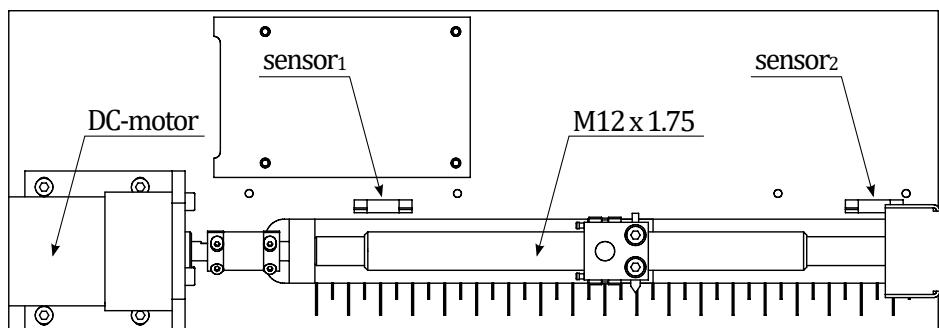


Figure 7 - Assemblage détaillé du chariot du curseur

### 3.2 Circuit de commande de moteur

Le moteur à courant continu du chariot est alimenté en 12V. La carte d'alimentation possède un pont en H commandé par des signaux numériques. Sur la plaque d'alimentation, un régulateur 5V génère la tension alimentant la plaque **FPGA** [2].



### 3.2.1 Moteur à courant continu

Le moteur à courant continu est commandé par un driver de pont en H L6207 [3], voir figure [Figure 8](#). La fréquence de commutation maximale du pont en H est de 100kHz. Ceci doit être pris en compte lors de la création du signal **PWM**.

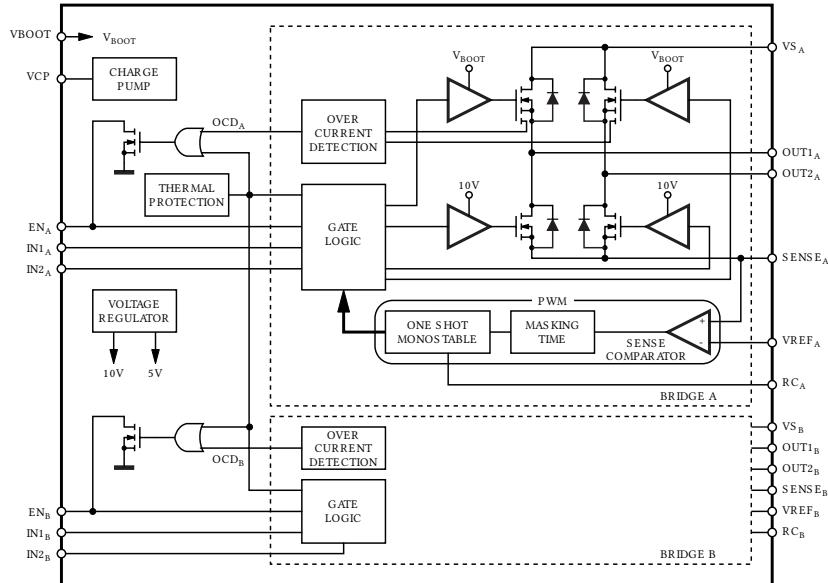


Figure 8 - Schéma bloc du circuit du pont-H L6207N [3]

Pour ajuster la vitesse du moteur à courant continu, un signal **PWM** doit être appliqué aux signaux side<sub>1</sub> ou side<sub>2</sub>, la fréquence maximale étant de 100kHz. Plus la tension est appliquée longtemps au moteur, plus il tourne vite. Alimenter soit side<sub>1</sub> ou side<sub>2</sub> contrôle le sens du moteur.

#### PWM - Stratégie d'implémentation

Une stratégie consiste à créer un compteur en dents de scie ( $0, 1, 2 \dots 2^N - 2, 2^N - 1, 0, 1, \dots$ ) et à comparer sa valeur à un seuil. La règle est de sortir "1" lorsque la valeur du compteur est inférieure au seuil, et "0" sinon. En modifiant, le rapport cyclique du signal **PWM** est modifié. De cette façon, la fréquence du signal **PWM** est définie par la vitesse du compteur (dents de scie), et le rapport cyclique est défini par la valeur du seuil.

Dans la [Figure 9](#), le moteur tourne plus lentement avec le **signal vert** qu'avec le **signal bleu** et qu'avec le **signal rouge**.

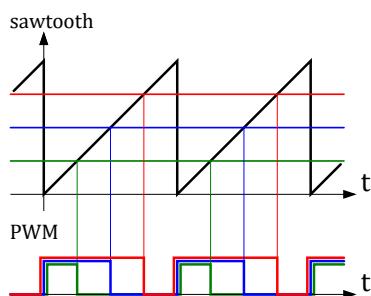


Figure 9 - Signaux PWM



Plus haut est le nombre de pas (bits) de la **PWM**, plus le contrôle de la vitesse est précis. Cependant, trop de pas peuvent entraîner une différence de vitesse non notable en raison des systèmes de contrôle et des caractéristiques du moteur. Une bonne base est d'utiliser 8 bits pour la résolution du **PWM**, donnant 256 niveaux de vitesse de 0 à 255.

### Rampes - Stratégie d'implémentation

Pour créer des rampes d'accélération et de décélération, une méthode courante consiste à utiliser un profil de vitesse trapézoïdal. Ce profil se compose de trois phases : accélération, vitesse constante et décélération, comme le montre la [Figure 10](#).

Plusieurs stratégies existent pour créer de telles rampes. Une - plus simple à mettre en œuvre - est décrite en [Chapitre 4](#).

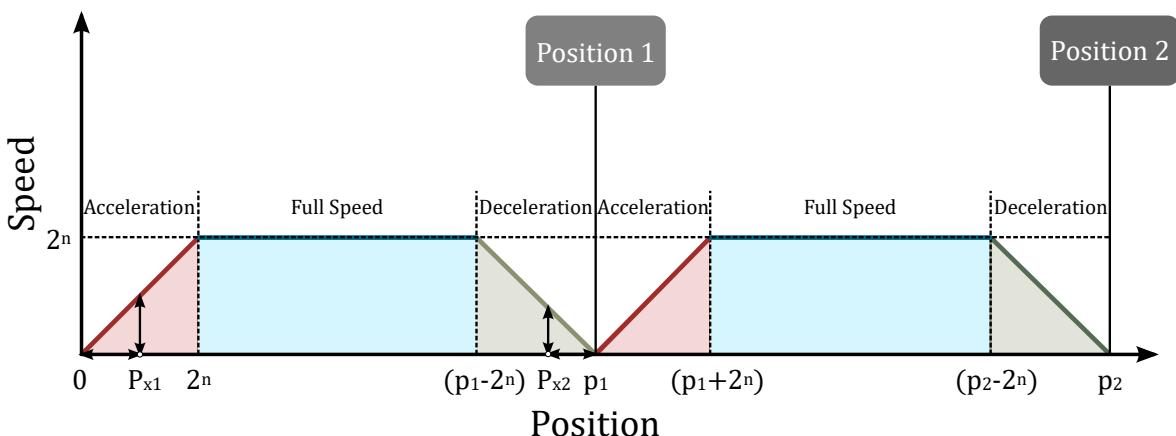


Figure 10 - Diagramme de la vitesse du chariot

### 3.3 Codeur (encodeur)

L'angle de la vis peut être mesuré à l'aide d'un **codeur incrémental**. Le modèle utilisé dans l'assemblage est un AEDB-9140-A12 [4] (Figure 11) avec 2 canaux de feedback avec 500 **Counts per Revolution (CPR)** (impulsions par tour) par canal, représenté dans la [Figure 12](#).



Figure 11 - Encodeur AEDB-9140-A12 [4]

### Encodeurs - Stratégie d'implémentation

Il est possible de l'utiliser de deux manières:

- En utilisant un seul canal, ce qui donne une résolution de  $500 \frac{\text{pulses}}{\text{tour}}$ , c.à.d  $1'000 \frac{\text{fronts}}{\text{tour}}$  si l'on compte les fronts montants et descendants.
- En utilisant les deux canaux comme interface **QEI (Quadrature Encoder Interface)**, en comptant les fronts montants et descendants des deux canaux, augmentant la résolution à  $2000 \frac{\text{fronts}}{\text{tour}}$ . Cet interfaçage permet aussi de détecter le sens de rotation du moteur et est une méthode courante pour contrôler les moteurs à courant continu avec des encodeurs de feedback dans l'industrie.

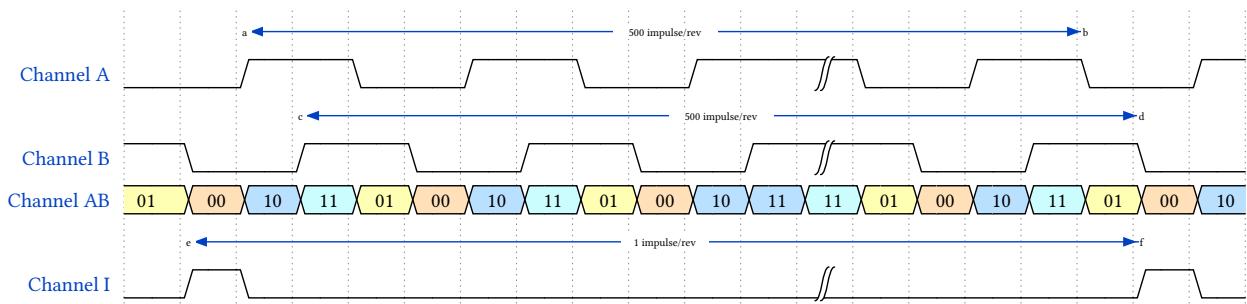


Figure 12 - Signaux de codeurs incrémentaux

### 3.4 Reed-Relais

Le relais Reed est un interrupteur qui peut être commuté à l'aide d'électro-aimants [1]. Lorsqu'un aimant se trouve à proximité du capteur, le contact se ferme (Figure 13). 2 relais Reed sont utilisés ( $\text{sensor}_1$  et  $\text{sensor}_2$ ) pour identifier les limites gauche et droite du chariot.

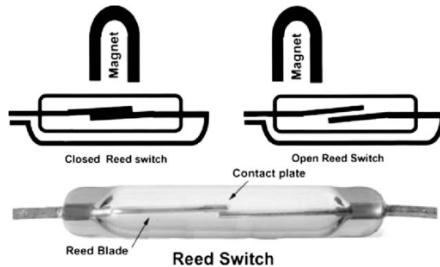


Figure 13 - Reed relais [5]

Les signaux  $\text{sensor}_1$  et  $\text{sensor}_2$  sont à "1" lorsque le contact est fermé (aimant à proximité), sinon "0".

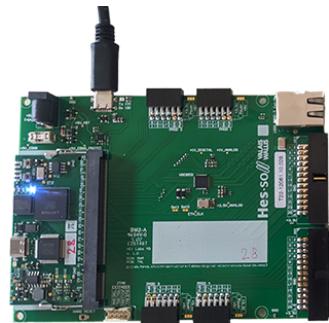
### 3.5 Carte FPGA

La carte principale est la carte de développement de laboratoire FPGA-EBS 2 de l'école [6]. Elle héberge une puce Xilinx Spartan xc3s500e FPGA [7], [8] et dispose de nombreuses interfaces différentes (Universal Asynchronous Receiver Transmitter (UART), Universal Serial Bus (USB), Ethernet, etc.). L'oscillateur utilisé produit un signal d'horloge (**clock**) avec une fréquence de  $f_{\text{clk}} = 66\text{MHz}$  [9].



Figure 14 - Carte électronique FPGA [6]

Sur la carte EBS3, l'oscillateur utilisé produit un signal d'horloge (**clock**) avec une fréquence de  $f_{\text{clk}} = 100\text{MHz}$ , réduit par PLL à  $f_{\text{clk}} = 60\text{MHz}$ .

Figure 15 - Carte électronique **FPGA** [10]

Les simulateurs sont réglés par défaut pour les boards EBS2. Pour les modifier, ouvrez un bloc de testbench **xxx\_tb** et double-cliquez sur les déclarations **Pre-User** (en haut à gauche de la page) pour modifier la variable **clockFrequency** selon la valeur de clock souhaitée.

### 3.6 Boutons et LEDs

La platine avec les boutons et les **LEDs** [11] est connectée à la platine **FPGA**. Elle comprend 4 boutons et 8 **LEDs** qui peuvent être utilisés dans le design. Cette platine comprend un affichage **LCD** [12], [13] qui peut être utilisé dans le projet.

Figure 16 - Carte électronique boutons-**LEDs-LCD** [11]



## 4 | Implémentation des rampes moteur

Ce chapitre est une proposition d'implémentation de rampes d'accélération. D'autres méthodes existent et peuvent très bien être utilisées.

Pour créer des rampes d'accélération et de décélération, une méthode courante est d'utiliser un profil de vitesse trapézoïdal. Ce profil se compose de trois phases : accélération, vitesse constante et décélération comme illustré sous [Figure 17](#).

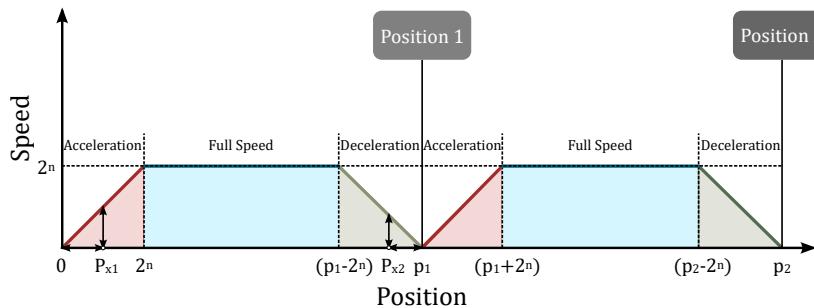


Figure 17 - Diagramme de la vitesse du chariot

Les rampes d'accélération et de décélération utilisent le même profil.

### 4.1 Concept de base des rampes

Typiquement, les profils de vitesse sont basés sur le temps. Cette méthode est souvent rencontrée dans les applications industrielles.

Cependant, de tels systèmes nécessitent une boucle de rétroaction complète pour mesurer précisément la vitesse réelle du moteur en fonction du temps ainsi qu'une boucle de contrôle pour permettre un positionnement précis. Le moteur pourrait ralentir en raison de variations de charge, de frictions, d'usure, de vieillissement... De plus, il serait difficile de définir quand commencer la phase de décélération sans caractériser complètement le système.

Pour contrer cela, cette proposition de rampes est basée sur la position plutôt.

En raison de la nature de ce concept, le système se régule automatiquement (mais n'est pas capable de compenser les variations de charge).



## 4.2 Profil de rampe naïf

Pour obtenir un profil complet, les phases d'accélération + décélération doivent se compléter dans la plus petite distance disponible entre deux positions cibles. Ici, cette distance est de 4cm entre la position p1 (80mm) et p2 (120mm).

Ainsi, les rampes devraient utiliser environ 1cm (pour être suffisamment visibles) à moins de 2cm (pour laisser le temps à la vitesse maximale de se mettre en place) pour se compléter.

La rampe naïve est illustrée sous [Figure 18](#) :

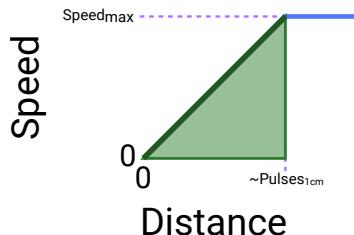


Figure 18 - Rampe d'accélération naïve

Pour créer cette rampe, une équation linéaire représentant la vitesse en fonction de la position est utilisée :

$$v_x = m * x + b$$

avec

$$m = \frac{\Delta y}{\Delta x} = \frac{\text{speed}_{\max} - 0}{\sim \text{pulses}_{\text{for\_1_cm}} - 0}$$

$$b = 0$$

Pour simplifier tous les calculs et ne pas nécessiter d'implémenter des multiplicateurs et diviseurs hardware, seules des puissances de deux sont utilisées comme arguments de la fonction :

- speed<sub>max</sub> correspond à la résolution PWM, c'est-à-dire 2<sup>pwmBitNb</sup> avec pwmBitNb = 8 par défaut
- ~ pulses<sub>for\_1\_cm</sub> dépend de la façon dont les impulsions sont comptées :
  - En utilisant un seul encodeur, c'est-à-dire 500  $\frac{\text{pulses}}{\text{revolution}}$  :
 
$$\sim \text{pulses}_{\text{for\_1_cm}} = \frac{1 \text{ cm}}{3.5 \frac{\text{mm}}{\text{pulse}}} = 2'857 \text{ pulses} \approx 2^{12} = 4'096 \text{ pulses} \rightarrow 4'096 \text{ pulses} * 3.5 \mu\text{m} = 1.425 \text{ cm}$$
  - En utilisant un seul encodeur tout en comptant les deux fronts, c'est-à-dire 1'000  $\frac{\text{pulses}}{\text{revolution}}$  :
 
$$\sim \text{pulses}_{\text{for\_1_cm}} = \frac{1 \text{ cm}}{1.75 \frac{\text{mm}}{\text{pulse}}} = 5'714 \text{ pulses} \approx 2^{13} = 8'192 \rightarrow 8'192 \text{ pulses} * 1.75 \mu\text{m} = 1.425 \text{ cm}$$
  - En utilisant les deux signaux de l'encodeur, c'est-à-dire 2'000  $\frac{\text{pulses}}{\text{revolution}}$  (mode QEI) :
 
$$\sim \text{pulses}_{\text{for\_1_cm}} = \frac{1 \text{ cm}}{875 \frac{\text{mm}}{\text{pulse}}} = 11'428 \text{ pulses} \approx 2^{14} = 16'384 \rightarrow 16'384 \text{ pulses} * 875 \text{ nm} = 1.425 \text{ cm}$$

**Le reste de cette proposition est basé sur la méthode QEI, la plus précise des trois.**

Avec ces variables définies, les paramètres de la pente sont :

$$m = \frac{\text{speed}_{\max} - 0}{\sim \text{pulses}_{\text{for\_1_cm}} - 0} = \frac{2^8}{2^{14}} = \frac{1}{2^6} = \frac{1}{64}$$

$$b = 0$$



Cela signifie que la pente prendrait la forme :  $v_x = \frac{x}{2^6}$ .

La division par une puissance de deux peut être simplifiée par un décalage à droite :  $v_x = x >> 6$



Puisque le multiplicateur  $m$  est inférieur à 1 et que le circuit ne gère que des entiers, la courbe de vitesse n'augmentera que tous les  $2^N$  impulsions. Cela signifie que la vitesse augmentera par paliers, pas de manière fluide. Cela est acceptable pour cette application. Voir [Figure 19](#) pour illustration.

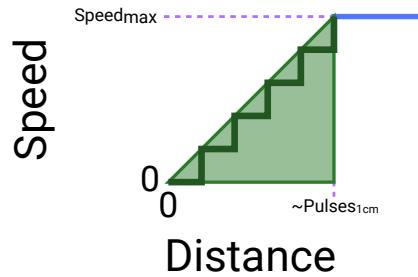


Figure 19 - Rampe d'accélération naïve - effet escaliers



### 4.3 Problèmes et améliorations de la solution naïve

Un problème subsiste dans cette approche : au début du mouvement, la vitesse est de 0. Le moteur n'est pas alimenté et ne commencera pas à bouger. Si l'équation dépendait du temps, la courbe progresserait lentement et démarrerait le moteur. Ici, la position ne change pas si le moteur ne bouge pas, donc la vitesse reste à 0 tant que X est à 0.

Pour éviter ces problèmes, un offset minimal est ajouté à l'équation comme montré en [Figure 20](#) :

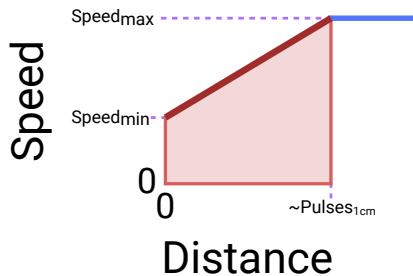


Figure 20 - Rampe d'accélération modifiée

L'équation devient alors :

$$v_x = m * x + b$$

avec

$$m = \frac{\Delta y}{\Delta x} = \frac{\text{speed}_{\max} - \text{speed}_{\min}}{\sim \text{pulses}_{\text{for}_1\text{cm}} - 0}$$

$$b = \text{speed}_{\min}$$

$\text{speed}_{\min}$  dépend de « combien de puissance » est nécessaire pour que le moteur bouge, impacté par : les moteurs utilisés, leur usure, les contraintes mécaniques, l'implémentation PWM, l'électronique de pilotage... nécessitant de tester le système.

Pour l'exemple,  $\text{speed}_{\min}$  de  $2^6 = 64$  est utilisé.

Avec ces variables définies, les paramètres de la pente sont :

$$m = \frac{\text{speed}_{\max} - \text{speed}_{\min}}{\sim \text{pulses}_{\text{for}_1\text{cm}} - 0} = \frac{2^8 - 2^6}{2^{14}} = \frac{192}{16384}$$

Une telle valeur de  $m$  n'est pas facilement implémentable en hardware sans multiplicateurs/diviseurs. Il est possible d'essayer d'approcher ce facteur par des puissances de deux. Pour rester simple, la même pente que calculée auparavant est utilisée. Un offset est simplement ajouté pour compenser les faibles valeurs de vitesse aux basses positions X.



Faire de telles approximations implique de toujours re-calculer les hypothèses de base une fois tous les paramètres fixés. Dans ce cas, puisque la pente est approximée, il faut s'assurer que la rampe d'accélération ne devienne pas « trop rapide » (voir [Équation 1](#)).



#### 4.4 Rampe finale

L'équation finale est maintenant :

$$v_x = m * x + b = \frac{x}{2^6} + 2^6 = (x >> 6) + 64$$

Cette équation ne doit être appliquée que pour la phase d'accélération puis limitée à la valeur maximale de la PWM une fois atteinte.

##### Exemple

En supposant que le chariot est à la position  $X = 0$  et qu'il doit rejoindre le point P1:

- la phase d'accélération se fait entre  $X_{\text{chariot}} - X_0$  jusqu'à ce que la PWM atteigne 255
- suit la phase de pleine vitesse jusqu'à ce que  $X_{P1} - X_{\text{chariot}}$  donne une valeur de PWM inférieure à 255
- la phase de décélération s'entame jusqu'à ce que la position cible soit atteinte, moment où le moteur doit être arrêté



## 4.5 Vérification des hypothèses

Puisque la pente précédente a été conservée, le moteur sera à pleine vitesse avant d'atteindre la distance prédéfinie comme montré en [Figure 21](#) (courbe bleue):

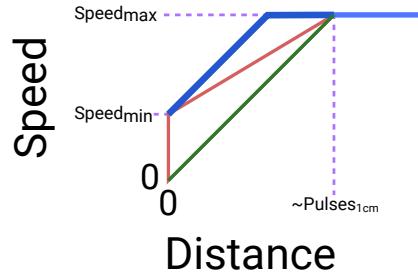


Figure 21 - Rampe d'accélération finale

Cette valeur peut être calculée comme suit :

$$x = \frac{v_x - b}{m} = \frac{v_x - b}{\frac{\text{speed}_{\text{max}}}{\sim \text{pulses}_{\text{for 1 cm}}}} = \frac{(v_x - b) * (\sim \text{pulses}_{\text{for 1 cm}})}{\text{speed}_{\text{max}}} = \frac{(2^8 - 2^6) * 2^{14}}{2^8} = \frac{3'145'728}{256} = 12'288 \text{ pulses}$$

$$\rightarrow 12'288 * 875 \text{ nm} = 1.075 \text{ cm}$$

Équation 1 - Distance finale d'accélération

La distance minimale entre deux points étant de 4cm (p1 - p2), les courbes d'accélération + décélération prennent 2.15cm au total, laissant 1.85cm à pleine vitesse. Les courbes n'ont jamais besoin d'être « coupées » pour atteindre les positions cibles.

Notez que si la speed<sub>min</sub> devait être réglée plus haut en raison des caractéristiques du moteur, la pente *m* devrait être réduite en conséquence pour s'adapter toujours aux exigences de distance de rampe de 1cm à 2cm.



Du point de vue temporel, comme la courbe dépend de la position, elle n'aura pas l'air linéaire. Étant donné que les faibles positions X ont de faibles vitesses moteur et prennent donc plus de temps pour se déplacer par rapport aux valeurs X plus élevées, la courbe apparaîtra plutôt exponentielle comme dans [Figure 22](#).



Figure 22 - Rampe moteur en fonction du temps



## 5 | Evaluation

Dans le dossier **doc/**, le fichier **evaluation-bewertung-cursor.pdf** montre le schéma d'évaluation détaillé, voir [Table 1](#).

La note finale contient le rapport, le code ainsi qu'une présentation de votre système.

Aspects évalués	Points
<b>Rapport</b>	<b>55</b>
Introduction	3
Spécification	5
Projet	20
Vérification et validation	10
Intégration	9
Conclusion	3
Aspects formels du rapport	5
<b>Fonctionnalité du circuit</b>	<b>30</b>
<b>Qualité de la solution</b>	<b>10</b>
<b>Présentation</b>	<b>10</b>
<b>Total</b>	<b>105</b>

Table 1 - Grille d'évaluation



La grille d'évaluation donne des indications sur la structure du rapport. Pour un bon rapport, consultez le document « Comment rédiger un rapport de projet » [\[14\]](#).



## 6 | Premières étapes

Pour commencer le projet, procédez de la manière suivante :

- Lisez attentivement les spécifications et les informations ci-dessus.
- Examinez le matériel et testez le programme préprogrammé.
- Parcourez les documents dans le dossier **doc/** de votre projet.
- Développez un schéma fonctionnel détaillé. Vous devez pouvoir expliquer les signaux et leurs fonctions.
- Implémentez et simulez les différents blocs.
- Testez la solution sur le circuit imprimé et trouvez les éventuelles erreurs .

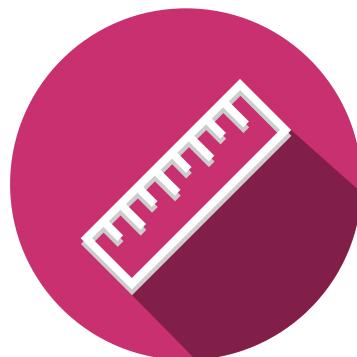
### 6.1 Tips

Ci-joint quelques conseils supplémentaires pour éviter les problèmes et les pertes de temps:

- Divisez le problème en différents blocs, utilisez pour cela le document Toplevel vide (**cursor-toplevel-empty.pdf**). Il est recommandé d'avoir un mélange équilibré entre le nombre de composants et la taille/complexité des composants.
- Analysez les différents signaux d'entrée et de sortie. Il est conseillé d'utiliser en partie les fiches techniques.
- Respectez le chapitre DiD « Méthodologie de conception de circuits numériques (MET) » lors de la création du système [15].
- Il est recommandé de réaliser le système de façon incrémentale, par exemple:
  - D'abord un système qui réagit aux boutons et qui amène le chariot à la position correspondante (toujours à la vitesse maximale).
  - Intégrer les phases d'accélération dans le système existant.



N'oubliez pas de vous amuser.





# Glossaire

*CPR* – Counts per Revolution [9](#)

*FPGA* – Field Programmable Gate Array [5, 7, 10, 11](#)

*LCD* – Liquid Crystal Display [3, 11](#)

*LED* – Light Emitting Diode [5, 7, 11](#)

*PCB* – Printed Circuit Board [7](#)

*PWM* – Pulse Width Modulation [5, 6, 8, 9](#)

*UART* – Universal Asynchronous Receiver Transmitter [10](#)

*USB* – Universal Serial Bus [10](#)



# Bibliographie

- [1] « Reed Relay ». 5 décembre 2020. Consulté le: 24 novembre 2021. [En ligne]. Disponible sur: [https://en.wikipedia.org/w/index.php?title=Reed\\_relay&oldid=992433034](https://en.wikipedia.org/w/index.php?title=Reed_relay&oldid=992433034)
- [2] Olivier Walpen, « Schematic: Cursor Chariot Power Circuit ». 2009.
- [3] STMicroelectronics, « Datasheet: DMOS Dual Full Bridge Driver with PWM Current Controller ». 2003.
- [4] Agilent Technologies, « Datasheet Agilent AEDB-9140 Series Three Channel Optical Incremental Encoder Modules with Codewheel, 100 CPR to 500 CPR ». 2005.
- [5] « Magnetic-Reed-Switch-Above-Closed-and-open-reed-switch-in-response-to-magnet-placement.Png (850×345) ». Consulté le: 24 novembre 2021. [En ligne]. Disponible sur: <https://www.researchgate.net/profile/Sidakpal-Panaich-2/publication/51169357/figure/fig1/AS:394204346896388@1470997048549/Magnetic-reed-switch-Above-Closed-and-open-reed-switch-in-response-to-magnet-placement.png>
- [6] Silvan Zahno, « Schematic: FPGA-EBS v2.2 ». 2014.
- [7] Xilinx, « Spartan-3 FPGA Family ». Consulté le: 20 novembre 2021. [En ligne]. Disponible sur: <https://www.xilinx.com/products/silicon-devices/fpga/spartan-3.html>
- [8] Xilinx, « Datasheet Spartan-3E FPGA Family ». 2008.
- [9] CTS, « Datasheet CTS Model CB3 & CB3LV HCMOS/TTL Clock Oscillator ». 2006.
- [10] A. Amand et S. Zahno, « FPGA-EBS3 Electornic Technical Documentation ». 2022.
- [11] Silvan Zahno, « Schematic: Parallelport HEB LCD V2 ». 2014.
- [12] Sitronix, « Datasheet Sitronix ST7565R 65x1232 Dot Matrix LCD Controller/Driver ». 2006.
- [13] Electronic Assembly, « Datasheet: DOGM Graphics Series 132x32 Dots ». 2005.
- [14] Christophe Bianchi, François Corthay, et Silvan Zahno, « Comment Rédiger Un Rapport de Projet? ». 2021.
- [15] François Corthay, Silvan Zahno, et Christophe Bianchi, « Méthodologie de Conception de Circuits Numériques ». 2021.