



---

# Rapport Mini Projet

CENTRALE DCC SUR FPGA

---

**Jorge MENDIETA  
Youba FERHOUNE**

**Master Informatique**  
Parcours Systèmes Électroniques et Systèmes Informatiques  
Sorbonne Université  
Faculté de Sciences et Ingénierie  
2023/2024

## Table des matières

<b>1</b>	<b>Introduction</b>	<b>2</b>
<b>2</b>	<b>Protocole DCC</b>	<b>2</b>
2.1	Représentation des bits . . . . .	2
2.2	Génération des trames . . . . .	3
<b>3</b>	<b>Architecture de la centrale DCC</b>	<b>4</b>
3.1	Diviseur horloge . . . . .	4
3.2	Tempo . . . . .	5
3.3	Bits DCC 0 et 1 . . . . .	6
3.4	Registre DCC . . . . .	7
3.5	Machine à États . . . . .	9
3.6	Fichier Top . . . . .	11
3.7	Simulation du fichier Top . . . . .	11
<b>4</b>	<b>Conception de l'IP DCC</b>	<b>13</b>
<b>5</b>	<b>Bilan d'avancement</b>	<b>13</b>

# 1 Introduction

Ce projet a pour objectif de concevoir la partie de génération du signal de contrôle pour le train. Nous devons générer un signal spécifique en utilisant les informations fournies par les interrupteurs sur la carte FPGA. Avec ces informations, nous produirons un signal conforme au protocole DCC, afin que le système de train puisse exécuter des commandes.

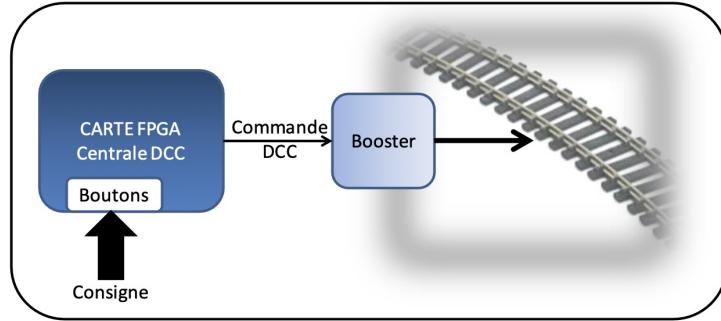


FIGURE 1 – Synoptique du système de commande

Dans un premier temps nous allons décrire et implémenter les modules restants de la Centrale DCC en VHDL, faire une simulation comportementale à l'aide d'un testbench, assembler tous ces modules dans un module `Top_DCC`, et valider le fonctionnement du système sur la plate-forme trains.

Dans un second temps, il faut intégrer la Centrale DCC au sein d'un système **Microblaze**. Pour cela, la centrale devra être packagée sous forme d'un IP pour interagir avec le Microblaze via le bus AXI.

# 2 Protocole DCC

Le protocole DCC (*Digital Command Control*) est un standard utilisé dans le modélisme ferroviaire pour commander individuellement des locomotives ou des accessoires en modulant la tension d'alimentation de la voie.

## 2.1 Représentation des bits

Un bit est représenté par une impulsion à 0 du signal de sortie, suivie d'une impulsion à 1. La durée des impulsions permet de différencier un bit à 0 d'un bit à 1. L'ordre des impulsions (à 0 puis à 1) doit être obligatoirement respecté.

Bit	Représentation
0	Impulsion à 0 de 100 µs puis Impulsion à 1 de 100 µs
1	Impulsion à 0 de 58 µs puis Impulsion à 1 de 58 µs

TABLE 1 – Durée des bits DCC

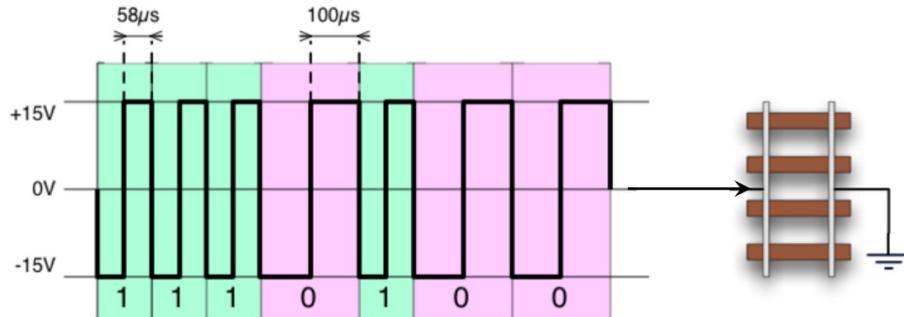


FIGURE 2 – Représentation des bits

## 2.2 Génération des trames

Chaque trame est composée de 4 champs dont le détail est décrit dans la figure 3.

Le champ contrôle est fondamentalement un *checksum* afin de détecter d'éventuelles erreurs de transmission de la trame.

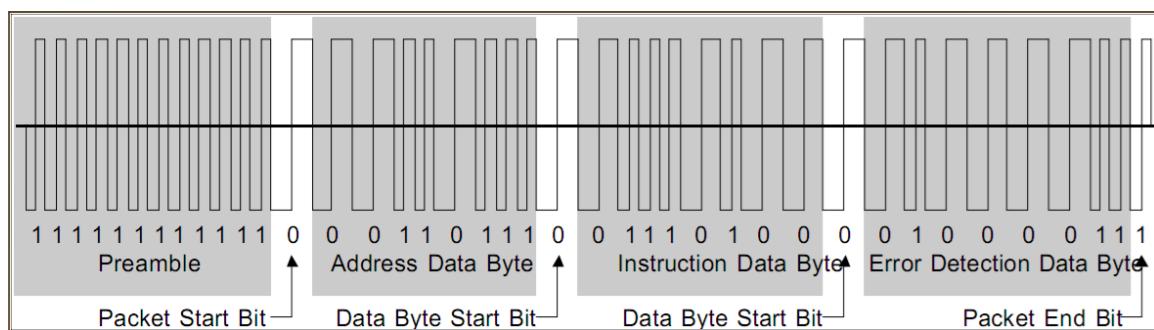


FIGURE 3 – Structure des trames DCC

### 3 Architecture de la centrale DCC

L'architecture de la centrale DCC est présentée dans la figure ci-dessous :

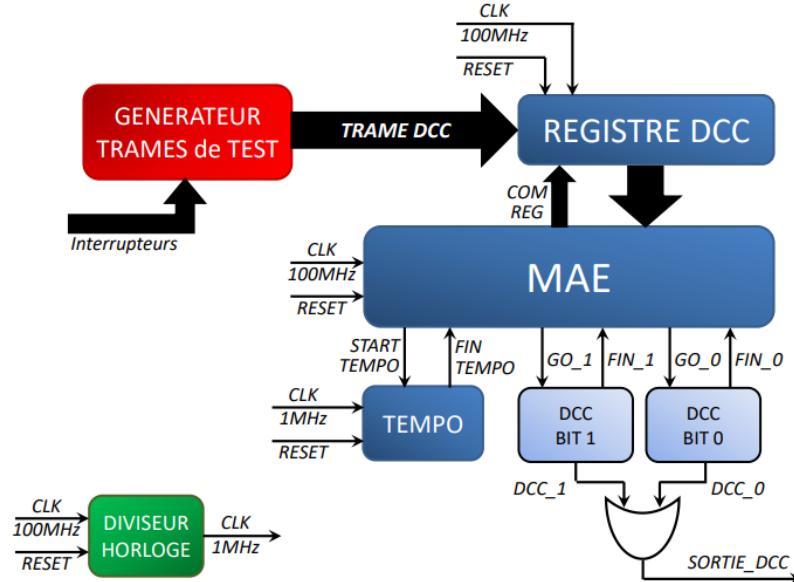


FIGURE 4 – Architecture de la Centrale DCC

Cette architecture compose 6 parties principaux :

1. Diviseur horloge
2. Tempo
3. Bits DCC 0 et 1
4. Registre DCC
5. Machine à États
6. Générateur Trames de Test

#### 3.1 Diviseur horloge

Ce module génère un signal d'horloge de fréquence **1 MHz** à partir de l'horloge de **100 MHz** de la carte FPGA. L'horloge **CLK\_1MHz** est utilisée pour générer les délais requis par le protocole DCC.

Le fonctionnement de ce module est assez simple : il contient un compteur interne qui, lorsqu'il atteint la valeur 49, change l'état du port de sortie. Ainsi, chaque deux cycles de commutation correspondent à une période d'horloge.

Un reset asynchrone permet de réinitialiser le module, remettant à zéro à la fois le signal de sortie et le compteur interne.



FIGURE 5 – Diviseur d’horloge

```

1 entity CLK_DIV is
2 port
3 (
4     Reset      : in std_logic;    -- Reset Asynchrone
5     Clk_In     : in std_logic;    -- Horloge 100 MHz de la carte Nexys
6     Clk_Out    : out std_logic); -- Horloge 1 MHz de sortie
7 end CLK_DIV;
8 architecture Behavioral of CLK_DIV is
9 signal Div      : integer range 0 to 49;
10    -- Compteur de cycles d'horloge
11 signal Clk_Temp : std_logic;           -- Signal temporaire

```

Programme 1 – Entité du diviseur d’horloge

## 3.2 Tempo

Ce module, cadencé par l’horloge CLK\_1MHz, est conçu pour mesurer l’intervalle de temps requis entre deux trames DCC, soit 6 ms. La commande *Start\_Tempo*, générée par la machine à états (MAE), déclenche le compteur interne pour débuter la mesure de ce délai.

```

1 entity COMPTEUR_TEMPO is
2 port
3 (
4     Clk          : in std_logic;    -- Horloge 100 MHz
5     Reset        : in std_logic;    -- Reset Asynchrone
6     Clk1M        : in std_logic;    -- Horloge 1 MHz
7     Start_Tempo  : in std_logic;    -- Démarrage de la Temporisation
8     Fin_Tempo    : out std_logic;   -- Drapeau de Fin de la Temporisation
9 );
10 end COMPTEUR_TEMPO;

```

Programme 2 – Entité du module Tempo

Lorsque le signal *Start\_Tempo* est activé, le compteur commence à incrémenter en syn-

chronisation avec l'horloge **CLK\_1MHz**. Une fois que le compteur atteint la valeur correspondant à 6 ms (soit 6000 cycles d'horloge à 1 MHz), il génère une impulsion d'un cycle d'horloge sur le signal *Fin\_Tempo* pour indiquer que le délai de temporisation est écoulé.

Après la génération de l'impulsion *Fin\_Tempo*, le compteur est automatiquement remis à zéro, prêt à mesurer un nouvel intervalle de temps lorsque la commande *Start\_Tempo* sera de nouveau activée.

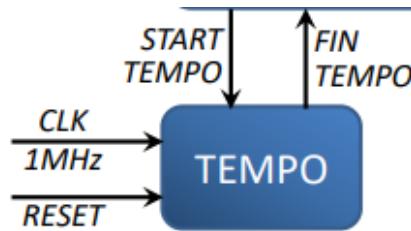


FIGURE 6 – Le compteur Tempo

### 3.3 Bits DCC 0 et 1

Ces deux blocs permettent de générer respectivement un bit à 1 ou à 0 au format DCC. Ils sont tous les deux cadencés par les horloges **CLK\_100MHz** et **CLK\_1MHz**, et sont connectés au Reset du système (ce n'est pas représenté sur le schéma pour plus de clarté). Chaque module est piloté par la machine à états (MAE) globale du système.

```

1  entity DCC_Bitx is
2    port
3    (
4      Clk      : in std_logic;    -- Horloge 100 MHz
5      Reset   : in std_logic;    -- Reset Asynchrone
6      Clk1M   : in std_logic;    -- Horloge 1 MHz
7      GO_x    : in std_logic;    -- Démarrage de la Temporisation
8      FIN_x   : out std_logic;   -- Drapeau de Fin de la Temporisation
9      DCC_x   : out std_logic;
10 );
11 end DCC_Bitx;
```

Programme 3 – Entité des modules des bits DCC

La commande "Go" active la génération du bit, et le signal "Fin" indique que la transmission du bit est terminée. Chaque bloc possède une machine à états et un compteur. La machine à états, cadencée par l'horloge **CLK\_100MHz**, réalise l'interaction avec la MAE globale du système et positionne le signal de sortie au niveau logique approprié. Le compteur, cadencé par l'horloge **CLK\_1MHz**, compte les temps pendant lesquels le signal de sortie doit être à 1 ou 0, conformément au protocole DCC. Lorsque le module n'est pas activé, le

signal de sortie doit être au niveau bas. En sortie des modules DCC\_Bit\_0 et DCC\_Bit\_1, une porte OU joint les signaux de sortie de ces deux modules.

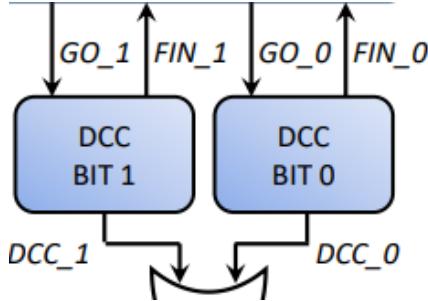


FIGURE 7 – DCC\_Bit\_1 ET DCC\_Bit\_0

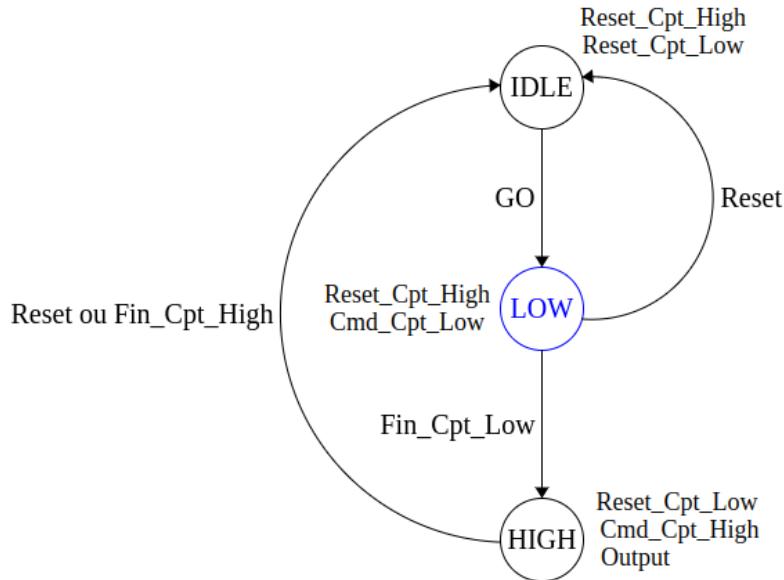


FIGURE 8 – MAE DE DCC\_BIT 0 ET 1

### 3.4 Registre DCC

Ce module est un registre à décalage conçu pour charger la trame DCC préparée par le Générateur de Trames de Test, puis effectuer des décalages successifs à mesure que les bits de la trame sont transmis. La taille du registre est adaptée pour contenir la trame la plus longue du protocole DCC, soit 51 bits. La commande du registre, incluant les opérations de chargement et de décalage, est gérée par la Machine à Etats (MAE)

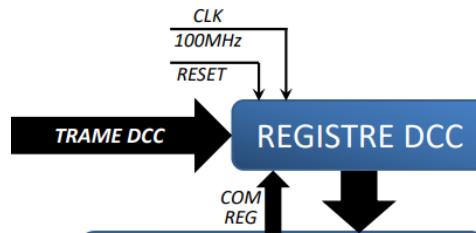


FIGURE 9 – Registre DCC

```

1  entity REGISTRE_DCC is
2      port
3      (
4          Clk      : in std_logic;           -- Horloge 100 MHz
5          Reset    : in std_logic;           -- Reset Asynchrone
6          Trame_DCC : in std_logic_vector(50 downto 0); -- Trame DCC
7          Com_REG   : in std_logic_vector(1 downto 0);
8              -- Command for register
9          Bit_out   : out std_logic           -- Bit send to MAE
10     );
11 end REGISTRE_DCC;
12
13 architecture Behavioral of REGISTRE_DCC is
14     signal nb_shifted : integer range 0 to 50;
15     signal reg_data   : std_logic_vector(50 downto 0);
16     signal bit_out_s  : std_logic;

```

Programme 4 – Entité du registre DCC

Le signal de commande Com\_REG peut prendre trois valeurs différentes, chacune ayant un rôle spécifique :

**Com\_REG = "00"** : Le module ne fait rien et le signal de sortie reste inchangé.  
**Com\_REG = "01"** : Le module effectue un décalage à droite, envoyant ainsi le bit le plus à gauche. **Com\_REG = "10"** : Le module recharge une nouvelle trame, recommençant le processus de transmission.

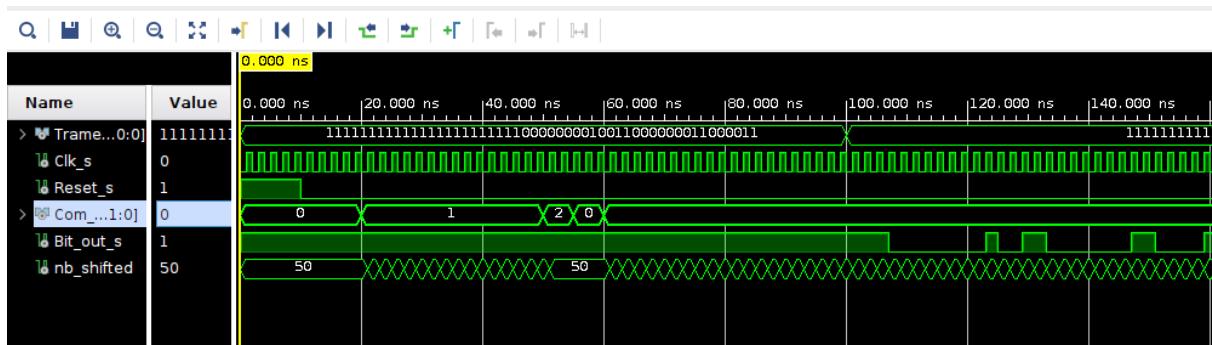


FIGURE 10 – Simulation du Registre DCC

### 3.5 Machine à États

Ce module commande les autres blocs du système. Il permet de générer en continu la trame de commande DCC préparée par l'utilisateur, laquelle est ensuite envoyée sur le signal Sortie\_DC. Le fonctionnement de notre machine à états est le suivant :

```

1  entity MAE is
2  port
3  (
4      Clk          : in std_logic;    -- Horloge 100 MHz
5      Clk1M        : in std_logic;    -- Horloge 1 MHz
6      Reset         : in std_logic;    -- Reset Asynchrone
7      Reset_Tempo   : out std_logic;
8      Start_Tempo   : out std_logic;   -- Demarrage de la Temporisation
9      Fin_Tempo     : in std_logic;    -- Drapeau de Fin de la Temporisation
10     Reset_DDC0    : out std_logic;
11     GO_0          : out std_logic;   -- Demarrage de la Temporisation
12     FIN_0         : in std_logic;    -- Drapeau de Fin de la Temporisation
13     Reset_DDC1    : out std_logic;
14     GO_1          : out std_logic;
15     FIN_1         : in std_logic;
16     Com_REG       : out std_logic_vector(1 downto 0);
17     DCC_in        : in std_logic    -- Bit send to MAE
18 );
19 end MAE;
```

Programme 5 – Entité de la MAE

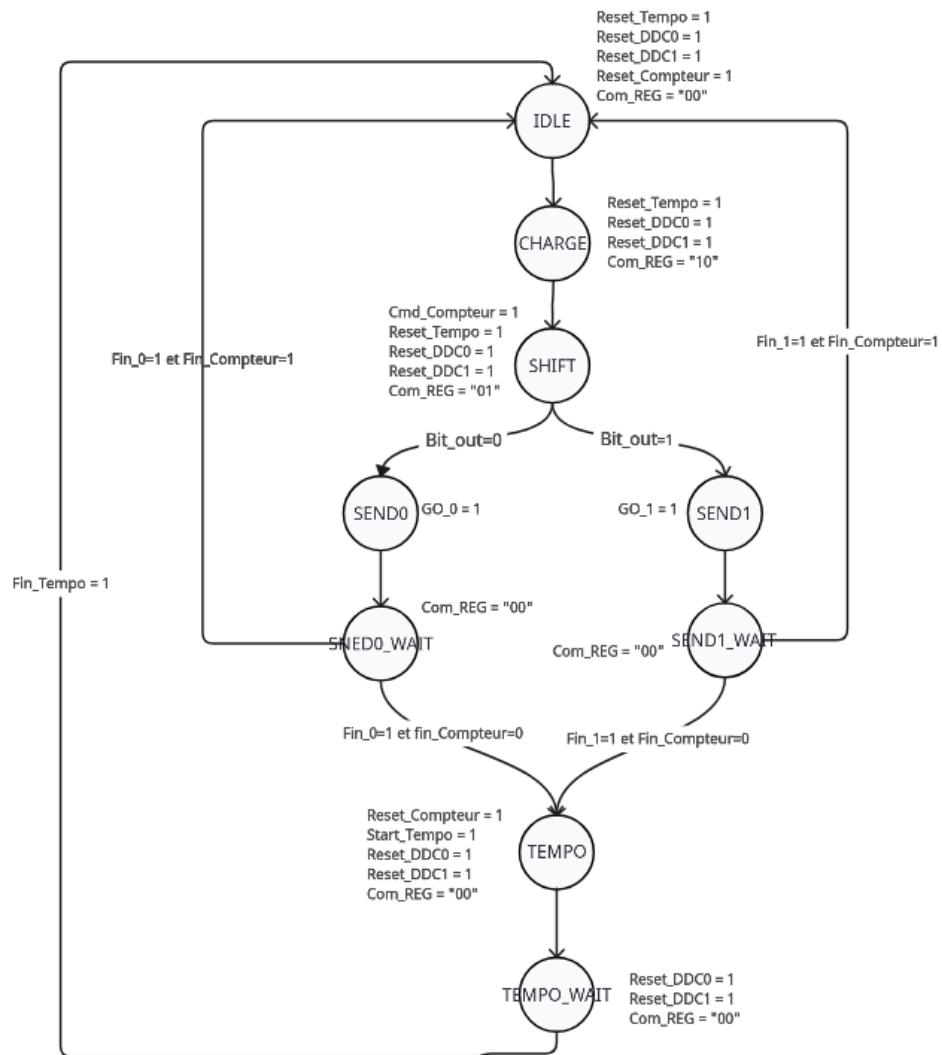


FIGURE 11 – la machine à état

### 3.6 Fichier Top

```

1  entity TOP is
2    port
3    (
4      Clk          : in std_logic; -- Horloge 100 MHz
5      Reset        : in std_logic; -- Reset Asynchrone
6      Interrupteur : in std_logic_vector(7 downto 0);
7      SORTIE_DCC   : out std_logic -- Bit sent
8    );
9  end TOP;
10
11 architecture Behavioral of TOP is
12   signal Clk1M_s       : std_logic; -- Horloge 1 MHz
13   signal Reset_Tempo_s : std_logic;
14   signal Start_Tempo_s : std_logic; -- Dmarrage de la Temporisation
15   signal Fin_Tempo_s   : std_logic;
16   -- Drapeau de Fin de la Temporisation
17   signal Reset_DDC0_s  : std_logic;
18   signal G0_0_s         : std_logic; -- Démarrage de la Temporisation
19   signal FIN_0_s        : std_logic;
20   signal Reset_DDC1_s  : std_logic;
21   signal G0_1_s         : std_logic;
22   signal FIN_1_s        : std_logic;
23   signal Com_REG_s     : std_logic_vector(1 downto 0);
24   -- Command for register
25   signal DCC_in_s      : std_logic;
26   signal Trame_DCC_s   : std_logic_vector(50 downto 0); -- Trame DCC
27   signal DCC_0_s        : std_logic;
28   signal DCC_1_s        : std_logic;

```

Programme 6 – Entité du fichier Top

### 3.7 Simulation du fichier Top

Comme tous les modules testés fonctionnent correctement individuellement, nous les avons assemblés et testés par simulation (Top), puis vérifiés à l'aide de l'oscillateur, et enfin testés avec le modèle final (la maquette).

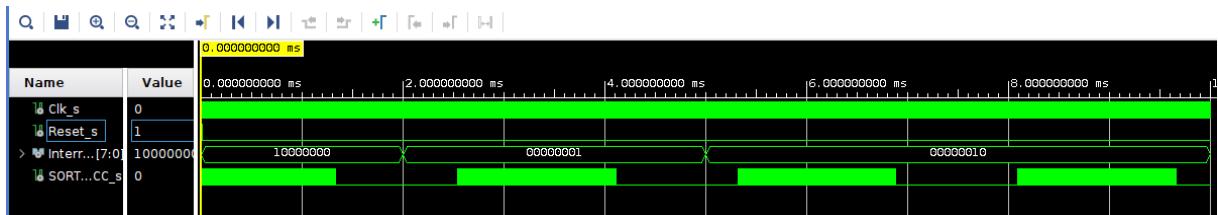


FIGURE 12 – Simulation du Top montrant une suite de trames

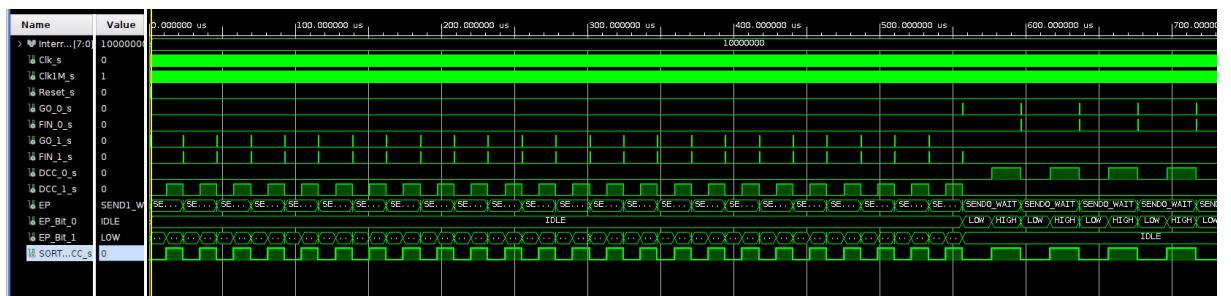


FIGURE 13 – Simulation du Top montrant une trame de plus près ainsi que les signaux internes

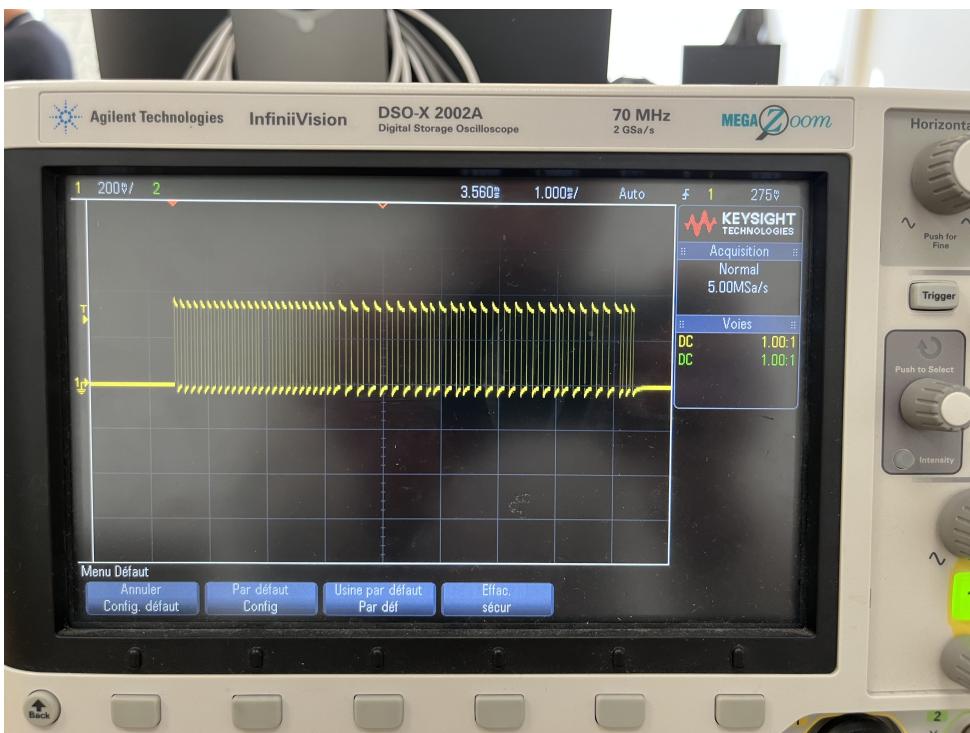


FIGURE 14 – Simulation sur oscilloscope

## 4 Conception de l'IP DCC

Cette deuxième partie consiste principalement à l'intégration de la Centrale DCC au sein d'un système **Microblaze**.

## 5 Bilan d'avancement

Nous avons bien complété la première partie concernant les modules VHDL. Le fonctionnement de la centrale DCC a été bien testé sur la plate-forme des trains.

Néanmoins, du au temps restant nous n'avons pas réussi a finir la mise en oeuvre de l'IP avec le Microblaze.

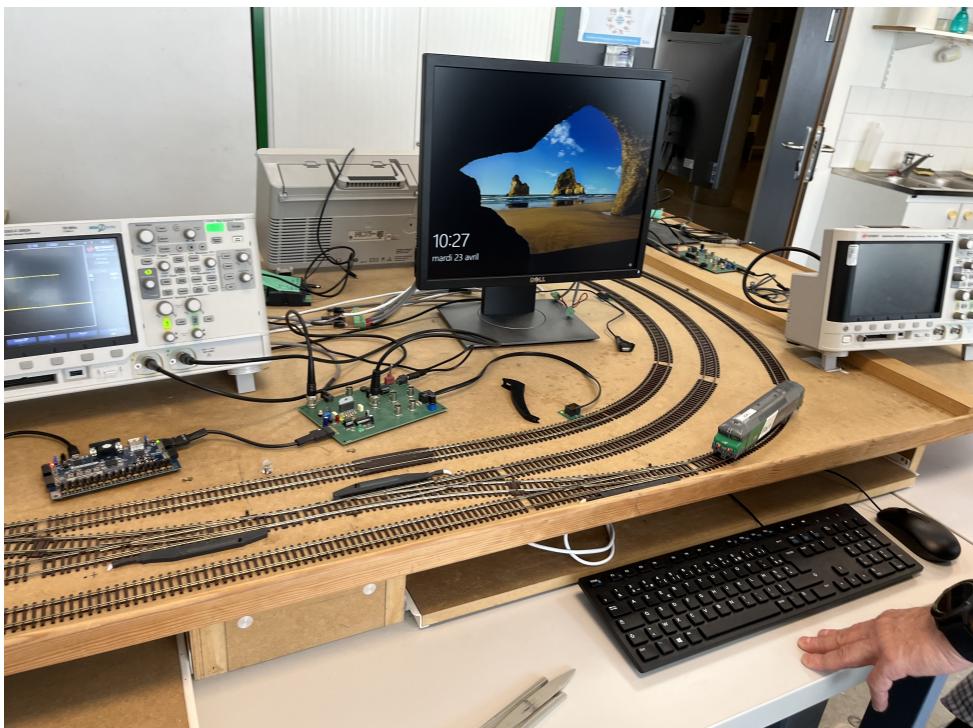


FIGURE 15 – Le Test sur la maquette

