Compte rendu TP Centrale DCC :

Réalisé par :

- MEDJAHED Mounia 28615513
- Xindan Zhang 21111176

Master 1 :

Systèmes Communicants SyScom.

Systèmes électroniques et Systèmes informatiques SESI.

Tous les codes et testbenchs sont en annexe

❖ Introduction :

Pour bien mener à terme ce projet on doit tout d'abord réaliser une partie matérielle en VHDL qui pourra générer en sortie la trame DCC souhaitée sur 51 bits afin de pouvoir piloter les trains, par la suite on réalisera une partie logicielle en langage C sur VITIS pour générer automatiquement cette dernière.

❖ Développement de la partie matérielle :

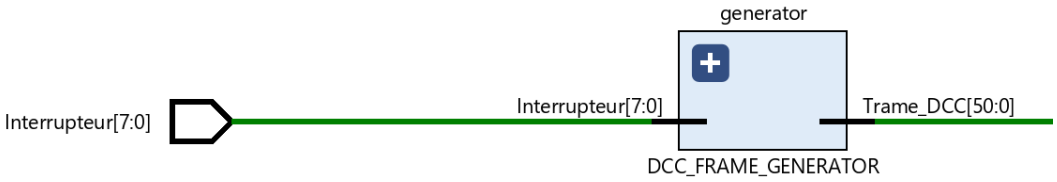
➢ *Générateur de trame :*

Pour générer la trame on utilisera le fichier générateur de trames fournie pour le teste, il a une entrée pour les interrupteurs sur 8 bits qui correspond aux 8 switches de la carte Basys 3 et une sortie sur 51 bits qui correspond à la trame DCC.

Ce module de test nous permet de choisir une trame grâce à une combinaison d'interrupteurs en entrée, c'est pour s'assurer dans un premier temps du bon fonctionnement de la centrale DCC avant de l'intégrer à la partie logicielle MicroBlaze, le code a déjà été fournie pour remplir les 51 bits en sortie et choisir les commandes dans on a besoin, on s'est inspirée du modèle suivant :

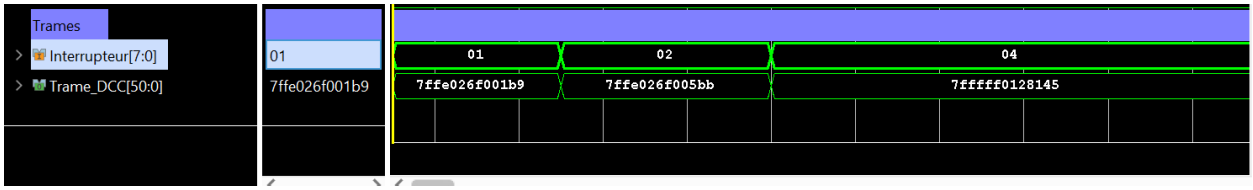
Composition de la Trame	Détail
Préambule	Suite d'au moins 14 bits à 1
Start Bit	1 bit à 0
Champ d'adresse	Adresse de la locomotive à commander (1 octet)
Start Bit	1 bit à 0
Champ de commande	Commande envoyée au train (de 1 à 2 octets*)
Start Bit	1 bit à 0
Champ de contrôle	XOR entre les octets des deux champs précédents, (1 octet). Permet de détecter d'éventuelles erreurs de transmission
Stop Bit	1 bit à 1

Au début on a un préambule sur 14 bits à 1, puis un bit de start à 1 en suite on indique l'adresse de la locomotive par exemple on a choisi la deuxième locomotive x"02" en hexadécimale, après un bit de start, par la suite viens le choix de la commande par exemple allumage des phares ,marche avant…ect ça dépendra des fonctions choisit, puis tout le reste champ de contrôle, bit de stop..ect, on précisera seulement qu'il y'a certaines commandes qui sont sur 2 octets(16 bits) par exemple les messages d'annonces les fonctions de f13 à 20 donc on fera bien attention au champ de commande sur 1 ou 2 octets qui dépendra du choix de la fonction. Tout cela fera un totale de 51 bits pour chaque trame.



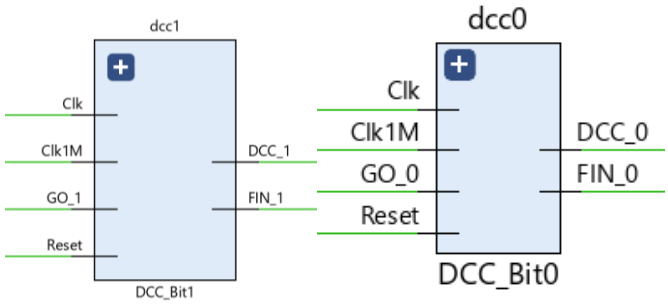
Voir le code en annexe Generateur_Trames.vhd :

Simulation Générateur_Trames :

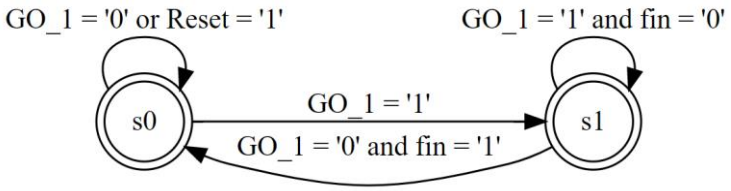


Ici on a fait le test avec les 3 premiers interrupteurs switchs 0,1,2.

- ***DCC_ Bit_ 1_0 :***



Ces deux blocs permettent de générer respectivement un bit à 1 ou à 0 au format ***DCC.***
Pour la réalisation du module DCC on doit utiliser une machine à état MAE comme représenté sur la figure ci-dessous (c'est la même pour les deux modules) sauf le nom des signaux qui changent.
Elle est cadencée par deux horloges une à 100 Mhz =>Clk100M et l'autre à 1 Mhz=>Clk.



On a simplement 2 états pour être le plus optimale que possible.

Notre code VHDL comprend trois processus principaux : le registre d'état, la Machine à états et le compteur.

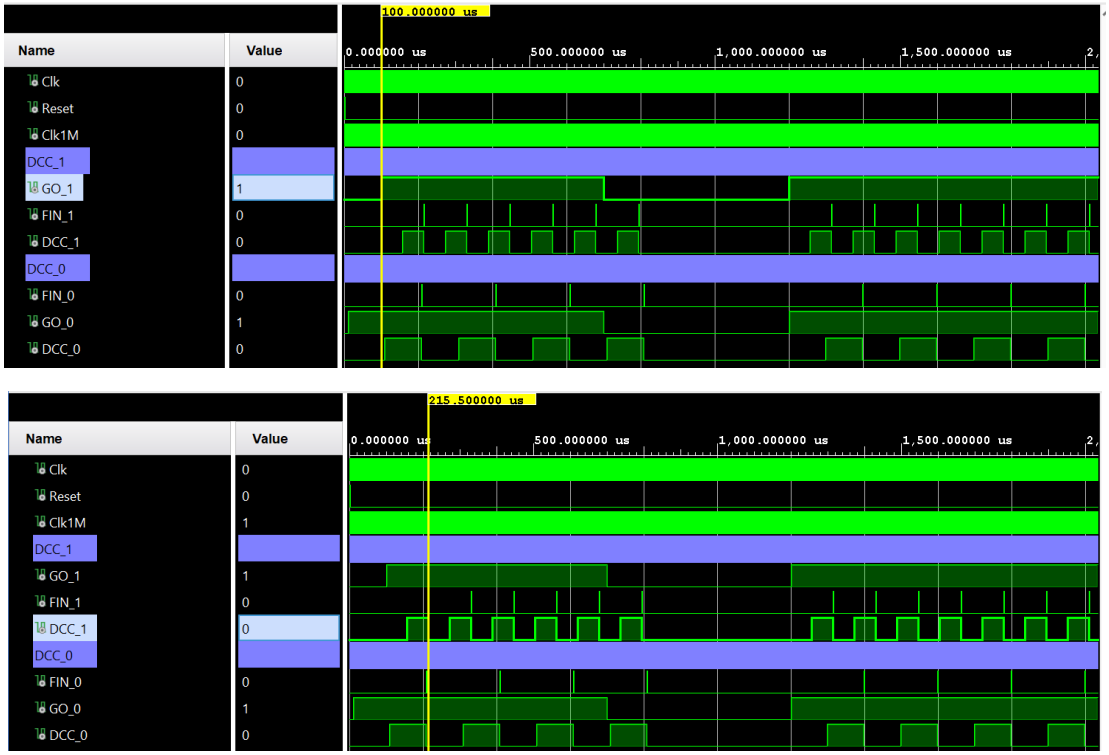
Par exemple, pour DCC_1 : Le registre d'état prend en charge de gérer l'état actuel du système, qui est mis à jour à chaque front montant de l'horloge. La Machine à état, quant à elle détermine la sortie et l'état futur en fonction de l'état présent. En parallèle, le compteur est incrémenté à chaque cycle d'horloge de 1 MHz lorsque l'état présent est S1. Le compteur détermine si une impulsion est à '0' ou à '1'. En ce qui concerne la transition c'est-à-dire 58 µs à '0' puis à '1' on a utilisé un signal interne dcc_out qui dépendra de la valeur du compteur. Puis on enverra dcc_out en sortie sur DCC_1. Le signal fin_cpt est mis à 1 lorsque le compteur atteint la valeur 115, indiquant que la durée spécifique de l'impulsion a été atteinte.

Tout en tenant compte des temps comme le montre le tableau ci-dessous :

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

Voir les codes en annexe DCC_Bit1.vhd, DCC_Bit0.vhd.

Simulation DCC_1 :

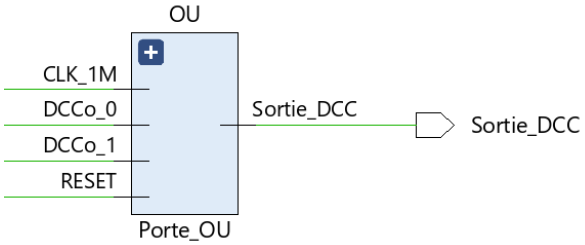


La durée de 1 bit à '1' pour DCC_1 et de 115 µs comme le montre la simulation.

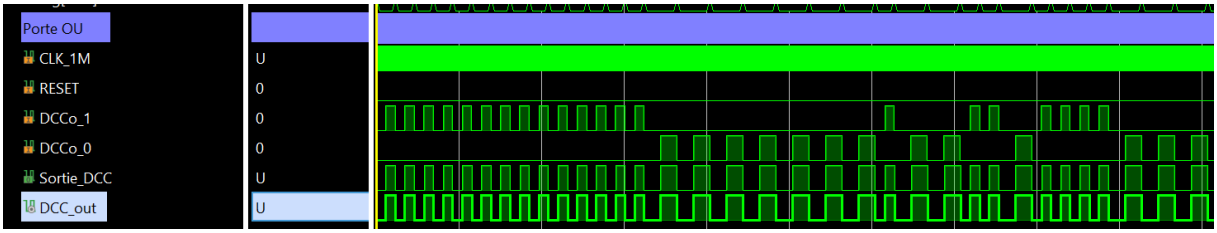
D'après la simulation la sortie DCC_1 est envoyé quand G0_1 est à '1' et dès qu'on finit la transmission du bit une impulsion sera visible au niveau la sortie FIN_1 qui est mise à '1' (pareillement pour le DCC_0).

- ***Porte Ou :***

Elle joint les deux signaux de sorties DCC_1 et DCC_0 en entrée et elle sort notre trame DCC qu'on visualisera sur l'oscilloscope et qu'on enverra sur les locomotives.

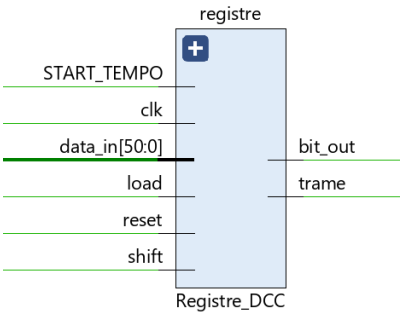


Simulation Porte_OU:



Voir le code en annexe PORTE_OU.vhd.

➢ *Registre DCC :*



Pour ce module, on a implémenté un registre qui fonctionne soit en chargement parallèle, mémorisation ou décalage, on a trois entrées Load, Shift et START_Tempo et deux sorties trame et bit_out.

Load =’1’ : pour le chargement parallèle de la trame venant du générateur de trame.

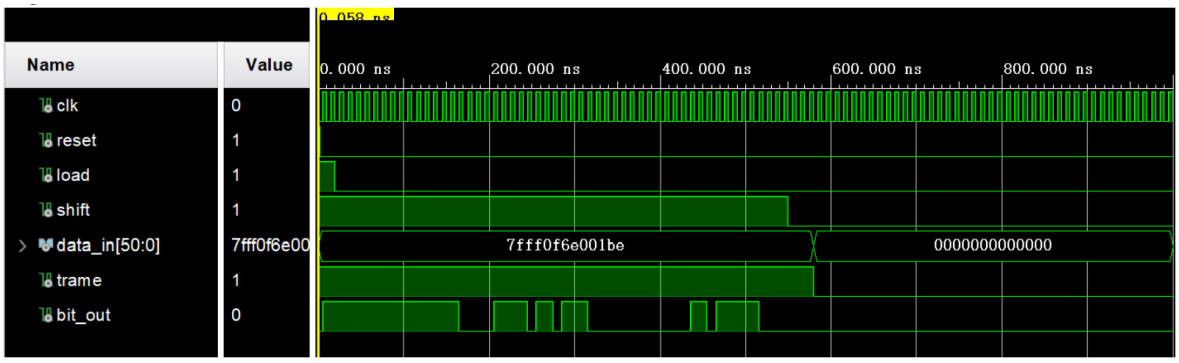
Shift =’1’ : on décale tous les bits du registre d'un rang vers la droite

START_Tempo =’1’ : quand il est à ‘1’ c’est pour indiquer qu’on entame le comptage du délai 6 ms entre chaque trame.

La sortie bit_out correspond au bit de poids fort MSB du registre c’est-à-dire bit (50), on a rajouté un signal trame pour communiquer à la MAE qu’il y’a une trame en entrée. Ce choix est plutôt judicieux par rapport à la MAE afin d’éviter de la complexifier.

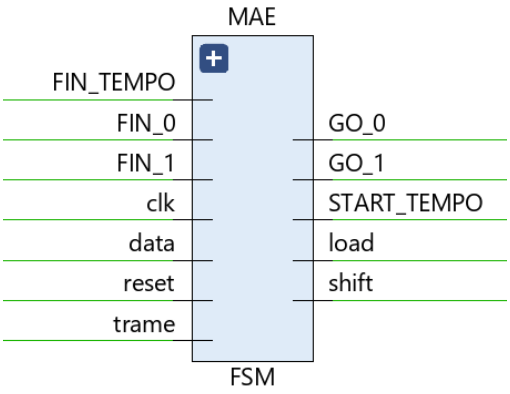
Voir le code en annexe Registre_DCC.vhd.

Simulation Registre DCC :

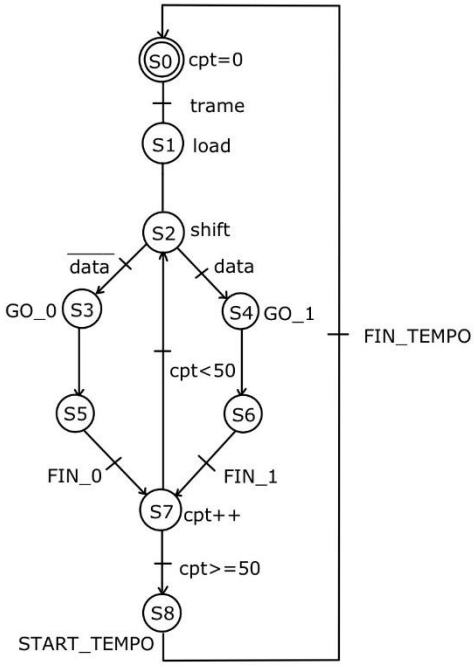


Avant d’écrire la machine à état, on a testé le fonctionnement du Registre_DCC avec shift égale toujours à 1. On voit bien la trame de sortie bit_out dans la simulation.

- ***Machine à état MAE globale :***



Pour la Machine à état globale elle va recevoir la sortie du registre DCC le bit_MSB sur data et c'est elle qui va commander les deux machines à état de DCC_1 et 0 pour leurs permettre de générer un bit à 0 ou à 1 ça dépendra du bit de la trame transmit par le REG_DCC.



S0 : Etat initial.

S1 : Lancement de la trame, commande envoyée par le registre DCC pour indiquer la présence d'une nouvelle trame, on commence le chargement.

S2 : Prend le nouveau bit en entrée puis effectue un décalage shift dans le registre.

S3 : Lancement de la création du bit 0 en format DCC c'est-à-dire Go_0='1'.

S4 : Lancement de la création du bit 1 en format DCC c'est-à-dire Go_1='1'.

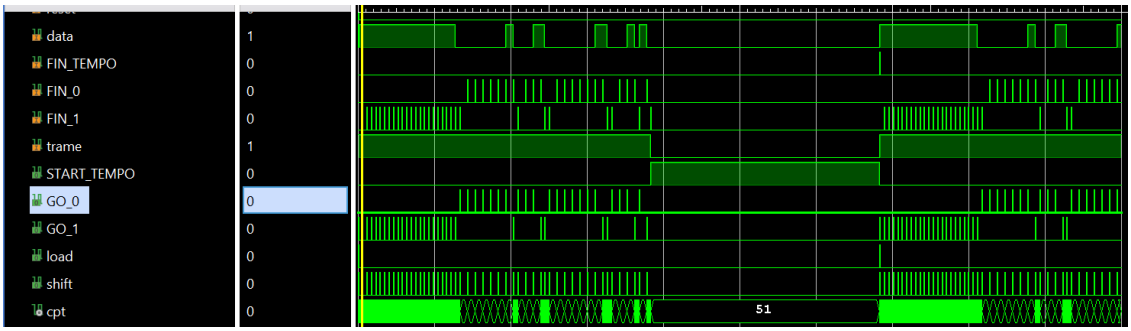
S5 et S6 : Génération du bit 0 ou 1 (200 µs, 115 µs).

S7 : Incrémentation du compteur cpt++ pour compter les 0 ou 1 bits de la trame.

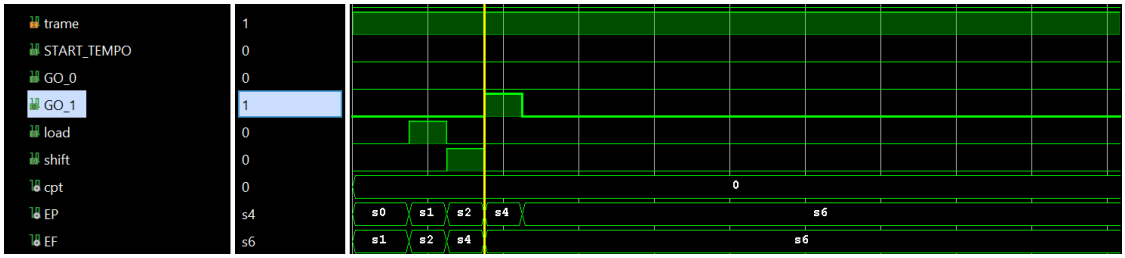
S8 : START_TEMPO est activé pour poursuivre de compter le délai, puis on repart vers l'état initiale en mettant FIN_TEMPO à '1' quand le délai est écroulé.

Voir le code en annexe FSM.vhd.

Simulation MAE :

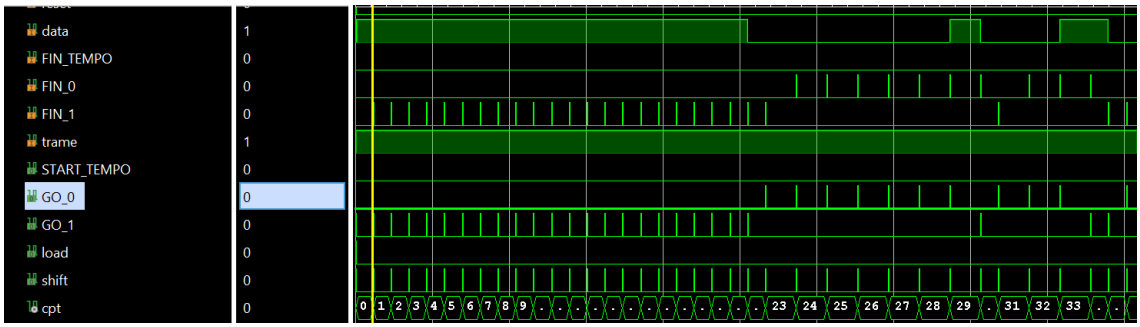


Pour simuler la machine à état, on a utilisé la même trame que bit_out du registre_DCC .



D'après la simulation et vu que la trame est entamée par un préambule à 14 bits à 1, on voit le signal load qui passe à '1 ' pour charger la trame, et puis Go_1 qui est mis à '1' pour générer le premier bit de la trame qui est à '1' logiquement, par la suite on fait le décalage shift='1'pour la suite de la trame.

En poursuivant la simulation un signal FIN_1 sera mis à '1' à la fin de transmission du premier bit à 1.



Ainsi de suite on poursuit la transmission de la trame bit à bit en décalage à partir de la sortie du registre (bit MSB) comme on l'observe sur data jusqu'à la fin de la première trame.

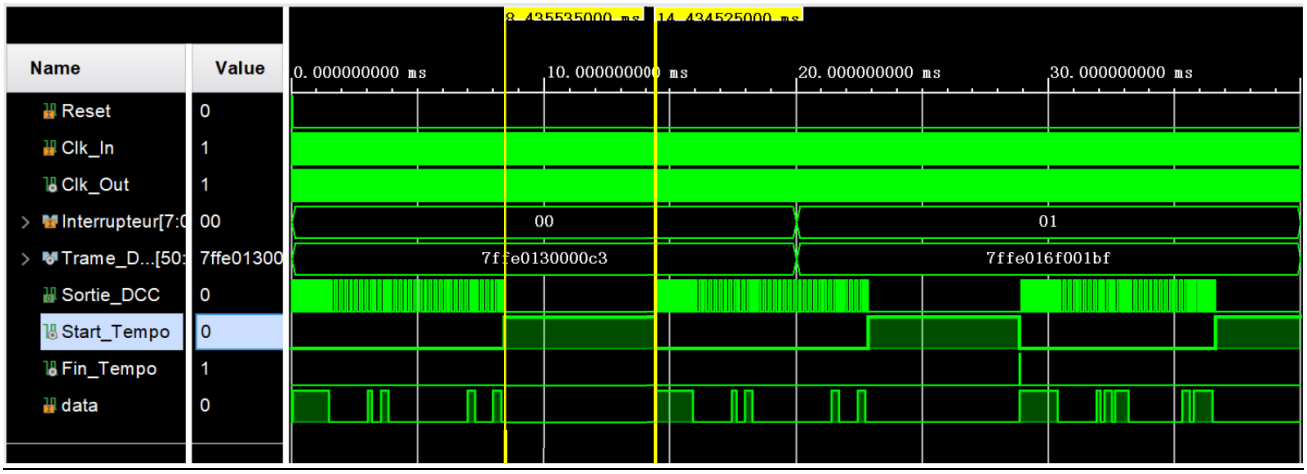
A la fin de la première trame START_TEMPO est mis à '1' pendant 6 ms et data=0 pas de trame pendant 6 ms.

- ***Top_DCC :***

Après avoir écrit tous les modules on fera un fichier TOP de la centrale qui regroupera tous les modules et après on fera une simulation globale.

Voir le code en annexe TOP.vhd.

Simulation Top_DCC :



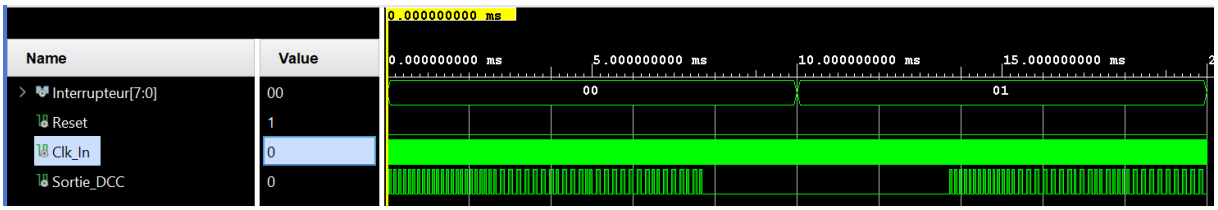
On observe la Sortie_DCC qui représente notre trame qu'on enverra sur les locomotives. On continue de tracer les chronogrammes jusqu'à la fin de la première trame c'est-à-dire atteindre 51 bits.

On voit clairement d'après les chronogrammes que START_Tempo passe à '1' pour permettre de démarrer le compteur de temporisation de délai qui est de 6 ms d'où la sortie sera mise à '0' Sortie_DCC='0'pendant ce temps, puis on voit le changement de l'état de l'interrupteur (switch) à la fin, pour charger la seconde trame.

Sortie_DCC représente la sortie de la porte OR entre DCC_0 et DCC_1 qu'on enverra par la suite sur nos trains.

On s'assurera que le délai de 6 ms est bien respecté entre les deux trames, on remarquera également que la largeur des impulsions sur la sortie DCC est variable ce qui est logique selon si c'est un '1' ou un '0' qui a été envoyé, car la durée d'un bit à '1' est (116 µs) est plus courte que celle du bit à 0 est (200 µs) on vérifiera bien ces délais car sinon les locomotives ne pourront pas déchiffrer les commandes envoyées ce qui peut entrainer des problèmes pour la mise en marche des trains.

Simulation Post-Synthèse de la centrale DCC :



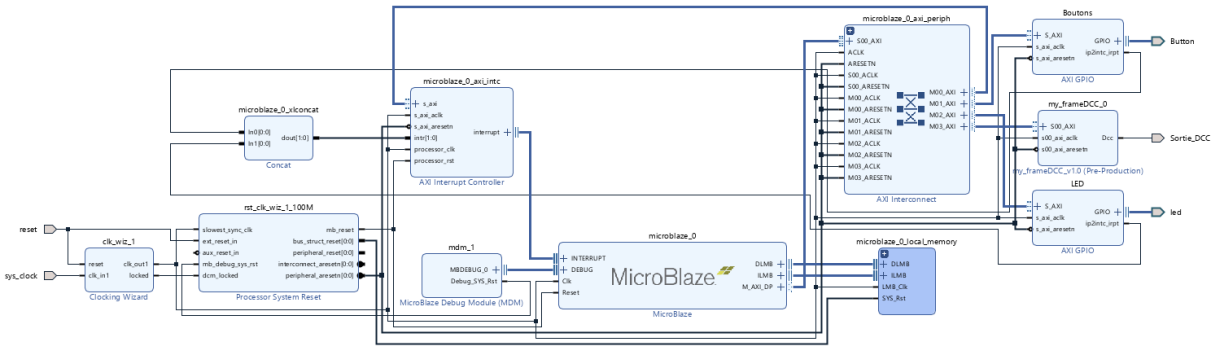
On observe un comportement quasi parfait dans la simulation Post-Synthèse qui définit plus précisément le comportement sur la carte au niveau de la sortie DCC et un délai de 6 ms entre chaque trame.

❖ Développement de la partie logicielle VITIS :

En ce qui concerne cette partie du projet, nous allons intégrer notre centrale DCC comme IP au sein d'un système MicroBlaze afin de pouvoir interagir avec ce dernier via le bus AXI.

Pour ce faire on devra packagée la centrale DCC sous forme d'IP, le seul changement par rapport à l'architecture précédente c'est que le module Générateur_Trames sera remplacé par un AXI Wrapper qui permettra de rattacher la Centrale DCC au bus AXI du MicroBlaze.

Block design :



On avait intégré l'IP sous le nom my_frameDCC_0 d'où on récupère la sortie DCC.

Il y'a deux interfaces GPIO, une pour les 16 LEDs et l'autre pour les trois boutons qu'on a utilisé dans notre code c bouton_gauche (BG), bouton_centre (BC), bouton_droit (BD) que vous verrez plus en détails ci-dessous.

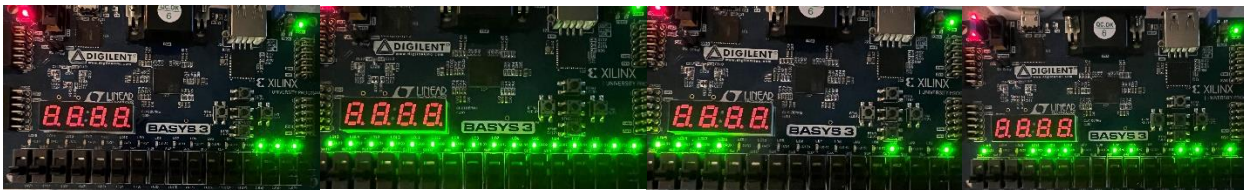
Explication code C :

Au début on a défini les constantes et initialiser les variables nécessaires pour le bon fonctionnement du programme. Ensuite, l'initialisation des GPIO qui sont utilisés pour interagir avec les LEDs et les boutons a été mise en œuvre.

Le programme entre ensuite dans une boucle principale, où il lit l'état des boutons pour former une trame et l'enregistrer dans des registres du DCC (REG0 et 1) et l'afficher en utilisant les LEDs pour bien vérifier le fonctionnement du code. Par exemple, dans l'état STATE_ADRESSE, le programme attend que l'utilisateur appuie sur le bouton droit pour indiquer l'adresse du train. Lorsque l'utilisateur appuie sur le bouton centre, le programme passe à l'état STATE_COMMANDE. La logique similaire s'applique à tous les autres états.

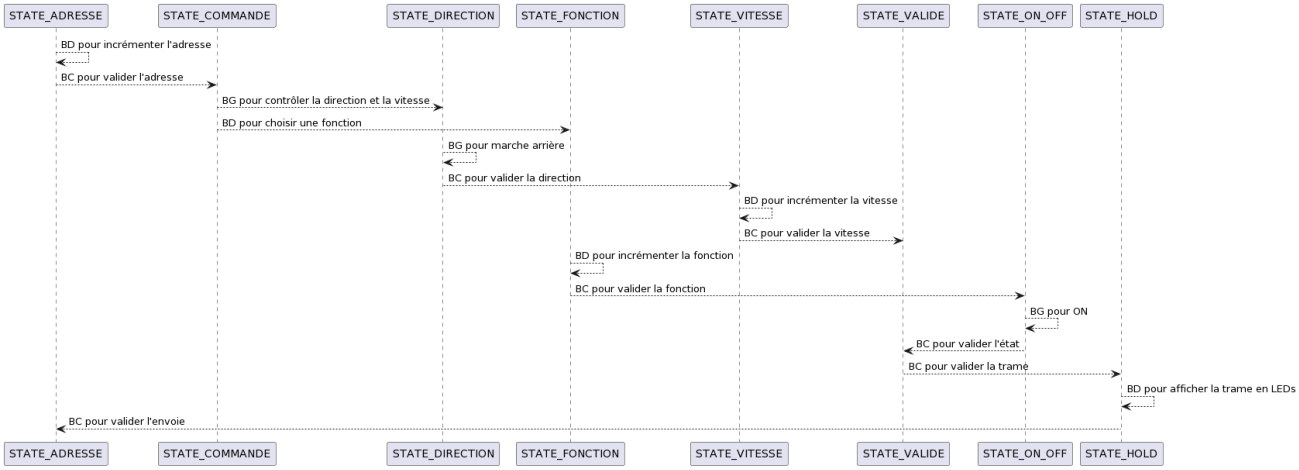
Enfin, en fonction de l'entrée de l'utilisateur et de l'état actuel du système, une trame est construite pour être envoyée par le bus AXI aux trains.

Voici l'affichage des LEDs indiquant le contenue des registres sur 51 bits à la fin de l'envoie de trame. Ici, on a fait une marche avant du train numéro 1 de vitesse 00111.



Voir le code en annexe main.c :

Organigramme du code C :

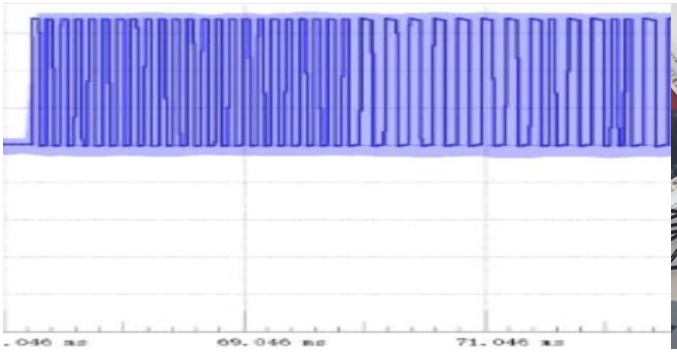


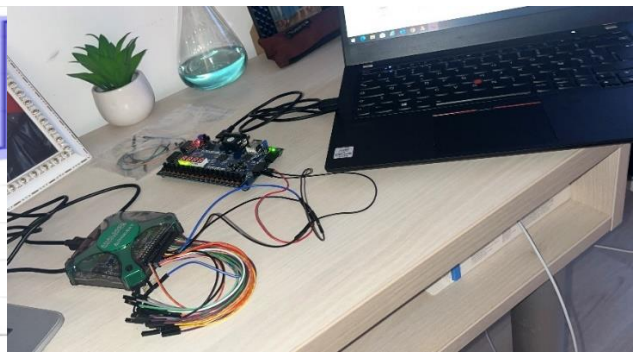
❖ Bilan d'avancement :

En ce qui concerne la partie développement matérielle en VHDL, on a réussi à faire marcher les trains et afficher les différentes trames souhaitées sur l'oscilloscope de la salle de TP en manipulant les switches de la carte Basys 3, on a fait marcher la locomotive 2 en avant, en arrière, klaxon, phares, l'annonce…etc.

Pour la partie développement logicielle on a fait le code C on l'a testé la première fois il ne marchait pas sur les trains, par la suite on a continué chez nous et j'ai testé la sortie DCC sur l'oscilloscope la carte AD2 en branchant la sortie de Basys 3 JC4 et GND à l'oscilloscope de la carte AD2, je n'ai pas eu totalement la trame en sortie tout en sachant qu'on ne pouvait plus manipuler les trains, car malheureusement les séances de tests été déjà achevés.

Vous trouverez ci-dessous quelques photos de la suite du test sur la carte Discovery AD2 :





C'était un projet très constructif et intéressant, merci !

Fin du Projet

Tous les codes et testbenchs sont en annexe