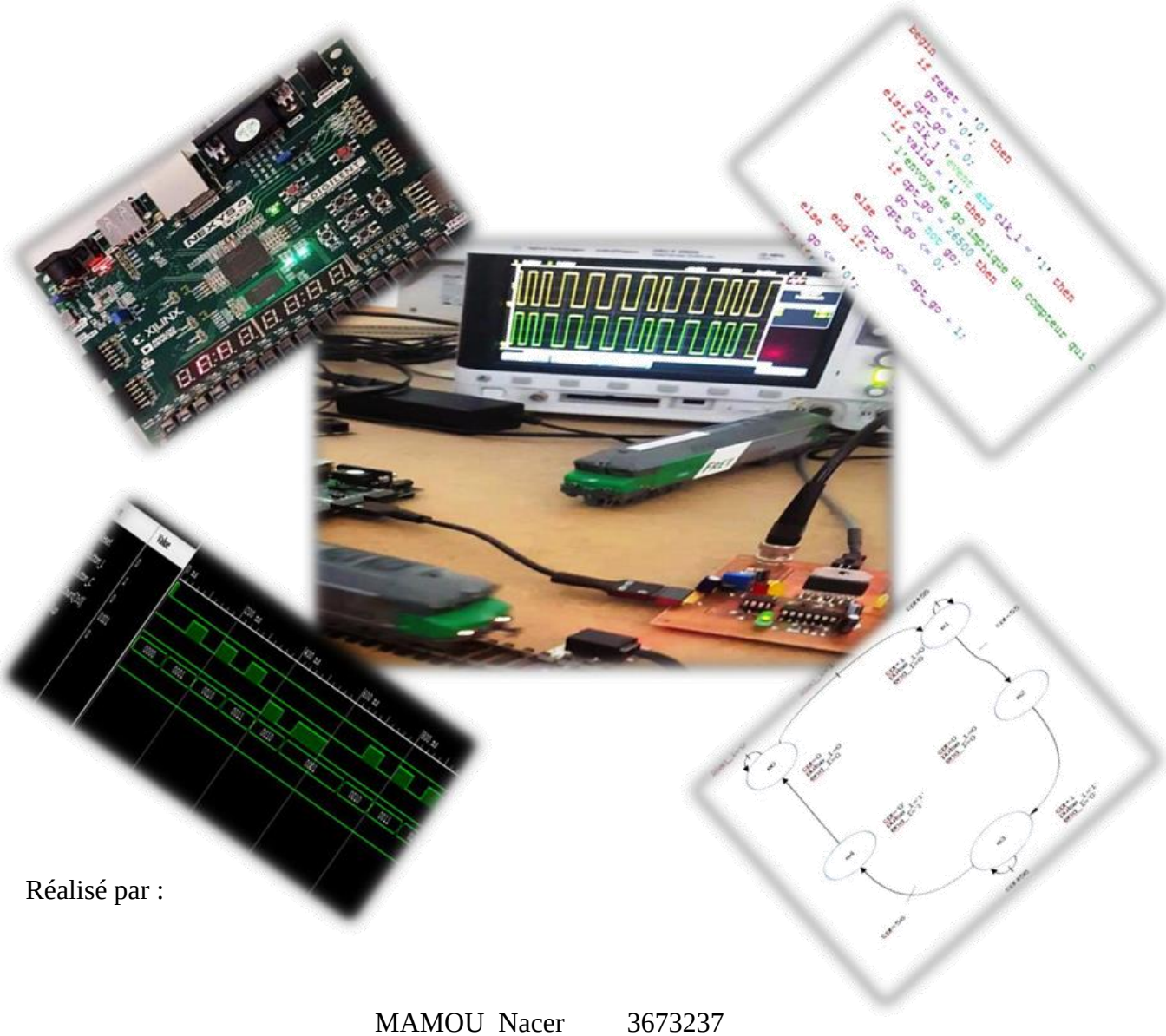
SORBONNE UNIVERSITÉS

Master Informatique

M1 Spécialité SESI

UE 4I108 - J.Denoulet

Rapport projet FPGA1: Systèmes Programmables



Réalisé par :

MAMOU Nacer 3673237

CHAHBOUNE Rafiq 3526615

Mai, 2018

Sommaire :

I)-Introduction :

On se propose dans cette 4ème partie, de concevoir et implémenter un système H/S sur FPGA pour commander des petits trains, en leur envoyant un message numérique construit en respectant le protocole DCC.

Ce protocole est utilisé dans le modélisme ferroviaire pour commander individuellement des locomotives ou des accessoires en modulant la tension d'alimentation de la voie. Les locomotives reçoivent donc les informations directement à partir des raies.

L'implémentation de notre système sera sur la carte **FPGA-Nexus 4 DDR** de chez le fabriquant **XILINX** , qui ne peut pas fournir suffisamment de courant de sortie, c'est pour cela une carte (Booster) est utilisé pour amplifier le signal avant de l'envoyer aux raies.

Ce projet est composé de deux parties :

Une première partie dans laquelle, Nous nous sommes intéressés principalement à :

- Concevoir une architecture fonctionnelle permettant de répondre aux spécifications du protocole DDC.
- Développer le code **VHDL** de tous les modules nécessaires pour construire l'IP.
- Ecrire des Testbenchs permettant de simuler correctement notre système
- Adapter le codage pour assurer une synthèse correcte par les **outils de Vivado.**
- Tester l'ensemble du système sur la carte en vérifiant La génération de la trame en utilisant un oscilloscope.
- Enfin, valider notre système sur la plateforme.

Dans la deuxième partie, Nous avons intégré notre IP sur un processeur de type microblaze, cela permet choisir la provenance des commandes vers notre IP, ainsi offrir la flexibilité d'écrire des applications de haut niveau pour commander plusieurs IP en parallèle.

Pour réaliser cette partie , nous avons suivi le flot de conception qui nous a été fourni durant les partie 2 et 3 des TPs déjà effectués.

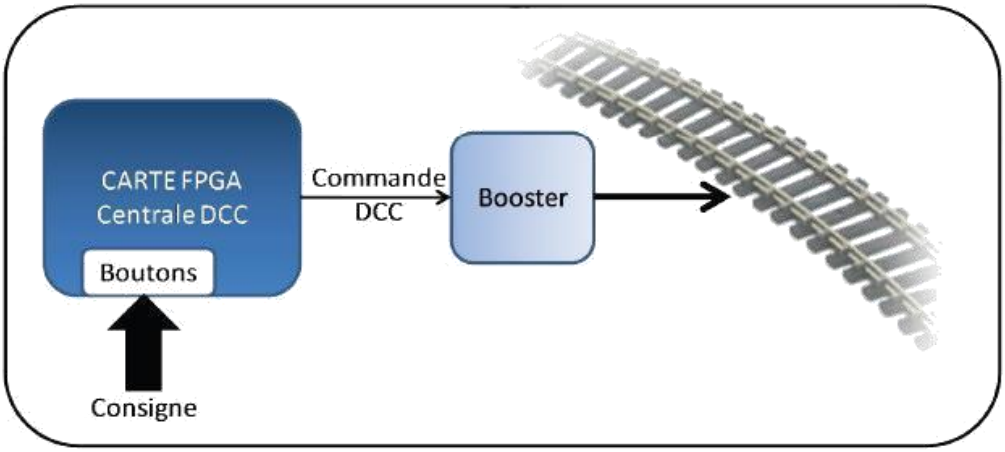


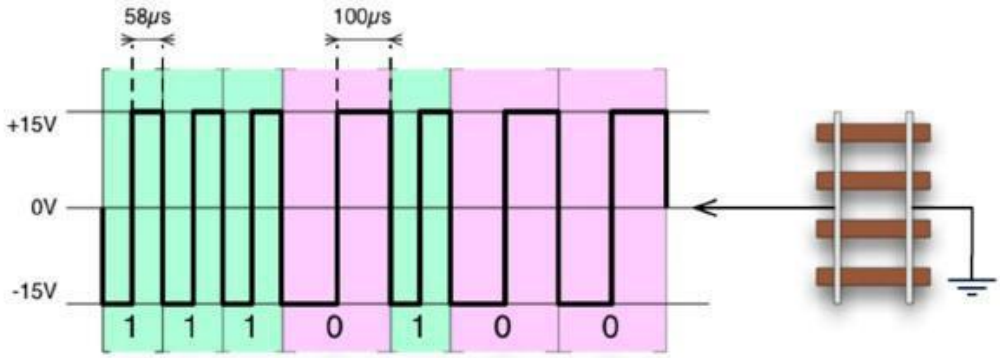
Figure 1: Synoptique du système de commande

II. Description du protocole DCC :

Le protocole DCC (Digital Command Control) est un standard qui présente les règles pour construire le message numérique utilisé pour commander les locomotives. Les trames sont formées d'une suite de bits.

Un bit est représenté par une période d'un signal carré, la durée de la demi-période permet de différencier la valeur du bit:

100 µS : permet d'envoyer un 0.
58 µS : permet d'envoyer un 1.



La commande d'un train se fait en envoyant périodiquement des trames espacées de 6 mS. Chaque trame est composée de 4 champs :

1. **Préambule :** (une suite de 14 bits à 1) pour l'initialisation du système
2. **Octet d'adresse :** (8 bits qui désigne l'adresse du train)
3. **Octet de données :** (commande envoyée au train : vitesse ou fonction)
4. **Octet de contrôle :** (XOR entre les deux octets précédents).

Les champs sont séparés avec l'insertion d'un start bit (bit à 0), et un stop bit (bit à 1) est toujours utilisé pour marquer la fin de la trame.

Étudions plus en détail l'octet de données, il peut correspondre à une vitesse ou à une fonction. Dans le cas d'une vitesse l'octet est de la forme : **01DXXXXX** où :

D indique la direction du train, si ce bit est à 1 le train se déplace en marche avant, sinon en marche arrière. Les 5 bits de poids faible sont des variantes de vitesse, selon le codage donné sur le TP.

Il existe plusieurs fonctions telles que l'activation d'un klaxon ou l'allumage des phares…, nous avons choisi d'implémenter que la commande de vitesse.

III. Architecture générale de la centrale DCC :

L'architecture de l'IP de notre centrale DCC est la suivante :

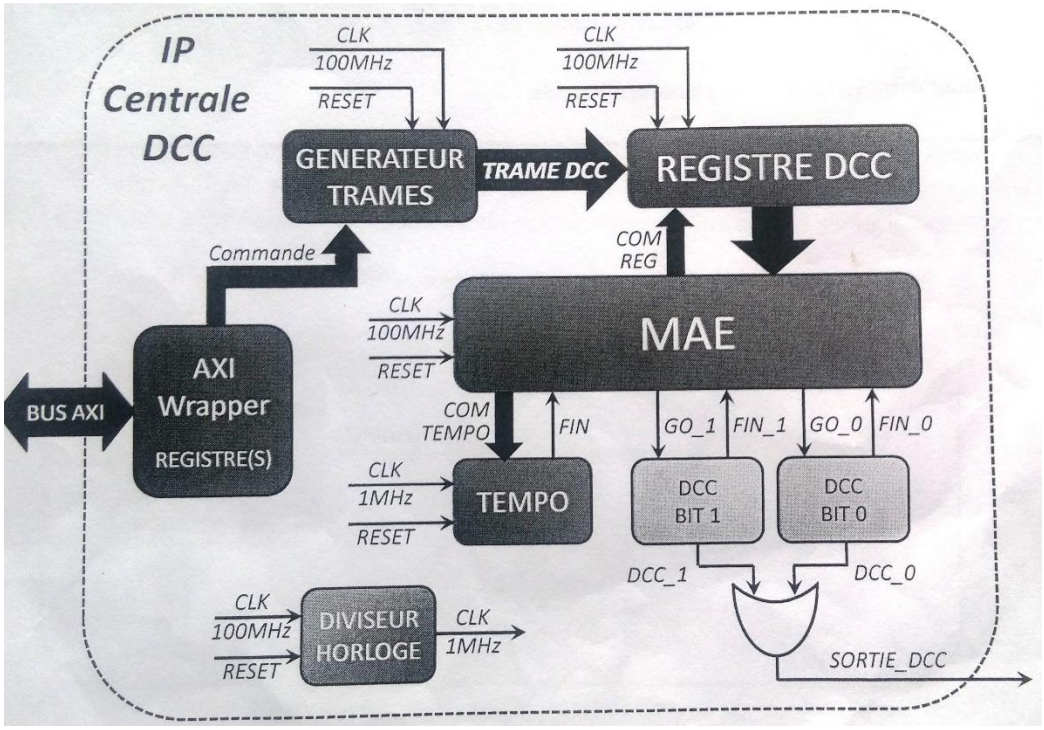


Figure 2: Architecture de l'IP centrale DCC

La centrale DDC est construite de plusieurs modules, chacun possède une fonction spécifique. Tous ces modules sont commandés par une machine à états principale.

Dans un premiers temps, le générateur de trame lit les valeurs des boutons et interrupteurs de la carte, et construit une trame de 42 bits (14+1+8+1+8+1+8+1=42).

Le registre DCC enregistre cette trame et effectue des décalages, la machine à état lit le MSB de la trame après chaque décalage.

Les modules BIT0 et BIT1 sont commandés par la machine à état pour générer le signal correspondant au bit lu.

Le temporisateur est commandé aussi par la machine à états afin d'assurer une temporisation de 6 **mS** entre les trames.

La sortie de IP sera la sortie d'une porte logique OU prenant en entrée les sorties des module BIT0 et BIT1.

Le diviseur d'horloge sert à adapter l'horloge interne du FPGA pour alimenter les modules BIT0, BIT1 et le temporisateur.

IV)-Architecture détaillée de la centrale DCC :

IV).1- Le diviseur d'horloge:

A partir de l'horloge du system (**F=100 Mhz**, **T=10 nS**), Ce module est utilisé pour générer une horloge de fréquence **1 Mhz** (période de **1 uS**).

La période de l'horloge à générer correspond à 100 périodes de l'horloge d'entrée, pour cela on inverse le signal de sortie (demi-période) à chaque 50 périodes comptées du signal d'entrée.

Nous avons décrit en VHDL le comportement de ce module , puis effectuer une simulation en écrivant un testbench :

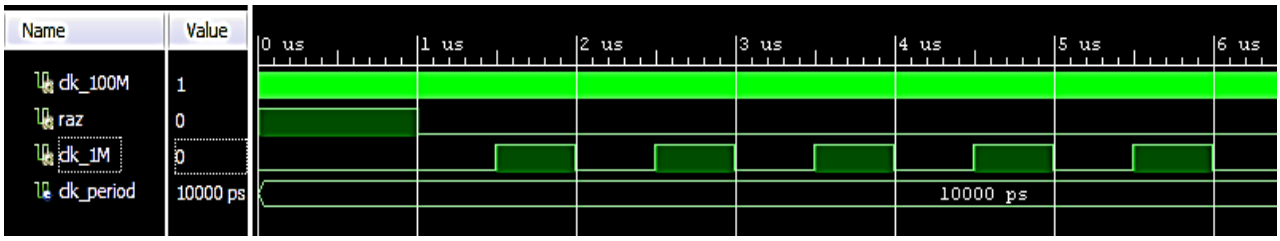


Figure 3 : graphe simulation du diviseur d'horloge.

On voit bien que le signal de sortie a une fréquence de **1 Mhz**, et cela après le relâchement de la remise à zéro.
Ce module a été synthétisé correctement par les outils de vivado, aucune erreur au warning est générée :

```
---------------------------------------------------------------------------
Synthesis finished with 0 errors, 0 critical warnings and 0 warnings.
Synthesis Optimization Runtime : Time (s): cpu = 00:00:26 ; elapsed = 00:00:
Synthesis Optimization Complete : Time (s): cpu = 00:00:44 ; elapsed = 00:0]
INFO: [Project 1-571] Translating synthesized netlist
INFO: [Netlist 29-17] Analyzing 2 Unisim elements for replacement
INFO: [Netlist 29-28] Unisim Transformation completed in 0 CPU seconds
INFO: [Project 1-570] Preparing netlist for logic optimization
INFO: [Opt 31-138] Pushed 0 inverter(s) to 0 load pin(s).
INFO: [Project 1-111] Unisim Transformation Summary:
No Unisim elements were transformed.
```

Figure 4 : Résumé du rapport de synthèse du module CLK_DIV

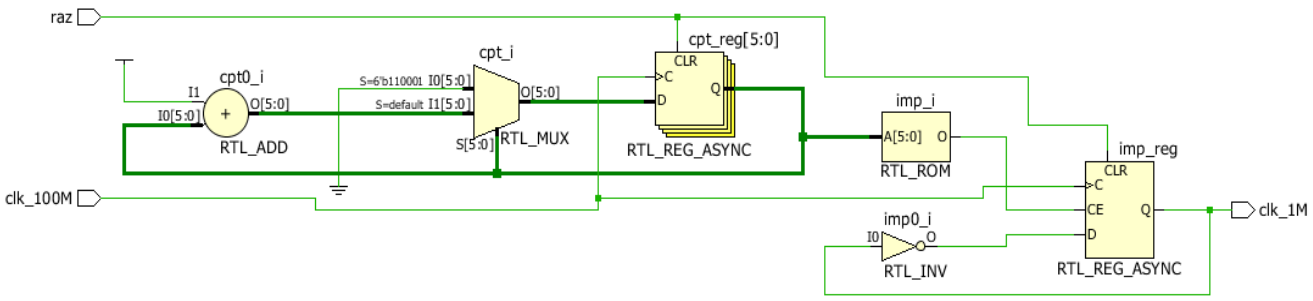


Figure 5 : Schéma RTL du Module CLK_DIV

IV).2- Le temporisateur:

Ce module assure une temporisation de **6 ms** à chaque appel de la machine à état à travers le signal d'entrée **START** , et à la fin génère un signal de sortie **FIN** pour la MAE lui indiquant la fin.

L'horloge d'entrée est de fréquence 1 Mhz (T=1 us) reçu à travers la sortie du diviseur d'horloge.

La temporisation correspond à 6000 périodes de l'horloge d'entrée. A la réception du signal **START**, le compteur s'incrémente après chaque période d'horloge, Le signal FIN est activé quand le compteur atteint 6000 périodes .

Nous avons décrit ce comportement en VHDL, puis on l'a simulé en écrivant un testbench approprié :

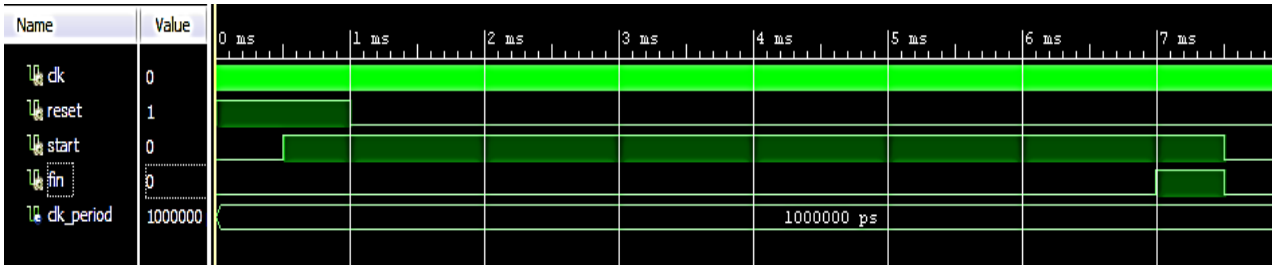


Figure 6 : Graphe de simulation du temporisateur

Ce module a été synthétisé correctement par les outils de Vivado, aucune erreur ou warning a été générée.

```
---------------------------------------------------------------------------
Synthesis finished with 0 errors, 0 critical warnings and 0 warnings.
Synthesis Optimization Runtime : Time (s): cpu = 00:00:22 ; elapsed = (
Synthesis Optimization Complete : Time (s): cpu = 00:00:40 ; elapsed =
INFO: [Project 1-571] Translating synthesized netlist
INFO: [Netlist 29-17] Analyzing 6 Unisim elements for replacement
INFO: [Netlist 29-28] Unisim Transformation completed in 0 CPU seconds
INFO: [Project 1-570] Preparing netlist for logic optimization
INFO: [Opt 31-138] Pushed 0 inverter(s) to 0 load pin(s).
INFO: [Project 1-111] Unisim Transformation Summary:
```

Figure 7 : Résumé du rapport de synthèse

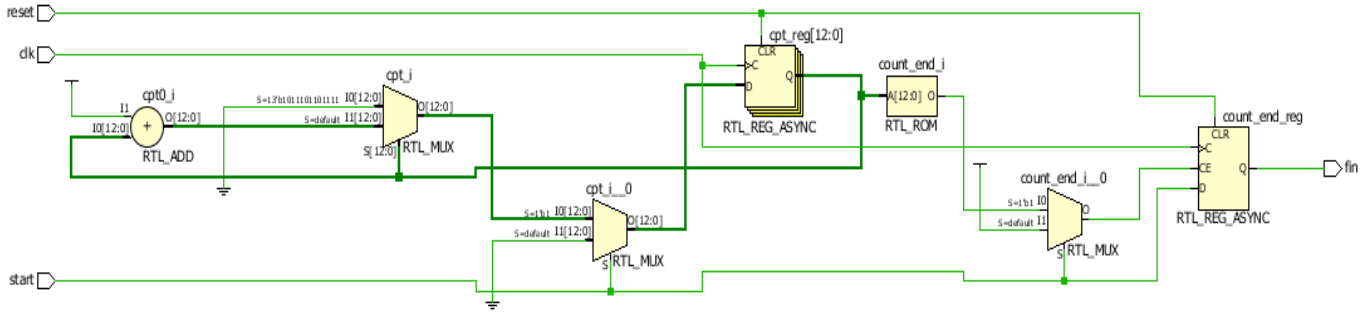


Figure 8 : Schéma RTL du Module TEMPORISATEUR

IV).3- Le Module SEND ZERO:

Ce module assure La génération d'un signal, dans un premier temps de 0 pendant 100 **us**, puis à 1 pendant 100 **uS**,

Nous avons décrit ce module en utilisant une machine de Moore à 4 états qui sont:

IDLE : le Module est inactif
GEN_LOW : Début de génération, le signal de sortie est maintenu à 0 pendant **100 uS**
GEN_HIGH : Début de génération, le signal de sortie est maintenu à 1 pendant **100 uS**
GEN_END : Information de la MAE de la fin de génération.

Les signaux d'entrée sont :

Clk : horloge de 1 Mhz
reset : remise à zéro go :
go : ordre émit par la MAE pour débuter la génération.

Les signaux de sortie sont :
dcc : sortie
started : à 1 pour signifier le début de génération
fin : à 1 pour signifier la fin de génération

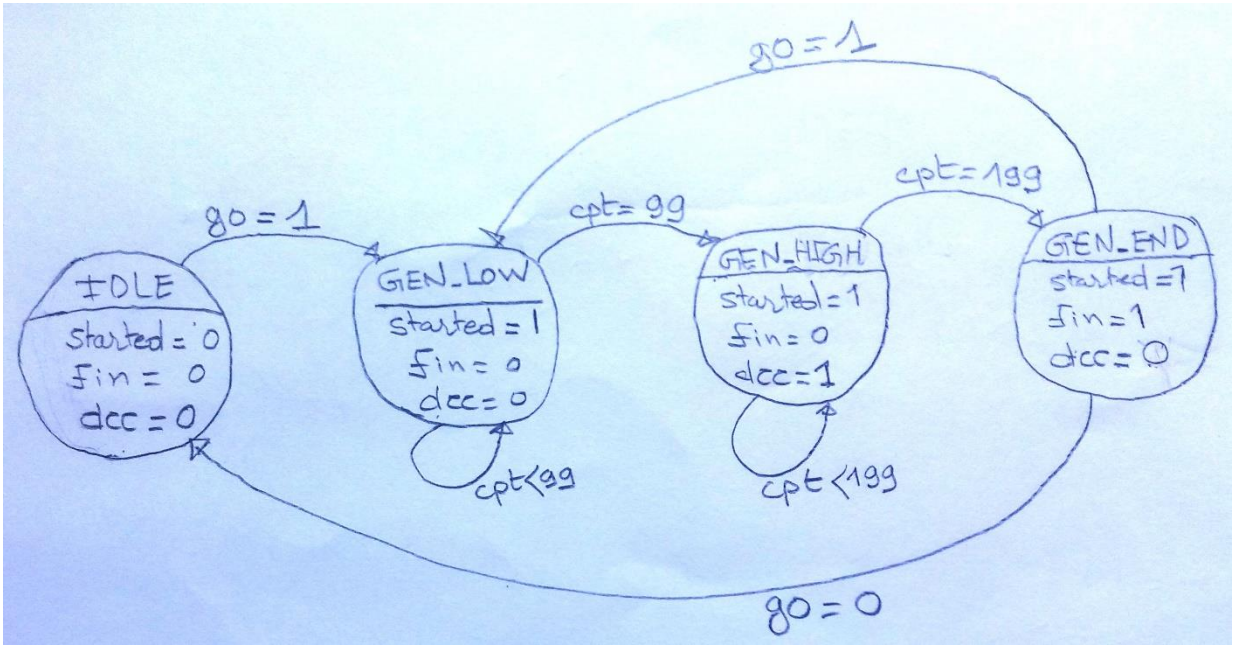


Figure 9 : diagramme des états du Module BIT0

Nous avons décrit ce comportement en VHDL, puis on l'a simulé en écrivant un testbench approprié :

La simulation est résumée dans les figures suivantes :

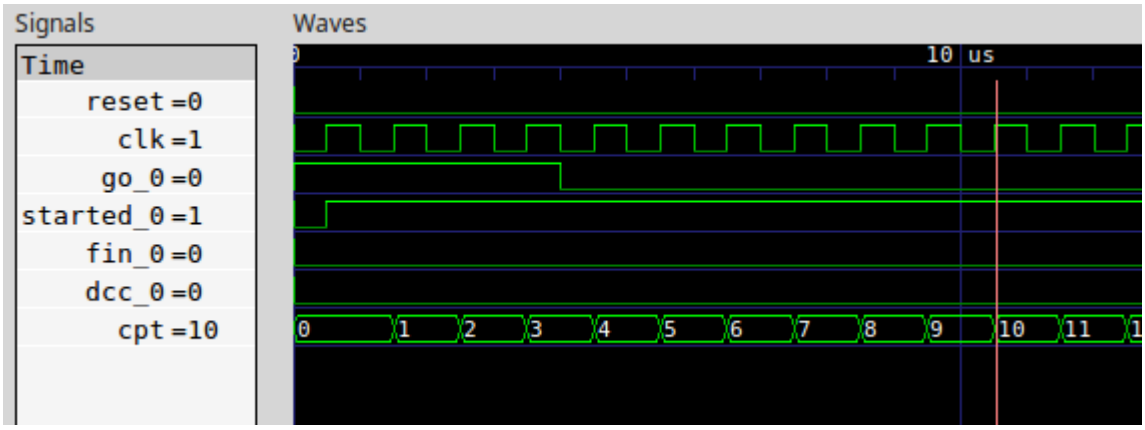


Figure 10 : Simulation du module BIT0, début de génération

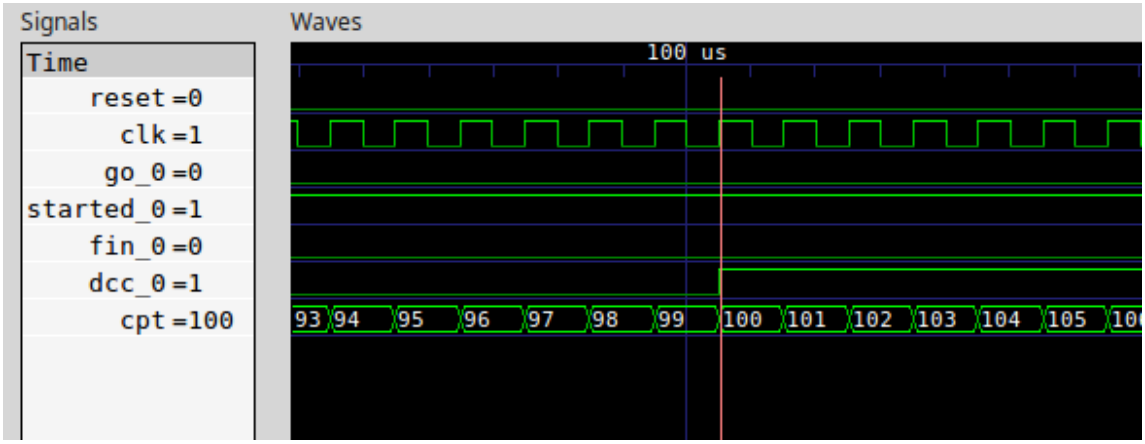


Figure 11 : Simulation du module BIT0, passage à l'état GEN_HIGH

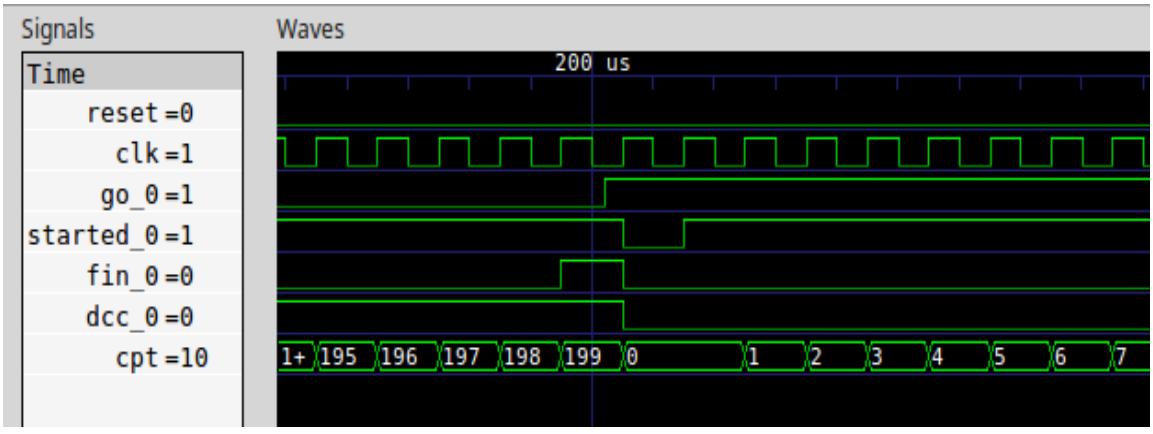


Figure 12 : Simulation du module BIT0, fin de génération

Ce module a été synthétisé correctement par les outils de vivado

```
---------------------------------------------------------------------------------
Synthesis finished with 0 errors, 0 critical warnings and 0 warnings.
Synthesis Optimization Runtime : Time (s): cpu = 00:00:23 ; elapsed = 00:00:34 . I
Synthesis Optimization Complete : Time (s): cpu = 00:00:41 ; elapsed = 00:01:04 .
INFO: [Project 1-571] Translating synthesized netlist
INFO: [Netlist 29-17] Analyzing 3 Unisim elements for replacement
INFO: [Netlist 29-28] Unisim Transformation completed in 0 CPU seconds
INFO: [Project 1-570] Preparing netlist for logic optimization
INFO: [Opt 31-138] Pushed 0 inverter(s) to 0 load pin(s).
```

Figure 13 : Résumé du rapport de synthèse

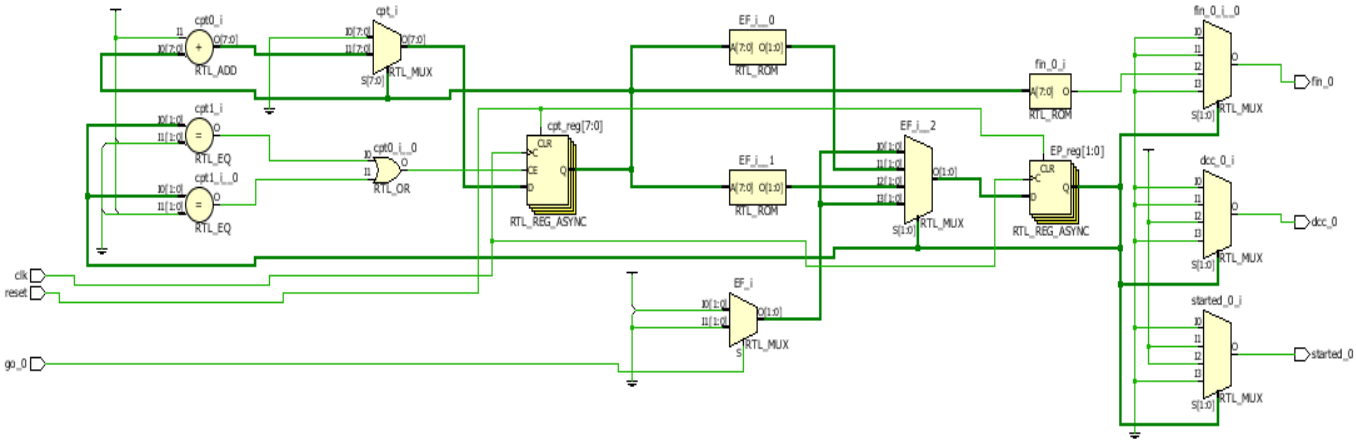


Figure 14 : Schéma RTL du Module BIT0

IV).4- Le Module SEND ONE:

Ce module assure La génération d'un signal, dans un premier temps de 0 pendant 58 uS , puis à 1 pendant 58 uS.
Nous avons gardé la même architecture du module SEND ZERO, en changeant juste la valeur du compteur :
Cpt= 57 pour la passassion de **GEN_LOW** à **GEN_HIGH**
Cpt= 115 pour la passion de **GEN_HIGH** à **GEN_END**

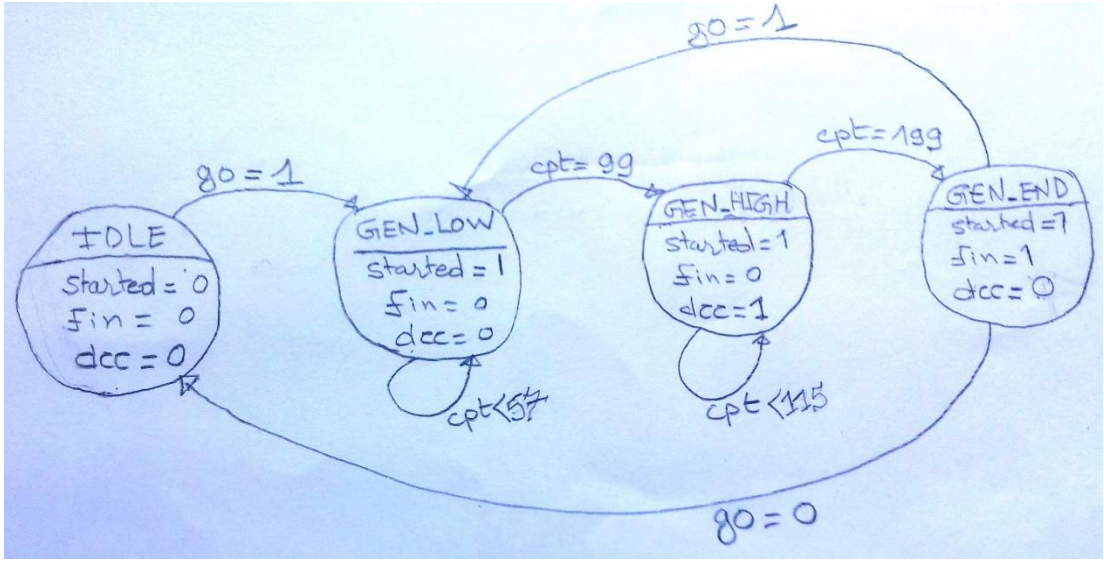


Figure 15 : diagramme des états du Module BIT1

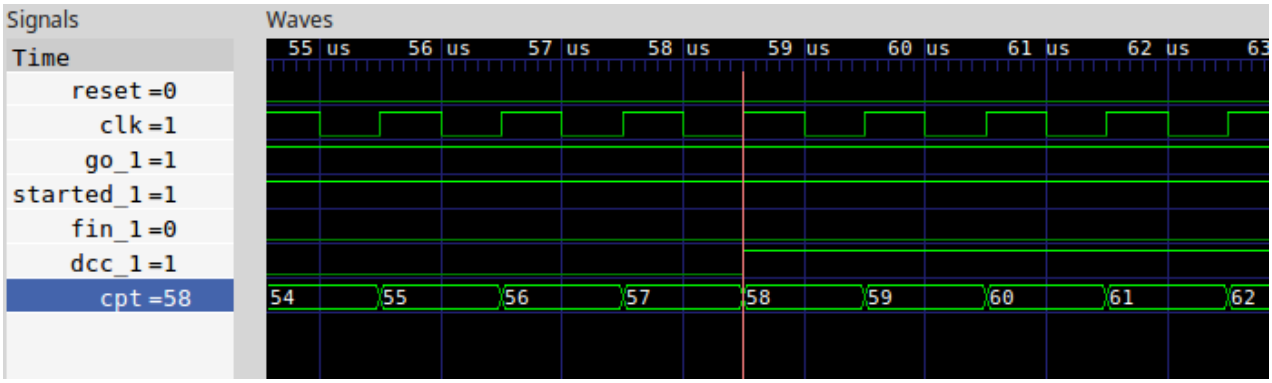


Figure 16 : Simulation du module BIT1, passage à l'état GEN_HIGH

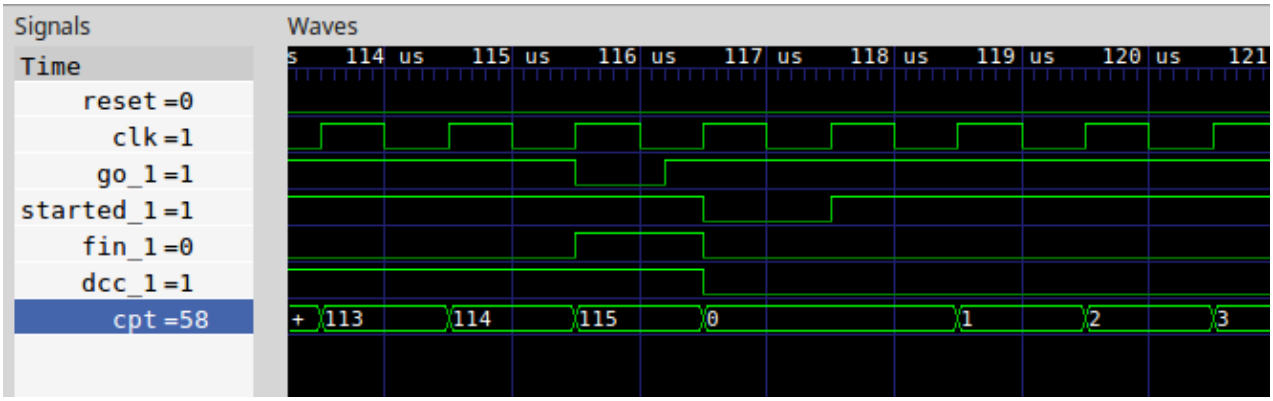


Figure 17 : Simulation du module BIT1, fin de génération

IV).5- Le Générateur de Trame:

Ce module est utilisé pour construire et valider la trame, il est constitué :

a)- D'une partie combinatoire qui assemble dans un signal de 42 bits : le préambule , l'octet de l'adresse, L'octet de commande et l'octet de contrôle , les bits start entre les champs, et le stop bit mis à la fin de la trame.

b)-D'une partie séquentielle qui s'agit d'une mémoire pour enregistrer la trame générée par la partie combinatoire (Le signal send est utilisé à ce propos)

Nous avons décrit ce comportement en VHDL, puis on l'a simulé en écrivant un testbench approprié :

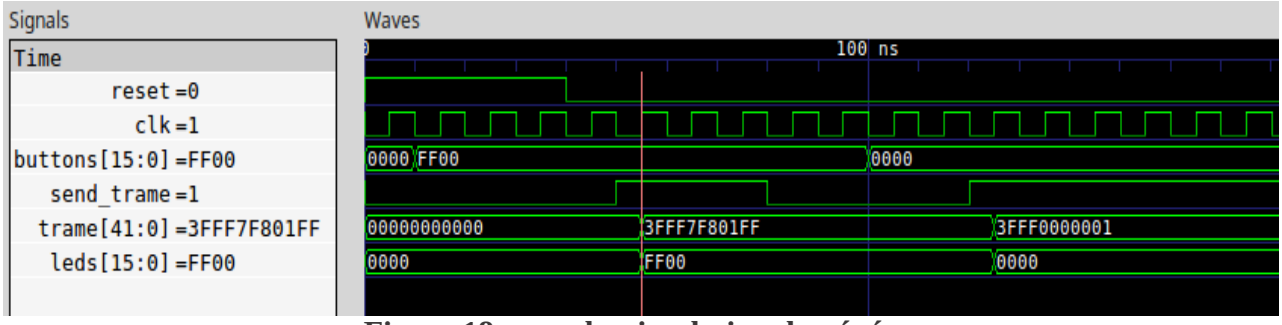


Figure 18 : graphe simulation du générateur.

Ce module a été synthétisé correctement par les outils de vivado

```
---------------------------------------------------------------------------------
Synthesis finished with 0 errors, 0 critical warnings and 1 warnings.
Synthesis Optimization Runtime : Time (s): cpu = 00:00:23 ; elapsed = 00:00:31
Synthesis Optimization Complete : Time (s): cpu = 00:00:41 ; elapsed = 00:00:4
INFO: [Project 1-571] Translating synthesized netlist
INFO: [Netlist 29-17] Analyzing 19 Unisim elements for replacement
INFO: [Netlist 29-28] Unisim Transformation completed in 0 CPU seconds
INFO: [Project 1-570] Preparing netlist for logic optimization
INFO: [Opt 31-138] Pushed 0 inverter(s) to 0 load pin(s).
INFO: [Project 1-111] Unisim Transformation Summary:
```

Figure 19 : Résumé du rapport de synthèse

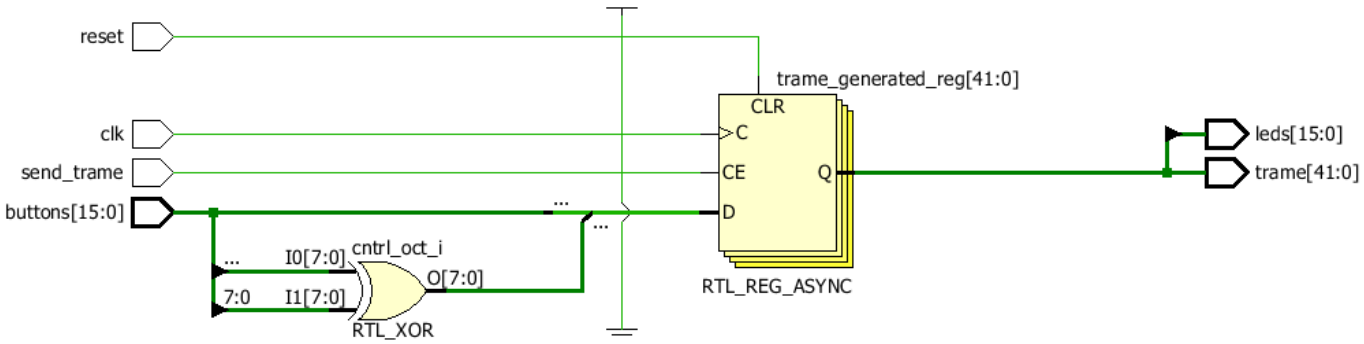


Figure 20 : Schéma RTL du Module GEN_TRAME

IV).6- Le Registre DCC:

Ce module reçoit des trames (42 bits) construites par le générateur de trame, et validés par l'utilisateur.

Il fournit toujours le MSB de la trame pour la machine à état, il accepte des commandes de la machine à état pour effectuer des décalages de la trame.

Ce module est construit deux blocs de base :

1). Un registre (42 bits) : sert à enregistrer soit :

- Une trame décalée d'un bit à gauche.
- La trame courante si pas de décalage.
- Une nouvelle trame reçue du générateur de trame
- Une trame nulle à la réinitialisation

2). Une machines à états : qui sert gérer les communications avec la machine à états principale, en analysant correctement les commandes reçues et générer des signaux :

- pour la commande du registre.
- Réponse à la machine à état principale.

1). Fonctionnement du registre :

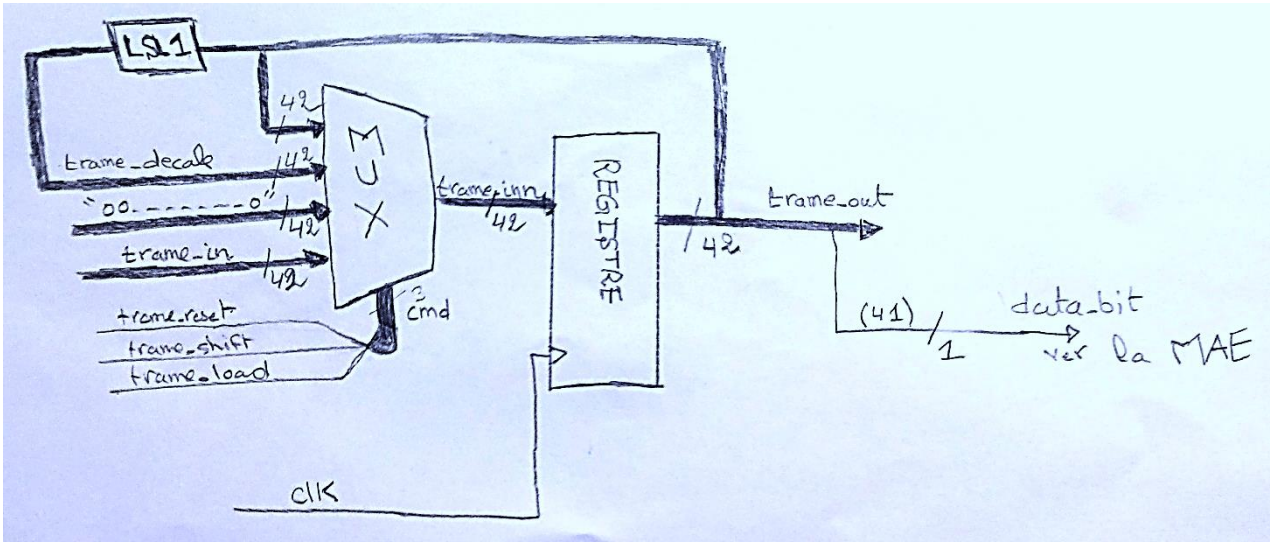


Figure 21 : graphe architecture détaillé du registre

Un multiplexeur est utilisé pour sélectionner la trame à enregistrer dans le registre.
Il est commandé par trois signaux générés par la machine à états qui sont :
- trame_reset : pour enregistrer une trame nulle.
- trame_shift : pour enregistrer une version décalée de la trame.
- trame_load : pour enregistrer une nouvelle trame à charger.

Si aucun de ses signaux n'est actif, la trame courant sera enregistrée

1). La machine à états :
Ses entrées sont :

clk : horloge 100Mhz
reset : réinitialisation émit par le module MAE
load : émit par la mae pour demander le chargement d'une nouvelle trame
shift : reçu pour effectuer un décalage
read : fin de lecture émit par la mae
trame_in : trame générée par le module générateur de trame à travers un bus de 42 bits

Ses sorties sont :

emptyy : pour indiquer que le registre est vide.
decale : envoyé à la mae pour indiquer que le décalage demandé est éffectué.
data_bit : MSB de la trame passé à la machine à états.

Cette machine a 6 états qui sont :

REG_EMPTY : état initial ou le registre est vide.
REG_LOAD : réservé pour charger une nouvelle trame dans le registre.
LOAD_END : pour indiquer à la machine à état la fin de chargement
REG_SHIFT : réservé pour effectuer un décalage.
REG_SHIFTED : pour indiquer à la machine la fin de décalage.
REG_END : fin, état pour attente d'une nouvelle commande de décalge.

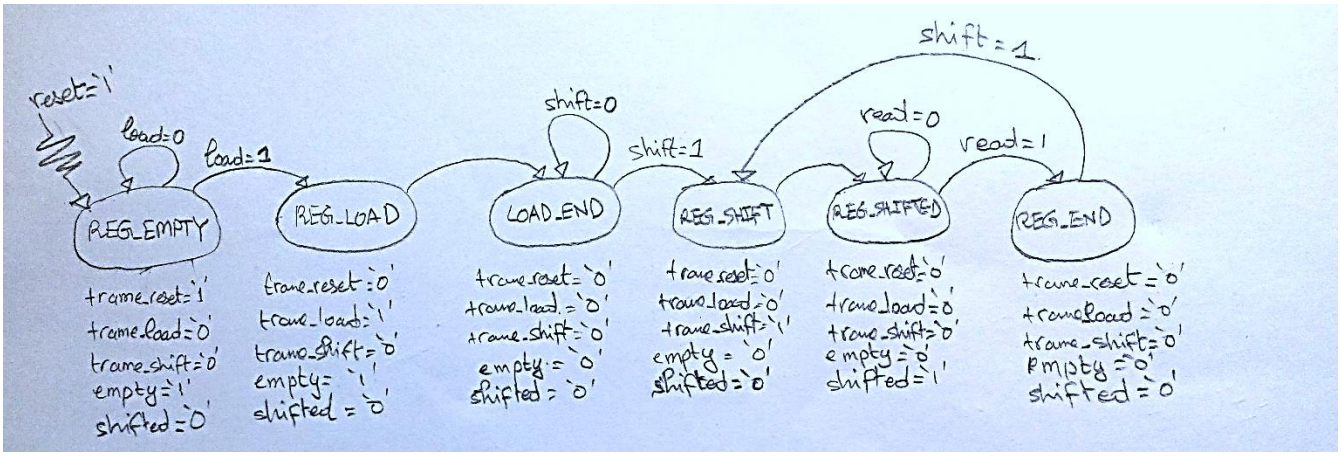


Figure 22 : Diagramme de la machine à état du module registre DCC

Nous avons décrit ce module en VHDL, puis on l'a simulé en écrivant un Test bench approprié

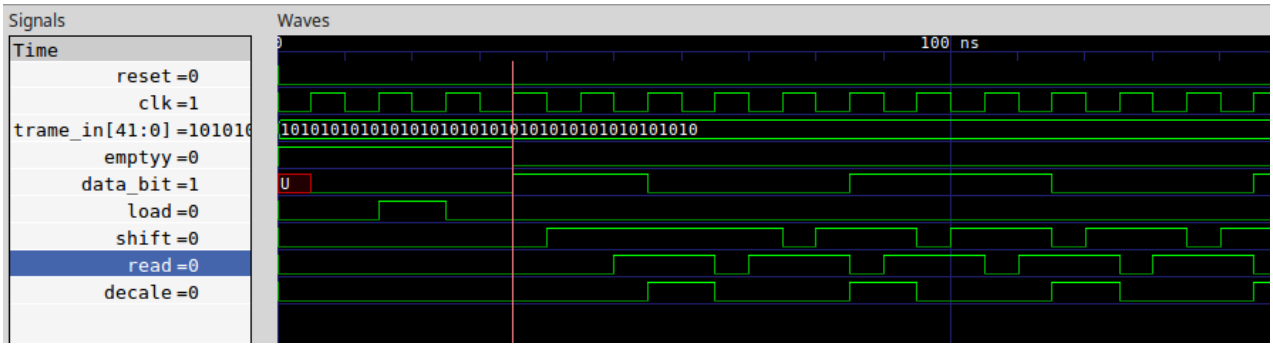


Figure 23 : graphe simulation du module registre dcc

Pendant ce test, l'entrée du module s'agit d'une trame construire s'une alternation des zéros et des uns.

Les fonctionnalités du chargement et décalage ont été bien effectuées, et les signaux de sorties attendus ont étés correctement générée.

La synthèse s'est déroulée correctement par les outils de Vivado.

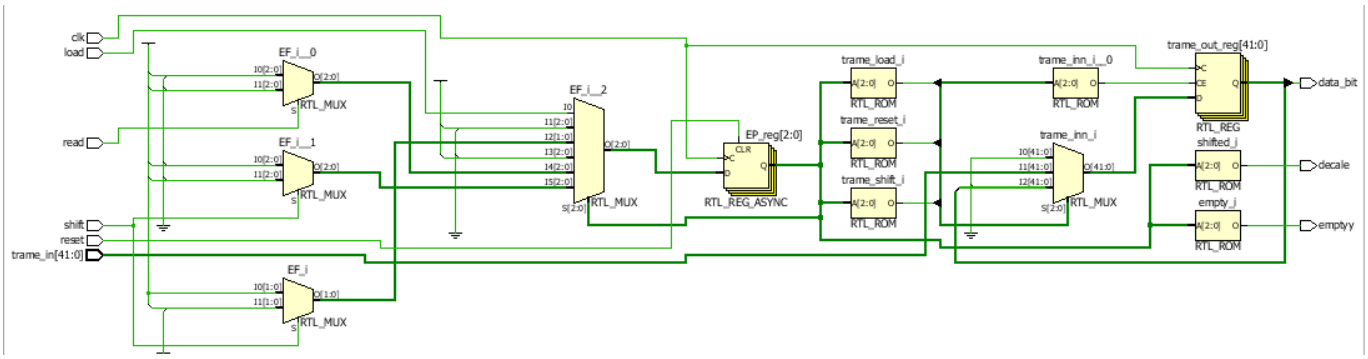


Figure 24 : Schéma RTL du Module Registre DCC

IV).7- La Machine à états principale:

Elle est responsable d'assurer le bon séquencement de toutes les phases nécessaires à la transmission d'une trame.

Elle lit et analyse les signaux envoyés par tous les modules, et génère les signaux de commande appropriés.

Un bloc pour le comptage est mis en place qui s'agit d'une machine à état pour incrémenter le compteur après chaque envoie d'un bit de la trame.

Les entrées émises par le registre dcc :

emptyFreg : quand le registre est vide
bitFreg : le MSB de la trame à transmettre
shiftedFreg : reçu après chaque décalage effectué

Les entrées émises par le module BIT0 :

startedFzero : reçu si le module BIT0 a débuté la génération
endFzero : reçu à la fin de génération

Les entrées émises par le module BIT1 :

startedFzone : reçu si le module BIT1 a débuté la génération
endFone : reçu à la fin de génération

Les entrées émises par le temporisateur :

endFtempo : reçu à la fin de chaque temporisation

Ses sorties sont :

rst_sys : utilisé pour la réinitialisation tous les modules au démarages

Les commandes du module reg :

load2reg : demander un chargement d'une nouvelle trame
shift2reg : effectuer un décalage
read2reg : communiquer la fin de lecture du MSB

Les commandes du module BIT1 et BIT0 :

enable2one : activer le module BIT1 pour effectuer une transmission d'un 1
enable2zero : activer le module BIT0 pour effectuer une transmission d'un 0

Les commandes du module temporisateur

enable2tempo : commander une nouvelle temporisation de 6ms

Cette machine a 9 états qui sont :

STRATN	:	état initial : réinitialisation de tous les modules.
LOAD_TRAME	:	réservé pour charger une nouvelle trame.
READING	:	réservé pour la lecture d'un bit de la trame
SENDING_ZERO	:	réservé pour la transmission d'un bit 0.
SENDING_ONE	:	réservé pour la transmission d'un bit 1.
COUNT_INC	:	réservé pour incrémenter le compteur.
COUNT_READ	:	réservé pour la lecture de la nouvelle valeur du compteur.
CHECK_EMPTY	:	pour vérifier la valeur du compteur
SHIFTING	:	réservé pour effectuer un décalage.

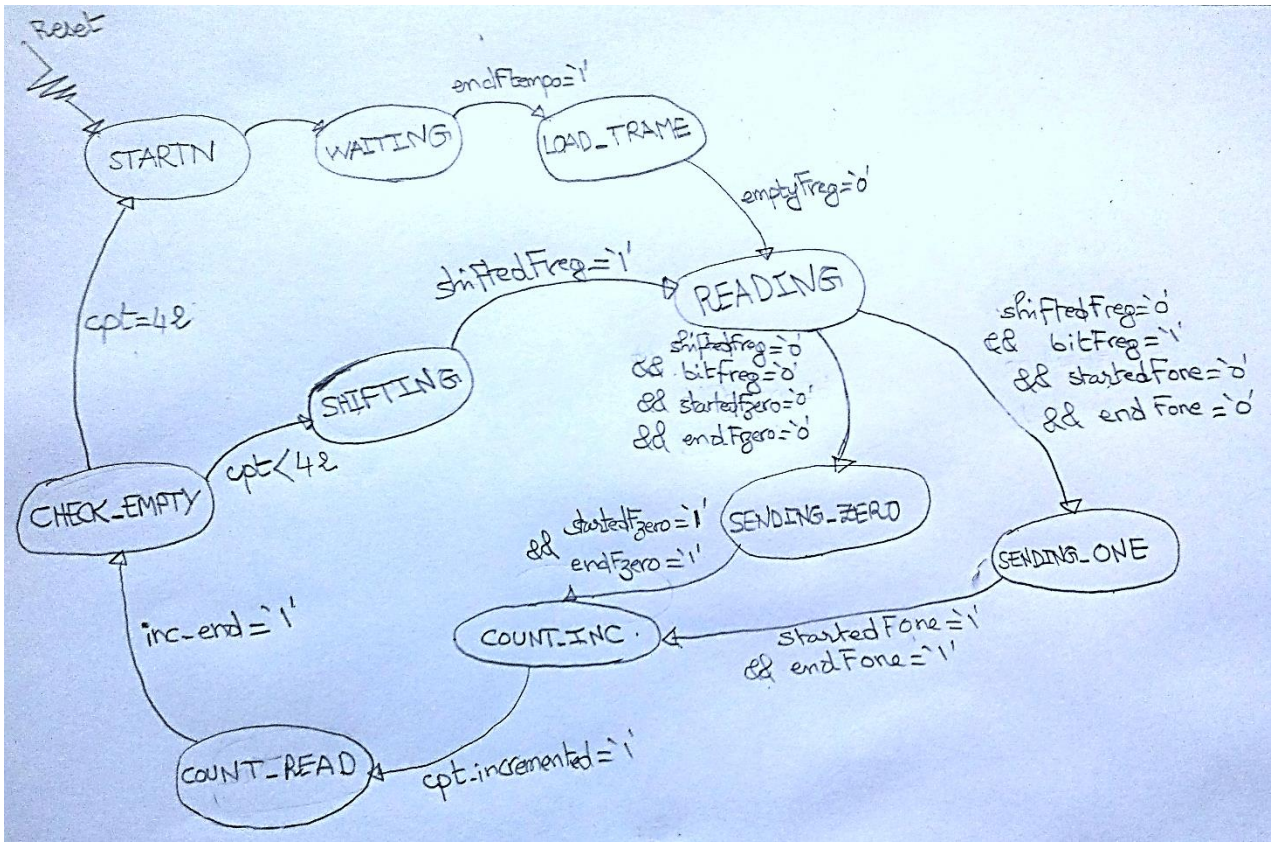


Figure 25 : Diagramme de la machine à états pricipale.

Les signaux d'entrées pour la machine à état de l'incrémenteur:

cpt_inc	:	émit par la machine à état pour effectuer l'incrémentation
cpt_read	:	émit par la machine à état pour indiquer la lecture de la nouvelle valeur

Ses sorties sont :

cpt_incremented	:	envoyé à la mae pour indiquer que le compteur a été incrémenté
inc_end	:	envoyé à la machine à état pour indiquer la fin du cycle d'incrémentation

Cette machine a 4 états qui sont :

IDLE	:	état initial.
INCREMENT	:	réservé pour effectuer l'incrémentation.
INCREMENTED	:	réservé pour indiquer la validité du compteur pour la lecture
END_INC	:	état fin du cycle d'incrémentation, attente

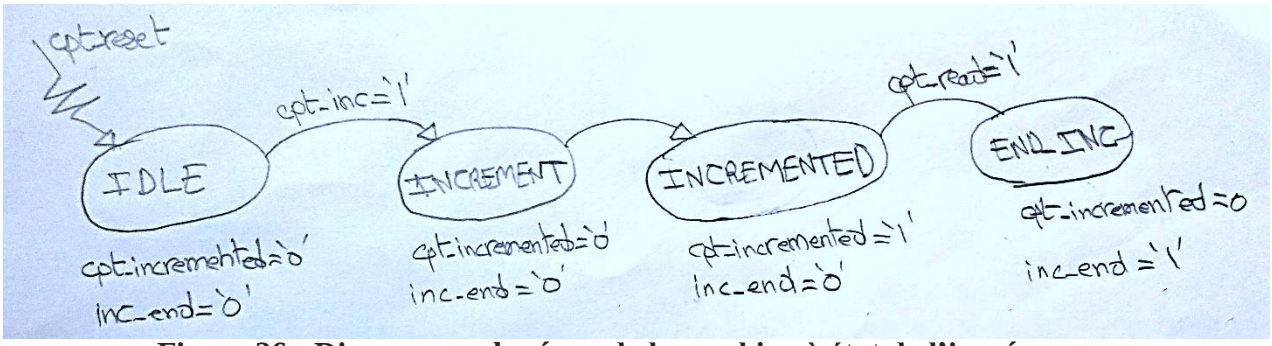


Figure 26 : Diagramme des états de la machine à état de l'incrémenteur.

Nous avons décrit ce module en VHDL, puis on l'a simulé en écrivant un Test bench approprié

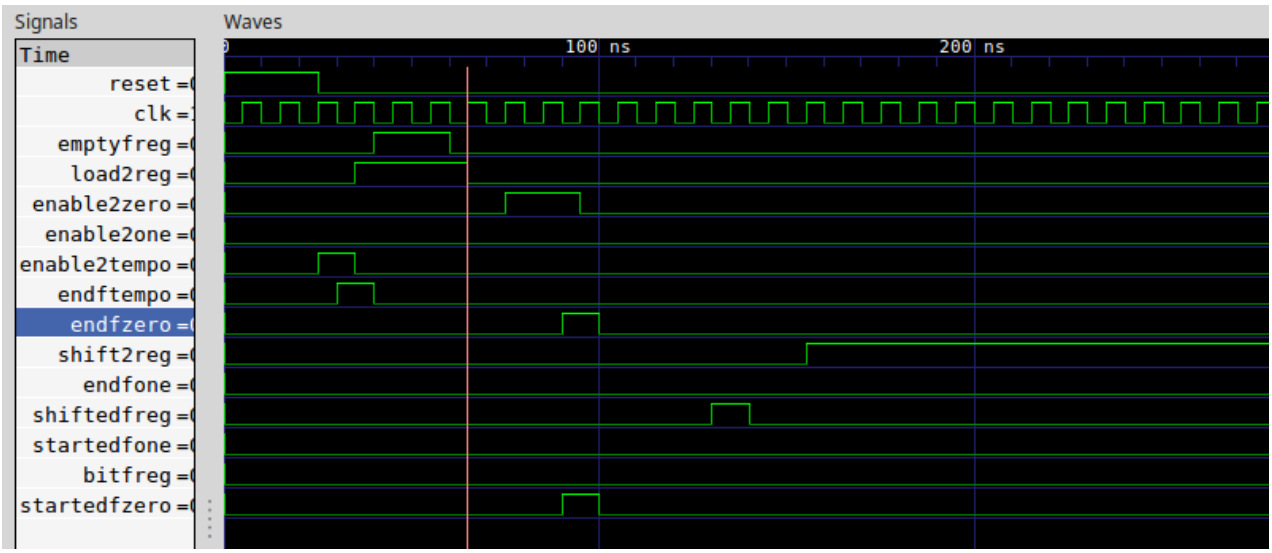


Figure 27 : Simulation du module machine à états principale.

La synthèse s'est déroulée correctement par les outils de Vivado.

IV).7- Construction de l'IP centrale DCC:

Après la validation de tous les modules de notre architecture, nous avons écrit en VHDL un modèle structurel qui construit notre IP.

La figure suivante montre l'architecture de notre IP et les interconnexions entre tous les modules :

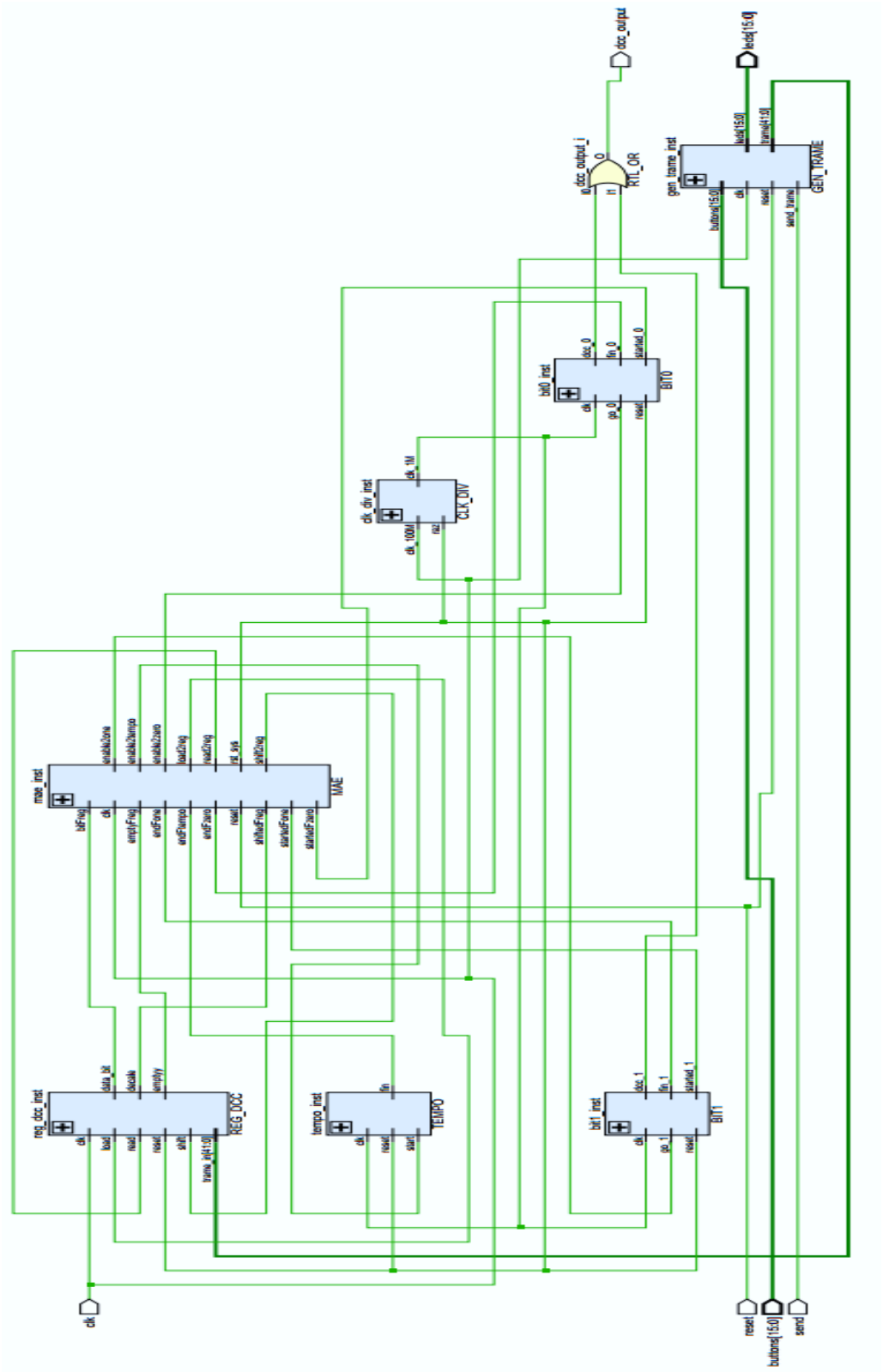


Figure 28 : Schéma RTL de l'IP généré par Vivado

Puis en écrivant un testbench , on a obtenu des trames périodiques espacées de 6 mS à la sortie de l'IP.

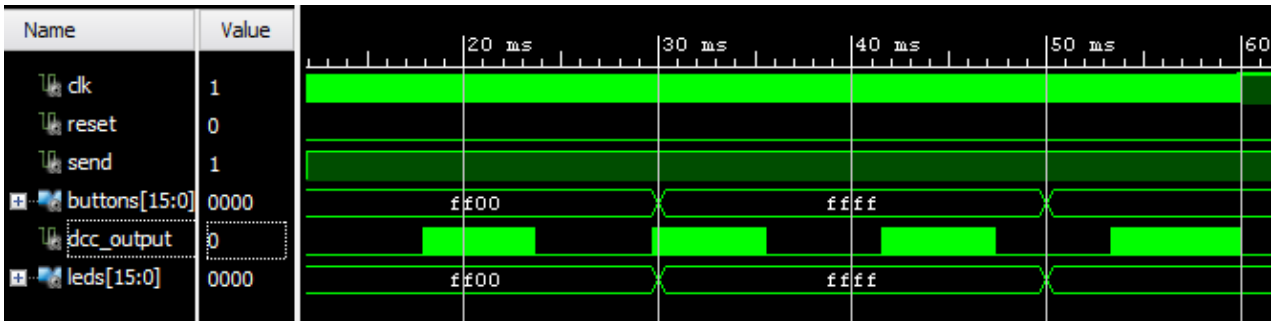


Figure 29 : Génération de trames périodiques espacées de 6 mS

Pour vérifier la composition des trames correctement :

Prenant dans un premier cas :

Addresse = 11111111
commande = 00000000
Contrôle = 11111111

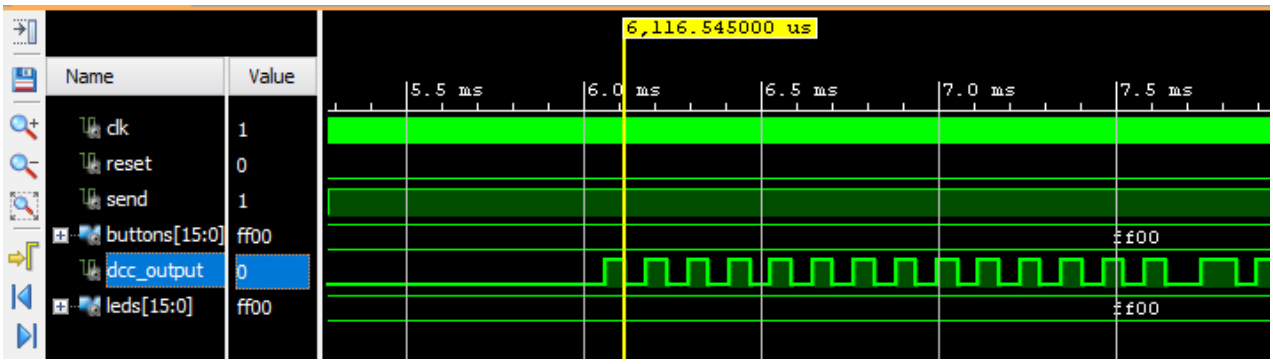


Figure 30 : Simulation de l'IP, cas 1 : préambule + Start bit

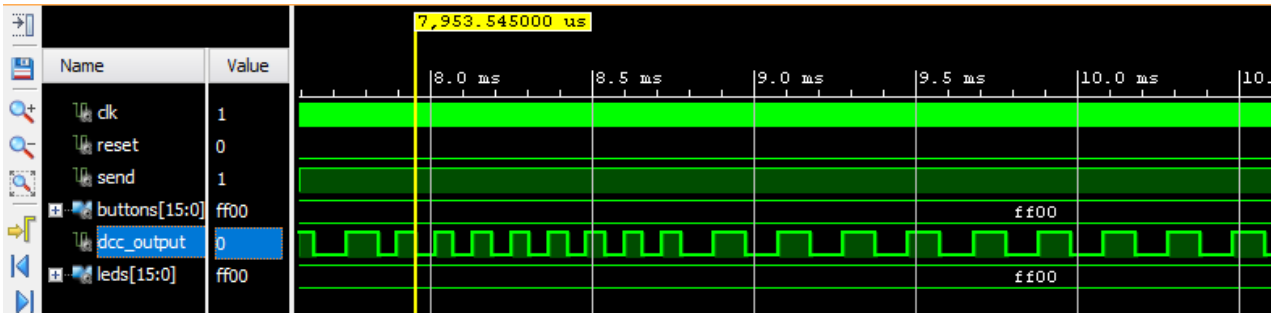


Figure 31 : : Simulation de l'IP, cas 1 : adresse + Start bit + commande

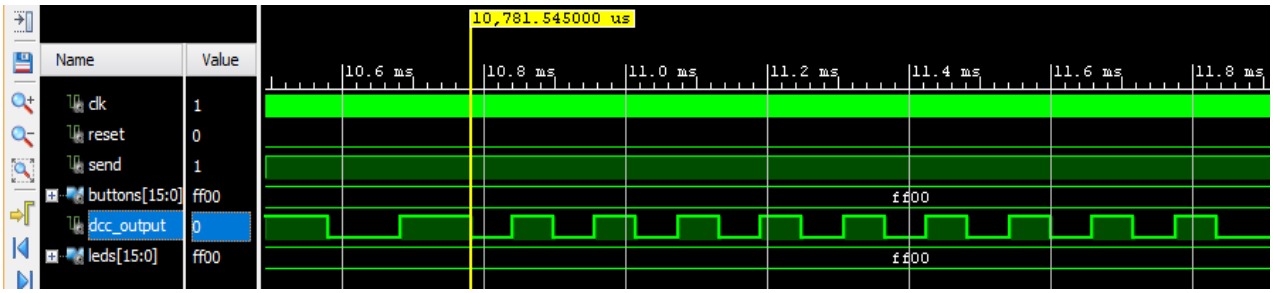


Figure 32 : : Simulation de l'IP, cas 1 : Start bit + contrôle+ stop bit

Deuxième cas :

Adresse = 11111111
Commande = 11111111
Contrôle = 00000000

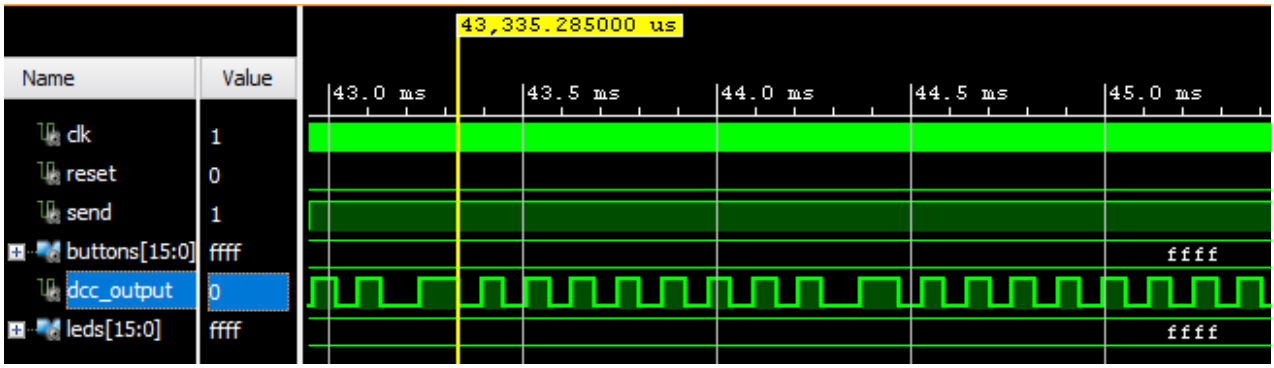


Figure 33 : Simulation de l'IP, cas 2 : Start bit + adresse+ start bit + commande

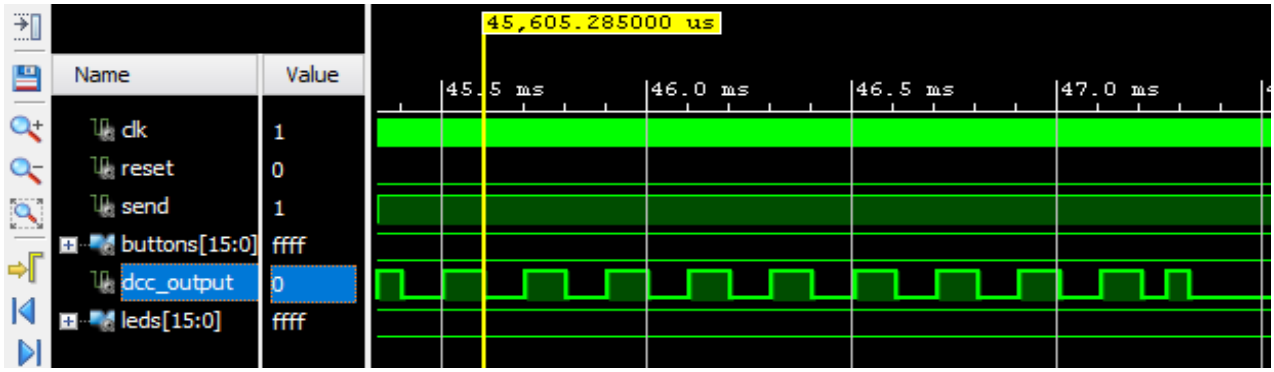


Figure 34 : Simulation de l'IP, cas 2 : Start bit + contrôle+ stop bit

V)- Test et validation de la centrale DCC :

L'architecture présentée précédemment est l'architecture finale obtenue après plusieurs modifications apportées pour que la synthèse soit correctement effectuée sur vivado.

Nous avons testé cette architecture sur une carte Nexus DDR4, en vérifiant à l'aide d'un oscilloscope que le signal de sortie est présent et qu'il correspond parfaitement aux trames construites par l'état des interrupteurs.

Tous les tests étaient concluants, nous avons ainsi testé et validé notre IP sur la plateforme, la commande des trains marchait très bien.

VI)- Intégration de l'IP centrale DCC sur microblaze :

VI).1- Création du Package de L'IP :

Nous avons utilisé le projet codesign matériel et logiciel déjà crée au deuxième tp, ou un microprocesseur microblaze a été ajouté avec deux blocs de GPIO.

L'intégration de notre IP sur ce système, nécessite la création d'un nouveau bloc qui sera visible pour un ajout sur microblaze.

Ainsi , on a suivi les étapes présentées dans le troisième tp :

Après la création d'un nouveau package :

Dans le fichier dcc_ip_centrale_v1_0.vhd :

1). Ajout des ports de sorties :

```
port (
    -- Users to add ports here
       dcc_output : out std_logic;
       leds : out std_logic_vector(15 downto 0);
    -- User ports ends
```

2). Ajout des ports de sorties dans la déclaration du composant :

```
-- component declaration
component dcc_ip_centrale_v1_0_S00_AXI is
    generic (
    C_S_AXI_DATA_WIDTH  : integer   := 32;
    C_S_AXI_ADDR_WIDTH  : integer   := 4
    );
    port (
    dcc_output      : out std_logic;
    leds            : out std_logic_vector(15 downto 0);
    S_AXI_ACLK      : in std_logic;
```

3). Ajout des instantiations sur le port map :

```
port map (
    dcc_output => dcc_output,
    leds => leds,
    S_AXI_ACLK  => s00_axi_aclk,
```

Dans le fichier dcc_ip_centrale_v1_0_S00_AXI.vhd :

1). Ajout des ports de sorties :

```
port (
    -- Users to add ports here
       dcc_output : out std_logic;
       leds : out std_logic_vector(15 downto 0);
    -- User ports ends
```

2). Ajout des signaux pour les entrées de l'ip:

```
    -- AXI4LITE signals
    signal clk          : std_logic;
    signal reset        : std_logic;
    signal send         : std_logic;
    signal buttons      : std_logic_vector( 15 downto 0);
    signal axi_awaddr   : std_logic_vector(C_S_AXI_ADDR_WIDTH-1 downto 0);
```

3). Instanciation du composant:

```
    -- Add user logic here
L0: entity work.dcc port map (clk => clk, reset => reset, send => send, dcc_output => dcc_output, buttons => buttons);
clk   <= S_AXI_ACLK;
reset <= slv_reg0(0);
send  <= slv_reg0(1);
buttons <= slv_reg1(15 downto 0);
```

Les boutons reset et send seront donc connectés aux deux bits de poids faible du registre **slv_reg0**
Les interrupteurs seront connectés aux 15 bits de poids faible registre **slv_reg1**
L'horloge de l'IP est connectée à l'horloge interne de la carte 100 Mhz

VI).2- Intégration sur microblaze :

On a suivi les étapes suivantes :

1). Modification de la taille du GPIO **bouttons** à 2 bits (reset et send).

2). Modification du GPIO **led_switch** en un nouveau avec le nom **sw,** qui gardera qu'un seul port d'entrée pour récupérer la valeur des interrupteurs, sa taille est de 14 bits car les deux bits de poids fort sont réservé à la réinitialisation du microblaze, et ne doivent pas affecter la construction de la trame (seront remplacé par ''00'' lors de l'écriture dans le registre **slv_reg0**), et donc le nouveau champs d'adresse est sur 6 bits, la taille du champs de commande reste inchangée.

3). Ajout de l'IP centrale DCC et de deux connecteurs pour ses sortie
4). Connection de tous le système.

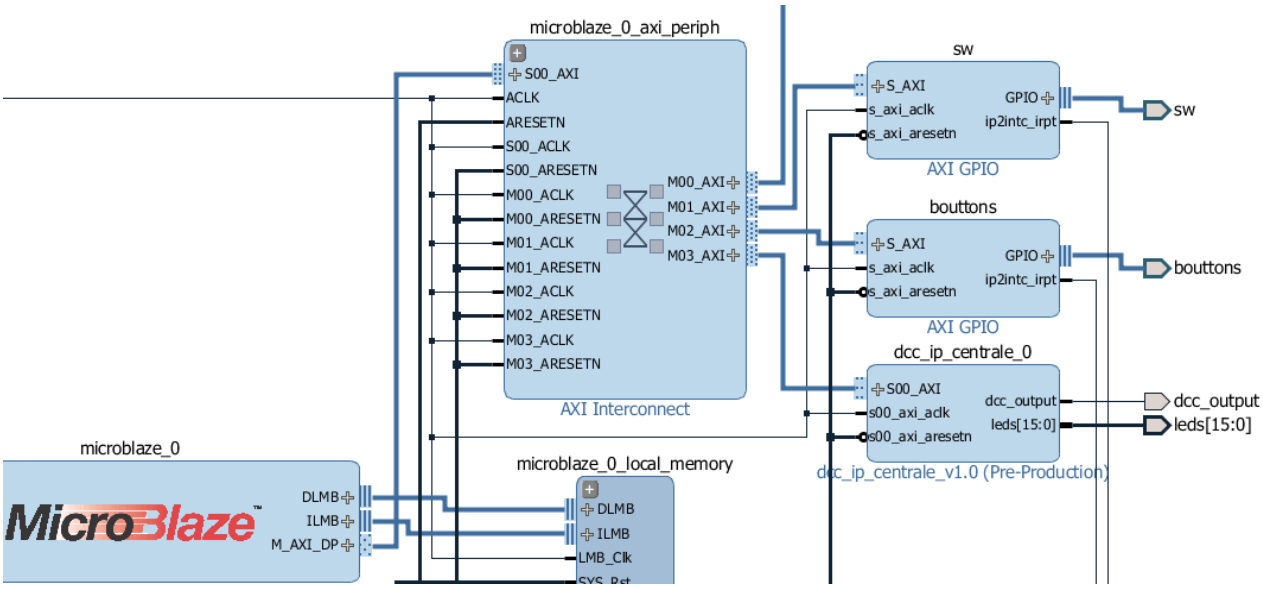


Figure 35 : schéma bloc après intégration de l'IP centrale DCC

VI).3- Création de l'application sur SDK :

Après avoir adapté le fichier de contrainte XDC, et le suivi des étapes pour exporter notre système.

On a créé une application sur SDK en utilisant la fonction **`MY_LED_mWriteReg()`** qui sert à écrire dans les registres de l'IP.

```
/* Definitions for peripheral DCC_IP_CENTRALE_0 */
#define XPAR_DCC_IP_CENTRALE_0_DEVICE_ID 0
#define XPAR_DCC_IP_CENTRALE_0_S00_AXI_BASEADDR 0x44A00000
#define XPAR_DCC_IP_CENTRALE_0_S00_AXI_HIGHADDR 0x44A0FFFF
```

Les décalages pour atteindre les registres de l'IP, sont définis dans le fichier dcc_ip_centrale.h:

```
#define DCC_IP_CENTRALE_S00_AXI_SLV_REG0_OFFSET 0
#define DCC_IP_CENTRALE_S00_AXI_SLV_REG1_OFFSET 4
#define DCC_IP_CENTRALE_S00_AXI_SLV_REG2_OFFSET 8
#define DCC_IP_CENTRALE_S00_AXI_SLV_REG3_OFFSET 12
```

```c
#include "xgpio.h"
#include "xparameters.h"
#include "dcc_ip_centrale.h"

int main ()
{

XGpio bouttons;
XGpio sw;

int sw_val;         //valeur des 14 interrupteurs
int bouttons_val;  //valeur des deux bouttons reset et send de 0 (inactifs) à 3 (les deux sont actifs)

XGpio_Initialize(&sw, XPAR_SW_DEVICE_ID); //initialisation du Gpio Sw
XGpio_SetDataDirection(&sw,1 ,1);       //configuration comme une entrée

XGpio_Initialize(&bouttons, XPAR_BOUTTONS_DEVICE_ID); //initialisation du Gpio bouttons
XGpio_SetDataDirection(&bouttons,1 ,1);                //configuration comme une entrée

while (1){

 sw_val = XGpio_DiscreteRead(&sw,1);                //lecture de la valeur des 14 interrupteurs
 bouttons_val=XGpio_DiscreteRead(&bouttons,1);     // lecture de la valeur des deux bouttons reset et send

  //ecriture de la valeur des bouttons dans le registre0
 //reset<=slv_reg0(0);
 //reset<=slv_reg0(1);
  if (bouttons_val < 4){
    MY_LED_mWriteReg(XPAR_DCC_IP_CENTRALE_0_S00_AXI_BASEADDR, DCC_IP_CENTRALE_S00_AXI_SLV_REG0_OFFSET,bouttons_val);
  }
  else{
    MY_LED_mWriteReg(XPAR_DCC_IP_CENTRALE_0_S00_AXI_BASEADDR, DCC_IP_CENTRALE_S00_AXI_SLV_REG0_OFFSET,0);
  }
   //ecriture de la valeur des interrupteurs dans le registre 1
   // buttons<=slv_reg1(15 downto 0);
    MY_LED_mWriteReg(XPAR_DCC_IP_CENTRALE_0_S00_AXI_BASEADDR, DCC_IP_CENTRALE_S00_AXI_SLV_REG1_OFFSET,sw_val);
 }
return 0;
}
```

Figure 36 : Application écrite sur SDK

VII)- Conclusion:

En participant à ce projet, nous avons pu approfondir nos connaissances en architecture des systèmes numériques ainsi en conception et codage des machines à états de Moore.

Il nous a également permis d'observer de prêt qu'un code peut être simulé correctement mais peut ne pas être synthétisable.

Enfin, nous avons pu nous familiariser avec une plateforme FPGA complète.