

Département Electronique et Technologie Numérique

Conception d'un contrôleur I2C

Conception de SOC

LAGHA Chaima

Table des matières

1. Cahier des charges	6
1.1 Objectif du circuit	6
1.2 Les fonctionnalités attendues	6
1.3 Utilisation du circuit	6
1.4 Chronogrammes caractéristiques	8
1.4.1 Chronogrammes des cycle d'échange avec le processeur	8
1.4.2 Chronogrammes des cycle d'échange avec un périphérique externe	9
1.5 Contraintes du projet	10
1.5.1 Contraintes technologiques	10
1.5.2 Contraintes temporelles	11
2. Spécifications	11
2.1 caractérisation de l'environnement	11
2.1.1 Délimitation des entités	11
2.1.2 Comportement des entités	11
2.1.3 Délimitation des entrées sorties du système:	12
2.2 Spécifications fonctionnelles	13
2.2.1 Comportement du contrôleur I2C	13
2.2.2 Diagramme d'activité	17
2.2.3 Spécifications des registres	17
2.2.4 Ecriture des procédures de base	19
2.3 Spécifications opératoires	23
2.4 Spécifications technologiques	23
3. Conception	24
3.1 Délimitation fonctionnelle	24
3.2 Première décomposition fonctionnelle	24
3.3 Raffinement	25
3.4 Ecriture des algorithmes	25
3.4.1. Algorithme de la fonction GestionHorloge	26
3.4.1 Algorithme de la fonction Emission_Reception	26
3.5 Introduction des interfaces	30

3.5.1 Raffinement de la fonction Interface	31
3.5.2 Algorithme de la fonction Write	32
3.5.3 Algorithme de la fonction Read	32
3.6 Description de la solution pour les tests	33
4. Saisie de la solution sous HDL Designer	34
4.1 Le composant de test associer au contrôleur I2C	35
4.2 Le contrôleur I2C	36
4.2 Interface pour le processeur	37
5. Résultat de simulation	38
5.1 Validation du Reset asynchrone	38
5.2 Validation de la fonction ClockGenerator	39
5.3 Validation de l'envoi des données par le contrôleur I2C	40
5.3.1 Configuration du contrôleur I2C pour l'émission de données	41
5.3.2 Validation du bit Start	42
5.3.3 La transmission de l'adresse	43
5.3.4 La transmission du bit RnW	44
5.3.5 Le bit d'accusé de réception de l'adresse	44
5.3.6 L'envoi de données	45
5.3.7 Validation du bit Stop	48
5.3.8 Lecture de l'état du contrôleur I2C	49
5.4 Validation de la réception des données par le contrôleur I2C	50
5.4.1 Configuration du contrôleur I2C pour la réception de données	50
5.4.2 Validation du bit Start	51
5.4.3 La transmission de l'adresse	52
5.4.4 La transmission du bit RnW	53
5.4.5 Le bit d'accusé de réception de l'adresse	53
5.4.6 La réception de données	54
5.4.7 Validation du bit Stop	56
5.5 Validation de cas d'erreur	57
6. Résultat de synthèse	58
6.1 Codage de la machine à état	58
6.2 Résultat des ressources utilisées	59
6.3 Analyse de synchronisation statique	60

6.4	Résultats graphiques de la synthèse	63
6.4.1	Schéma RTL de l'IP	63
7.	Conclusion	67
A.	Annexes	68
A.1	Code VHDL du package utilisé pour le projet	68
A.2	Code VHDL de la fonction Write de l'interface processeur	70
A.3	Code VHDL de la fonction Read de l'interface processeur	72
A.4	Code VHDL de la fonction ClockGenerator	74
A.5	Code VHDL de la fonction Emission_Reception	75
A.6	Code VHDL de l'environnement de test	80
A.7	Schéma RTL de la fonction ClockGenerator	87
A.8	Schéma RTL de la fonction Emission_Reception	88

Table des figures

Figure 1: Architecture classique du SOC.....	6
Figure 2: Schéma de câblage du contrôleur	7
Figure 3: Cycles de lecture et écriture du processeur	8
Figure 4:Format d'une trame I2C	9
Figure 5:Cycle de lecture de données par le contrôleur I2C	10
Figure 6:Comportement du système à µP vis à vis le contrôleur I2C	11
Figure 7:comportement de l'esclave I2C vis à vis le contrôleur I2C.....	12
Figure 8:Délimitation des E/S du contrôleur I2C	12
Figure 9:Comportement du contrôleur I2C.....	14
Figure 10:Raffinement de l'état Sending Start bit	15
Figure 11:Raffinement de l'état Sending Stop bit.....	15
Figure 12:Raffinement des état d'envoi de données	16
Figure 13:Raffinement des état de réception de données.....	16
Figure 14:Diagramme d'activité du contrôleur I2C	17
Figure 15:Délimitation des entrées sorties fonctionnelles	24
Figure 16:Première décomposition fonctionnelle du circuit.....	24
Figure 17:Raffinement de la première décomposition fonctionnelle de l'IP	25
Figure 18:Introduction d'interface	30
Figure 19:Raffinement de la fonction Interface	31
Figure 20: Schéma bloc du contrôleur I2C associé à l'environnement de test.....	35
Figure 21: Schéma bloc du contrôleur I2C sous HDL Designer	36
Figure 22: Schéma bloc de l'interface pour le processeur.....	37
Figure 23: Reset asynchrone du processeur	38
Figure 24: ClockGenerator en mode de transmission standard $T_{sck}=10000\text{ns}$	39
Figure 25: ClockGenerator en mode de transmission rapide $T_{sck}=1000\text{ns}$	39
Figure 26: Une trame I2C : Envoi de données par le contrôleur I2C.....	40
Figure 27: Ecriture par le processeur dans le registre I2C_Config	42
Figure 28: Le bit de démarrage d'une trame I2C	43
Figure 29: Envoi d'adresse du périphérique de destination.....	43
Figure 30: Envoi du bit RnW	44
Figure 31: Attente du bit d'accusé de réception d'adresse.....	45
Figure 32: Envoi du premier octet	46
Figure 33: Envoi du deuxième octet	47
Figure 34: Envoi du dernier octet	48
Figure 35: Le bit d'arrêt de la trame I2C	49
Figure 36: Lecture de l'état de l'IP par le processeur	49
Figure 37: Une trame I2C reçu par le contrôleur I2C	50
Figure 38 : Configuration du contrôleur I2C pour recevoir les données	51
Figure 39: Bit Start de la trame I2C pour recevoir les données	52
Figure 40: Emission de l'adresse du périphérique émetteur.....	52
Figure 41: Emission du bit RnW	53
Figure 42:Réception du bit d'accusé de réception d'adresse	54

Figure 43: Réception du deuxième octet par l'IP	56
Figure 44: Envoi du bit Stop à la fin de la réception de données.....	56
Figure 45: Erreur d'écriture : non-réception du bit d'acquittement	57
Figure 46 : Codage de la machine à états par Precision Synthesis	58
Figure 47: Résultats des ressources utilisées dans le FPGA Xilinx 7Z010CLG400	59
Figure 48: Timing Violation Report	60
Figure 49: 1 ^{er} chemin critique (Timing Report).....	61
Figure 50: 2 ^{ème} chemin critique (Timing Report).....	62
Figure 51: RTL Schématic du contrôleur I2C	63
Figure 52:RTL Schématic de l'Interface du processeur	64
Figure 53: Schéma RTL de la fonction Write.....	65
Figure 54: Schéma RTL de la fonction Read.....	66

1. Cahier des charges

1.1 Objectif du circuit

Le contrôleur I2C a pour rôle d'assurer le couplage du microprocesseur avec des modules extérieurs via une liaison I2C.

1.2 Les fonctionnalités attendues

- Le circuit est configuré par le microprocesseur, les paramètres à configurer sont:
 - La génération des interruptions à l'émission ou/et à la réception.
 - La vitesse de transmission (Mode standard 100 kbit/s, Mode rapide 4000 kbit/s)
 - Le mode de communication: Lecture ou écriture
 - Le nombre de données à transmettre.
- Le circuit assure les échanges de données en émission et en réception.
- Un bit d'acquittement doit être envoyé après chaque octet pour indiquer que l'octet a été reçu avec succès et qu'un autre octet peut être envoyé (registre d'état).
- Le circuit peut générer des interruptions à l'issue d'une émission ou d'une réception, ce selon la configuration faite par le microprocesseur.

1.3 Utilisation du circuit

La figure 1.1 présente l'organisation interne du système sur puce final. Ce document est consacré à la conception du contrôleur I2C.

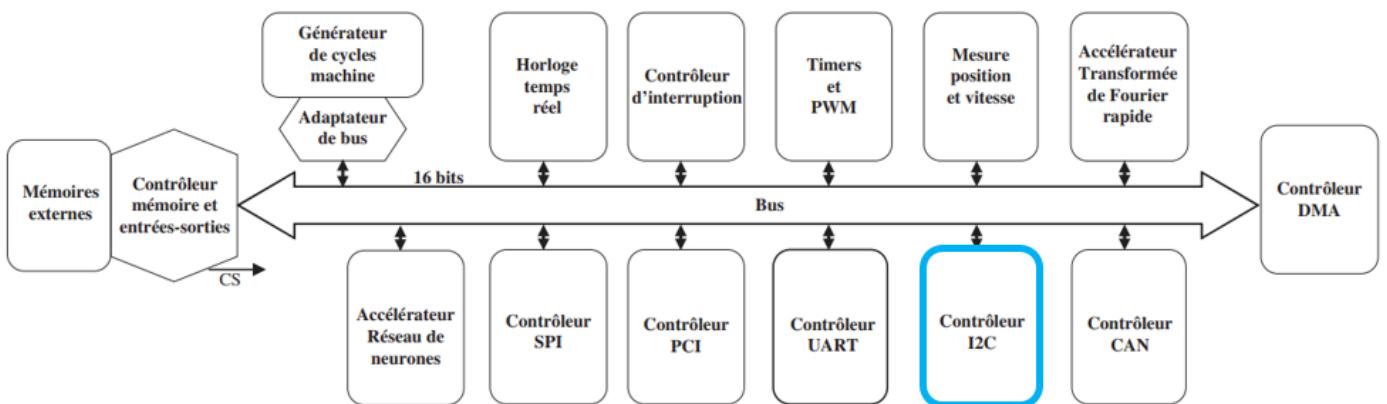


Figure 1: Architecture classique du SOC

Le schéma de câblage détaillé du circuit à concevoir est donné sur la figure ci-après.

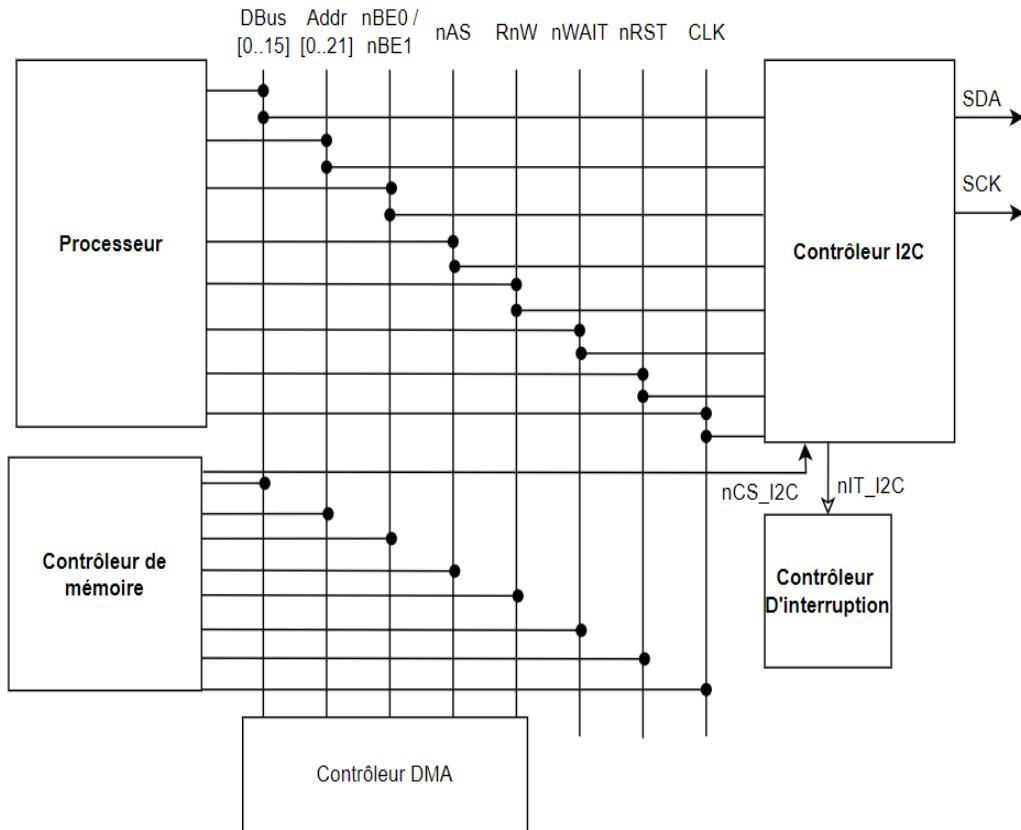


Figure 2: Schéma de câblage du contrôleur

Les signaux logiques d'entrée/sortie du composant à concevoir sont résumés dans le tableau ci-après:

Le sens des signaux logiques est indiqué du point de vue du contrôleur I2C.

<u>Nom signal</u>	<u>Le sens</u>	<u>Rôle</u>
DBus	Bidirectionnel	Bus de données du processeur sur 16 bits.
Addr	Entrée	Bus de l'adresse du processeur sur 22 bits.
nBE0, nBE1	Entrée	Byte Enable, signaux établissant l'ordre des octets sur le bus de données.
nAS	Entrée	Adresse strobe, signal validant le positionnement d'une valeur sur le bus d'adresse,

RnW	Entrée	Signal de sélection de lecture ou écriture
nWAIT	Entrée	signal permettant l'introduction de cycles d'attente spécifiques
nRST	Entrée	signal d'activation du circuit.
CLK	Entrée	Signal d'horloge
SDA	Bidirectionnel	Les données du contrôleur I2C série
SCK	Sortie	Signal d'horloge généré par le contrôleur I2C
nIT_I2C	Sortie	signal d'interruption produit par le contrôleur I2C
nCS_I2C	Entrée	Chip select, signal du sélection du contrôleur I2C.

1.4 Chronogrammes caractéristiques

Les chronogrammes caractéristiques du contrôleur I2C portent sur les cycles d'échange avec le processeur et sur les données échangées avec les périphériques externes via le signal SDA.

1.4.1 Chronogrammes des cycle d'échange avec le processeur

Le contrôleur I2C doit pouvoir échanger avec le processeur. Le chronogramme de la figure 1.3 présente les échanges entre le processeur et le contrôleur I2C.

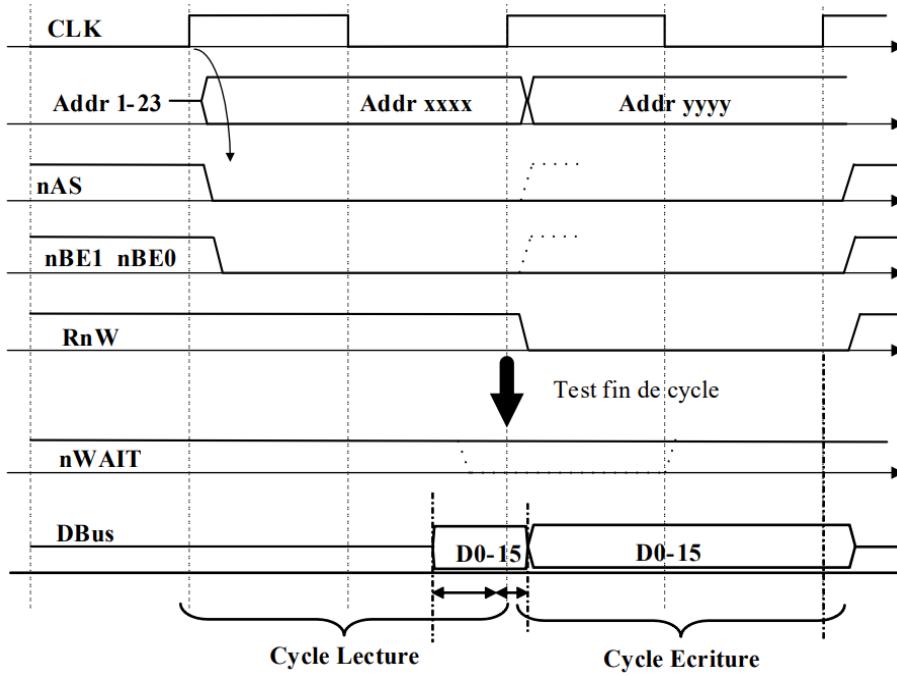


Figure 3: Cycles de lecture et écriture du processeur

Les cycles d'écriture correspondent à l'écriture des données de configuration du contrôleur ou à l'écriture des données à transmettre en série au périphérique externe.

Les cycles de lecture correspondent à la lecture de données d'état du contrôleur ou de données provenant de la réception de trames en série.

Pour échanger avec un périphérique en I2C, le processeur doit fournir au contrôleur I2C une adresse, une donnée et une commande (lecture, écriture).

Ces informations sont transmises au contrôleur I2C par le bus de données **DBus** du processeur.

Le bus **ADDR** du processeur permet de sélectionner un des registres internes du contrôleur I2C.

Dans le cas où le processeur veut **écrire** une donnée sur un périphérique I2C, il doit fournir l'adresse du périphérique, les données et une commande d'écriture ($RnW='0'$) au contrôleur I2C via le bus de données DBus.

Dans le cas où le processeur veut **lire** une donnée d'un périphérique I2C, le processeur doit fournir l'adresse de la lecture et une commande de lecture($RnW='1'$).

1.4.2 Chronogrammes des cycle d'échange avec un périphérique externe

Les échanges entre le contrôleur I2C et un périphérique externe se fait via les signaux **SDA** et **SCK**.

Une trame I2C est composée de :

- Un bit START (SDA passe à '0' et SCK reste à 1).
- 8 bits d'adresse (sélectionner le périphérique externe avec lequel le circuit échange de données).
- Un bit de direction des données (R / W) ("0" écriture , '1' lecture).
- 8 bits de données.
- Un bit d'accusé de réception
- Un bit Stop (SDA passe à '1' et SCK reste à '1').

Le nombre d'octets pouvant être échangés est défini lors de la configuration du circuit par le processeur. Les données sont transmises à partir du bit le plus significatif (MSB) de l'octet.

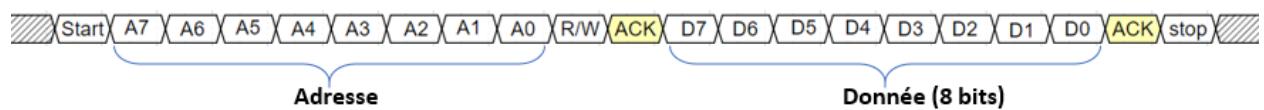


Figure 4:Format d'une trame I2C

Chaque octet doit être suivi d'un bit d'accusé de réception (ACK).

La figure ci-dessous représente un cycle de lecture de deux octets.

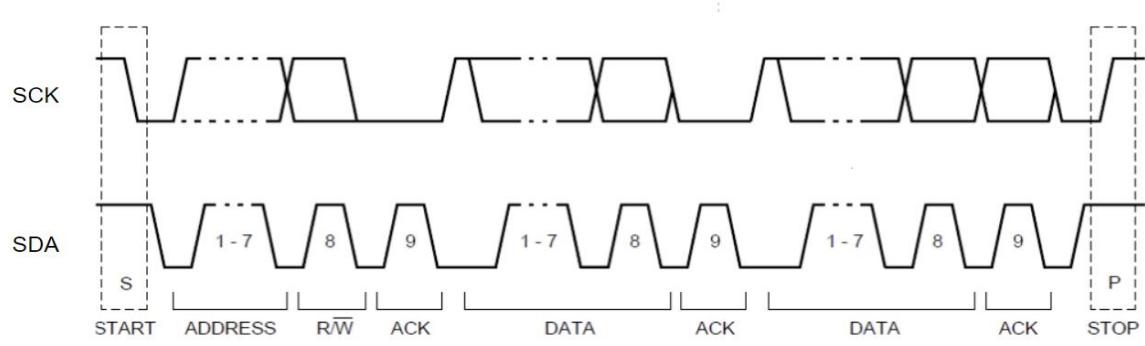


Figure 5: Cycle de lecture de données par le contrôleur I2C

Le bit d'accusé de réception qui suit le bit R/W est toujours envoyé par l'esclave I2C pour indiquer qu'il a bien reçu les bits d'adresse et le bit R/W (lecture ou écriture). La source du bit d'accusé de réception qui suit chaque octet de données dépend de la direction des données du côté du contrôleur. S'il s'agit d'une écriture, le bit d'accusé de réception provient de l'esclave I2C et s'il s'agit d'une lecture, il provient du contrôleur I2C pour indiquer qu'il a reçu un octet de l'esclave. A la fin de la transmission ou de la réception d'une trame de données série, un signal d'interruption (**nIT_I2C**) peut être activé en direction du contrôleur d'interruption si la configuration effectuée au préalable du contrôleur I2C par le processeur l'autorise dès qu'il y a le bit stop. La désactivation de ce signal se fait dès que le registre d'état du circuit est lu par le processeur.

1.5 Contraintes du projet

1.5.1 Contraintes technologiques

- Le circuit doit pouvoir fonctionner soit en Big-Endian soit en Little-Endian.
- Lors d'un échange avec le processeur, les signaux nBE[0] et nBE[1] sont utilisés pour définir les octets à utiliser sur le bus DBus.
- Le circuit sera intégré par la suite dans le système donné par la figure 1.1, il faut ainsi prendre en considération la relation entre les différentes composantes de ce système.
- L'environnement de la synthèse et la simulation du contrôleur est FPGA Advantage de Mentor Graphics.

1.5.2 Contraintes temporelles

- La date de livraison du rapport final est durant la semaine 49.

2. Spécifications

L'objectif de cette étape est de décrire de façon fonctionnelle le circuit en gardant une vue externe sans se préoccuper de la solution interne.

2.1 caractérisation de l'environnement

2.1.1 Délimitation des entités

L'environnement du circuit à concevoir est constitué de deux entités :

- **Système à µP** : il comprend le processeur, le contrôleur de mémoire et le contrôleur d'interruption. Le système µP fournit les paramètres de configuration du contrôleur I2C. Il peut fournir une adresse pour effectuer un cycle de lecture de données comme il peut fournir des données et l'adresse de destination pour effectuer un cycle d'écriture au contrôleur I2C. Il peut également récupérer l'état du contrôleur et gérer ses interruptions.
- **Le périphérique I2C externe** : Il peut transmettre ou recevoir des données du circuit.

2.1.2 Comportement des entités

Dans cette partie, on décrit le comportement des entités vis-à-vis du contrôleur I2C.

➤ Le comportement du système à microprocesseur :

La figure 2.1 présente le comportement de l'entité système à µP.

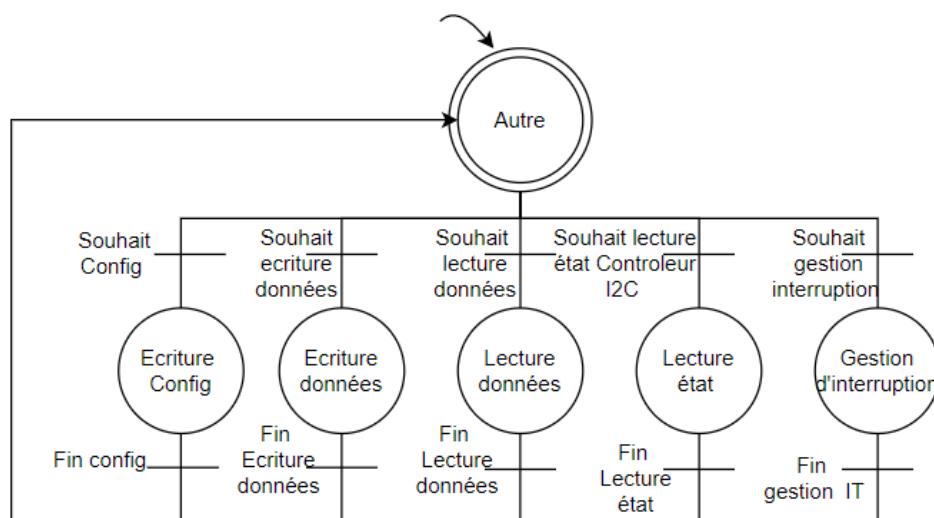


Figure 6: Comportement du système à µP vis à vis le contrôleur I2C

Le système µP pilote le système final et donc le contrôleur I2C. Il peut demander de configurer le contrôleur I2C, de lire les données reçues, d'écrire les données à

transmettre, de lire l'état du circuit et de traiter les interruptions produites par le contrôleur.

➤ **Le comportement du périphérique I2C externe:**

La figure 2.2 illustre le comportement du périphérique I2C vis-à-vis du contrôleur I2C.

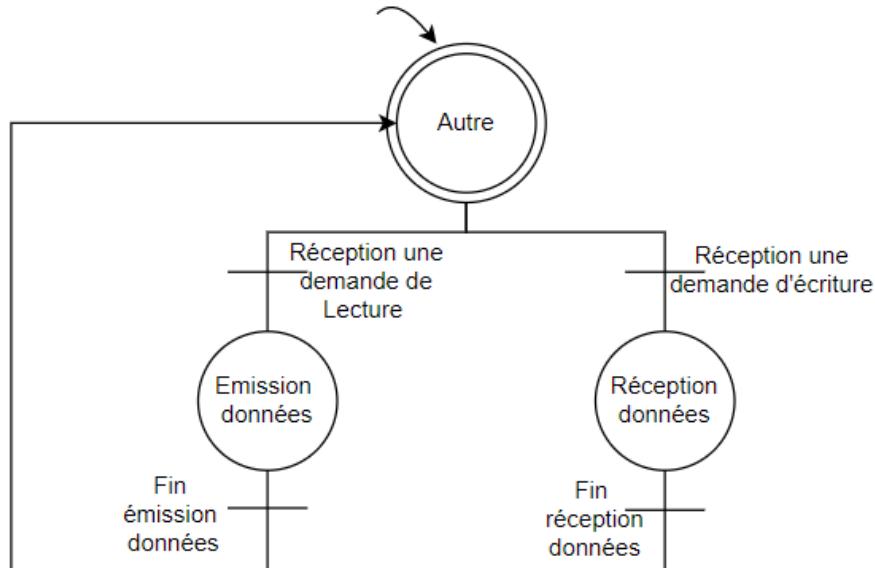


Figure 7:comportement de l'esclave I2C vis à vis le contrôleur I2C

L'entité périphérique I2C joue le rôle d'un esclave I2C puisque le contrôleur est configuré par défaut en tant que maître. Il peut répondre à une demande de lecture ou d'écriture de données issue du contrôleur I2C.

2.1.3 Délimitation des entrées sorties du système:

La caractérisation de l'environnement du contrôleur I2C conduit à l'identification des relations fonctionnelles qui le lient aux entités. Ces relations sont présentées sur la figure 2.3.

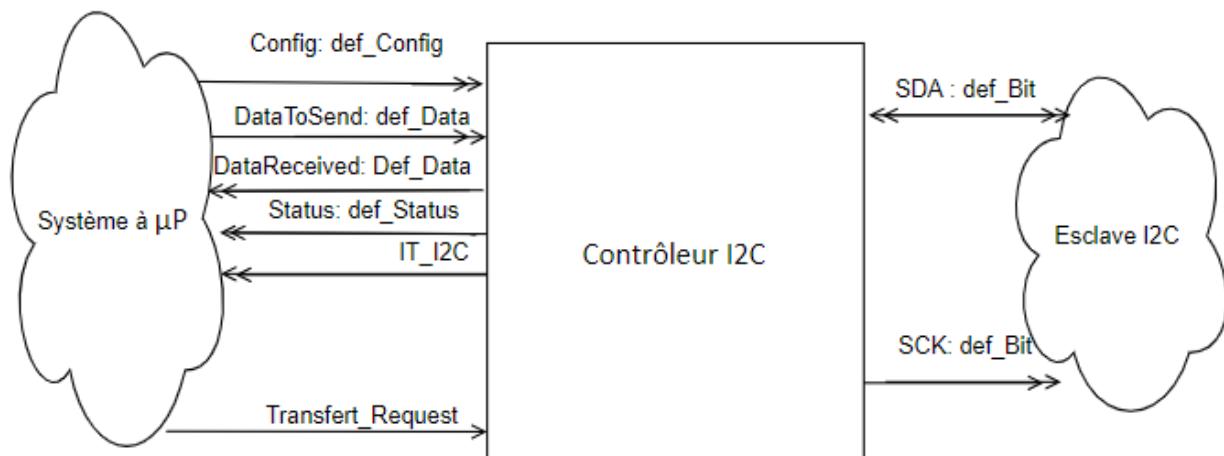


Figure 8:Délimitation des E/S du contrôleur I2C

Le tableau ci-après donne les caractéristiques des relations identifiées.

Tableau 2.1: Caractéristiques des relations d'entrées-sorties vis-à-vis le Contrôleur I2C

Nom de la relation	Sens (vis-à-vis du contrôleur I2C)	Nature	Nom du type
Config	entrée	permanente	def_Config
Status	sortie	permanente	def_Status
DataToSend	entrée	permanente	def_Data
DataReceived	sortie	permanente	def_Data
transfert_Request	entrée	événement	def_Request
IT_I2C	sortie	permanente	def_IT
SCK	sortie	permanente	def_Bit
SDA	sortie	permanente	def_Bit

Les types introduits dans les tableaux sont les suivants :

def_Config :

IT_Enable: defConfig_IT +
Mode_Vitesse: def_ModeVitesse +
Sens_Echange: def_SensEchange +
Nb_Données: def_NBDonnées

DefConfig_IT=(Enabled, NotEnabled)

def_ModeVitesse=(Standard , Rapide)

def_SensEchange=(Read, Write)

def_NBDonnées =1..16

def_Status = (Busy,Write_Error, Read_Error, Waiting_ACK, Write_Done, Read_Done)

Busy means that the I2C controller is sending or receiving data.

def_Data = Array[0..15] of def_Bit

def_Request =(Active,Inactive)

def_IT=(IT_active, IT_inactive)

def_Bit=(0,1)

2.2 Spécifications fonctionnelles

2.2.1 Comportement du contrôleur I2C

Les spécifications fonctionnelles ont pour objectif de décrire les fonctions que doit assurer le système pour son environnement. Il s'agit de définir le comportement du système vis à vis des entités de manière à ce que le système assure les fonctions demandées.

Le comportement du contrôleur I2C vis-à-vis de son environnement est donné ci-dessous:

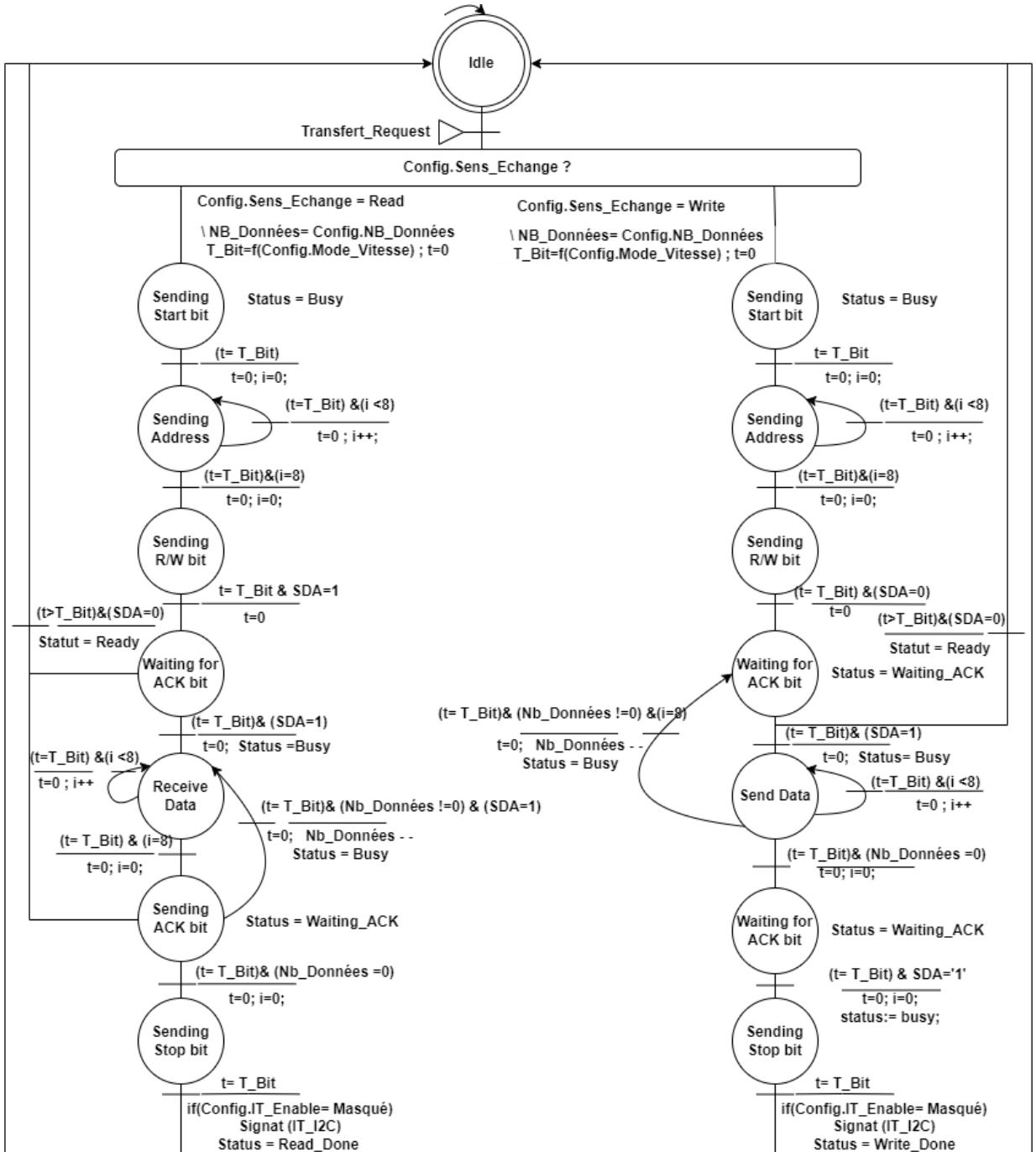


Figure 9: Comportement du contrôleur I2C

Cet automate décrit le comportement du contrôleur I2C en cas d'écriture et de lecture sans présenter la création du signal d'horloge SCK et son lien avec la génération du signal SDA. Pour cela, nous procédonss au raffinement des états.

- **Raffinement de l'état Sending Start bit**

La condition pour démarrer le transfert de données est que le signal SDA passe de l'état haut à l'état bas avant que SCK ne passe de l'état haut à l'état bas (SCK=1, SDA=0). L'automate de la figure 2.5 correspond au raffinement de l'état Sending Start bit.

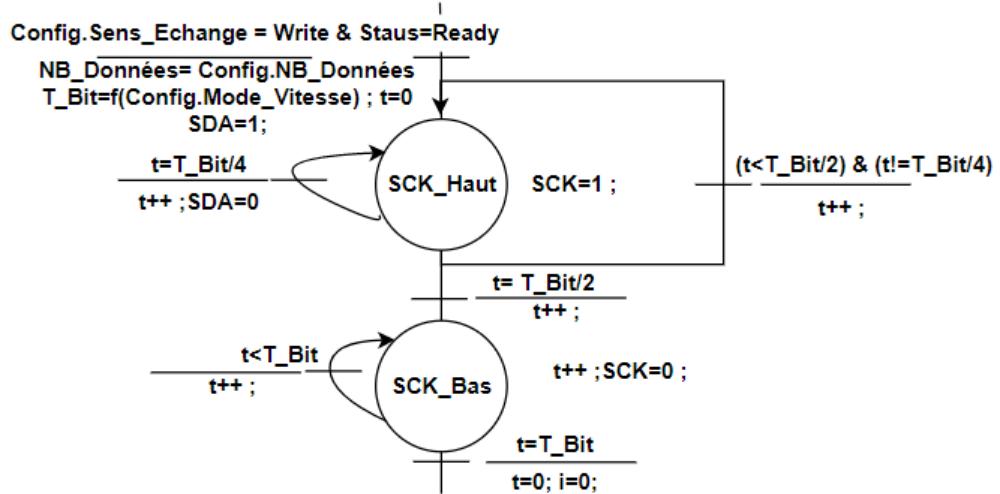


Figure 10: Raffinement de l'état Sending Start bit

- **Raffinement de l'état Sending Stop bit**

La condition pour arrêter le transfert de données est que le signal SDA passe de l'état bas à l'état haut après que SCK passe de bas en haut (SCK=1, SDA=1). L'automate de la figure 2.6 correspond au raffinement de l'état Sending Stop bit.

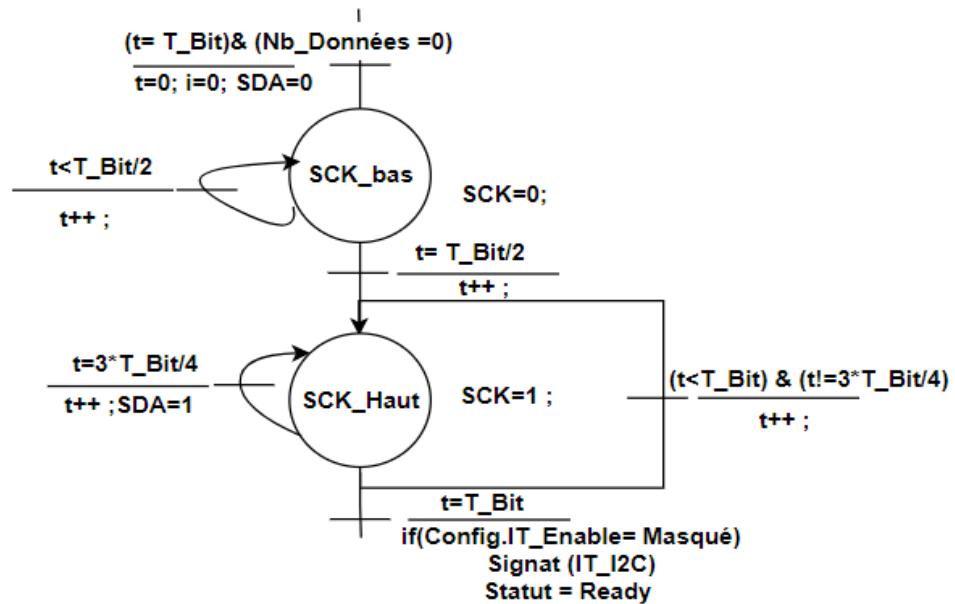


Figure 11: Raffinement de l'état Sending Stop bit

- Raffinement des états qui correspondent à un envoi de données par le contrôleur I2C

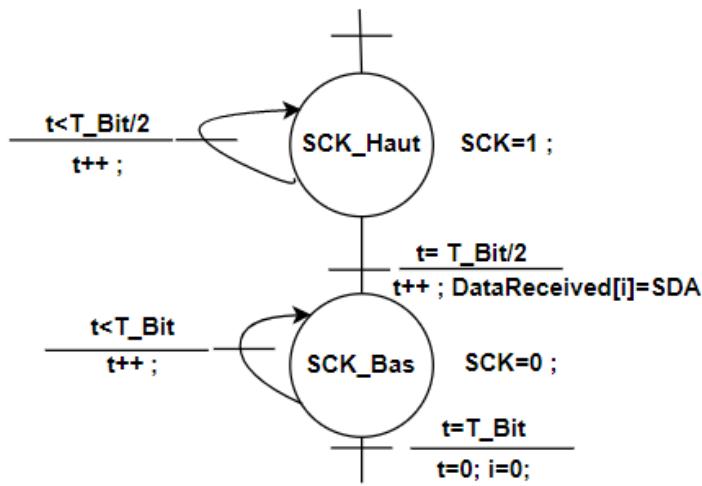


Figure 12: Raffinement des état d'envoi de données

- Raffinement des états qui correspondent à une réception de données par le contrôleur I2C

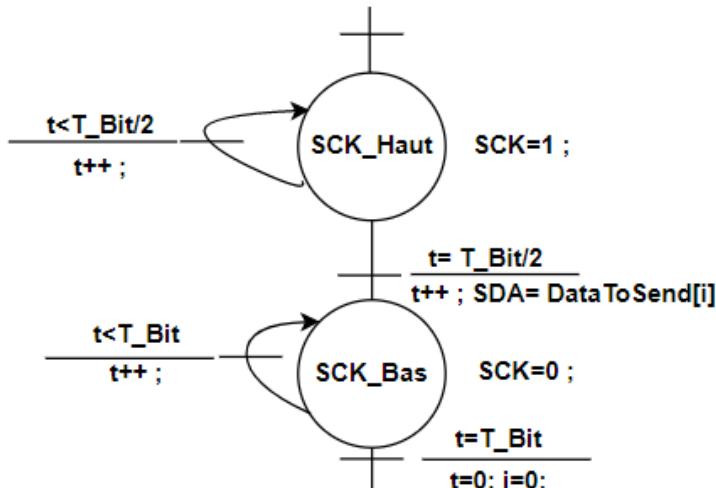


Figure 13: Raffinement des état de réception de données

2.2.2 Diagramme d'activité

Cette étape permet de définir, à partir des comportements décrits dans les spécifications fonctionnelles, les activités qui fonctionnent en parallèle au sein du système.

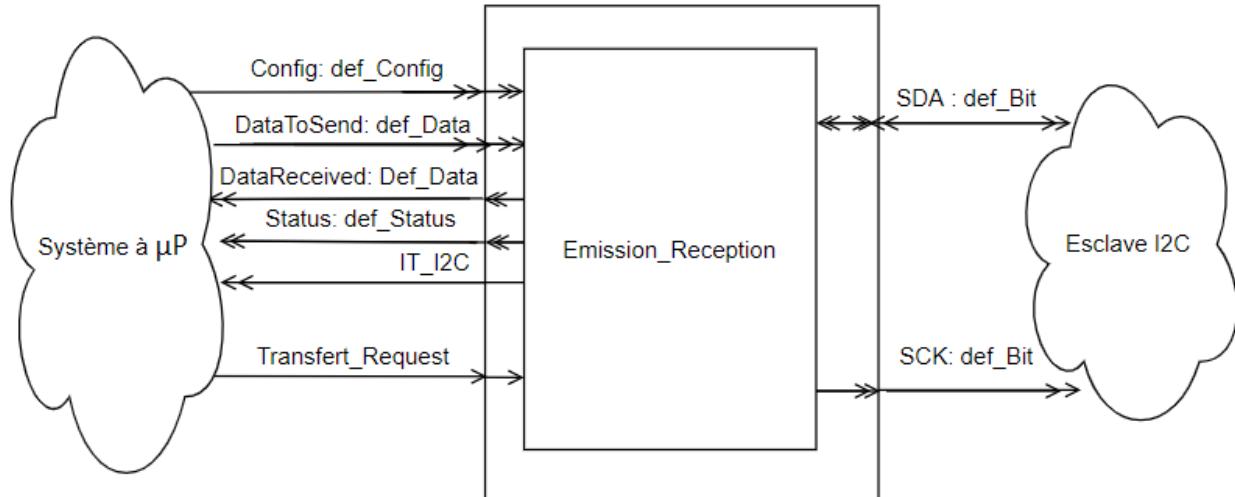


Figure 14: Diagramme d'activité du contrôleur I2C

- L'activité Gestion_Horloge est utilisée pour générer l'horloge interne du système ainsi que le signal d'horloge SCK qui arbitre les échanges de données entre le système et le dispositif I2C externe. La génération d'horloge commence à la réception d'un Transfert_Request et en se basant sur la vitesse de transmission définie par le processeur.
- L'activité Emission_Reception est utilisée pour effectuer les transferts de données en fonction des paramètres de configuration définis par le système à µP.

2.2.3 Spécifications des registres

Cette étape permet de définir les registres internes du contrôleur I2C. L'objectif est de définir une structure que le processeur pourra manipuler pour contrôler le circuit.

Chacun des registres internes est lié à un offset par rapport à l'adresse de base du composant. Les registres du circuit, ainsi que leurs offsets, sont donnés dans le tableau ci-dessous.

La taille des registres dans le circuit est de 16 bits.

Tableau 2.2: Les registres internes du Contrôleur I2C

Offset (Adresse de base +0x__)	Nom du registre	BITS	7	6	5	4	3	2	1	0
0x0	I2C_Config	15 : 8						RnW_select	Baud_Rate	IT_Mode
0x1		7 : 0	NB_Données							
0x2	I2C_Status	15 : 8								
0x3		7 : 0			Read_Done	Write_Done	Busy	waitting_ACK	Read_Error	Write_Error
0x4	I2C_Addr	15 : 8								
0x5		7 : 0	Address[7:0]							
0x6	I2C_DataToSend	15 : 8								
0x7		7 : 0	DataToSend[7:0]							
0x8	I2C_DataReceived	15 : 8								
0x9		7 : 0	DataReceived[7:0]							

Les champs en gris sont des bits non utilisés.

Le contrôleur I2C possède les registres suivants :

- **Registre de configuration** de l'IP (**I2C_Config**), pour stocker les données de configuration envoyés par le système µP, de taille 16 bits organisés comme suit en partant du LSB:
 - Un bit pour les interruptions: **IT_Mode**
1: mode interruption est activée
0: pas d'interruption
 - Un bit pour la vitesse de transmission: **Baud_Rate**
1: mode standard 100Kbit/s
0: mode rapide 4000 Kbit/s
 - Un bit pour définir le sens de la transmission: **RnW_Select**
1: Il s'agit d'une lecture

0: Il s'agit d'une écriture

- 8 bits pour définir le nombre de données à transmettre : **NB_Données** .

Ce qui fait que le nombre maximal de données que je peux transmettre est égale à 255 octets.

- **Registre de statut (I2C_Status)** pour stocker l'état de l'IP.

- Un bit pour définir si j'ai une erreur lors de l'écriture.

1: une erreur d'écriture apparaît.

0: pas d'erreur d'écriture.

- Un bit pour définir si j'ai une erreur lors de la lecture.

1: une erreur lors de la lecture apparaît.

0: pas d'erreur de lecture.

- Un bit pour définir si le bit d'acquittement est reçu

1: Le bit d'acquittement est reçu.

0: L'IP est en attente du bit d'acquittement.

- Un bit pour définir si l'IP est disponible.

1: l'IP n'est pas disponible **Busy**.

0: l'IP est disponible **Ready**.

- **Registre d'adresse (I2C_Addr)** pour stocker l'adresse de destination ou de réception de données sur 8 bits.

- **Registre d'envoi de données (I2C_DataToSend)** La valeur de ce registre représente la donnée à envoyer lors de l'écriture dans un périphérique externe.

- **Registre de réception de données (I2C_DataReceived)** pour stocker les données reçues lors d'une lecture.

2.2.4 Ecriture des procédures de base

Cette partie présente les primitives de base nécessaires qui permettent l'utilisation du circuit par le processeur. Ces primitives regroupent:

- La déclaration des types de données,
- La déclaration des constantes utiles,
- La déclaration des procédures de base pour accéder aux registres internes de l'IP.

- Les procédures de configuration de l'IP (Écriture dans le registre I2C_Config)
- La procédure pour lire l'état de l'IP.
- La procédure pour l'écriture des données.
- La procédure pour la lecture de données.
- La procédure pour définir l'adresse de la destination de données.
- La procédure pour lire l'adresse de la réception de données.

- **Déclaration des constantes et des types de données**

```
// Définition des types de base
//-----
typedef uint8_t (unsigned char);
typedef uint16_t (unsigned short);
typedef uint32_t (unsigned int);
typedef def_Address uint8_t;
typedef def_Data    uint8_t;
// Structure of the I2C_Status register
//-----
TypeDef union{
  struct
  {
    uint16_t Write_Error :1 ;
    uint16_t Read_Error :1 ;
    uint16_t Waiting_ACK :1 ;
    uint16_t Busy        :1 ;
    uint16_t Write_Done  :1 ;
    uint16_t Read_Done   :1 ;
    uint16_t             :10 ;
  } bit ;
  uint16_t reg ;
} Def_Status ;

//Structure of the I2C_Config register
//-----
TypeDef union {
  struct
  {
    uint16_t IT_Mode      :1 ;
    uint16_t Baud_Rate    :1 ;
    uint16_t RnW_select   :1 ;
    def_Data Nb_Données  :8 ;
    uint16_t             :5 ;
  } bit ;
  uint16_t reg ;
} Def_Config ;
```

```

//I2C structure
//-----

typedef struct _IP_I2C {
    Def_Config I2C_Config;
    Def_Status I2C_Status;
    Def_Data I2C_DataToSend;
    Def_Data I2C_DATAReceived;
    Def_Address I2C_Address;

} _IP_I2C, * IP_I2C;

// Constantes
//-----

// Base Address value
#define I2C_Register_Base_Address 0x00
//I2C Status
#define Status_WriteError 1<<0
#define Status_ReadError 1<<1
#define Status_WaitingAck 1<<2
#define Status_Busy 1<<3
#define Status_WriteDone 1<<4
#define Status_ReadDone 1<<5

//I2C configuration
#define Config_IT_Active 1<<0
#define Config_IT_Inactive 0<<0
#define Config_BaudRate_100 0<<1
#define Config_BaudRate_4000 1<<1
#define Config_WriteRequest 0<<2
#define Config_ReadRequest 1<<2

//Function prototypes
//-----
def_Status I2C_GetStatus(IP_I2C *ip_I2C);
void I2C_SetAddress(IP_I2C *ip_I2C , def_Address addr);
def_Address I2C_GetAddress(IP_I2C *ip_I2C);
void I2C_SetFastTransmissionMode(IP_I2C *ip_I2C);
void I2C_SetStandardTransmissionMode(IP_I2C *ip_I2C);
void I2C_IT_Enable(IP_I2C *ip_I2C);
void I2C_SetTransfertRequest(IP_I2C *ip_I2C, Boolean WnR);
void I2C_SetDataToSend(IP_I2C *ip_I2C, def_Data data);
def_Data I2C_GetReceivedData(IP_I2C *ip_I2C);

```

- Définition des procédures de bas niveau

```
// Procedures
//-----
def_Status I2C_GetStatus(IP_I2C *ip_I2C)
{
    return(ip_I2C ->I2C_Status);
}
void I2C_SetAddress(IP_I2C *ip_I2C , def_Address addr)
{
    ip_I2C ->I2C_Address= addr;
}
def_Address I2C_GetAddress(IP_I2C *ip_I2C)
{
    return(ip_I2C ->I2C_Address);
}
void I2C_SetFastTransmissionMode(IP_I2C *ip_I2C)
{
    ip_I2C ->I2C_Config =Config_BaudRate_4000;
}
void I2C_SetStandardTransmissionMode(IP_I2C *ip_I2C)
{
    ip_I2C ->I2C_Config =Config_BaudRate_100;
}

void I2C_IT_Enable(IP_I2C *ip_I2C)
{
    ip_I2C ->I2C_Config= Config_IT_Active ;
}
void I2C_SetTransfertRequest(IP_I2C *ip_I2C, Boolean WnR)
{
    if(WnR)
    {
        ip_I2C ->I2C_Config = Config_WriteRequest;
    }
    else
    {
        ip_I2C ->I2C_Config = Config_ReadRequest;
    }
}
void I2C_SetDataToSend(IP_I2C *ip_I2C, def_Data data)
{
    ip_I2C ->I2C_DataToSend= data;
}
def_Data I2C_GetReceivedData(IP_I2C *ip_I2C)
{
    return(ip_I2C ->I2C_DataReceived);
}
```

2.3 Spécifications opératoires

Les spécifications opératoires regroupent les modalités de test du circuit :

- Vérifier la mise à jour du registre de configuration suite à une configuration par le processeur.
- Valider la condition de début et de fin de transmission (Start & Stop Bit)
- Valider la réception du bit d'accusé de réception à l'issue de l'envoie de l'adresse et du bit Rnw pour une trame de données I2C.
- Valider la réception d'un bit d'acquittement après l'envoi d'une donnée.
- Valider l'envoi d'un bit d'acquittement après la réception d'une donnée.
- Tester la transmission et la réception des données selon les deux modes de transmission de données (standard et rapide).
- Valider la génération du signal d'interruption à la fin de la réception ou l'émission du nombre de données défini dans le registre de configuration lorsque l'IP fonctionne en mode interruption.

2.4 Spécifications technologiques

- Le circuit sera mis en œuvre au sein d'un FPGA Xilinx Zynq 7000.
- Le circuit fonctionne soit en mode standard avec une vitesse de transmission =100kbit/s, soit en mode rapide avec une vitesse de transmission =4000kbit/s.
- Les niveaux de tension des signaux SDA et SCK doivent être compris entre 0V et 5V.
- Le circuit est contrôlé par le processeur via les signaux du bus du processeur et est synchronisé sur l'horloge Clk. Tous les transferts de lecture et d'écriture peuvent être effectués en un seul cycle d'horloge. Dans le cas d'un composant plus lent, le signal nWAIT permet l'insertion de cycles d'attente.

3. Conception

3.1 Délimitation fonctionnelle

La délimitation fonctionnelle établit les entrées et sorties fonctionnelles de l'IP à concevoir. A partir de la délimitation des entrées et sorties dans la phase de spécification, nous pouvons déduire la délimitation fonctionnelle du contrôleur I2C donnée ci-dessous.

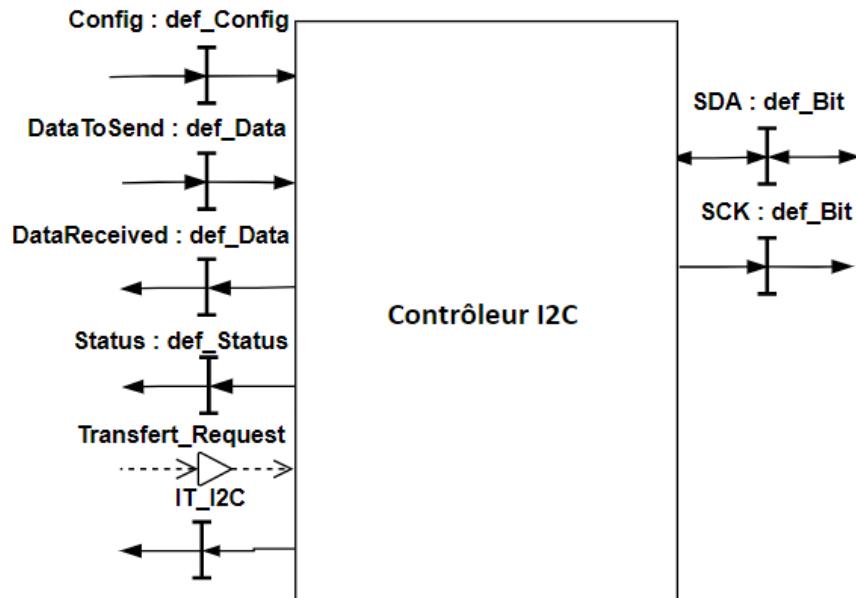


Figure 15:Délimitation des entrées sorties fonctionnelles

3.2 Première décomposition fonctionnelle

La première décomposition vise à établir une première description structurelle du circuit. Cette décomposition se base sur les spécifications et notamment le diagramme d'activité. Une première décomposition fonctionnelle du contrôleur I2C est donnée sur la figure 3.2.

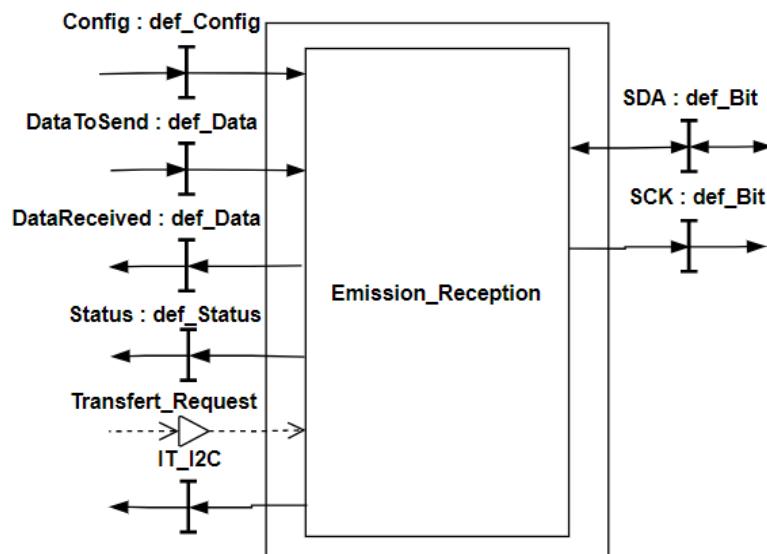


Figure 16:Première décomposition fonctionnelle du circuit.

La décomposition fonctionnelle doit être cohérente avec le diagramme d'activité. De plus, toutes les entrées et sorties sont utilisées, et la fonction Emission_Reception peut produire à partir de ses entrées, les sorties SDA et SCK.

3.3 Raffinement

Cette partie est utilisée pour décrire la solution fonctionnelle de notre IP. Le but est de parvenir à une description plus détaillée au niveau RT du circuit à concevoir dont chaque fonction qu'il constitue doit avoir un comportement synchrone à une horloge de référence.

Pour le contrôleur I2C, la fonction Emission_Reception assure le transfert des données entre le processeur et les périphériques I2C (esclaves) selon le mode de transmission des données (rapide ou standard) défini dans le champ Baud_Rate du registre I2C_Config. D'où la nécessité d'introduire la fonction GestionHorloge afin d'établir les temps d'échantillonnage de SDA et SCK.

Le résultat du raffinement est donné dans la figure ci-dessous.

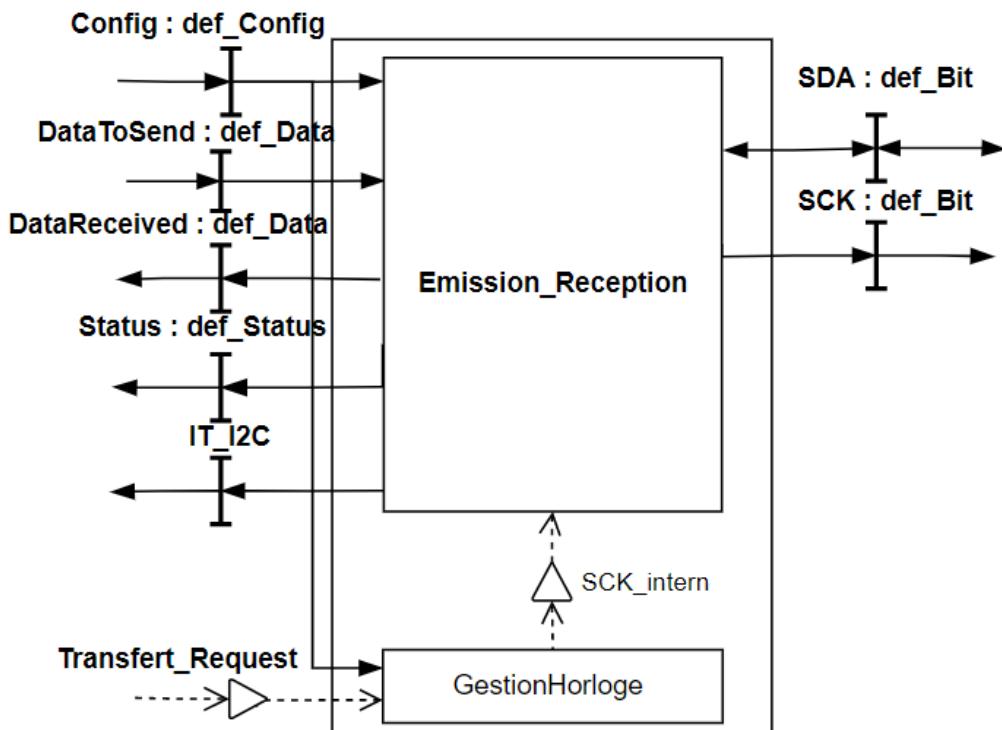


Figure 17: Raffinement de la première décomposition fonctionnelle de l'IP

3.4 Ecriture des algorithmes

Le raffinement de la première décomposition fonctionnelle de l'IP donne lieu à deux fonctions au comportement séquentiel. Ainsi, cette phase vise à compléter la conception

fonctionnelle en décrivant directement le comportement interne de chaque fonction via une description algorithmique.

3.4.1. Algorithme de la fonction GestionHorloge

L'évolution de la fonction **ClockGenerator** dépend du signal **Transfert_Request** qui déclenche le transfert de données et de **Config**, notamment le champ définissant le mode de transmission des données.

```
Action ClockGenerator sur Transfert_Request avec
(entrée var Config : Def_config;
sortie SCK_intern);
begin
cycle:
    Transfert_Request: begin
        if(Config.Baud_rate=1) then
            Tbit := 10E+5;
        else then
            Tbit := 2.5*10E+7;
        end if;
        wait(Tbit / 4);
        signal(SCK_intern);
    end cycle;
end GestionHorloge;
```

3.4.1 Algorithme de la fonction Emission_Reception

```
Action Emission_Reception sur ev SCK_intern avec
(entrée var Config: def_Config;
entrée var DataToSend: def_Data;
Sortie var Status: def_Status;
Sortie var DataReceived: def_Data;
Sortie var IT_I2C : def_Bit ;
Sortie var SDA : def_Bit ;
Sortie var SCK: def_Bit; )

Var state : ( idle , sending_StartBit, sending_StopBit, sending_Address,
waiting_AckBit, sending_AckBit, send_Data, Receive_Data, sending_RnWBit);
Var i: 0 .. N-1 //N=4; Fréquence_SCK_intern=4* Fréquence_SCK
Var j: 0 .. 7 // Nombre de bit
Var CptData: integer;
var SCk_flip : signal:='0';// signal intermédiaire pour générer le signal SCK
begin
cycle : SCK_intern
begin
case state of :
    Idle:
        i:=0;
        status.Busy :=0;
        SCK :=1;
```

```

SDA :=1;
j := 7;
CptData:=Config.NB_Données-1;
state :=sending_StartBit;

sending_StartBit:
//SDA passe à zéro pour t=Tbit/4
status.Busy:=1;
if (i=N-1)
    i:=0;
    SCK_flip:=0;
    state:=sending_Address;
else
begin
    if (i=1)
        SDA :=0;
    end if
    if (i=2)
        SCK:=0;
    end if;
    i:=i++;
end;
end if;

sending_Address:
if (i=2 || i=0)then
SCK_flip=not SCK_flip;
SCK := SCK_flip;
end if ;
if((i=N-1) & (j=0) then
    i:=0;
    j:=7;
    status.Write_done :=1;
    state:= sending_RnWBit;
else if ((i=N-1) & (j!=0))then
    j:=j-1;
    i:=0;
Else
    i:=i+1;
end if ;
SDA:= DataToSend(j);

sending_RnWBit:
status.Write_done :=0;
if(i=2 || i=0) then
SCK_flip= not SCK_flip;
SCK := SCK_flip;
end if ;
if ((i:=N-1) then
    status.Write_done :=1;
    i:=0;

```

```

        state:= waiting_AckBit;
    else
        i:=i++;
    end if ;
    SDA:=Config.RnW_Select;
waiting_AckBit:
    if (i=2 || i=0)then
        SCK_flip=not SCK_flip;
        SCK := SCK_flip;
    end if ;
    if (((i=N-1) & (SDA=1)& (CptData=0) then
        i:=0;
        state:= sending_StopBit;
    else if (((i=N-1) & (SDA=1)& (CptData!=0)
        CptData=CptData--;
        i :=0;
        status.Write_error := 0;

        if (Config.RnW_Select=1) then
            state:= receive_Data;
        else
            state:=send_Data;
        end if;
    else if ((i=N-1) &(SDA=0))then
        status.Write_error:=1;
        i:=0;
        state:=idle;
    else
        status.waiting_Ack :=1;
        i:=i++;
    end if ;

send_Data :
    if (i=2 || i=0)then
        SCK_flip=not SCK_flip;
        SCK := SCK_flip;
    end if ;
    if((i=N-1) & (j=0) then
        i:=0;
        j:=7;
        status.waitting_ACK :=0;
        state:= waiting_AckBit;
    else if ((i=N-1) & (j !=0))then
        j:=j--;
        i:=0;
    else
        i:=i++;
    end if;
    SDA:= DataToSend(j);

receive_Data:

```

```

    if (i=2 || i=0)then
      SCK_flip=not SCK_flip;
      SCK := SCK_flip;
    end if ;
    if((i=N-1) & (j=0) then
      i:=0;
      j:=7;
      state:= sending_AckBit;
    else if ((i=N-1) & (j !=0))then
      j:=j-- ;
      i:=0;
    else
      i:=i++;
    end if;
    DataReceived(j):=SDA ;

  sending_AckBit:
    if (i=2 || i=0)then
      SCK_flip=not SCK_flip;
      SCK := SCK_flip;
    end if ;
    if (i=N-1)& (CptData=0) then
      status.Read_Done:=1;
      state:= sending_StopBit;
      i:=0;
    else if ((i=N-1) & (CptData!=0))
      CptData=CptData--;
      state:= receive_Data;
    else
      i:=i++;
    end if ;
    SDA:=1;
  sending_StopBit:
    if (i=N-1)
      i:=0;
      state:=idle;
    else
      begin
        if (i=2)
          SCK :=1;
        end if
        if (i=3)
          SDA:=1;
        end if;
      i:=i++;
    end;

  end case;
end cycle;
end Emission_Reception;

```

3.5 Introduction des interfaces

Cette étape vise à introduire les signaux logiques assurant l'échange entre le contrôleur I2C et le processeur. Pour cela une fonction d'interface est alors introduite afin de faire le lien entre les relations fonctionnelles et les signaux logiques du processeur. De plus, la prise en compte de ses signaux logiques implique l'introduction d'une horloge de référence pour le circuit **CLK** produite par le processeur. L'évolution de chaque fonction est donc régie par rapport à ce signal d'horloge, ce qui implique un changement dans la nature des relations **Transfer_Request** et **SCK_intern**.

Ce changement apporte des modifications aux algorithmes des fonctions **ClockGenerator** et **Emission_Reception** dont le comportement est régi par le signal d'horloge **CLK**.

La solution fonctionnelle obtenue après l'introduction des interfaces est donnée à la figure 3.4.

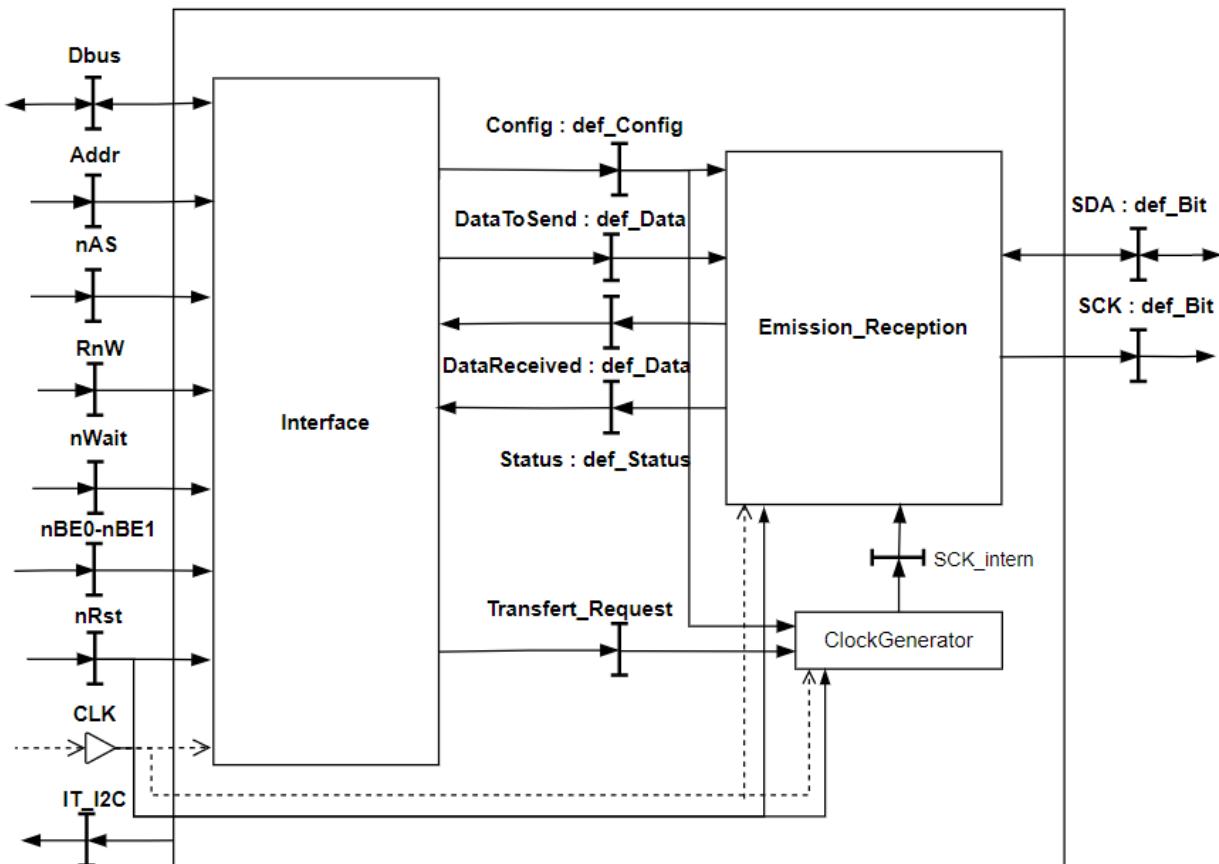


Figure 18:Introduction d'interface

3.5.1 Raffinement de la fonction Interface

Du point de vue du processeur, le circuit est vu comme un ensemble de registres qui sont définis dans la section de spécification des registres. Le processeur a alors deux actions : lire ou écrire dans les registres de l'IP.

Ces deux comportements sont présentés dans la figure ci-dessous. La fonction "Write" est synchronisée à l'horloge du processeur CLK alors que la fonction "Read" est asynchrone afin de respecter les spécifications qui imposent une lecture et une écriture dans un seul cycle d'horloge.

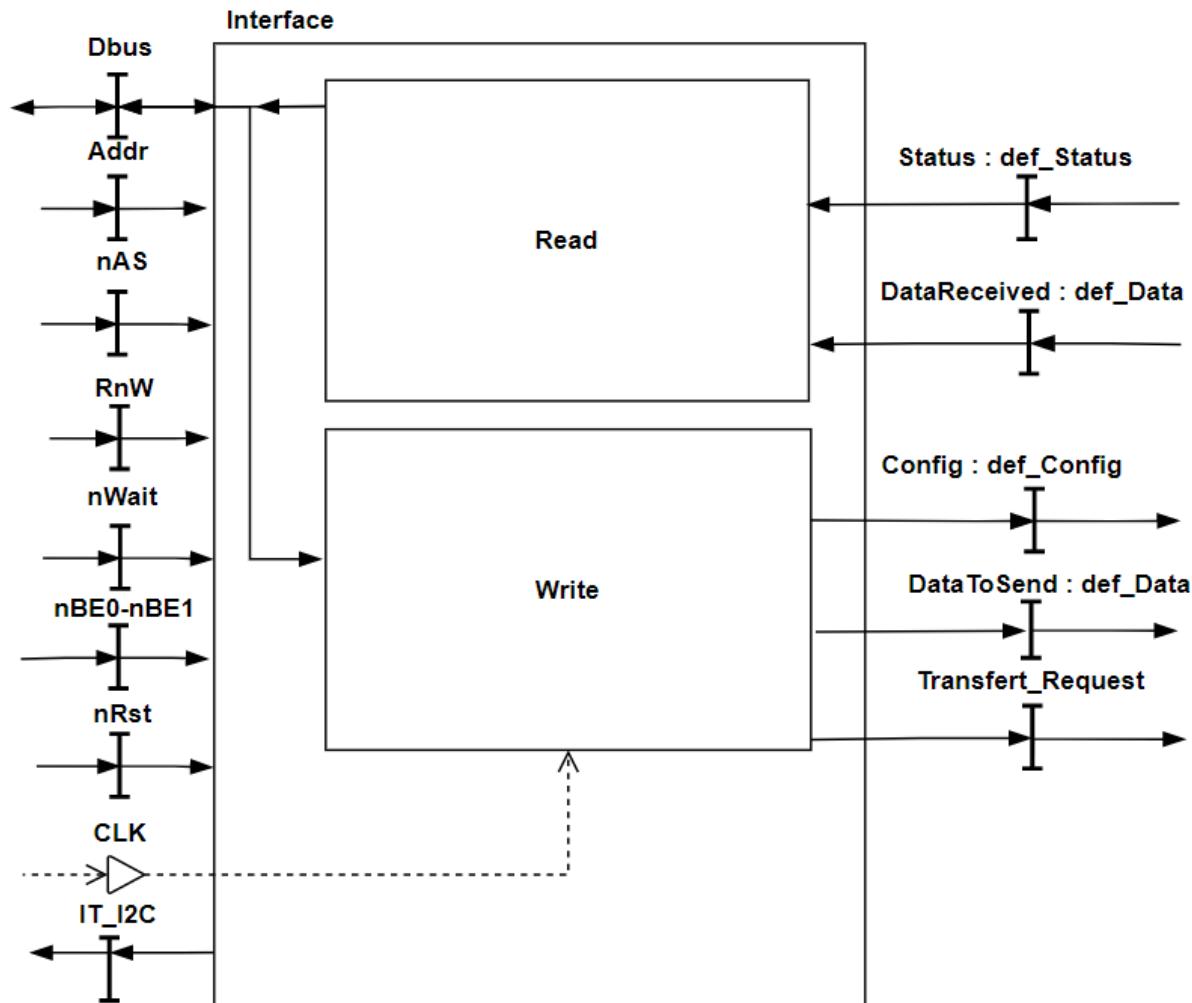


Figure 19: Raffinement de la fonction Interface

3.5.2 Algorithme de la fonction Write

Action Write sur Ev CLK avec

```
(  
Entrée:  
var Addr : Def_Addr ;  
var nAS : Def_Bit ;  
var RnW : Def_Bit ;  
var nBE : Def_Bit ;  
var nRst : Def_Bit ;  
var DBus : Def_DBus ;  
Sortie:  
var transfert_Request : Def_Bit ;  
var DataToSend: Def_Data ;  
var Config: def_Config;  
);  
begin  
cycle CLK : begin  
if( RnW = 0 & nAS = Active ) then  
  case Addr of  
    I2C_Addr :  
      DataToSend := DBus ;  
    I2C_Config:  
      Config :=DBus;  
    I2C_DataToSend:  
      DataToSend := DBus ;  
    I2C_DataReceived:  
      // Nothing to do  
    I2C_Status:  
      // Nothing to do  
  default:  
    // Nothing to do  
  end case ;  
end if;  
end cycle ;  
end Write ;
```

3.5.3 Algorithme de la fonction Read

Action Read avec

```
(  
Entrée:  
var Addr : Def_Addr ;  
var nAS : Def_Bit ;  
var RnW : Def_Bit ;  
var nBE : Def_Bit ;  
var nRst : Def_Bit ;  
var DataReceived: Def_Data ;  
var Status: def_Status;  
Sortie:  
var DBus: def_Data;  
);
```

```

begin
cycle : begin
  if( RnW = 1 & nAS = Active ) then
    case Addr of
      I2C_Addr :
        // Nothing to do;
      I2C_Config:
        // Nothing to do;
      I2C_DataToSend:
        // Nothing to do ;
      I2C_DataReceived:
        DBus:=DataReceived;
      I2C_Status:
        DBus:= Status ;
    default:
      // Nothing to do;
  end case ;
end if;
end cycle ;
end Read ;

```

3.6 Description de la solution pour les tests

L'objectif de cette partie est de définir les tests à mettre en œuvre pour vérifier le fonctionnement du contrôleur I2C par rapport aux spécifications. Les tests à réaliser seront les suivants :

- Test du reset asynchrone : vérification de l'affectation par défaut des entrées et sorties et des signaux internes de l'IP.
- Test de la transmission des données selon les deux vitesses de transmission (standard et rapide). Pour cela, la fonction d'interface doit être capable d'écrire dans le registre interne I2C_Config de l'IP spécifiquement le champ Baud Rate.
- Ecrire dans les registres internes du circuit dans les deux modes possibles : Big Endian et Little Endian.
- Lire les registres internes du circuit dans les deux modes possibles : Big Endian et Little Endian.
- Tester l'envoi d'une trame I2C à un dispositif externe (esclave I2C).
- Tester la réception d'une trame I2C en provenance d'un périphérique externe.

Pour les deux derniers tests, vous devez vérifier tous les champs d'une trame I2C en validant :

- La condition de démarrage d'une trame I2C.

- La réception d'un bit d'accusé de réception suite à l'envoi de l'adresse et du bit RnW indiquant s'il s'agit d'une lecture ou d'une écriture.
 - La réception de bits d'acquittement après chaque transmission de données.
 - L'envoi d'un bit d'acquittement à chaque fois que le circuit reçoit des données.
 - La condition d'arrêt de la transmission de données.
- Lecture du registre I2C_Status pour récupérer le changement d'état du circuit.
- Tester les cas d'erreurs possibles du circuit

4. Saisie de la solution sous HDL Designer

Cette section présente la solution saisie en VHDL avec HDL Designer. Nous avons procédé selon une approche de description descendante.

Tous les types nécessaires, déclarés en phase de spécifications et conception, ont été définis dans un package VHDL afin d'être utilisé dans tous les fichiers VHDL du projet.

Les sections suivantes montrent le résultat de la saisie graphique avec HDL Designer. Pour ne pas alourdir la section, les codes VHDL sont donnés en annexes :

- Le code VHDL du package pour le projet est disponible en annexe A.1 Code VHDL du package utilisé pour le projet.
- Le code VHDL de la fonction **Interface_Write** est disponible en annexe A.2 Code VHDL de la fonction Write de l'interface processeur.
- Le code VHDL de la fonction **Interface_Read** est disponible en annexe A.3 Code VHDL de la fonction Read de l'interface processeur
- Le code VHDL de la fonction **ClockGenerator** est disponible en annexe A.4 Code VHDL de la fonction ClockGenerator.
- Le code VHDL de la fonction **Emission_Reception** est disponible en annexe A.5 Code VHDL de la fonction Emission_Reception.
- Le code VHDL de la fonction **l'environnement de test** est disponible en annexe A.6 Code VHDL de l'environnement de test.

L'entité Environnement de test est décrite pour définir les cycles d'écriture et de lecture produits par le **CPU** et le **périphérique I2C externe** et affectant le fonctionnement de l'IP, ainsi que les fonctions **ResetGenerator** et **ClockGenerator**.

4.1 Le composant de test associer au contrôleur I2C

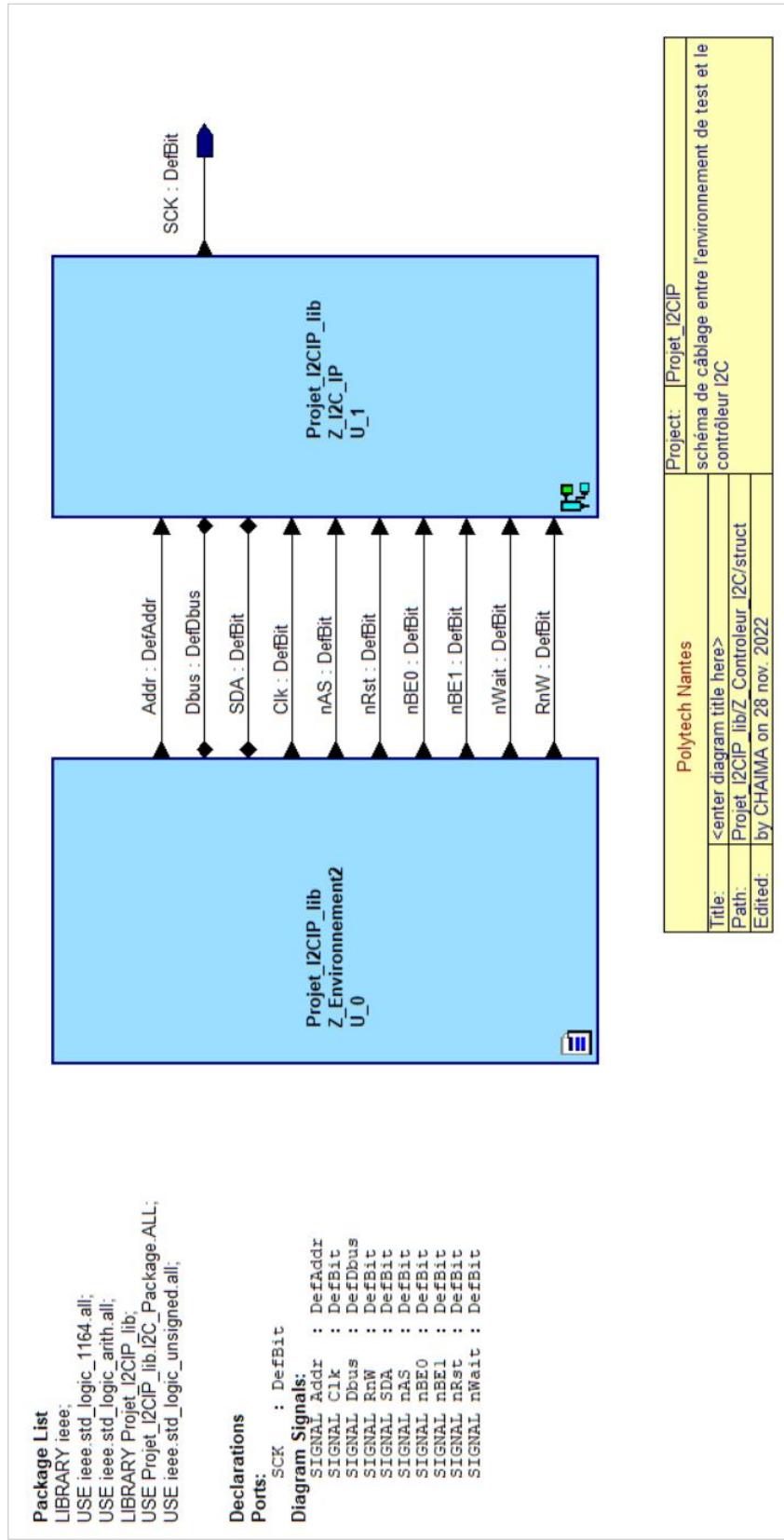


Figure 20: Schéma bloc du contrôleur I2C associé à l'environnement de test

4.2 Le contrôleur I2C

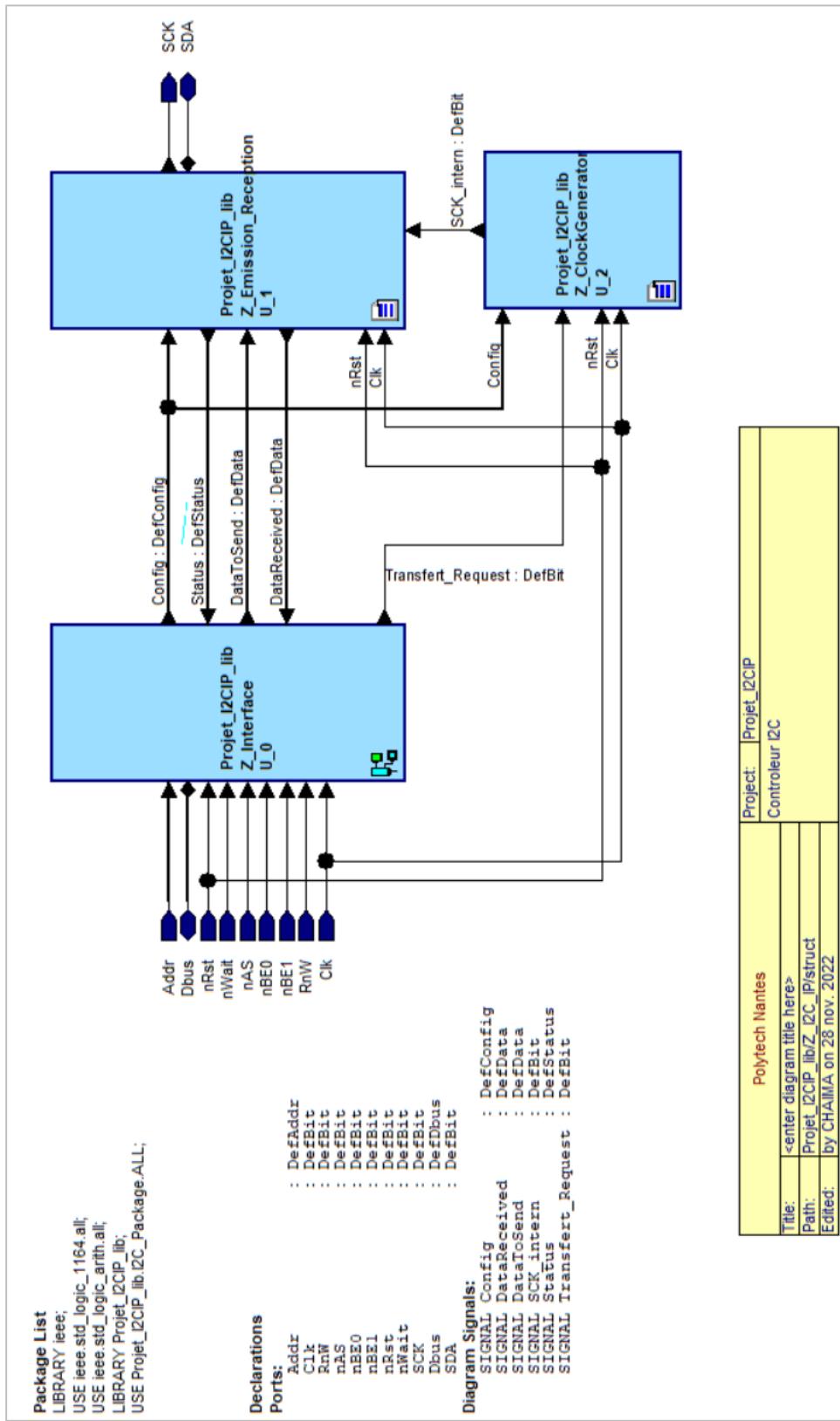


Figure 21: Schéma bloc du contrôleur I2C sous HDL Designer

4.2 Interface pour le processeur

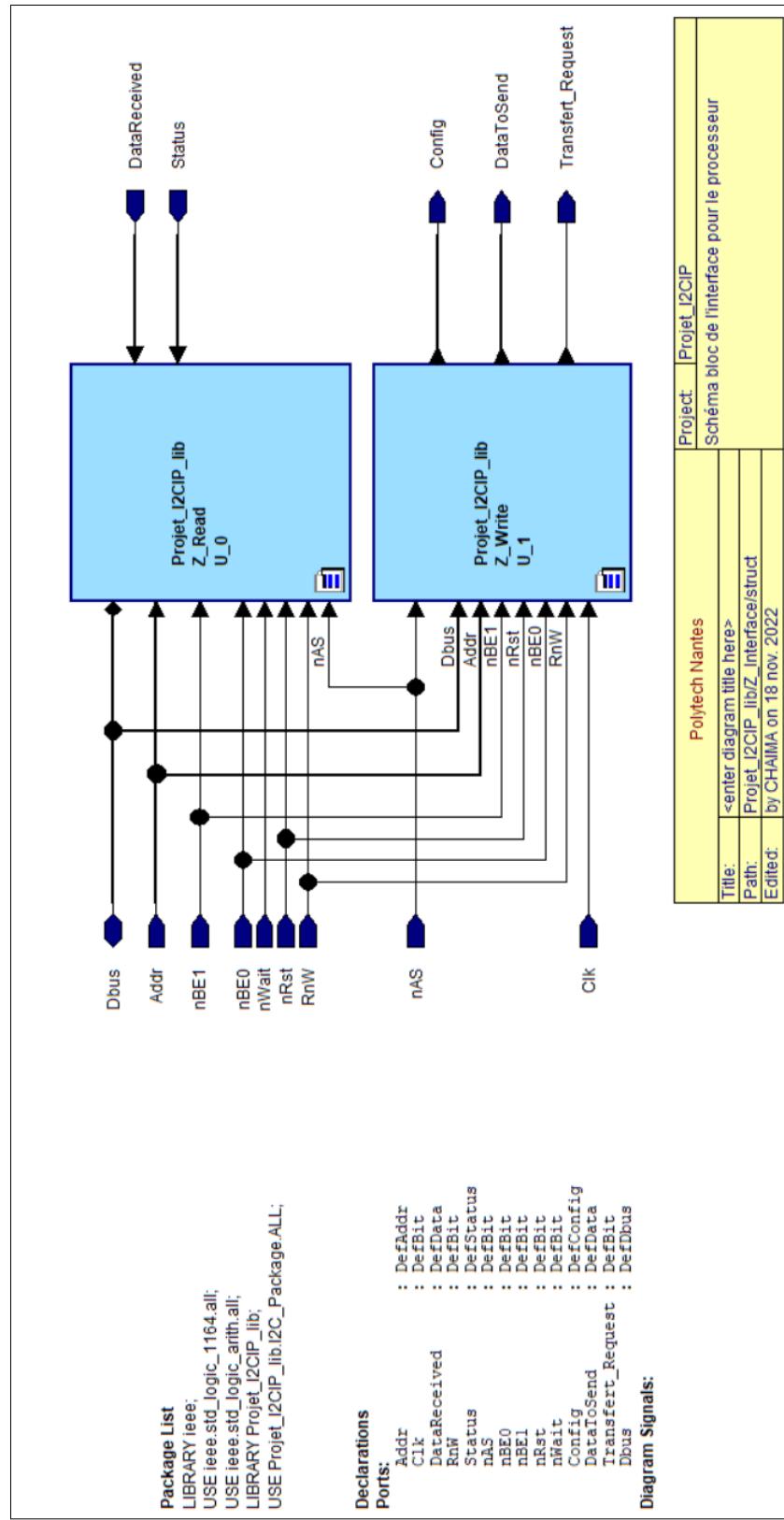


Figure 22: Schéma bloc de l'interface pour le processeur

5. Résultat de simulation

L'objectif de cette étape est d'analyser les chronogrammes caractéristiques du circuit selon les tests définis dans la section 3.6. Le but est de valider le fonctionnement du contrôleur I2C présenté dans le cahier des charges.

La simulation est décomposée en deux parties:

- La première partie pour vérifier le comportement du circuit par rapport au signal asynchrone Reset.
- La deuxième partie pour valider le fonctionnement du circuit vis-à-vis des signaux du processeur, ce qui revient à valider l'écriture et la lecture dans les registres internes du circuit, à vérifier les échanges entre le circuit et le périphérique I2C externe selon la configuration donnée par le processeur.

5.1 Validation du Reset asynchrone

Le circuit mis en œuvre est synchrone à une horloge, il est obligatoire de trouver un signal asynchrone de Reset.

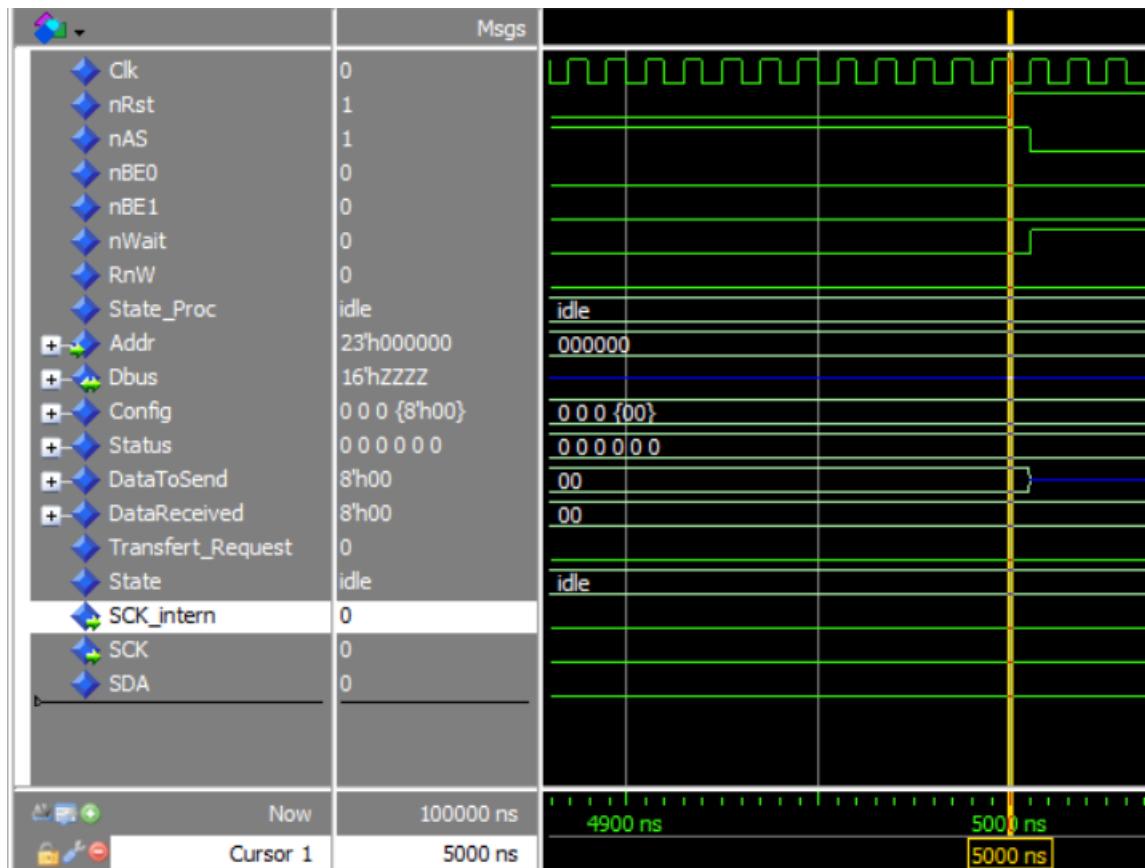


Figure 23: Reset asynchrone du processeur

Le signal de réinitialisation nRst est actif de $t = 0$ à $t = 5000$ ns. Pendant toute cette période, les signaux doivent être dans leur état inactif, ce qui est le cas. Les signaux internes et les sorties restent dans cet état tant que le reset est actif, ce qui valide le fonctionnement de la réinitialisation de l'interface pour le processeur. De plus, les registres internes du contrôleur I2C sont dans leur état inactif, pas de génération de l'horloge interne SCK_intern nécessaire à la synchronisation de la transmission des données par la fonction Emission_Reception. Lorsque nRst est inactif, les signaux internes seront dans leur état actif au prochain front montant du signal d'horloge Clk.

5.2 Validation de la fonction ClockGenerator

Pour le contrôleur I2C, la fonction **ClockGenerator** génère le signal d'horloge interne SCK_intern en fonction de la vitesse de transmission définie lors de la configuration du circuit. L'introduction de ce signal est nécessaire afin d'établir les temps d'échantillonnage de SDA et SCK. La période de SCK_intern doit être égale au quart de la période de transmission des données afin de définir la condition de début et de fin de la trame I2C.

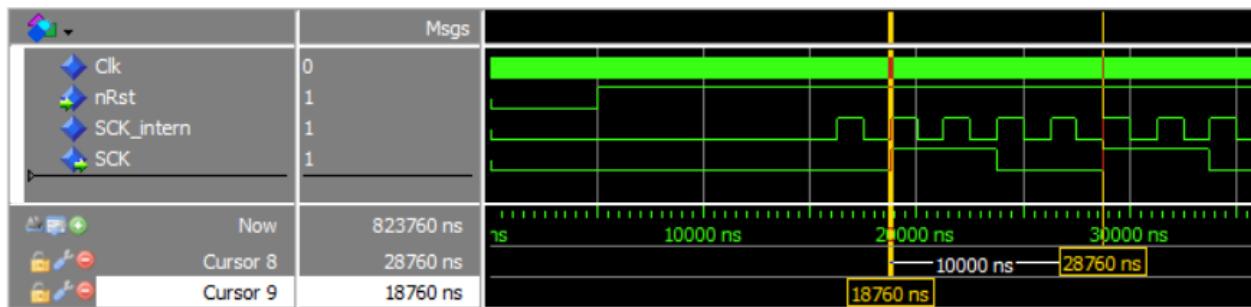


Figure 24: ClockGenerator en mode de transmission standard $T_{sck}=10000\text{ns}$

La vitesse de transmission standard est égale à 100 Kbits/s ce qui correspond à une période $T_{sck}=10000\text{ns}$. Elle correspond à 4 cycles d'horloge de SCK_intern.

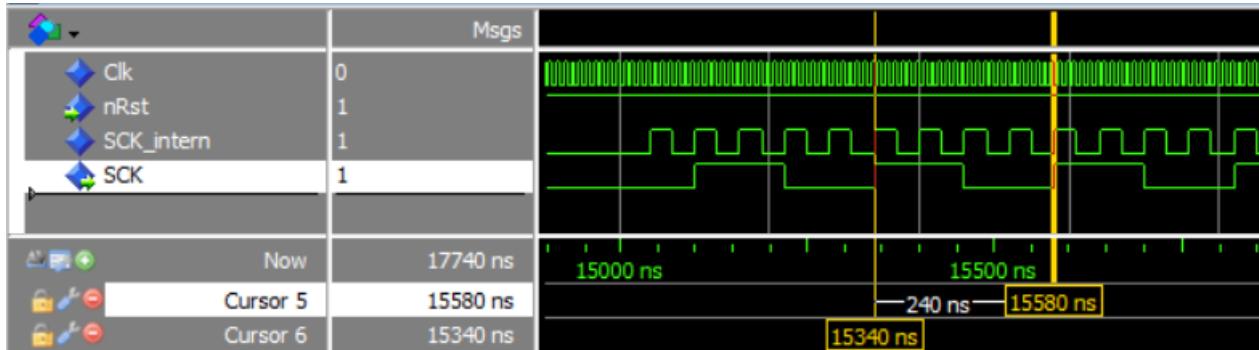


Figure 25: ClockGenerator en mode de transmission rapide $T_{sck}=10000\text{ns}$

La vitesse de transmission rapide est égale à 4000 Kbits/s ce qui correspond à une période $T_{sck}=250\text{ns}$. Elle correspond à 4 cycles d'horloge de SCK_intern. On a obtenu une période de 240ns.

- **Formats du bus de données du processeur**

Le processeur possède un bus de données de 16 bits, il peut donc, grâce à un signal appelé nBE[1 :0], transporter des données selon trois formats :

- Le format 8 bits : Dans ce cas, les données sont présentes sur Dbus[0 :7], le bus d'adresse Addr[0 :21] permet de sélectionner des registres 8 bits dans l'IP.
- **Little Endian**, le format 16 bits: dans ce cas, les données sont présentées sur Dbus[0 :15], les bits [0:7] correspondent à l'octet de poids faible de la donnée à transmettre, les bits [8 :15] correspondent à l'octet de poids fort de la donnée à transmettre.
- **Big Endian**, le format 16 bits: dans ce cas les données sont présentés sur Dbus[0 :15] comme suit : les bits [0 :7] correspondent à l'octet de poids fort de la donnée à transmettre. Les bits [8:15] correspondent à l'octet de poids faible de la donnée à transmettre.

⇒ Le format Little Endian est testé lors de la validation de l'envoi des données par le contrôleur I2C (maître) et le format Big Endian sera testé lors de la réception des données envoyées par le périphérique I2C (esclave).

5.3 Validation de l'envoi des données par le contrôleur I2C

Dans cette partie, nous procédons à la validation de l'envoi d'une trame de données par le contrôleur I2C. La figure suivante présente **l'envoi d'une trame I2C** composée de **trois octets** en mode de **transmission standard** en format **Little Endian**.



Figure 26: Une trame I2C : Envoi de données par le contrôleur I2C

Pour rappel, une trame I2C est constituée de :

- Un bit de START,
- 8 bits d'adresse,
- Un bit de direction des données (R / W) ('0' écriture, '1' lecture),
- Un bit d'accusé de réception de l'adresse,
- 8 bits de données,
- Un bit d'acquittement après chaque octet,
- Un bit d'arrêt.

Alors, pour envoyer trois octets de données, la trame I2C est composée de 39 bits dont la période standard de transmission est de 10000 ns ce qui fait qu'une trame nécessite 39000 ns comme le montre la figure ci-dessus (la trame est limitée par les deux curseurs). Nous commençons alors à vérifier les champs de la trame.

5.3.1 Configuration du contrôleur I2C pour l'émission de données

Pour assurer les échanges des trames en émission ou en réception, le processeur doit configurer le contrôleur I2C en définissant la vitesse de transmission (100 kbit/s en mode standard, 4000 kbit/s en mode rapide), le type d'échange : une lecture ou une écriture, le nombre de données à transmettre sans génération d'IT (la génération d'interruption est abordée au niveau des spécifications et la conception mais il ne sera pas pris en compte dans cette partie).

Le chronogramme ci-dessous montre un cycle d'écriture par le processeur dans le registre de configuration du circuit I2C_Config.

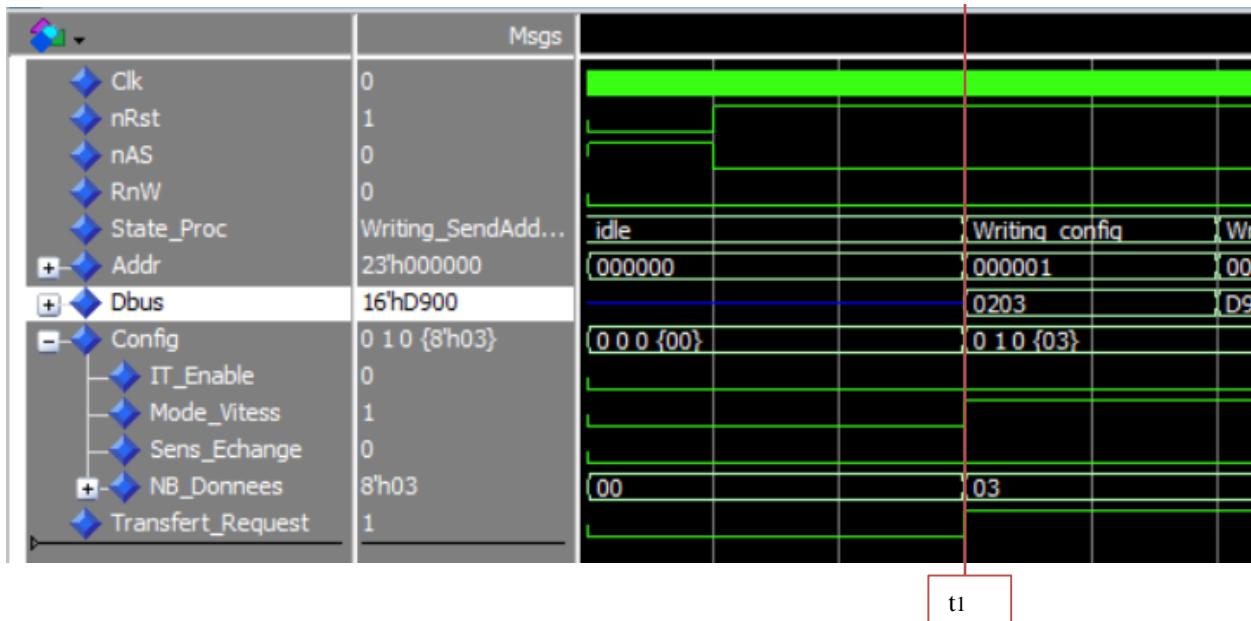


Figure 27: Ecriture par le processeur dans le registre I2C_Config

Avant t1, le processeur est dans un état de repos, le Dbus est en haute impédance et les signaux de contrôle sont tous inactifs. A partir de t1 le processeur entre dans l'état writing_config. Il écrit la valeur 0x0203 à l'adresse Addr=1 qui correspond à l'adresse du registre I2C_Config qui est organisé comme suit :

0x0	I2C_Config	15 : 8				RnW_select	Baud_Rate	IT_Mode
0x1		7 : 0	NB_Données					

En observant le contenu du bus Config, nous pouvons déduire que la valeur 0x0203 correspond à l'envoi de 3 octets à un dispositif I2C sans générer d'interruption avec une vitesse de transmission standard de 100 kbits/s.

Une fois la configuration du circuit effectuée par le processeur suite à l'écriture dans le registre I2C_Config, la fonction "Write" de l'interface Processeur déclenche une demande de transfert via le signal Transfer_Request destiné à la fonction de génération d'horloge interne qui contrôlera la transmission des données via le signal d'horloge interne SCK_intern. L'envoi des données par le contrôleur I2C est déclenché, nous pouvons procéder à la validation des champs nécessaires à une trame I2C.

5.3.2 Validation du bit Start

La condition pour démarrer le transfert de données est que le signal SDA passe de l'état haut à l'état bas avant que SCK ne passe de l'état haut à l'état bas (SCK=1, SDA=0). L'envoie de ce bit de correspond à l'état Sending_StartBit du circuit comme le montre la figure suivante :

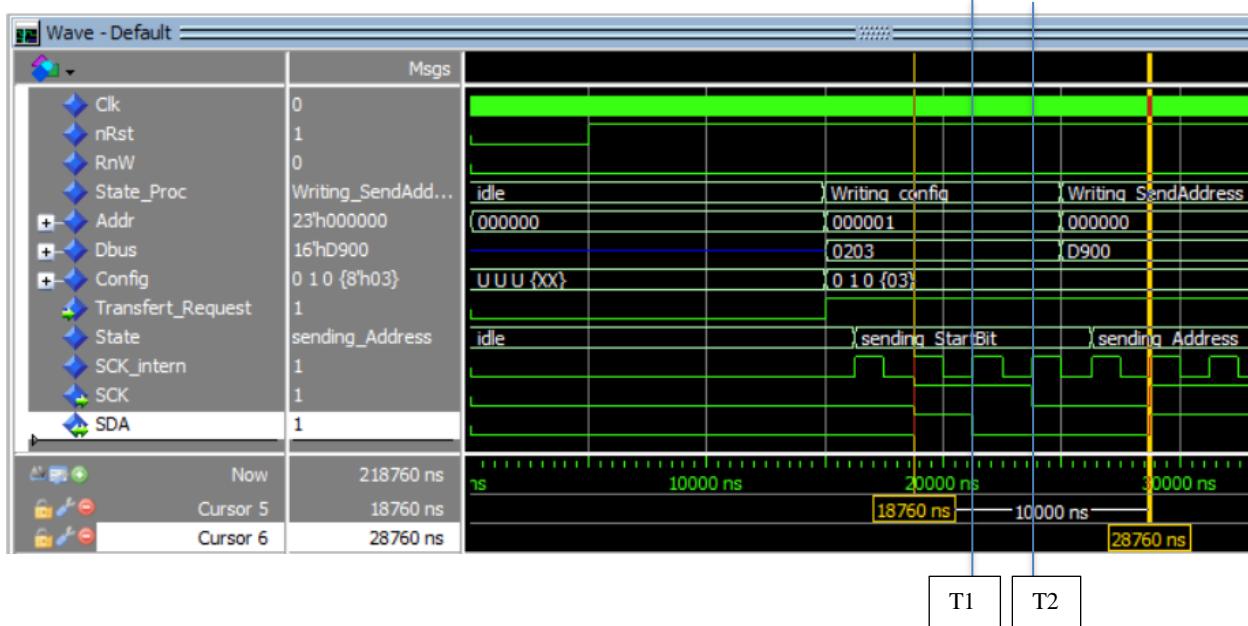


Figure 28: Le bit de démarrage d'une trame I2C

L'envoi du bit de départ se fait en un cycle d'horloge de SCK comme il est limité sur la figure par les deux curseurs. Les deux signaux SDA et SCK démarrent à l'état haut.

$T_1 = T_{sck}/4$: SDA passe de l'état haut à l'état bas et SCK est maintenu au niveau haut.

$T_2 = 3T_{sck}/4$: le passage de SCK de l'état haut à l'état bas et SDA est maintenu au niveau bas.

⇒ La condition de début d'une trame I2C est vérifiée.

5.3.3 La transmission de l'adresse

Vu que le nombre de composants qu'il est possible de connecter sur un bus I2C étant largement supérieur à deux, il est nécessaire de définir pour chacun une adresse unique. L'adresse d'un circuit, codée sur huit bits. Cette adresse est transmise en commençant par le bit le plus significatif(MSB). La transmission de l'adresse nécessite 80000 ns comme le montre la figure ci-dessous.

Avant de transmettre l'adresse, le processeur doit écrire cette adresse dans le registre I2C_Addr.

T_1 : un cycle d'écriture par le processeur est déclenché. Le contenu de Dbus 0x00D9 est écrit à l'adresse Addr=0, ce qui fait que le contenu du bus DataToSend =D9.

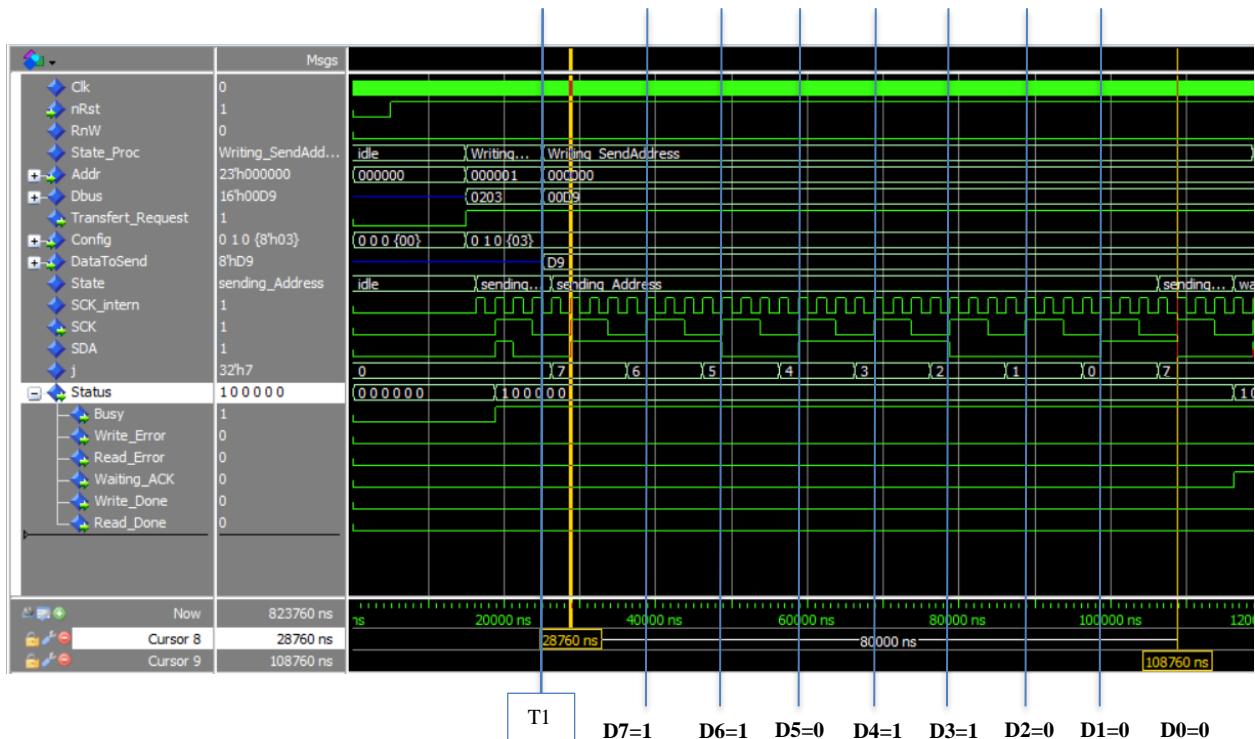


Figure 29: Envoi d'adresse du périphérique de destination

- ⇒ Les huit bits qui suivent le bit Start correspondent à la valeur de l'adresse D9 définie par le processeur.
- ⇒ L'état du circuit est Busy puisque l'envoi de la trame est en cours.

5.3.4 La transmission du bit RnW

Après l'envoi du Start bit et de l'adresse, le contrôleur I2C doit envoyer le bit RnW qui permet de signaler s'il veut lire ou écrire une donnée.

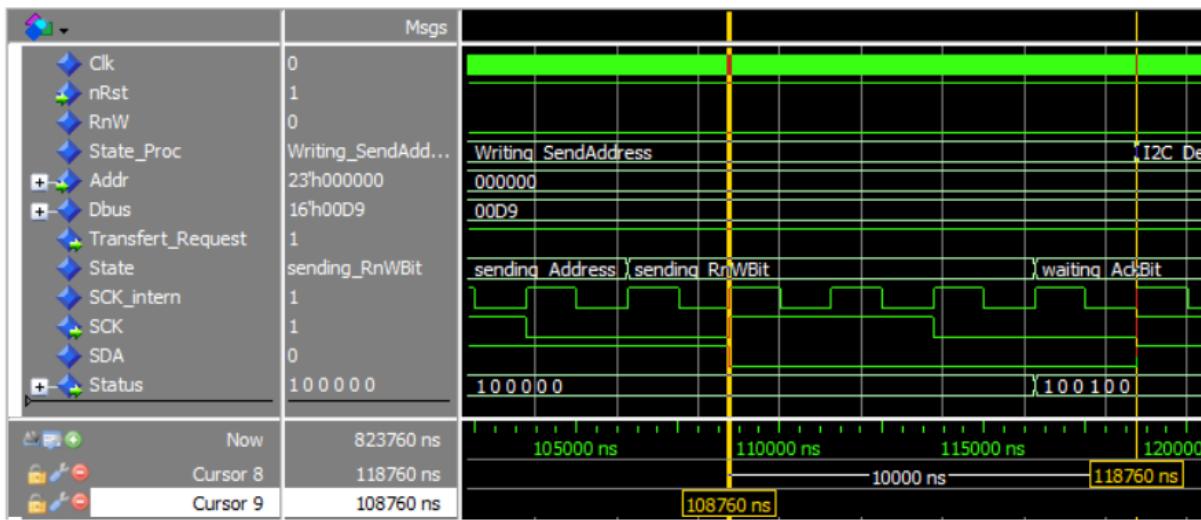


Figure 30: Envoi du bit RnW

- ⇒ Il s'agit d'envoi de données donc le bit RnW est égale à 0.

5.3.5 Le bit d'accusé de réception de l'adresse

Après l'envoi de l'adresse et du bit RnW, le contrôleur I2C (maître) doit vérifier la disponibilité du périphérie I2C (esclave) en attendant le bit d'acquittement ACK.

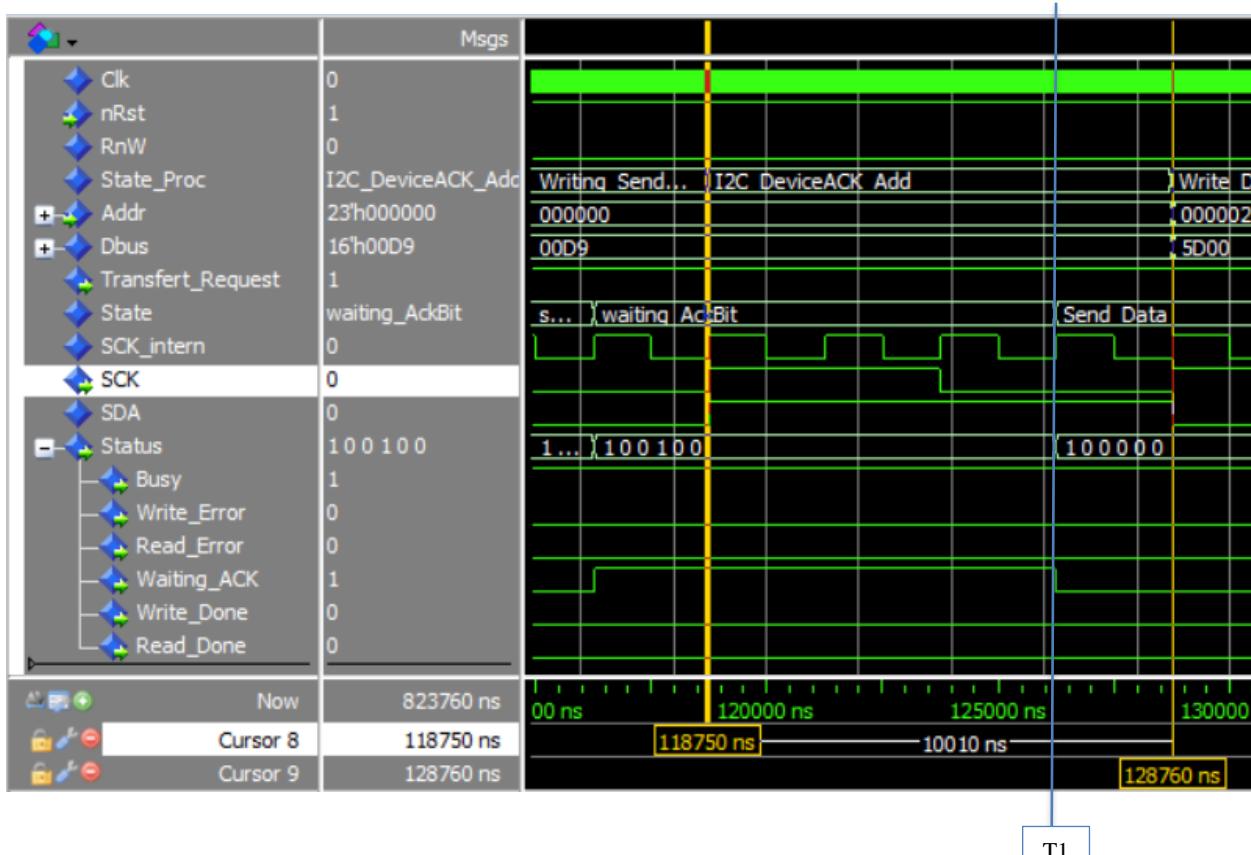


Figure 31: Attente du bit d'accusé de réception d'adresse

Le contrôleur I2C attend le bit d'accusé de réception d'adresse (Status=100100). Les données ne peuvent être envoyées que si le signal SDA est à l'état haut. La vérification de l'état du signal SDA par le circuit se fait à partir de $3T_{sck}/4$.

Le comportement de l'esclave I2C est défini dans l'environnement de test de l'IP.

T1 : Le périphérique I2C est disponible, l'IP a détecté le bit d'acquittement et il a changé de statut (Status =100000).

⇒ Le contrôleur I2C peut commencer à envoyer des données.

5.3.6 L'envoi de données

Pour rappel, le contrôleur I2C doit envoyer 3 octets au périphérique I2C (esclave). La figure suivante montre l'envoi d'un octet de données suivi de la réception du bit d'accusé de réception.

➤ L'envoi du premier octet :

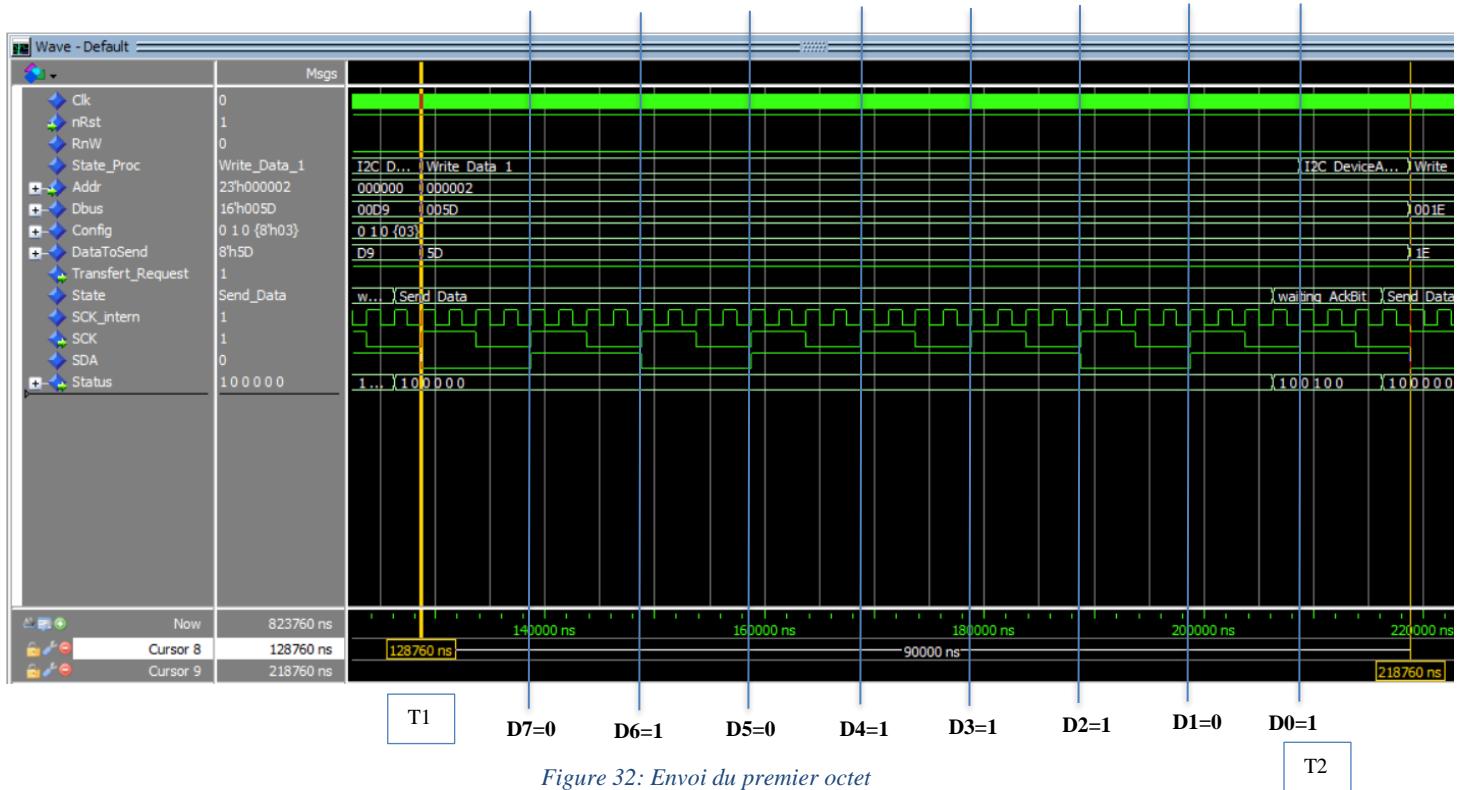


Figure 32: Envoi du premier octet

T1 : Le processeur commence un cycle d'écriture de données dans le registre I2C_DataToSend qui est situé à l'adresse Addr=2. Le contenu du Dbus est égal à 0x005D. Les bits de 0 à 7 sont écrits dans le bus DataToSend via la fonction "Write" de l'interface du processeur puisqu'il s'agit de données au format Little Endian. Le processeur reste dans l'état Write_Data_1 pendant 80 000 ns afin que le comportement de l'interface soit synchronisé avec la fonction Emission_Reception puisque $T_{sck}=1000*T_{Clk}$.

[T1 , T2[: le contrôleur I2C, fonctionnant en mode maître, applique le bit de poids fort D7 de donnée à SDA. Il répète l'opération jusqu'à ce que l'octet complet soit transmis comme le montre la figure ci-dessous, qui correspond à 5D. Le contrôleur I2C change de statut (Status =100100), il est occupé et attend le bit d'acquittement (2.2.3 Spécifications des registres).

T2 : L'esclave doit alors imposer un niveau '1' au signal SDA pour signaler au maître que la transmission a réussi. Dès que l'IP a détecté le bit d'acquittement, il est revenu à son état initial (Status=100000) et peut procéder à l'envoi de l'octet suivant.

➤ L'envoi du deuxième octet :

La figure suivant présente l'envoi du deuxième octet au périphérique I2C suivi de la réception du bit d'acquittement.

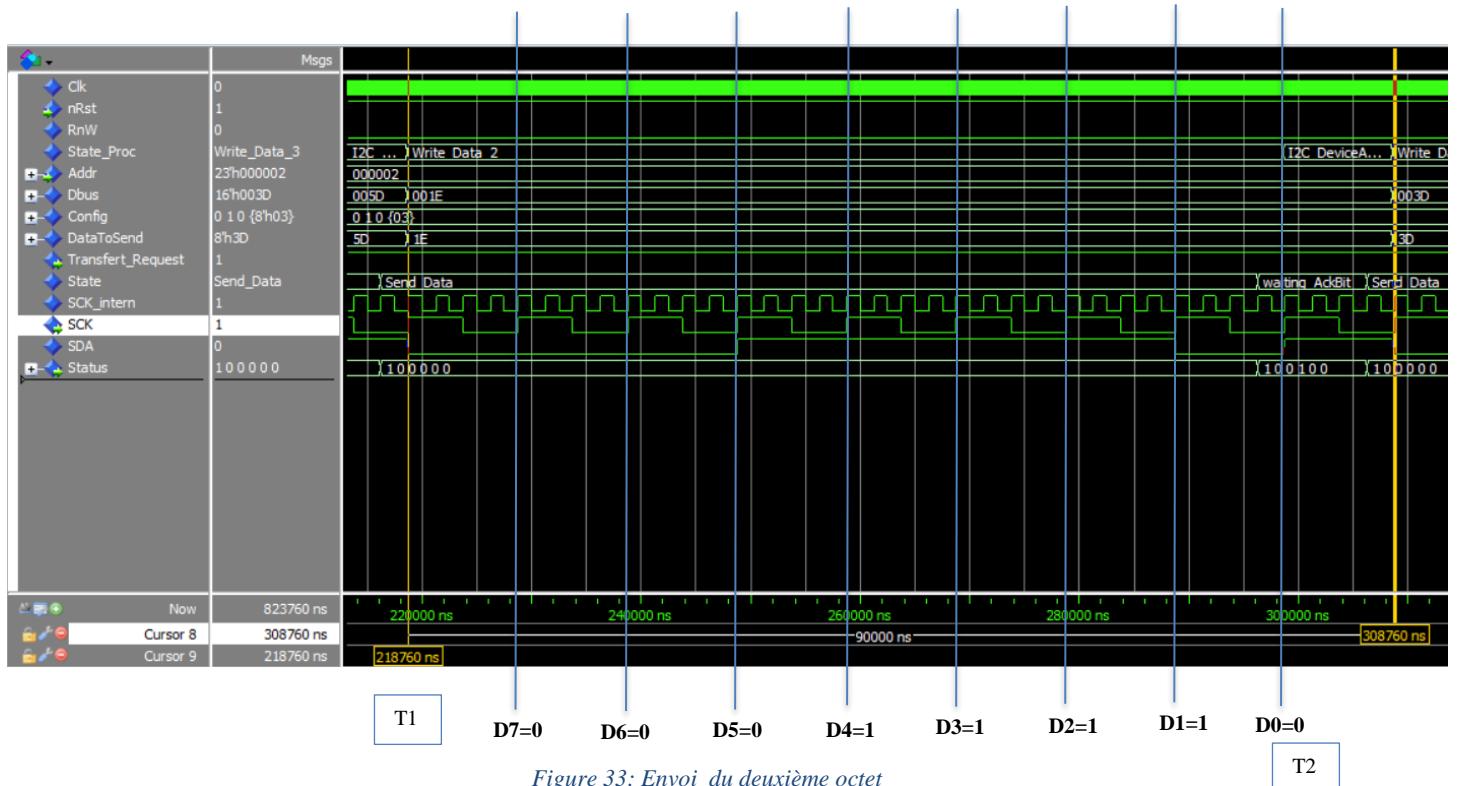


Figure 33: Envoi du deuxième octet

L'analyse est identique à la précédente et permet l'envoi du deuxième octet. L'esclave a imposé le niveau '1' sur le signal SDA à la fin de l'envoi des sept bits. L'octet envoyé est égal à 1E.

⇒ Le contrôleur I2C peut procéder à l'envoi du dernier octet.

➤ L'envoi du troisième octet :

La figure suivant présente l'envoi du dernier octet au périphérique I2C suivi de la réception du bit d'acquittement.

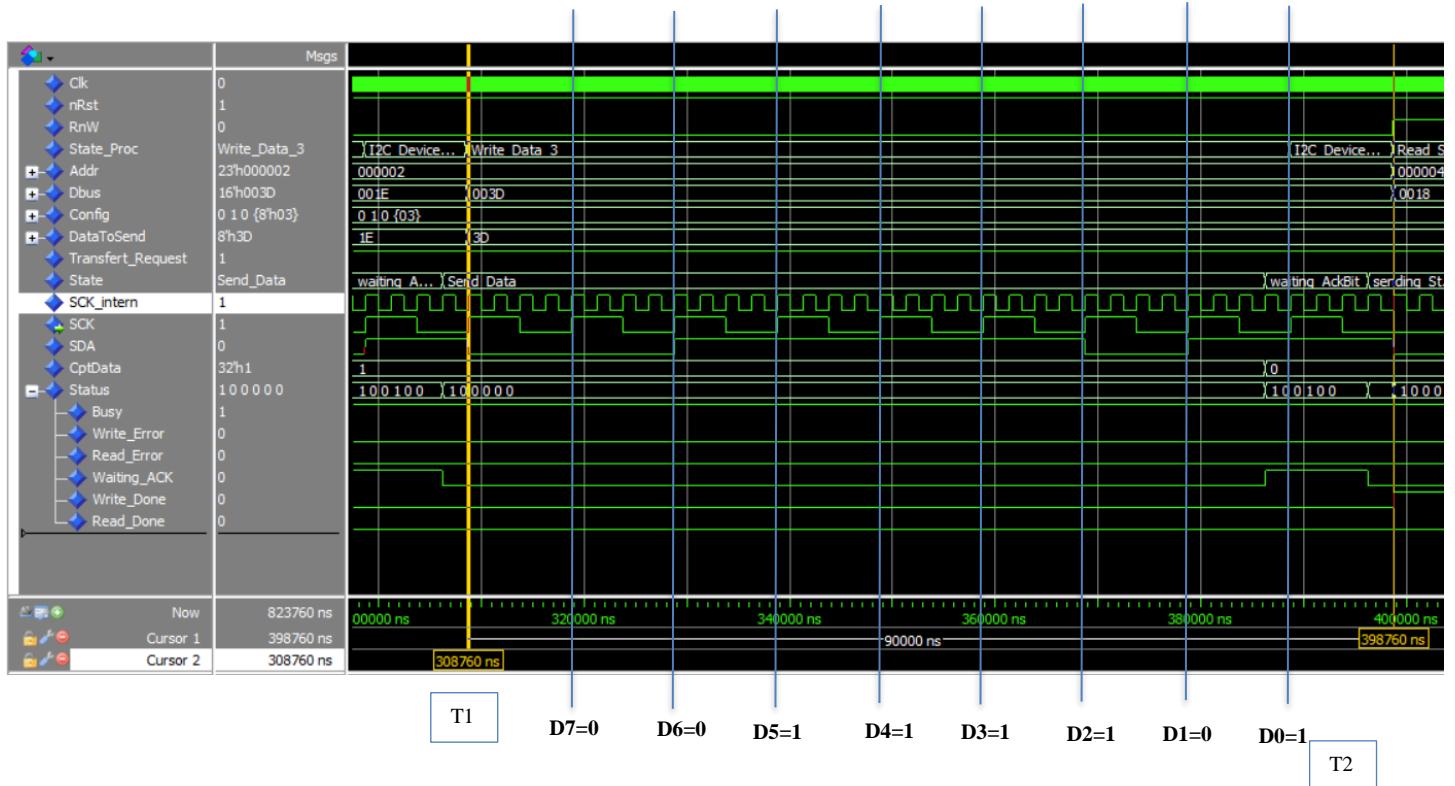


Figure 34: Envoi du dernier octet

L'analyse est identique à la précédente et permet l'envoi du dernier octet. L'esclave a imposé le niveau '1' sur le signal SDA à la fin de l'envoi des sept bits. L'octet envoyé est égal à 3D.

⇒ Le contrôleur I2C peut envoyer le bit d'arrêt pour déclarer la fin de la trame I2C.

5.3.7 Validation du bit Stop

La condition pour arrêter le transfert de données est que le signal SDA passe de l'état bas à l'état haut après que SCK passe de l'état bas à l'état haut (SCK=1, SDA=1). L'envoie de ce bit de correspond à l'état Sending_StopBit du circuit comme le montre la figure suivante :

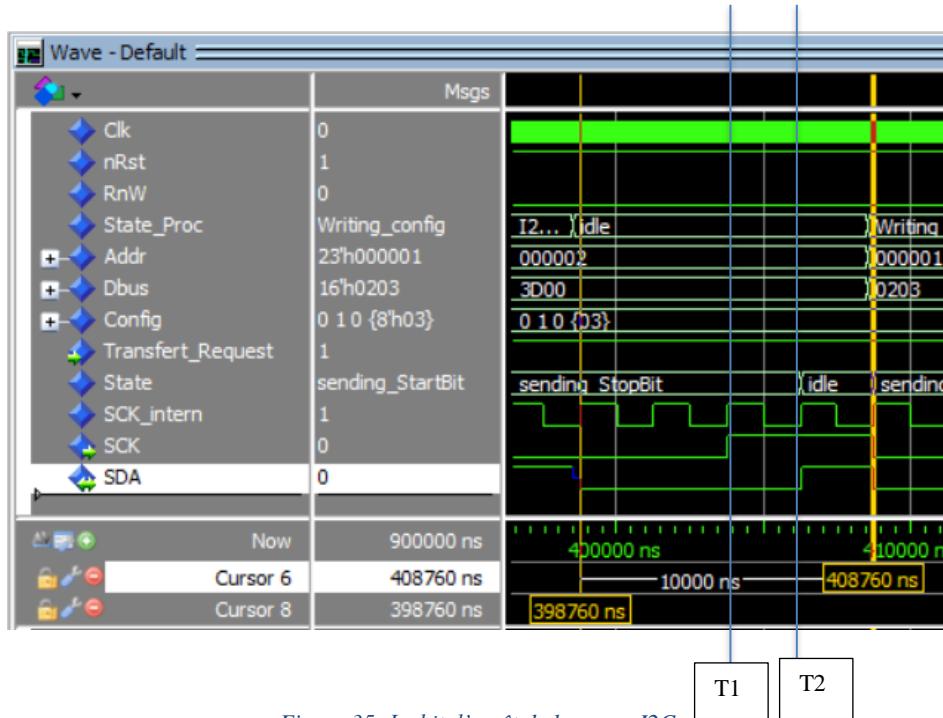


Figure 35: Le bit d'arrêt de la trame I2C

L'envoi du bit d'arrêt se fait en un cycle d'horloge de SCK comme il est limité sur la figure par les deux curseurs. Les deux signaux SDA et SCK démarrent à l'état bas.

$T_1 = T_{SCK}/2$: SCK passe de l'état bas à l'état haut et SDA est maintenu au niveau bas.

$T_2 = 3T_{SCK}/4$: le passage de SDA de l'état bas à l'état haut et SCK est maintenu au niveau haut.

⇒ La condition d'arrêt d'une trame I2C est vérifiée.

5.3.8 Lecture de l'état du contrôleur I2C

Après la fin de la transmission de la trame I2C, le processeur effectue une lecture de l'état du contrôleur I2C comme le montre la figure ci-dessous.

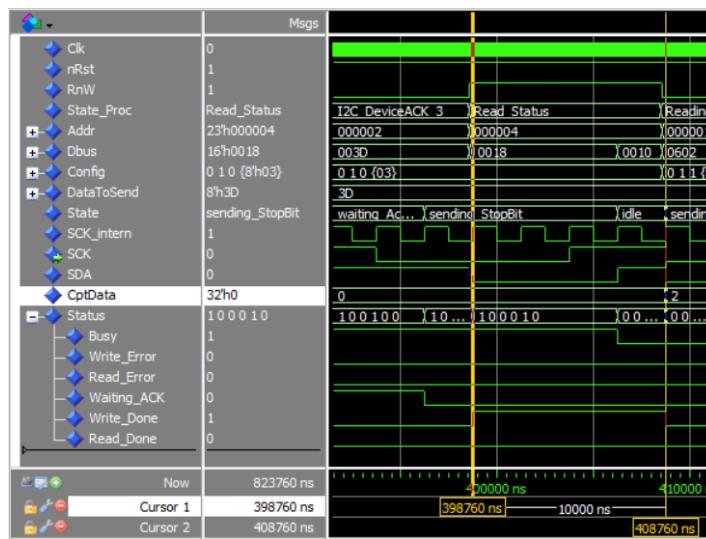


Figure 36: Lecture de l'état de l'IP par le processeur

Le processeur effectue un cycle de lecture du registre I2C_Status qui se trouve à l'adresse Addr = 4. Le contenu de I2C_Status est lu sur le Dbus dont la valeur est égale à 0x0018. Le champ « Write_Done » passe à '1' pour indiquer que l'envoi de la trame s'est bien déroulé et que le circuit est prêt à lancer une deuxième transmission de données.

5.4 Validation de la réception des données par le contrôleur I2C

Dans cette partie, nous procédons à la validation de la réception d'une trame de données par le contrôleur I2C. La figure suivante présente **la réception d'une trame I2C** composée de **deux octets** en mode de **transmission rapide** en format **Big Endian**.

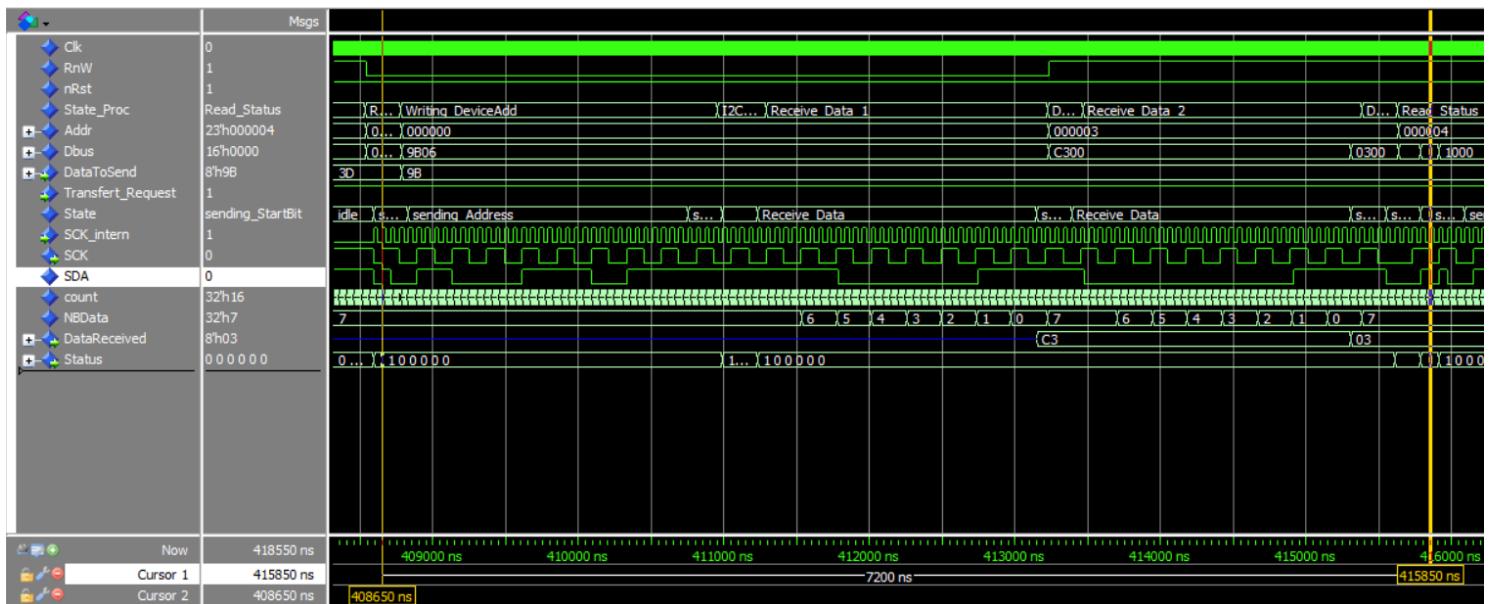


Figure 37: Une trame I2C reçue par le contrôleur I2C

Pour recevoir deux octets de données, la trame I2C est composée de 30 bits (un bit Start, 8 bits d'adresse, bit RnW, 16 bits de données, 3 bits d'acquittement, un bit Stop) dont la période de transmission rapide est de 240 ns ce qui fait qu'une trame nécessite 7200 ns comme le montre la figure ci-dessus (la trame est limitée par les deux curseurs). Nous commençons alors à vérifier les champs de la trame.

5.4.1 Configuration du contrôleur I2C pour la réception de données

Pour assurer la réception de données, le processeur doit configurer le contrôleur I2C. Le chronogramme ci-dessous montre un cycle d'écriture par le processeur dans le registre de configuration du circuit I2C_Config.

Avant T1, le Dbus contient la valeur du registre Status lue à la fin de l'émission de la trame I2C.

T1 : Le processeur entre dans l'état Reading_config. Il écrit la valeur 0x0204 à l'adresse Addr=1 qui correspond à l'adresse du registre I2C_Config

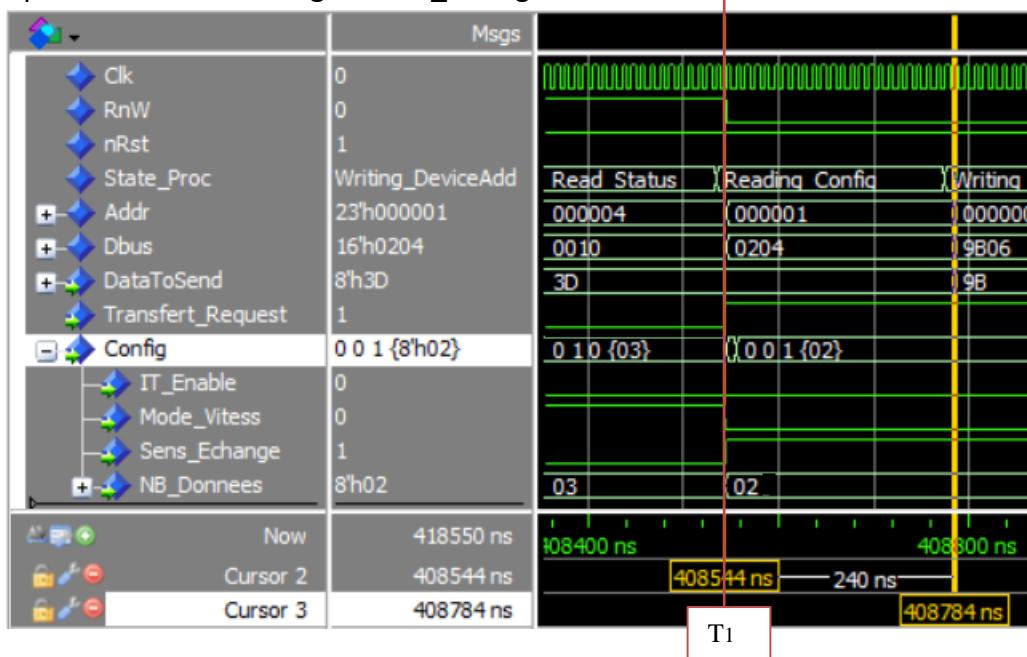


Figure 38 : Configuration du contrôleur I2C pour recevoir les données

En observant le contenu du bus Config, nous pouvons déduire que la valeur 0x0204 correspond à la réception de deux octets à un dispositif I2C sans générer d'interruption avec une vitesse de transmission rapide de 4000 kbits/s.

Une fois la configuration du circuit effectuée par le processeur suite à l'écriture dans le registre I2C_Config, la fonction "Write" de l'interface Processeur déclenche une demande de transfert via le signal Transfer_Request destiné à la fonction de génération d'horloge interne qui contrôlera la transmission des données via le signal d'horloge interne SCK_intern. La réception des données par le contrôleur I2C est déclenché, nous pouvons procéder à la validation des champs nécessaires à une trame I2C.

5.4.2 Validation du bit Start

Comme défini avant la condition pour démarrer le transfert de données est que le signal SDA passe de l'état haut à l'état bas avant que SCK ne passe de l'état haut à l'état bas (SCK=1, SDA=0).

L'envoi du bit de départ se fait en un cycle d'horloge de SCK comme il est limité sur la figure par les deux curseurs.

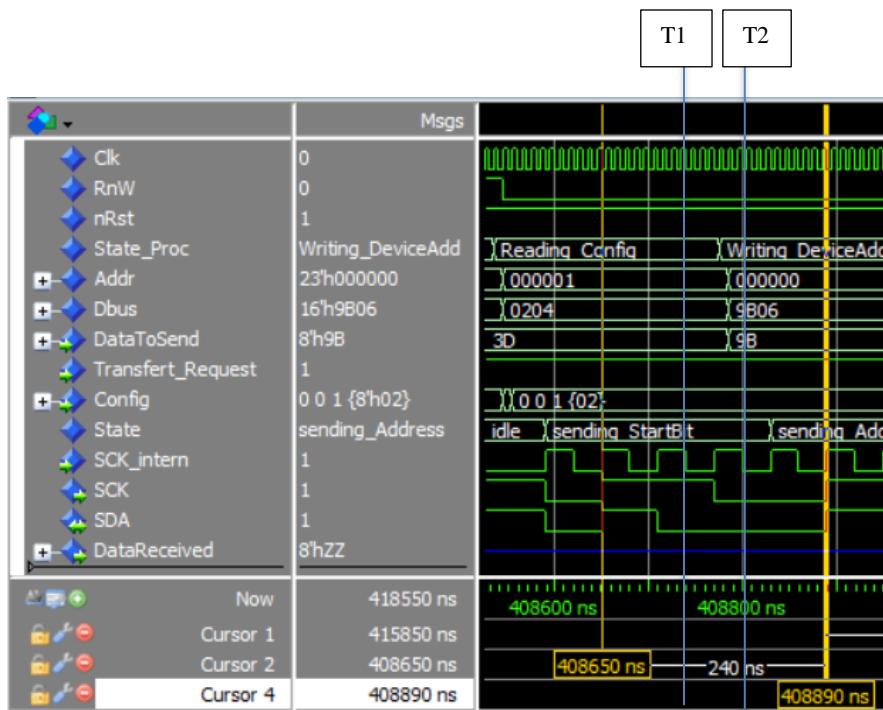


Figure 39: Bit Start de la trame I2C pour recevoir les données

Les deux signaux SDA et SCK démarrent à l'état haut.

T1 = $T_{sck}/4$: SDA passe de l'état haut à l'état bas et SCK est maintenu au niveau haut.

T2 = $3T_{sck}/4$: le passage de SCK de l'état haut à l'état bas et SDA est maintenu au niveau bas.

⇒ La condition de début d'une trame I2C est vérifiée.

5.4.3 La transmission de l'adresse

Pour recevoir les données, la trame I2C doit définir l'adresse de l'esclave I2C émetteur.

L'adresse d'un circuit, codée sur huit bits. Cette adresse est transmise en commençant par le bit le plus significatif(MSB). La transmission de l'adresse nécessite 1920 ns comme le montre la figure ci-dessous.

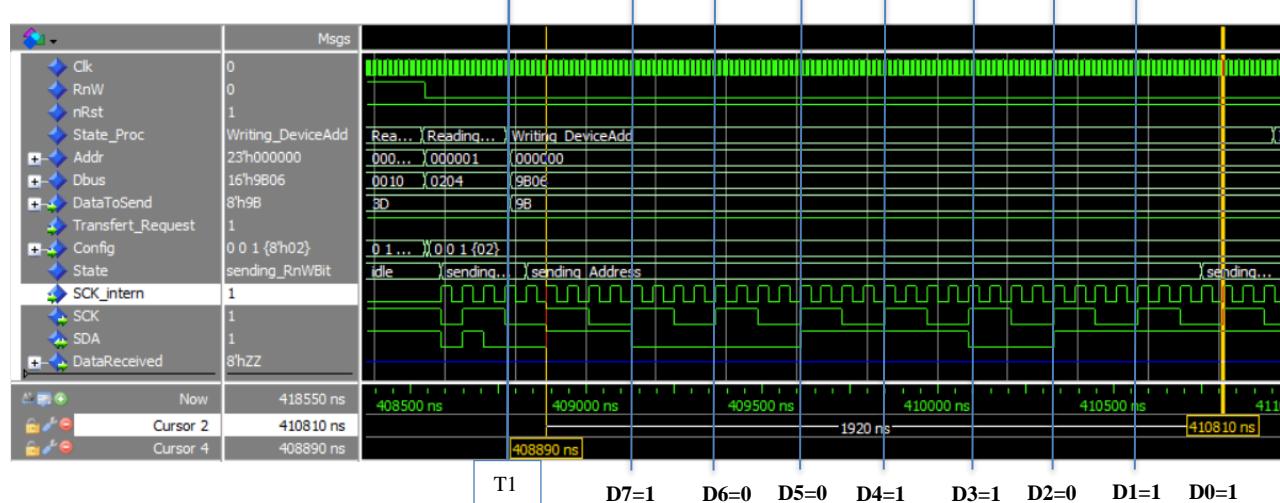


Figure 40: Emission de l'adresse du périphérique émetteur

Avant de transmettre l'adresse, le processeur doit écrire cette adresse dans le registre I2C_Addr. T1 : un cycle d'écriture par le processeur est déclenché. Le contenu de Dbus 0x9B06 est écrit à l'adresse Addr=0, ce qui fait que le contenu du bus DataToSend =9B vu que le processeur adopte le format Big Endian pour les données.

- ⇒ Les huit bits qui suivent le bit Start correspondent à la valeur de l'adresse 9B définie par le processeur.

5.4.4 La transmission du bit RnW

Après l'envoi du Start bit et de l'adresse, le contrôleur I2C doit envoyer le bit RnW qui permet de signaler s'il veut lire ou écrire une donnée

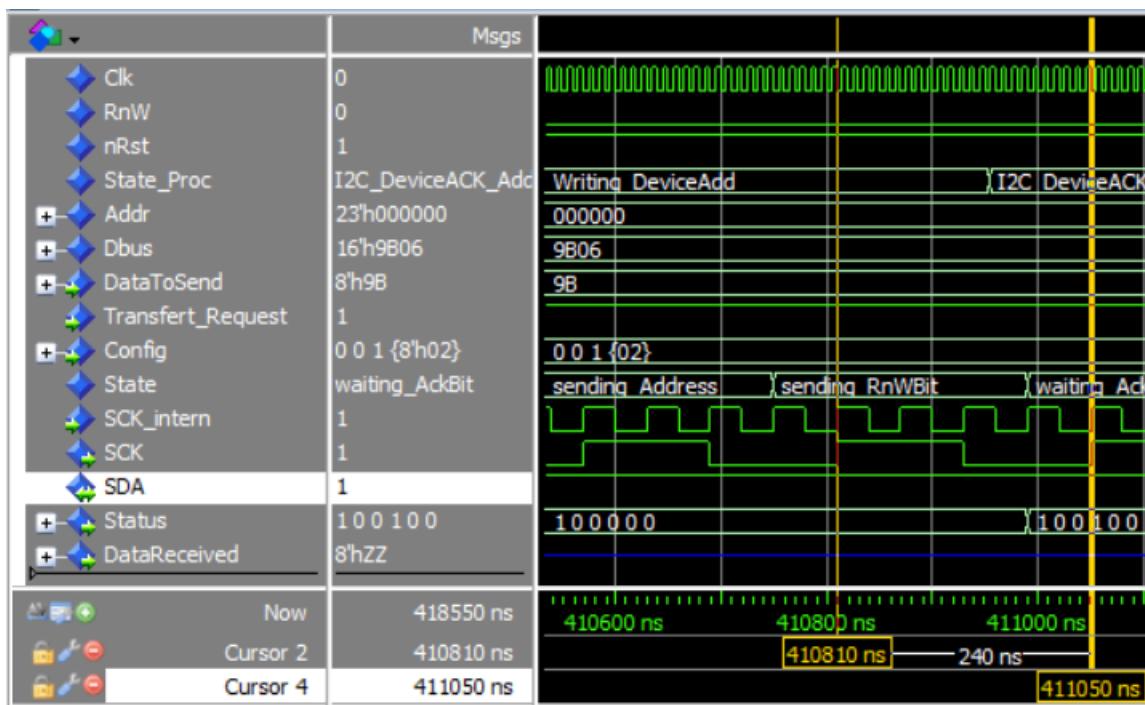


Figure 41: Emission du bit RnW

- ⇒ Il s'agit d'envoi de données donc le bit RnW est égale à 1.
- ⇒ L'état du circuit est Busy puisque l'envoi de la trame est en cours.

5.4.5 Le bit d'accusé de réception de l'adresse

Après l'envoi de l'adresse et du bit RnW, le contrôleur I2C (maître) doit vérifier la disponibilité du périphérique I2C (esclave) en attendant le bit d'acquittement ACK.

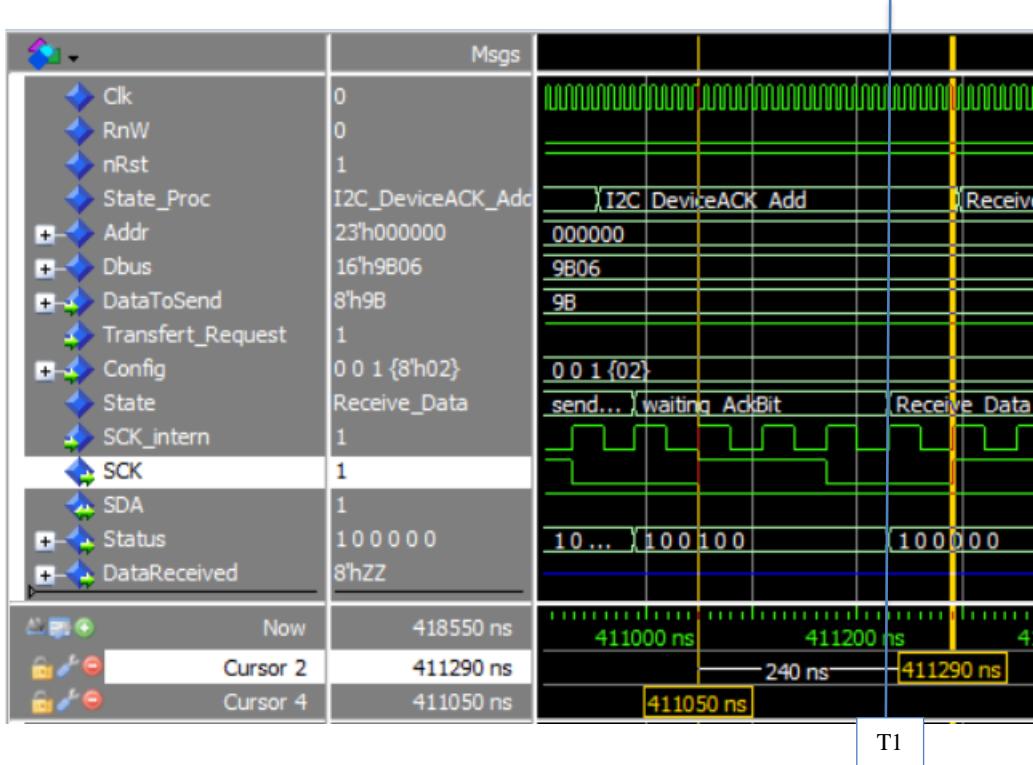


Figure 42: Réception du bit d'accusé de réception d'adresse

Le contrôleur I2C attend le bit d'accusé de réception d'adresse (Status=100100) pour vérifier la disponibilité de l'esclave I2C. Les données ne peuvent être reçus par l'IP que si le signal SDA est à l'état haut. La vérification de l'état du signal SDA par le circuit se fait à partir de $3T_{sck}/4$.

T1 : Le périphérique I2C est disponible, l'IP a détecté le bit d'acquittement et il a changé de statut (Status =100000).

⇒ Le contrôleur I2C peut commencer à recevoir des données.

5.4.6 La réception de données

Pour rappel, le contrôleur I2C doit recevoir 2 octets envoyés par le périphérique I2C (esclave). La figure suivante montre la réception d'un octet de données suivi de l'envoi du bit d'accusé de réception par l'IP.

➤ La réception du premier octet :

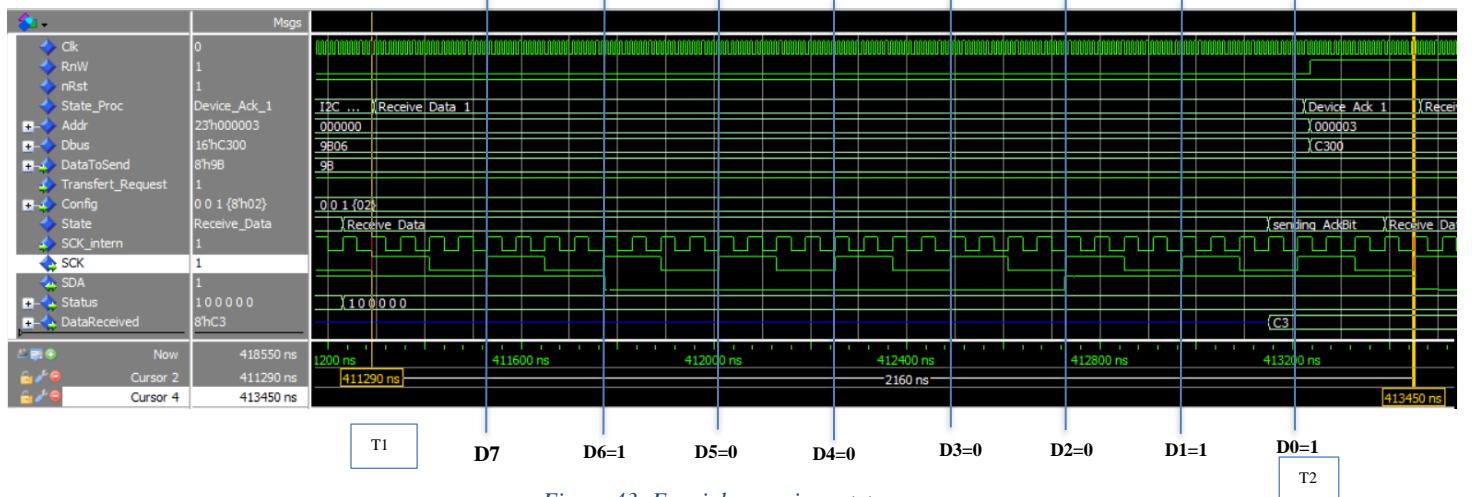


Figure 43: Envoi du premier octet

T1 : Le contrôleur I2C démarre un cycle de réception de données qui les stocke dans le registre I2C_DataReceived qui est situé à l'adresse Addr=3. Les bits de 0 à 7 sont envoyés par l'esclave I2C sur le signal SDA. Le processeur reste dans l'état Receive_Data_1 pendant 1920 ns afin que le comportement de l'interface soit synchronisé avec la fonction Emission_Reception puisque $T_{sck}=25*T_{Clk}$.

]T1 , T2[: le contrôleur I2C, fonctionnant en mode maître, reçoit le bit de poids fort D7 des données par l'intermédiaire du signal SDA. Il répète l'opération jusqu'à la réception de l'octet complet comme le montre la figure ci-dessus, ce qui correspond au contenu de dataReceived qui est égal à C3.

T2 : Après avoir lu un octet, l'IP met SDA à '1' pour informer l'esclave qu'il a reçu le premier octet et qu'il veut lire la donnée suivante.

➤ La réception du deuxième octet :

La figure suivant présente la réception du deuxième octet envoyé par le périphérique I2C

suivi de l'envoi du bit d'acquittement.

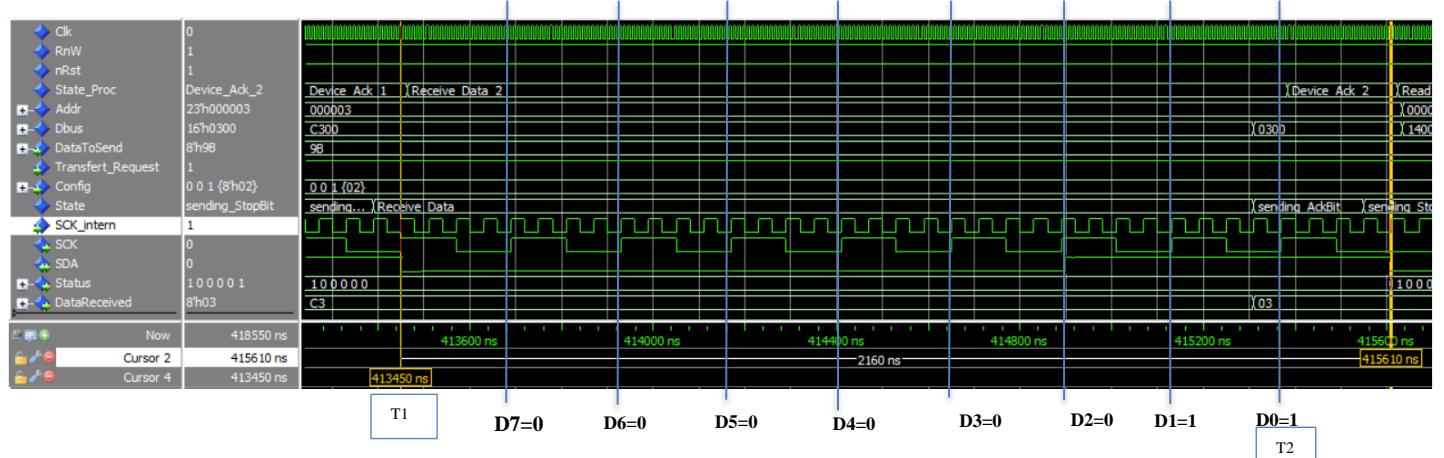


Figure 43: Réception du deuxième octet par l'IP

L'analyse est identique à la précédente et permet la réception du dernier octet. L'IP a imposé le niveau '1' sur le signal SDA à la fin de la réception des sept bits. L'octet envoyé est égal à 03.

⇒ Le contrôleur I2C peut envoyer le bit d'arrêt pour déclarer la fin de la trame I2C.

5.4.7 Validation du bit Stop

La condition pour arrêter le transfert de données est que le signal SDA passe de l'état bas à l'état haut après que SCK passe de l'état bas à l'état haut (SCK=1, SDA=1). L'envoie de ce bit de correspond à l'état Sending_StopBit du circuit comme le montre la figure suivante :

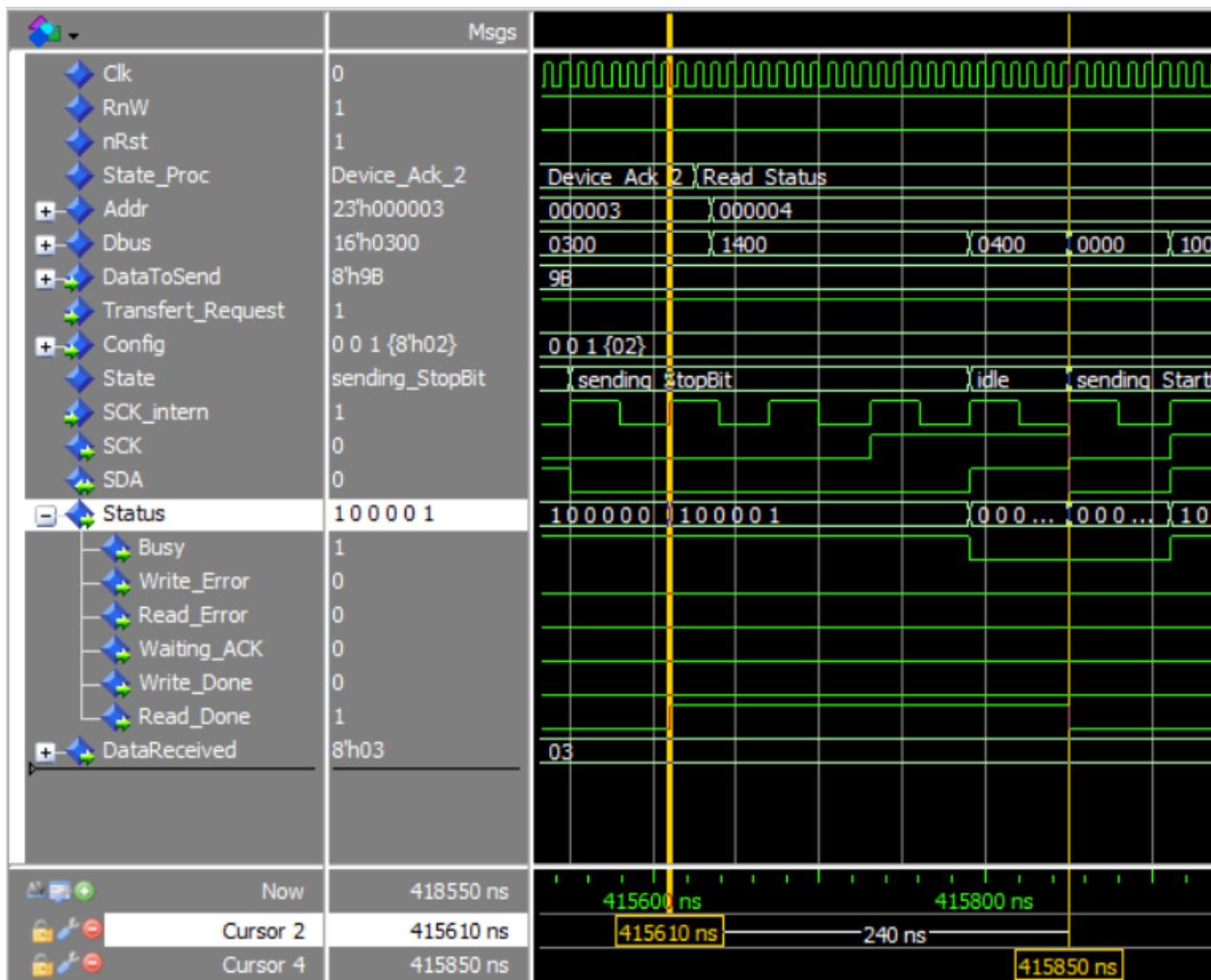


Figure 44: Envoi du bit Stop à la fin de la réception de données

Après la fin de la transmission de la trame I2C, le processeur effectue une lecture de l'état du contrôleur I2C. Le processeur effectue un cycle de lecture du registre I2C_Status qui se trouve à

l'adresse Addr = 4. Le contenu de I2C_Status est lu sur le Dbus dont la valeur est égale à 0x1400 (format Big Endian). Le champ « Read_Done » passe à '1' pour indiquer que la réception de données s'est bien déroulée et que le circuit est prêt à lancer une deuxième transmission de données.

5.5 Validation de cas d'erreur

T1 : En supposant qu'après avoir envoyé le bit Start, les bits d'adresse et le bit RnW, le contrôleur I2C n'a pas reçu le bit d'acquittement après un temps de T_{SCK}. Dans ce cas, la transmission des données est interrompue et une erreur d'écriture se produit. Le contenu du registre I2C est égal à 110.000 comme le montre la figure ci-dessous.

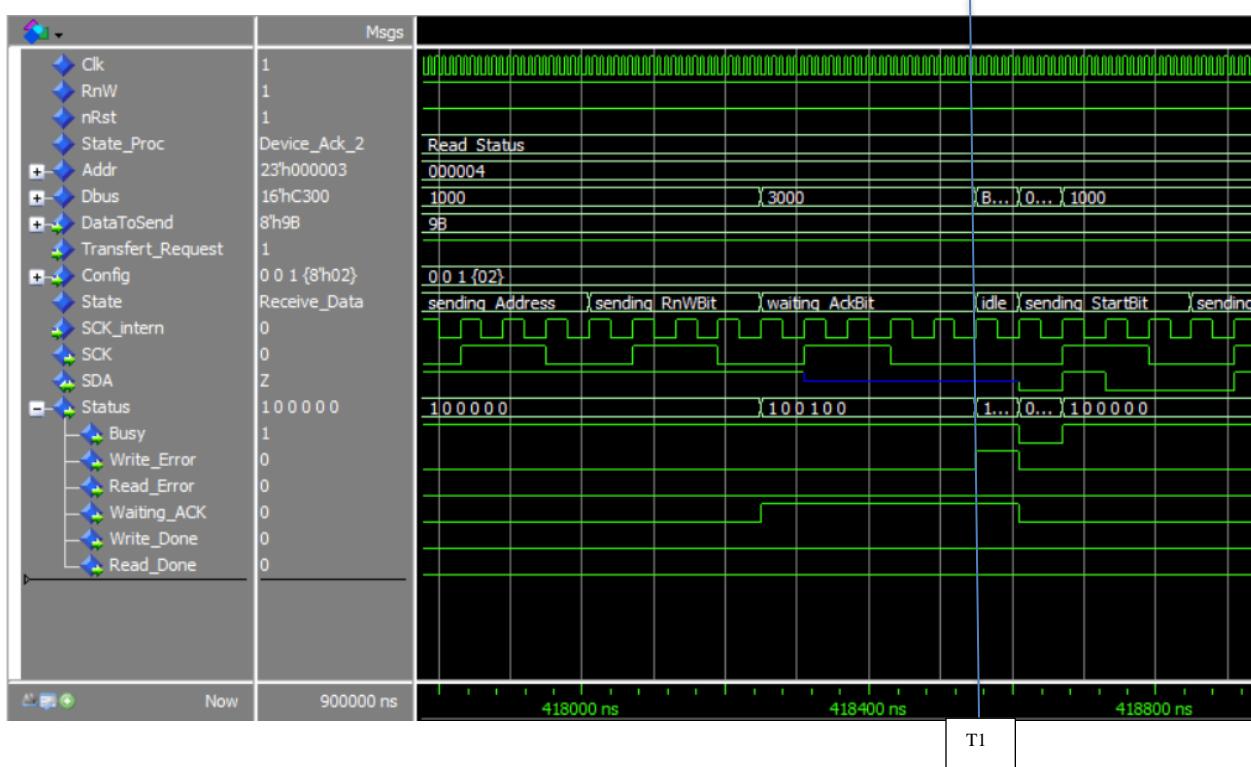


Figure 45: Erreur d'écriture : non-réception du bit d'acquittement

6. Résultat de synthèse

Dans cette partie, nous allons vérifier les résultats de synthèse réalisés avec Precision Synthesis. Le but est principalement de vérifier le nombre et le type de ressources utilisées : LUTs, bascules.

6.1 Codage de la machine à état

Le résultat de l'encodage de la machine à états du circuit fourni par Precision Synthesis est donné par la figure suivante. L'encodage utilisé est du type One-Hot appelé aussi encodage 1 de n, il consiste à encoder la variable Etat avec n états sur n bits dont un seul prend la valeur 1, le nombre de bit valant 1 étant le numéro de l'état pris par la variable.

FSM EXTRACTION ANALYSIS			
=====			
=====			
Module : Projet_I2CIP_lib.Z_Emission_Reception(arch)			
=====			
Number of FSMs Extracted : 1			
1.			

Number of States : 9			
Primary State Variable : State[8:0]			
Async set/reset state(s) : 00000001			
Re-encoding Scheme : ONEHOT			
FSM: Info, state encoding table:			
FSM: Index State Name Literal Encoding			
FSM: 0	idle	00000001	00000001
FSM: 1	sending_StartBit	00000010	00000010
FSM: 2	sending_Address	00000100	00000100
FSM: 3	sending_RnWBit	000001000	000001000
FSM: 4	waiting_AckBit	000010000	000010000
FSM: 5	Send_Data	000100000	000100000
FSM: 6	Receive_Data	001000000	001000000
FSM: 7	sending_AckBit	010000000	010000000
FSM: 8	sending_StopBit	100000000	100000000

Figure 46 : Codage de la machine à états par Precision Synthesis

Le principal avantage de ce type de codage est que pour passer d'un état à un autre, seules deux transitions sont nécessaires : un chiffre passe de 1 à 0 et un autre de 0 à 1. Une seule bascule est nécessaire pour chaque état, donc pour le contrôleur I2C 9 bascules sont nécessaires.

6.2 Résultat des ressources utilisées

L'analyse des fichiers produits par Precision Synthesis permet de connaître le nombre de ressources utilisées dans le FPGA Xilinx Zynq 7000. En particulier, on constate que le nombre de bascules utilisées est de 126 bascules qui représente moins de 1% du nombres totales des bascules disponibles dans le FPGA. Elles sont réparties comme suit :

- 35 bascules pour la fonction **ClockGenerator**,
- 72 bascules pour la fonction **Emission_Réception**,
- 19 bascules pour la fonction **Write** de l'interface du processeur.

La fonction **Read** de l'interface du processeur ne présente pas de bascules (processus combinatoire).

Device Utilization for 7Z010CLG400			
Resource	Used	Avail	Utilization
IOs	48	100	48.00%
Global Buffers	1	32	3.13%
LUTs	173	17600	0.98%
CLB Slices	23	4400	0.52%
Dffs or Latches	126	35200	0.36%
Block RAMs	0	60	0.00%
DSP48E1s	0	80	0.00%

Library: projet_i2cip_lib	Cell: Z_I2C_IP	View: struct

Cell	Library	References
BUFGP	xcz7	1 x
GND	xcz7	1 x
IBUF	xcz7	28 x
IOBUF	xcz7	17 x
LUT4	xcz7	1 x 1 1 LUTs
LUT5	xcz7	1 x 1 1 LUTs
OBUF	xcz7	1 x
VCC	xcz7	1 x
Z_ClockGenerator_0	projet_i2cip_lib	1 x 35 35 Dffs or Latches 46 46 LUTs 31 31 MUX CARRYs
Z_Emission_Reception_notri_0	projet_i2cip_lib	1 x 72 72 Dffs or Latches 31 31 MUX CARRYs 101 101 LUTs
Z_Write_notri_0	projet_i2cip_lib	1 x 24 24 LUTs 19 19 Dffs or Latches
Number of ports :		48
Number of nets :		153
Number of instances :		54
Number of references to this view :		0
Total accumulated area :		
Number of Dffs or Latches :		126
Number of LUTs :		173
Number of Primitive LUTs :		200
Number of LUTs with LUTNM/HLUTNM :		54
Number of MUX CARRYs :		62
Number of accumulated instances :		510

Figure 47: Résultats des ressources utilisées dans le FPGA Xilinx 7Z010CLG400

Le nombre d'entrées sorties nécessaires est de 48, si on fait le calcul du nombre total d'entrées sorties, on trouve effectivement 48.

En termes de ressources en LUTs, le circuit occupe 0,98% des ressources disponibles dans le FPGA. Cela représente l'équivalent de 173 LUTs. Nous pouvons donc facilement conclure que l'implémentation du système, avec un processeur et les autres IPs conçus par les autres groupes, ne posera aucun problème.

6.3 Analyse de synchronisation statique

L'analyse de synchronisation statique (STA) est une méthode de validation des performances de synchronisation d'un IP par la vérification de tous les chemins possibles pour les violations de synchronisation. Dans cette phase, le contrôleur I2C sera décomposé en chemins de synchronisation, Precision Synthesis se charge de calculer le délai de propagation du signal le long de chaque chemin et vérifie les violations des contraintes de synchronisation des E/S.

La figure suivante présente un extrait du rapport de violation de synchronisation qui indique que toutes les contraintes de port d'E/S sont respectées.

Clock Constraint Violations			
Clock Name	Constrained Period	Slack	
All clock frequency constraints were met; no violations			
Input Constraint Violations			
Port Name	Input Constraint	Clock	Slack
All input constraints were met; no violations			
Output Constraint Violations			
Port Name	Output Constraint	Clock	Slack
All output constraints were met; no violations			

Figure 48: Timing Violation Report

Dans l'analyse de synchronisation statique(STA), le **slack** indique si le timing est respecté le long d'un chemin de synchronisation. Un jeu positif signifie que le signal peut se rendre du point de départ au point d'arrivée du chemin suffisamment rapidement pour que le circuit fonctionne correctement. Un jeu négatif signifie que le signal de données n'est pas en mesure de traverser la logique combinatoire entre le point de départ et le point d'arrivée du chemin assez rapidement pour assurer le bon fonctionnement du circuit.

En analysant le rapport de timing, il apparaît qu'il y a deux chemins critiques avec un jeu positif (Path slack) :

- Chemin critique N°1 : se situe au niveau du block ClockGenerator. Le point de départ est le signal count-max à la valeur maximale du diviseur de fréquence et le point d'arrivée est le registre count avec un slack positif égal à 2.725, ce qui signifie que le signal de génération d'horloge interne de l'IP peut aller du point de départ au point d'arrivée du chemin assez rapidement pour synchroniser la transmission des données via SDA.

Setup Timing Analysis of Clk									
Setup Slack Path Summary									
Index	Setup	Path	Source	Clock	Dest.	Data Start Pin	Data End Pin	Data	Logic
Index	Slack	Delay	Source	Clock	Dest.	Data Start Pin	Data End Pin	End Edge	Logic Levels
1	2.725	2.705	U_2/ix792z1361/0	Clk	U_2/lat_count_max(1)/G	U_2/reg_count(1)/R		Rise	3
2	2.725	2.705	U_2/ix792z1361/0	Clk	U_2/lat_count_max(2)/G	U_2/reg_count(1)/R		Fall	3
3	4.576	5.075	Clk	Clk	U_2/reg_count(0)/C	U_2/reg_count(1)/R		Fall	36
4	4.605	5.046	Clk	Clk	U_2/reg_count(1)/C	U_2/reg_count(1)/R		Fall	35
5	4.634	5.017	Clk	Clk	U_2/reg_count(2)/C	U_2/reg_count(1)/R		Fall	34
6	4.663	4.988	Clk	Clk	U_2/reg_count(3)/C	U_2/reg_count(1)/R		Fall	33
7	4.692	4.959	Clk	Clk	U_2/reg_count(4)/C	U_2/reg_count(1)/R		Fall	32
8	4.721	4.930	Clk	Clk	U_2/reg_count(5)/C	U_2/reg_count(1)/R		Fall	31
9	4.750	4.901	Clk	Clk	U_2/reg_count(6)/C	U_2/reg_count(1)/R		Fall	30
10	4.779	4.872	Clk	Clk	U_2/reg_count(7)/C	U_2/reg_count(1)/R		Fall	29
CTE Path Report									
Critical path #1, (path slack = 2.725), Logic Levels = 3									
SOURCE CLOCK: name: U_2/ix792z1361/0 period: 10.000000									
Times are relative to the 1st falling edge									
DEST CLOCK: name: Clk period: 10.000000									
Times are relative to the 2nd rising edge									
NAME GATE DELAY ARRIVAL DIR FANOUT LEVEL									
U_2/lat_count_max(1)/G	LDCE				0.000	dn			
U_2/lat_count_max(1)/Q	LDCE	0.419			0.419	up			
U_2/count_max(1)	(net)	0.333					1	0	
U_2/ix5077z1530/I1	LUT3				0.752	up			
U_2/ix5077z1530/O	LUT3	0.124			0.876	up			
U_2/nx5077z19	(net)	0.333					1	1	
U_2/ix5077z1319/I0	LUT5				1.209	up			
U_2/ix5077z1319/O	LUT5	0.124			1.333	up			
U_2/nx5077z18	(net)	0.354					4	2	
U_2/ix5077z17699/I2	LUT4				1.687	up			
U_2/ix5077z17699/O	LUT4	0.124			1.811	up			
U_2/nx5077z3	(net)	0.894					31	3	
U_2/reg_count(1)/R	FDRE				2.705	up			
Initial edge separation: 5.000									
Source clock delay: - 0.335									
Dest clock delay: + 1.114									

Edge separation: 5.779									
Setup constraint: - 0.349									

Data required time: 5.430									
Data arrival time: - 2.705 (29.24% cell delay, 70.76% net delay)									

Slack: 2.725									

Figure 49: 1^{er} chemin critique (Timing Report)

- Chemin critique N°2 : se situe au niveau du block Emission_Reception. Le point de départ est le signal de configuration Config précisément le champ responsable à définir le nombre de données à transmettre et le point d'arrivée est registre CptData avec un slack positif égal à 8,716.

```

Setup Timing Analysis of U_2/reg_tmp/Q

Setup Slack Path Summary

      Data
      Setup Path
Index Slack Delay Source Clock   Dest. Clock   Data Start Pin   Data End Pin   Data
      -----   -----   -----   -----   -----   -----   -----   -----   -----
 1  8.716  1.268  Clk    U_2/reg_tmp/Q  U_0_U_1/reg_Config_NB_Donnees(7)/C  U_1/reg_CptData(7)/D  Rise   1
 2  8.716  1.268  Clk    U_2/reg_tmp/Q  U_0_U_1/reg_Config_NB_Donnees(6)/C  U_1/reg_CptData(6)/D  Rise   1
 3  8.716  1.268  Clk    U_2/reg_tmp/Q  U_0_U_1/reg_Config_NB_Donnees(5)/C  U_1/reg_CptData(5)/D  Rise   1
 4  8.716  1.268  Clk    U_2/reg_tmp/Q  U_0_U_1/reg_Config_NB_Donnees(1)/C  U_1/reg_CptData(1)/D  Rise   1
 5  8.716  1.268  Clk    U_2/reg_tmp/Q  U_0_U_1/reg_Config_NB_Donnees(0)/C  U_1/reg_CptData(0)/D  Rise   1
 6  8.716  1.268  Clk    U_2/reg_tmp/Q  U_0_U_1/reg_Config_NB_Donnees(3)/C  U_1/reg_CptData(3)/D  Rise   1
 7  8.716  1.268  Clk    U_2/reg_tmp/Q  U_0_U_1/reg_Config_NB_Donnees(2)/C  U_1/reg_CptData(2)/D  Rise   1
 8  8.716  1.268  Clk    U_2/reg_tmp/Q  U_0_U_1/reg_Config_NB_Donnees(4)/C  U_1/reg_CptData(4)/D  Rise   1
 9  16.527 3.457 U_2/reg_tmp/Q U_2/reg_tmp/Q  U_1/reg_CptData(0)/C           U_1/reg_CptData(31)/D  Rise   33
10  16.891 3.093 U_2/reg_tmp/Q U_2/reg_tmp/Q  U_1/reg_CptData(1)/C           U_1/reg_CptData(31)/D  Rise   32

      CTE Path Report

Critical path #1, (path slack = 8.716):, Logic Levels = 1

SOURCE CLOCK: name: Clk period: 10.000000
  Times are relative to the 2nd rising edge
DEST CLOCK: name: U_2/reg_tmp/Q period: 20.000000
  Times are relative to the 2nd rising edge

NAME          GATE      DELAY     ARRIVAL DIR  FANOUT LEVEL
U_0_U_1/reg_Config_NB_Donnees(7)/C FDCE    0.000    up
U_0_U_1/reg_Config_NB_Donnees(7)/Q FDCE    0.478    up
U_0_U_1/Config_NB_Donnees(7)      (net)    0.333
U_0_U_1/ix30895z1498/I0        LUT3     0.811    up
U_0_U_1/ix30895z1498/O        LUT3     0.935    up
U_0_U_1/nx30895z1               (net)    0.333    1      1
U_1/reg_CptData(7)/D           FDRE    1.268    up

Initial edge separation: 10.000
Source clock delay: - 1.114
Dest clock delay: + 1.114
-----
Edge separation: 10.000
Setup constraint: - 0.016
-----
Data required time: 9.984
Data arrival time: - 1.268  ( 47.48% cell delay, 52.52% net delay )
-----
Slack: 8.716

```

Figure 50: 2^{ème} chemin critique (Timing Report)

6.4 Résultats graphiques de la synthèse

6.4.1 Schéma RTL de l'IP

Le schéma RTL montre les 3 blocks : ClockGenerator, Emission_Reception et Interface. Il montre également les buffers tri-states pour les signaux du contrôleur I2C (DataReceived , DataToSend, SDA) et le bus du processeur Dbus.

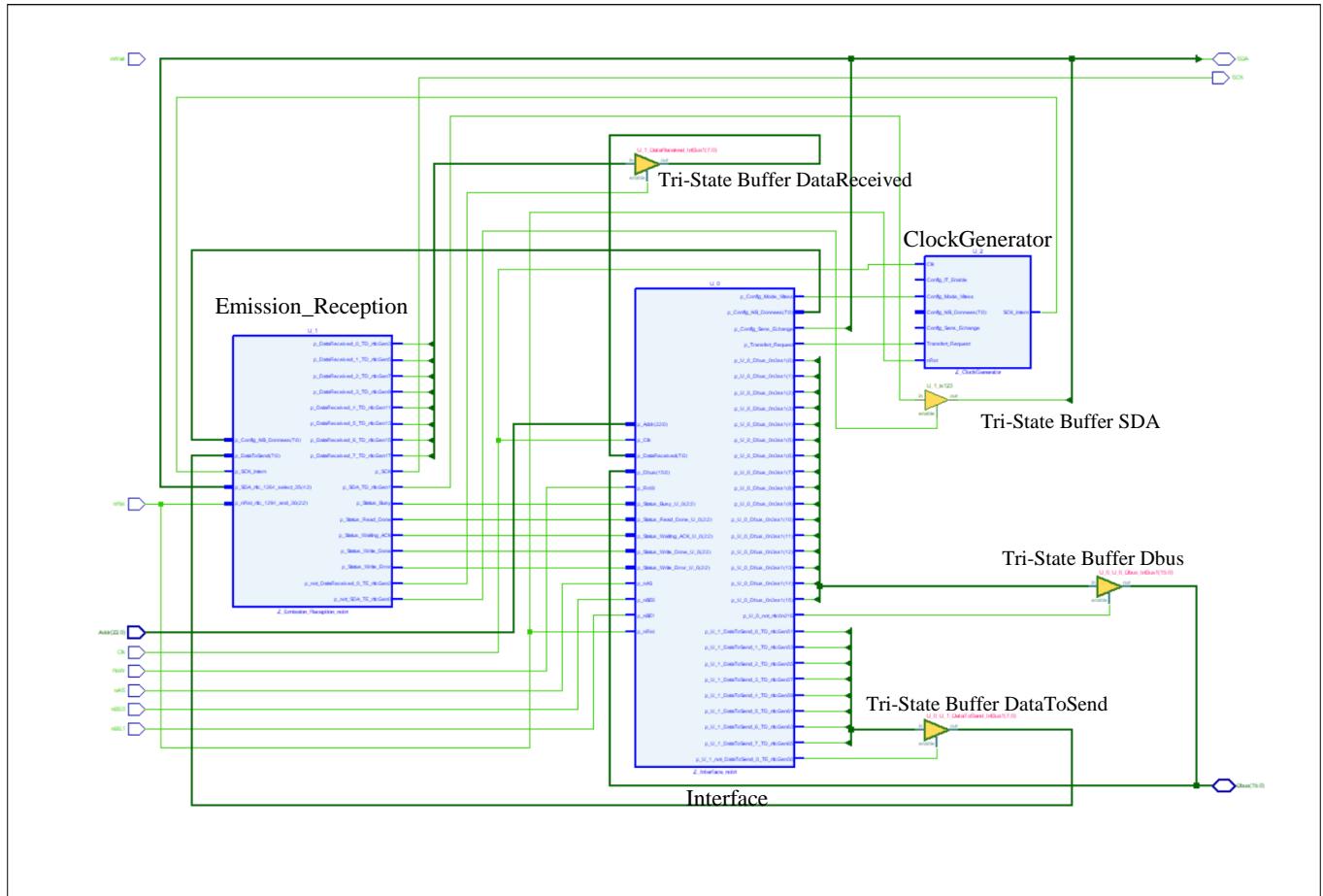


Figure 51: RTL Schématique du contrôleur I2C

➤ Schéma RTL du block Interface

Les figures ci-dessous montrent les deux fonctions Write et Read qui assurent la communication entre le processeur et le contrôleur I2C.

On remarque bien que la fonction Write possède des bascules D pour sauvegarder les données de configuration de l'IP, les données à transmettre et les données à recevoir (voir Figure 53) et que la fonction Read de l'interface du processeur ne possède pas de bascules vu qu'elle s'agit d'un processus combinatoire.

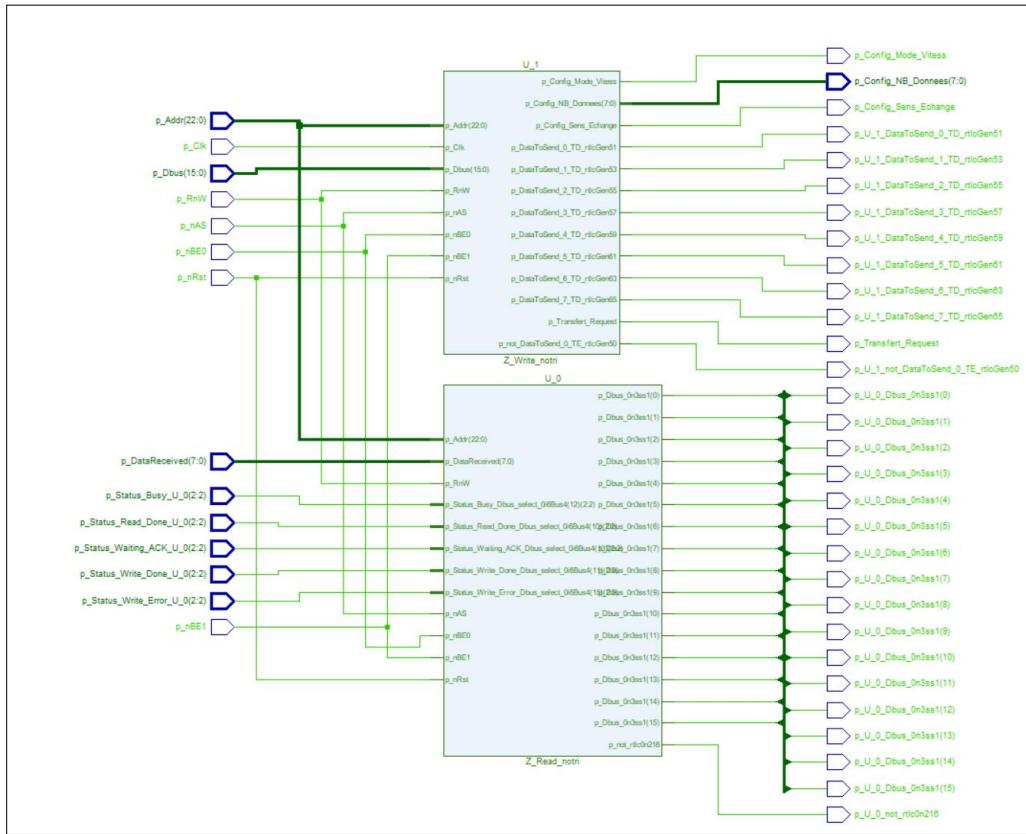


Figure 52:RTL Schématique de l'Interface du processeur

- Schéma RTL de la fonction Write

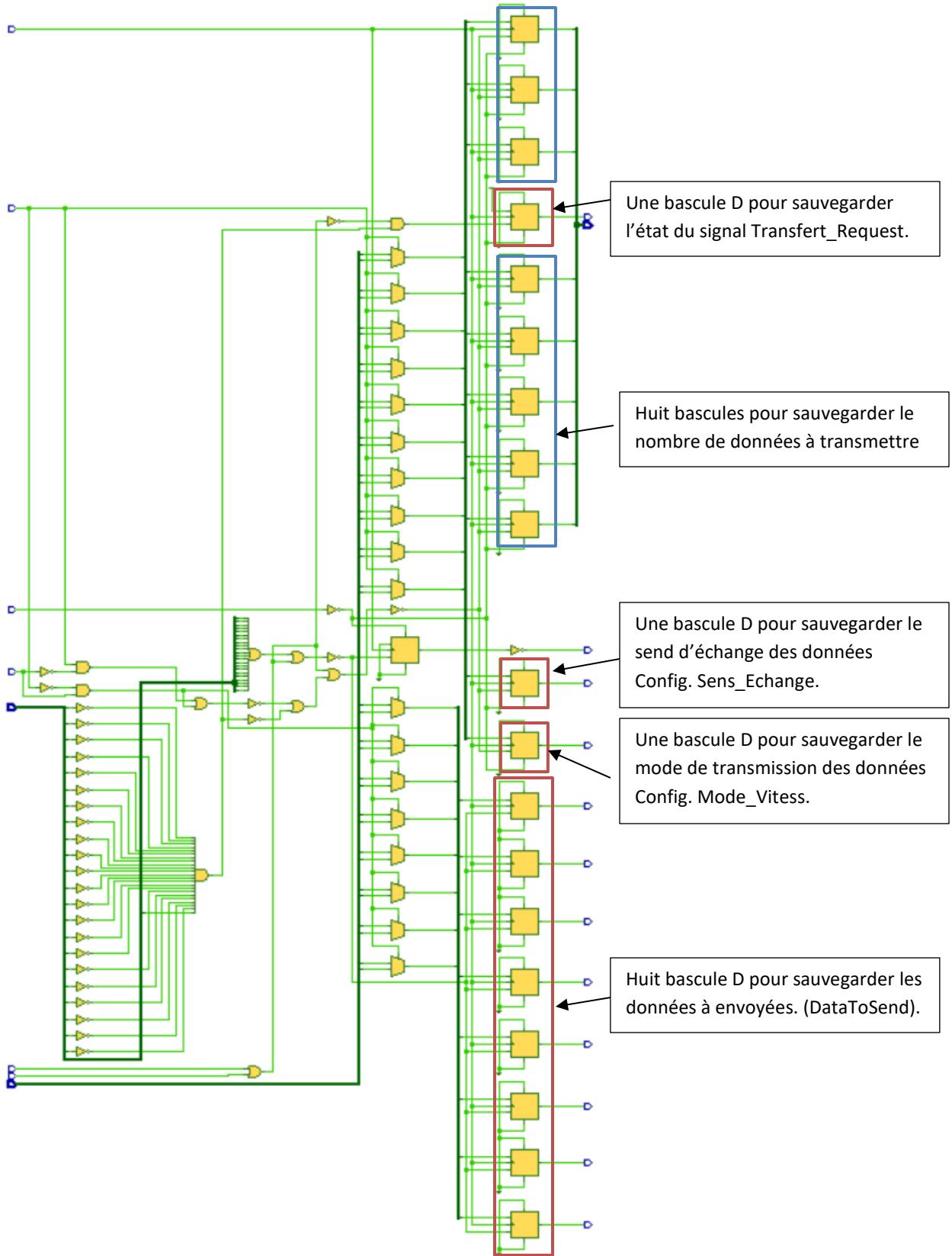


Figure 53: Schéma RTL de la fonction Write

- Schéma RTL de la fonction **Read**

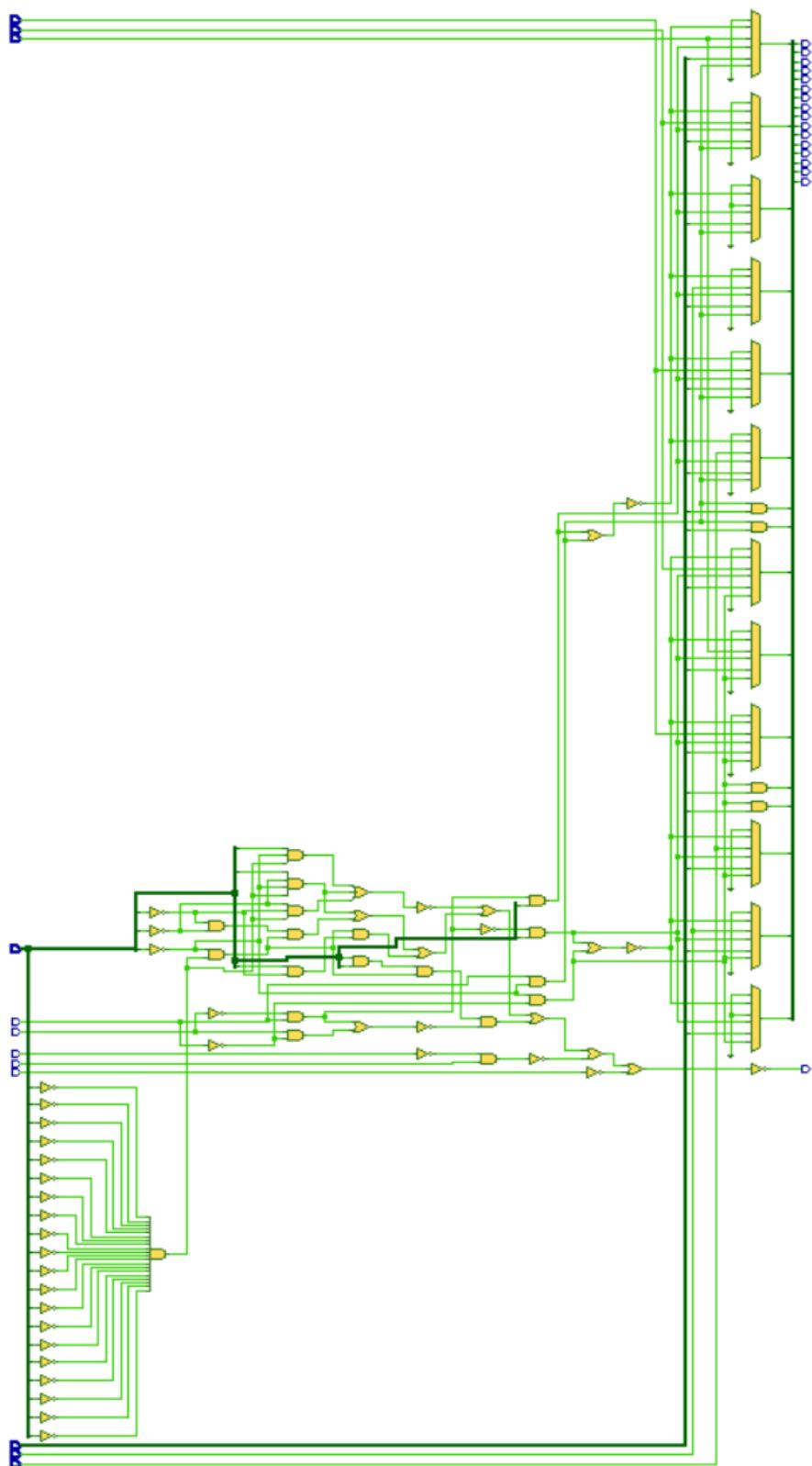


Figure 54: Schéma RTL de la fonction Read

➤ Schéma RTL du block ClockGenerator

Voir Annexe A.7.

➤ Schéma RTL du block Emission_Reception

Voir annexe A.8.

Globalement, la synthèse donne les résultats attendus, on retrouve bien les bascules de sauvegardes des données et les registres internes. On retrouve également les buffers tri-states pour les signaux du contrôleur I2C et le bus du processeur Dbus.

7. Conclusion

Ce projet présente la conception d'un circuit numérique basé sur l'approche MCSE. Le circuit à concevoir est un contrôleur I2C dont le rôle est d'assurer le couplage du microprocesseur avec des modules externes via une liaison I2C. Le circuit final permet de valider les spécifications initiales. L'approche de conception MSCE a été appliquée pour aboutir à une solution qui peut être implémentée dans un Xilinx Zynq 7000 FPGA. A l'issue de la saisie de la solution sous HDL Designer, l'étude en simulation de chronogrammes du circuit a permis de valider le comportement fonctionnel du circuit défini dans le cahier des charges.

La phase de synthèse a montré les ressources logiques nécessaires à la réalisation de l'IP, ce qui a permis de vérifier la cohérence des ressources par rapport à la spécification du système.

A. Annexes

A.1 Code VHDL du package utilisé pour le projet

```
--  
-- VHDL Package Header Projet_I2CIP_lib.I2C_Package  
--  
-- Created:  
--     by - E21C396C.LAGHA (irc107-04)  
--     at - 13:49:57 17/11/2022  
--  
-- using Mentor Graphics HDL Designer(TM) 2022.1  
--  
LIBRARY ieee;  
USE ieee.std_logic_1164.all;  
USE ieee.numeric_std.ALL;  
PACKAGE I2C_Package IS  
  
    --Constants for the project  
    constant AddrBusSize : integer := 23;  
    constant DataBusSize : integer := 16;  
    constant DataSize : integer := 8;  
    constant Nb_BitDonnees :integer := 8;  
    constant EndiannessSize : integer :=2;  
    constant Nb_CycleStandadard :integer:=999;  
    constant Nb_CycleRapide :integer :=23;  
  
    -- Constants for endianism of processeur  
    constant BIG_ENDIAN : std_logic_vector ( EndiannessSize -1 downto 0 ) := "01";  
    constant LITTLE_ENDIAN : std_logic_vector ( EndiannessSize -1 downto 0 ) := "10";  
    constant EIGHT_BITS : std_logic_vector ( EndiannessSize -1 downto 0 ) := "00";  
  
    --Data type for the project  
    subtype DefAddr is std_logic_vector(AddrBusSize-1 downto 0);  
    subtype DefDbus is std_logic_vector(DataBusSize-1 downto 0);  
    subtype DefData is std_logic_vector(DataSize-1 downto 0);  
    subtype DefNb_Donnees is std_logic_vector(Nb_BitDonnees-1 downto 0);  
    subtype DefBit is std_logic;  
  
    type DefConfig is record  
        IT_Enable : DefBit;  
        Mode_Vitess: DefBit; --1 : Standard 100Kbit/s 0: Rapide 4000Kbit/s  
        Sens_Echange: DefBit;  
        NB_Donnees: DefNb_Donnees;  
    end record;  
    type DefStatus is record  
        Busy : DefBit;  
        Write_Error :DefBit;  
        Read_Error: DefBit;  
        Waiting_ACK: DefBit;  
        Write_Done : DefBit;  
        Read_Done: DefBit;  
    end record;  
    -- constant for the I2C controller  
    constant TriState : DefDbus := (others => 'Z');  
    constant I2C_Addr : DefAddr := "00000000000000000000000000000000";  
    constant I2C_Config : DefAddr := "00000000000000000000000000000001";  
    constant I2C_DataToSend : DefAddr := "00000000000000000000000000000010";
```


A.2 Code VHDL de la fonction Write de l'interface processeur

```
--  
-- VHDL Architecture Projet_I2CIP_lib.Z_Write.arch  
--  
-- Created:  
--     by - CHAIMA.LAGHA (LAPTOP-I71U7FVH)  
--     at - 22:57:43 18/11/2022  
--  
-- using Mentor Graphics HDL Designer(TM) 2018.2 (Build 19)  
--  
LIBRARY ieee;  
USE ieee.std_logic_1164.all;  
USE ieee.std_logic_arith.all;  
LIBRARY Projet_I2CIP_lib;  
USE Projet_I2CIP_lib.I2C_Package.ALL;  
  
ENTITY Z_Write IS  
  PORT(  
    Addr : IN      DefAddr;  
    Clk  : IN      DefBit;  
    Dbus : IN      DefDbus;  
    RnW  : IN      DefBit;  
    nAS  : IN      DefBit;  
    nBE0 : IN      DefBit;  
    nBE1 : IN      DefBit;  
    nRst : IN      DefBit;  
    Config : OUT   DefConfig;  
    DataToSend : OUT  DefData;  
    Transfert_Request : OUT  DefBit  
  );  
  
  -- Declarations  
  
END Z_Write ;  
  
--  
ARCHITECTURE arch OF Z_Write IS  
  signal nBE: std_logic_vector (EndiannessSize-1 downto 0);  
BEGIN  
  WriteBehavior: process (nRst,Clk)  
    begin  
      nBE(0)<= nBE0;  
      nBE(1)<= nBE1;  
      if nRst='0' then  
        DataToSend<=(others=>'Z');  
        Transfert_Request<='0';  
        Config.NB_Donnees <= (others=>'0');  
        Config.IT_Enable <= '0';  
        Config.Mode_Vitess <= '0';  
        Config.Sens_Echange <= '0';  
      elsif rising_edge(Clk)then  
        if nAS='0' and RnW='0'then  
          case Addr is  
            when I2C_Addr=>  
              case nBE is  
                when BIG_ENDIAN => DataToSend <= Dbus(DataBusSize-1 downto 8);
```

```

        when others => DataToSend <= Dbus(DataBusSize-9 downto 0);
    end case;
when I2C_Config=>
    case nBE is
        when BIG_ENDIAN =>
            Config.NB_Donnees <= Dbus(DataBusSize-1 downto 8);
            Config.IT_Enable <= Dbus(0);
            Config.Mode_Vitess <= Dbus(1);
            Config.Sens_Echange <= Dbus(2);
        when LITTLE_ENDIAN =>
            Config.NB_Donnees <= Dbus(7 downto 0);
            Config.IT_Enable <= Dbus(8);
            Config.Mode_Vitess <= Dbus(9);
            Config.Sens_Echange <= Dbus(10);
        when others =>
    end case;
    Transfert_Request<= '1';
when I2C_DataToSend =>
    case nBE is
        when BIG_ENDIAN => DataToSend <= Dbus(DataBusSize-1 downto 8);
        when others => DataToSend <= Dbus(DataBusSize-9 downto 0);
    end case;
    when others =>
    end case;
end if;
end if;
end process WriteBehavior;
END ARCHITECTURE arch;

```

A.3 Code VHDL de la fonction Read de l'interface processeur

```
--  
-- VHDL Architecture Projet_I2CIP_lib.Z_Read.arch  
--  
-- Created:  
--     by - CHAIMA.LAGHA (LAPTOP-I71U7FVH)  
--     at - 22:56:16 18/11/2022  
--  
-- using Mentor Graphics HDL Designer(TM) 2018.2 (Build 19)  
--  
LIBRARY ieee;  
USE ieee.std_logic_1164.all;  
USE ieee.std_logic_arith.all;  
LIBRARY Projet_I2CIP_lib;  
USE Projet_I2CIP_lib.I2C_Package.ALL;  
  
ENTITY Z_Read IS  
  PORT(  
    Addr : IN      DefAddr;  
    DataReceived : IN   DefData;  
    RnW : IN      DefBit;  
    Status : IN     DefStatus;  
    nAS : IN      DefBit;  
    nBE0 : IN      DefBit;  
    nBE1 : IN      DefBit;  
    nRst : IN      DefBit;  
    nWait : IN      DefBit;  
    Dbus : INOUT   DefDbus  
  );  
  
  -- Declarations  
  
END Z_Read ;  
  
--  
ARCHITECTURE arch OF Z_Read IS  
  signal nBE: std_logic_vector (EndiannessSize-1 downto 0);  
BEGIN  
  ReadBehavior:process (nRst, RnW, Addr, DataReceived, Status)  
  begin  
    Dbus<=TriState;  
    nBE(0)<= nBE0;  
    nBE(1)<= nBE1;  
    if (nRst='0') then  
      Dbus<= (others'Z');  
    elsif nAS='0' and RnW='1' then  
      case Addr is  
      when I2C_Addr=>  
      when I2C_Config=>  
      when I2C_DataToSend=>  
      when I2C_DataReceived=>  
        case nBE is  
          when BIG_ENDIAN =>  
            Dbus(DataBusSize-1 downto 8)<= DataReceived;  
            Dbus(DataBusSize-9 downto 0)<= "00000000";  
          when LITTLE_ENDIAN =>
```

```

        Dbus(DataBusSize-9 downto 0)<= DataReceived;
        Dbus(DataBusSize-1 downto 8)<= "00000000";
    when others => --nothing to do
end case;
when I2C_Status =>
    case nBE is
        when BIG_ENDIAN=>
            Dbus(15)<= Status.Write_Error;
            Dbus(14)<= Status.Read_Error;
            Dbus(13)<= Status.Waiting_ACK;
            Dbus(12)<= Status.Busy;
            Dbus(11)<= Status.Write_Done;
            Dbus(10)<= Status.Read_Done;
            Dbus(DataBusSize-7 downto 0)<= "0000000000";
        when others =>
            Dbus(0)<= Status.Write_Error;
            Dbus(1)<= Status.Read_Error;
            Dbus(2)<= Status.Waiting_ACK;
            Dbus(3)<= Status.Busy;
            Dbus(4)<= Status.Write_Done;
            Dbus(5)<= Status.Read_Done;
            Dbus(DataBusSize-1 downto 6)<= "0000000000";
        end case;
    when others=>
        end case;
    end if;
end process ReadBehavior;
END ARCHITECTURE arch;

```

A.4 Code VHDL de la fonction ClockGenerator

```
-- VHDL Architecture Projet_I2CIP_lib.Z_ClockGenerator.arch
--
-- Created:
--         by - CHAIMA.LAGHA (LAPTOP-I71U7FVH)
-- at - 00:35:38 19/11/2022
--
-- using Mentor Graphics HDL Designer(TM) 2018.2 (Build 19)
LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.std_logic_arith.all;
LIBRARY Projet_I2CIP_lib;
USE Projet_I2CIP_lib.I2C_Package.ALL;

ENTITY Z_ClockGenerator IS
PORT(
    Clk : IN      DefBit;
    Config : IN      DefConfig;
    Transfert_Request : IN      DefBit;
    nRst : IN      DefBit;
    SCK_intern : OUT     DefBit
);
END Z_ClockGenerator ;
ARCHITECTURE arch OF Z_ClockGenerator IS

    signal tmp : std_logic := '0';
    signal current_Mode:std_logic:='1'; --mode standard
BEGIN
    ClockGeneratorByDivider : process (nRst,Clk, Config)
    variable count_max : integer :=3;
    variable count: integer:=1;

    begin
        if(nRst='0') then
            count_max:=125;-- Mode standard
            tmp<='0';
        elsif (transfert_Request='1')then
            if (config.Mode_Vitess/=current_Mode)then --added
                count :=1;
                current_Mode<= config.Mode_Vitess;
            end if;
            if(Config.Mode_Vitess='0')then
                count_max:=3; ---
            end if;
            if rising_edge (Clk) then
                count :=count+1;
                if (count = count_max) then
                    tmp <= NOT tmp;
                    count := 1;
                end if;
            end if;
            end if ;
            SCK_intern <= tmp;
        end process ClockGeneratorByDivider;
    END ARCHITECTURE arch;
```

A.5 Code VHDL de la fonction Emission_Reception

```
--  
-- VHDL Architecture Projet_I2CIP_lib.Z_Emission_Reception.arch  
--  
-- Created:  
-- by - CHAIMA.LAGHA (LAPTOP-I71U7FVH)  
-- at - 20:17:30 19/11/2022  
--  
-- using Mentor Graphics HDL Designer(TM) 2018.2 (Build 19)  
--  
LIBRARY ieee;  
USE ieee.std_logic_1164.all;  
USE ieee.numeric_std.all;  
USE ieee.math_real.all;  
  
LIBRARY Projet_I2CIP_lib;  
USE Projet_I2CIP_lib.I2C_Package.ALL;  
  
ENTITY Z_Emission_Reception IS  
  PORT(  
    Clk          : IN      DefBit;  
    Config       : IN      DefConfig;  
    DataToSend   : IN      DefData;  
    SCK_intern   : IN      DefBit;  
    nRst         : IN      DefBit;  
    DataReceived : OUT     DefData;  
    SCK          : OUT     DefBit;  
    Status        : OUT     DefStatus;  
    SDA           : INOUT   DefBit  
  );  
END Z_Emission_Reception ;  
  
ARCHITECTURE arch OF Z_Emission_Reception IS  
  
  signal SCK_flip : std_logic := '0';  
  signal Stuck_Data: std_logic_vector (7 downto 0);  
  signal cpt : integer ;  
-----  
-- | FONCTION NB_Donnees_to_integer (s: DefNb_Donnees)  
-- | -> Conversion le type DefNb_Donnees en integer  
-----  
  FUNCTION NB_Donnees_to_integer (s : DefNb_Donnees ) RETURN integer IS  
    VARIABLE NB_Data : integer:=0 ;  
    variable n:integer :=0;  
    BEGIN  
      loop  
        if(s(n)= '1')then  
          NB_Data:= NB_Data + 2**n;  
        end if;  
        n:=n+1;  
        exit when (n=8);  
      end loop;  
    RETURN NB_Data ;
```

```

END FUNCTION NB_Donnees_to_integer ;

--  

BEGIN
    Emission_ReceptionProc : process(nRst,SCK_intern)
    --
    variable State : DefState;
    variable CptData : integer ;
    variable i : integer range 0 to 4;
    variable j : integer range 0 to 7;

begin
    if (nRst = '0')then
        SCK <= '0';
        SDA<='0';
        DataReceived<= (others=>'Z');
        Status<= (others=>'0');
        State := idle;
    elsif rising_edge(SCK_intern)then
        case State is
            when idle =>
                i:= 0;
                Status.Busy<='0';
                Status.Write_Error<='0';
                Status.Read_Error<='0';
                Status.Waiting_ACK<='0';
                Status.Write_Done <='0';
                Status.Read_Done<='0';
                SDA <= '0';
                SCK <= '0';
                CptData:= NB_Donnees_to_integer(Config.NB_Donnees);
                State := sending_StartBit;

            when sending_StartBit =>
                --SDA passe à zéro pour t=Tbit/4
                Status.Busy <='1';
                if (i=0) then
                    SDA <= '1';
                    SCK <= '1';
                end if;
                if (i=3) then --4
                    i:=0;
                    SCK_flip <= '0';
                    j:=7;
                    State:=sending_Address;
                else
                    if (i=1) then
                        SDA <= '0';
                    end if;
                    if (i=2) then
                        SCK<='0';
                    end if;
                    i:=i+1;
                end if;

            when sending_Address =>
                if (i=2 or i=0)then
                    SCK_flip <= not SCK_flip;

```

```

        SCK <= not SCK_flip;
        SDA <= DataToSend(j);
    end if ;
    if(i=3 and j=0) then
        i:=0;
        j:=7;
        State:= sending_RnWBit;
    elsif (i=3 and j/=0)then
        i:=0;
    else
        i:=i+1;
    end if ;
    if (i=0 and j/=0) then
        j:=j-1;
    end if ;

when sending_RnWBit =>
    --Status.Write_done <= '0';
    if(i=2 or i=0) then
        SCK_flip <= not SCK_flip;
        SCK <= not SCK_flip;
    end if ;
    if (i=3) then
        i:=0;
        State:= waiting_AckBit;
        Status.Waiting_ACK <='1';
    else
        i:=i+1;
    end if ;
    SDA<=Config.Sens_Echange;

when waiting_AckBit =>
    SDA<= 'Z';
    if (i=2 or i=0)then
        SCK_flip <=not SCK_flip;
        SCK <= not SCK_flip;
    end if ;
    if (i=3 and SDA='1' and CptData=0) then
        i:=0;
        Status.Waiting_ACK <='0';
        State:= sending_StopBit;
    elsif (i=3 and SDA='1' and CptData/0) then
        i :=0;
        if (Config.Sens_Echange= '1') then
            State:= Receive_Data;
        else
            State:=Send_Data;
        end if;
        j:=7;
        Status.Waiting_ACK <='0';
    elsif (i=4 and SDA/='1')then
        Status.Write_Error <='1';
        i:=0;
        State:=idle;
    else
        i:=i+1;
    end if ;

```

```

when Send_Data =>
  if (i=0 or i=2)then
    SCK_flip <=not SCK_flip;
    SCK <= not SCK_flip;
  end if ;
  SDA <= DataToSend(j);
  if(i=3 and j=0) then
    i:=0;
    j:=7;

    CptData:=CptData-1;
    State:= waiting_AckBit;
    Status.Waiting_ACK <='1';
  elsif (i=3 and j/=0)then
    --j:=j-1;
    i:=0;
  else
    i:=i+1;
  end if;
  if (i=0 and j/=0) then
    j:=j-1;
  end if ;

when Receive_Data =>
  SDA<= 'Z';
  if (i=2 or i=0)then
    SCK_flip <=not SCK_flip;
    SCK <= not SCK_flip;
  end if ;
  if(i=3 and j=0) then
    i:=0;
    j:=7;
    CptData:=CptData-1;
    DataReceived<= Stuck_Data;
    State:= sending_AckBit;
  elsif (i=3 and j/=0)then
    j:= j-1 ;
    i:=0;
  else
    i:=i+1;
  end if;
  Stuck_Data(j)<=SDA ;
when sending_AckBit =>
  if (i=2 or i=0)then
    SCK_flip <= not SCK_flip;
    SCK <= not SCK_flip;
  end if ;
  if (i=3 and CptData=0) then
    i:=0;
    State:= sending_StopBit;
    SDA <= '0';
  elsif (i=3 and CptData/=/0)then
    State:= Receive_Data;
    i:=0;
  else
    i:=i+1;
    SDA <='1';
  end if ;

```

```

when sending_StopBit =>
    SDA <= '0';
    if (i=3) then
        i:=0;
        SDA<= '1';
        Status.Busy<='0';
        State:=idle;
    elsif (i=2)then
        SCK <='1';
    end if;
    i:=i+1;
    if (Config.Sens_Echange= '0') then
        Status.Write_Done <='1';
    else
        Status.Read_Done <='1';
    end if;
end case ;
end if ;
end process Emission_ReceptionProc;

END ARCHITECTURE arch;

```

A.6 Code VHDL de l'environnement de test

```
--  
-- VHDL Architecture Projet_I2CIP_lib.Z_Environnement2.arch  
--  
-- Created:  
--     by - CHAIMA.LAGHA (LAPTOP-I71U7FVH)  
--     at - 16:00:05 26/11/2022  
--  
-- using Mentor Graphics HDL Designer(TM) 2018.2 (Build 19)  
--  
LIBRARY ieee;  
USE ieee.std_logic_1164.all;  
USE ieee.std_logic_arith.all;  
USE ieee.std_logic_unsigned.all;  
LIBRARY Projet_I2CIP_lib;  
USE Projet_I2CIP_lib.I2C_Package.ALL;  
  
ENTITY Z_Environnement2 IS  
    PORT(  
        Dbus : INOUT DefDbus;  
        Addr : OUT DefAddr;  
        Clk : OUT DefBit;  
        RnW : OUT DefBit;  
        nAS : OUT DefBit;  
        nBE0 : OUT DefBit;  
        nBE1 : OUT DefBit;  
        nRst : OUT DefBit;  
        nWait : OUT DefBit;  
        SDA : INOUT DefBit  
    );  
END Z_Environnement2 ;  
ARCHITECTURE arch OF Z_Environnement2 IS  
  
    SIGNAL ClkInt, RstInt : std_logic;  
    SIGNAL CptDATA : std_logic_vector(7 downto 0);  
BEGIN  
    Clk <= ClkInt;  
    nRst <= RstInt;  
  
    -- Definition des processus  
  
    ResetGenerator : process  
        begin  
            RstInt <= '0';  
            wait for 5000 ns;  
            RstInt <= '1';  
            wait;  
    end process ResetGenerator;  
  
    ClockGenerator : process  
        begin  
            ClkInt <= '0';  
            wait for 5 ns;  
            ClkInt <= '1';  
            wait for 5 ns;
```

```

end process ClockGenerator;

CPUonlyBehavior : process (ClkInt,RstInt)
variable DbusValue : DefDbus;
variable count : integer :=0;
variable ReadNWrite : integer range 0 to 1;
variable NBData : integer range 0 to 7 :=7;
variable c_Standard :integer :=Nb_CycleStandadard;
type DefState is (idle, Writing_config, Writing_SendAddress,
I2C_DeviceACK_Add,I2C_DeviceACK_1,I2C_DeviceACK_2,I2C_DeviceACK_3,
Write_Data_1,Write_Data_2,Write_Data_3, Read_Status, Reading_Config,
Writing_DeviceAdd, Receive_Data_1,Receive_Data_2 ,Device_Ack_1,Device_Ack_2);
variable State_Proc : DefState;
begin
-- -- Reset I2C controller
if (RstInt = '0') then
    Addr <= (others=>'0');
    Dbus <= (others=>'Z');
    nWait <='0';
    RnW <= '0';
    nAS <= '1';
    SDA<= '0';
    nBE0 <='0';
    nBE1 <='1';
    State_Proc:= Idle;
elsif rising_edge (ClkInt) then
    Case State_Proc is
        when idle =>
            -- -- Preparing the I2C controller for data transfer
            nWait <='1';
            nAS <= '0';
            ReadNWrite:=0;
            SDA<='Z';
            Dbus <= (others=>'Z');
            if (count=Nb_CycleStandadard) then
                State_Proc:= Writing_config;
                count:=0;
            else
                State_Proc:= idle;
                count:=count+1;
            end if;
        when Writing_config =>
-- -- Configuration of a 3 bytes send data in standard mode and Little
Endian format without interruption

            DbusValue:="0000001000000011"; --0203
            RnW <= '0';
            Addr <= I2C_Config ;--01
            Dbus <= DbusValue(DataBusSize-1 downto 0);
            CptDATA<=DbusValue(DataBusSize-9 downto 0);
            SDA<= 'Z';
            if (count=Nb_CycleStandadard) then
                State_Proc :=Writing_SendAddress;
            else
                State_Proc:= Writing_config;
                count:=count+1;
            end if;
    end case;
end if;
end process;

```

```

When Writing_SendAddress =>
    DbusValue:="0000000011011001"; --D9
    RnW <= '0';
    Addr <= I2C_Addr ; --00
    Dbus <=DbusValue(DataBusSize-1 downto 0);
    SDA<= 'Z';
    if(count=((10*Nb_CycleStandadard)+383) )then
        count:=0;
        ReadNWrite:=0;
        State_Proc:= I2C_DeviceACK_Add;
    else
        State_Proc:= Writing_SendAddress;
        count:= count+1;
    end if;
When I2C_DeviceACK_Add =>
    SDA <='1';
    if (count=c_Standard) then
        if (ReadNWrite=0) then
            State_Proc:= Write_Data_1;
        else
            State_Proc:= Receive_Data_1;
    end if;
    count:=0;
    else
        count:=count+1;
    end if ;
When Write_Data_1=>
    -- -- Sending data once we have received the acknowledgement bit

    DbusValue:="0000000001011101"; --5D
    Addr <= I2C_DataToSend; --02
    Dbus <=DbusValue(DataBusSize-1 downto 0);
    SDA <='Z';
    if(count=(8*Nb_CycleStandadard ))then
        count:=0;
        cptData<= cptData-1;
        State_Proc:= I2C_DeviceACK_1;
    else
        State_Proc:= Write_Data_1;
        count:= count+1;
    end if;
When I2C_DeviceACK_1 =>
    SDA <='1';
    if (count=Nb_CycleStandadard) then
        State_Proc:= Write_Data_2;
        count:=0;
    else
        count:=count+1;
    end if ;
When Write_Data_2=>
    -- -- Sending data once we have received the acknowledgement bit
    DbusValue:="0000000000011110"; --1E
    Addr <= I2C_DataToSend; --02
    Dbus <=DbusValue(DataBusSize-1 downto 0);
    SDA <='Z';
    if(count=(8*Nb_CycleStandadard ))then
        count:=0;
        cptData<= cptData-1;

```

```

State_Proc:= I2C_DeviceACK_2;
else
  State_Proc:= Write_Data_2;
  count:= count+1;
end if;
When I2C_DeviceACK_2 =>
  SDA <='1';
  if (count=Nb_CycleStandadard) then
    State_Proc:= Write_Data_3;
    count:=0;
  elsif( count=Nb_CycleStandadard ) then --and cptData=0
    State_Proc:= idle;
    count:=0;
  else
    count:=count+1;
  end if ;
when Write_Data_3=>
  -- -- Sending data once we have received the acknowledgement bit
  DbusValue:="00000000000111101"; --3D
  Addr <= I2C_DataToSend; --02
  Dbus <=DbusValue(DataBusSize-1 downto 0);
  SDA <='Z';
  if(count=(8*Nb_CycleStandadard ))then
    count:=0;
    cptData<= cptData-1;
    State_Proc:= I2C_DeviceACK_3;
  else
    State_Proc:= Write_Data_3;
    count:= count+1;
  end if;
when I2C_DeviceACK_3 =>
  SDA <='1';
  if (count=Nb_CycleStandadard)then
    State_Proc:= Read_Status;
    count:=0;
  else
    count:=count+1;
  end if ;

when Read_Status =>
  RnW<= '1';
  Addr <= I2C_Status; --02
  Dbus <= (others=>'Z');
  SDA<='Z';
  if (count=Nb_CycleStandadard) then
    if(ReadNWrite=0)then
      State_Proc:= Reading_Config;
    else
      State_Proc:=idle;
    end if;
    count:=0;
  else
    State_Proc:= Read_Status;
    count:=count+1;
  end if;

```

```

When Reading_Config =>
  -- -- Configuration of a 2 bytes read data in standard mode and Big
  Edian format without interruption

    nBE0<= '1';
    nBE1 <= '0';
    DbusValue:="0000001000000100"; --0204
    Addr <= I2C_Config ;--01
    Dbus <= DbusValue(DataBusSize-1 downto 0);
    RnW <= '0';
    SDA<= 'Z';
    if (count=Nb_CycleRapide) then
      State_Proc :=Writing_DeviceAdd;
    else
      State_Proc:= Reading_Config;
      count:=count+1;
    end if;
When Writing_DeviceAdd=>
  DbusValue:="1001101100000110"; --9B06
  RnW <= '0';
  Addr <= I2C_Addr ;
  Dbus <= DbusValue(DataBusSize-1 downto 0);
  SDA<= 'Z';
  if(count=10*Nb_CycleRapide)then --24
    count:=0;
    ReadNWrite :=1;
    c_Standard:=33;
    State_Proc:= I2C_DeviceACK_Add;
  else
    State_Proc:= Writing_DeviceAdd;
    count:= count+1;
  end if;
When Receive_Data_1=>
  DBusValue:="0000000011000011";
  SDA<=DBusValue(NBData);
  if(NBData=0) then
    if (count=Nb_CycleRapide) then
      State_Proc :=Device_Ack_1;
      NBData:=7;
      count:=0;
    else
      count:=count+1;
    end if;
  else
    if (count=Nb_CycleRapide)then
      NBData:=NBData-1;
      count:=0;
    else
      count:=count+1;
    end if ;
  end if;

When Device_Ack_1=>
  RnW<= '1';
  Addr<= I2C_DataReceived;
  Dbus<=(others=>'Z');
  if (count=Nb_CycleRapide) then
    State_Proc:= Receive_Data_2;

```

```

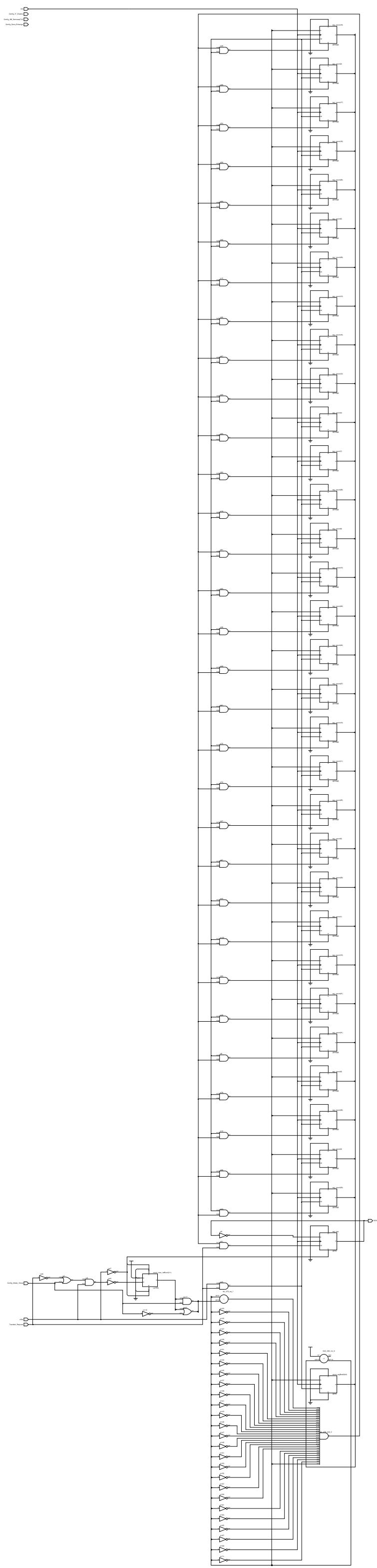
        count:=0;
      else
        State_Proc:= Device_Ack_1;
        count:=count+1;
      end if;
    When Receive_Data_2=>
      DBusValue:="0000000000000001";
      SDA<=DBusValue(NBData);

      if(NBData=0) then
        if (count=Nb_CycleRapide) then
          State_Proc :=Device_Ack_2;
          NBData:=7;
          count:=0;
        else
          count:=count+1;
        end if;
      else
        if (count=Nb_CycleRapide)then
          NBData:=NBData-1;
          count:=0;
        else
          count:=count+1;
        end if ;
      end if;
    When Device_Ack_2=>
      SDA<= 'Z';
      if (count=Nb_CycleRapide) then
        State_Proc:= Read_Status;
        count:=0;
      else
        State_Proc:= Device_Ack_2;
        count:=count+1;
      end if;
      end case;
    end if;
  end process CPUonlyBehavior;

END ARCHITECTURE arch;

```

A.7 Schéma RTL de la fonction ClockGenerator



A.8 Schéma RTL de la fonction Emission_Reception

