Rapport projet architecture des calculateurs - Hérons

Radicalisez-vous ?

Gautier Gwenlan ; Gaël Thomas ; Yannis Zongo

INSA Rennes

Janvier 2026

Supervisor : Jean-Gabriel Cousin

Department : Electronique et informatique industrielle

Table des matières

I INTRODUCTION **1**

II La nouvelle unité d'exécution (*Pj_Unit*) **2**

II.A Algorithme itératif de type Héron 2

II.B Machine d'états : suivi et sorties 2

II.C Gestion des sorties 3

II.D Opérateur arithmétique 3

II.E Synthèse et performances 3

II.F Vérification et validation 4

III Iterative Divide **4**

III.A Architecture de l'opérateur 4

III.B Évolution de la machine d'états 5

III.C Validation 5

IV Unité de Division de Fréquence (FDU) **5**

IV.A Spécifications et objectifs 5

IV.B Choix architecturaux 6

IV.C Implémentation RTL 6

IV.D Vérification par simulation 6

IV.E Synthèse et performances 7

V Conclusion et réflexions diverses **8**

V.A Justification de la caractéristique "768 bits" de mémoire issue de la synthèse 8

V.B Annexe E – Message d'alerte 332125 dans la synthèse de la CPU 8

V.C Annexe E – Impact de l'encodage des états sur les caractéristiques de la CPU . . . 8

Appendix **i**

projet architecture des calculateurs - Hérons

Radicalisez-vous ?

Gwenlan Gautier, Gael Thomas, Yannis Zongo

25 juin 2026

Résumé

Ce projet porte sur l'amélioration d'une architecture de CPU existante par l'intégration d'une unité d'exécution dédiée, *Pj_Unit*, mettant en œuvre un algorithme itératif inspiré de la méthode de Héron. La conception repose sur une unité de contrôle locale à base de machine à états finis, deux implémentations alternatives (combinatoire et itérative), ainsi qu'un diviseur de fréquence programmable. Le système est décrit au niveau RTL, simulé et synthétisé sous *Quartus Prime*. Les performances sont analysées en termes de ressources logiques et de fréquence maximale, afin de comparer les compromis architecturaux entre les solutions proposées.

Keywords : *Architecture CPU, Unité d'exécution, Algorithme de Héron, Machine à états finis, Conception RTL, Division itérative, Synthèse logique*

I. INTRODUCTION

Ce projet s'inscrit dans un contexte visant à l'amélioration d'un système de type CPU (Central Processing Unit). Face à l'évolution des besoins en performance et en flexibilité, l'architecture existante nécessite une mise à niveau fonctionnelle. Notre équipe a été chargée de reprendre ce projet en cours pour implémenter des fonctionnalités critiques manquantes ou perfectibles, dans une logique de compétition et d'efficacité. L'objectif principal de cette étude est de faire évoluer le processeur sur deux axes majeurs. Premièrement, il s'agit d'enrichir l'unité de traitement en y intégrant une nouvelle unité d'exécution capable de réaliser des calculs plus spécifiques et complexes. Deuxièmement, le système de gestion de l'horloge, jusqu'alors figé, doit être rendu dynamique et programmable pour adapter la vitesse d'exécution aux besoins réels du système . En parallèle de ces développements, un travail d'analyse est requis pour caractériser l'impact de ces modifications en termes de consommation de ressources matérielles et de rapidité d'exécution. Pour mener à bien cette mission, nous avons adopté une démarche d'ingénierie structurée, privilégiant l'analyse préalable des spécifications, la justification des choix de conception et la vérification systématique par simulation. Cette rigueur vise à garantir la livraison d'un système opérationnel, fiable et optimisé. Ce rapport retrace le cheminement technique et les choix effectués par l'équipe pour répondre à ce cahier des charges. Dans un premier temps, nous présenterons la conception et l'intégration de la nouvelle unité d'exécution au sein du processeur. Nous détaillerons ensuite la refonte du diviseur de fréquence pour le rendre programmable et synchrone. Enfin, le développement s'achèvera par une analyse de synthèse caractérisant les performances globales de la CPU améliorée au regard des ressources technologiques engagées.

II. La nouvelle unité d'exécution (*Pj_Unit*)

Principe général La *Pj_Unit* est une unité d'exécution dédiée intégrée à la CPU, chargée de réaliser un calcul itératif sur une donnée d'entrée. La CPU fournit la valeur à traiter ainsi qu'une commande de lancement, puis attend la fin du calcul signalée par le signal *done*. Le calcul est séquencé par une machine à états finis locale, garantissant un fonctionnement synchrone et déterministe.

II.A. Algorithme itératif de type Héron

La *Pj_Unit* implémente un algorithme itératif inspiré de la méthode de Héron, paramétré par un entier N correspondant au nombre d'itérations. Une valeur intermédiaire it_k est initialisée à partir de la donnée d'entrée Din, puis mise à jour à chaque cycle d'horloge selon la relation :

$$it_{k+1} = \frac{1}{2}\left(it_k + \frac{Din}{it_k}\right),$$

jusqu'à atteindre la condition d'arrêt $k = N$. Le paramètre N permet ainsi d'ajuster le compromis entre la précision du résultat obtenu et le temps de calcul nécessaire.

II.B. Machine d'états : suivi et sorties

La machine d'états constitue le cœur de la *Pj_Unit*. Elle est composée d'une fonction de suivi, chargée de mémoriser et faire évoluer l'état courant, et d'une fonction de sortie, chargée de générer les signaux de commande.

L'état global est codé sur cinq bits : quatre bits stockés dans un registre à décalage *74194*, complétés par un bit supplémentaire mémorisé dans une bascule dédiée. À chaque front d'horloge, l'état évolue de manière synchrone selon les signaux de contrôle, assurant l'enchaînement correct des différentes phases du calcul.

Les signaux de sortie principaux sont :

— *Litk*, commandant la mise à jour du registre intermédiaire ;

— *done*, indiquant la fin du calcul et la disponibilité du résultat ;

— *start*, pilotant le lancement ou la réinitialisation de l'opérateur itératif.

II.C. Gestion des sorties

Le résultat final *PjU[15..0]* est généré à l'aide de multiplexeurs 16 bits permettant de sélectionner la valeur pertinente issue du chemin de données. Le *Zero flag* est généré par une logique combinatoire analysant l'ensemble des bits du résultat afin de détecter une valeur nulle. Un indicateur supplémentaire (*blink*) est mémorisé dans une bascule D afin de fournir une information de fin de calcul stable et synchronisée à la CPU.

II.D. Opérateur arithmétique

L'opération principale de la partie traitement repose sur un additionneur–soustracteur 16 bits modulaire, construit par chaînage de quatre blocs 4 bits. Cette structure hiérarchique assure une propagation correcte du report et fournit, en plus du résultat, des indicateurs utiles à l'exploitation du calcul.

II.E. Synthèse et performances

La synthèse met en évidence l'impact des choix architecturaux sur les ressources et les performances.

Version initiale

— Éléments logiques : 2

— Registres : 1

Version améliorée

— Éléments logiques : 464

— Registres : 76

La fréquence maximale de fonctionnement obtenue après amélioration est :

$$f_{\max} = 19.92 \text{ MHz}$$

Cette augmentation des ressources est cohérente avec l'introduction d'un calcul itératif séquencé, d'une machine à états complète et de registres intermédiaires. Le caractère itératif permet de limiter

la complexité du chemin critique, au prix d'un calcul réparti sur plusieurs cycles.

II.F. *Vérification et validation*

La validation fonctionnelle et temporelle de la *Pj_Unit* a été réalisée par simulation RTL sous ModelSim. Les chronogrammes montrent un déroulement correct du calcul, une évolution cohérente des états internes et une activation du signal *done* uniquement lorsque le résultat est stabilisé. Toutes les transitions sont synchrones avec l'horloge et aucun glitch n'est observé, validant ainsi les choix architecturaux retenus.

Remarque Les figure A10 a A14 reprennent les point importants abordé dans cette section

III. Iterative Divide

Principe général L'opérateur *iterative_divide* réalise une division de manière séquentielle, en répétant une opération élémentaire sur plusieurs cycles d'horloge. Une unité de contrôle pilote le déroulement du calcul (initialisation, itérations et terminaison), tandis qu'une unité de traitement exécute les opérations arithmétiques. Le résultat final est validé après un nombre déterminé d'itérations.

III.A. *Architecture de l'opérateur*

L'opérateur est structuré en deux parties :

- une **unité de traitement (PU)**, chargée d'exécuter l'algorithme de division itérative fourni ;
- une **unité de contrôle (CU)**, responsable du séquencement des opérations et de la gestion des itérations.

La CU repose sur un compteur implémenté à l'aide de bascules imposées (JK, D, T, RS et JK), utilisées pour piloter la progression du calcul. La PU s'appuie sur un additionneur–soustracteur 16 bits et travaille en arithmétique signée, le bit de poids fort étant utilisé pour détecter le signe du résultat intermédiaire et orienter la mise à jour du quotient.

III.B. Évolution de la machine d'états

Une première implémentation reposait sur une machine à quatre états (*idle*, *init*, *iter*, *done*). Bien que fonctionnelle au niveau du calcul arithmétique, cette version présentait un défaut de séquencement empêchant la terminaison correcte du compteur.

Afin de fiabiliser le fonctionnement, une architecture simplifiée à deux états a été retenue :

— **IDLE** : état d'attente, l'opérateur est inactif ;

— **RUN** : état d'exécution, durant lequel les itérations sont effectuées.

Le passage de *IDLE* à *RUN* est déclenché par le signal *start*. Le compteur est alors activé et effectue les itérations nécessaires jusqu'à leur achèvement, avant un retour à l'état *IDLE*.

L'erreur initiale a été identifiée comme provenant d'une mauvaise initialisation des variables internes. Conformément à la spécification, une itération d'avance est requise : le quotient est initialisé à zéro et un premier décalage est appliqué avant le démarrage du comptage.

III.C. Validation

Dans la version finale, l'état *RUN* active un compteur 4 bits comptant de 0 à 15, correspondant au nombre d'itérations de l'algorithme. Les simulations en Gate Level Simulation confirment le bon déroulement des itérations ainsi que la validité des résultats obtenus.

Cette version simplifiée assure un fonctionnement robuste, synchrone et conforme aux spécifications de l'opérateur de division itérative.

IV. Unité de Division de Fréquence (FDU)

IV.A. Spécifications et objectifs

La FDU est une unité programmable chargée de générer une horloge interne divisée par 2, 4, 8 ou 16 à partir de l'horloge maître. Le facteur de division est sélectionné dynamiquement via une commande *Sclk* codée sur deux bits.

Le module doit respecter deux contraintes majeures : un fonctionnement strictement synchrone avec l'horloge maître et une commutation propre, sans génération de glitch lors des changements de fréquence. Une fonctionnalité de verrouillage (*Chip Select*) permet en outre de forcer la sortie à l'état bas lorsque la FDU n'est pas active, tout en conservant une remise à zéro asynchrone.

IV.B. Choix architecturaux

L'architecture retenue repose sur un domaine d'horloge unique : le signal *Clock* est appliqué simultanément à toutes les bascules du système. Ce choix garantit un comportement déterministe et synchrone, indépendamment de la fréquence sélectionnée.

La division de fréquence est réalisée à l'aide de quatre étages utilisant chacun un type de bascule différent (D, JK, SR et T), permettant d'obtenir simultanément les fréquences $F/2$, $F/4$, $F/8$ et $F/16$. La sélection de la fréquence s'effectue via un multiplexeur combinatoire structuré en arbre binaire.

Afin d'éviter tout parasite de commutation, la sortie du multiplexeur est enregistrée par une bascule D finale, cadencée par l'horloge maître. Cette bascule agit comme un filtre temporel et garantit un signal *CPUclk* propre et parfaitement aligné. Le signal *Chip Select* est pris en compte en amont de cette bascule afin de forcer la sortie à zéro de manière synchrone.

IV.C. Implémentation RTL

Le cœur de la FDU est constitué de quatre étages de division successifs. Chaque étage bascule uniquement lorsque les conditions imposées par les étages précédents sont vérifiées, en respectant la contrainte de portes logiques à deux entrées maximum. La remise à zéro asynchrone est assurée par le signal de reset global connecté aux entrées *CLR* de l'ensemble des bascules.

(voir figures A15 à A17 pour l'implémentation sous Quartus)

IV.D. Vérification par simulation

La validation fonctionnelle et temporelle a été réalisée par simulation comportementale. Les chronogrammes montrent que toute variation de la commande *Sclk* est prise en compte avec un retard constant d'un cycle d'horloge, sans apparition de glitch ni impulsion tronquée. Les fronts montants du signal *CPUclk* restent alignés avec ceux de l'horloge maître, confirmant le caractère strictement synchrone de l'architecture. Le fonctionnement du *Chip Select* est également validé : l'arrêt et la reprise de l'horloge s'effectuent proprement, sur des fronts complets.

IV.E. Synthèse et performances

Une comparaison entre la FDU d'origine (division fixe par 2) et la FDU améliorée met en évidence l'impact des choix architecturaux :

FDU d'origine

- Éléments logiques : 1
- Registres : 1

FDU améliorée

- Éléments logiques : 7
- Registres : 5

Les fréquences maximales obtenues sont :

- f_{max} FDU d'origine : 1265.82 MHz
- f_{max} FDU améliorée : 551.57 MHz

Malgré l'ajout de la logique de sélection, la fréquence maximale reste largement compatible avec les besoins de la CPU. L'utilisation d'un registre en sortie permet de maîtriser le chemin critique et d'assurer des contraintes temporelles robustes. (voir figures A18 à A20 pour les chronogrammes)

V. Conclusion et réflexions diverses

Ce projet a permis de concevoir et de valider une chaîne de détection d'obstacles à courte distance basée sur la propagation ultrasonore. Les différentes étapes ont mis en évidence l'importance des choix de conception électronique ainsi que les écarts entre modélisation théorique et comportement réel. Malgré les difficultés expérimentales, le système fonctionne correctement et constitue une base solide pour des améliorations futures, notamment en précision de mesure et en traitement numérique.

V.A. Justification de la caractéristique "768 bits" de mémoire issue de la synthèse

L'analyse des dumps mémoire (voir Annexes) montre que les dernières adresses utilisées débutent à `0x38` et `0x18`, soit respectivement 56 et 24 en décimal. Selon l'interprétation retenue, l'espace mémoire effectivement inféré correspond à un total de **96 octets**. La synthèse exprimant la taille mémoire en bits, on obtient alors :

$$96 \times 8 = 768 \text{ bits.}$$

Cette valeur est donc cohérente avec l'espace mémoire réellement implémenté et justifie précisément la caractéristique indiquée par l'outil de synthèse.

V.B. Annexe E – Message d'alerte 332125 dans la synthèse de la CPU

Le message d'alerte `332125` signale la présence d'une **boucle combinatoire**, c'est-à-dire un chemin logique où une sortie est réinjectée vers une entrée sans élément de mémorisation. Ce type de structure peut entraîner des comportements instables et empêche l'outil d'analyse temporelle de garantir un délai de propagation fini, indiquant ainsi une portion du design à corriger.

V.C. Annexe E – Impact de l'encodage des états sur les caractéristiques de la CPU

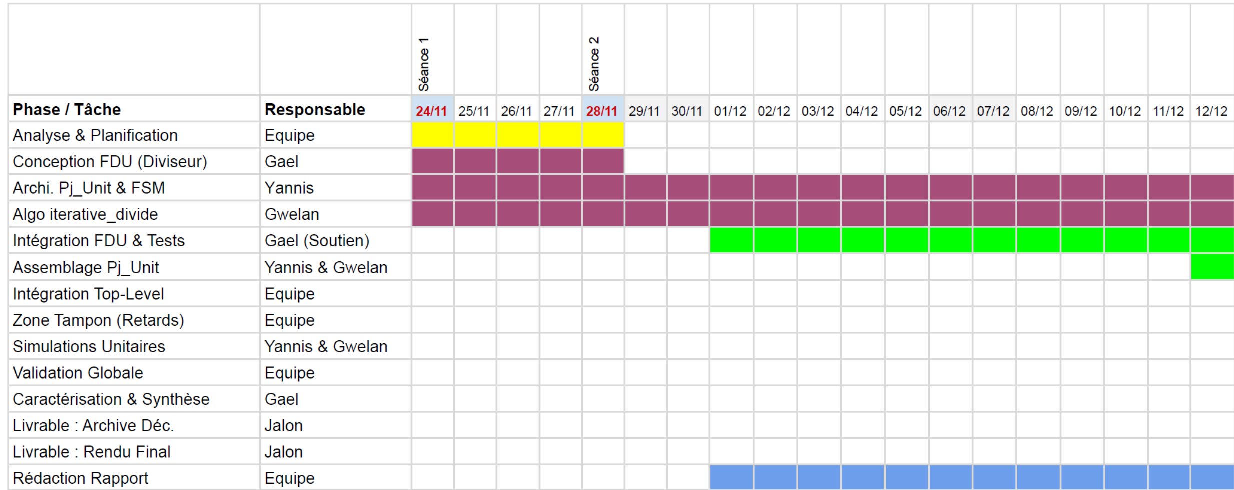
La CPU a été synthétisée en comparant trois modes d'encodage des états : **Auto**, **Johnson** et **Minimal Bits**. Les résultats obtenus sont présentés ci-dessous.

Encodage	**Logic Elements**	**Registers**	**Fmax**
Auto	1786	468	20.47 MHz
Johnson	1900	447	20.99 MHz
Minimal Bits	1902	421	19.50 MHz

L'encodage **Minimal Bits** minimise le nombre de registres mais augmente la complexité de la logique combinatoire, ce qui réduit la fréquence maximale. À l'inverse, le mode **Auto** consomme davantage de bascules tout en conservant de bonnes performances globales. L'encodage **Johnson** apparaît comme le meilleur compromis dans notre cas, offrant la fréquence maximale la plus élevée.

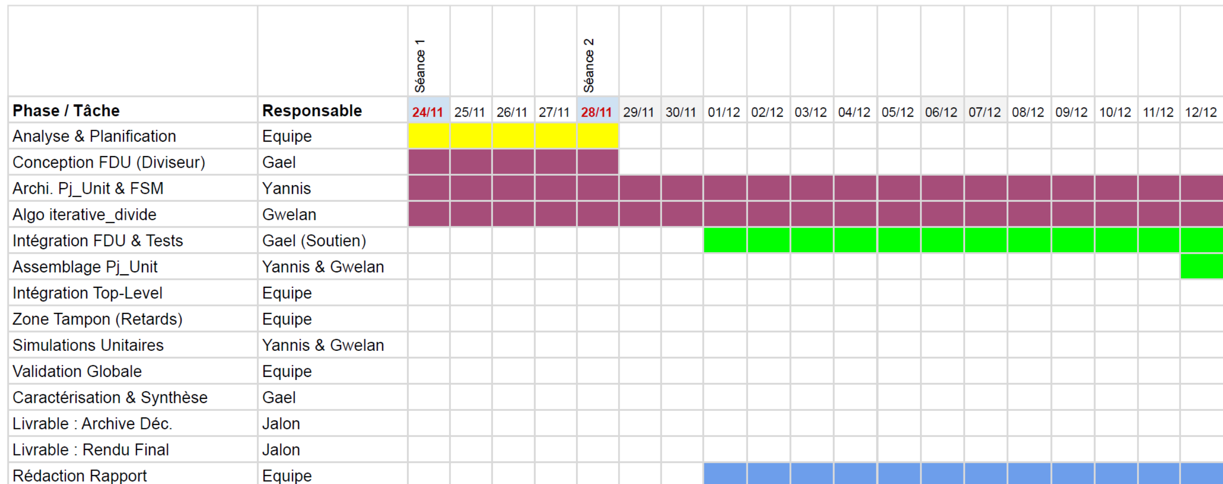
Appendix

A. Additional tables and figures



		Séance 1				Séance 2														
Phase / Tâche	**Responsable**	24/11	25/11	26/11	27/11	28/11	29/11	30/11	01/12	02/12	03/12	04/12	05/12	06/12	07/12	08/12	09/12	10/12	11/12	12/12
Analyse & Planification	Equipe																			
Conception FDU (Diviseur)	Gael																			
Archi. Pj_Unit & FSM	Yannis																			
Algo iterative_divide	Gwelan																			
Intégration FDU & Tests	Gael (Soutien)																			
Assemblage Pj_Unit	Yannis & Gwelan																			
Intégration Top-Level	Equipe																			
Zone Tampon (Retards)	Equipe																			
Simulations Unitaires	Yannis & Gwelan																			
Validation Globale	Equipe																			
Caractérisation & Synthèse	Gael																			
Livrable : Archive Déc.	Jalon																			
Livrable : Rendu Final	Jalon																			
Rédaction Rapport	Equipe																			

FIGURE A1 – Diagramme de Gantt prévisionnel du projet (partie 1)



		Séance 1				Séance 2														
Phase / Tâche	**Responsable**	24/11	25/11	26/11	27/11	28/11	29/11	30/11	01/12	02/12	03/12	04/12	05/12	06/12	07/12	08/12	09/12	10/12	11/12	12/12
Analyse & Planification	Equipe																			
Conception FDU (Diviseur)	Gael																			
Archi. Pj_Unit & FSM	Yannis																			
Algo iterative_divide	Gwelan																			
Intégration FDU & Tests	Gael (Soutien)																			
Assemblage Pj_Unit	Yannis & Gwelan																			
Intégration Top-Level	Equipe																			
Zone Tampon (Retards)	Equipe																			
Simulations Unitaires	Yannis & Gwelan																			
Validation Globale	Equipe																			
Caractérisation & Synthèse	Gael																			
Livrable : Archive Déc.	Jalon																			
Livrable : Rendu Final	Jalon																			
Rédaction Rapport	Equipe																			

FIGURE A2 – Diagramme de Gantt prévisionnel du projet (partie 2)

Phase / Tâche	Responsable	01/01	02/01	03/01	04/01	05/01	Séance 4 (Fin) 06/01	07/01	08/01	09/01	10/01	11/01	12/01	13/01	14/01	15/01	16/01
Analyse & Planification	Equipe																
Conception FDU (Diviseur)	Gael																
Archi. Pj_Unit & FSM	Yannis																
Algo iterative_divide	Gwelan																
Intégration FDU & Tests	Gael (Soutien)																
Assemblage Pj_Unit	Yannis & Gwelan																
Intégration Top-Level	Equipe																
Zone Tampon (Retards)	Equipe																
Simulations Unitaires	Yannis & Gwelan																
Validation Globale	Equipe																
Caractérisation & Synthèse	Gael																
Livrable : Archive Déc.	Jalon																
Livrable : Rendu Final	Jalon						♦										
Rédaction Rapport	Equipe																

FIGURE A3 – Diagramme de Gantt prévisionnel du projet (partie 3)

Phase / Tâche	Responsable	Séance 1 24/11	25/11	26/11	27/11	Séance 2 28/11	29/11	30/11	01/12	02/12	03/12	04/12	05/12	06/12	07/12	08/12	09/12	10/12	11/12	12/12	13/12	14/12	15/12	16/12	17/12	18/12	Séance 3 19/12	20/12	21/12	22/12
Analyse & Planification	Equipe																													
Conception FDU (Retard)	Gael																													
Archi. Pj_Unit & FSM	Yannis																													
Algo iterative_divide	Gwelan																													
Aide Pj_Unit & Tests	Gael																													
Intégration Top-Level	Equipe																													
Simulations & Debug	Equipe																													
Caractérisation & Synthèse	Equipe																													
Livrable : Archive Déc.	Jalon																										♦			
Livrable : Rendu Final	Jalon																													
Rédaction Rapport	Equipe																													

FIGURE A4 – Diagramme de Gantt réel du projet (partie 1)

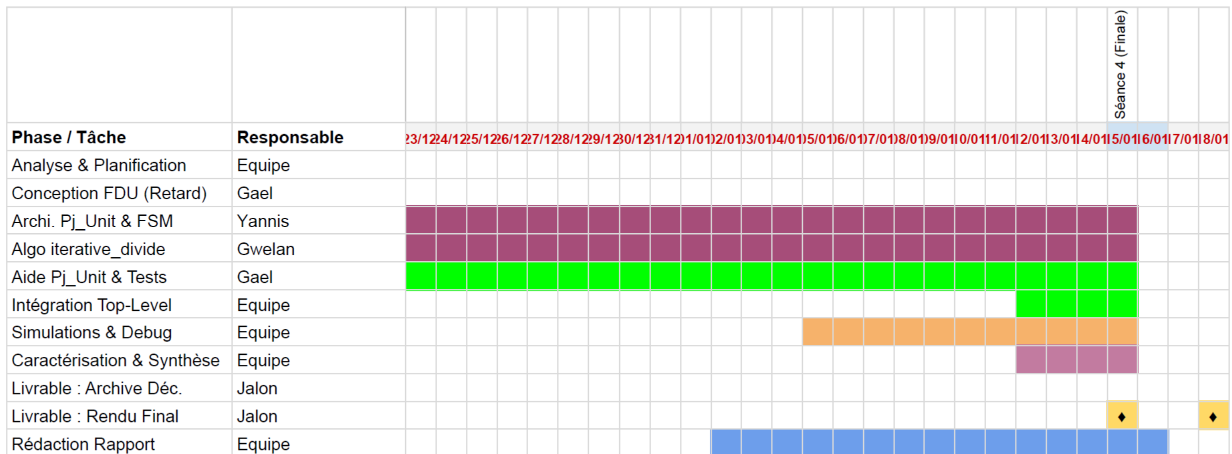


FIGURE A5 – Diagramme de Gantt réel du projet (partie 2)

Comparison between the planned and actual schedules La comparaison entre les diagrammes de Gantt prévisionnel et réel met en évidence des écarts dus aux contraintes techniques rencontrées au cours du projet. Le planning initial, fondé sur des estimations optimistes, a dû être ajusté en raison d'itérations supplémentaires lors de la conception et de l'intégration. Le diagramme réel reflète ainsi un allongement des phases de tests et une réorganisation des priorités, illustrant l'écart entre planification théorique et déroulement réel d'un projet d'ingénierie.

example | global characteristics

instruction	IDIV R_d,R_s					.v1
comments	Calcul le quotient de la division de R_d par R_s et écrit le résultat dans R_d					
type / classe	CISC					
Byte encoding	number	2	opcode	Fpju d : s		
Byte contents	Byte 1	0x 70	Byte 2	0x 61	Byte 3	0x
generic description	If(Rs !=0) Rd =Rd/Rs					
generic mnemonic(s)	IDIV Rd, Rs			IDIV 0xds		
specific state(s)	number	3	s151, s152, s153			

s 151 Active Lpju et Epju pour envoyer les opérandes dans le bloc division
s 152 Maintient Epju actif et boucle tant que done n'est pas reçu
s 153 Enregistre le quotient dans le registre Rd

states	state subdiagram	CPI	20
s 1	AbArb(T_{pc}) . E_{as} . DbArb(T_{mem}) . E-U_{pc} . E_{ir}		
s 2	AbArb(T_{pc}) . E_{as} . DbArb(T_{mem}) . E-U_{pc} . X_{flag} . E_{ds} . E_{pl}		
s 151	Xrf, DbArb(EdbL && TrfL), 00.Lpju.Epju		
s 152	Epju, DbArb(EdbL && TpjuL)		
s 153	Wrfd, DbArb(EdbL && TpjuL), Wsr		

version with bus Arbiters

instruction	IDIV R_d,R_s					.v2
comments	Calcul le quotient de la division de R_d par R_s et écrit le résultat dans R_d					
type / classe	CISC					
Byte encoding	number	2	opcode	Fpju d : s		
Byte contents	Byte 1	0x 70	Byte 2	0x 61	Byte 3	0x
generic description	If(Rs !=0) Rd =Rd/Rs					
generic mnemonic(s)	IDIV Rd, Rs			IDIV 0xds		
specific µI(s)	number	3	s151, s152, s153			

µI 151 Active Lpju et Epju pour envoyer les opérandes dans le bloc division
µI 152 Maintient Epju actif et boucle tant que done n'est pas reçu
µI 153 Enregistre le quotient dans le registre Rd

µIs	µProgram	CPI	20
µI 1	T_{pc} . E_{as} . T_{mem} . E-U_{pc} . E_{ir}		
µI 2	T_{pc} . E_{as} . T_{mem} . E-U_{pc} . X_{flag} . E_{ds} . E_{pl}		
µI 151	Xrf, EdbL && TrfL, 00.Lpju.Epju		
µI 152	Epju, EdbL && TpjuL		
µI 153	Wrfd, EdbL && TpjuL, Wsr		

version without bus Arbiters ,
T_ commands directly generated by CU

FIGURE A6 – iterative divide byte square root

example | global characteristics

instruction	CDIV R_d,R_s					.v1
comments	Calcul le quotient de la division de R_d par R_s et écrit le résultat dans R_d (logique combinatoire asynchrone)					
type / classe	CISC					
Byte encoding	number	2	opcode	Fpju d : s		
Byte contents	Byte 1	0x 74	Byte 2	0x 51	Byte 3	0x
generic description	If(Rs !=0) Rd =Rd/Rs					
generic mnemonic(s)	CDIV Rd, Rs			CDIV 0xds		
specific state(s)	number	3	S151,s152,s153			

s 151 Active Lpju et Epju pour envoyer les opérandes dans le bloc division
s 152 Maintient Epju actif et boucle tant que done n'est pas reçu
s 153 Enregistre le quotient dans le registre Rd

states	state subdiagram	CPI	5
s 1	AbArb(T_{pc}) . E_{as} . DbArb(T_{mem}) . E-U_{pc} . E_{ir}		
s 2	AbArb(T_{pc}) . E_{as} . DbArb(T_{mem}) . E-U_{pc} . X_{flag} . E_{ds} . E_{pl}		
S 151	Xrf, DbArb(EdbL && TrfL), 00.Lpju.Epju		
S 152	Epju, DbArb(EdbL && TpjuL)		
S 153	Wrfd, DbArb(EdbL && TpjuL), Wsr		

version with bus Arbiters

instruction	CDIV R_d,R_s					.v2
comments	Calcul le quotient de la division de R_d par R_s et écrit le résultat dans R_d (logique combinatoire asynchrone)					
type / classe	CISC					
Byte encoding	number	2	opcode	Fpju d : s		
Byte contents	Byte 1	0x 74	Byte 2	0x 51	Byte 3	0x
generic description	If(Rs !=0) Rd =Rd/Rs					
generic mnemonic(s)	CDIV Rd, Rs			CDIV 0xds		
specific µI(s)	number	3	S151,s152,s153			

µI 151 Active Lpju et Epju pour envoyer les opérandes dans le bloc division
µI 152 Maintient Epju actif et boucle tant que done n'est pas reçu
µI 153 Enregistre le quotient dans le registre Rd

µIs	µProgram	CPI	5
µI 1	T_{pc} . E_{as} . T_{mem} . E-U_{pc} . E_{ir}		
µI 2	T_{pc} . E_{as} . T_{mem} . E-U_{pc} . X_{flag} . E_{ds} . E_{pl}		
µI 151	Xrf, EdbL && TrfL, 00.Lpju.Epju		
µI 152	Epju, EdbL && TpjuL		
µI 153	Wrfd, EdbL && TpjuL, Wsr		

version without bus Arbiters ,
T_ commands directly generated by CU

FIGURE A7 – combinatory divide byte square root

example	global characteristics

instruction	CDIVW R_d,R_s					.v1
comments	Calcul le quotient de la division de R_s par R_t et écrit le résultat dans R_s (logique combinatoire asynchrone)					
type / classe	CISC					
Byte encoding	number	2	opcode Fpju d : s			
Byte contents	Byte 1	0x 7C	Byte 2	0x 71	Byte 3	0x
generic description	If(Rs !=0) Rd =Rd/Rs					
generic mnemonic(s)	CDIVW Rd,		CDIVW 0xds			
specific state(s)	number	3	S161, s162, s153			

s 161 Chargement des registres 16-bits dans la Pju
s 162 Boucle d'itération
s 153 Enregistre le quotient 16-bits dans le registre Rd

states	state subdiagram	CPI	5
s	1 AbArb(T_{pc}) . E_{as} . DbArb(T_{mem}) . E-U_{pc} . E_{ir}		
s	2 AbArb(T_{pc}) . E_{as} . DbArb(T_{mem}) . E-U_{pc} . X_{flag} . E_{ds} . E_{pl}		
S	161 Xrf, DbArb(EdbH.EdbL && TrfH & TrefL), 00.Lpju.Epju		
S	162 Epju, DbArb(EdbH.EdbL && TpjuH & TpjuL)		
S	153 Wrfd, DbArb(EdbL && TpjuL), Wsr		

version with bus Arbiters

instruction	CDIVW R_d,R_s					.v2
comments	Calcul le quotient de la division de R_s par R_t et écrit le résultat dans R_s (logique combinatoire asynchrone)					
type / classe	CISC					
Byte encoding	number	2	opcode Fpju d : s			
Byte contents	Byte 1	0x 7C	Byte 2	0x 71	Byte 3	0x
generic description	If(Rs !=0) Rd =Rd/Rs					
generic mnemonic(s)	CDIVW Rd, Rs		CDIVW 0xds			
specific µl(s)	number	3	S161, s162, s153			

µl 161 Chargement des registres 16-bits dans la Pju
µl 162 Boucle d'itération
µl 153 Enregistre le quotient 16-bits dans le registre Rd

µls	µProgram	CPI	5
µl	1 T_{pc} . E_{as} . T_{mem} . E-U_{pc} . E_{ir}		
µl	2 T_{pc} . E_{as} . T_{mem} . E-U_{pc} . X_{flag} . E_{ds} . E_{pl}		
µl	161 Xrf, EdbH.EdbL && TrfH & TrefL, 00.Lpju.Epju		
µl	162 Epju, EdbH.EdbL && TpjuH & TpjuL		
µl	153 Wrfd, EdbL && TpjuL, Wsr		

version without bus Arbiters ,
T_ commands directly generated by CU

FIGURE A8 – combinatory divide word square root

example	global characteristics

instruction	IDIVW R_d,R_s					.v1
comments	Calcul le quotient de la division de R_s par R_t et écrit le résultat dans R_s					
type / classe	CISC					
Byte encoding	number	2	opcode Fpju d : s			
Byte contents	Byte 1	0x 78	Byte 2	0x 81	Byte 3	0x
generic description	If(Rs !=0) Rd =Rd/Rs					
generic mnemonic(s)	IDIVW Rd, Rs		IDIVW 0xds			
specific state(s)	number	3	s161, s162, s153			

s 161 Chargement des registres 16-bits dans la Pju
s 162 Boucle d'itération
s 153 Enregistre le quotient 16-bits dans le registre Rd

states	state subdiagram	CPI	20
s	1 AbArb(T_{pc}) . E_{as} . DbArb(T_{mem}) . E-U_{pc} . E_{ir}		
s	2 AbArb(T_{pc}) . E_{as} . DbArb(T_{mem}) . E-U_{pc} . X_{flag} . E_{ds} . E_{pl}		
s	161 Xrf, DbArb(EdbH.EdbL && TrfH & TrefL), 00.Lpju.Epju		
s	162 Epju, DbArb(EdbH.EdbL && TpjuH & TpjuL)		
s	153 Wrfd, DbArb(EdbL && TpjuL), Wsr		

version with bus Arbiters

instruction	IDIVW R_d,R_s					.v2
comments	Calcul le quotient de la division de R_s par R_t et écrit le résultat dans R_s					
type / classe	CISC					
Byte encoding	number	2	opcode Fpju d : s			
Byte contents	Byte 1	0x 78	Byte 2	0x 81	Byte 3	0x
generic description	If(Rs !=0) Rd =Rd/Rs					
generic mnemonic(s)	IDIVW Rd, Rs		IDIVW 0xds			
specific µl(s)	number	3	s161, s162, s153			

µl 161 Chargement des registres 16-bits dans la Pju
µl 162 Boucle d'itération
µl 153 Enregistre le quotient 16-bits dans le registre Rd

µls	µProgram	CPI	20
µl	1 T_{pc} . E_{as} . T_{mem} . E-U_{pc} . E_{ir}		
µl	2 T_{pc} . E_{as} . T_{mem} . E-U_{pc} . X_{flag} . E_{ds} . E_{pl}		
µl	161 Xrf, EdbH.EdbL && TrfH & TrefL, 00.Lpju.Epju		
µl	162 Epju, EdbH.EdbL && TpjuH & TpjuL		
µl	153 Wrfd, EdbL && TpjuL, Wsr		

version without bus Arbiters ,
T_ commands directly generated by CU

FIGURE A9 – iterative divide word square root

FIGURE A10 – machine d'état

FIGURE A11 – adsub 16 bits

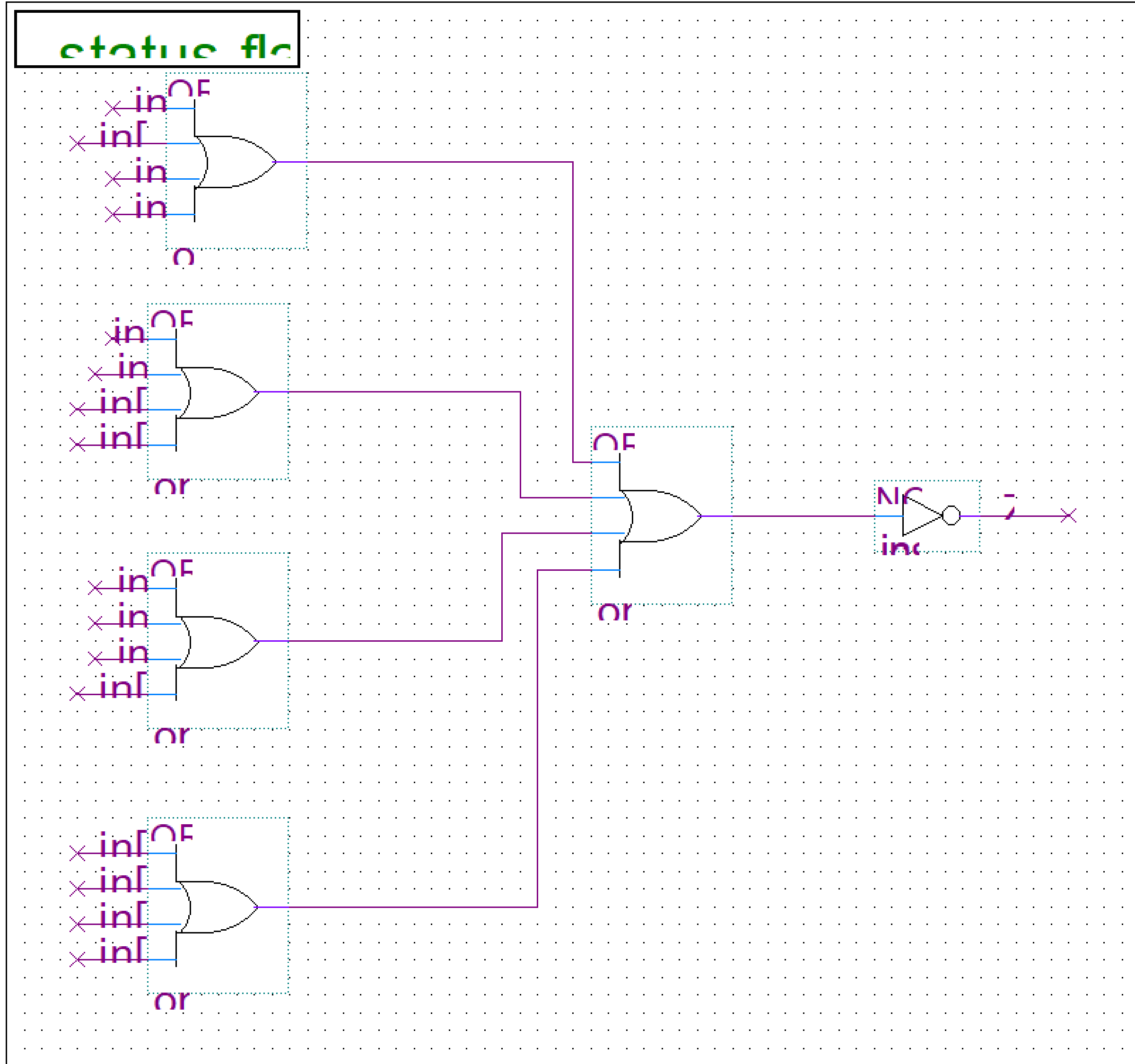


FIGURE A12 – zero flags

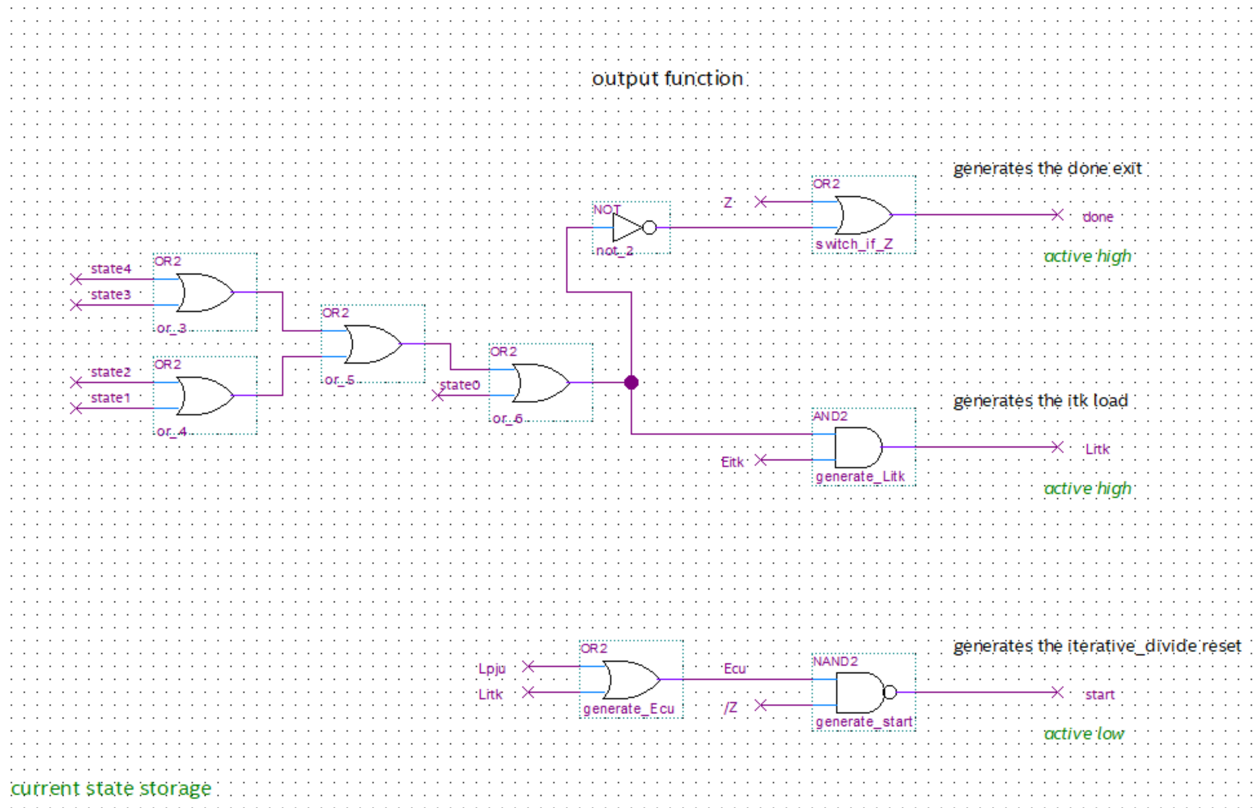


FIGURE A13 – output function

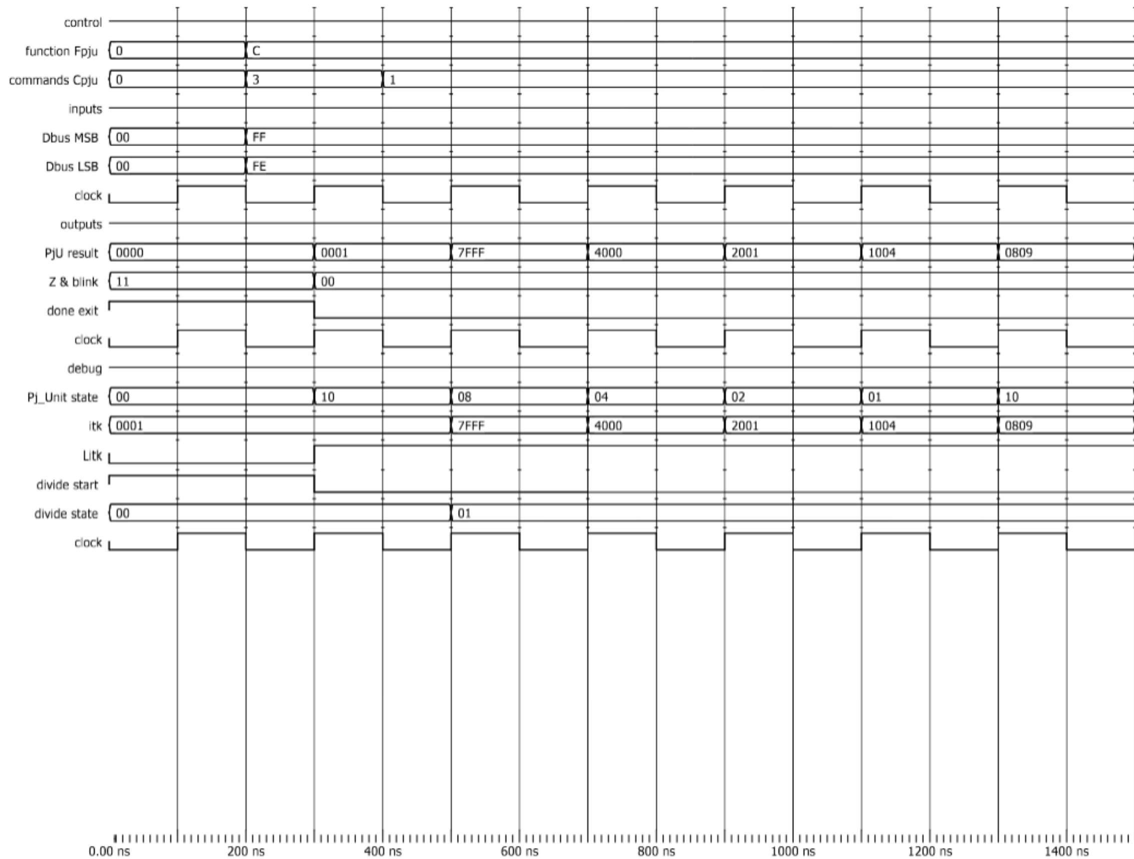


FIGURE A14 – Chronogramme Pj_unit

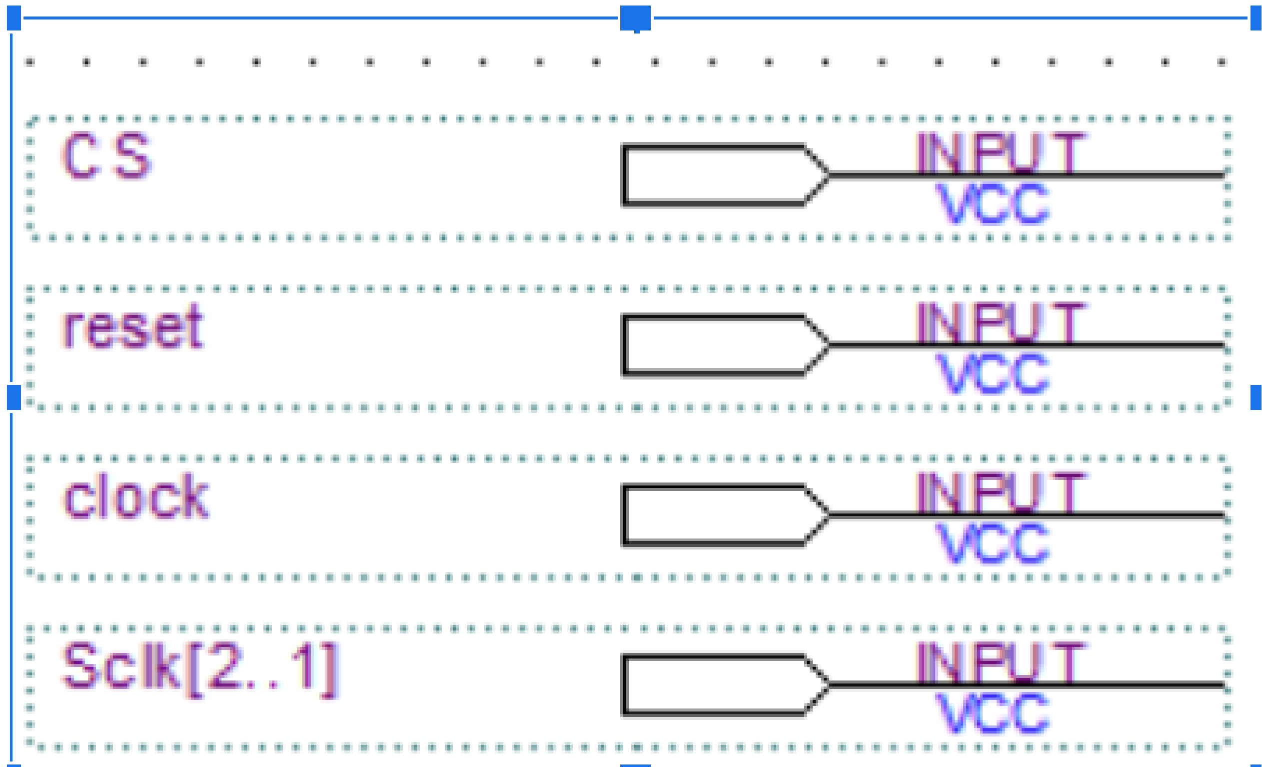


FIGURE A15 – implementation quartus FDU 1

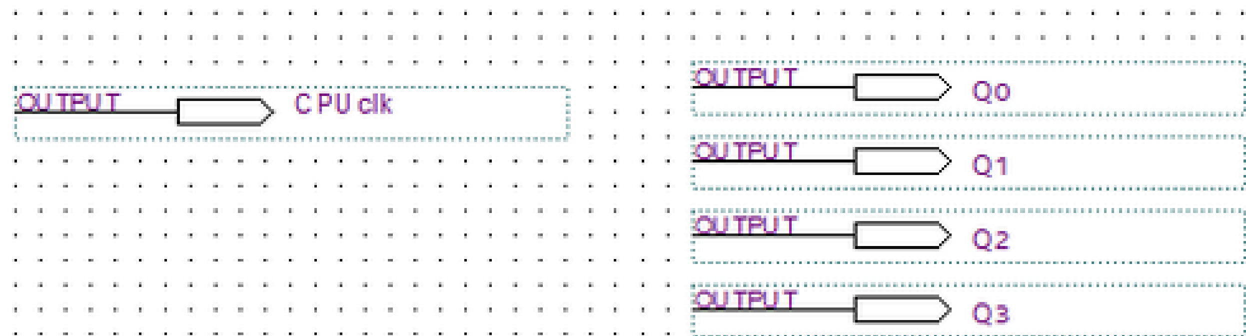


FIGURE A16 – implementation quartus FDU 2

Figure A17 – implementation quartus FDU 3

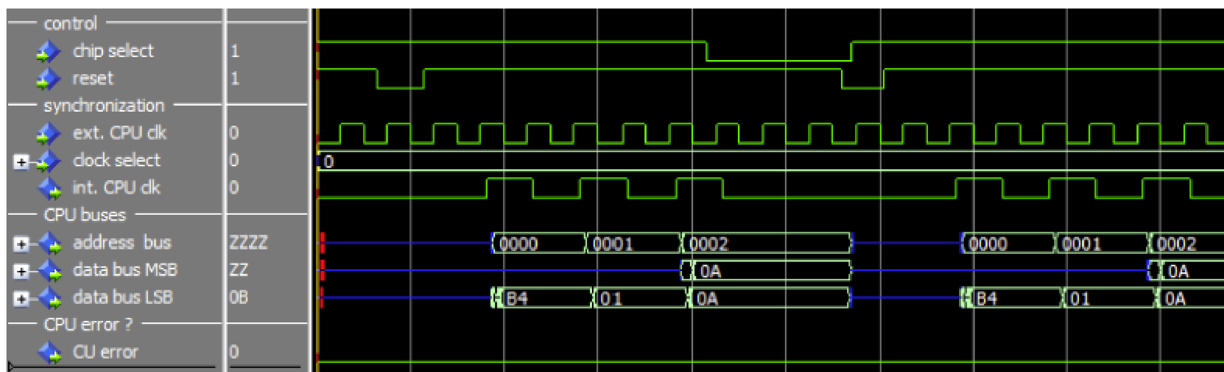


Figure A18 – Chronogramme FDU 1

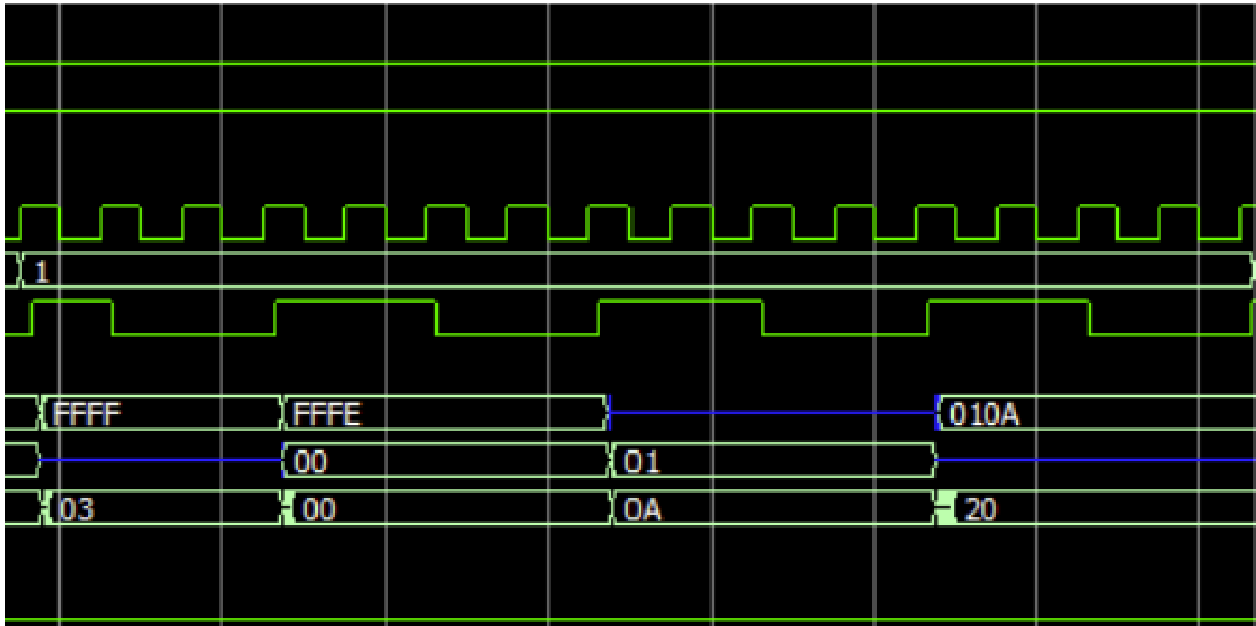
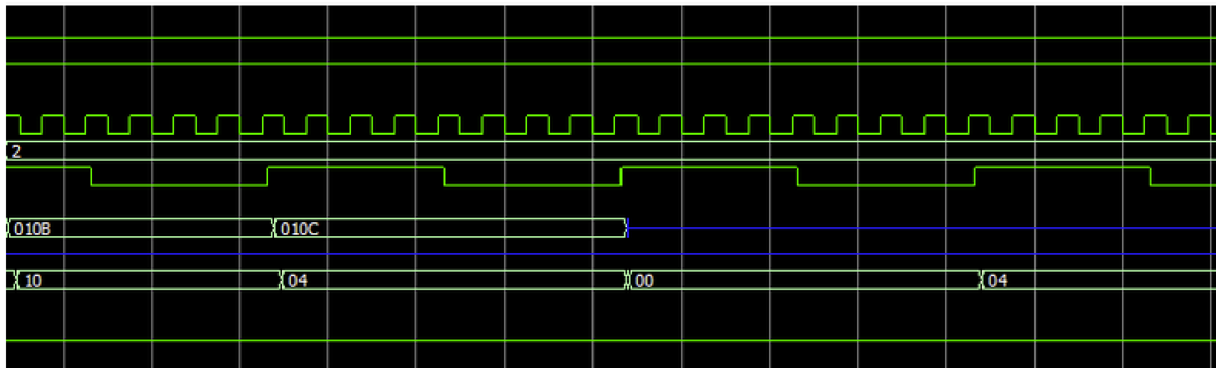


Figure A19 – Chronogramme FDU 2



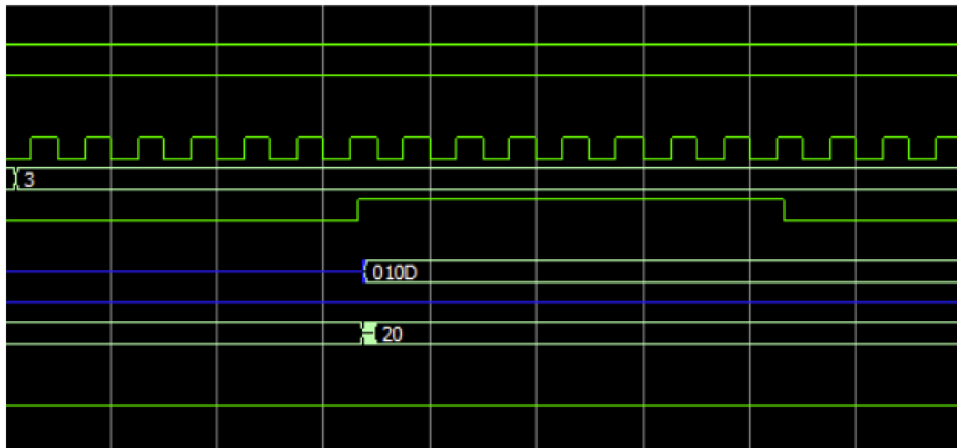


Figure A20 – Chronogramme FDU 3

Addr	+0	+1	+2	+3	+4	+5	+6	+7	ASCII
00	20	48	65	6C	6C	6F	20	20	Hello
08	20	77	6F	72	6C	64	21	20	world!
10	20	41	72	63	68	69	00	20	Archi.
18	20	20	20	20	69	73	20	20	is
20	20	67	72	65	61	74	21	20	great!
28	20	8D	7E	20	20	20	01	20	.~ .
30	49	74	27	73	20	4F	4B	3F	It's OK?
38	50	65	72	66	65	63	74	21	Perfect!

FIGURE A21 – matiere a reflexion 1

Addr	+0	+1	+2	+3	+4	+5	+6	+7	ASCII
00	00	00	00	00	00	00	00	00	....
08	00	00	00	00	00	00	00	00	....
10	00	00	00	00	00	00	00	00	....
18	00	00	00	00	00	00	00	00	....

FIGURE A22 – matiere a reflexion 2