INSA INSTITUT NATIONAL DES SCIENCES APPLIQUÉES TOULOUSE

# Conception d'un microprocesseur de type RISC

Ecrit par Simon Paris et Cédric Chanfreau le 02/12/2023

Ingénieur en Alternance Automatique Électronique

à l'Institut National des Sciences Appliquées

INSA TOULOUSE

INSA INSTITUT NATIONAL DES SCIENCES APPLIQUÉES TOULOUSE

# Conception d’un microprocesseur de type RISC

Ecrit par Simon Paris et Cédric Chanfreau le 02/12/2023

Ingénieur en Alternance Automatique Électronique

à l’Institut National des Sciences Appliquées

INSA TOULOUSE

## TABLE DES MATIERES

# INTRODUCTION

Notre projet est de concevoir un microprocesseur de type RISC, désigné en VHDL sur l'outil de développement, de simulation et d'implémentation Vivado. Cette architecture promet une exécution plus rapide des programmes grâce a ses 5 niveaux de pipeline. Ce rapport sera structuré de manière à couvrir les différentes phases du projet.

Nous commencerons par présenter l'architecture du microprocesseur avec les différents composants clés tels que l'Unité Arithmétique et Logique (ALU), le Banc de Registres, la Mémoire d'Instructions, la Mémoire de Données, et d'autres éléments essentiels qui guideront la conception de notre microprocesseur. Nous aborderons également la gestion des aléas, un aspect crucial de la conception.

Enfin, nous analyserons les résultats de synthèse soulignant les optimisations réalisées pour augmenter la fréquence de fonctionnement ainsi que la diminution de sa consommation énergétique.

Note :

Le présent rapport n'a pas vocation à être imprimé, il serait illisible. Nous conseillons donc sa lecture au format PDF. Les captures d'écrans étant qualitatives, il est possible zoomer dessus pour analyser leur contenu (notamment les chronogrammes). Nous conseillions également de lire en priorité la partie I.5.4 avant la partie I.5.3, cela évitera toute confusion.

# I) L'ARCHITECTURE DU MICROPROCESSEUR RISC

Dans cette partie nous nous pencherons sur l'architecture d'un microprocesseur RISC, nous ne rentrerons cependant pas dans les détails. En effet il existe beaucoup de documentation à ce sujet, qui sont claires et synthétiques, elles n'ont pas besoin de copie (qui elle serait approximative). Nous présenterons donc de manière fonctionnelle les 5 éléments présents dans notre microprocesseur l'ALU, la mémoire données, la mémoire d'instruction et le banc de registres.

## I.1) UNITE ARITHMETIQUE ET LOGIQUE

### I.1.1) DESCRIPTIONS FONCTIONNELLES

L'Unité Arithmétique et Logique (ALU) est le composant central permettant l'exécution des opérations arithmétiques et logiques du microprocesseur. Le schéma fonctionnel de l'ALU est donné ci-dessous en figure 1. Le Bus de contrôle (Ctrl_ALU) détermine le type d'opération à effectuer. L'ALU prend deux entrées A et B de 8 bits, qui seront les opérandes et S le résultat de l'opération également sur 8 bits. Nous avons également en sortie 4 flags, Negatif, Carry, Zero et Overflow.

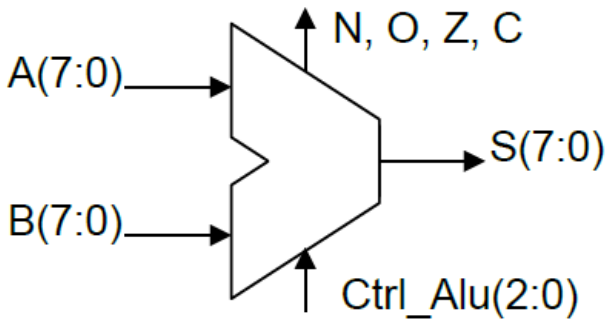



**Figure 1 Schéma fonctionnel d'un ALU**

### I.1.2) SIMULATION FONCTIONNELLE

Vous trouverez ci-dessous en Figure 2 l'une des simulations fonctionnelles de notre ALU. Cette simulation ne dépendant pas du temps et de l'implémentation elle nous permet uniquement de vérifier le fonctionnement logique du circuit.

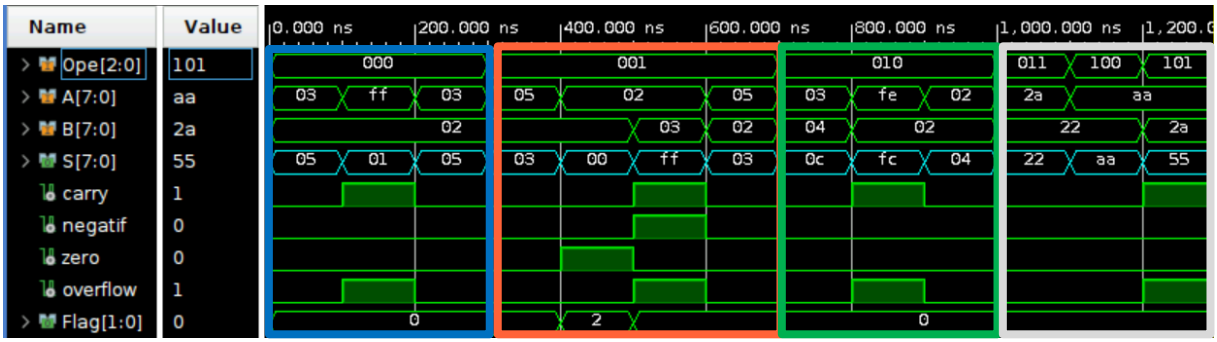



**Figure 2 Simulation fonctionnelle ALU**

Ci-dessous vous trouverez le tableau N°1 permettant de faire le lien entre bus de contrôle et les opérations arithmétiques et logiques réalisées.

**Tableau 1 OPCODE ALU**

| | | |
|---|---|---|
| Addition | Ope="000" | S = A + B |
| Soustraction | Ope="001" | S = A - B |
| Multiplication | Ope="010" | S = A x B |
| ET logique | Ope="011" | S = A AND B |
| OU logique | Ope="100" | S = A OR B |
| NOT logique | Ope="101" | S = NOT A |

Le résultat de chaque opération est stocké dans la sortie S. De plus, le composant génère des informations sur le résultat sous forme de drapeaux dans la sortie Flag, indiquant si une opération a généré une retenue, si le résultat est négatif, s'il est égal à zéro, ou s'il y a eu un dépassement.

## I.2) BANC DE REGISTRES

### I.2.1) DESCRIPTIONS FONCTIONNELLE

Le Banc de Registres, intégré dans notre architecture, représente un élément fondamental pour le stockage temporaire des données pendant l'exécution des instructions. Il est constitué de 16 registres de 8 bits, offrant une capacité de stockage significative pour les opérations en cours. Cette configuration permet d'effectuer simultanément des opérations d'accès en lecture via les ports QA et QB, ainsi que des opérations d'écriture via le port W_in.

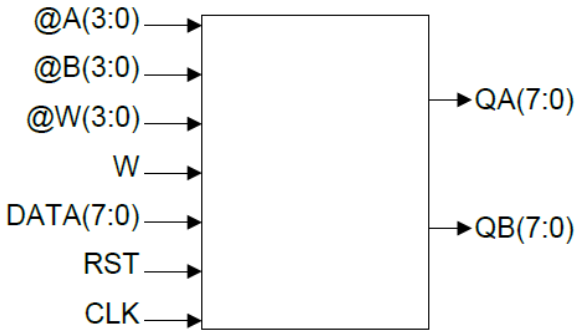



**Figure 3 Schéma fonctionnel Banc de registres**

### I.2.2) SIMULATION FONCTIONNELLE

Une simulation a été effectuée pour évaluer la lecture, l'écriture et le comportement général du banc de registres. Les valeurs ont été manipulées pour garantir une gestion appropriée des données en fonction de l'horloge. Cela inclut des scénarios tels que la lecture et l'écriture simultanée sur une sortie (bypass), ou encore la lecture simple dans un registre via le port d'écriture.

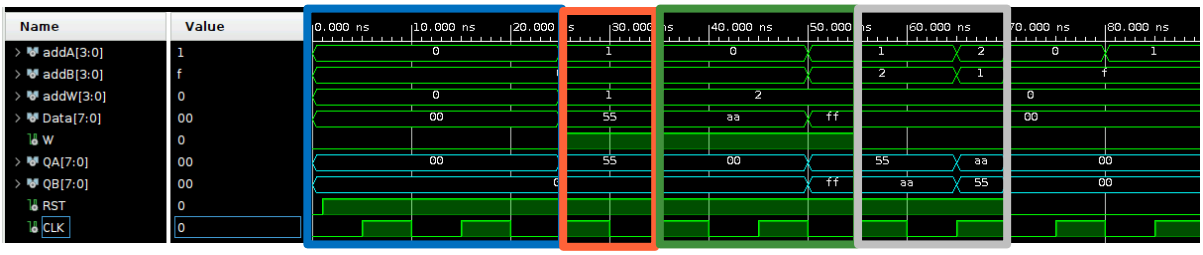



**Figure 4 Simulation fonctionnelle Banc de registres**

| | |
|---|---|
| Test 1 | - Sélection de l'adresse 0 sur addA et addB<br>- Mode lecture de la donnée 0 |
| Test 2 | - Sélection de l'adresse 1 sur addA<br>- Ecriture de la donnée 55 sur addA (**bypass***) |
| Test 3 | - Sélection de l'adresse 0 puis 2 sur addB<br>- Ecriture de la donnée aa puis ff sur addB (sélectionné avec addW) |
| Test 4 | - Lecture de addA et addB avec la donnée précédemment écrite suivi par l'activation du reset |

*Le bypass, est une technique permettant de lire et d'écrire sur un registre en même temps. Il est utilisé dans les processeurs pour améliorer l'efficacité en redirigeant directement les résultats calculés vers les instructions suivantes qui en ont besoin, sans attendre l'écriture dans le registre.

## I.3) MEMOIRE DE DONNEES

### I.3.1) DESCRIPTIONS FONCTIONNELLES

La mémoire de données est utilisée pour stocker des informations temporaires nécessaires au fonctionnement d'un programme. Les opérations sur la mémoire de données incluent la lecture des valeurs stockées à des adresses spécifiques et l'écriture de nouvelles valeurs à ces adresses.

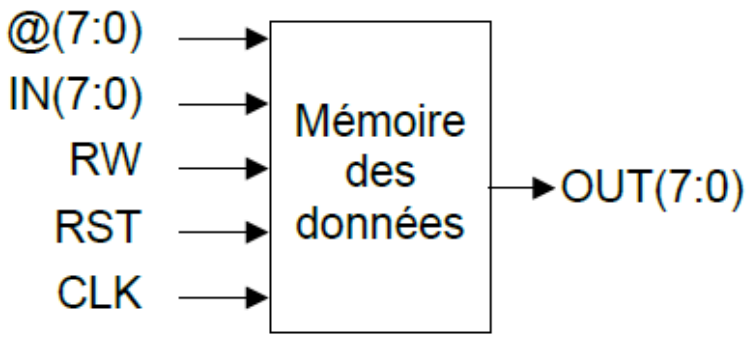



**Figure 5 Schéma fonctionnel Mémoire de données**

@(7:0) : Adresse à laquelle les opérations de lecture ou écriture doivent être effectuées.
IN(7:0) : Entrée de données à écrire dans la mémoire à l'adresse spécifiée.
RW : Mode d'opération de la mémoire (0 pour lecture, 1 pour écriture).
RST : Entrée de réinitialisation de la mémoire.
CLK : Entrée de l'horloge servant à déclencher les opérations de la mémoire.
OUT(7:0) : Sortie renvoyant les données lues à partir de l'adresse spécifiée.

### I.3.2) Simulation fonctionnelle

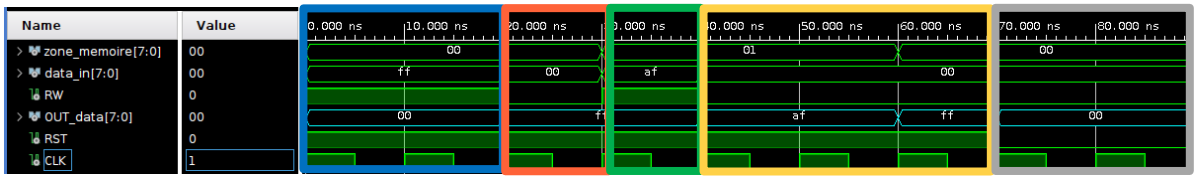



**Figure 6 Simulation fonctionnelle Mémoire de données**

| | |
|---|---|
| Test 1 | Ecriture de la donnée ff sur le registre 0 |
| Test 2 | Lecture de la donnée écrite sur le registre 0 |
| Test 3 | Ecriture de la donnée af sur le registre 1 |
| Test 4 | Lecture du registre 1 puis le registre 0 |
| Test 5 | Mise à 1 du reset |

À chaque front montant de l'horloge (CLK), la mémoire réagit en fonction des signaux de contrôle (RW) et de réinitialisation (RST). En cas de réinitialisation (RST à 1), la mémoire est remise à zéro. En mode écriture (RW à 1), les données d'entrée (data) sont écrites à l'adresse spécifiée. En mode lecture (RW à 0), les données situées à l'adresse spécifiée sont renvoyées via la sortie (OUT).

## I.4) MEMOIRE D'INSTRUCTION

### I.4.1) DESCRIPTIONS FONCTIONNELLE

La mémoire d'instructions stocke la liste d'instructions machine qui composent le programme exécuté par le processeur. Chaque instruction est associée à une adresse mémoire spécifique. Chaque instruction est représentée sur 32 bits et est accessible à une adresse spécifique dans la mémoire.

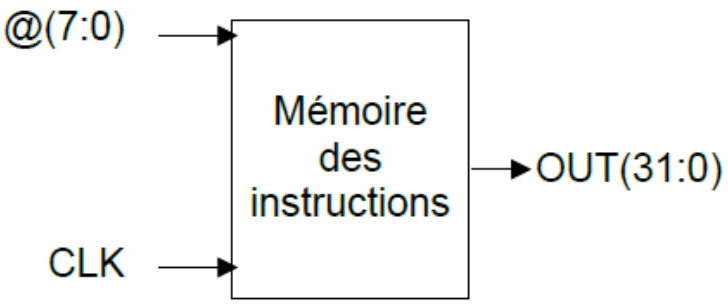



**Figure 7 Schéma fonctionnel Mémoire d'instructions**

A [7 :0] : Entrée représentant l'adresse à laquelle les opérations de lecture doivent être effectuées.

S [31 :0] : Sortie renvoyant l'instruction de 32 bits lue à partir de l'adresse spécifiée

### I.4.2) Simulation fonctionnelle

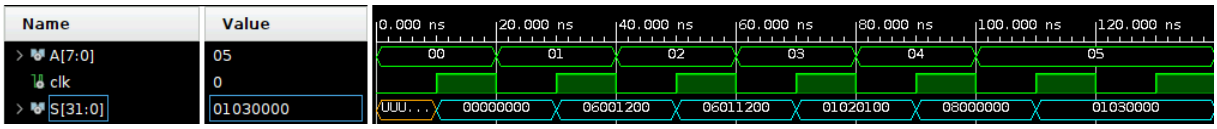



**Figure 8 Simulation fonctionnelle Mémoire d'instructions**

| | |
|---|---|
| Test | Ecriture de l'instruction dans le registre sélectionné par A[7:0] sur front montant |

À chaque front montant de l'horloge (clk), la mémoire réagit en renvoyant l'instruction située à l'adresse spécifiée via la sortie S.

### I.5.1) GESTION DESCRIPTIONS FONCTIONNELLE DES ALEA

On distingue les aléas de données et les aléas de branchement. Nous nous intéresserons aux aléas de données. Un alea de donnée est un évènement, cet évènement est déclenché par au moins deux instructions s'exécutant de manière consécutive et de manière incompatible. Une première instruction modifie la valeur d'une donnée dans une zone mémoire, la seconde utilise cette zone mémoire alors que la valeur n'a pas été mise à jour. Pour répondre à cette problématique nous avons créé un nouveau composant dans notre chemin de données, que vous pouvez voir ci-dessous en figure 9.

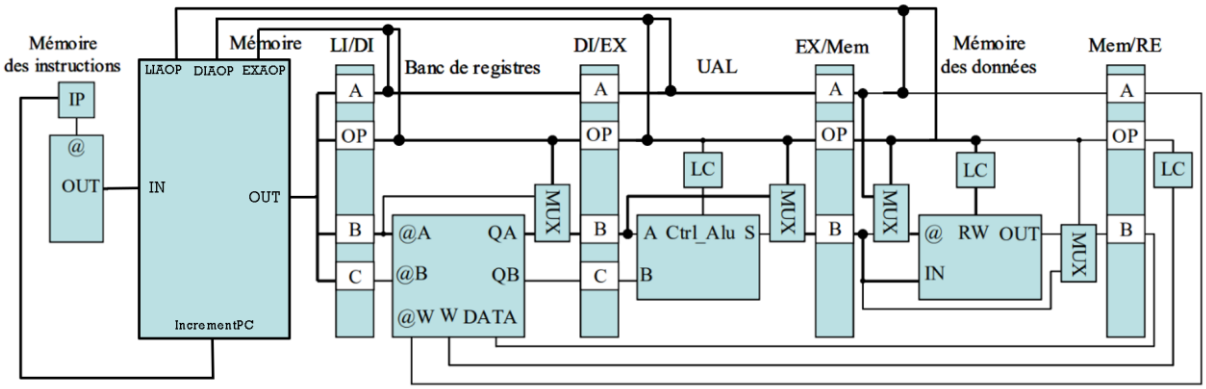



**Figure 9 Chemin de données avec gestion des aléas**

Si le composant détecte que l'instruction envoyée par la mémoire d'instruction, provoque un aléa avec une des instructions déjà présentes dans le chemin de données, alors il bloque la mémoire d'instruction et **envoie des NOP** dans le chemin de donnée. Une fois l'aléa supprimé il envoie l'instruction bloquée et débloque la mémoire d'instruction.

On peut souligner que la détection d'aléa se fera uniquement sur les 3 premiers étages, en effet le banc registre contient un bypass qui permet de lire et d'écrire en même temps.

### I.5.2) CONDITIONS DE DETECTION D'ALEA

Nous souhaitons avoir les conditions les plus général possible pour détecter nos alea, cela évitera toute sorte d'ambigüité et nous permettra de gérer simplement tous les cas possibles. On peut commencer par les aléas provoqués par les instructions MUL, SOU et ADD.

| Opération | Code | Format d'instruction | | | | Description |
|---|---|---|---|---|---|---|
| | | OP | A | B | C | |
| Addition | 0x01 | ADD | Ri | Rj | Rk | [Ri] ← [Rj] + [Rk] |
| Multiplication | 0x02 | MUL | Ri | Rj | Rk | [Ri] ← [Rj] * [Rk] |
| Soustraction | 0x03 | SOU | Ri | Rj | Rk | [Ri] ← [Rj] - [Rk] |
| | | | | | | |
| Copie | 0x05 | COP | Ri | Rj | _ | [Ri] ← [Rj] |
| Affectation | 0x06 | AFC | Ri | j | _ | [Ri] ← j |
| Chargement | 0x07 | LOAD | Ri | @j | _ | [Ri] ← [@j] |
| Sauvegarde | 0x08 | STORE | @i | Rj | _ | [@i] ← [Rj] |

Si l'instruction rentrante est ADD, MUL ou SOU alors zone B et C doivent être différentes des zones A des 3 instructions en cours d'exécution. On ne déclenche pas d'alea sur les instructions en cours d'exécution qui seraient des STORE.

On peut maintenant poursuivre par les aléas provoquer par l'instructions COP et STORE.

| Opération | Code | Format d'instruction | | | | Description |
|---|---|---|---|---|---|---|
| | | OP | A | B | C | |
| Addition | 0x01 | ADD | Ri | Rj | Rk | [Ri] ← [Rj] + [Rk] |
| Multiplication | 0x02 | MUL | Ri | Rj | Rk | [Ri] ← [Rj] * [Rk] |
| Soustraction | 0x03 | SOU | Ri | Rj | Rk | [Ri] ← [Rj] - [Rk] |
| | | | | | | |
| Copie | 0x05 | COP | Ri | Rj | _ | [Ri] ← [Rj] |
| Affectation | 0x06 | AFC | Ri | j | _ | [Ri] ← j |
| Chargement | 0x07 | LOAD | Ri | @j | _ | [Ri] ← [@j] |
| Sauvegarde | 0x08 | STORE | @i | Rj | _ | [@i] ← [Rj] |

Si l'instruction rentrante est COP ou STORE la zone B doit être différente des zones A des 3 instructions en cours d'exécution.

On peut souligner qu'une instruction rentrante de type AFC ou LOAD, ne peuvent pas provoquer d'alea, et exerceront une influence uniquement si elles sont déjà présentes dans le chemin de données. On doit également détecter un alea uniquement sur des instructions, il faut donc vérifier l'opcode pour ne pas déclencher d'alea sur un NOP.

Note :( Nous conseillions de lire en priorité la partie I.5.4 avant la partie I.5.3)

### I.5.3) SIMULATION FONCTIONNELLE DES ALEA

Un point a souligné est l'instruction NOP, en effet un NOP ne doit rien faire est par cela on n'entend qu'il ne doit pas modifier un registre ou un espace mémoire, il peut cependant rentrer dans l'alu, provoquer une lecture du banc de registre ou de la mémoire. D'un point de vue extérieur, ces interactions ne provoquent pas de comportement indésirable de la part du microprocesseur, cela étant il provoque une consommation inutile.

#### I.5.3.1) ALEA SIMPLE (EXEMPLE : ADD)

Une première alea simple à percevoir, est une succession deux 2 AFC suivie d'une addition des deux registres, cette situation est simulée dans le chronogramme ci-dessous.

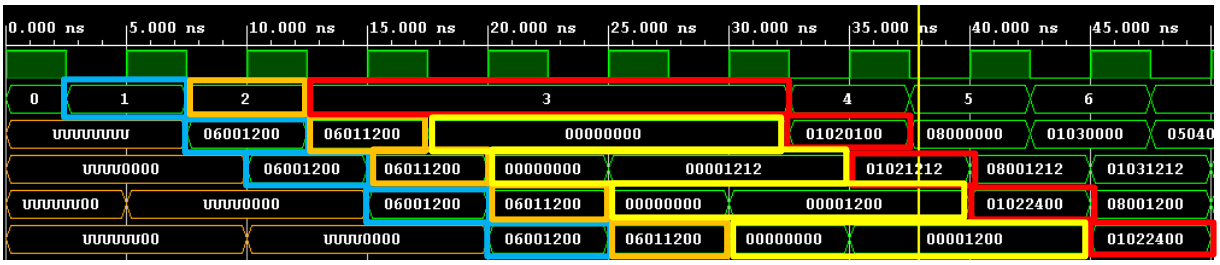



**Figure 10 Simulation fonctionnelle alea exemple ADD**

On envoie une première instruction « 06001200 » = R0<= 12

On envoie une seconde instruction « 06001200 » = R1<= 12

On envoie une troisième instruction « 01020100 » = R2<= R1+R0

Evidement les registres R0 et R1 ne sont pas à jour au moment de l'addition donc une « bulle » (ici en jaune) est insérer, cette bulle est d'une durée de 3 NOP.

### I.5.3.2) ALEA COMPLEXE (EXEMPLE : COP)

Un second exemple d'alea, est un alea entrainé par deux instructions non successives, cette situation est simulée dans le chronogramme ci-dessous.

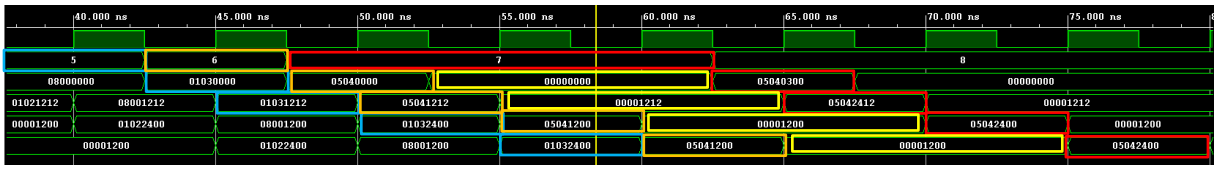



**Figure 11 Simulation fonctionnelle alea exemple COP**

On envoie une première instruction « 01030000 » = R3<= R0+R0

On envoie une seconde instruction « 05040000 » = R4<= R5

On envoie une troisième instruction « 05040300 » = R4<= R3

Ici l'alea est entre la troisième et la première instruction, la bulle(ici en jaune) est équivalente à 2 NOP. On constate à travers cette situation que le nombre de NOP s'adapte en fonction de la distance séparant les aléas.

### I.5.3.3) ALEA COMPLEMENTAIRE (EXEMPLE : STORE)

Enfin on peut observer un dernier exemple d'alea, c'est un alea entrainé par deux instructions non successives, cette situation est simulée dans le chronogramme ci-dessous.

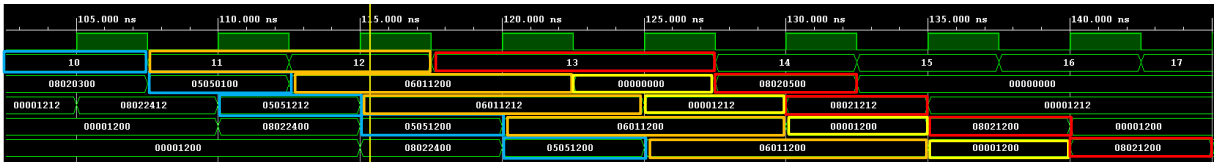



**Figure 12 Simulation fonctionnelle alea exemple Store**

On envoie une première instruction « 05050100 » = R5<= R1

On envoie une seconde instruction « 06001200 » = R0<= 12

On envoie une troisième instruction « 06001200 » = R0<= 12

On envoie une quatrième instruction « 08020500 »= @02 <=R5

Ici l'aléa est entre la quatrième et première instruction la bulle(ici en jaune) est équivalente à 1 NOP. On constate a travers cette situation que le nombre de NOP s'adapte en fonction de la distance séparant les aléas et que nous avons traiter tous les cas possibles.

### I.5.4) / ! \NOTE SUR LES CHRONOGRAMMES DES SIMULATIONS

On ne peut pas présenter un test qui soit exhaustif sur les aléas, cela serait trop long. Nous avons donc choisi de vous présenter trois cas possibles, ils illustrent les différentes conditions d'un alea. En parcourant la simulations complète du processeur vous trouverez les autres cas d'aléas.

Si on observe attentivement les simulations un détail surprenant apparait. En effet, sur les simulations il semble que l'étage 2 soit décalé des autres étages, ce qui est en contradiction avec un fonctionnement pipeline. En réalité il s'agit d'un faux décalage, sur les simulations précédentes on observe l'entrée des buffers mais la transmission se fait en sortie sur une clock qui est mutualisée. Un moyen simple de le vérifier est de regarder l'une des sorties des MUXS.

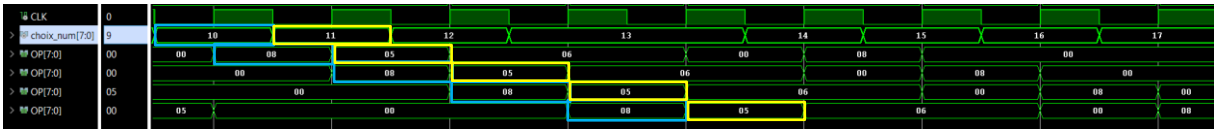



**Figure 13 Simulation du Pipeline**

L'effet pipeline est bien nominale, seul le compteur PC (ici nommée choix_num) est décalé cela ne pose évidemment pas de soucis et cela sera explicité dans la partie 3 de ce rapport.

# II) IMPLEMENTATION DU MICROPROCESSEUR RISC

## II.1) CHOIX D'IMPLEMENTATION

Nous aurions pu implémenter nos composants et les faires fonctionner de différentes manières. Pour les composants nous nous sommes contentés de les concevoir celons les indications du sujet. Pour le chemin de donnes nous avons décidé d'implanter les 4 MUX délimitant les étages du pipeline sous forme de composant. Les composants discrets type LC et simple MUX ont été implémentés de manière concurrente directement dans le chemin de données. Nous pensons que malgré la perte de lisibilité, cela accroit les performances de notre microprocesseur.

## II.2) CHOIX D'AMELIORATION ANALYSE

### II.2.1) AXE D'AMELIORATION

A la fin du projet plusieurs possibilités s'offrent à nous, on peut améliorer notre processeur.

Pour cela différentes opportunités se présentent à nous , le premier est de compléter notre jeu d'instruction notamment en ajourant des fonctions telles que le OU logique, ET logique ect. Ces fonctions sont déjà intégrées dans notre ALU, il faudrait donc modifier notre chemin de données et notre gestion des aléas.

Un autre axe d'amélioration est l'ajout d'instruction de type saut et saut conditionné. Cet axe est intéressant car il permet de réaliser des algorithmes plus complexes.

Un quatrième axe d'amélioration est de réduire aux maximum la consommation de notre microprocesseur et sa superficie.

Enfin un dernier axe possible est l'optimisation du système pour augmenter sa fréquence de fonctionnement. Il s'agit du choix que nous avons décidé de faire, il est purement arbitraire.

## II.2.2) ANNALYSE FREQUENTIELLE ET CHEMIN CRITIQUE

Notre première rapport Timming nous indique une fréquence une fréquence MAX de 65MHz, ce résultat ne nous satisfait évidemment pas.

| Setup | | Hold | | Pulse Width | |
|---|---|---|---|---|---|
| Worst Negative Slack (WNS): | 4,762 ns | Worst Hold Slack (WHS): | 0,030 ns | Worst Pulse Width Slack (WPWS): | 9,500 ns |
| Total Negative Slack (TNS): | 0,000 ns | Total Hold Slack (THS): | 0,000 ns | Total Pulse Width Negative Slack (TPWS): | 0,000 ns |
| Number of Failing Endpoints: | 0 | Number of Failing Endpoints: | 0 | Number of Failing Endpoints: | 0 |
| Total Number of Endpoints: | 2116 | Total Number of Endpoints: | 2116 | Total Number of Endpoints: | 1099 |

**All user specified timing constraints are met.**

**Figure 14 Rapport Timming**

On pourrait penser que la gestion des aléas sur front descendant diminue notre fréquence max, cependant le chemin critique nous indique que nous devrions nous pencher sur nos mémoires.

Après avoir optimisé notre chemin de mémoire, nous avons un schématique d'implémentation plus claire :

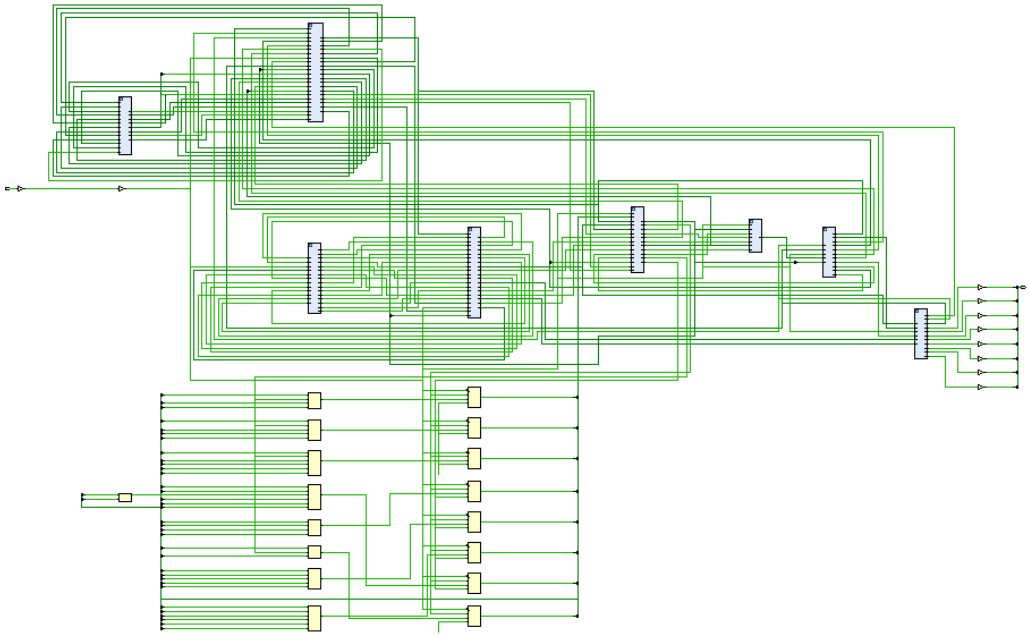


**Figure 15 Schématique TOP**

Cependant notre fréquence n'est pas plus importante, on regarde donc de nouveau notre chemin critique :

```
Max Delay Paths
--------------------------------------------------------------------------------------
Slack (MET) :             4.442ns  (required time - arrival time)
  Source:                 MEMOIR_INSTRUCTION/OUT_instru_reg[8]/C
                            (rising edge-triggered cell FDRE clocked by CLKTOP  {rise@0.000ns fall@10.000ns period=20.000ns})
  Destination:            choix_num_reg[6]/R
                            (falling edge-triggered cell FDRE clocked by CLKTOP  {rise@0.000ns fall@10.000ns period=20.000ns})
  Path Group:             CLKTOP
  Path Type:              Setup (Max at Slow Process Corner)
  Requirement:            10.000ns  (CLKTOP fall@10.000ns - CLKTOP rise@0.000ns)
  Data Path Delay:        4.979ns  (logic 1.090ns (21.890%)  route 3.889ns (78.110%))
  Logic Levels:           4  (LUT6=4)
  Clock Path Skew:        -0.024ns (DCD - SCD + CPR)
    Destination Clock Delay (DCD):    5.078ns = ( 15.078 - 10.000 )
    Source Clock Delay      (SCD):    5.389ns
    Clock Pessimism Removal (CPR):    0.286ns
  Clock Uncertainty:      0.035ns  ((TSJ^2 + TIJ^2)^1/2 + DJ) / 2 + PE
    Total System Jitter     (TSJ):    0.071ns
    Total Input Jitter      (TIJ):    0.000ns
    Discrete Jitter          (DJ):    0.000ns
    Phase Error              (PE):    0.000ns

    Location             Delay type                Incr(ns)  Path(ns)    Netlist Resource(s)
```

Malgré nos efforts nous n'avons pas réussi à diminuer les temps de propagation à travers ce chemin.

## II.3) PERFORMANCE

### III.3.1) CONSOMMATION ENERGETIQUE

Bien que nos modifications n'aient pas permis d'augmenter la fréquence la consommation énergétique à fortement diminué. La diminution est de l'ordre de 175%. Vous trouverez ci-dessous le rapport récapitulatif de puissance.

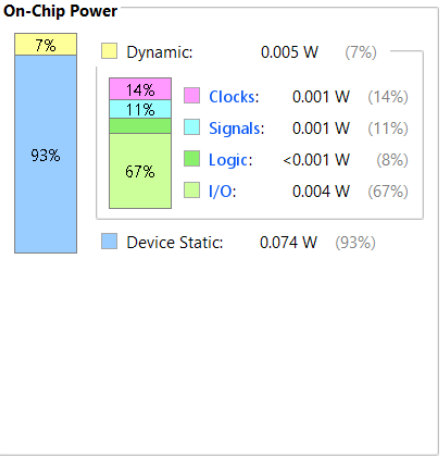
**Summary**

Power analysis from Implemented netlist. Activity derived from constraints files, simulation files or vectorless analysis.

| | |
|---|---|
| **Total On-Chip Power:** | **0.079 W** |
| **Design Power Budget:** | **Not Specified** |
| **Process:** | typical |
| **Power Budget Margin:** | **N/A** |
| **Junction Temperature:** | **25,4°C** |
| Thermal Margin: | 59,6°C (11,9 W) |
| Ambient Temperature: | 25.0 °C |
| Effective ϑJA: | 5,0°C/W |
| Power supplied to off-chip devices: | 0 W |
| Confidence level: | Medium |

Launch Power Constraint Advisor to find and fix invalid switching activity

**On-Chip Power**



### III.3.2) COMPOSANTS

Notre consommation est intrinsèquement liée à notre fréquence et à notre nombre de composants. Vous trouverez ci-dessous, des extraits de rapport indiquant les différents composant utilisés pour notre projet et en quelles proportions.

| Site Type | Used | Fixed | Prohibited | Available | Util% |
|---|---|---|---|---|---|
| Slice LUTs* | 159 | 0 | 0 | 20800 | 0.76 |
| LUT as Logic | 159 | 0 | 0 | 20800 | 0.76 |
| LUT as Memory | 0 | 0 | 0 | 9600 | 0.00 |
| Slice Registers | 160 | 0 | 0 | 41600 | 0.38 |
| Register as Flip Flop | 160 | 0 | 0 | 41600 | 0.38 |
| Register as Latch | 0 | 0 | 0 | 41600 | 0.00 |
| F7 Muxes | 0 | 0 | 0 | 16300 | 0.00 |
| F8 Muxes | 0 | 0 | 0 | 8150 | 0.00 |

| Site Type | Used | Fixed | Prohibited | Available | Util% |
|---|---|---|---|---|---|
| BUFGCTRL | 1 | 0 | 0 | 32 | 3.13 |
| BUFIO | 0 | 0 | 0 | 20 | 0.00 |
| MMCME2_ADV | 0 | 0 | 0 | 5 | 0.00 |
| PLLE2_ADV | 0 | 0 | 0 | 5 | 0.00 |
| BUFMRCE | 0 | 0 | 0 | 10 | 0.00 |
| BUFHCE | 0 | 0 | 0 | 72 | 0.00 |
| BUFR | 0 | 0 | 0 | 20 | 0.00 |

| Ref Name | Used | Functional Category |
|---|---|---|
| FDRE | 160 | Flop & Latch |
| LUT6 | 84 | LUT |
| LUT4 | 34 | LUT |
| LUT2 | 34 | LUT |
| LUT5 | 25 | LUT |
| CARRY4 | 9 | CarryLogic |
| OBUF | 8 | IO |
| LUT3 | 6 | LUT |
| LUT1 | 2 | LUT |
| IBUF | 1 | IO |
| BUFG | 1 | Clock |

# CONCLUSION

Le projet a été très intéressant à réaliser, la découverte du logiciel Vivado a été instructive pour nous qui n'avions travaillé que sur Qartus. Ce projet nous a également permis d'appliquer de manière concrète les différentes notions vues en cours de modélisation des composants et architectures numériques.

Nous avons rencontré de nombreuses difficultés, notamment lors de la tentative d'optimisation des chemins critiques, et nous ne sommes finalement pas parvenus à nos fins. Malgré cet échec nous sommes satisfaits de notre gestions des aléas. De plus, la démarche nous a permis d'aller plus loin dans l'utilisation des différents outils proposés par Vivado, et sur la conception Vhdl de façon générale.

Les connaissances et la compréhension de l'architecture d'un microprocesseur nous sont également utiles, on peut qualifier ces connaissances de compétences transversales.