Conception et Synthèse d'un Microprocesseur 32-Bits

Contrôle de connaissances

du cours

Synthèse

Markus Mück

Partie I

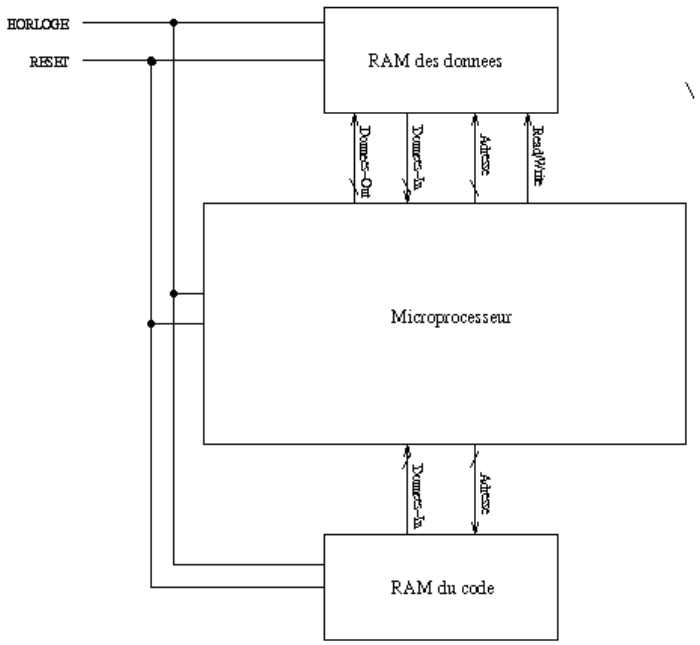
Spécifications du Microprocesseur 32-Bits

[redacted]

1. Spécification d'interfaces

Ce document présentera la concept d'un microprocesseur 32-Bits qui est capable d'effectuer des additions floating-point d'après la norme E754. La structure du microprocesseur sera d'une façon très flexible. Il ne pose pas trop de problèmes d'ajouter autres fonctionnalités et d'élargir l'ensemble des commandes.

La figure suivante montre un système utilisant le microprocesseur :



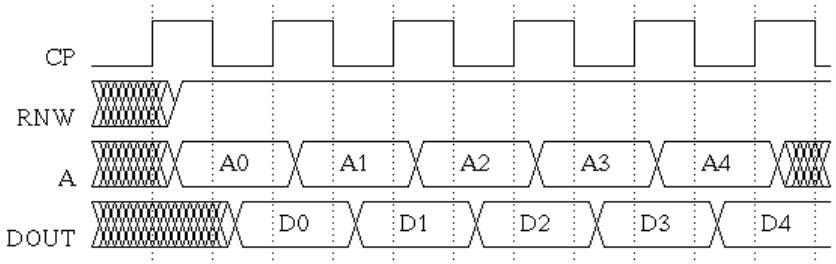
Toutes les entreés ou fils se composent de 16 canaux, sauf le « Read/Write », l'« Horloge » et le « RESET ». Ils n'ont qu'un canal.

Le microprocesseur peut recevoir des données de la RAM des données ou de la RAM/ROM du code. Il traite les données et sauvegarde le résultat dans la RAM des données.

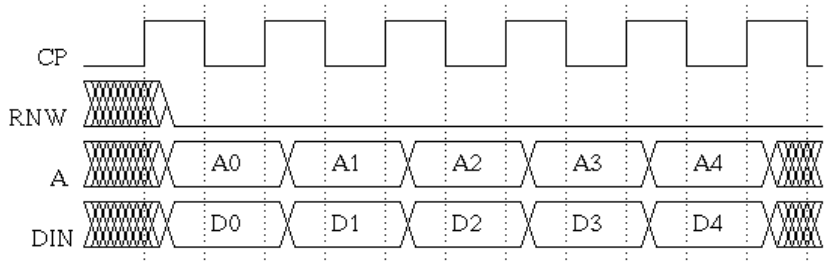
Le tableau suivant montre les entrées et sorties du microprocesseur qui a 84 pins utilisés par l'utilisateur. Le reste des pins sert au test est donc ne sont pas intéressants pour l'utilisateur.

Nombre de pins	utilisation
1 (entrée)	alimentation
1 (entrée)	"0"
1 (entrée)	RESET (active au niveau « 0 », asynchrone)
16 (entrée)	Code-In Bus
16 (entrée)	Data-In Bus
1 (sortie)	Read/Write Data-RAM (« 1 »: Read, « 0 »: WRITE)
16 (sortie)	Code-Adresse Bus
16 (sortie)	Data-Adresse Bus
16 (sortie)	Data-Out Bus

La communication entre la RAM/ROM et le microprocesseur sera organisée comme proposée dans les TPs du cours *VHDL*. Le tableau suivant montre l'horloge (« CP ») , le « Read/Write » (« RNW »), l'adresse (« A ») et les données (« DIN ») d'un cycle de lecture.



Le tableau suivant montre l'horloge (« CP ») , le « Read/Write » (« RNW »), l'adresse (« A ») et les données (« DIN ») d'un cycle d'écriture.

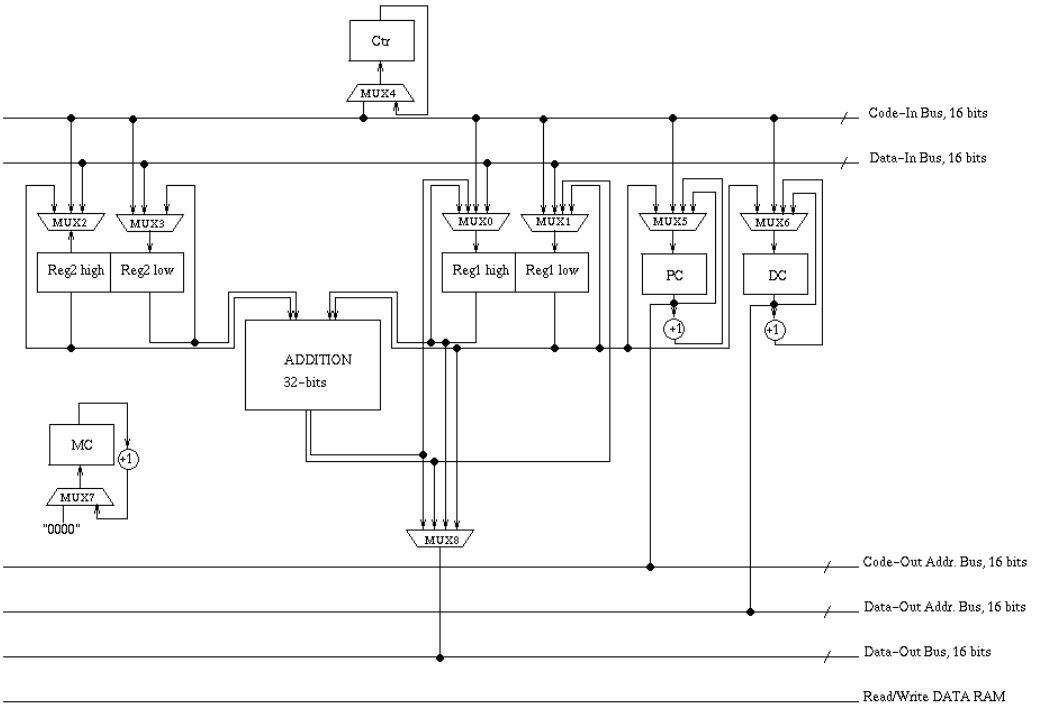


2. Spécification fonctionnelle

La suite présentera premièrement la concept du cœur du microprocesseur, qui traite l'additionneur comme boîte noire, et deuxièmement l'additionneur soi-même.

2.1. Le concept du cœur du microprocesseur

Le cœur du microprocesseur se compose de plusieurs multiplexeurs, registres, compteurs et d'une mémoire non-volatile qui contient des micro-commandes qui déterminent les états des multiplexeurs dépendants de la command d'assembleur actuelle. La figure suivante montre la structure.



Cette structure permet à faire quelques observations:

- Le microprocesseur utilise plusieurs bus extérieurs de 16 bits chacun:
  - *Code-Adresse-Bus*: Adresse actuelle de la Code-RAM ou Code-ROM
  - *Code-In-Bus*: Commande actuelle dans la Code-RAM ou CODE-ROM à l'adresse du *Code-Adresse-Bus*.
  - *Data-Adresse-Bus*: Adresse actuelle de la Data-RAM
  - *Data-In-Bus*: Donnée actuelle à lire de la Data-RAM à l'adresse *du Data-Adresse-Bus.*
  - *Data-Out-Bus:* Donnée actuelle à écrire à la Data-RAM.
- Le processeur travaille à l'extérieure avec 16 bits et à l'intérieure avec 16 ou 32 bits. Ce fait limite les nombre des pins du puce.
- Le processeur a 6 registres internes:
  - Le Program-Counter *PC* qui mémorise l'adresse de la commande actuelle dans la Code-RAM.
  - Le Data-Counter *DC* qui mémorise l'adresse des données actuelle dans la DATA-RAM.
  - Le registre *Reg1* qui mémorise une partie des données à l'entrée de l'additionneur et le résultat après une addition.
  - Le registre *Reg2* qui mémorise une partie des données à l'entrée de l'additionneur.
  - Le registres *RegCtr (16 bits)* qui mémorise la commande actuelle et le registre *RegMC (4 bits)* qui est le compteur des micro-commandes. La micro-commande à exécuter prochainement se trouve toujours dans la mémoire non-volatile à la position *RegCtr + RegMC*. Chaque micro-commande consiste d'un certain nombre des bits qui défissent l'état de chaque multiplexeur.

Ce microprocesseur sera au plupart utilisé dans petits systèmes où le programme d'assembleur ne change pas, contrairement aux données. C'est pourquoi, la mémoire du Code a été séparée de la mémoire des données. Pendant le développement d'un circuit qui utilise le microprocesseur, il est bien possible à brancher une RAM volatile sur le puce. Au système final, on substitue la RAM par une ROM non-volatile qui contient le programme.

Tous les registres sont initialisés à "0" après une RESET (asynchrone, active à « 0 »). Donc les premières commandes, respectivement les premières données doivent se trouver à l'adresse "0" de la Code-RAM ou de la DATA-RAM respectivement.

Pour chaque code d'assembleur il faut écrire une micro-commande consistante de 17 bits pour définir les états de tous multiplexeurs:

	MUX0	MUX1	MUX2	MUX3	MUX4	MUX5
bits	0..1	2..3	4..5	6..7	9	10..11

	MUX6	MUX7	MUX8	MUX9
bits	12..13	9	14..16	8

Les tableaux suivantes montrent les dépendances entre les bits de micro-commandes et la sortie des multiplexeurs:

état de la micro-commande	sortie MUX0 (bits 0..1)	sortie MUX1 (bits 2..3)	sortie MUX2 (bits 4..5)
00	Code-In Bus	Code-In Bus	Code-In Bus
01	Data-In Bus	Data-In Bus	Data-In Bus
10	sortie additionneur high	sortie additionneur low	Reg2 high
11	Reg1 high	Reg1 low	Reg2 high

état de la micro-commande	sortie MUX3 (bits 6..7)	sortie MUX4 (bit 9)	sortie MUX5 (bits 10..11)
00	Code-In Bus	Code-In Bus	Code Bus
01	Data-In Bus	RegCtr	Reg1 low
10	Reg2 low	--	RegPC + 1
11	Reg2 low	--	RegPC

état de la micro-commande	sortie MUX6 (bits 12..13)	sortie MUX7 (bit 9)	sortie MUX9 (bit 8)
00	Code Bus	"0000"	Reg1/2 connectés à l'additionneur
01	Reg1 low	RegMC + 1	Reg1/2 séparés de l'additionneur
10	RegDC + 1	--	--
11	RegDC	--	--

état de la micro-commande	sortie MUX8 (bits 14..16)
000	sortie de l'additionneur high
001	sortie de l'additionneur low
010	Reg1 high
011	Reg1 low
100 (et reste)	DIN_DATA_RAM

Ces tableaux montrent qu'on peut réaliser une multitude des commandes différentes. Similaires aux microprocesseurs de Motorola et Intel, différentes méthodes d'adressage et beaucoup plus est faisable. Dans le cadre du projet, seulement deux micro-commandes ont été écrites; une pour charger la prochaine commande de la Code-RAM (code "0x00") et une pour charger et additionner deux chiffres de la Data-RAM. Le résultat sera sauvegardé dans le registre *Reg1* et puis envoyé à la RAM des donées.

Le tableau suivant montre la suite des micro-commandes pour charger deux chiffres de la Data-RAM et pour les additionner. A l'instant nous présumons que l'additionneur n'a pas besoin de wait-states et peut donc livrer le résultat dans un cycle:

N° commande	micro-commande (bits 0..16)	qu'est-ce qui se passe
1	`11111101111111111`	DC -> Adresse Data-RAM, RegPC += 1
2	`11110111111011111`	Data-RAM -> Reg2 high, RegDC += 1
3	`11111111111111111`	DC -> Adresse Data-RAM
4	`11110111101111111`	Data-RAM -> Reg2 low, RegDC += 1
5	`11111111111111111`	DC -> Adresse Data-RAM
6	`11110111111111101`	Data-RAM -> Reg1 high, DC += 1
7	`11111111111111111`	DC -> Adresse Data-RAM
8	`11110111111110111`	Data-RAM -> Reg1 low, DC += 1
9	`11111111011111111`	Reg1, Reg2 -> additionneur
10	`11111111011111111`	résultat addition -> Reg1
11	`11111111011111111`	MC = 0, réception de la prochaine commande

D'après ce tableau, l'exécution de toutes les micro-commandes dure 11 cycles. La RAM et la ROM consistent de processus qui sont activés au front d'horloge descendant au niveau "0"; ils ont la même horloge que le microprocesseur, mais ils n'ont pas de wait-states.

2.2. La concept de l'additionneur

L'addition de deux chiffres d'après la norme E754 se déroule de principe comme expliqué dans la suite:

Il y a le cas *spécial*:

- Si un des deux chiffres est non défini ("NaN") ou infini, ce chiffre est déjà le résultat; sauf si un des deux chiffres est "+∞" et l'autre "-∞". Dans ce cas, le résultat est "0".

Et il y a le cas *normal*:

- On ajoute les chiffre à gauche aux deux chiffres ("00." ou "01").
- On garde le chiffre avec le plus grand exposant et décale la mantisse de l'autre chiffre (*Shift Right*) pour égaliser l'exposant sans modifier la valeur. On accepte les pertes à cause de la précision limitée.
- On additionne/soustraie les mantisses.
- On cherche le *'1'* du plus fort poids.
- On décale la mantisse du résultat (*Shift Right/Left*) pour n'obtenir qu'un '1' au premier nombre à gauche ("*01.xxxxxxxx…*"); parallèlement on modifie la valeur de l'exposant pour garder la même valeur du résultat.
- Si l'exposant est déjà minimal ("*0-126 = -126*") avant pouvoir finir la décalage de la mantisse du résultat, on garde l'exposant minimal ("*0-126 = -126*") et la mantisse correspondante. Le résultat représente le cas d'un chiffre *dénormalisé* après la norme E754.

La figure suivante montre la démarche en détail.

La suite présente un exemple de l'addition de deux chiffres. Points entre les différentes partie d'un chiffre essayent à clarifier le processus ("s.*eeeeeeee*.*ffffffffffffffffffffff*"):

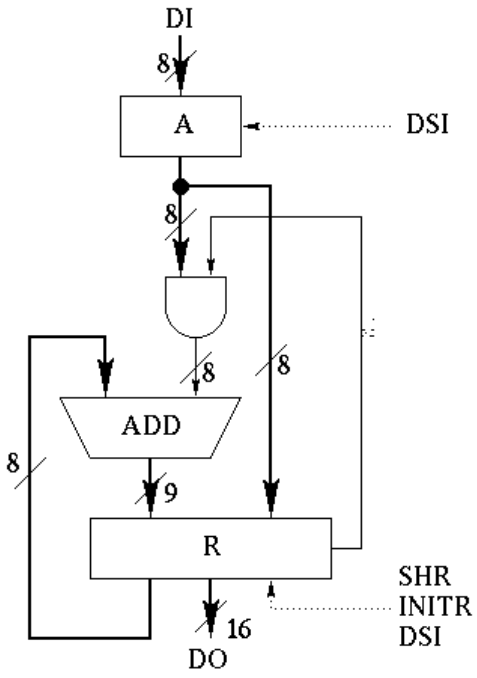
Chiffres	Action
C1 = 1.00000001.00111010000000000000 C2 = 0.00000010.00011000000000000000	départ
C1 = 1.00000001.(01,)00111010000000000000 C2 = 0.00000010.(01,)00011000000000000000	ajouter les bits à gauche
C1' = 1.00000010.(00,)10011101000000000000 C2 = 0.00000010.(01,)00011000000000000000	égaliser des exposantes (décalage du chiffre de plus petit exposant)
C3 = C1' + C2 = (S1).(E1).(M2-M1) = 0.00000010.(00,)01111011000000000000	effectuer l'addition, ou plutôt la soustraction
C4 = 0.00000000.(01,)11101100000000000000	décaler à gauche pour avoir un '1' à gauche: (00,)01111011000000000000<<2 = (01,)11101100000000000000, 00000010 – 2 = 00000000
résultat : 0.00000001.11101100000000000000	L'exposant est « 1 » (et pas « 0 »), parce que il y a un « 1 » à gauche.

3. Estimation des caractéristiques

La suite essayera d'estimer la vitesse, la surface nécessaire et la consommation du cœur du microprocesseur et de l'additionneur. L'estimation s'appuiera sur les résultats obtenus pendant les TPs du cours *Synthèse*.

3.1. Le cœur du microprocesseur

Le cœur du microprocesseur consiste au plupart de dix cellules de 16 bits (sauf une qui a quatre bits). Ces cellules semblent toutes l'une l'autre et avant tout ils semble - un peu - la cellule du figure 3 du TP 3 du cours *Synthèse*:



Cette cellule a des caractéristiques suivantes:

Surface	Consommation (ports + metal)	Vitesse
Area: $78.68 \cdot 10^3$ μ^2	total: 2.7285 mW	25 MHz

Pour obtenir une première estimation pour le cœur du microprocesseur, nous multiplions la surface et la consommation avec la nombre des cellules (donc avec "10") et laissons la vitesse invariante:

Surface	Consommation (ports + metal)	Vitesse
Area: $786.8 \cdot 10^3$ μ^2	total: 27.285 mW	25 MHz

3.2. L'additionneur

L'additionneur est beaucoup plus complexe et donc beaucoup plus difficile a estimer que le cœur du microprocesseur. En comparaison avec le cœur, on peut un peu près estimer que la surface et la consommation puissent être deux à trois fois plus grandes que les valeurs correspondantes du 2.1. Nous allons choisir le facteur "2,5". Nous venons choisir une structure PIPE-LINE pour pouvoir obtenir une vitesse acceptable, mais la complexité limitera malgré tout la vitesse. Nous estimons que la vitesse sera diminuée par le même facteur "2.5". Le tableau suivant montre les valeurs estimées:

Surface	Consommation (ports + metal)	Vitesse
Area: $1922 \cdot 10^3$ μ^2	total: 68.2125 mW	10 MHz

3.3. Le système

Le valeur caractéristiques se calculent par simple addition (Surface, Consommation), ou plutôt par choix de la pire valeur (vitesse):

Surface	Consommation (ports + metal)	Vitesse
Area: $2690.8 \cdot 10^3$ μ^2	total: 95.4975 mW	10 MHz

Partie II

Conception du Microprocesseur 32-Bits

[redacted]

4. Les commandes de la Synthèse

Le code du microprocesseur consiste de trois fichiers :

- *UNIT_PROC.VHD*
  Ce fichier contient le code du microprocesseur *sans additionneur*.
- *UNIT_ADDI.VHD*
  Ce fichier contient le code de l'additionneur.
- *TOTAL.VHD*
  Ce fichier combine les deux parties *UNIT_PROC.VHD* et *UNIT_ADDI.VHD.*

Grâce à cette répartition on peut optimiser les deux parties *additionneur* et *cœur du microprocesseur* séparément et donc gagner du temps chez la synthèse.

Les commandes en détail sont :

- *CMPSYN UNIT_PROC*
  Créer une netlist de l'entité *PROCESSEUR* (*PROCESSEUR_P*).
- *CMPSYN UNIT_ADDI*
  Créer une netlist de l'entité *ADDI32* (*ADDI 32_P*).
- *CMPSYN –c UNIT_PROC –c UNIT_ADDI TOTAL*
  Créer une netlist de l'entité *TOT_PROCESSEUR* (*TOT_PROCESSEUR_P*) sans recréer les netlists des entités *ADDI32, UNIT_ADDI TOTAL*.
- *CMPOPT -periode 100 -horloge CP -de 3 -ds 3 -ce 1.3 -cs 0.7 -fe 2 -rampe 3 -portes 15000 processeur*
  Créer le fichier d'optimisation. On sélectionne une « période » de 100 ns (→ fréquence 10 MHz) , l' « horloge » du système s'appelle « CP » ; on choisit comme « délai » d'entrée et sortie *3 ns,* comme charge d'entrée *1.3 pF* et charge de sortie *0.7 pF*. Les valeur des charges ont été choisis après une première optimisation du système qui avait calculé une charge maximale d'entrée /sortie de *1.3 pF/0.7 pF*. On choisit une force d'entrée de *2 ns/pF* et une rampe de *3 ns* qui sont des valeurs de standard et *15000 portes* en total, car notre système consiste d'environ *7000 portes* et sera intégré dans un système plus complex.
- *CMPOPT -periode 100 -horloge CP -de 3 -ds 3 -ce 1.3 -cs 0.7 -fe 2 -rampe 3 -portes 15000 addi 32*
  Créer le fichier d'optimisation. On choisit les mêmes valeurs d'avant.
- *CMPOPT processeur*
  Lancer de l'optimisation *PROCESSEUR_P*. Les contraintes visées sont remplis.

- *CMPOPT addi 32*
  Lancer de l'optimisation de *ADDI 32_P*. Les contraintes visées sont remplis.
- *CMPRPP –frequence 10 –portes 15000 TOT_PROCESSEUR*
  Effectuer l'analyse syntaxique. On trouve que les contraintes (p. ex. délai de rampe) sont remplis.
- *CMPN2V TOT_PROCESSEUR_P*
  On laisse créer des fichiers *VHDL* à partir de la netlist créée (*ADDI32_P.VHD, PROCESSEUR_P.VHD et TOT_PROCESSEUR_P.VHD*). Ils permettent à simuler la logique crée et donc à la tester.
- *CMPTST –insere –longueur 4 TOT_PROCESSEUR_P*
  Connecter le chaînes de scan utilisant le plan de masse crée avant et générer les vecteurs de test. Nous prenons une longueur des chaînes de scan qui est egal ou inférieur à 4.
- *CMPFLR –frequence 10 –horloge CP –delai 0.5 –derive 100 –forme a –aspect 1 -facteur 0.5 TOT_PROCESSEUR_P*
  Créer d'un plan de masse. On choisit comme *délai 0.5* pour limiter le temps de propagation de chaque buffer de l'arbre à *0.5 ns*, comme *derive 100* pour limiter la dérive d'horloge à 100 ps et comme *forme* l'option *facteur du forme* et définit le *facteur du forme* à l'aide de l'option *aspect* à *1*. On sélectionne un facteur de routage de *0.5*, car le circuit consiste de beaucoup des lignes et cette valeur réserve 50 % de la surface pour la routage.
- *CMPPER –place 5 –recuit –arbre TOT_PROCESSEUR_P*
  Placer des cellules, optimiser de placement par recuit stimulé et générer l'arbre d'amplification d'horloge. L'option *–place 5* sert à une optimisation de placement.
- *CMPTST –connecte –vecteurs TOT_PROCESSEUR_P*
  Connecter le chaînes de scan utilisant le plan de masse crée avant et générer les vecteurs de test.
- *RM TOT_PROCESSEUR_P_TA.FLR*
  Détruire le floor-plan pour pouvoir le créer de nouveau. On le fait après avoir connecté les vecteurs de scan pour obtenir une structure optimisé quant au processeurs utilisé par le client et pas quant au materiél du test.
- *CMPFLR –frequence 10 –horloge CP –delai 0.5 –derive 100 –forme a –aspect 1 -facteur 0.5 TOT_PROCESSEUR_P*
  Créer de nouveau le plan de masse.
- *CMPPER –place 5 –recuit –arbre –route 5 TOT_PROCESSEUR_P*
  Placer des cellules, optimiser de placement par recuit stimulé, générer l'arbre d'amplification d'horloge et **router le circuit avec optimisation de routage.**
  On trouve que le facteur de routage (*environ 1.5*) est plus grand que choisi avant (*0.5*). On a donc sousestimé la taille des interconnections.

- *CMPPWR –portes 15000 TOT_PROCESSEUR_P TOT*
  La simulation montre que le circuit consomme environ *11.3637 mW*. Cette valeur est beaucoup plus basse que la valeur estimée avant (*95.49 mW*) :
  - L'additionneur change ses états rarement. A l'instant il y a une nouvelle addition chaque 16 cycles. Notre estimation s'appyierait sur un changement des données dans chaque cycle.
  – La fréquence est de 10 MHz dans notre cas et pas à 25 MHz comme à la celle de TP3. Les valeurs estimées était très pessimistes.
- *CMPRPP –frequence 10 –portes 15000 TOT_PROCESSEUR_P*
  Effectuer l'analyse syntaxique. On obtient une surface de $1823 \cdot 10^3 \mu^2$. Notre estimation de $2690.8 \cdot 10^3 \ \mu^2$ n'était pas mal.
- *compass -checkoutpackages Cell_Based_Place_and_Route\(ClockTreeCompiler\) chipcompiler '[cc]TOT_PROCESSEUR_p_ta'*
  Regarder du résultat.

Partie III

Documentation technique du Microprocesseur 32-Bits

[redacted]

5. Présentation du microprocesseur 10 MHz

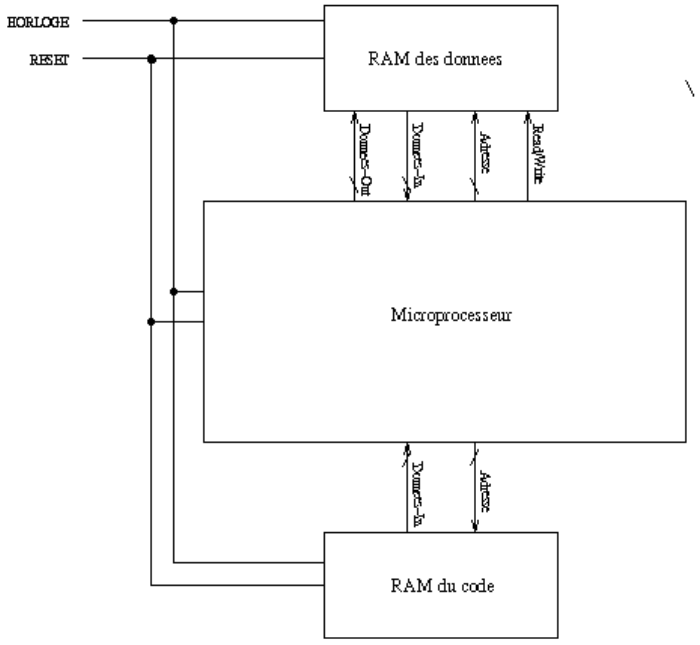
La suite expliquera comment intégrer le microprocesseur dans un système à l'aide d'un exemple simple. Le prochain chapitre montrera les valeurs caractéristiques comme la fréquence du fonctionnement, la consommation, etc.

5.1. Les systèmes cible du microprocesseur

Le microprocesseur vise petits systèmes qui consistent d'une ROM ou RAM du code et d'une RAM pour sauvegarder les données et les résultats du calcul effectué avec le microprocesseur. C'est pourquoi, on utilise deux mémoires différentes. L'une pour le code et les données non-volatile (RO M/RAM) et l'autre pour les données du calcul.

5.2. Un système utilisant le microprocesseur

La figure suivante montre un système simple utilisant le microprocesseur :



Cette structure permet à faire quelques observations:

- Le microprocesseur utilise plusieurs bus extérieurs de 16 bits chacun:
  - *Code-Adresse-Bus*: Adresse actuelle de la Code-RAM ou Code-ROM
  - *Code-In-Bus*: Commande actuelle dans la Code-RAM ou CODE-ROM à l'adresse du *Code-Adresse-Bus*.
  - *Data-Adresse-Bus*: Adresse actuelle de la Data-RAM
  - *Data-In-Bus*: Donnée actuelle à lire de la Data-RAM à l'adresse *du Data-Adresse-Bus.*
  - *Data-Out-Bus:* Donnée actuelle à écrire à la Data-RAM.
- Le système a une horloge unique (Fréquence 10 MHz).
- Le système a un RESET unique.

Au début le système autour du microprocesseur doit effectuer un RESET. Puis tous les registres interne sont initialisés à *0*. La première commande va donc être lue de l'adresse « 0 » de la mémoire du code.

Les registres après l'initialisation :

	Reg1	**Reg2**	**PC (Code Adresse)**	**DC (Data Adresse)**
Valeur d'initialisation	0 (32 bits)	0 (32 bits)	0 (16 bits)	0 (16 bits)

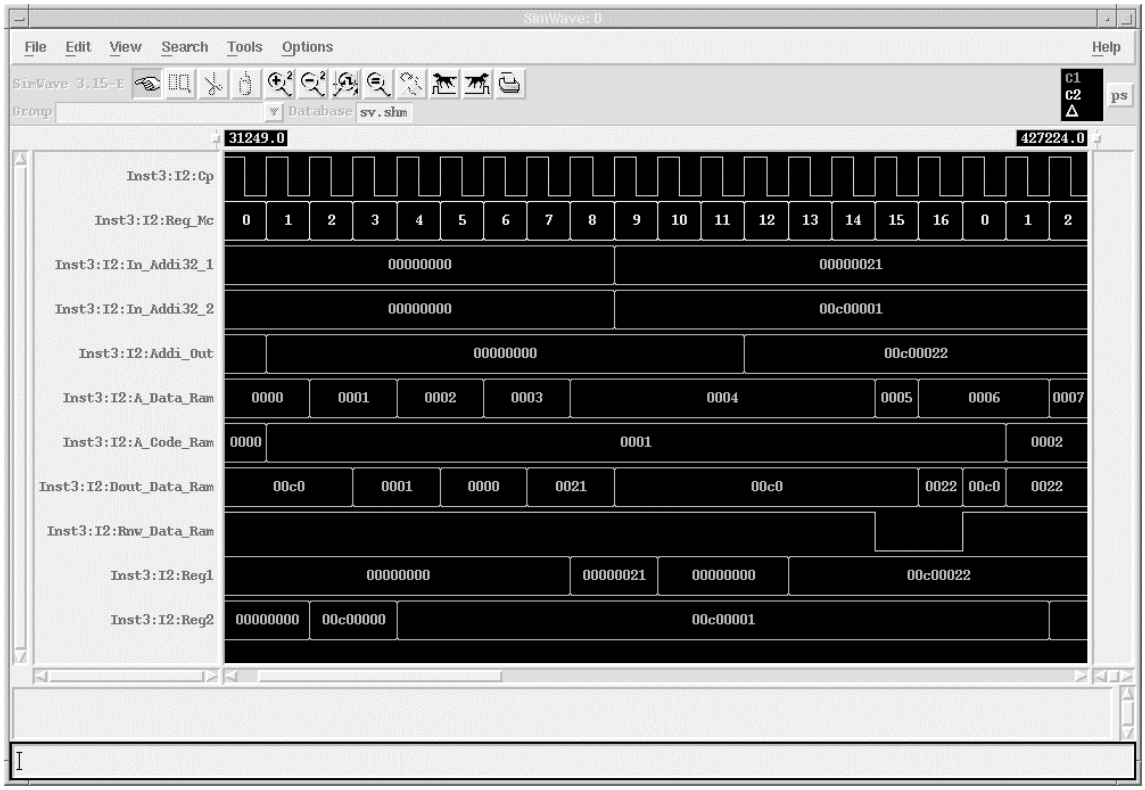
(A l'instant) le microprocesseur connaît deux commandes:

Commande (16-bits-code)	**Explication**
0000000000000000	No operation (NOP)
0000000000000001	Cette commande charge la valeur 32 bits de l'adresse actuelle de (DC/DC+1) de la RAM des données au registre Reg1 (suite : high/low) et la prochaine valeur 32 bits l'adresse actuelle de (DC+2/DC+3) de la RAM des données au registre Reg2 (suite : high/low). Puis elle effectue une addition 32 bits d'après la norme E754 (voir chapitre 2), sauvegarde le résultat dans le registre Reg1 et l'écrit dans la RA M à l'adresse (DC+4/DC+5). Le résultat est 16 cycles plus tard dans la RAM.

La norme E754 des chiffre sera expliquée dans le chapitre 2.

Les figures suivantes montrent les signaux entre le microprocesseurs et les mémoires pour deux exemples: Une addition et une soustraction.

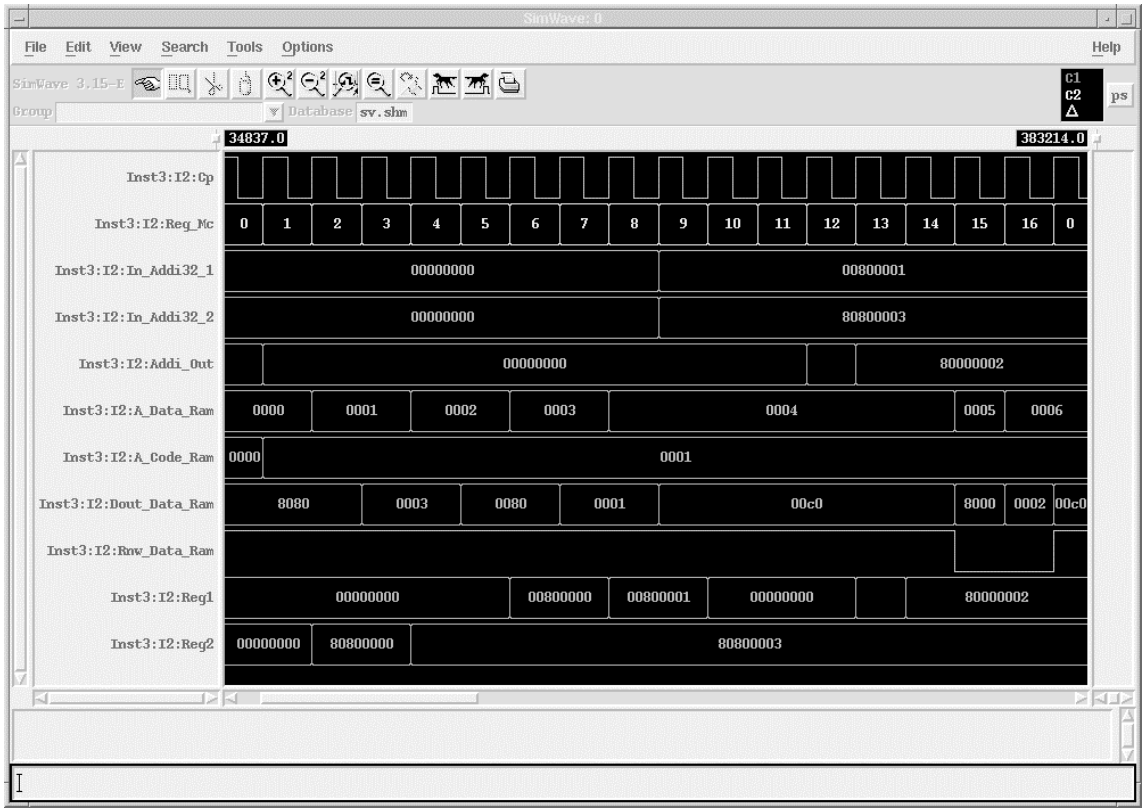
Exemple 1: Addition des chiffres "$1.10000000000000000000001 \cdot \mathbf{2}^{\mathbf{1-127}}$" et "$0.00000100000000000000001 \cdot \mathbf{2}^{\mathbf{-126}}$". D'après la norme E754 le premier chiffre est codé en "0.00000001.10000000000000000000001" (hexadécimal = "00C00001") et le deuxième en "0.00000000.00000100000000000000001" (hexadécimal = "00000021").
Le résultat sera "0.00000001.10000100000000000000010" (hexadécimal = "00000021"). La figure suivante montre les signaux aux entrées et sorties et l'état des registres *Reg1* et *Reg2* du microprocesseur.



Le tableau suivant explique les noms des sorties, entrées et registres dans la figure au-dessus.

Nom	Explication
CP	*CP* est l'horloge du système
MC	*MC* compte les cycles déjà passés depuis le lancement de la commande actuelle
IN_ADDI32_1	*IN_ADDI32_1* est le premier chiffre à additionner
IN_ADDI32_2	*IN_ADDI32_2* est le deuxième chiffre à additionner
ADDI_OUT	*ADDI_OUT* contient le résultat de l'addition
A_DATA_RAM	Adresse actuelle de la RAM des données
A_CODE_RAM	Adresse actuelle de la RAM du code
DOUT_DATA_RAM	Données à sauvegarder dans la RAM des données si *RNW_DATA_RAM* = niveau bas
RNW_DATA_RAM	voir *DOUT_DATA_RAM*
Reg1	Registre 1. Le premier chiffre à additionner est plus tard le résultat de l'addition est sauvegardé dans ce registre.
Reg2	Le deuxième chiffre à additionner est sauvgardé dans ce registre.

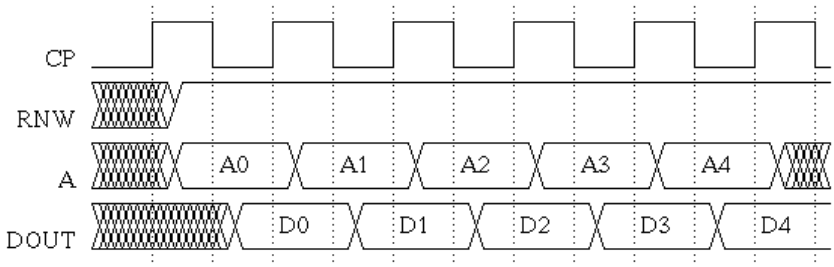
Exemple 2: Addition des chiffres "$1.00000000000000000000001 \cdot \mathbf{2}^{\mathbf{1-127}}$" et "$-1.00000000000000000000011 \cdot \mathbf{2}^{\mathbf{1-127}}$". D'après la norme E754 le premier chiffre est codé en "0.00000001.10000000000000000000001" (hexadécimal = "00800001") et le deuxième en "1.00000001.00000000000000000000011" (hexadécimal = "80800003"). La figure suivante montre les signaux aux entrées et sorties et l'état des registres *Reg1* et *Reg2* du microprocesseur.



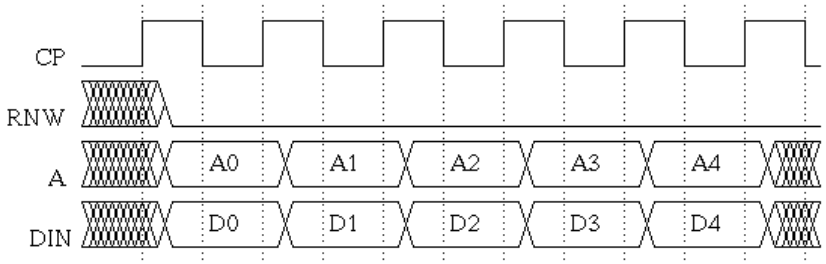
5.3. Les mémoires externes

La RAM et la ROM consistent de processus qui sont activés au front d'horloge descendant au niveau "0"; ils ont la même horloge que le microprocesseur, mais ils n'ont pas de wait-states.

Le tableau suivant montre l'horloge (« CP ») , le « Read/Write » (« RNW »), l'adresse (« A ») et les données (« DIN ») d'un cycle de lecture de la RAM.



Le tableau suivant montre l’horloge (« CP ») , le « Read/Write » (« RNW »), l’adresse (« A ») et les données (« DIN ») d’un cycle d’écriture de la RAM.



La ROM ne connaît que la lecture. Donc, le signal *RNW* n'existe pas. Mais le processus de la lecture fonctionne dans la même façon comme présenté au-dessus.

6. Les contraintes et caractéristiques

6.1. Les entrées et sorties du microprocesseur

Le tableau suivant montre les pins du microprocesseur:

Noms des pins	Explication
CP	*CP* est l'horloge du système (Fréquence: 10 MHz)
RESET	RESET du microprocesseur
DIN_CODE_RAM[0..15]	Les données de la CODE-RAM
A_CODE_RAM[0..15]	Adresse actuelle de la CODE-RAM
DIN_DATA_RAM[0..15]	Les données de la DATA-RAM
DOUT_DATA_RAM[0..15]	Les données visantes la DATA-RAM
A_DATA_RAM[0..15]	Adresse actuelle de la DATA-RAM
RNW_DATA_RAM	"niveau bas" = écriture DATA-RAM "niveau haut" = lecture DATA-RAM
SCAN TG	Pins de test, ne pas à utiliser
SCAN_IN[1..87]	Pins de test, ne pas à utiliser
SCAN_OUT[1..87]	Pins de test, ne pas à utiliser

6.2. Quelques caractéristiques du microprocesseur

Les 175 pins de test garantissent une *FAULT COVERAGE* de *97,80 %*. La probabilité de mal-fonctionnement du microprocesseur est donc *2,2 %*.

La surface totale du microprocesseur est *77,26 mil-yards x 112,47 mil-yards*. Il consiste de 33.903 transistors. La consommation moyenne est environ 11.4 mW. Le microprocesseur consiste de 6700 portes et il est optimisé pour un système de 15000 portes en total. Le chemin critique à l'intérieure du microprocesseur est de 91.97 ns.

Le tableau suivant montre les capacités importants:

Nom	Valeur
Le capacité maximal se trouve à l'entrée *CP*.	C_{CP} = 21.79 pF
Le capacité maximal aux sorties.	FANOUT = 346.0 pF
Capacité de l'entrée RESET	C_{RESET} = 1.42 pF
Capacité de l'entree DIN_CODE_RAM[0..15]	$C_{DIN\_CODE\_RAM}$ < 1.50 pF par pin
Capacité de l'entree DIN_DATA_RAM[0..15]	$C_{DIN\_DATA\_RAM}$ < 1.43 pF par pin
Capacité de la sortie RNW_DATA_RAM	$C_{RNW\_DATA\_RAM}$ = 0.83 pF
Capacité de la sortie A_DATA_RAM[0..15]	$C_{A\_DATA\_RAM}$ < 1.05 pF par pin
Capacité de la sortie DOUT_DATA_RAM[0..15]	$C_{DOUT\_DATA\_RAM}$ < 0.84 pF par pin
Capacité de la sortie A_CODE_RAM[0..15]	$C_{A\_CODE\_RAM}$ < 0.97 pF par pin

Le circuit est optimise pour des *delais maximales* d'entrées et sorties de *3ns* et d'une *force d'entrée de 2ns/pF*.