

VHDL

Environnement

Nous utiliserons le simulateur **Modelsim** de la suite logicielle **Intel (Altera) Quartus** qui regroupe sous une même interface graphique un ensemble intégrant les outils de compilation, élaboration et simulation de code HDL, mais également Verilog et SystèmeC.

Arborescence de travail

<some_path>/R3K

Fichiers de base

copy files from http://m2siame.univ-tlse3.fr/teaching/francois/UE-M1-VHDL/R3K_Base/

Lancement de l'outil

modelsim

Une fois l'application lancée, il vous faut configurer l'environnement de travail :

File ⇒ New Project :

Design Directory <some path>/R3K

puis Add Existing Files → select all .vhd files to reference within your project

Quelques fonctions de conversions :

- *conv_std_logic_vector (integer, size)* ⇒ *std_logic_vector*
- *conv_integer (std_logic_vector)* ⇒ *integer*
- *to_stdlogicvector (bit_vector)* ⇒ *std_logic_vector*
- *F <= std_logic(signal A or signalB)* ⇒ *std_logic*

use ieee.std_logic_arith.all
use ieee.std_logic_unsigned.all
use ieee.std_logic_1164.all
conversion en std_logic

Design Flow

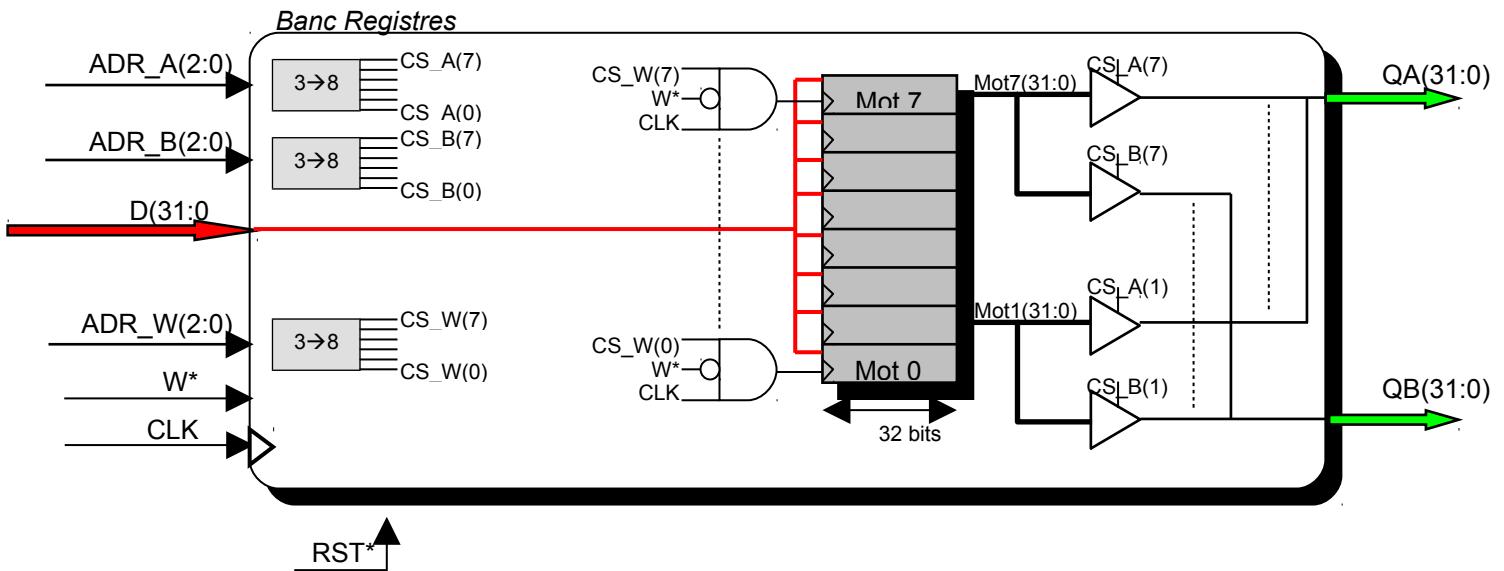
- **Compilation** des entités, architectures, [configurations] et d'une architecture de test.
- **Elaboration** de l'architecture de test précédemment compilée générant un *snapshot*.
- **Simulation** du *snapshot*.

PAGE BLANCHE

Travaux Pratiques

Tp1-1 : Banc Registres double port lecture pour µP RISC

On se propose de réaliser un banc de 8 registres de 32 bits double accès en lecture avec pour particularité d'avoir le registre 0 dont la lecture renvoie toujours la valeur **0**.



Les signaux possédant le suffixe ****** signifie qu'ils sont actifs au niveau bas.

Le signal **RST*** de même que les opérations de lectures au travers des deux bus d'adresses **ADR_A** et **ADR_B** sont **asynchrones**. Les opérations d'écritures elles sont **synchrones** sur front **montant** de l'horloge et conditionnés par la présence du signal **W*** actif.

L'activation du signal **RST*** initialise tous les registres du banc à la valeur **0**.

Vous commencerez par compléter la partie **entity** puis **architecture** du fichier **registres.0.vhd**, puis **compilation**.

Notes

Déclaration d'un type tableau de 8 registres de 32 bits :

```
type FILE_REGS is array (integer range 0 to (2**3)-1) of std_logic_vector (31 downto 0);
```

Cette définition de type se trouvera dans la partie déclaration de l'architecture suivie de la définition matérielle associée :
`signal REGS : FILE_REGS;`

Tp1-2 : Simulation Banc Registres

Vous avez donc à présent un banc de registres que l'on s'apprête à intégrer dans un environnement de test et pour ce vous commencerez par éditer le fichier de base **test_registres.0.vhd**.

- ### Notes
- Etant donné que l'architecture de test ne possède que très peu de composants et d'instances de composants, il n'est pas nécessaire d'ajouter une unité de configuration, on se contentera seulement de spécifier l'architecture à utiliser pour toutes les instances d'un même composant.
Une unité de configuration vous permet de spécifier pour chaque instance de composant l'architecture à utiliser.

Vous devez à présent avoir tous vos fichiers VHDL compilés; Il vous faut faire l'élaboration qui consiste à créer la vue destinée au simulateur. Cette vue tient compte d'une éventuelle unité de configuration ou d'une éventuelle spécification d'architecture d'instances de composants au sein de l'architecture de test. Pour ce vous sélectionnez **cds.lib** → **worklib** → **test_registres** → **behavior** et cliquez la commande **elaborate**. Un lien est alors créé dans **Snapshots**

Sélectionnez l'architecture dans le **Snapshots** puis cliquez la commande **simulate**.

Vous avez à présent le *Design Browser* de l'architecture de test qui apparaît. Ouvrez le *scope level ":"* qui est le point d'entrées des diverses unités de l'architecture (process, instances...), vous sélectionnez alors **regf0** qui est notre instance de banc de registre. Avec le bouton droit de la souris, sélectionnez l'option **Send to Waveform Window**.

Pour accéder à votre code source, depuis n'importe quelle fenêtre du simulateur **Window → New → Source Browser**. Après avoir lancé la simulation (bouton RUN), revenez à la fenêtre *SimVision* dans laquelle vous sélectionnez l'option *Zoom Out Full*.

Vous explorerez les possibilités offertes par le simulateur en matière d'exécution pas à pas, de création de sonde, de visualisation d'évolution des objets *Watch Objects*...et sa commande *Reinvoke* permettant de relancer une simulation.

Notes

- La dernière instruction du jeu de test provoque un *FAILURE* pour stopper la simulation. Vous pouvez aussi mettre un point d'arrêt en double cliquant sur celle-ci. Il existe également un *TIMOUT* qui peut, le cas échéant, lui aussi stopper le déroulement du test.

Tp1-3 : Banc Registres générique

Nous allons reprendre le code de notre précédent exercice en éliminant toutes les définitions de constantes en 'dur' en rendant génériques, avec une valeur par défaut, les paramètres suivants :

- DBUS_WIDTH Taille d'un mot du banc avec 32 bits par défaut.
- ABUS_WIDTH Largeur des bus d'adresses en lecture et écriture avec 5 bits par défaut - soit $2^{**}5$ mots -

Modifiez le fichier *registres.0.vhd* pour rendre le banc de registres générique. Vous modifierez le fichier *test_registres.0.vhd* pour tenir compte de la généricité du composant et procéderiez à une nouvelle simulation d'un banc de 8 registres de 32 bits à écriture sur front **montant**.

Notes

Lors du *mapping* du composant avec les signaux internes de l'unité de test, n'oubliez pas le *generic map* avec des paramètres de type constantes pour faire un *override* des paramètres par défaut.

Tp1-4 : Bypass D→Q

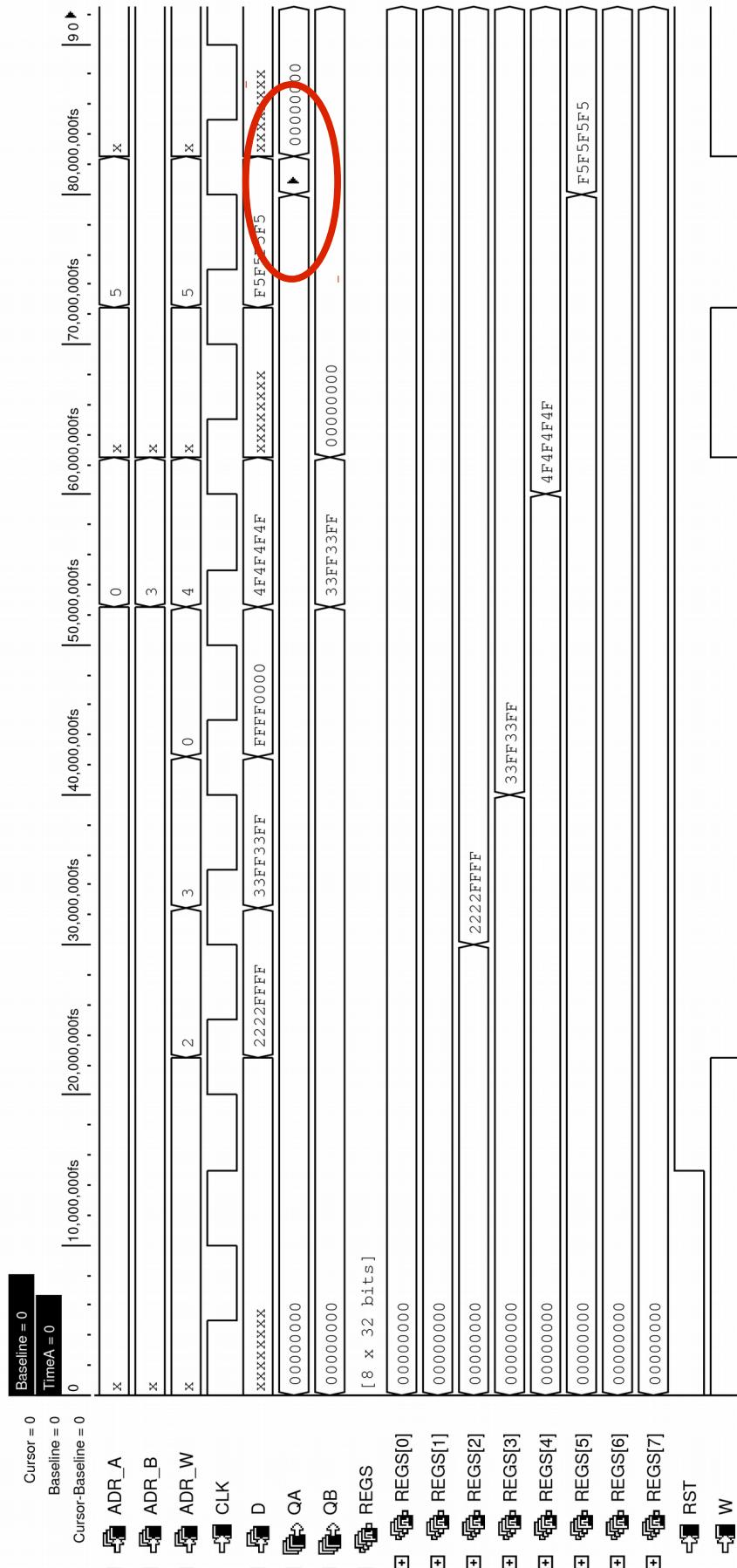
Il arrive qu'en cours d'exécution le processeur fasse simultanément une requête de lecture et d'écriture sur le même registre, ceci constituant un aléa de données pouvant être traité par l'unité d'envoi. Afin de simplifier la conception de cette dernière, on se propose d'implémenter cette fonctionnalité directement dans le banc de registres.

Si écriture et ADR_A ou ADR_B = ADR_W alors QA ou QB <= D

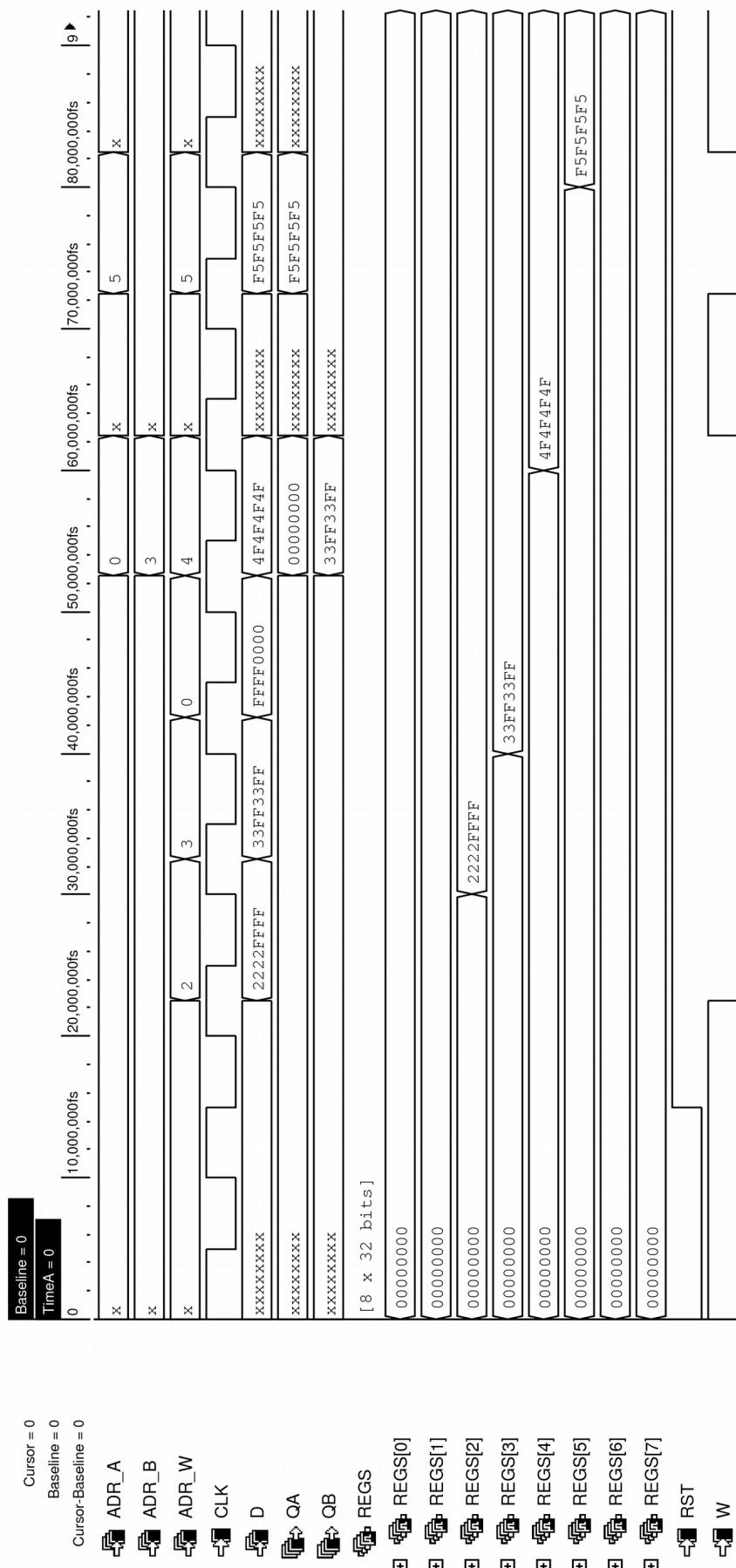
A partir du fichier *registres.1.vhd*, réécrivez dans le domaine combinatoire l'affectation des bus QA et QB selon les conditions décrites dans les deux process *P_ReadQA* et *P_ReadQB*.

Vous prendrez également soin de rendre la fonction **RESET synchrone**. Enfin, vous utiliserez le même fichier *test_registres.0.vhd* pour une simulation d'un banc de 8 registres de 32 bits à écriture sur front **montant**.

Banc Registers



Banc Registres avec bypass

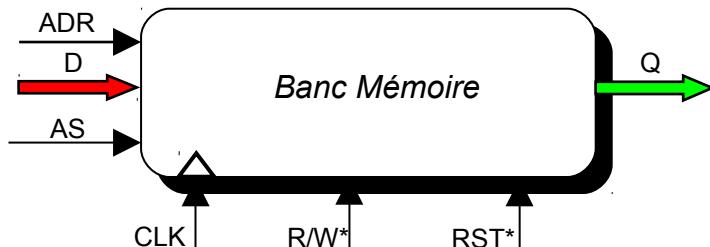


Tp2-1 : Banc Mémoire pour µP RISC

On se propose de réaliser un banc mémoire générique qui servira à alimenter le processeur en données et instructions :

Les paramètres génériques de cet élément sont les suivants :

- DBUS_WIDTH Taille d'un mot du banc avec 32 bits par défaut.
- MEM_SIZE Nombre de mots mémoire avec 8 par défaut.
- FILENAME Fichier d'initialisation avec <chaîne vide> par défaut.



Principe de fonctionnement

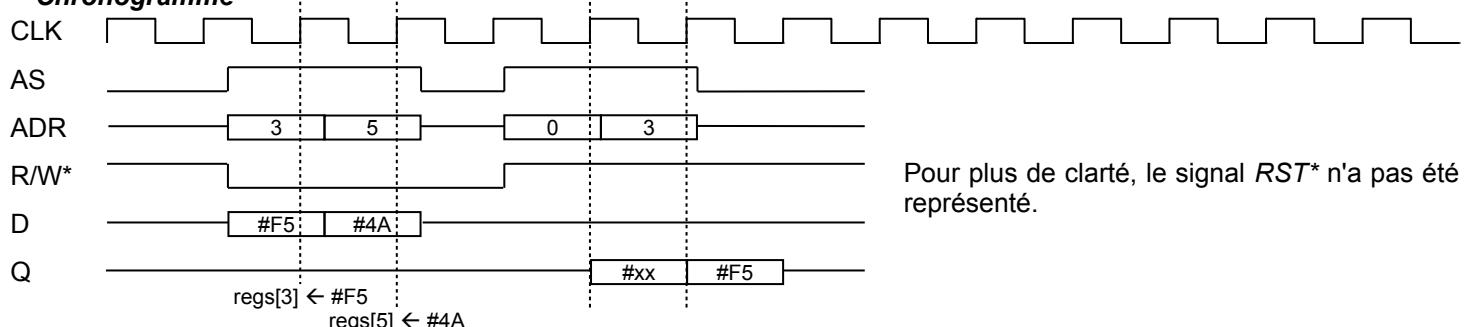
La prise en compte du signal **RST*** sera **synchrone**. Elle permet soit de charger le contenu d'un éventuel fichier passé en paramètre générique soit initialisera tous les éléments du banc mémoire à la valeur **0**.

La validation par le signal **AS** actif de l'adresse présente sur le bus **ADR** permet au prochain front montant d'horloge une lecture ou écriture suivant l'état 1 ou 0 du signal **R/W***.

Ces opérations de lectures / écritures feront référence à un emplacement mémoire défini par la valeur sur le bus **ADR** et ce dans une relation de type **1@ → 1mot mémoire**.

Dans le cas où il n'y a pas d'opération sur la mémoire, signal **AS** inactif, la **sortie Q est à l'état haute impédance noté Z**.

Chronogramme



Implémentation

Nous devons à présent introduire la notion de *package* dans lesquels se font les définitions de types, constantes, spécifications de fonctions, procédures...et la notion de *package body* dans lesquels se trouvent les corps de fonctions / procédures.

La largeur du bus d'adresse **ADR** étant fonction du nombre bits nécessaires à l'adressage des **MEM_SIZE** mots mémoire, vous aurez besoin d'une fonction **log2** que vous complèterez dans le fichier *cpu_package.0.vhd*. Vous compléterez le fichier *memory.0.vhd* contenant la définition de l'entité et son architecture.

Notes

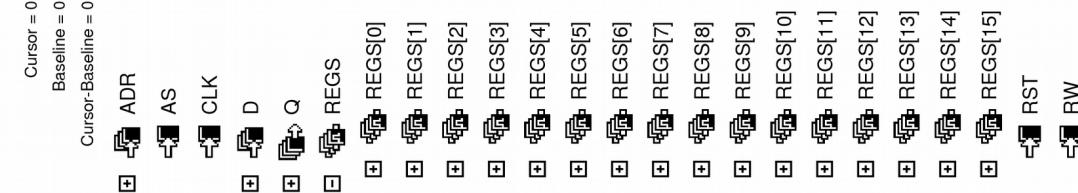
- Les signaux possédant le suffixe ****** signifie qu'ils sont actifs au niveau bas.

Tp2-2 : Simulation Banc Mémoire

Vous recopierez le fichier d'initialisation *rom_file.0.txt* et le testbench *test_memory.0.vhd* que vous ajouterez à votre design afin d'élaborer un *snapshot* que vous simulerez.

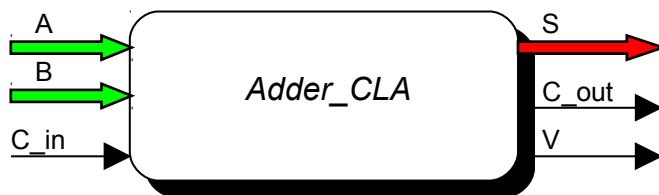
Notes

- *cpu_package.0.vhd* défini le test d'un banc mémoire à **16** éléments de **32** bits à écriture sur front **montant**.

Banc Mémoire

Tp3-1 : Additionneur Carry Look-Ahead

On se propose de réaliser un additionneur à retenue anticipée possédant l'interface suivante :



Vous utiliserez le fichier *cpu_package.1.vhd* dans lequel vous définirez les spécifications d'interfaces et les corps de fonction/procédure.

```

procedure adder_cla (A,B : in std_logic_vector;
                     C_in : in std_logic;
                     S : out std_logic_vector;
                     C_out,V : out std_logic);

```

Vous utiliserez la formulation suivante pour le calcul :

```

G=A and B; P=A or B;
C_CLA(i+1)=G(i) or (P(i) and C_CLA(i));
S=(A xor B) xor C_CLA(____);

```

Enfin vous surchargerez l'opérateur "+" de façon à utiliser votre procédure *adder_cla*

```
function "+" (A,B : in std_logic_vector) return std_logic_vector;
```

Notes

- Vous utiliserez une structure de boucle


```

for I in 0 to ... loop
    C_CLA(i+1)=G(i) or (P(i) and C_CLA(i));
    ...
end loop;
      
```
- Drapeau Overflow $V = C_{out} \text{ xor } < C_{in} \text{ du dernier bit} >$ est un débordement accidentel de l'avant dernier bit sur le bit de signe.
- L'appel d'une procédure est une instruction (process only) qui peut modifier des signaux globaux en plus des paramètres *out inout* alors qu'un fonction s'utilise partout et renvoie une seule information pouvant être une structure.

Tp3-2 : Simulation Adder_CLA

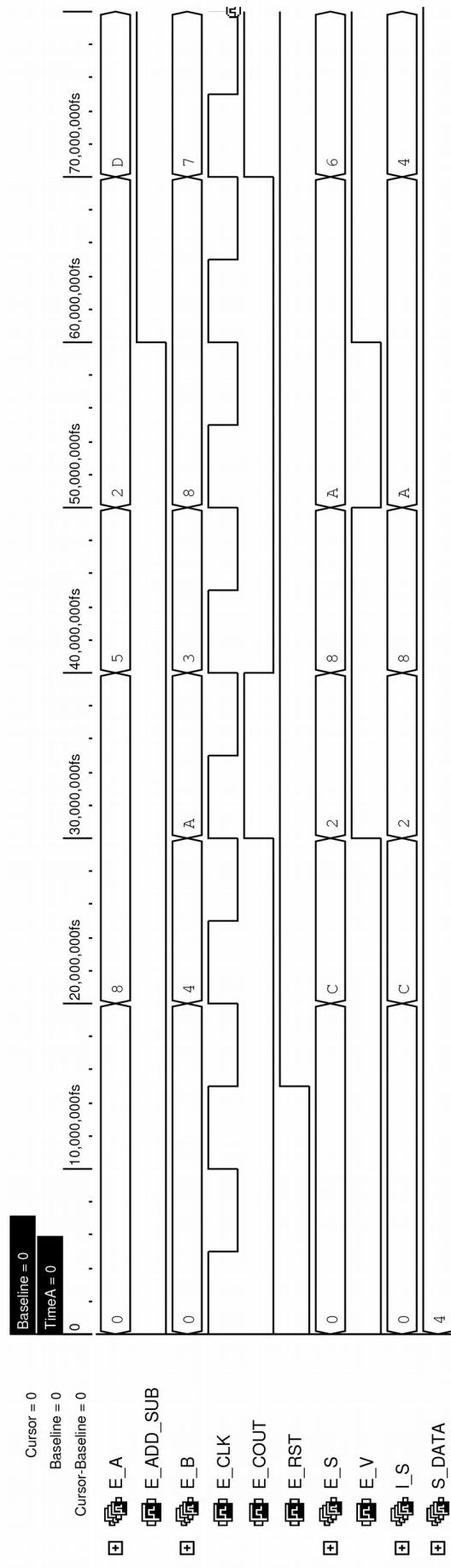
Vous compléterez le fichier *test_adder_cla.vhd* dans lequel vous inclurez l'architecture nécessaire à des opérations **additions** mais également **soustractions**.

Notes

- Opérandes de 4bits pour le test; les valeurs signés iront donc de +7 à -8.
- Les nombres négatifs sont représentés en complément à 2 (c.a.d complément à 1 + 1).
- Une **soustraction** $A - B$ est équivalente $A + (\text{not } B) + 1$.
- Dans quel type d'opération faut-il tenir compte du drapeau d'overflow V ?

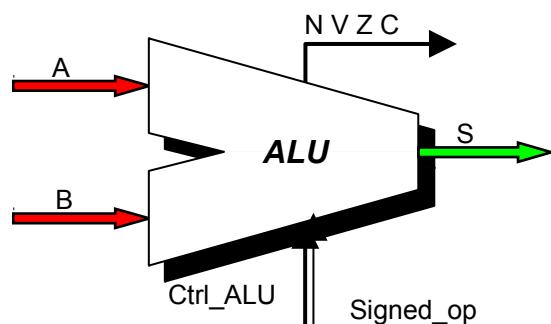
Adder Carry Look-Ahead

Page 1 of 1



Tp4-1 : Unité Arithmétique Logique (ALU)

On se propose de réaliser une ALU possédant l'interface suivante :



Dans votre fichier `cpu_package.1.yhd`, vous utiliserez le type énuméré :

type ALU OPS is (ALU ADD, ALU SUB, ALU SLT, ALU LSL, ALU LSR, ALU AND, ALU OR, ALU NOR, ALU XOR);

représentant les opérations suivantes - *signal CTRL ALU* - :

- ◆ ADD Addition signée ou non, $S \leq A + B$
 - ◆ SUB Soustraction signée ou non, $S \leq A - B$
 - ◆ SLT Positionner si < signé ou non, $S \leq 1$ si $A < B$, 0 sinon
 - ◆ LSL Décalage logique à gauche, $S \leq A$ décalé de B bits vers la gauche
 - ◆ LSR Décalage logique à droite, $S \leq A$ décalé de B bits vers la droite
 - ◆ AND, OR, NOR, XOR, $S \leq A \text{ AND } B$, $S \leq A \text{ OR } B$, $S \leq A \text{ NOR } B$, $S \leq A \text{ XOR } B$

Le signal SIGNED_OP indiquera si l'opération en cours est signée ou non.

Les signaux N, V, Z, C représentent respectivement une valeur négative en sortie, un overflow, la sortie S à 0 et la carry de l'adder.

```
procedure alu (A,B : in std_logic_vector;
              signal S : out std_logic_vector;
              signal N,V,Z,C : out std_logic;
              SIGNED_OP : in std_logic;
              CTRL_ALU : in ALU_OPS);
```

Notes

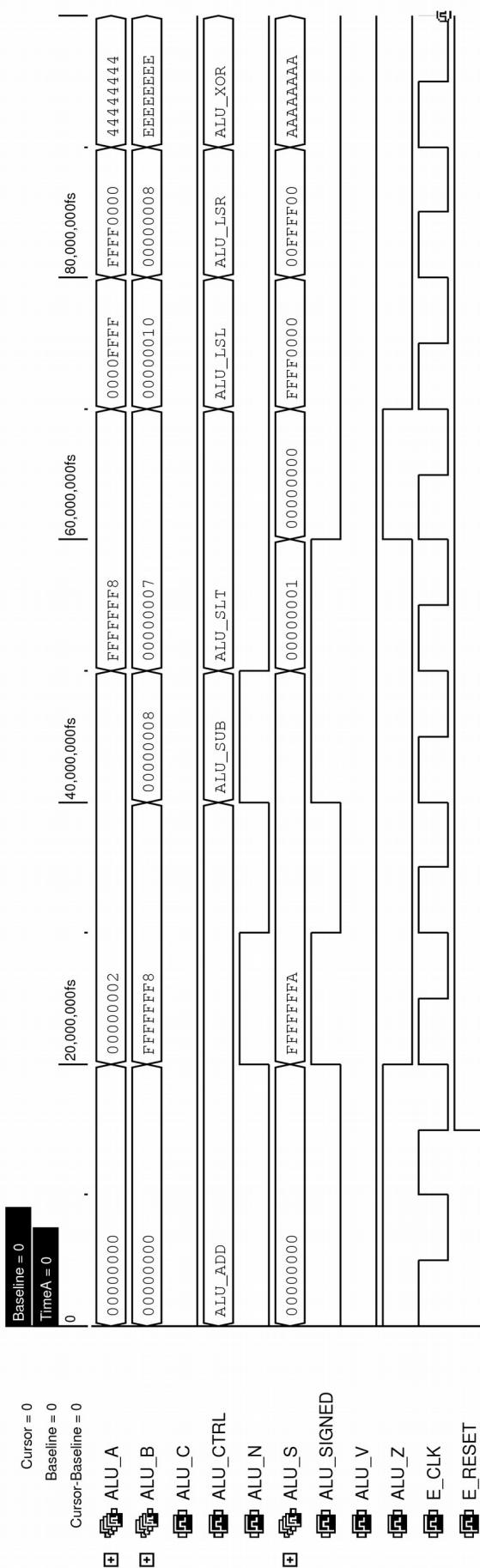
- Notes**

 - Vous remarquerez l'**attribut signal** sur les paramètres de sortie de la procédure.
 - Les fonctions *shl* et *shr* respectivement décalage à gauche et à droite sont disponibles via la librairie arithmétique.
 - Logique de test pour Positionner si $< : \text{SLT} = \text{signed.(N xor V)} + / \text{signed./Carry}$
 - Les flags NVZC ne devront être affectés que si l'opération est significative et n'auront pas à être mémorisés dans l'attente d'une éventuelle nouvelle opération les affectant.

Tp4-2 : Simulation ALU

Vous utiliserez le fichier *test_alu.vhd* qui se focalise sur le test des fonctionnalités autre que l'additionneur, ce dernier ayant déjà fait l'objet de tests lors du précédent exercice.

Notes

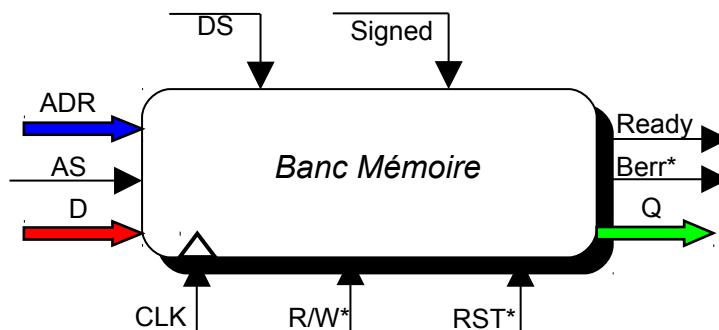
ALU

Tp5-1 : Banc Mémoire pour µP RISC

On se propose de réaliser une nouvelle version du banc mémoire générique à **lecture combinatoire** qui cette fois sera capable d'opérer des accès au format 8/16/32/64bits et plus avec ou sans extension de signe. Cette nouvelle version n'est cependant toujours pas un cache à proprement parler.

Les paramètres génériques de cet élément sont les suivants :

- DBUS_WIDTH, Taille d'un mot du banc avec 32 bits par défaut.
- ABUS_WIDTH, Largeur du bus d'adresses avec 32 bits par défaut.
- MEM_SIZE, Nombre de mots mémoire avec 16 par défaut.
- FILENAME, Fichier d'initialisation avec <chaîne vide> par défaut.



Principe de fonctionnement

L'activation du signal **RST*** provoque le chargement immédiat soit d'un fichier passé en paramètre générique soit de la valeur **0** dans tous les éléments du banc mémoire.

L'activation du signal **AS** indique une opération valide de **lecture immédiate** si **R/W*=1** ou bien d'une écriture au prochain front montant d'horloge si **R/W*=0**.

Une opération de lecture / écriture fait référence à un emplacement mémoire défini par la valeur sur le bus **ADR** et le type d'accès 8/16/32/64bits défini par le signal **DS**. Attention aux accès non alignés —**Berr<=0'** ;

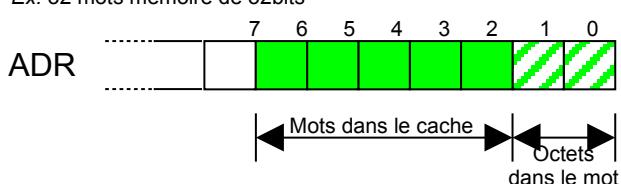
Le signal **Signed** enfin ne concerne que les accès lecture et permet l'extension de signe de la donnée lue.

Dans le cas où il n'y a pas d'opération sur la mémoire, signal **AS** inactif, la sortie **Q** est à l'état haute impédance noté **Z**.

Implémentation

Le bus d'adresse **ADR** est au format octet et cette fois d'une taille indépendante des **MEM_SIZE** mots mémoire. Ces mots mémoire étant eux au format **DBUS_WIDTH**, vous devez donc évaluer le nombre de bits d'adresse nécessaires à l'encodage du nombre d'octets dans un mot puis le nombre de bits d'adresse pour l'accès aux **MEM_SIZE** éléments.

Ex: 32 mots mémoire de 32bits



Ex : Extension de signe d'une lecture (signée) avec 32 bits DATA

lb	31	23	15	7	0
lh	S	S	S	S	
lw	S	S	S	S	

Ex : Ecriture avec 32 bits DATA

sb	31	23	15	7	0
sh					
sw					

Les exceptions liées au signal **Berr*** seront traitées :

- Accès 8bits → tous octets,
- Accès 16bits → alignment sur mots de 16bits.

Vous utiliserez le fichier *memory.1.vhd* pour la définition de l'entité et son architecture.

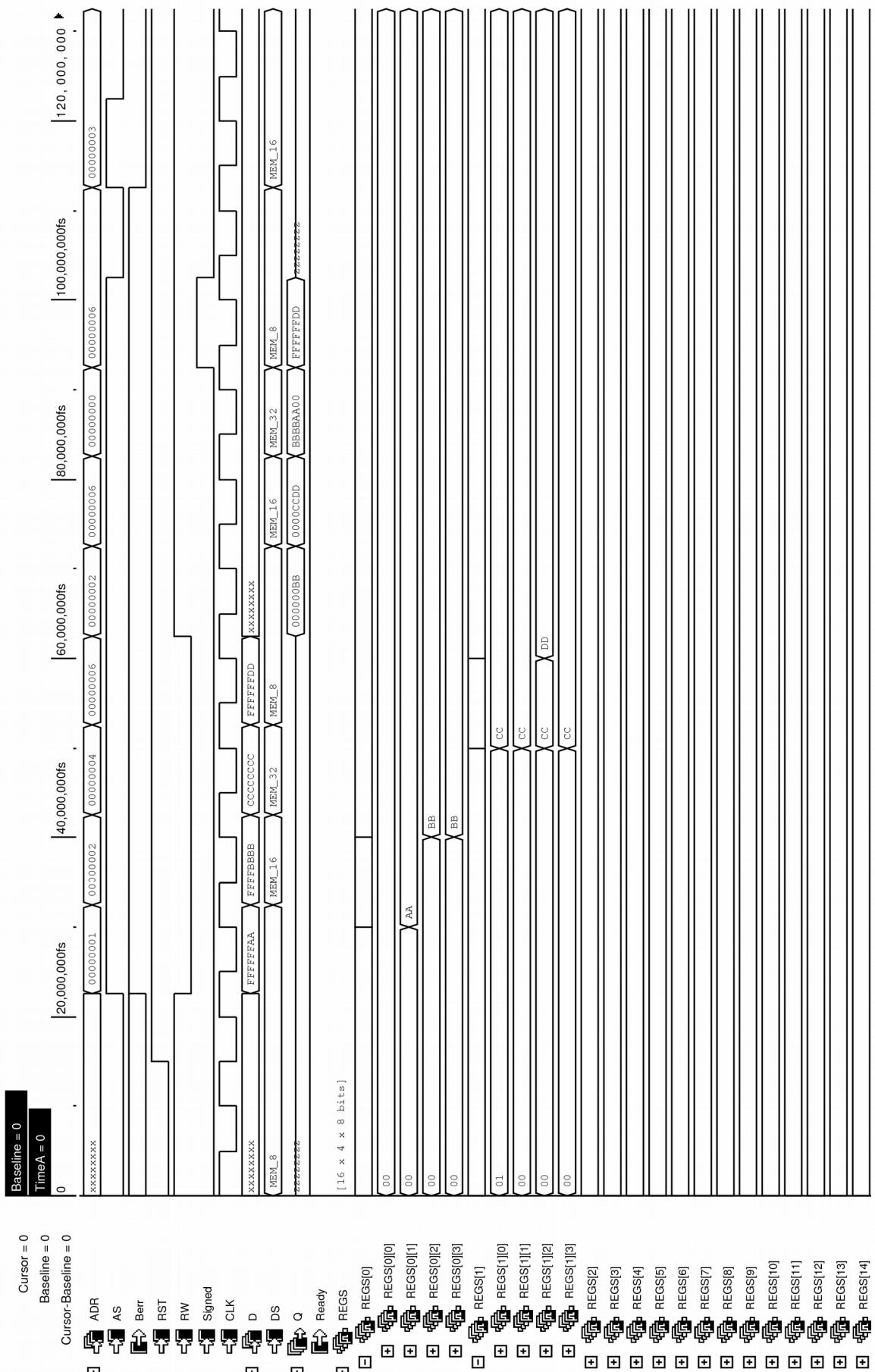
Notes :

- Les signaux possédant le suffixe ****** signifie qu'ils sont actifs au niveau bas.
- **TOUS** les accès se font en **un seul cycle**.

Tp5-2 : Simulation Banc Mémoire

Vous utiliserez le testbench *test_memory.1.vhd* et votre *cpu_package* de façon à procéder à la simulation d'un banc mémoire de **16** éléments de **32** bits à écriture sur front **montant**.

Memory with sign extension

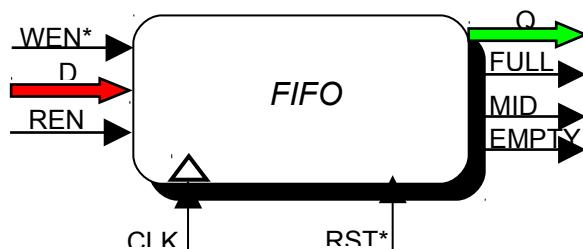


Tp8-1 : FIFO double port

On se propose de réaliser une FIFO* synchrone double port selon les caractéristiques génériques suivantes :

- DBUS_WIDTH Taille d'un mot du banc avec 32 bits par défaut.
- ABUS_WIDTH Profondeur d'utilisation avec 3 bits d'adresse par défaut - soit 2^{ABUS_WIDTH} mots -.

*First In First Out



Principe de fonctionnement

Le signal RST*, synchrone, remet à **0** les pointeurs lecture et écriture **mais n'efface pas** le contenu des registres. Il positionne également la sortie **Q <= 'Z'**.

Tant que WEN* est actif, la FIFO continue de se remplir de façon circulaire à chaque **front montant** d'horloge écrasant ainsi les données les plus anciennes.

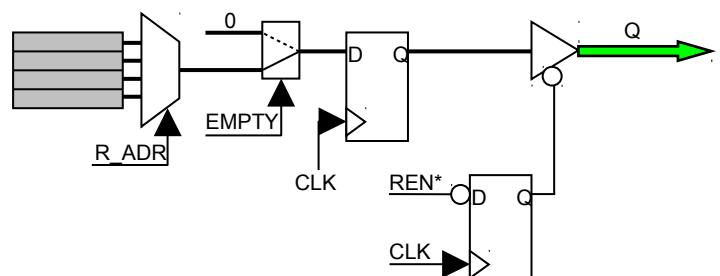
Tant que REN* est actif, la FIFO continue de présenter une donnée sur le bus à chaque **front montant** d'horloge jusqu'à être vide, auquel cas elle présentera la valeur **0**.

Le bus de sortie Q devant être raccordé à d'autres unités, la sortie passe en haute impédance quand REN* inactif.

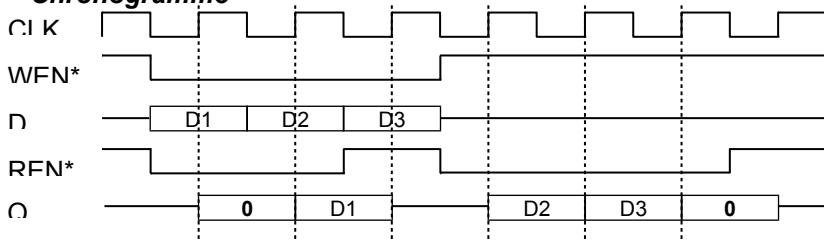
Les signaux EMPTY, MID et FULL représentent respectivement l'état vide, au moins à moitié plein* et plein. Ces derniers seront mis à jour sur front montant d'horloge.

*Le signal MID=1' quand remplissage FIFO $\geq 50\%$.

Les signaux WEN* et REN* peuvent être actifs simultanément.



Chronogramme



Implémentation

Vous compléterez le fichier *fifo.0.vhd* comprenant entité et architecture.

La lecture / écriture étant circulaire, vous ferez attention aux conditions limites pour l'établissement des indicateurs de remplissage. Ensuite, afin d'accélérer les transferts, les pointeurs devront déjà être positionnés sur le futur emplacement de lecture ou d'écriture, l'incrément se faisant après l'opération proprement dite. Enfin, **le pointeur de lecture référence toujours la donnée la plus ancienne**.

Cas particuliers

Lecture & Ecriture & FIFO vide \rightarrow Q=0 & mémorisation de la donnée en entrée.

FIFO pleine & Ecriture \rightarrow mémorisation de la donnée et incrément du pointeur de lecture.

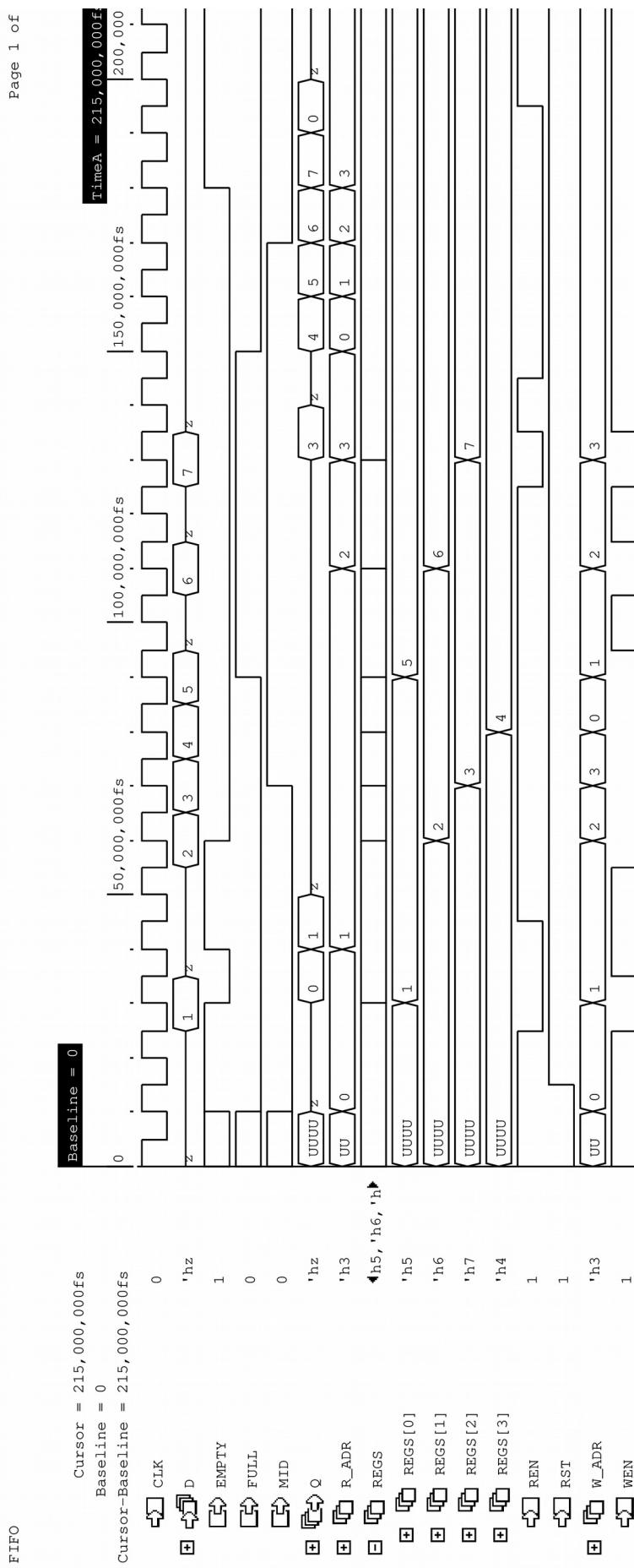
FIFO pleine & Lecture & Ecriture \rightarrow Q=donnée la plus ancienne et mémorisation de la nouvelle.

Notes

- Le type **buffer** associé à un signal permet de relire un port configuré en sortie.

Tp8-2 : Simulation FIFO double port

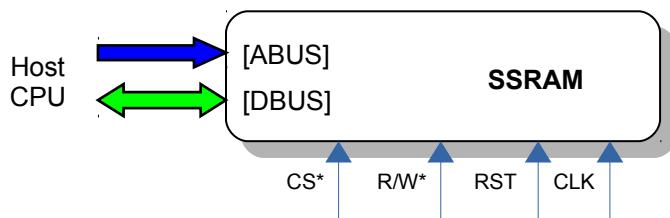
Vous compléterez le fichier *test_fifo.0.vhd* avec une instance du composant *fifo* possédant 4 mots mémoire de 4 bits.



Tp9-1 : Mémoire Statique Synchrone

Vous coderez en VHDL une SSRAM (Synchronous Static RAM) possédant les caractéristiques génériques suivantes :

- ◆ ABUS_WIDTH Taille du bus d'adresse avec une valeur par défaut fixée à **4** bits, soit 16 mots mémoire.
- ◆ DBUS_WIDTH Taille du bus de données avec une valeur par défaut fixée à **8** bits.
- ◆ CS_LATENCY Latence d'accès après activation du CS*, avec **4** cycles d'horloge par défaut.
- ◆ I2Q Délai de propagation en lecture entre l'adresse et la donnée en sortie sur [DBUS] avec **2** ns par défaut.



Principe

La SSRAM échantillonne l'adresse du mot mémoire concerné par une future opération uniquement sur le **front descendant** du signal CS*. Ces opérations de type lecture(s) ou écriture(s) sont définies par l'état du signal R/W* respectivement à 1 ou 0.

Un cycle lecture ou écriture sera amorcé CS_LATENCY front montant d'horloge après activation du signal CS*. Il peut durer une ou plusieurs périodes d'horloge avec, et dans **presque** tous les cas, la donnée présente sur [DBUS] qui sera

- stockée à l'adresse en cours (cas d'une écriture).
- fournie par le mot mémoire spécifié par l'adresse en cours (cas d'une lecture).

Cette adresse est alors tacitement incrémentée **jusqu'à la capacité maximale** de la SSRAM. Au delà, les écritures sont inefficaces et les lectures renvoient l'état haute impédance.

Lors d'un cycle de lecture, les données sont présentées sur le bus après un délai défini par le paramètre générique I2Q, cela permet de simuler le délai de propagation d'une vraie bascule. Enfin, CS*=1' termine la transaction en cours.

Notes

Les flèches doubles définissent un signal bidirectionnel *inout*.

*' associé à un signal dénote que ce dernier est actif à l'état bas.

Implémentation

Après délai de latence et sur front montant d'horloge, les données sont échantillonnées par la SSRAM (R/W*=0') ou par le CPU hôte (R/W*=1'). Tous les signaux générés par l'hôte changent d'état après le front montant CLK.

La taille d'un mot mémoire = DBUS_WIDTH.

Le bus [DBUS] devra être positionné à l'état haute impédance lorsqu'il n'y a aucune transaction en cours (CS*=1') ou lorsqu'il y a écriture par le processeur hôte.

Lorsque RST*=0', les éléments mémoire sont initialisés avec la valeur 0.

Mise en œuvre et tests

Complétez le fichier ssram.0.vhd

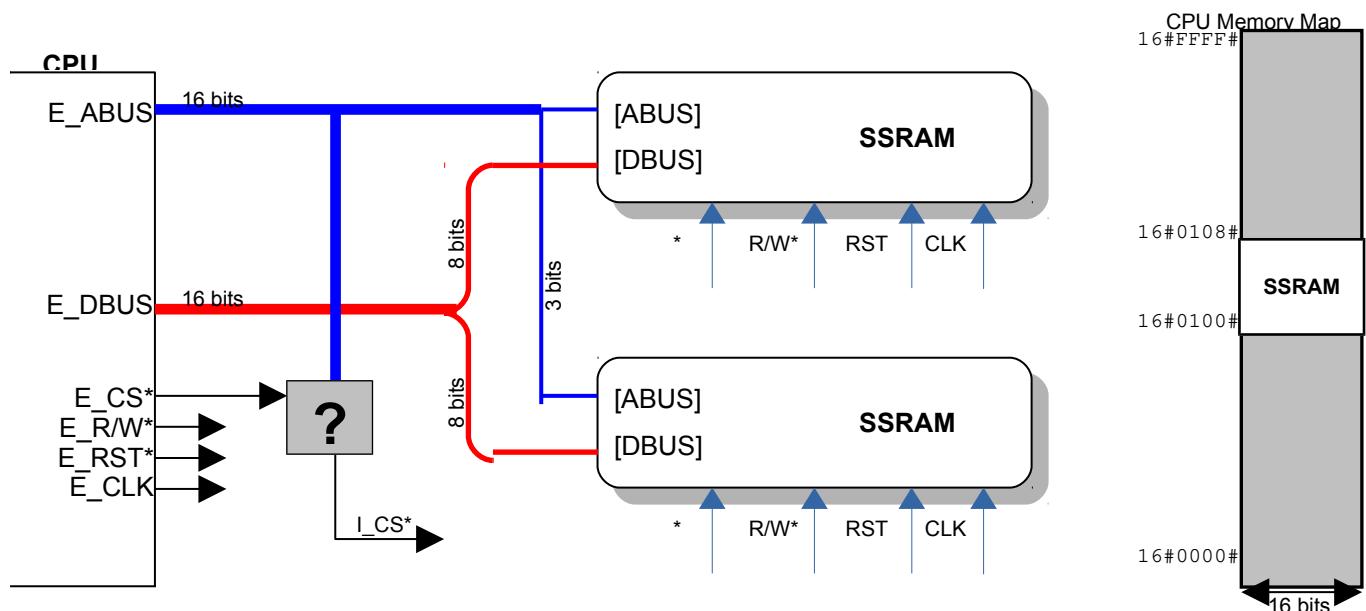
Le fichier test_ssram.0.vhd correspond à la simulation d'une SSRAM 8 mots mémoire de **8** bits.

Le second fichier test_ssram.0.1.vhd concerne la seconde partie.

tournez la page svp ...

Tp9-2 : Banc mémoire pour processeur

Vous allez à présent mettre en œuvre une architecture pour le test d'un processeur 16 bits [DBUS] et 16 bits [ABUS] qui accède à deux SSRAMs 8 bits [DBUS] et 3 bits [ABUS] en parallèle à partir de l'adresse 16#0100#.



Implémentation

Le processeur 16 bits accède aux deux SSRAMs 8 bits en parallèle de l'adresse 16#0100# → 16#0107# puisque chacune à une capacité de 8 octets. Enfin le processeur possède un adressage 16 bits uniquement, c.à.d que chaque adresse représente un mot de 16 bits.

Mise en œuvre et tests

Vous implémenterez l'architecture décrite ci-dessus en complétant fichier *test_ssram.0.1.vhd*.

Mémoire statique synchrone

