Unité de multiplication en virgule flottante

1 Introduction

Ce document constitue le cahier des charges de ce qui vous est demandé dans ce laboratoire de matériel informatique: concevoir et réaliser une unité de multiplication en virgule flottante.

La section 2 décrit le format des nombres que vous allez manipuler. La section 3 expose l'algorithme à utiliser pour la multiplication. Finalement la section 4 décrit les protocoles d'entrée / sortie.

2 Format

Les différentes façons dont les nombres réels peuvent être représentés dans un ordinateur sont traitées en détail dans le livre *Informatique Industrielle I*, aux pages 43 à 53.

Le format adopté dans le cadre de ce laboratoire est un format simplifié, codé sur 16 bits, dont le détail est donné dans la figure 1.

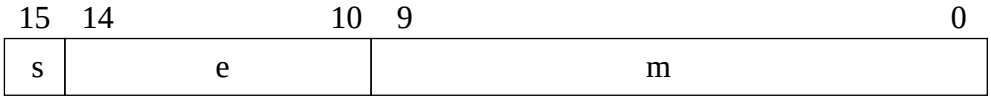


Fig. 1 Format

Ce format se divise en trois champs.

- La *mantisse* (m), ou fraction, d'une largeur de 10 bits, est *normalisée*, ce qui signifie que le bit de poids fort est le premier chiffre binaire significatif du nombre codé.

- L'*exposant* (e), d'une largeur de 5 bits, est *biaisé*, ce qui signifie qu'un *biais* de 16 est systématiquement ajouté de manière à ramener les valeurs de l'intervalle - 16 .. + 15 à l'intervalle 0 .. 31.

- Le *signe* (s), d'une largeur de 1 bit, vaut 1 si la mantisse est négative.

Soit par exemple à représenter le nombre - 9.625 dans ce format. En binaire, ce nombre s'écrit, (1)

$$- 1001.101 \quad \textbf{(1)}$$

Si on introduit une puissance de 2, on obtient, (2)

$$- 1001.101 * 2^0 \quad \textbf{(2)}$$

Et si l'on élimine la partie entière, (3)

$$- 0.1001101 * 2^4 \quad \textbf{(3)}$$

Le codage du nombre - 9.625 est donné dans la figure 2.

15	14				10	9									0
1	1	0	1	0	0	1	0	0	1	1	0	1	0	0	0

Fig. 2 Codage de - 9.625

Par convention le zéro est codé par une mantisse nulle, un exposant nul et un signe positif (fig. 3).

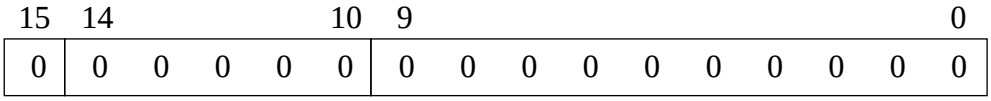


Fig. 3 Codage de zéro

3 Algorithme

3.1 Principe

Soient deux nombres réels, A et B, de mantisse m_A, resp. m_B, d'exposant (non-biaisé) e_A, resp. e_B, et de signe s_A, resp. s_B, (4), (5).

$$A = (- 1)^{sA} * m_A * 2^{eA} \quad \textbf{(4)}$$
$$B = (- 1)^{sB} * m_B * 2^{eB} \quad \textbf{(5)}$$

Le produit P est alors donné par (6), (7) et (8).

$$P = A * B \quad \textbf{(6)}$$
$$P = ((- 1)^{sA} * m_A * 2^{eA}) * ((- 1)^{sB} * m_B * 2^{eB}) \quad \textbf{(7)}$$
$$P = (- 1)^{(sA + sB)} * (m_A * m_B) * 2^{(eA + eB)} \quad \textbf{(8)}$$

Si l'on décompose à son tour le produit P en mantisse, exposant et signe, alors la mantisse m_P est donnée par (9), l'exposant e_P par (10) et le signe s_P par (11).

$$m_P = m_A * m_B \quad \textbf{(9)}$$
$$e_P = e_A + e_B \quad \textbf{(10)}$$
$$s_P = s_A + s_B \quad \textbf{(11)}$$

La multiplication de deux nombres à virgule flottante peut donc se diviser en trois opérations.

- ❏ Multiplication des mantisses.
- ❏ Addition des exposants.
- ❏ Détermination du signe.

On remarque que ces trois opérations sont indépendantes les unes des autres et qu'elles se prêtent donc à une exécution en parallèle.

3.2 Dépassement de capacité

La multiplication des mantisses ne peut pas générer de dépassement de capacité car elle est calculée en double précision dans une mantisse intermédiaire d'une largeur de 20 bits (paragraphe 3.4).

Il n'en n'est pas de même de l'addition des exposants. Il se peut très bien que deux nombres soient tels que l'exposant résultant ne puisse pas être représenté correctement sur 5 bits. Deux cas sont à distinguer.

- ❏ L'exposant résultant est supérieur à +15 (non-biaisé), il y a alors *overflow* et le résultat ne peut pas être récupéré.
- ❏ L'exposant résultant est inférieur à -16 (non-biaisé), il y a alors *underflow* mais il est possible de récupérer le résultat en l'arrondissant à zéro.

3.3 Normalisation

L'hypothèse est faite que les deux nombres à multiplier sont toujours correctement normalisés. Or même sous cette hypothèse, il se peut que la mantisse du produit soit dénormalisée, ce qui signifie que son bit de poids le plus fort ne vaut pas 1, mais 0. Le produit doit alors être normalisé.

Pour normaliser un nombre, on décale sa mantisse vers la gauche jusqu'à ce qu'un 1 occupe le bit de poids le plus fort, et on retranche à son exposant le nombre de bits qu'il a fallu décaler.

Soit par exemple à normaliser le nombre de la figure 4.

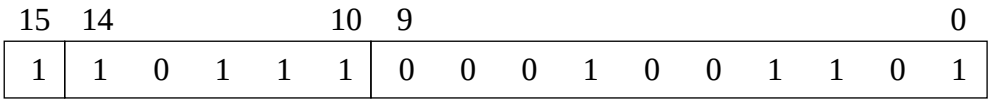


Fig. 4 Nombre dénormalisé

Il faut opérer un décalage de 3 bits vers la gauche pour obtenir une mantisse normalisée. L'exposant est donc diminué de 3 (figure 5).

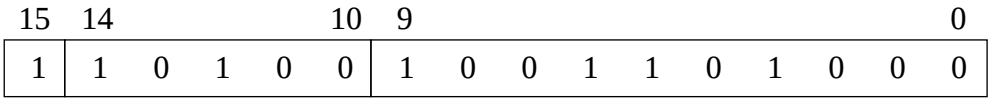


Fig. 5 Nombre normalisé

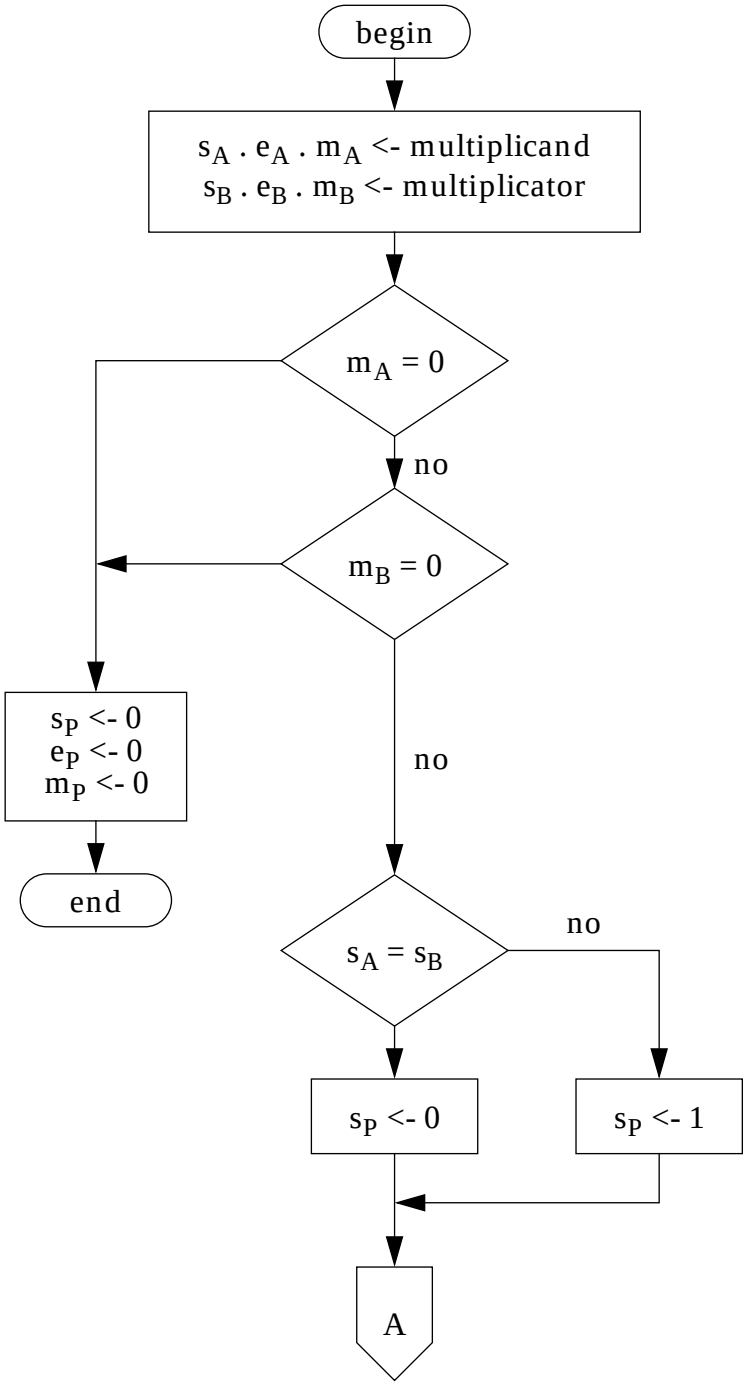
On remarque que la normalisation, par son action sur l'exposant, peut conduire à un underflow, mais peut aussi faire disparaître un overflow.

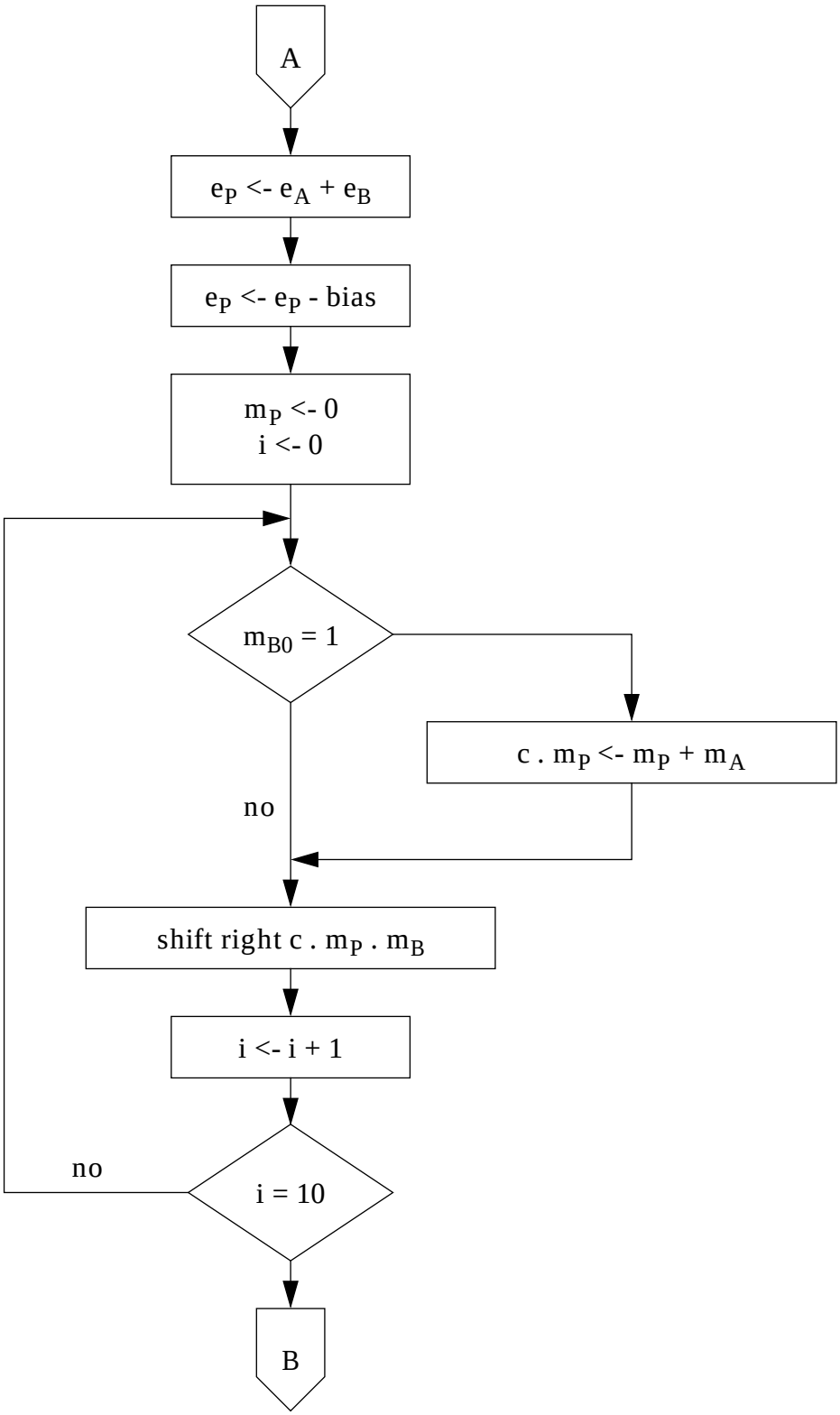
3.4 Organigramme

L'organigramme donné dans la figure 6 représente une version séquentielle de l'algorithme de multiplication proposé: l'algorithme *add-shift*. Comme il a été indiqué au paragraphe 3.1, on peut imaginer une version parallèle de cet algorithme, et vous êtes encouragés à essayer.

Lors du calcul de la mantisse du produit, cet algorithme concatène les registres m_P et m_B afin de doubler la précision du résultat temporaire. En outre il fait appel à un registre de 1 bit, c, pour mémoriser une éventuelle retenue.

L'opérateur . indique la concaténation des registres, l'opérateur <- indique le chargement des registres.


begin
s_A . e_A . m_A <- multiplicand
s_B . e_B . m_B <- multiplicator
$m_A = 0$
no
$m_B = 0$
s_P <- 0
e_P <- 0
m_P <- 0
end
no
$s_A = s_B$
no
s_P <- 0
s_P <- 1
A


A
e_P <- e_A + e_B
e_P <- e_P - bias
m_P <- 0
i <- 0
m_{B0} = 1
no
c . m_P <- m_P + m_A
shift right c . m_P . m_B
i <- i + 1
i = 10
no
B

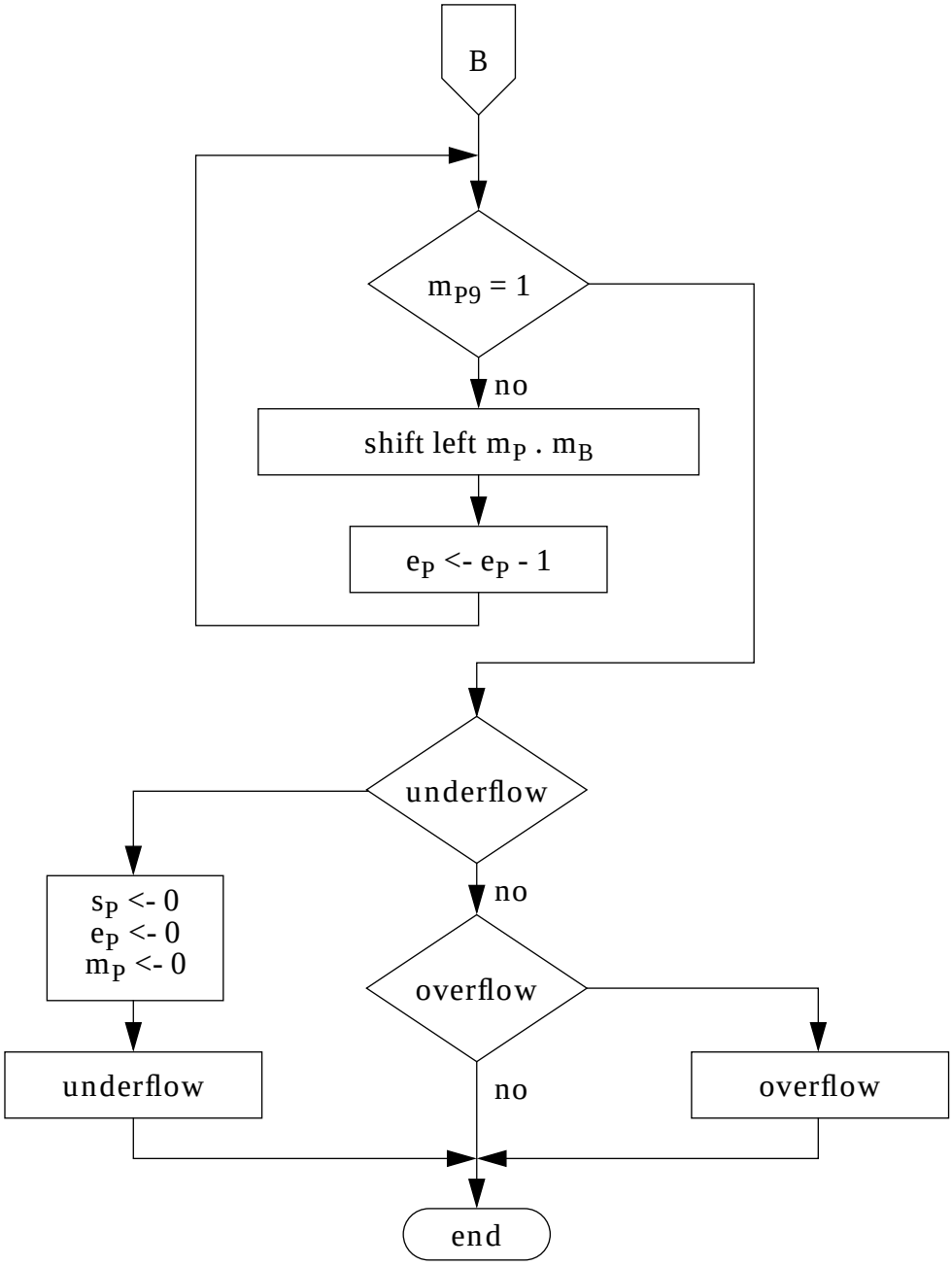


Fig. 6 Organigramme

4 Protocoles d'entrée/sortie

L'unité de multiplication va fonctionner comme co-processeur, ce qui implique qu'elle va être connectée à un processeur maître (fig. 7), avec lequel elle va devoir communiquer pour échanger des données telles qu'opérandes, résultat, mot de contrôle et mot d'état.

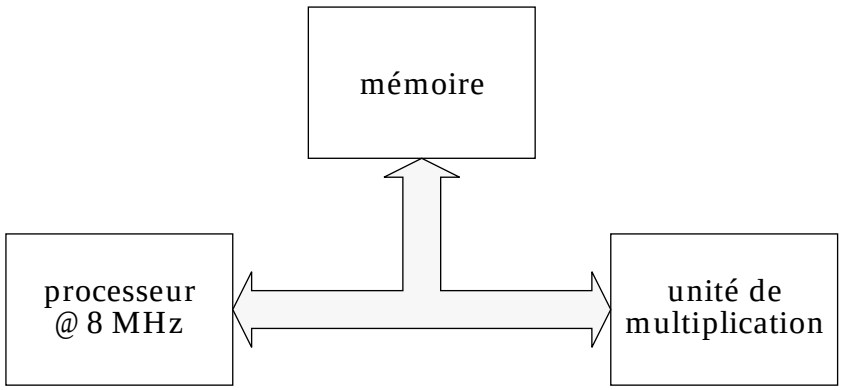


Fig. 7 Schéma-bloc

Pour effectuer une multiplication, les choses se déroulent de la façon qui suit.

- ❏ Le processeur écrit les opérandes en mémoire.
- ❏ Le processeur ordonne à l'unité de multiplication de multiplier.
- ❏ L'unité devient maître du bus, lit les opérandes en mémoire, et relâche le bus.
- ❏ L'unité effectue la multiplication.
- ❏ Le processeur boucle en attente de la fin de la multiplication.
- ❏ L'unité devient maître du bus, écrit le résultat en mémoire et relâche le bus.
- ❏ Le processeur lit le résultat en mémoire.
- ❏ Le processeur récupère les informations de dépassement de capacité.

La figure 8 détaille l'ensemble des signaux qui relient le processeur, l'unité de multiplication et la mémoire.

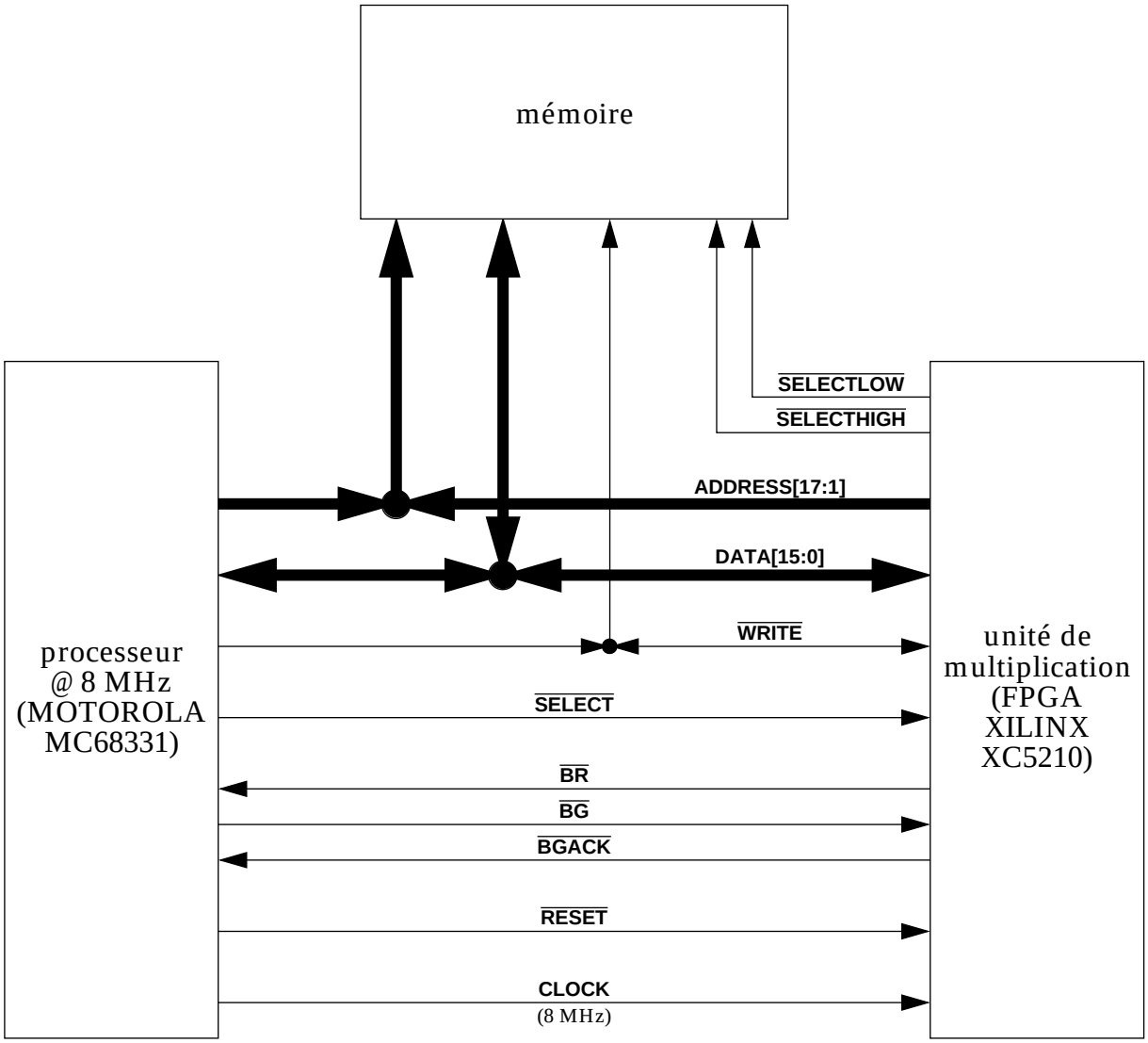


Fig. 8 Signaux

4.1 Opérandes et résultat

En temps normal, c'est le processeur qui contrôle le bus. Il accède à la mémoire en permanence pour y chercher en particulier la prochaine instruction à exécuter. Pour pouvoir accéder à la mémoire dans laquelle elle va lire les opérandes et écrire le résultat, l'unité de multiplication doit être capable de prendre le bus au processeur.

La prise du bus se fait par le jeu de trois signaux.

- ❏ $\overline{\text{BR}}$ (*Bus Request*)
- ❏ $\overline{\text{BG}}$ (*Bus Grant*)
- ❏ $\overline{\text{BGACK}}$ (*Bus Grant ACKnowledge*)

Elle se déroule de la façon suivante (fig. 9, fig. 10).

- ❏ L'unité veut prendre le bus: elle active $\overline{\text{BR}}$.
- ❏ Le processeur voit $\overline{\text{BR}}$ actif: il termine le cycle en cours, place le bus en état de haute-impédance et active $\overline{\text{BG}}$.
- ❏ L'unité voit $\overline{\text{BG}}$ actif: elle désactive $\overline{\text{BR}}$ et active $\overline{\text{BGACK}}$.
- ❏ Le processeur voit $\overline{\text{BGACK}}$ actif: il désactive $\overline{\text{BG}}$.
- ❏ L'unité est maître du bus: elle maintient $\overline{\text{BGACK}}$ activé tant qu'elle occupe le bus et peut accéder à la mémoire.
- ❏ L'unité veut rendre le bus: elle place le bus en état de haute-impédance et désactive $\overline{\text{BGACK}}$.
- ❏ Le processeur voit $\overline{\text{BGACK}}$ inactif: il reprend le contrôle du bus.

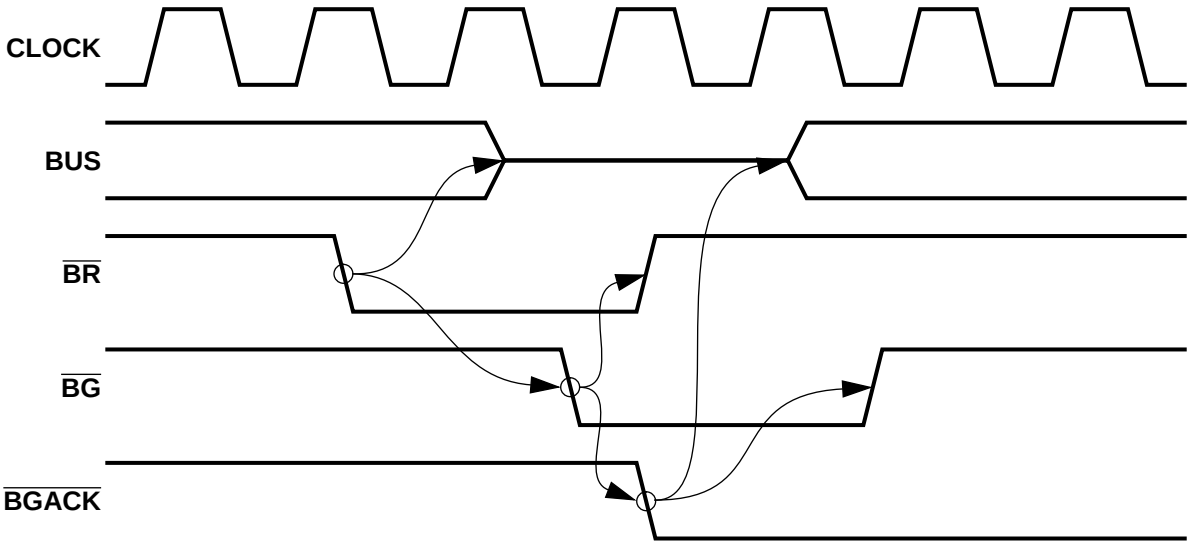


Fig. 9 Prise du bus

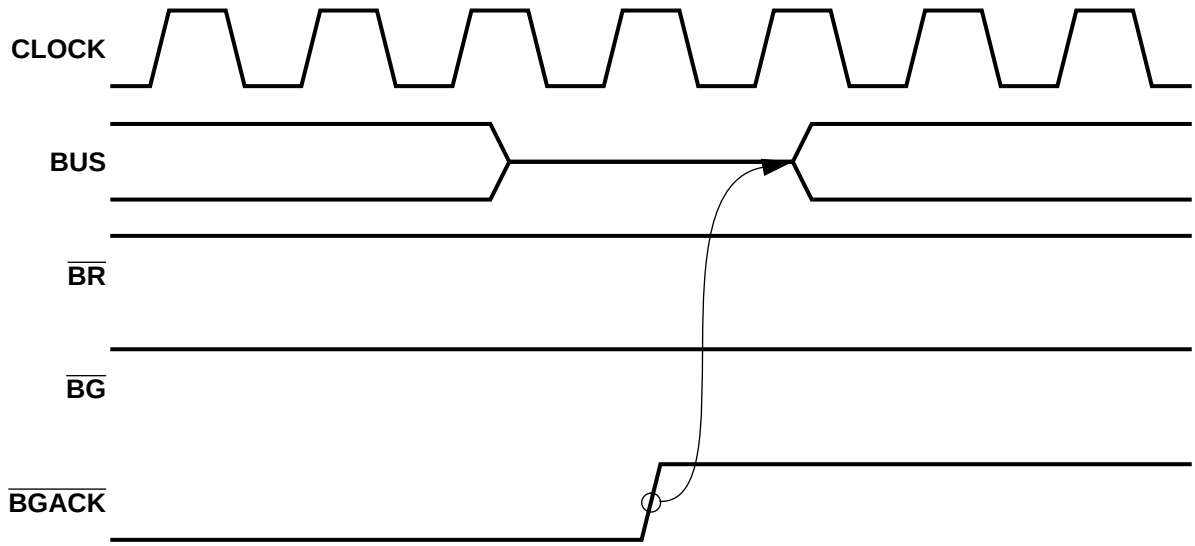


Fig. 10 Reddition du bus

La mémoire que l'unité va accéder est une mémoire de 128k x 16. La figure 11 décrit son interface.

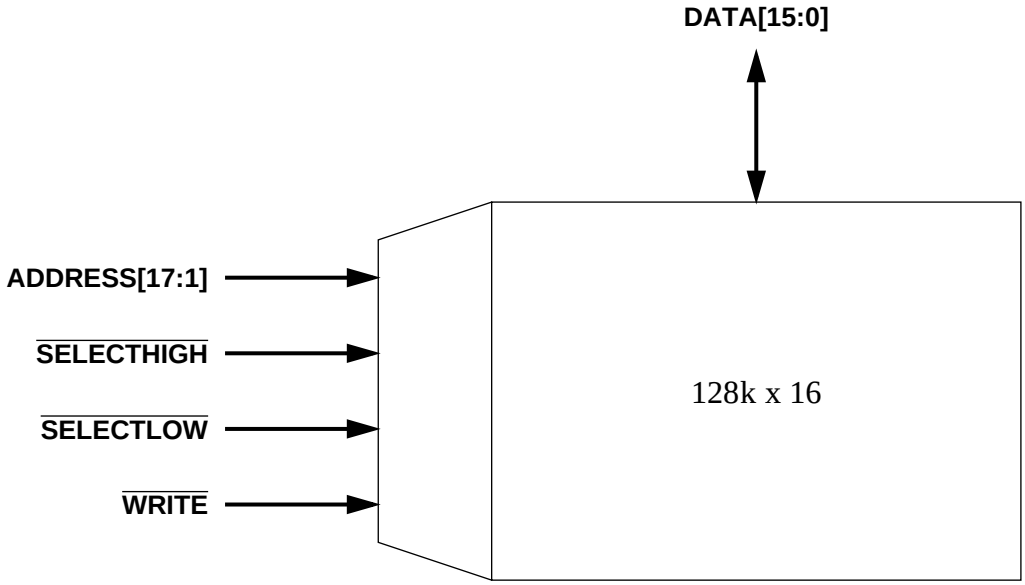


Fig. 11 Mémoire

Les figures 12 et 13 donnent respectivement les chronogrammes d'accès à cette mémoire en lecture et en écriture.

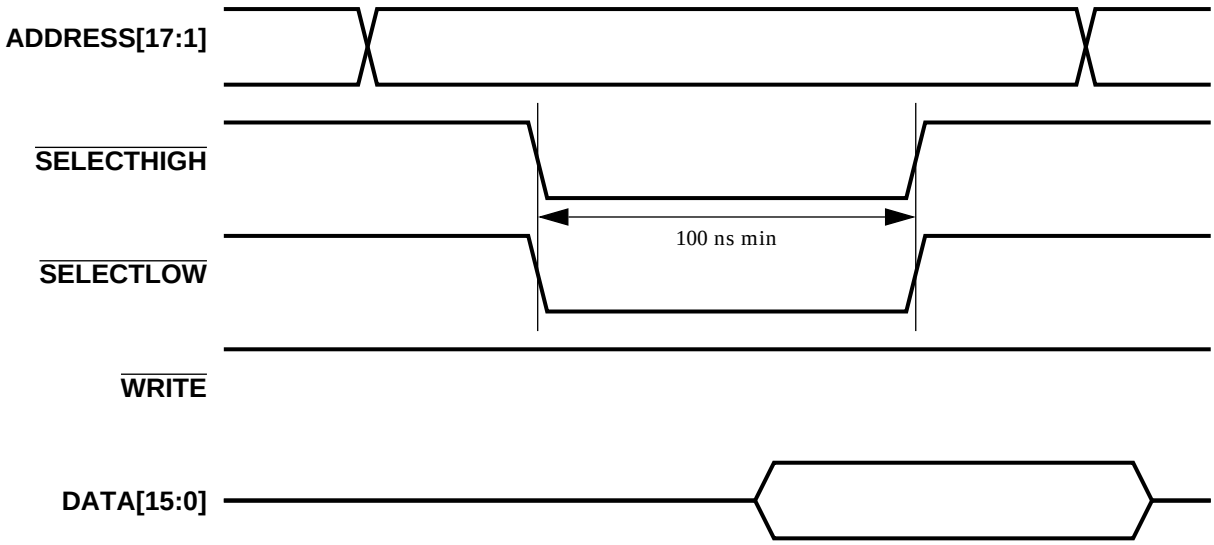


Fig. 12 Accès en lecture

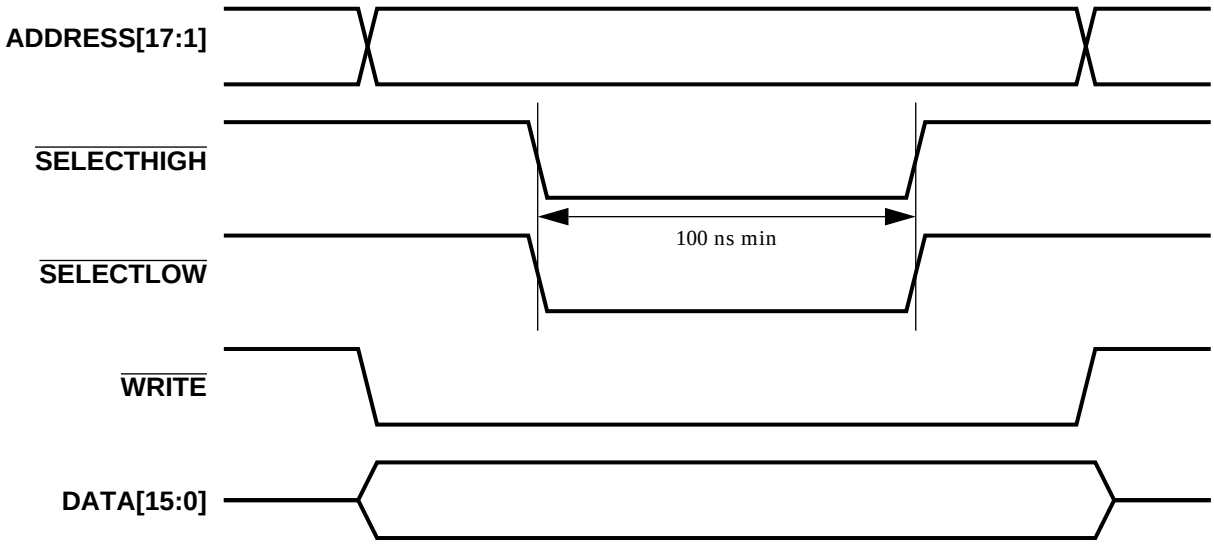


Fig. 13 Accès en écriture

Pendant l'activation de $\overline{\text{SELECTHIGH}}$ et $\overline{\text{SELECTLOW}}$, vous devez vous assurer que toutes les autres lignes qui entrent dans la mémoire sont stables.

Les adresses des opérandes et du résultat sont données par (12), (13) et (14).

Multiplicande lu en ADDRESS[17:1] = 1 0000 0000 0000 0000 (0x10000) **(12)**
Multiplicateur lu en ADDRESS[17:1] = 1 0000 0000 0000 0001 (0x10001) **(13)**
Résultat écrit en ADDRESS[17:1] = 1 0000 0000 0000 0000 (0x10000) **(14)**

Remarquez que le résultat vient écraser le multiplicande en mémoire.

4.2 Contrôle et état

Le processeur, lorsqu'il est maître du bus, peut écrire un mot de contrôle dans l'unité et lire un mot d'état depuis l'unité. Les chronogrammes des figures 14 et 15 illustrent ces opérations.

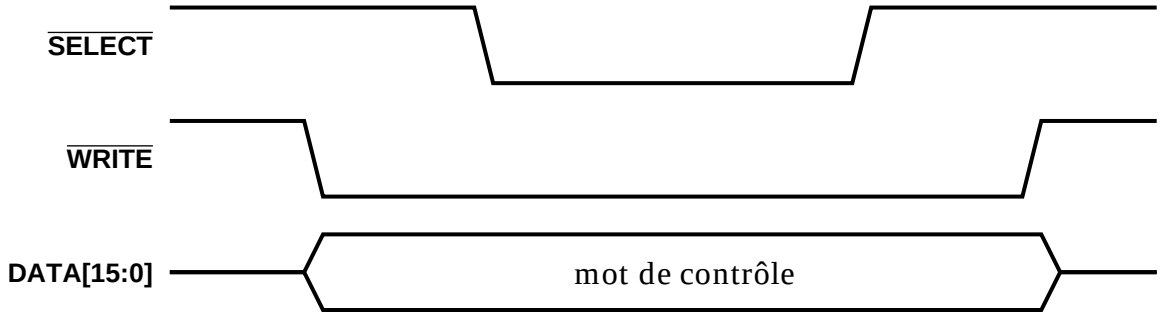


Fig. 14 Ecriture du mot de contrôle

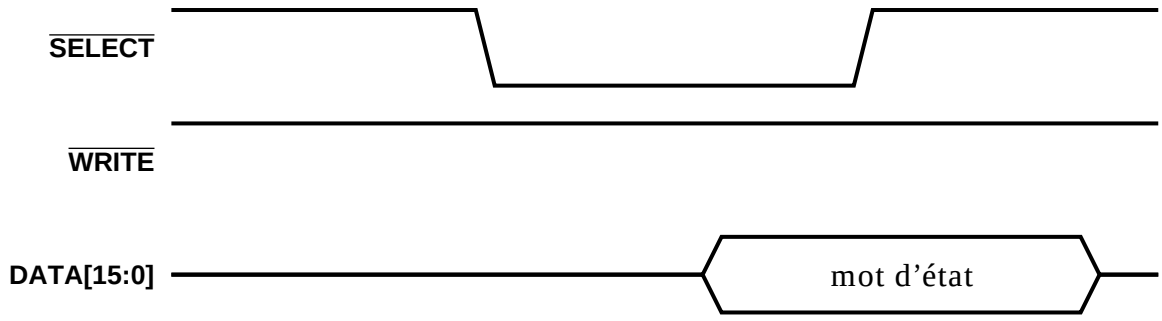


Fig. 15 Lecture du mot d'état

Le mot de contrôle ne comporte que 1 bit significatif (fig. 16). Ce bit (G, *Go*) permet d'ordonner à l'unité de commencer la multiplication. Quand l'unité est en train de multiplier, elle doit ignorer tout nouveau mot de contrôle jusqu'à ce qu'elle ait terminé.

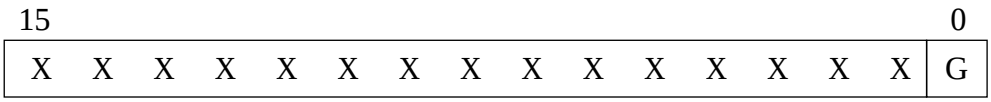


Fig. 16 Mot de contrôle

Le mot d'état comporte 3 bits significatifs (fig. 17). Le bit B, *Busy*, indique si l'unité est en train de multiplier. Ce bit doit passer à 1 dès que l'unité reçoit l'ordre de multiplier et doit repasser à 0 dès qu'elle est prête pour une nouvelle multiplication. C'est ce bit que le processeur vient lire périodiquement pour savoir s'il peut aller récupérer le résultat en mémoire. Les bits U, *Underflow*, et O, *Overflow*, ne sont significatifs que lorsque le bit B vaut 0. Ils indiquent naturellement les éventuels dépassements de capacité de la multiplication précédente. Si par exemple elle s'est soldée par un overflow, le bit O sera mis à 1, sinon, il sera mis à 0.

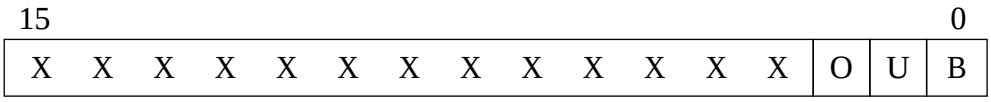


Fig. 17 Mot d'état

4.3 Contraintes

Un fichier de contraintes est un fichier qui indique aux outils de placement et de routage du fabricant du FPGA sur quelle patte doit venir quel signal. Le fichier de contraintes pour votre unité de multiplication est le suivant.

```
place instance PADDRESS1_pad        : P94 ;
place instance PADDRESS2_pad        : P93 ;
place instance PADDRESS3_pad        : P92 ;
place instance PADDRESS4_pad        : P91 ;
place instance PADDRESS5_pad        : P89 ;
place instance PADDRESS6_pad        : P88 ;
place instance PADDRESS7_pad        : P87 ;
place instance PADDRESS8_pad        : P86 ;
place instance PADDRESS9_pad        : P83 ;
place instance PADDRESS10_pad       : P82 ;
```

```
place instance PADDRESS11_pad       : P81 ;
place instance PADDRESS12_pad       : P80 ;
place instance PADDRESS13_pad       : P71 ;
place instance PADDRESS14_pad       : P70 ;
place instance PADDRESS15_pad       : P69 ;
place instance PADDRESS16_pad       : P68 ;
place instance PADDRESS17_pad       : P66 ;
place instance PBG_pad              : P164 ;
place instance PBGACK_pad           : P163 ;
place instance PBR_pad              : P167 ;
place instance PCLOCK_pad           : P4 ;
place instance PDATA0_pad           : P178 ;
place instance PDATA1_pad           : P179 ;
place instance PDATA2_pad           : P186 ;
place instance PDATA3_pad           : P187 ;
place instance PDATA4_pad           : P188 ;
place instance PDATA5_pad           : P189 ;
place instance PDATA6_pad           : P192 ;
place instance PDATA7_pad           : P193 ;
place instance PDATA8_pad           : P151 ;
place instance PDATA9_pad           : P147 ;
place instance PDATA10_pad          : P138 ;
place instance PDATA11_pad          : P132 ;
place instance PDATA12_pad          : P128 ;
place instance PDATA13_pad          : P122 ;
place instance PDATA14_pad          : P113 ;
place instance PDATA15_pad          : P109 ;
place instance PRESET_pad           : P21 ;
place instance PSELECT_pad          : P146 ;
place instance PSELECTHIGH_pad      : P169 ;
place instance PSELECTLOW_pad       : P170 ;
place instance PWRITE_pad           : P11 ;
```

Ce fichier vous est fourni, et de ce fait, il vous impose les noms des signaux à utiliser. Vos schémas doivent donc faire apparaître ces noms, sans le suffixe “_pad”, sur les fils ou les bus qui relient les symboles de type BUFFER (IBUF, OBUF, OBUFT ou BUFGP) aux symboles de type PAD (IPAD, OPAD ou IOPAD), d’où la lettre P ajoutée systématiquement au début de chaque nom. Ainsi par exemple, le signal de sélection de l’unité porte le nom PSELECT et le nom SELECT reste disponible pour nommer ce signal à l’intérieur de votre unité de multiplication (fig. 18).

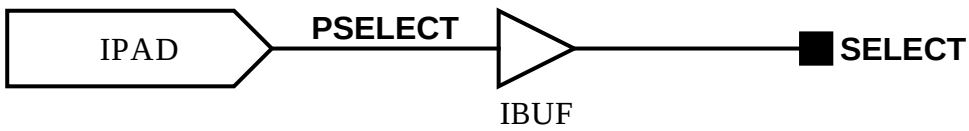


Fig. 18 Nom de signal