Compte-rendu TP n°1, MULTI, 2019-2020

Aymeric Agon-Rambosson

Mercredi 12 février 2020

Pour ce premier TP, on va d'intéresser au bus, qui est un bon exemple d'automate, et en particulier au Pi-bus, qui suit les règles suivantes :

Le pi-bus est un bus multimaître synchrone.

Deux types de composants matériels qui doivent communiquer entre eux :

- les maîtres sont par exemple les coeurs (et leurs caches privés)
- les cibles, comme la ram, la rom, le tty

La fonction du bus, c'est l'aiguillage et le routage.

On a avant toutes choses un bus de commandes, qui se décline en trois nappes de fils, ADDR (30 bits, puisqu'on ne ramène que des mots), OPC (4 bits), READ (1 bit).

Les seuls qui écrivent sur ce bus de commandes, ce sont les maîtres (et on peut d'ailleurs les définir comme ça).

On a un deuxième bus, le bus de réponses. Ce bus est unique, il s'appelle ACK (2 bits, car on a 3 réponses possibles READY, WAIT, ERROR) et seules les cibles peuvent écrire sur ce bus (on peut d'ailleurs les définir comme ça).

On a un bus de données qui est utilisé par les maîtres et les cibles (en effet, un maître veut pouvoir lire comme écrire).

On a un fil req et un fil gnt par maître qui arrive au BCU, bus control unit. (un des rarissimes automates non-Moore, donc Mealy général).

On peut spécifier un protocole du bus par son chronogramme :

- Phase d'allocation : la demande du bus et la réponse doivent avoir lieu dans le même cycle d'horloge.
- Phase d'adresse : les signaux Addr, lock, etc...
- Phase de donnée : D et ACK

L'allocation d'un nouveau maître peut se faire pendant le dernier (LOCK est à 0) cycle de la transaction n à condition que la valeur envoyée soit RDY ou ERROR.

Question C1

Noeud IDLE

F : C(SEL)
A : C(READ).DELAY.SEL.ADR_OK
B : C(READ).C(DELAY).SEL.ADR_OK
C : READ.C(DELAY).SEL.ADR_OK
D : READ.DELAY.SEL.ADR_OK
E : C(ADR_OK).SEL

On a l'orthogonalité sur ce noeud :

F d'intersection nulle avec tous les autres (car SEL.C(SEL) = 0).
On se permet donc de l'éliminer.

E est d'intersection nulle avec tous les autres (car ADR_OK.C(ADR_OK) = 0)
On se permet donc de l'éliminer.

A est d'intersection nulle avec C et D (car READ.C(READ) = 0)
B est d'intersection nulle avec C et D (car READ.C(READ) = 0)

A est d'intersection nulle avec B (car DELAY.C(DELAY) = 0)

C est d'intersection nulle avec A et B (car READ.C(READ) = 0)
D est d'intersection nulle avec A et B (car READ.C(READ) = 0)
On a montré que A et B étaient d'intersection nulle avec tous les autres, et entre eux.
On peut donc éliminer A et B.

C est d'intersection nulle avec D (car DELAY.C(DELAY)).
On a montré que C et D étaient d'intersection nulle avec tous les autres, et entre eux.
On peut donc éliminer C et D.

On a bien l'orthogonalité pour IDLE.

La complétude :

A + B = C(READ).DELAY.SEL.ADR_OK + C(READ).C(DELAY).SEL.ADR_OK
A + B = C(READ).SEL.ADR_OK

C + D = READ.C(DELAY).SEL.ADR_OK + READ.DELAY.SEL.ADR_OK
C + D = READ.SEL.ADR_OK

Donc :

A + B + C + D = C(READ).SEL.ADR_OK + READ.SEL.ADR_OK
A + B + C + D = SEL.ADR_OK

Donc :

A + B + C + D + E = SEL.ADR_OK + SEL.C(ADR_OK)
A + B + C + D + E = SEL

Donc :

A + B + C + D + E + F = SEL + C(SEL) = 1

On a bien la somme de toutes les conditions de transition égale à 1.

Noeud R_WAIT

V : GO
V' : C(GO)

L'orthogonalité et la complétude sont triviale.

Noeud W_WAIT

U : GO
U' : C(GO)

L'orthogonalité et la complétude sont triviale.

Noeud R_OK

R : C(ADR_OK).SEL
S : ADR_OK.SEL
T : C(SEL)

L'orthogonalité est évidente :

T.R = 0
T.S = 0
R.S = 0

La complétude :

R + S = C(ADR_OK).SEL + ADR_OK.SEL
R + S = SEL

R + S + T = SEL + C(SEL) = 1

On a bien la complétude.

Noeud W_OK

X : ADR_OK.SEL
Y : C(ADR_OK).SEL
Z : C(SEL)

La démonstration de l'orthogonalité et de la complétude est symétrique à celle du noeud R_OK.

Noeud ERROR

On n'a pas de transition qui nous fait rester dans l'état ERROR.

On admet donc qu'on va dans l'état IDLE de manière inconditionnée.

G : 1

Question C2

Noeud IDLE

Dans ces état, le matériel n'écrit pas sur le bus :

On a donc ACK_EN = 0
La valeur de ACK_VALUE n'a pas d'importance, on lui met NULL
On a DT_EN = 0 puisqu'on écrit pas sur le bus de données.
On n'écrit ni ne lit dans la mémoire, donc MEM_CMD est à NOP.

Noeud R_WAIT

Dans cet état, on écrit sur le bus ACK la valeur WAIT. On a donc besoin de mettre le signal ACK_VALUE à WAIT, et ACK_EN à 1.

On n'écrit pas sur le bus de données, puisque celles-ci ne sont pas encore prêtes (on envoie WAIT pour une raison). Donc DT_EN est à 0.

Le maître a demandé d'accéder à une zone de la mémoire en lecture, c'est donc la commande READ qu'on envoie vers la mémoire.

Noeud R_OK

Dans cet état, on écrit sur le bus ACK la valeur READY. On a donc besoin de mettre de signal ACK_VALUE à READY, et ACK_EN à 1.

On écrit sur le bus de données, puisque les données sont prêtes.

La commande à mettre dans MEM_CMD est ambigüe :

- Si on admet que la mémoire n'est pas capable d'obtenir une donnée dans le même cycle pendant lequel on lui a demandé, on n'aura jamais READ en sortie sur MEM_CMD pendant qu'on envoie la donnée sur le bus.

- Par contre, si on admet qu'elle en est capable, on a dans le même cycle READ sur MEM_CMD, puis dès la réception de la donnée (dans le même cycle donc) READY sur ACK_VALUE.

La valeur de ce fil en sortie dépend du paramètre L (NOP s'il est non nul, READ s'il est nul). (attention, on ne vient pas de transformer cette machine en machine de Mealy : L est un paramètre, choisi au démarrage de la machine, qui ne change plus ensuite.)

Noeud W_WAIT

Dans cet état, on écrit sur le bus ACK la valeur WAIT. On a donc besoin de mettre le signal ACK_VALUE à READY, et ACK_EN à 1.

On n'écrit pas sur le bus de données, on a reçu une instruction d'écriture, donc DT_EN est à 0.

Le maître a demandé d'accéder à une zone de la mémoire en écriture, c'est donc la commande WRITE qu'on envoie vers la mémoire.

Noeud W_OK

Dans cet état, on écrit sur le bus ACK la valeur READY. On a donc besoin de mettre de signal ACK_VALUE à READY, et ACK_EN à 1.

On n'écrit pas sur le bus de données, on a reçu une instruction d'écriture, donc DT_EN est à 0.

Pour la même raison que pour R_OK, la valeur de MEM_CMD dépendraa du paramètre L

Noeud ERROR

Dans cet état, on écrit sur le bus ACK pour signaler l'erreur.

On n'écrit pas sur le bus de données, on n'a rien à y écrire, ni aucune commande non plus à envoyer à la mémoire.

Résumé

	ACK_EN	ACK_VALUE	DT_EN	MEM_CMD
IDLE	0	NULL	0	NOP
R_WAIT	1	WAIT	0	READ
R_OK	1	READY	1	NOP (si !L), READ (si L)
W_WAIT	1	WAIT	0	WRITE
W_OK	1	READY	0	NOP (si !L), WRITE (si L)
ERROR	1	ERROR	0	NOP

A priori, on n'a écrit sur les bus ACK comme DT qu'au moment où on avait le droit de le faire. En effet, si on se rappelle le chronogramme du PIBUS, les différents états : R_WAIT R_OK W_WAIT W_OK ERROR n'arrivent qu'après une demande du maître. On a donc le bus réservé pour la réponse (donc aucun problème pour le bus ACK), et éventuellement le bus DT réservé si on a une donnée à transmettre.

Donc notre fonction de génération ne créé pas de court-circuits.

Question D1

Noeud INIT

On passe de INIT à RAM_REQ de manière inconditionnée.

Donc A = 1

Noeud RAM_REQ

On passe de RAM_REQ à RAM_A0 (demande de la première adresse) seulement si le bus est alloué. On a donc

B = GNT, et donc B' = C(GNT)

Noeud RAM_A0

On passe de RAM_A0 à RAM_A1_D0 (attente de la première donnée et demande de la deuxième adresse) de manière inconditionnée.

Même si la mémoire fait attendre le maître, on demande quand même la prochaine adresse.

Donc C = 1

Noeud RAM_A1_D0

Cet état signifie très exactement : **attente** de la première donnée et demande de la deuxième adresse.

La consigne précise :

> Dans le cas des transactions de type rafale, on utilise une technique de pipe-line, pour effectuer, dans le même cycle et sur deux nappes de fils séparées, le transfert de l'adresse (i+1), en même temps que le transfert de la donnée (i).

Moi, le maître, je n'ai le droit de demander l'adresse i+1 que si je reçois READY sur le bus ACK pour ma demande de l'adresse i.

Donc

D = RDY et D' = C(RDY)

Noeud RAM_A2_D1

De même :

E = RDY et E' = C(RDY)

Noeud RAM_A3_D2

De même :

F = RDY et F' = C(RDY)

Noeud RAM_D3

De même :

G = RDY et G' = C(RDY)

Noeud W_REQ

Dans cet état, on demande l'accès au bus pour écrire dans le tty.

On ne passe dans l'état W_AD que si le bus nous a été donné, donc :

H = GNT et H' = C(GNT)

Noeud W_AD

Dans cet état, on envoie une instruction d'écriture sur le bus, à destination du tty.

On ne demande rien dans cet état, on en sort de manière inconditionnée.

Donc I = 1

Noeud W_DT

Dans cet état, le maître envoie le caractère et doit vérifier :

- Que le caractère qu'il a transmis est bien arrivé et a bien été traité.
- Qu'il a bien envoyé tous les caractères.

S'il a envoyé tous les caractères, il doit passer dans sa boucle d'attente de saisie du clavier.

Sinon, il doit envoyer le caractère suivant. Et pour cela, il doit demander le bus.

Donc K = RDY.LAST et L = RDY.C(LAST) et J = C(RDY)

On a bien la complétude et l'orthogonalité.

Noeud STS_REQ

Dans cet état, le maître veut vérifier la valeur du registre status du tty pour savoir si quelqu'un a écrit dans le terminal.

Pour lire cette valeur, il doit obtenir le bus.

Donc M = GNT et M' = C(GNT)

Noeud STS_AD

Dans cet état, le maître envoie l'adresse du registre status du tty, il ne demande rien, il sort de cet état de manière inconditionnée.

Donc N = 1

Noeud STS_DT

Dans cet état, le maître doit vérifier :

- Qu'il a bien reçu le contenu du registre status du tty.
- Que la valeur de ce registre est bien non nulle, auquel cas il va demander de lire le registre keybuf du terminal.

Donc O = C(RDY) et P = RDY.C(NUL) et Q = RDY.NUL

On a bien la complétude et l'orthogonalité.

Noeud BUF_REQ

Dans cet état, le maître veut lire la valeur du registre keybuf du terminal.

Pour ça, il doit obtenir le bus.

Donc R = GNT et R' = C(GNT)

Noeud BUF_AD

Dans cet état, le maître envoie une demande en lecture vers le terminal, il n'attend rien.

On sort de cet état de manière inconditionnée.

Donc S = 1

Noeud BUF_DT

Dans cet état, le maître doit simplement attendre la donnée du registre keybuf du terminal.

Donc T = RDY et T' = C(RDY)

Question D2

Noeud INIT

Dans cet état, le maître ne demande rien, n'a pas le bus de commandes ni le bus de données.

On a donc C(REQ), C(CMD_EN), C(DT_EN).

Les signaux ADR_VALUE, READ_VALUE, et LOCK_VALUE ne sont pas applicables, ils ne sont pas envoyés.

Noeud RAM_REQ

Dans cet état, le maître demande le bus, mais il ne l'a pas encore. Il n'a pas le droit d'écrire, ni sur le bus de données, ni sur le bus de commandes.

On a donc REQ, C(CMD_EN), C(DT_EN).

Les signaux ADR_VALUE, READ_VALUE, et LOCK_VALUE ne sont toujours pas applicables, ils ne sont pas envoyés.

Noeud RAM_A0

Dans cet état, le maître a le bus, il demande l'adresse RAM_BASE.

Le signal REQ passe à 0, parce que la consigne spécifie que celui-ci est utilisé seulement pour demander le bus, pas pour le garder (c'est le signal LOCK_VALUE qui remplit ce rôle).

On envoie une demande, il faut donc autoriser l'émission sur le bus de commandes, donc le signal CMD_EN est activé.

Il s'agit d'une requête en lecture, donc READ_VALUE est à 1.

Il s'agit d'une demande rafale, et cette demande n'est pas la dernière de la rafale. On mettra donc le signal LOCK_VALUE.

On n'écrit pas sur le bus de données, donc DT_EN est à 0.

Noeud RAM_A1_D0

Dans cet état, le maître a le bus, il demande l'adresse RAM_BASE+4.

Le signal REQ passe à 0, parce que la consigne spécifie que celui-ci est utilisé seulement pour demander le bus, pas pour le garder (c'est le signal LOCK_VALUE qui remplit ce rôle).

On envoie une demande, il faut donc autoriser l'émission sur le bus de commandes, donc le signal CMD_EN est activé.

Il s'agit d'une requête en lecture, donc READ_VALUE est à 1.

Il s'agit d'une demande rafale, et cette demande n'est pas la dernière de la rafale. On mettra donc le signal LOCK_VALUE.

On n'écrit pas sur le bus de données, donc DT_EN est à 0.

Noeud RAM_A2_D1

Pareil que l'état précédent, on prendra bien garde à changer la valeur de la variable ADR_VALUE.

Noeud RAM_A3_D2

Pareil que l'état précédent, on prendra bien garde à changer la valeur de la variable ADR_VALUE.

On signalera aussi que la commande est la dernière de la rafale en mettant le signal LOCK_VALUE à 0.

Noeud RAM_D3

Dans cet état, le maître n'envoie plus de commandes, donc ADR_VALUE et READ_VALUE et LOCK_VALUE ne sont plus applicables, et CMD_EN est à 0.

On ne demande pas le bus, donc REQ est à 0.

On n'écrit pas sur le bus de données, donc DT_EN est à 0.

Noeud W_REQ

Dans cet état, le maître demande le bus, on met donc REQ à 1.

Il ne l'a pas reçu, donc il n'a pas le droit d'écrire sur le bus de commandes. Donc CMD_EN est à 0, et ADR_VALUE, READ_VALUE et LOCK_VALUE ne sont pas applicables.

On n'écrit pas sur le bus de données non plus, donc DT_EN est à 0.

Noeud W_AD

Dans cet état, le maître a obtenu le bus, il lance une requête d'écriture simple sur le bus à destination de l'adresse TTY_BASE.

On a donc REQ à 0, CMD_EN à 1, ADR_VALUE à TTY_BASE, READ_VALUE à 0, LOCK_VALUE à 0.

Ici, on a une ambigüité : le maître envoie-t-il les données dans cet état, ou dans l'état suivant ? Le modèle fourni par soclib semble pencher pour la deuxième option, même s'il n'est nulle part question de cycle de décalage entre la requête d'écriture et l'envoi effectif des données.

Le chronogramme (tp1_chronogramme.png) semble aussi pencher pour la deuxième option.

On met donc DT_EN à 0

Noeud W_DT

Dans cet état, le maître envoie les données du caractère sur le bus de données, on met donc DT_EN à 1.

Sinon, il n'envoie pas de commandes, ni ne demande le bus.

Noeud STS_REQ

Dans cet état, le maître veut obtenir le bus. On a donc REQ à 1.

Puisqu'il ne l'a pas, il n'écrit ni sur le bus de commandes, ni sur le bus de données.

Noeud STS_AD

Dans cet état, le maître envoie une requête en lecture simple vers l'adresse TTY_BASE+4. Il n'utilise pas le bus de données.

Noeud STS_DT

Dans cet état, le maître reçoit la réponse du terminal. Il ne demande pas le bus, il n'envoie pas de commandes, ni de données.

Noeud BUF_REQ

Dans cet état, le maître veut obtenir le bus. On donc REQ à 1.

Puisqu'il ne l'a pas, il n'écrit ni sur le bus de commandes, ni sur le bus de données.

Noeud BUF_AD

Dans cet état, le maître a obtenu le bus, il lance une requête simple en lecture vers l'adresse TTY_BASE+8. Il n'utilise pas le bus de données.

Noeud BUF_DT

Dans cet état, le maître attend la réponse du terminal. Il n'utilise aucun bus.

Résumé

	REQ	CMD_EN	ADR_VALUE	READ_VALUE	LOCK_VALUE	DT_EN
INIT	0	0	NULL	NULL	NULL	0
RAM_REQ	1	0	NULL	NULL	NULL	0
RAM_A0	0	1	RAM_BASE	1	1	0
RAM_A1_D0	0	1	RAM_BASE+4	1	1	0
RAM_A2_D1	0	1	RAM_BASE+8	1	1	0
RAM_A3_D2	0	1	RAM_BASE+12	1	0	0
RAM_D3	0	0	NULL	NULL	NULL	0
W_REQ	1	0	NULL	NULL	NULL	0
W_AD	0	1	TTY_BASE	0	0	0
W_DT	0	0	NULL	NULL	NULL	1
STS_REQ	1	0	NULL	NULL	NULL	0
STS_AD	0	1	TTY_BASE+4	1	0	0
STS_DT	0	0	NULL	NULL	NULL	0
BUF_REQ	1	0	NULL	NULL	NULL	0
BUF_AD	0	1	TTY_BASE+8	1	0	0
BUF_DT	0	0	NULL	NULL	NULL	0

On a tenu à distinguer NULL et 0 pour distinguer les cas où le signal était effectivement signifiant. Dans les fait, NULL peut prendre n'importe quelle valeur, le signal n'est pas transmis. On imagine qu'il prend la valeur 0.

Question E1

Noeud IDLE

On reste dans l'état IDLE seulement si personne ne demande le bus.

Donc X' = C(REQ) et X = REQ

On a de manière évidente l'orthogonalité et la complétude.

Noeud AD

Dans cet état, le bus a été alloué à un maître, et c'est la première commande.

La première commande est en même temps la dernière commande si, et seulement si, LOCK est à 0. Sinon, on est la première commande d'une rafale, et donc on va dans l'état DTAD.

Donc Y = LOCK et Y' = C(LOCK)

On a de manière évidente l'orthogonalité et la complétude.

Noeud DTAD

Dans cet état, le bus a été alloué à un maitre, et on est au milieu d'un transaction rafale : CMD(i) / RSP(i-1).

On ne sort de cet état que si :

- la commande envoyée est bien la dernière
- On a bien reçu soit READY, soit ERROR de la cible.

Donc Z = C(WAIT).C(LOCK) et Z' = WAIT + LOCK

On a de manière évidente l'orthogonalité et la complétude.

Noeud DT

On ne sort de cet état que si la réponse de la cible est READY ou ERROR.

Donc J = WAIT

On ne retourne dans l'état IDLE que si personne n'a demandé le bus.

Donc K = C(WAIT).C(REQ)

On ne retourne dans l'état AD que si quelqu'un a demandé le bus.

Donc L = C(WAIT).REQ

On a l'orthogonalité et la complétude par construction.

Question E2

Noeud IDLE

On se rappelle ici le fait que cet automate est un automate de Mealy. Le BCU est censé répondre à la requête du maître dans le même cycle.

Le signal GNT est donc activé à la suite de la réception du signal REQ.

Le maître n'a encore sélectionné personne, les signaux de sélection sont à 0.

Noeud AD

Dans cet état, le maître a sélectionné un maître et une cible. Dans ce PIBUS simplifié, on a seulement deux cibles possibles, la RAM et le tty.

On active bien entendu qu'un seul des deux signaux SEL0 et SEL1, sous peine de court-circuit.

On active SEL0 si les bits de poids fort correspondent à une adresse de la RAM, SEL1 sinon.

On n'a pas besoin d'activer le signal GNT, le bus a déjà été attribué.

Noeud AD/DT

Pareil qu'au noeud précédent.

Noeud DT

Dans ce noeud, le bus a été alloué à un maître, et c'est la réponse à la dernière commande.

On n'a pas de commande, donc SEL0 et SEL1 sont à 0.

Si la réponse de la cible est WAIT, on reste dans cet état, on continue à attendre.

Si la réponse de la cible n'est pas WAIT, c'est que le maître courant a fini sa demande (qu'elle soit valable ou non). On peut donc déjà attribuer le bus à un autre maître (ou au même), pour qu'il puisse faire sa commande au prochain cycle. Si le maître requiert le bus (signal REQ), on le lui donne (signal GNT).

Résumé

	GNT	SEL0	SEL1
IDLE	REQ	0	0
AD	0	DEC(A) == RAM	DEC(A) != RAM
DT/AD	0	DEC(A) == RAM	DEC(A) != RAM
DT	REQ.C(WAIT)	0	0

Question E3

Comme on l'a déjà expliqué, les contraintes se limitent strictement à :

- on doit laisser un cycle entre la commande d'un maître et la commande d'un autre maître
- le bcu doit répondre au maître dans le même cycle que la demande

Donc, lors de la réponse à la dernière commande d'un maître, la dernière fois que le maître en question a parlé, c'était au cycle précédent. On peut donc se permettre de faire parler un autre maître au cycle suivant. Pour pouvoir faire parler un maître au cycle suivant, il faut lui accorder le bus là maintenant.

Donc, on alloue le maître non seulement dans l'état IDLE, mais aussi dans l'état DT, pour gagner un cycle.

Question F1

Instanciation des deux matériels manquants :

```
PibusSimpleMaster     master  ("master", SEG_RAM_BASE, SEG_TTY_BASE);
PibusSimpleRam        ram ("ram"  , 0, segtable, ram_latency, loader);
```

Connexion des deux matériels manquants :

Commençons par la ram :

- on connecte le signal d'horloge, le signal de reset, et le signal de "tout" (on ne sait pas ce qu'il fait)

```
ram.p_ck(signal_ck);
ram.p_resetn(signal_resetn);
ram.p_tout(signal_pi_tout);
```

- on connecte le signal de sélection (SEL0) :

```
ram.p_sel(signal_sel_ram);
```

- On connecte le signal d'adresse, d'opcode, de read, de data, de ack :

```
ram.p_a(signal_pi_a);
ram.p_read(signal_pi_read);
ram.p_opc(signal_pi_opc);
ram.p_ack(signal_pi_ack);
ram.p_d(signal_pi_d);
```

Ensuite, le maître :

- On commence toujours par le signal d'horloge, de reset, et de tout :

```
master.p_ck(signal_ck);
master.p_resetn(signal_resetn);
master.p_tout(signal_pi_tout);
```

- On connecte le signal de req, de gnt, de lock :

```
master.p_req(signal_req_master);
master.p_gnt(signal_gnt_master);
master.p_lock(signal_pi_lock);
```

- On connecte le signal de addr, opc, read, data, et ack :

```
master.p_a(signal_pi_a);
master.p_opc(signal_pi_opc);
master.p_read(signal_pi_read);
master.p_d(signal_pi_d);
master.p_ack(signal_pi_ack);
```

Question F2

Ajout du segment du tty :

```
segtable.addSegment("seg_tty", SEG_TTY_BASE, 0x00000010, 1, false);
```

On lui donne le nom "seg_tty", on le fait commencer à SEG_TTY_BASE, on sait par la consigne qu'il occupe 16 octets (4 registres d'un mot chacun), l'identifiant de la cible tty est 1, et on désactive le cache.

Question F3

Comment est initialisée la chaîne de caractères "Hello World!" dans la mémoire ?

Le constructeur de l'objet PibusSimpleRam prend en dernier argument une référence vers un objet Loader.

L'objet Loader qu'on lui a passé en paramètre est loader, qu'on a instancié trois lignes plus haut :

Le constructeur de cet objet loader prend en paramètre une chaîne de caractères, probablement parsée :

- string_file : un cheminom de fichier qui contient les données à charger
- 0x10000000 : l'adresse à laquelle charger ces données
- D : un flag, apparemment

Question G1

En exécutant la commande :

```
time ./simul.x -NCYCLES 1000000
```

On obtient :

real 0m3.993s
user 0m1.578s
sys 0m1.146s

Donc 1000000 de cycles simulés en 4 secondes, soient 250000 cycles par secondes, soit 250 kHz.

La condition selon laquelle SystemC ne doit pas être pire que 1000 fois moins rapide que le matériel est à peu près tenue, si on suppose un matériel simulé à 250 MHz.

Question G2

Combien y-a-t-il de cycles d'attente dans les états de l'automate du composant maître où celui-ci demande au BCU l'allocation du bus ? Expliquez ce comportement.

On admet que la question signifie : combien y a-t-il de cycles d'attente entre la demande du bus par le maître et son allocation ?

Il n'y a aucun cycle d'attente entre la demande du bus et son allocation : on a l'activation du signal REQ, qui demande le bus, et du signal GNT, qui l'accorde, dans les mêmes cycles.

Ce comportement est voulu. On veut pouvoir répondre au maître tout de suite. Cette chose est rendus possible par le fait que le BCU est un automate de Mealy : son signal GNT peut être activé de manière asynchrone aux fronts d'horloge.

Question G3

Combien y-a-t-il de cycles d'attente dans les états de l'automate du composant maître ou celui-ci attend la réponse de la RAM ? Expliquez ce comportement.

On admet que la question signifie :

Combien faut-il de cycles entre l'arrivée de la demande du maître à la RAM et le passage de la RAM dans l'état READ_OK (qui signifie que la RAM envoie les données demandées sur le bus) ?

Avec les paramètres qu'on a choisi, on a deux cycles d'attente.

Regardons pour cela les cycles 1, 2, 3 et 4.

(Pour une raison inexpliquée, le maître ne commence pas dans l'état INIT, mais directement dans l'état RAM_A0.)

——- cycle = 1 ——-
bcu : fsm = AD | selected target = 0
master : state = RAM_A0
ram : IDLE
tty : IDLE keyboard status[0] = 0 display status[0] = 0
req = 0
gnt = 0
sel_ram = 1
sel_tty = 0
avalid = 1
read = 1
lock = 1
address = 0x10000000
ack = 0
data = 0
——- cycle = 2 ——-
bcu : fsm = DTAD | selected target = 0
master : state = RAM_A1_D0
ram : READ_WAIT
tty : IDLE keyboard status[0] = 0 display status[0] = 0
req = 0
gnt = 0
sel_ram = 1
sel_tty = 0
avalid = 1
read = 1
lock = 1
address = 0x10000004
ack = 0
data = 0
——- cycle = 3 ——-
bcu : fsm = DTAD | selected target = 0
master : state = RAM_A1_D0
ram : READ_WAIT
tty : IDLE keyboard status[0] = 0 display status[0] = 0
req = 0
gnt = 0
sel_ram = 1
sel_tty = 0

avalid = 1
read = 1
lock = 1
address = 0x10000004
ack = 0
data = 0
——- cycle = 4 ——-
bcu : fsm = DTAD | selected target = 0
master : state = RAM_A1_D0
ram : READ_OK
tty : IDLE keyboard status[0] = 0 display status[0] = 0
req = 0
gnt = 0
sel_ram = 1
sel_tty = 0
avalid = 1
read = 1
lock = 1
address = 0x10000004
ack = 0x2
data = 0x6c6c6548

Le cycle 1 est la demande du maître. Celle-ci arrive à la fin du cycle 1, la RAM sort de l'état IDLE avec le front montant du cycle 2.

La RAM reste dans l'état READ_WAIT jusqu'à la fin du cycle 3, soient deux cycles entiers.

On a donc deux cycles d'attente pour la première demande, ce qui correspond bien au paramètre ram_latency qu'on a choisi. (pas d'attente en revanche pour les demandes suivantes de la rafale, ce qui correspond à notre modélisation de la RAM)

Question G4

On admet que la question signifie :

Combien faut-il de cycles, **depuis un état initial INIT du maître**, pour afficher un caractère sur le composant PIBUS_MULTI_TTY ?

On commence donc par se placer dans le premier état INIT du maître (dont l'index du cycle n'est pas déterministe, il dépend en fait du temps qu'on a fait tourner le maître à attendre notre input au clavier).

Il se trouve que dans ma trace, ce premier état initial INIT est au cycle 9389. On recherche donc à partir de ce cycle-là le premier état DISPLAY du TTY, avec 0x48 ('H') en data.

C'est le cycle 9400.

Il faut donc 9400 - 9389 = 11 cycles pour afficher un caractère dans le composant PIBUS_MULTI_TTY depuis l'état inital INIT du maître, étant donné les valeurs de latence de la RAM données en paramètres (soit 2 dans notre cas, variable ram_latency que nous n'avons pas modifié).

Question G5

Comme on l'a déjà dit, le maître commence directement dans l'état RAM_A0 pour une raison inexpliquée.

Le chronogramme tiendra compte de cet état de choses, et commencera aussi dans cet état-là.

On joint en annexe un chronogramme des cycles 1 à 20 compris.

On s'est permis de spécifier à certains endroits un signal XXX pour "don't care". On voulait en fait faire remarquer que personne n'écrivait à ce moment-là. Laisser la valeur résiduelle des cycles précédents aurait fonctionné aussi,

mais on aurait eu plus de mal à faire la différence entre un signal effectivement envoyé et un signal résiduel dont il ne faut pas tenir compte.