Date : 10 octobre 2008

École Polytechnique de Montréal

INF3610: Systèmes embarqués

Questionnaire de l'examen intra : Automne 2008

PRENEZ NOTE DES DIRECTIVES SUIVANTES :

- Documentation non permise
- **Vous devez remettre la page 5 de l'examen**
- Durée de l'examen : 110 minutes i.e. 9 :30 à 11 :20
- Ne pas écrire en rouge
- L'utilisation de la calculatrice non programmable est permise
- L'examen comporte 5 questions pour un total de 20 points et il compte pour 20% de la note finale.

Automne A2008 **Guy Bois, coordonnateur et professeur**

Question 1 (5 points) Programmation avec la librairie SystemC

a) **(.5 pt)** Pourquoi est-il toujours préférable, lorsque cela est possible, d'utilisez le SC_METHOD plutôt que le SC_THREAD ou même le SC_CTHREAD.

C'est d'abord et avant tout une question de performance (vitesse d'exécution). En effet un SC_METHOD c'est comme une fonction C/C++ alors qu'un SC_THREAD ou SC_CTHREAD c'est implémenté à partir d'une librairie de thread POSIX (Linux) ou encore FIBERS pour Windows. Or à cause de son implémentation dans le noyau du OS, le thread est beaucoup moins rapide que la fonction en temps d'exécution.

b) **(.5 pt)** Pour quel type de lecture n'a-t-on pas le choix que d'utiliser le SC_THREAD. Expliquez

Lecture bloquante

c) **(1 pt)** Quelle(s) différence(s) existent-ils entre les 2 modèles suivants (Fig. 1.1) du point de vue comportement. Expliquez.

```
#include "systemc.h"

SC_MODULE (code1) {
  sc_in    <bool> data, clk, reset  ;
  sc_out   <bool> q;

  bool q_l ;

  void tff () {
    if (reset.read()) {
      q_l = 0;
    } else if (data.read()) {
      q_l = !q_l;
    }
    q.write(q_l);
  }

  SC_CTOR(code1) {
    SC_METHOD (tff);
      sensitive << clk.pos();
  }

};
```

```
#include "systemc.h"

SC_MODULE (code2) {
  sc_in    <bool> data, clk, reset  ;
  sc_out   <bool> q;

  bool q_l ;

  void tff () {
    if (reset.read()) {
      q_l = 0;
    } else if (data.read()) {
      q_l = !q_l;
    }
    q.write(q_l);
  }

  SC_CTOR(code2) {
    SC_METHOD (tff);
      sensitive << reset;
      sensitive << clk.pos();
  }

};
```

Figure 1.1

À gauche on a une bascule T dont le signal reset est synchrone avec l'horloge alors qu'à droite on n'a une bascule T dont le signal reset est asynchrone.

d) **(3 pts)** En temps qu'ingénieur en vérification, on vous donne le code qui modélise l'algorithme d'Euclid (plus grand diviseur commun) en SystemC ainsi que le circuit test (*tesbench*) qui l'accompagne (fig. 1.2 et 1.3 resp.). Toutefois, il manque la synchronisation entre ces deux modèles qui est la suivante : 1) les donnée sont lus pour un traitement entre les fronts montants des périodes *i-1* et *i*, 2) le traitement proprement dit est réalisé entre les fronts montants des périodes *i* et *i+1* et 3) le résultat est disponible quelque part entre les fronts montants des périodes *i+1* et *i+2*, puis ainsi de suite (voir la bouche *while* de la fig. 1.2). Le *testbench* génère lui-même le signal d'horloge. Vous devez donc compléter les 6 morceaux de code des figures 1.2 et 1.3 pour 3 jeux de données. Le bloc1 concerne la lecture et la synchronisation, le bloc2 concerne l'écriture et la synchronisation, les blocs 3 à 5 la synchronisation seulement et le bloc 6 la génération de l'horloge. Considérez une période d'horloge de 100 ns avec un rapport cyclique de 50% (50% du temps à 1, 50% du temps à 0). Répondez dans votre cahier de réponse en indiquant bien le bloc auquel correspond chacun des morceaux de code que vous donnez.

Voir page suivante pour la solution

```
#include "systemc.h"


SC_MODULE(euclid_gcd) {
    sc_in<bool>   CLOCK;
    sc_in<int>    A, B;
    sc_out<int>   C;

    void compute();
    SC_CTOR(euclid_gcd) {
        SC_THREAD(compute);
        sensitive << CLK;
    }
};

void
euclid_gcd::compute()
{
    // reset section
    unsigned tmp_a, tmp_b;  // le résultat est dans tmp_a

    while (true) {      // main loop


        // On lit les données et synchronisation
        Bloc 1 à compléter
        wait(clk.posedge_event()) ;  //cycle de lecture
        Tmp.a = A. read() ;
        Tmp.b = B. read() ;
        wait(clk.posedge_event()) ;  //cycle de traitement

        // Calcul du GCD.
        while (tmp_b != 0) {  // Euclid's algorithm
            unsigned r = tmp_a;
            tmp_a = tmp_b;
            while (r >= tmp_b) {
                r = r - tmp_b;
            }
            tmp_b = r;
        }

        // Envoie du résultat et synchonisation
        Bloc 2 à compléter
        wait(clk.posedge_event()) ;  //cycle d'écriture
        c.write(temp_a) ;
    }
}
```

Figure 1.1 Code du plus grand diviseur commun

```
#include "systemc.h"
#include "euclid.h"

SC_MODULE (generateur) {
  sc_out<bool>  CLK;
  sc_out<int>   A,B;
  sc_in<int>    C;
  void genere();
  void affiche();
  void horloge();

  SC_CTOR(generateur) {
    SC_METHOD(affiche)
    sensitive << C;
    SC_THREAD(genere);
    SC_THREAD(horloge)
};

  void generateur::genere() {

// 1er jeu de données à synchroniser
    Bloc 3 à compléter
    Wait(10,SC_NS) ;  // où n'importe quelle valeurs entre 0 et 50ns
    A = 40;
    B = 64;

// 2e jeu de données à synchroniser
    Bloc 4 à compléter
    Wait(10,SC_NS) ;  // où n'importe quelle valeurs entre 100 et 150ns
    A = 80;
    B = 30;

// 3e jeu de données à synchroniser
    Bloc 5 à compléter
    Wait(210,SC_NS) ;  // où n'importe quelle valeurs entre 200 et 250ns
    A = 8;
    B = 4;
};

// Génération de l'horloge
  void generateur::horloge(){
    while (1) {

    Bloc 6 à compléter

    Clk.write(false) ;
    Wait(50,SC_NS) ;
    Clk.write(true) ;
    Wait(50,SC_NS) ;

};

  void generateur::affiche()
{
       printf(" %d\n",(int)C.read());
}

int sc_main (int argc , char *argv[]) {
  sc_signal<int>  A, B, C;
  sc_signal<bool>  CLK;
  generateur gen("gen");
  euclid_gcd gcd("gcd");

  gen.A(A);
  gen.B(B);
  gen.C(C);
  gen.CLOCK(CLK);
  gcd.A(A);
  gcd.B(B);
```

```
  gcd.C(C);
  gcd.CLK(CLK);
  sc_start(200, SC_NS);
  return 0;
}
```

Figure 1.2 Testbench du plus grand diviseur commun

Question 2 (4 points) Pipeline et parallélisme d'instructions

Soit une boucle qui réalise l'opération vectorielle Y=a*X+Y/b pour un vecteur de longueur de 100 selon le jeu d'instructions du tableau 2.1:

```
L:      LD      F2,  0(R1)
        MULTF   F4,  F2, F0
        LD      F6,  0(R2)
        DIVF    F8,  F6, F1
        ADDF    F8,  F8, F4
        SD      0(R2), F8
        ADDI    R1, R1, 8
        ADDI    R2, R2, 8
        STGI    R3, R1, fait
        BEQZ    R3, L
```

où F0 = a, F1=b et fait = 100

Instructions pipelinées	Signification	Cycle du pipeline où l'opération termine (le résultat étant disponible 1 cycle plus tard)
LD F0, 0(R3)	**F0 ← Mem[0 + R3]**	**ER**
ADDF F8, F6, F1	**F8 ← F6 + F1**	**EX2 i.e. après 2 cycles dans EX (le résultat est dans F8)**
MULTF F4, F2, F0	**F4 ← F2 * F0**	**EX3 i.e. après 3cycles dans EX (le résultat est dans F4)**
DIVF F8, F6, F1	**F8 ← F6 + F1**	**EX4 i.e. après 4 cycles dans EX (le résultat est dans F8)**
ADDI R1, R1, 8	**R1 ← R1 + 8**	**EX (le résultat est dans R1)**
SD 0(R2), F8	**Mem[0 + R2] ← F8**	**MEM**
STGI R3, R1, fait	**si R1 > fait alors R3=0**	**EX**
BEQZ R3, etiq	**Branchement si zéro**	**EX**

Tableau 2.1

a) **(2 pts)** Complétez la trace de ce programme (Tableau 2.2, page 5) pour l'exécution d'une seule boucle pour le pipeline DLX à 5 étages avec les unités points flottants pipelinées. Donnez le nombre de cycles pour une itération.

b) **(2 pts)** Déroulez le code donné en a) autant de fois que nécessaire pour l'ordonnancer sans aucune suspension, en déplaçant du code au besoin et en diminuant les instructions de gestion de boucle, le tout en évitant les aléas. Montrez l'ordonnancement et le gain par rapport à a).

Voir solution page 9

##		Instruction/Cyc	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26
1	L:	LD F2, 0(R1)	*LI*	*DI*	*EX*	*ME*	*ER*																					
2		MULTF F4,F2,F0		*LI*	*DI*	*SU*	*SU*	*E1*	*E2*	*E3*																		
3		LD F6,0(R2)			*LI*	*SU*	*SU*	*DI*	*SU*	*SU*	*EX*	*ME*	*ER*															
4		DIVF F8,F6,F1						*LI*	*SU*	*SU*	*DI*	*SU*	*E1*	*E2*	*E3*	*E4*												
5		ADDF F8, F8, F4									*LI*	*SU*	*DI*	*SU*	*SU*	*SU*	*E1*	*E2*										
6		SD 0(R2),F8											*LI*	*SU*	*SU*	*SU*	*DI*	*EX*	*M*	*ER*								
7		ADDI R1, R1, 8															*LI*	*DI*	*EX*									
8		ADDI R2, R2, 8																*LI*	*DI*	*EX*								
9		STGI R3,R1,fait																	*LI*	*DI*	*EX*							
10		BEQZ R3, L																		*LI*	*DI*	*EX*						

Tableau 2.2 À compléter pour la question a)
(à remettre avec votre cahier de réponse)

b)

```
L:      LD F20, 0(R1)
        LD F21, 0(R1+8)
        LD F22, 0(R1+16)
        LD F23, 0(R1+24)

        MULTF   F40, F20, F0
        MULTF   F41, F21, F0
        MULTF   F42, F22, F0
        MULTF   F43, F23, F0

        LD F60, 0(R2)
        LD F61, 0(R2+8)
        LD F62, 0(R2+16)
        LD F63, 0(R2+24)

        DIVF F80, F60, F1
        DIVF F81, F61, F1
        DIVF F82, F62, F1
        DIVF F83, F63, F1

        ADDF F80, F80, F40
        ADDF F81, F81, F41
        ADDF F82, F82, F42
        ADDF F83, F83, F43

        SD 0(R2), F80,
        SD 0(R2+8), F81,
        SD 0(R2+16), F82,
        SD 0(R2+24), F83,

        STGI R3,R1,fait
        BEQZ R3, L
        ADDI R2, R2, 32
        ADDI R1, R1, 32
```

*En a) on a 20 cycles * 100 = 2000 cycles au total*
*En b) on a 28 cycles * 25 = 700 cycles au total*
Donc, en déroulant la boucle on obtient une accélération de 2000/700 ≅ 2.9

Question 3 (3 points) Architectures embarqués et application du MP3

a) **(.5 pt)** Qu'est-ce que le ID3 ?

En-tête ne faisant pas parti de la norme MP3 mais qui est souvent rajouté au fichier MP3 pour fournir des informations divers comme le nom d'une chanson, de l'auteur ou de l'album. Ces informations ne peuvent être prises en charge par le format normalisé ISO MP3 lui-même.

b) **(.5 pt)** Quelle est l'utilité d'un bloc court («short block») par rapport à un bloc long («long block») dans le traitement MP3 ?

Les blocs courts sont utilisés pour isoler les transitions brusques d'un signal sonore alors que les blocs longs le sont pour des sons stables.

c) **(.5 pt)** Que permet de faire le filtre de synthèse durant la dernière grande étape du décodage MP3 ?

Regrouper en un seul signal final de sortie les 32 signaux provenant des 32 sous-bandes de fréquence de départ.

d) **(.5 pt)** Nommez les deux grandes catégories de masquage de la psycho-acoustique.

Masquage temporel et masquage fréquentiel.

e) **(1 pt)** Pour chacune des étapes de décodage, donnez le type de calcul (en vous référant au trois types d'application du chapitre 1). À quelle colonne de la figure 3.2 appartient donc selon vous l'application MP3?

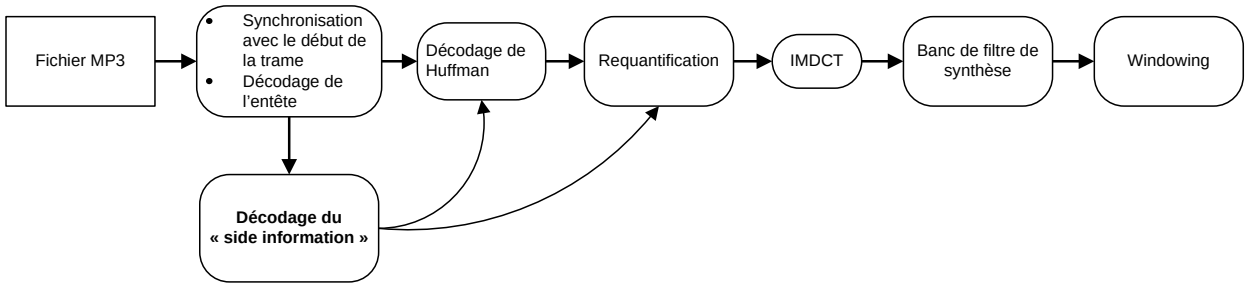


Figure 3.1 Étapes de décodage

Les 3 blocs de gauche (Synchronisation, Huffman et Requantification) sont plutôt orientés contrôle alors que IMDCT, Synthèse et Windowing sont orientés flot de données. Aussi, j'ai accepté les 3, 4 et 5 (à partir de gauche) pour la figure 3.2 (page suivante) , mais en réalité la 5 serait la bonne si on veut maximiser l'utilisation du processeur.

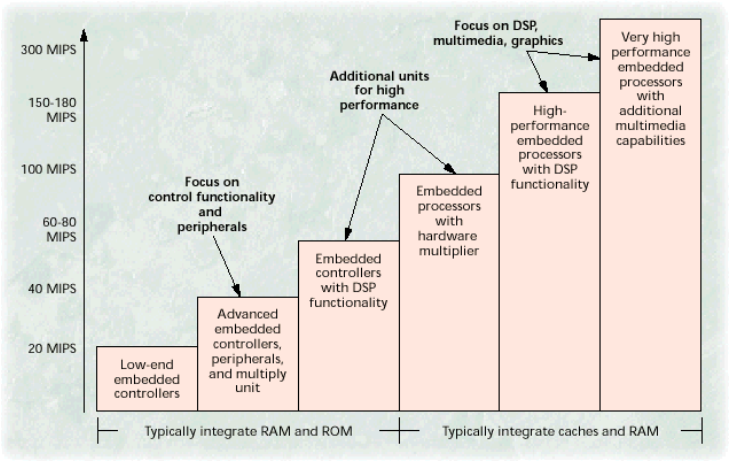
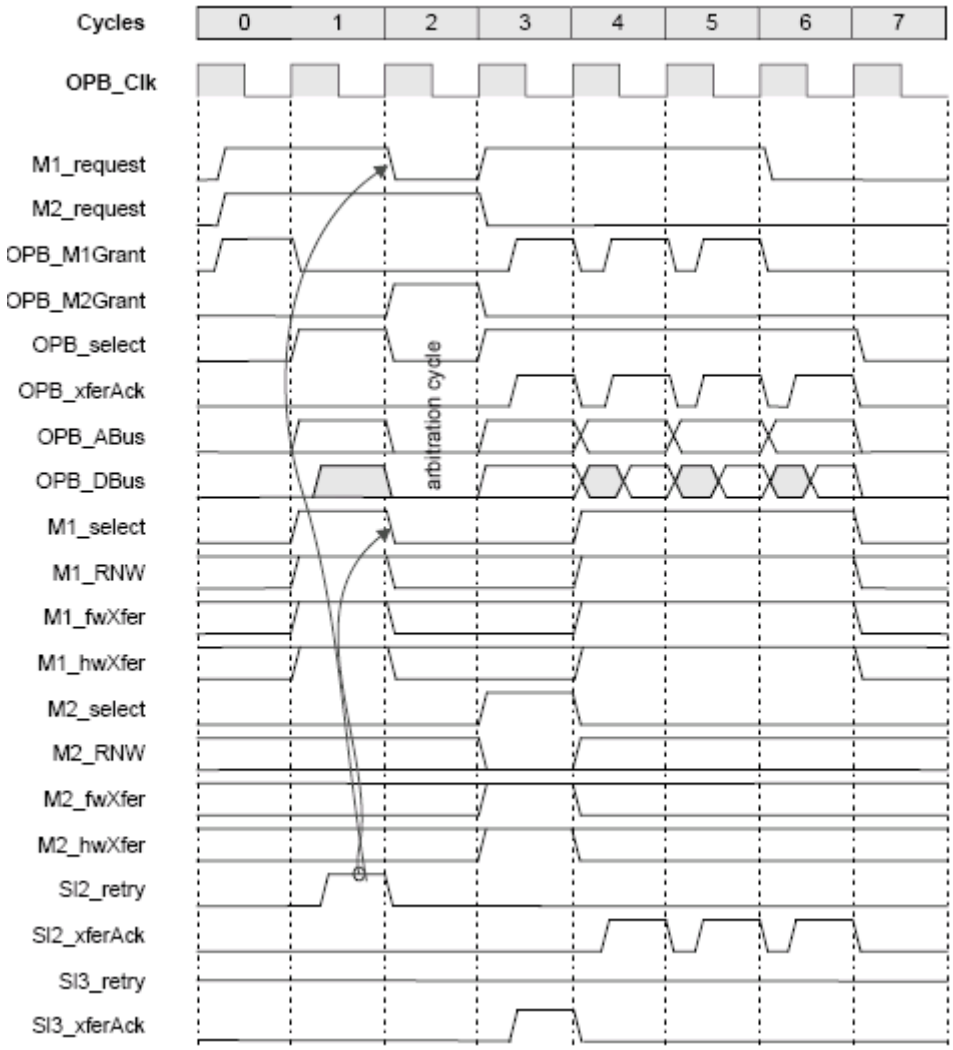

300 MIPS
150-180 MIPS
100 MIPS
60-80 MIPS
40 MIPS
20 MIPS
Low-end embedded controllers
Advanced embedded controllers, peripherals, and multiply unit
Embedded controllers with DSP functionality
Embedded processors with hardware multiplier
High-performance embedded processors with DSP functionality
Very high performance embedded processors with additional multimedia capabilities
Focus on control functionality and peripherals
Additional units for high performance
Focus on DSP, multimedia, graphics
Typically integrate RAM and ROM
Typically integrate caches and RAM

Figure 3.2

Question 4 (3 points) Protocole OPB

Soit le diagramme temporel suivant que nous avons vu en classe, expliquez cycle par cycle le comportement auquel correspond ce diagramme correspond. (Attention : un périphérique peut être à la fois maître ou esclave).



Cycle 0 : *M1 et M2 demandent chacun le bus pour effectuer une requête à un esclave (requêtes à S2 et S3 respectivement).*

Cycle 1 : *M2 maintient sa demande mais M1 obtient le droit car il a une priorité plus élevé que M2. Par contre, comme M2 est occupé (à demandé) il ne peut servir tout de suite en tant qu'esclave S2, et il indique donc à M1 qu'il ne peut répondre tout de suite (en envoyant un signal retry).*

Cycle 2 : *M1 se retire un cycle et M2 obtient alors le bus pour effectuer sa requête avec S3.*

Cycle 3 : *M2 effectue sa requête à S3 (écriture) et M1 redemande pour le cycle suivant.*

Cycle 4 à 6 : *M1 obtient le bus et effectue sa requête (lecture) à S2 (maintenant disponible)*

Cycle 7 : *Terminé*

Question 5 (3 points) SystemC OPB (laboratoire 1)

Soit la situation suivante rencontrée par une équipe du laboratoire :

Bien qu'il soit indiqué que le ThermoMatrix nécessite 4 cycles afin de retourner un pixel, à la suite de divers tests sur puce FPGA, cette équipe se rend compte que les modules ne traitent pas tous les pixels (affichage erroné). En effet, seulement certains pixels sont traités tandis que d'autres sont totalement ignorés.

Après une analyse plus poussée, cette même équipe remarque que le ThermoMatrix peut requérir jusqu'à 26 cycles. Il remarque également que cette latence est principalement due au type de mémoire utilisée et que la latence de la mémoire augmente avec la charge sur le ThermoMatrix.

Il est alors impératif d'apporter des modifications au modèle haut niveau afin de corriger le problème de l'affichage erroné. Après avoir bien réfléchi à une solution, un des membres de l'équipe procède aux modifications du module ThermoMatrix.

a) (.5 point) Expliquez le problème que l'équipe a rencontré.

Lorsque le ThermoMatrix reçoit une certaine charge, le module prend au-delà de 16 cycles pour répondre. Ceci a pour effet de lever le signal ToutSup (valeur perdus). Puisque les modules ne sont pas configurés pour supporter ce cas, les pixels ne sont pas traités.

b) (.5 point) Expliquez quelle stratégie, à l'intérieur du ThermoMatrix, il est possible d'utiliser pour régler ce problème (astuce: utiliser les signaux du bus OPB).

Dans le ThermoMatrix, lorsqu'une requête va prendre au-delà des 16 cycles permis par le bus OPB, le module doit lever le signal RETRY afin que le maitre demande à nouveau le pixel.

c) (1 point) Pour que le correctif en b) soit effectif, un peu de travail s'impose. Modifier le module *Moyenne* de la Figure 5.1 (page suivante) afin de respecter votre stratégie décrite en b). Pour cela indiquez la (ou les) ligne(s) où vous ajouter votre code (exemple : après la ligne 9 s'ajoute le code suivant : …).
Voir page suivante

d) (1 point) Finalement, lors des essais en laboratoire, l'équipe s'aperçoit que les esclaves fonctionnent avec un bus de 8 bits de large et non 32 bits. On remarque également que les températures ne dépassent jamais 140°C et que les esclaves utilisent les bits les moins significatifs du bus OPB. Expliquez quels changements sont nécessaires dans le module *Moyenne*. De plus, réalisez ces changements.

BE[0] = true, BE[1] = false, BE[2] = false et BE[3] = false

```
1  {
2          // préparation des paramètres de requête de bus
3          m_opb_request.master_priority = m_master_priority;
4          m_opb_request.request = true;
5
6          // demande de la requête sur le bus
7          while (opb_bus_port->request_bus(&m_opb_request) != OPB_STATUS_BUS_GRANTED
8  )
9          {
10                 wait(opb_clock_port.posedge_event());
11         }
12
13         // préparation des paramètres de transaction et propagation des options
14         m_opb_request.request = false;
15         m_opb_request.select = true;
16         m_opb_request.seq_address = false;
17         m_opb_request.buslock = false;
18         m_opb_request.read_write = OPB_READ;
19         m_opb_request.xfer_size = OPB_XFER_FW;
20         m_opb_request.BE[0] = true;
21         m_opb_request.BE[1] = true;
22         m_opb_request.BE[2] = true;
23         m_opb_request.BE[3] = true;
24         opb_bus_port->set_options(&m_opb_request);
25
26         // émettre la requête sur le bus
27         m_opb_payload.address = address;
28         OPB_STATUS mystatus = opb_bus_port->put_request(&m_opb_request,
29         &m_opb_payload);
30
31         // Traitement de l'acquittement
32         switch(mystatus)
33         {
34                 case OPB_STATUS_SUCCESS:
35                         if (m_opb_request.ack.sl_xferAck == true)
36                                 {
37                                 *(unsigned int*)Data32 = (unsigned int)
38                                 m_opb_payload.data;
39                                 bLecture = false;
40                         }
41                         break;
42                 case OPB_STATUS_RETRY:
43                         wait(clockPort.posedge_event());
44                         break;
45         }
46 }
47
48
```

Figure 5.1 Code du module Moyenne à compléter

Question 6 (2 points) Niveaux d'abstraction

Soit le code d'un petit modèle transactionnel SystemC donné à la figure 6.2 des deux pages suivantes. Expliquez où se situent sur la figure 5.1 ce modèle SystemC. Notez que *_bus_mutex.lock()* et *_bus_mutex.unlock()* permettent d'entrer et de sortie d'une section critique. Justifiez bien votre choix en fonction des deux axes.

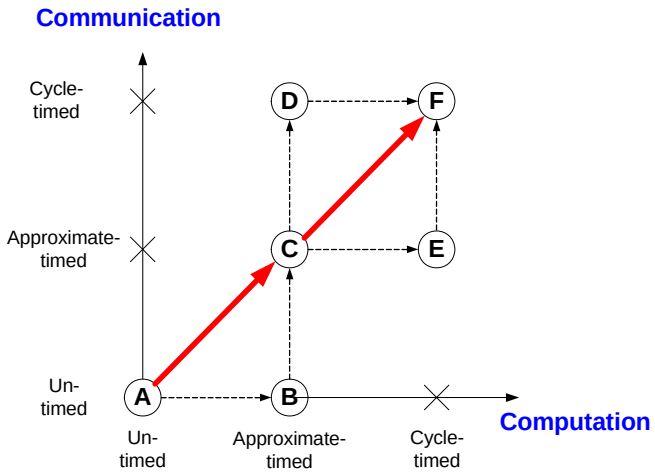


Figure 5.1 Classification des niveaux d'abstraction

Tout d'abord, ce numéro est passé de 2 points à 1 point. L'examen est donc sur 19 points plutôt que 20 points. La raison est qu'il n'y a pas de composante calcul (axe en X) dans se code. En effet, il n'y a que de la communication et ce type de communication est ***Approximated Timed****. Je n'ai donc tenu compte que de votre réponse en fonction de l'axe Y.*

```
#include <systemc.h>

class very_simple_bus_if: virtual public sc_interface
{
public:
  virtual void burst_read (char *data,
                           unsigned addr,
                           unsigned length) = 0;
  virtual void burst_write(char *data,
                           unsigned addr,
                           unsigned length) = 0;
};

class very_simple_bus
  : public very_simple_bus_if, public sc_channel
{
public:
  very_simple_bus(sc_module_name nm, unsigned mem_size,
     sc_time cycle_time) : sc_channel(nm), _cycle_time(cycle_time)
  {
    // we model bus memory accesses using an embedded memory array
    _mem = new char [mem_size];

    // set initial value of memory to zero
    memset(_mem, 0, mem_size);
  }

  ~very_simple_bus() { delete [] _mem; }

  virtual void burst_read(char *data, unsigned addr,
                          unsigned length)
  {
    // model bus contention using a mutex, but no arbitration rules
    _bus_mutex.lock();

    cout << "read starts at: " << sc_time_stamp() << endl;

    // block the caller for length of burst transaction
    wait(length * _cycle_time);

    //  copy the data from memory to requestor
    memcpy(data, _mem + addr, length);

    cout << "read ends at: " << sc_time_stamp() << endl;

    // unlock the mutex to allow others access to the bus
    _bus_mutex.unlock();
   }

  virtual void burst_write(char *data, unsigned addr,
                           unsigned length)
  {
    _bus_mutex.lock();

    cout << "write starts at: " << sc_time_stamp() << endl;

    wait(length * _cycle_time);

    //  copy the data from requestor to the memory
    memcpy(_mem + addr, data, length);

    cout << "write ends at: " << sc_time_stamp() << endl;

    _bus_mutex.unlock();
  }
```

```
protected:
  char* _mem;
  sc_time _cycle_time;
  sc_mutex _bus_mutex;
};

class writer : public sc_module
{
public:
  sc_port<very_simple_bus_if> bus_port;

  SC_HAS_PROCESS(writer);

  writer(sc_module_name nm) : sc_module(nm)
  {
    SC_THREAD(main);
  }

  void main()
  {
    char data[10];

    memset(data, 3, 10);

    while (true)
    {
        // every 50 NS (= 40 + 10) we cause some contention to occur...

       wait(40, SC_NS);
       bus_port->burst_write(data, 0, 10);
    }
  }
};

class reader : public sc_module
{
public:
  sc_port<very_simple_bus_if> bus_port;

  SC_HAS_PROCESS(reader);

  reader(sc_module_name nm) : sc_module(nm)
  {
    SC_THREAD(main);
  }

  void main()
  {
    char data[10];

    while (true)
    {
       bus_port->burst_read(data, 0, 5);
       wait(5, SC_NS);
    }
  }
};

int sc_main (int argc , char *argv[])
{
  very_simple_bus b("bus", 100, sc_time(1, SC_NS));
  reader r("reader");
  r.bus_port(b);
  writer w("writer");
  w.bus_port(b);

  sc_start(200, SC_NS);
  return 0;
}
```

Figure 6.2 Very simple bus

Annexe

Modèle transactionnel OPB

– paquetage OPB (opb_request.h)

- L'utilisateur travaille avec des classes (modules SystemC) et des structures de données

```
OPB_TRANSACTION_REQUEST
{
  unsigned char          master_priority
  bool                   request;
  bool                   grant;
  bool                   buslock;
  bool                   pending;
  bool                   select;
  bool                   seq_address;
  OPB_REQ_TYPE           read_write;
  OPB_XFER_SIZE          xfer_size;
  bool                   BE[4];

  OPB_TRANSACTION_ACK ack;
  bool                   opb_timeout;

  OPB_STATUS             status;
   bool                  nb_req;
  sc_event               blocking_ev;
};
```

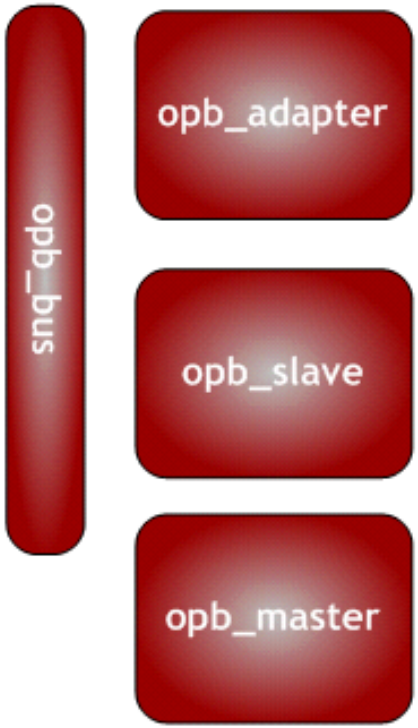
```
struct OPB_TRANSACTION_PAYLOAD
{
  unsigned long          data;
  unsigned long          address;
};
```

```
OPB_TRANSACTION_ACK
{
  bool          sl_BEack;

  bool          sl_xferAck;

  OPB_XFER_SIZE          slave_size;
  bool                   sl_retry;
  bool                   sl_toutSup;

  sc_event               sl_toutevent;
};
```



(30:31)

			32-bit Data Bus			
ABus ~~(28:31)~~	Mn_BE (0:3)	Request Transfer Size	Dbus 0:7 byte0	Dbus 8:15 byte1	Dbus 16:23 byte2	Dbus 24:31 byte3
00	1111	fullword	byte0	byte1	byte2	byte3
00	1110	halfword	byte0	byte1	byte2	
01	0111	byte	**byte1**	byte1	byte2	byte3
00	1100	halfword	byte0	byte1		
01	0110	byte	**byte1**	byte1	byte2	
10	0011	halfword	**byte2**	**byte3**	byte2	byte3
00	1000	byte	byte0			
01	0100	byte	**byte1**	byte1		
10	0010	byte	**byte2**		byte2	
11	0001	byte	**byte3**	**byte3**		byte3

32-bit Master Write Data Mirroring