IFT6223
Systèmes reconfigurables

Projet Bubble

Jean Pierre DAVID

david@iro.umontreal.ca

Université de Montréal

Plan

- Liste des tâches
  - La mémoire nécessaire
- Le programme Bubble
- Spécifications du processeur
  - Le jeu d'instructions
  - Les registres – La mémoire et les E/S
- Le programme interpréteur
- L'entité VHDL du processeur
  - Remarque sur l'utilisation des variables en VHDL
- Le contrôleur d'interruption
- La gestion des E/S
- Le programme Bubble1.0 détaillé

Liste des tâches

- Afficher 3 cercles de tailles et de position variables
  - Calculer pour chaque ligne : 3 abscisses « start » et 3 « stop »
    - ✓Calcul de 3 racines carrées avec un coprocesseur
- Afficher la position du curseur souris
  - Utiliser la ressource « Mouse » d'Altera
    - ✓Inversion de la couleur sous le curseur (XOR)
- Gérer la croissance des cercles
  - Mémoriser le rayon et le carré du rayon de chaque cercle
    - ✓Utiliser (X+1)^2=X^2+2X+1
- Apparition aléatoire d'un cercle à l'écran
  - Fonction (Pseudo-)Random Hardware
- Gérer les exceptions
  - Le clic de la souris sur un cercle
  - La fin du jeu si un cercle sort de l'écran

La mémoire

- Pour chaque cercle :
  - Position du centre (X,Y) = 2 x 10 bits
  - Rayon = 1 x 10 bits
  - Rayon au carré = 1 x 20 bits
  - Les positions Start et Stop = 2 x 10 bits
    - ✓Total : = 18 registres 20 bits
- En global
  - La ligne courante = 1 x 10 bits
  - La position de la souris (X,Y) = 2 x 10 bits
  - Les points = 2 x 4 bits
    - ✓Total : = 4 registres 20 bits
- Autres
  - Quelques registres de calcul intermédiaire

Le programme « Bubble » 1/2

- Gestion des interruptions
  - 0 = Reset « MAIN »
  - 1 = Début d'une nouvelle image « INTPAGE »
  - 2 = Début d'une nouvelle ligne « INTLIGNE »
  - 3 = Fin (un cercle sort de l'écran) « RESTART »
  - 4 = Cercle rouge cliqué « REDTOUCH »
  - 5 = Cercle vert cliqué « GREENTOUCH »
  - 6 = Cercle bleu cliqué « BLUETOUCH »
  - 7 = Clic droit souris « RESTART »
- Méthodes
  - INCCIRCLE : incrémente le rayon d'un cercle
  - CIRCLE : Calcule les positions start et stop d'un cercle sur la ligne d'affichage courante

Le programme « Bubble » 2/2

- MAIN
  - Initialise le processeur (Pile, interruption)
  - Boucle infinie en attente d'une interruption
- INTPAGE
  - Incrémente le rayon de chaque cercle en appelant 3 fois la méthode INCCERCLE
- INTLINE
  - Ajourne la valeur du compteur de ligne
  - Calcule les valeurs start et stop pour chaque cercle en appelant 3 fois la méthode CERCLE
- RESTART
  - Calcule 3 position aléatoires pour les centres des cercles et leur attribue un rayon nul. Les points sont remis à zéro.
- …TOUCH
  - Attribue un certain nombre de points
  - Change la position du cercle et lui attribue une taille unitaire

Spécifications du processeur 1/2

- La « puissance de calcul » nécessaire
  - Une fonction racine carrée (coprocesseur)
  - Une ALU assez simple :
    - ✓ ADD – SBT – SHL – SHR – AND – OR – XOR
  - Des registres 24 bits
    - ✓ Le calcul de Y=SQRT(R^2-X^2) demande au moins 20 bits
  - Un certain pipeline pour calculer les valeurs start et stop de la ligne suivante sur le temps que l'on affiche celles de la ligne courante
  - Un générateur pseudo-aléatoire
- Une architecture 16 bits pour les instructions
  - Pour avoir un jeu d'instructions assez étendu
- Une architecture 24 bits pour les données
  - Pour que tous les calculs se fassent sans débordement

Spécifications du processeur 2/2

- Types d'instructions
  - Lecture : LDR (Indexé)
  - Ecriture : STR (Indexé)
  - Branchement : JP (absolu)
  - Appel : CALL (absolu)
  - Constante 8 bits : MOV (immédiat)
  - Opérations ALU : ADD … XOR (3 registres)
  - Coprocesseur : SCO (Send) et GCO (Get)
  - Divers :
    - ✓ Retour : RET
    - ✓ Retour int : RETI
    - ✓ Autoriser int : EINT
    - ✓ Interdire int : DINT

Le jeu d'instructions

Assembleur	IR[15..12]	IR[11..8]	IR[7..4]	IR[3..0]
LDR RA,RB(D)	0	RA	RB	D
STR RA,RB(D)	1	RA	RB	D
JP CC,N	2	CC	N(7..4)	N(3..0)
CALL RA,N	3	RA	N(7..4)	N(3..0)
MOVL RA,N	4	RA	N(7..4)	N(3..0)
ADDI RA,N	5	RA	N(7..4)	N(3..0)
SHL RA,RB,N	6	RA	RB	N(3..0)
SHR RA,RB,N	7	RA	RB	N(3..0)
ADD RA,RB,RC	8	RA	RB	RC
SUB RA,RB,RC	9	RA	RB	RC
AND RA,RB,RC	A	RA	RB	RC
OR RA,RB,RC	B	RA	RB	RC
XOR RA,RB,RC	C	RA	RB	RC
SCO CMD,RB,RC	D	0 | CMD	RB	RC
GCO CMD,RB	D	1 | CMD	RB	X
…	E	X	X	X
RET RB	F	0	RB	X
RETI	F	1	X	X
EINT	F	2	X	X
DINT	F	3	X	X

Les registres – la mémoire et les E/S

- Les registres
  - RA,RB et RC spécifient un registre parmi 16 (R_0..R_{15})
  - Ces registres occupent le début de la mémoire de donnée
- La mémoire de donnée contient 128 mots de 24 bits
  - Toute opération sur les adresses 0x00 à 0x79 concerne donc la mémoire de donnée (et éventuellement un registre
- Les entrées sorties occupent l'espace 129-256
  - Toute opération sur les adresse 0x80 à 0xFF concerne donc les entrées sorties
- Les adresses sont toujours calculées en mode indexé :
  - LDR R3,R7(4) // R3=DATA_MEM[R7+4]
  - Remarque : si R7 valait 5, cela revient à faire R3=R9

Le programme interpréteur (ASM)

```
INIT:    if (true)                    {PC=0;goto FETCH;}
FETCH:   if (INTERRUPT and IENA)      {IENA=0;PC_BACK=PC;STATUS_BACK=STATUS;
                                       PC=INT_VECTOR; IACK=1; goto FETCH}
         else                         {IR=PROGMEM[PC]; PC=PC+1;  DATA_WE=0; EXT_WE=0;
                                       COPRO_START=0; IACK=0; goto EXECUTE;}
EXECUTE: switch (OPCODE) {
         case 0:  {ARITH=ADD; DATA_MAR=Rb; goto LDR_1;}          // LDR Ra,Rb(D)
         case 1:  {ARITH=ADD; DATA_MAR=Ra; goto STR_1;}          // STR Ra,Rb(D)
         case 2:  {goto JP1;}                                    // JP CC,N
         case 3:  {DATA_MAR=Ra; DATA_IN=PC; DATA_WE=1;PC=N; goto FETCH;}  // CALL Ra,N
         case 4:  {DATA_MAR=Ra; DATA_IN=N; DATA_WE='1';goto FETCH;}       // MOVL Ra,N
         case 5:  {ARITH=ADD; DATA_MAR=Ra; goto ADDI_1;}         //ADDI Ra,N
         case 6:  {ARITH=SHL; DATA_MAR=rb; goto SH_1;}           // SHL Ra,Rb,D
         case 7:  {ARITH=SHR; DATA_MAR=rb; goto SH_1;}           // SHR Ra,Rb,D
         case 8:  {ARITH=ADD; DATA_MAR=rb; goto ARITH_1;}        // ADD Ra,Rb,Rc
         case 9:  {ARITH=SUB; DATA_MAR=rb; goto ARITH_1;}        // SUB Ra,Rb,Rc
         case A:  {ARITH=AND; DATA_MAR=rb; goto ARITH_1;}        // AND Ra,Rb,Rc
         case B:  {ARITH=OR; DATA_MAR=rb; goto ARITH_1; }        // OR Ra,Rb,Rc
         case C:  {ARITH=XOR; DATA_MAR=rb; goto ARITH_1;}        // XOR Ra,Rb,Rc
```


Le programme interpréteur

```
         case D:  {DATA_MAR=Rb; goto SGCO_1;}    // SCO CMD,Rb,Rc or GCO CMD,Rc;
         case E:  {goto init;}                   // ??
         case F:  {DATA_MAR=Rb; goto OTH_1;}     // Others
         }
LDR_1:   if (true)                {ALU_INA=D; ALU_INB=DATA_OUT; goto LDR_2;}
LDR_2:   if (FLAGS(3))            {DATA_MAR=ALU_RESULT; EXT_MAR=ALU_RESULT;
                                   EXT_READ<=ALU_RESULT(7); goto LDR_3;
         else                     {DATA_MAR=ALU_RESULT; EXT_MAR=ALU_RESULT;
                                   EXT_READ<=0; goto LDR_3;
LDR_3:   if (FLAGS(3))            {DATA_MAR=Ra; DATA_WE=1; DATA_IN<=EXT_IN;
                                   EXT_READ<=0; goto FETCH;}
         else                     {DATA_MAR=Ra; DATA_WE=1; DATA_IN<=DATA_OUT; goto FETCH;}
STR_1:   if (true)                {TMP=DATA_OUT; DATA_MAR=Rb; goto STR_2;}
STR_2:   if (true)                {ALU_INA=D; ALU_INB=DATA_OUT; goto STR_3;}
STR_3:   if (FLAGS(3))            {EXT_MAR=ALU_RESULT; EXT_OUT=TMP;
                                   DATA_MAR=ALU_RESULT; EXT_WE=1; goto FETCH;}
         else                     {EXT_MAR=ALU_RESULT; EXT_OUT=TMP;
                                   DATA_MAR=ALU_RESULT; DATA_WE=1; goto FETCH;}
```

Le programme interpréteur

```
JP1:        switch (M(CC,0,2)) {
            case 0:     {PC=N; goto FETCH;}
            case 1:     {goto JP11;}
            case 2:     {goto JP12;}
            case 3:     {goto JP13;}
            }
JP11:       if (CARRY='1')      {PC=N; goto FETCH;}
            else                {goto FETCH;}
JP12:       if (ZERO='1')       {PC=N; goto FETCH;}
            else                {goto FETCH;}
JP13:       if (NEG='0')        {PC=N; goto FETCH;}
            else                {goto FETCH;}
ADDI_1:     if (true)           {ALU_INA=N; ALU_INB=DATA_OUT; goto ADDI_2;}
ADDI_2:     if (true)           {DATA_MAR=Ra; DATA_IN=ALU_RESULT;
                                 DATA_WE=1; goto FETCH;}
SH_1:       if (true)           {COUNT=D; ALU_INA=DATA_OUT; goto SHL_2;}
SH_2:       if (COUNT>1)        {COUNT--; ALU_INA=ALU_RESULT; goto SHL_2;}
            else                {COUNT--; DATA_MAR=Ra; DATA_IN=ALU_RESULT;
                                 WE=1; goto FETCH;}
```


Le programme interpréteur

```
ARITH_1:    if (true)           {ALU_INA=DATA_OUT; DATA_MAR=Rc; goto ADD_2;}
ARITH_2:    if (true)           {ALU_INB=DATA_OUT; goto ADD_3;}
ARITH_3:    if (true)           {DATA_MAR=Ra; DATA_IN=ALU_RESULT; DATA_WE=1; goto FETCH;}
SGCO_1:     if (SUBCODE(3))     {goto GCO_1;}
            else                {COPRO_INA=DATA_OUT;DATA_MAR=Rc; goto SCO_1;}
GCO_1:      if (complete)       {DATA_MAR=Rb; DATA_IN=COPRO_RESULT; DATA_WE=1; goto FETCH;}
            else                {goto GCO_1;}
SCO_1:      if (true)           {COPRO_INB=DATA_OUT;COPRO_START=1; goto FETCH;}
OTH_1:      switch (M(IR,0,2))  {
            case 0:  {DATA_MAR=Rb; REG_DATA_OUT=DATA_OUT; goto RET_1;}   // RET Rb
            case 1:  {PC=PC_BACK; STATUS=STATUS_BACK; IENA=1; goto FETCH;} // RETI
            case 2:  {IENA=1; goto FETCH;}                               // EINT
            case 3:  {IENA=0; goto FETCH;}                               // DINT
            }
RET_1:      if (true)           {PC=REG_DATA_OUT; goto FETCH;}
```


L'entité VHDL du processeur (1/2)

```
ENTITY UP IS
        PORT(
                NRESET          : IN STD_LOGIC;
                CLK             : IN STD_LOGIC;
                INTERRUPT       : IN STD_LOGIC;
                INT_VECTOR      : IN STD_LOGIC_VECTOR(7 downto 0);
                IACK            : OUT STD_LOGIC;
                EXT_MAR         : OUT STD_LOGIC_VECTOR(6 downto 0);
                EXT_OUT         : OUT STD_LOGIC_VECTOR(23 downto 0);
                EXT_WE          : OUT STD_LOGIC;
                EXT_READ        : OUT STD_LOGIC;
                EXT_IN          : IN STD_LOGIC_VECTOR(23 downto 0);
                COPRO_INA       : OUT STD_LOGIC_VECTOR(23 downto 0);
                COPRO_INB       : OUT STD_LOGIC_VECTOR(23 downto 0);
                COPRO_CMD       : OUT STD_LOGIC_VECTOR(2 downto 0);
                COPRO_START     : OUT STD_LOGIC;
                COPRO_RESULT    : IN STD_LOGIC_VECTOR(23 downto 0);
                COPRO_COMPLETE  : IN STD_LOGIC);
END UP;
```


L'entité VHDL du processeur (2/2)

- Une machine à états gère l'ASM du processeur et les transfers de registres
- Un process combinatoire gère les « commandes » de la partie de contrôle :
  - ✓ Adresse de la mémoire de programme
  - ✓ Adresse de la mémoire de données
  - ✓ Donnée d'écriture de la mémoire de données
  - ✓ Commande d'écriture de la mémoire de données
  - ✓ Gestion du signal Interrupt Acknowledge (IACK)
- L'utilisation de variables VHDL permet de simplifier grandement l'écriture en affectant une valeur par défaut aux signaux de commande.

Remarques sur l'utilisation de variables en VHDL

- Une variable est LOCALE à un process.
- Il faut commencer par lui attribuer une valeur (pas de mémorisation implicite car certains compilateurs ne les reconnaissent pas).
- Une variable est un « intermédiaire de calcul » qui permet de faciliter l'écriture des process en DECRIVANT (pas en exécutant !!!) séquentiellement une opération (mais cette opération est sensée être réalisée en un temps nul).

- Exemple :

```
Process(a,b,c,d,e)
Variable var_a : integer;
Begin
  var_a:=a+b;
  var_a:=var_a*c+d;
  var_a:=var_a*e;
  result<=var_a;
End process;
```

équivalent

```
Signal tmp1,tmp2    : integer;
…
Process(a,b)
Begin tmp1<=a+b; End process;
Process(tmp1,c,d)
Begin tmp2<=tmp1*c+d; End process;
Process(tmp2,e)
Begin result<=tmp2*e; End process;
```


Le contrôleur d'interruptions (1/2)

```
ENTITY INT_MANAGER IS
        PORT(   NRESET      : IN STD_LOGIC;
                CLK         : IN STD_LOGIC;
                IRPT        : IN STD_LOGIC_VECTOR(7 downto 0);
                INTERRUPT   : OUT STD_LOGIC;
                INT_VECTOR  : OUT STD_LOGIC_VECTOR(7 downto 0);
                IACK        : IN STD_LOGIC);
END INT_MANAGER;
```

- IRPT(7 downto 0) : 8 sources possibles d'interruption
- Déclenchement sur un flanc montant
- Priorité décroisante de IRPT0 à IRPT7
- Lors d'une interruption, le contrôleur active le signal INTERRUPT et transmet le vecteur d'interruption au moyen du signal INT_VECTOR.
- Il attend ensuite une réponse du processeur par le signal IACK.

Le contrôleur d'interruptions (2/2)

```
TYPE state_type is (Idle, Run);
SIGNAL state            : state_type;
SIGNAL s_irpt           : STD_LOGIC_VECTOR(7 downto 0);
SIGNAL s_last_irpt      : STD_LOGIC_VECTOR(7 downto 0);
SIGNAL s_irpt_pending   : STD_LOGIC_VECTOR(7 downto 0);
SIGNAL s_int_vector     : STD_LOGIC_VECTOR(7 downto 0);
```

- Idle : Aucune interruption n'est en cours
- Run : L'interruption s_int_vector est en attente d'une confirmation
- s_irpt et s_last_irpt mémorise l'état des lignes d'interruptions au cycle courant et au cycle précédent.
- s_irpt_pending mémorise les interruptions qui ont été reçues et qui doivent encore être traitées.
- Une machine à état gère tous les changements d'états des registres au moyen de variables VHDL.

Le contrôleur d'E/S (1/2)

```
ENTITY IO_INTERFACE IS
PORT(   NRESET      : IN STD_LOGIC;
        CLK         : IN STD_LOGIC;
        ADDR        : IN STD_LOGIC_VECTOR(6 downto 0);
        DATA_IN     : IN STD_LOGIC_VECTOR(23 downto 0);
        WE          : IN STD_LOGIC;
        DATA_OUT    : OUT STD_LOGIC_VECTOR(23 downto 0);
        MOUSE_X     : IN STD_LOGIC_VECTOR(9 downto 0);
        MOUSE_Y     : IN STD_LOGIC_VECTOR(9 downto 0);
        PIXEL_ROW   : IN STD_LOGIC_VECTOR(9 downto 0);
        LOW_R       : OUT STD_LOGIC_VECTOR(9 downto 0);
        LOW_G       : OUT STD_LOGIC_VECTOR(9 downto 0);
        LOW_B       : OUT STD_LOGIC_VECTOR(9 downto 0);
        HIGH_R      : OUT STD_LOGIC_VECTOR(9 downto 0);
        HIGH_G      : OUT STD_LOGIC_VECTOR(9 downto 0);
        HIGH_B      : OUT STD_LOGIC_VECTOR(9 downto 0);
        SCORE       : OUT STD_LOGIC_VECTOR(9 downto 0));
END IO_INTERFACE;
```

Le contrôleur d'E/S (2/2)

Adresses d'écriture:

0x80	:	LOW_R_tmp
0x81	:	HIGH_R_tmp
0x82	:	LOW_G_tmp
0x83	:	HIGH_G_tmp
0x84	:	LOW_B_tmp
0x85	:	HIGH_B_tmp

0x87 : Transfer des …tmp vers les sorties (pour le pipeline)

Adresses de lecture:

0x80	:	MOUSE_X
0x81	:	MOUSE_Y
0x82	:	PIXEL_ROW (numéro de la ligne balayée par le spot VGA)
0x83	:	Générateur pseudo-aléatoire

Le programme Bubble1.0 (1/4)

```
INTLINE:    MOVL   R12,H#80
            LDR    R8,R12,H#2
            STR R8,R12,H#7
            MOVL   R12,H#10
            CALL   R15,CIRCLE
            MOVL   R12,H#18
            CALL   R15,CIRCLE
            MOVL   R12,H#20
            CALL   R15,CIRCLE
            RETI
INTPAGE:    MOVL   R12,H#10
            CALL   R15,INCCIRCLE
            MOVL   R12,H#18
            CALL   R15,INCCIRCLE
            MOVL   R12,H#20
            CALL   R15,INCCIRCLE
            RETI
```

Adresse	Variable	Adr. Relative
0x10	CX1	0
0x11	CY1	1
0x12	R1	2
0x13	SR1	3
0x14	Y1	4
0x15	SY1	5
0x16	X1	6
0x17		7
0x18	CX2	
0x19	CY2	
0x1A	R2	
0x1B	SR2	
0x1C	Y2	
0x1D	SY2	
0x1E	DX2	
0x1F		
0x20	CX3	
0x21	CY3	
0x22	R3	
0x23	SR3	
0x24	Y3	
0x25	SY3	
0x26	DX3	
0x27		

Le programme Bubble1.0 (2/4)

```
INCCIRCLE:  LDR    R0,R12,3      ; R0=SR
            LDR    R1,R12,2      ; R1=R
            ADDI   R1,1          ; R1=R+1
            STR    R1,R12,2      ; R=R+1
            SHL    R1,R1,1       ; R1=R1*2
            ADD    R0,R0,R1
            ADDI   R0,H#FF       ; -1
            STR    R0,R12,3      ; SR=SR+2R-1
            RET    R15
CIRCLE:     LDR    R7,R12,1      ; R7=CY
            SBT    R0,R8,R7      ; R0=DELTA=LINE(R8)-CY
            JP POS,CIR2
            SBT    R0,R7,R8       ; R0=DELTA=CY-LINE
CIR2:       LDR    R6,R12,2      ; R6=R
            SBT    R1,R0,R6      ; R1=DELTA-R
            JP     Z,CIRSTART    ; Debut (ou fin) du cercle
            JP     POS,CIREND    ; Exterieur du cercle
```


Le programme Bubble1.0 (3/4)

```
            LDR    R5,R12,4      ; R5=Y
            SHL    R0,R5,1       ; R0=2Y
            LDR    R1,R12,5      ; R1=SY
            SBT    R0,R1,R0      ; R0=SY-2Y
            ADDI   R0,1          ; R0=R0+1
            STR    R0,R12,5      ; SY=SY-2Y+1
            ADDI   R5,H#FF       ; R5=R5-1
            STR    R5,R12,4      ; Y=Y-1;
            LDR    R1,R12,3      ; R1=SR
            SBT    R5,R1,R0      ; R5=N=SR-SY
CIRLOOP:    SCO    0,R5,R5
            GCO    0,R0          ; R0=SQRT(X)
            JP AL,CIRLR
CIRSTART:   LDR    R0,R12,3      ; R0=SR
            STR    R0,R12,5      ; SY=SR
            LDR    R0,R12,2      ; R0=R
            STR    R0,R12,4      ; Y=R
            MOVL   R0,0
```

Le programme Bubble1.0 (4/4)

```
CIRLR:    STR   R0,R12,6      ; X=NEWX
          LDR   R1,R12,0      ; R1=CX
          SBT   R2,R1,R0      ; R2=LEFT
          ADD   R3,R1,R0      ; R3=RIGHT
          SHR   R0,R12,2      ; R0=R12/4
          ADDI  R0,124        ; R0=R0+124
          STR   R2,R0,0       ; Store LEFT
          STR   R3,R0,1       ; Store RIGHT
CIREND:   RET   R15
```