Description du circuit-logique d'un Processeur.

Bruno Malenfant

6 février 2006

1 Introduction

Ce document contient un essaie pour décrire un circuit logique sans utiliser de dessin. Le graphique est décrit à l'aide de texte seulement. Cette description comporte deux éléments : des noms de variables et des fonctions.

Les noms de variables représentent les fils électrique portant les différents signaux logique : 0 et 1. Chaque identificateur de variable peut être suivit d'un caractère ':' et d'un nombre. Cette syntaxe indique que le signal est construit de plus de 1 bit, le nombre indique la largeur du signal (le nombre de bits qui le compose). Un identificateur de signal peut aussi être suivit d'accolades '[]'. Dans ce cas les valeurs comprise dans l'intervale représente les numéro des bits utilisés. Voici un exemple :

```
( valeur_saut:28 )
  Décalage_gauche_2( '', instruction[25..0] )

-- dans cette exemple les identificateurs suivant représentent des
-- variables~: valeur_saut, instruction.
-- Le signal valeur_saut:28 contient 28 bits.
-- Le signal instruction[25..0] représente les bits 0 à 25 inclusivement du
-- signal 'instruction'
```

Il est possible de concaténer deux signaux en plaçant les caractères '++' entre les deux. Les signaux à gauche du symbole deviennent les bits de poids fort et ceux a droite deviennent les bits de poids faible.

Les fonctions représentent les portes logiques simples et complexe. Une fonction est décrite par un identificateur et deux vecteurs de variables. Le premier vecteur indique les signaux de sorties et le deuxième vecteur indique ceux d'entrées. Ces vecteurs sont délimités par des paranthèses. Dans l'exemple précédant la fonction 'Décalage_gauche_2' a en entrée un signal du nom de 'instruction' et en sorties un signal du nom de 'valeur_saut'. Le premier élément du vecteur d'entrées contient une chaîne de caractères entre apostrophe, cette chaîne de caractère contient le nom d'une instance du circuit représenté. Si un

argument est inutilisé alors le symbole '-' représente l'absence d'une connection pour cette entrée ou sortie.

La description du fonctionnement d'un circuit peut être faite textuellement, dans ce cas les lignes de texte sont précédé de '–'. Sinon la description est faite en utilisant des signaux et des fonctions. Dans les deux cas la description est contenue entre les caractères '<' et '>'.

Pour construire une instance d'une porte déjà décrite il suffit d'écrire un vecteur contenant ses sorties suivit d'un '=', ensuite de l'identificateur de la porte et finalement d'un vecteur de signaux d'entrées.

Il est possible de décrire plusieurs instance similaire à l'aide du mot clef 'pour'. Par exemple pour inverser tout les bits d'un signal B:32 :

```
pour i = 0 a 31
  ( invB[i] ) = NON( '', B[i] )
```

remarque : ce langage sert à décrire l'agencement des portes logiques pour un circuit, pas nécessairement sont fonctionnement.

2 Syntaxe abstraite du langage

```
Circuit ::= ( Déclaration ) *

Déclaration ::= VecteurSorties Identificateur VecteurEntrées '<' ( Description ) * '>'

Description ::= '--' Texte
     |  DescriptionSimple
             |  'pour' identificateur '=' Expression 'a' Expression DescriptionSimple

DescriptionSimple ::= VecteurSorties '=' Identificateur VecteurEntrées

Expression ::= Nombre
            |  Identificateur
            |  Expression Operateur Expression

Operateur ::= '+' | '-'

VecteurSorties ::= '(' Signal ( ',' Signal ) * ')'

VecteurEntrées ::= '(' ''' Texte ''' ( ',' Signal ) * ')'

Signal ::= '-'
|  NomSignal
|  Signal '++' Signal
```

```
NomSignal ::= Identificateur
   |  Identificateur ':' Nombre
   |  Identificateur '[' Nombre [ '..' Nombre ] ']'
```

3 Description des portes logiques simples

```
( r ) IDENTITE( 'nom', x )
<
  -- retourne une copie du bit en entrée.
>

( r ) ET( 'nom', x_1, x_2, ..., x_n )
<
  -- La porte ET est une fonction qui retourne 0 si
  -- une de ses entrées est à 0.  Sinon elle retourne 1.
>

( r ) OU( 'nom', x_1, x_2, ..., x_n )
<
  -- La porte OU est une fonction qui retourne 1 si
  -- une de ses entrées est à 1.  Sinon elle retourne 0.
>

( r ) NON( 'nom', x )
<
  -- La porte NON inverse son entrée, un 0 devient un 1
  -- et un 1 devient un 0.
>
```

4 Modules de base

```
( r ) XorSimple( 'nom', a, b )
<
  ( nonA ) = NON( '', a )
  ( nonB ) = NON( '', b )
  ( r1 ) = ET( '', a, nonB )
  ( r2 ) = ET( '', nonA, b )
  ( r ) = OU( '', r1, r2 )
>
```

```
( r ) Xor( 'nom', x_1, x_2, ..., x_n )
<
  -- Calcul un Xor sur tout les bits.
  -- Donc calcul une parite pair.
>

( r ) MultiplexeurSimple( 'nom', c, a, b )
<
  ( d ) = NON( '', c )
  ( sa ) = ET( '', a, d )
  ( sb ) = ET( '', b, c )
  ( r ) = OU( '', sa, sb )
>

( r:n ) Multiplexeur( 'nom', c:k, x_0:n, x_1:n, ..., x_n:n )
<
  -- Un Multipexeur est une generalisation du MultiplexeurSimple.
  -- Il permet de choisir entre plusieurs entrées de n bits.
  -- Le contrôleur 'c' indique le numéro de l'entrée qui sera transmit
  -- en sortie.
>

( r:n ) Registre( 'nom', v:n )
<
  -- Un registre est une unitée mémoire.  La valeur mémorisée
  -- est continuellement émise à la sortie.  La valeur présente
  -- en entrées est mémorisé lorsque l'horloge du système passe
  -- de 1 à 0.
>

( r:p ) Extension( 'nom', v:n )
<
  -- Le module d'extension de signe permet de modifier une valeur de
  -- 'n' bits vers une valeur de 'p' bits.  Ce module ne change pas
  -- la magnétude ni le signe de la valeur.

  pour i = 0 a n-1
    ( r[i] ) = IDENTITE( '', v[i] )

  pour i = n a p-1
    ( r[i] ) = IDENTITE( '', v[n-1] )
>

( r:n+2 ) Décalage_gauche_2( 'nom', v:n )
<
```

```
  -- Le module de décalage vers la gauche multiplie la valeur en entrée
  -- par 4.

  ( r[0] ) = IDENTITE( '', 0 )
  ( r[1] ) = IDENTITE( '', 0 )
  pour i = 2 a n+1
    ( r[i] ) = IDENTITE( '', v[i] )
>
```

5 Modules avancés

```
( r:32 ) Unitée_mémoire( 'nom', écrire, lire, adresse:32, valeur:32 )
<
  -- une unitée mémoire est un module qui communique avec la mémoire via le
  -- bus.  Cette unitée est en charge de construire des requêtes qui sont
  -- expédiées à la mémoire.  Il y a deux requêtes possible, lire ou écrire.
  -- Si le bit de contrôle 'écrire' est à 1 alors l'unité envoie les
  -- valeur : 'adresse' et 'valeur' à la mémoire.  Il y aura écriture
  -- le la valeur à l'adresse indiqué.  Si le bit 'lire' est à 1 alors
  -- l'adresse est envoyée à la mémoire et la valeur lue à cette adresse
  -- est retourné par l'unitée mémoire.
>

( valeur_a:32, valeur_b:32 )
  Banc_registre( 'nom', écrire, source_a:5, source_b:5, dest:5, résultat:32 )
<
  -- Le banc de registre contient 32 registre de 32 bits.  Ils sont numérotés
  -- de 0 à 31.  Les valeurs 'source_a' et 'source_b' indique les numéro de 2
  -- registre qui seront lus.  Les valeurs lues sont retourné par le module.
  -- Si le bit 'écrire' est à 1 alors la valeur 'résultat' est écrite dans
  -- le registre numéro 'dest'.
>
```

6 Ual

```
( s, ro ) Additionneur( 'nom', a, b, ri )
<
  ( s ) = Xor( '', a, b, ri )
  ( r1 ) = ET( '', a, b )
  ( r2 ) = ET( '', a, ri )
```

```
  ( r3 ) = ET( '', b, ri )
  ( ro ) = OU( '', r1, r2, r3 )
>

( s, ro ) Ual1Biti( 'nom', Binv, c:2, a, b, ri )
<
  ( resET ) = ET( '', a, b )
  ( resOU ) = OU( '', a, b )
  ( nonB ) = NON( '', b )
  ( resB ) = Multiplexeur( '', Binv, b, nonB )
  ( resADD, ro ) = Additionneur( '', a, resB, ri )
  ( s ) = Multiplexeur( '', c:2, resET, resOU, resADD , 0 )
>

( s, ro ) Ual1Bit0( 'nom', Binv, c:2, a, b, ri, signe )
<
  ( resET ) = ET( '', a, b )
  ( resOU ) = OU( '', a, b )
  ( nonB ) = NON( '', b )
  ( resB ) = Multiplexeur( '', Binv, b, nonB )
  ( resADD, ro ) = Additionneur( '', a, resB, ri )
  ( s ) = Multiplexeur( '', c:2, resET, resOU, resADD , signe )
>

( s, ro, signe ) Ual1Bitn( 'nom', Binv, c:2, a, b, ri )
<
  ( resET ) = ET( '', a, b )
  ( resOU ) = OU( '', a, b )
  ( nonB ) = NON( '', b )
  ( resB ) = Multiplexeur( '', Binv, b, nonB )
  ( signe, ro ) = Additionneur( '', a, resB, ri )
  ( s ) = Multiplexeur( '', c:2, resET, resOU, signe , 0 )
>

( s:n, z ) Ual( 'nom', cont:3, a:n, b:n )
<
  -- L'ual permet de faire des calculs arithmetique et logique.
  -- Le calcul effectué sur
  -- les sources 'a' et 'b' est indiqué par les bits de contrôle
  -- 'cont' selon la table suivante :
  --
  -- cont   opération
  -- ----------------
  --  0     AND
  --  2     OR
  --  4     ADD
```

```
  --  5     SUB
  --  7     SLT
  --
  -- Les valeurs 1, 3 et 6 ne sont pas utilisées.

  ( s[0], r[0] ) = Ual1Bit0( '', cont[2], cont[1..0], a[0], b[0], cont[0], signe )

  pour i = 1 a n-2
    ( s[i], r[i] ) = Ual1Biti( '', cont[2], cont[1..0], a[i], b[i], r[i-1] )

  ( s[n-1], r[n-1], signe ) Ual1Bitn( '', cont[2], cont[1..0], a[n-1], b[n-1], r[n-2] )

  ( nonZ ) = OU( '', s[0], s[1], ..., s[n-1] )
  ( z ) = NON( '', nonZ )
>
```

7 Chemin de donnée

```
( pc_courrant:32 ) =
  Registre( 'PC', nouveau_pc:32 )

( instruction:32 ) =
  Unitée_mémoire( 'Mémoire Instruction', 0, 1, pc_courrant:32, _ )

( numéro_reg_dest:5 ) =
  Multiplexeur( '', RegDst, instruction[20..16], instruction[15..11] )

( valeur_a:32, valeur_b:32 ) =
  Banc_registre( '', RegWrite, instruction[25..21], instruction[20..16],
    numéro_reg_dest:5, résultat:32 )

( valeur_étendue:32 ) =
  Extension( '', instruction[15..0] )

( source_2:32 ) =
  Multiplexeur( '', ALUSrc, valeur_b:32, valeur_étendue:32 )

( résultat_ual:32, zéro ) =
  Ual( 'ALU', cont_ual:3, valeur_a:32, source_2:32 )

( donnée_lue:32 ) =
  Unitée_mémoire( 'Mémoire Données', MemWrite, MemRead, résultat_ual:32, valeur_b:32 )

( résultat:32 ) =
```

```
  Multiplexeur( '', MemtoReg, donnée_lue:32, résultat_ual:32 )

( PC_4:32, _ ) =
  Ual( 'Add', 4:3, pc_courrant:32, 4:32 )

( valeur_décalée:34 ) =
  Décalage_gauche_2( '', valeur_étendue:32 )

( adresse_branch:32 ) =
  Ual( 'Add', 4:3, PC_4:32, valeur_décalée:[31..0] )

( effectue_branch ) =
  ET( Brach, zéro )

( choix_branch:32 ) =
  Multiplexeur( '', effectue_branch, PC_4:32, adresse_branch:32 )

( valeur_saut:28 ) =
  Décalage_gauche_2( '', instruction[25..0] )

( nouveau_pc:32 ) =
  Multiplexeur( '', Jump, PC_4[31..28]++valeur_saut[27..0], choix_branch:32 )

( RegDst, Jump, Branch, MemRead, MemtoReg, ALUOp:2, MemWrite, ALUSrc, RegWrite ) =
  Contrôleur( '', instruction[31..26] )

( cont_ual:3 ) =
  Contrôleur_ual( '', ALUOp:2, instruction[5..0] )
```