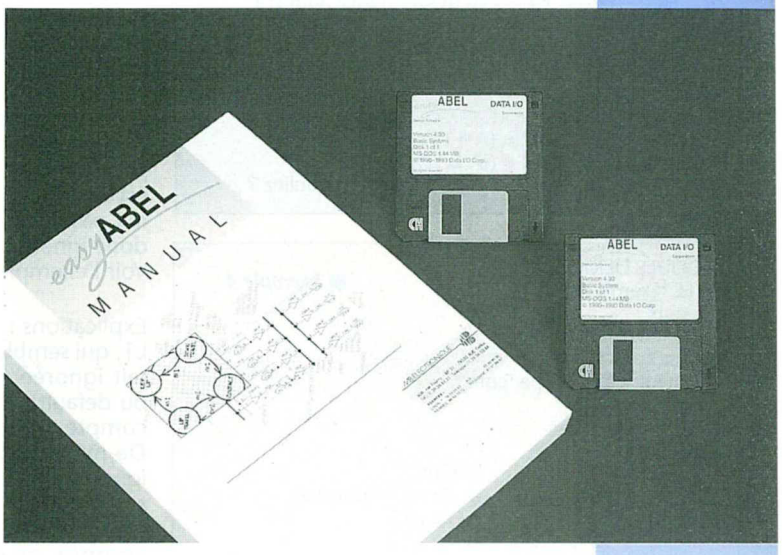
ABEL ET LES TABLES DE VÉRITÉ

Les Tables de Vérité (Truth Tables) sont une des méthodes de description logique qu'offre le langage Abel. Elles représentent une méthode simple et efficace, applicable à de nombreuses situations faisant appel à de la logique combinatoire. Cependant, leur usage, qui peut sembler évident, demande en fait au concepteur d'en avoir bien assimilé le principe. Ayant dû répondre à beaucoup de questions d'utilisateurs concernant cette méthode, il nous a semblé utile d'en décrire pas-à-pas l'utilisation. Les exemple listés sont bien entendu compilables sur les différentes moutures du compilateur ABEL, depuis la version easyABEL décrite dans cette revue (n° 567), jusqu'à l'environnement Synario pour les FPGAs / CPLDs.



PRINCIPE

Le principe de base d'une Table de Vérité est de construire une liste exhaustive des combinaisons d'entrées pour lesquelles chaque sortie devient active. Les termes de cette définition sont importants : nous allons voir que les implications sont nombreuses.

Syntaxe de base - Exemples simples

Pour une description syntaxique exhaustive, vous pourrez vous reporter au Manuel de Référence du langage ABEL, mais les exemples qui suivent sont assez clairs. Vous vous reporterez également avec profit à l'article déjà paru.
Dans l'exemple 1, les lignes L1 et L2 constituent le « ON-set » (ensemble des conditions d'activation de la sortie).
Comme l'équation obtenue le confirme, les lignes L3 et L4 sont ignorées car Out est de type défaut (donc '0' pour les cas non spécifiés).
Par contre, l'exemple 2 diffère du premier par Out qui est de type 'com, DC'. Dans ce cas, les lignes L3 et L4 constituent le « OFF-set » et les autres cas sont don't care (DC) c'est-à-dire 0 ou 1 suivant les besoins de l'optimisation.
L'équation qui en résulte, on le voit, est très simplifiée !
Une instruction « @DCSET » aurait eu le même effet que de déclarer Out de type « dc ». L'emploi de @DCSET doit être réfléchi dans le cas où la table spécifie plusieurs sorties : toutes seront affectées par la réduction.

```
MODULE DEMO1
TITLE 'Example 1'
« Inputs
      A, B, C  pin;
«Output
      Out  pin istype 'com';
Truth_Table
  (   [A,     B,     C]     -> Out )
      [0,     1,     0]     -> 1;  // L1
      [1,     1,     1]     -> 1;  // L2
      [0,     0,     1]     -> 0;  // L3
      [1,     0,     0]     -> 0;  // L4
END
// Equation Réduite résultat :
// Out = (!A & B & !C)  #  (A & B & C);
```

Exemple 1

```
MODULE DEMO1
TITLE 'Example 2'
« Inputs
      A, B, C  pin;
«Output
      Out  pin istype 'com, dc';
Truth_Table
  (   [A,     B,     C]     -> Out )
      [0,     1,     0]     -> 1;  // L1
      [1,     1,     1]     -> 1;  // L2
      [0,     0,     1]     -> 0;  // L3
      [1,     0,     0]     -> 0;  // L4
END
// Equation réduite résultat :
// Out = (B);
```

Exemple 2

Influence de la polarité

Nous allons voir que la polarité de la sortie (active à 1 ou à 0) a une influence importante.
Ici, Out1 et Out2 conduisent à deux résultats identiques. Dans le cas de !Out1, le « ON-set » correspond aux valeurs « 0 » :
la troisième ligne est ignorée.
Conclusion : pour les signaux de pola-

```
MODULE DEMO2
TITLE 'Example 3'
« Inputs
      A, B, C  pin;
«Output
      Out1  pin istype 'com, neg';
      Out2  pin istype 'com, neg';
      Out3  pin istype 'com, neg'; // BEWARE

Truth_Table
  (   [A,    B,    C]     -> [!Out1, Out2, Out3] )
      [0,    0,    1]     -> [ 0,   1,   0 ];  //L1
      [0,    1,    1]     -> [ 0,   1,   0 ];  //L2
      [1,    1,    0]     -> [ 1,   0,   1 ];  //L3
END
// Equations réduites :
//    !Out1     = !Out2 = (A # !C);
// ou : Out1   =  Out2 = (!A & C);
// MAIS Out3 = (A & B & !C);ce que vous vouliez ?
```

Exemple 3

```
MODULE DEMO3
TITLE 'Example 4'
« Inputs
      A, B, C          pin;
«Output
      Out    pin istype 'com';
« Equivalence
      X = .X.;
Truth_Table
  (   [A,    B,    C]     -> Out )
      [0,    0,    1]     -> 0; //L1 (ignorée)
      [0,    1,    0]     -> 1; //L2
      [1,    X,    X]     -> 1; //L3
      [0,    0,    1]     -> 1; //L4 incompatible L1
      [1,    1,    0]     -> 0; //L5 incompatible L3
END
// Resultat : Out = A # (B & !C) # (!B & C)
```

Exemple 4

```
MODULE DEMO5
TITLE 'Example 5'
« Inputs
      A, B, C  pin;
«Output
      Out    pin istype 'com, pos';
Truth_Table
  (   [A,    B,    C]     -> Out )
      [0,    0,    1]     -> 0;
      [0,    1,    0]     -> 0;
      [1,    0,    0]     -> 0;
//    [0,    0,    0]     -> 1;//changes everything!
END
Without line L4 :
!Out=(A & !B & !C)# (!A & B & !C)# (!A & !B & C);
WITH L4 : Out = (!A & !B & !C);
```

Exemple 5

```
MODULE DEMO6
TITLE 'Example 6'
« INPUTS
      A, B, C          PIN;
«OUTPUT
      OUT    PIN ISTYPE 'COM';
« EQUIVALENCE
      X = .X.;
TRUTH_TABLE
  (   [A,    B,    C]     -> OUT )
      [0,    0,    0]     -> 0;
      [0,    0,    1]     -> X;
      [0,    1,    0]     -> 1;
      [0,    1,    1]     -> X;
      [1,    X,    X]     -> X;
END
// AS IS : OUT = (!A & B & !C);
// WITH ISTYPE 'COM,DC' : OUT = (B);
```

Exemple 6

rité négative, il faut faire attention de spécifier « 1 » pour l'état actif si la sortie apparait sans « ! » —point d'exclamation—, et « 0 » si la sortie apparait complémentée.
La présentation conseillée est celle de Out1. Pour Out3, la ligne utile est L3, L1 et L2 ont été ignorées.

Utilisation de « .X. » dans les conditions

Les Don't Care que l'on peut utiliser dans la partie gauche des Tables de Vérité ne doivent pas être confondus avec les états « DC « servant à optimiser les équations. Les .X. ne servent qu'à regrouper plusieurs lignes de conditions en une seule, c'est une facilité d'écriture.
Caveat : Il faut utiliser avec précaution les .X. dans la mesure où l'on peut arriver à des conditions ambigües car se chevauchant et incompatibles. Ce type d'incohérence n'est pas détecté ni reporté par le compilateur. La raison en est simple : l'une des assignations ne fait pas partie du « ON-set » et elle est donc ignorée.
Voir l'exemple 4 :

Explications :
L1, qui semble en conflit avec L4 est en fait ignorée (Out de polarité positive ou default), donc L4 seule est prise en compte, sans rapport d'erreur.
De même, L5 intersecte L3, mais est ignorée car ne fait pas partie du « ON-set ».
Donc, seules L2, L3 et L4 sont prises en compte, ce qui conduit au résultat mentionné, sans qu'aucune erreur ne soit générée.
Il est donc très important de vérifier la cohérence de ce que l'on écrit, car des conflits « visibles » dans les Tables n'en sont en général pas du point de vue du compilateur.

Utilisation de .X. dans les assignations

La syntaxe autorise d'employer « .X. » comme valeur de sortie. Dans ce cas, il semble que le compilateur ignore alors la ligne, sans autre effet.
Dans tous les cas, ce n'est pas la méthode pour spécifier des états « don't care » optimisables automatiquement pour simplifier les équations obtenues. L'exemple 2, lui, montre comment spécifier ce type d'optimisation.
L'exemple 6 prouve que les lignes du type « -> .X. » ne sont pas optimisées si « DC » ou @DCSET ne sont pas spécifiés. Et dans ce dernier cas, elles ne servent à rien.
Elles semblent donc être d'aucune utilité, sauf peut-être pour documenter des conditions pour lesquelles la valeur de la sortie est réellement sans importance.

Cas spécial où le « ON-set » est vide

Il est rare que l'ensemble d'activation (« ON-set ») soit vide, mais ceci peut présenter un intérêt. Dans ce cas, le comportement du compilateur nous semble peu intuitif, aussi est-il important de comprendre de quoi il retourne. Voyons l'exemple :
Le résultat est un peu inattendu. On pense obtenir Out=0 puisque nous n'avons que des conditions où Out est nul, et que sa polarité est POS (ou défaut). Il n'en est rien !
Ce cas peut se présenter lorsque plusieurs sorties sont assignées dans la table, et que l'on souhaite « inactiver » une sortie pour une raison ou une autre.
Résultat : en l'absence de la ligne L4, la sortie vaut zéro uniquement pour les trois cas listés ! Pour les cinq (8-3) autres possibilités, Out vaut 1 même si il est de type POS ou DC !
Par contre, si la ligne L4 est présente, le « ON-set » est non vide, et l'équation obtenue est Out = (!A & !B & !C); ce qui redevient tout-à-fait normal.

CAS DE LA LOGIQUE SYNCHRONE

Il est possible de déclarer des sorties de type « registre » plutôt que combinatoires.
Dans ce cas, la syntaxe change légèrement. L'assignation se fait par « :> » au lieu de « -> ».
Tout ce qui précède s'applique. Une bacule est simplement insérée entre le signal et la sortie.

EN RÉSUMÉ

Les combinaisons de type « OFF-set » sont nécessaires lorsque la Table construit plusieurs sorties car elles ne sont pas activées toutes en même temps.
Il est important de se souvenir que les combinaisons de type « OFF-set » sont ignorées (cas par défaut), sauf si la sortie est déclarée de type « DC » (ou si l'instructions @dcset est active). Dans ce cas, un ensemble « DC-set » est construit à l'intérieur duquel les sorties se voient assigner des états propres à simplifier la logique générée.
Si les sorties ne sont pas de type « dc » et que @dcset n'est pas utilisé, alors on peut considérer chaque sortie une-à-une et ignorer les combinaisons ou elle n'est pas active.
.X. utilisé à droite de l'assignation n'a aucun effet d'optimisation par lui-même.
Lorsque la table spécifie plusieurs sorties de types différents, évitez @DCSET qui affectera toutes les sorties. Utilisez plutôt « istype '.....DC' » pour chaque concernée.
ATTENTION au cas où l'ensemble d'activation (« ON-set ») est vide !
En conclusion générale, ne vous fiez pas à une impression visuelle ou à l'intuition pour analyser ou construire une Table de Vérité, malgré son apparente simplicité. La seule « interprétation » qui fasse foi est celle du compilateur ! C'est lui qui mènera au fichier Jedec...
Présentez vos Tables de façon la plus claire possible, évitez les effets de bord, et documentez-les à l'aide de commentaires judicieux. Enfin, un « coup d'œil » aux équations réduites générées est une bonne pratique et permet de lever un doute éventuel ou détecter une anomalie.
Nous espérons que ces quelques lignes vous aideront à tirer le meilleur parti de la puissance du langage Abel, et en particulier de cette méthode très efficace que sont les Tables de Vérité.

ALSE Bertrand CUZEAU