N7_SdN_1A

Architecture des ordinateurs

TD1

Introduction (10 mn de lecture + 10 mn de questions /réponses)

A- Rappel rapide sur l'algèbre de Boole et ses théorèmes

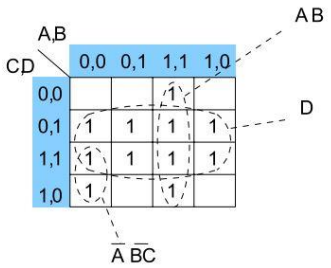
relation	relation duale	Propriété
$AB = BA$	$A + B = B + A$	Commutativité
$A.(B + C) = A.B + A.C$	$A + B.C = (A + B).(A + C)$	Distributivité
$1.A = A$	$0 + A = A$	Identité
$A.\overline{A} = 0$	$A + \overline{A} = 1$	Complément
$0.A = 0$	$1 + A = 1$	Zéro et un
$A.A = A$	$A + A = A$	Idempotence
$A.(B.C) = (A.B).C$	$A + (B + C) = (A + B) + C$	Associativité
$\overline{\overline{A}} = A$		Involution
$\overline{AB} = \overline{A} + \overline{B}$	$\overline{A + B} = \overline{A}.\overline{B}$	Théorèmes de De Morgan
$A.(A + B) = A$	$A + A.B = A$	Théorèmes d'absorption
$A + \overline{A}.B = A + B$	$A.(\overline{A} + B) = A.B$	Théorèmes d'absorption

B- Présentation des fonctions logiques

- Les fonctions logiques seront représentées sous la forme d'une somme de « mintermes », en utilisant les opérateurs logiques : / (complément binaire), + (OU logique) et * (ET logique)
- Pour un composant à 3 entrées E2, E1, E0 et une sortie S fonction de E2, E1, E0 ; on écrira par exemple S = /E2*E1*E0 + E2*/E1*E0 + E2*E1*/E0
- Ce sera la syntaxe utilisée par le langage shdl (langage de description utilisé en TP)

C- Simplification : on utilisera deux méthodes

- Regroupement de « mintermes » en utilisant les théorèmes de Boole
- Table de Karnaugh



Exercice 1 : (15 mn de travail + 5mn de correction)

Illustration avec un additionneur binaire, avec en entrée A, B, Cin (retenue entrante) ; et en sortie S et Cout (retenue sortante).

- Dresser la table logique
- Simplifier en utilisant les théorèmes de Boole
- Simplifier en utilisant la méthode de Karnaugh

- Ecriture le module en langage shdl

```
module addbin (a, b, cin : s, cout)        // ':' séparent les entrées des sorties (',' aussi)
        // équations logiques de s et de cout
end module
```

Le module doit être enregistré dans un fichier portant le même nom : ici « addbin.shdl ».

Exercice 2 : (10 mn de travail + 5mn de correction)
Sur le modèle d'une addition classique (addition des chiffres d'un étage et propagation de la retenue pour l'opération de l'étage suivant), réaliser le module adder8 (a[7..0], b[7..0], cin : s[7..0], cout) en utilisant le module addbin

- On peut utiliser des vecteurs de signaux, comme indiqué dans l'interface dessus. Par convention, 0 est l'indice de faible poids, et se met à droite
- L'utilisation d'un module existant se fait simplement en l'invoquant avec les bons signaux : par exemple : addbin(a[0], b[0], cin : s[0], c0)
  permet de décrire le premier étage de l'additionneur 8 bits en utilisant le module addbin
  c0 permet de propager la retenue intermédiaire vers l'étage suivant.

Pour la suite, on suppose disposer d'un module adder32 (a[31..0], b[31..0], cin : s[7310], cout)

Rappel sur le codage binaire en complément à 2 (10mn de lecture, 10mn d'explication / correction)

Codage des entiers naturels	Codage des entiers relatifs en complément à 2
$s = \sum_{i=0}^{n-1} 2^i b_i$	$s = -2^{n-1} b_{n-1} + \sum_{i=0}^{n-2} 2^i b_i$

- Pour les entiers relatifs, le bit numéro 'n-1' représente le signe (0 pour positif, 1 pour négatif)
- Le complément à 2 s'obtient aussi par la complémentation bit à bit et l'addition de 1
- Exemple : 10001000 représente la valeur :
  - 136 pour un entier naturel = 128 + 8
  - -120 pour un entier relatif = -128 + 8 (directement avec la formule dessus)
    En complément à 2 => 01110111 + 1 = 01111000 = 64+32+16+8 = 120 => -120
- Donnez les nombres naturels et relatifs représentées par : 10101010
- Donnez la représentation binaire sur 8 bits de -43

Exercice 3 (10mn de travail + 5mn de correction)
Réaliser un module addsub32 qui, en fonction d'une entrée sub, peut effectuer une addition (sub=0) ou une soustraction (sub=1) entre deux nombres de 32 bits

- Transformer l'opération A-B en une addition en remplaçant '-B' par le complément à 2 de 'B'
- Ecrire le module addsub32

Rappel sur les indicateurs (flags) C et V (10mn de lecture + 10mn de questions/réponses)

- une opération arithmétique peut engendrer une retenue finale (pour l'addition) ou emprunt (pour la soustraction). Cette information sera enregistrée dans un bit nommé **C (Carry)**
    - 1100 + 0100 => 1 0000 (C=1, résultat dépasse le 4 bits de codage)
    - 0100 – 0101 = 1 1111 (C=1, soustraction bit à bit avec propagation de l'emprunt). Il faut noter ici que le résultat obtenu avec addition du complément à 2 donne : 0100 + 1011 = 1111 (cout=0)
- Une opération entre nombre signés peut engendrer un débordement sur le bit de signe. Cette information sera enregistrée dans un bit nommé **V (oVerflow).**
    - C'est le cas pour l'addition entre deux nombres positifs avec un résultat ayant 1 sur le bit de signe : résultat faux si les opérandes sont des nombres signés. Même constat pour le cas symétrique.
      0111 + 0010 = 1001 peut être interprété comme 9 ou -7 (correct si valeurs non signées et faux si valeurs signée) **=> V=1**
    - C'est le cas aussi lorsqu'on soustrait un nombre avec le bit de signe S=0 à un nombre avec S=1 et on obtient un résultat avec le bit de signe=0 : résultat faux si les nombres sont signés, car négatif – positif doit donner un résultat négatif. Même constat pour le cas symétrique.
      1100 – 0110 = 1100 + 1010 = 0110 interprété comme 6 (correct si valeurs non signées et faux si valeurs signée) => **V=1**
    - L'indicateur V n'est utile que lorsqu'on travaille avec des opérandes signés.

Exercice 4 (10mn de travail + 5mn de correction)

- Donner les équations logiques de C et de V
- Compléter le module addsub32 en ajoutant en sortie les bits C et V