TD 1 - CIRCUITS COMBINATOIRES

Simplification de fonctions booléennes, familiarisation avec l'outil de conception de circuits, circuits combinatoires, représentation des nombres, arithmétique en complément à 2, arithmétique flottante.
Page web : `http://www-rocq.inria.fr/~acohen/teach/archi.html`

Exercice 1.1 - Afficheur numérique

On souhaite réaliser l'afficheur numérique décimal décrit sur la figure 1. Il est constitué de 7 diodes électro-luminescentes (LEDs) nommées *N*, *NW*, *NE*, *C*, *SW*, *SE* et *S*.

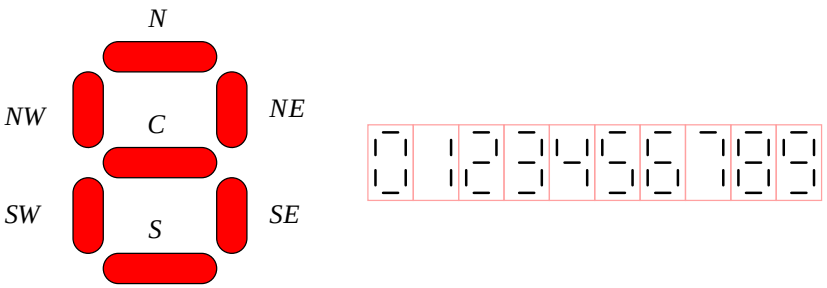

FIG. 1 – Afficheur numérique décimal

Question 1.1.1

Écrire une table de vérité pour chaque LED de l'afficheur, puis simplifier les fonctions algébriques correspondantes à l'aide des tableaux de Karnaugh. Pour ne pas perdre de temps, vous avez la possibilité de ne réaliser que deux tableaux, par exemple ceux du sud ouest (*SW*) et du centre (*C*).

Question 1.1.2

Réaliser l'afficheur en *DigLog*. Pour le circuit, on respectera la représentation des fonctions logiques sous forme de *sommes de produits*.

Support : commencer par récupérer le fichier `numeric_mask.lgf`, puis lancer *DigLog* par la commande `cohen/bin/diglog numeric_mask.lgf`.

Exercice 1.2 - Deux circuits combinatoires importants

On étudie deux circuits élémentaires fréquemment utilisés dans les composants des processeurs.

Question 1.2.1

Un *décodeur* $n \rightarrow 2^n$ est un circuit combinatoire comprenant une entrée I sur n bits et 2^n sorties $O_0, \ldots, O_{2^n-1}$ sur 1 bit ; seul le signal O_m doit être activé lorsque l'entrée I vaut m (en binaire sur n bits).

Construire un décodeur $3 \rightarrow 8$. On pourra construire une table de Karnaugh ou chercher un schéma général pour un décodeur $n \rightarrow 2^n$.

Question 1.2.2

Un *multiplexeur* $2^n \times 1$ est un circuit combinatoire comprenant une entrée A sur n bits, 2^n entrées $I_0, \ldots, I_{2^n-1}$ sur 1 bit, et une sortie O sur 1 bit ; le signal O est égal à l'entrée I_m lorsque A vaut m (en binaire sur n bits).

Construire un multiplexeur 8×1. On pourra construire une table de Karnaugh, réutiliser le décodeur de la question précédent ou chercher un schéma général.

Exercice 1.3 - Unité arithmétique et logique

Support : `alu_mask.lgf` propose une Unité Arithmétique et Logique (ALU) 1-bit (comprenant notamment un additionneur 1-bit et un multiplexeur 4×1) et un masque pour une ALU 16-bits.

L'ALU 1-bit vue en cours propose les opérateurs d'addition, de conjonction et de négation ; voir le fichier `alu_mask.lgf`. On utilise cette « brique de base » pour construire une unité arithmétique et logique 16-bits. On a transformé l'additionneur en additionneur/soustracteur en lui ajoutant un signal de contrôle $ALUK_{SUB}$ analogue au signal c vu en cours, et en insérant des XOR pour inverser la deuxième opérande de l'addition.

Question 1.3.1

La plupart des unités arithmétiques et logiques génèrent des signaux de condition ; il s'agit de signaux de sortie utilisés par les instructions de branchement conditionnel. Réaliser une ALU 16-bits avec les trois signaux de condition suivants :

- N vaut 1 lorsque la sortie (16-bits) est strictement négative ;
- Z vaut 1 lorsque la sortie est nulle ;
- P vaut 1 lorsque la sortie est strictement positive.

Question 1.3.2

Montrer que l'on peut effectuer une soustraction non signée à l'aide d'un opérateur de soustraction en complément à 2.

Question 1.3.3

On suppose que l'ALU dispose d'un signal *SIGNED* qui vaut 1 lorsque les calculs ont lieu en complément à 2 (calculs signés) et 0 sinon (calculs non signés). Définir un signal *V* de dépassement de capacité (*overflow*) pour cette ALU.

Question 1.3.4

En utilisant l'ALU ci-dessus et en minimisant le nombre de modifications, générer un signal de comparaison : soient a et b les deux entrées de l'ALU, le résultat de la comparaison est 1 si $a < b$, et 0 sinon ; le résultat de la comparaison doit être fourni sur un signal de sortie supplémentaire.

Exercice 1.4 (facultatif) - Accélération de l'addition

Cet exercice compare plusieurs compromis temps-espace pour des circuits d'accélération de l'addition. Il n'est pas nécessaire d'utiliser *DigLog* dans cet exercice.

On souhaite effectuer l'addition de n nombres $a_1, \ldots, a_n$ sur 4 bits. Pour cela, on peut utiliser une ALU pour calculer $a_1 + a_2$ puis ajouter le résultat à a_3, etc.

Question 1.4.1

Pour la somme de 3 entiers sur 4 bits, proposer un circuit dont le temps de traversée vaut 6 fois celui de l'additionneur 1-bit.

Question 1.4.2

Pour la somme de 3 entiers sur 4 bits, proposer un circuit dont le temps de traversée ne vaut que 5 fois celui de l'additionneur 1-bit, contre 6 pour le circuit de la question précédente. Indication : utiliser l'associativité de l'addition pour retarder la propagation des retenues jusqu'à la dernière étape du calcul ; le circuit obtenu est appelé additionneur *carry-save*.

Proposer une généralisation de ce circuit pour la somme de n opérandes et une estimation de son temps de traversée.

Exercice 1.5 (facultatif) - Multiplieur flottant

On étudie la structure d'un multiplieur pour des nombres à virgule flottante. On considère une représentation simplifiée, inspirée de la norme IEEE 754 pour les flottants 32 bits :

- on ne considérera que des nombres normalisés ;
- lorsqu'une valeur ne peut être représentée par un nombre normalisé, on déclenche un signal d'erreur *ERROR* ;
- le circuit ne pourra effectuer qu'un arrondi par élimination des bits de poids faible non représentables.

Question 1.5.1

Déterminer les différentes étapes nécessaires à l'opération de multiplication flottante

Question 1.5.2

Identifier tous les composants nécessaires à la réalisation du multiplieur (hors contrôle). Indiquer les jonctions nécessaires entre les différents composants et leur taille.

Question 1.5.3

Indiquer, pour chaque étape, les valeurs que doivent prendre les différents signaux d'entrée des composants.

Le circuit de contrôle du multiplieur flottant sera réalisé dans le cadre du prochain TD.