ARC TD1

Logique et arithmétique avec Digital
(1 séance sur machine $\simeq$ 4h), 9 avril 2024

Construit à partir du poly de TD-TP des cours IF-AC et IF-AO

Ce premier TP, on s'intéresse tout d'abord au calcul booléen puis on prendra en main de l'outil `Digital`. on verra ensuite comment on fait de l'arithmétique avec des portes logiques (section 2).

1 Partie 1 : Calcul booléen

On s'intéresse ici à l'équivalence expression booléenne $\leftrightarrow$ table de vérité $\leftrightarrow$ circuit combinatoire. Le dernier exercice détaille notamment la construction d'un multiplexeur et d'un démultiplexeur.

1.1 Calcul booléen

Relisez la *Boolean Cheat Sheet* (`cours1.pdf`, slide 14). En algèbre booléene, il y a distributivité de la multiplication sur l'addition ET distributivité de l'addition sur la multiplication (en arithmétique on n'a que la première). En d'autre termes, en algèbre de Boole on a $a + (bc) = (a + b)(a + c)$ ce qui est fort peu naturel ! C'est ce qui rend l'usage des notations (+, .) dangereux.
Faites bien attention à la position des parenthère en logique booléenne, Si vous oubliez (ou déplacez) les parenthèses au cours de cette opération, le résultat sera potentiellement très différent (et donc faux) par exemple :
$a(b + c) = (ab) + (ac) \neq (ab) + c$ et que $a + (bc) = (a + b)(a + c) \neq a + (ba) + c$ (attention donc aux parenthèses !)

QUESTION 1 ▶ Rappeler la règle de De Morgan sous ses deux formes.

QUESTION 2 ▶ Prouver les équivalences suivantes :

(1) $a.b + \overline{a}.b = b$ (2) $a + a.b = a$

(3) $a + \overline{a}.b = a + b$ (4) $a.b + \overline{a}.c = a.b + \overline{a}.c + b.c$

QUESTION 3 ▶ Simplifier les expressions suivantes, notamment grâce à la loi de De Morgan :

$$\begin{aligned} s_1 &= \overline{\overline{(x+y)}(x+z)}(y+z) \\ s_2 &= \overline{\overline{x.y} + yz} + \overline{(x+z)\overline{(y+\overline{x}z)}} \\ s_3 &= \overline{y(\overline{x}+z) + \overline{x(\overline{y}+z)}} + \overline{(y+z)}(xy + \overline{x(\overline{y}+\overline{z})}\,) \end{aligned}$$

1.2 Expression algébrique

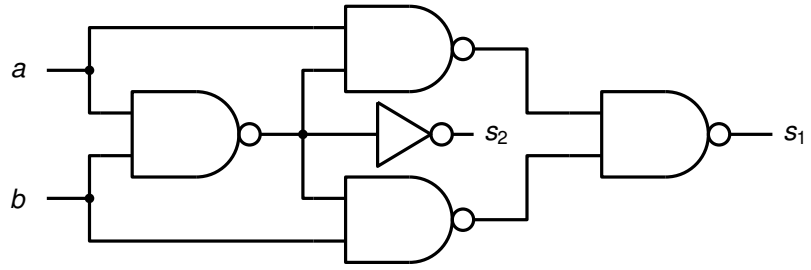
QUESTION 4 ▶ Donner une expression booléenne de la fonction $f(x, y, z)$ qui vaut 1 si et seulement si la majorité de ses trois arguments vaut 1.

QUESTION 5 ▶ Donner (sans chercher à la simplifier) une expression booléenne de la fonction $g(a, b, c)$ définie par la table de vérité suivante :

a	b	c	s
0	0	0	1
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	1

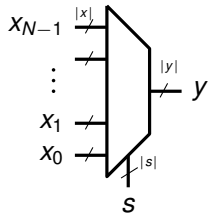
1.3 Circuits logiques

QUESTION 6 ▶ Donner une expression algébrique des sorties s_1 et s_2. Établir la table de vérité. La fonction calculée par ce circuit vous est-elle familière ?



1.4 Multiplexeur et démultiplexeur

On rappelle ci-dessous le symbole générique d'un multiplexeur à N entrées.



Intuitivement parlant, ce composant «recopie» l'une de ses entrées, x_i, sur sa sortie y. On choisit la valeur de i en positionnant l'*entrée de sélection* (notée s sur le symbole). Ainsi, si cette entrée de sélection est codée sur $|s|$ bits, elle permet de sélectionner parmi $N = 2^{|s|}$ entrées différentes. Par ailleurs, un multiplexeur peut être conçu pour travailler avec des données composées de plusieurs bits ; dans ce cas-là, la largeur $|y|$ de la sortie sera la même que la largeur $|x|$ des entrées.

QUESTION 7 ▶ Écrire la table de vérité complète d'un multiplexeur à 2 entrées de 1 bit chacune (on parle aussi de «multiplexeur 2-vers-1 à 1 bit»).

QUESTION 8 ▶ Donner une expression booléenne de la fonction réalisée par ce multiplexeur, et dessiner le circuit logique correspondant.

QUESTION 9 ▶ On s'intéresse maintenant à un multiplexeur à 4 entrées de 1 bits (on parle aussi de «multiplexeur 4-vers-1 à 1 bits»). Proposer une implémentation de ce circuit à l'aide de quelques instances du multiplexeur 2 vers 1 construit dans la question précédente. Dessiner le circuit logique correspondant.

2 deuxième partie – Circuits combinatoires

Dans cette partie, vous allez utiliser un simulateur ("Digital") pour "réaliser" vos premiers circuits numériques. De façon à éviter de perdre trop de temps à "tirer des fils" (ce qui est rapidement un

brin rébarbatif), nous limiterons la taille des mots manipulés par nos circuits à 4 bits [1]. Avant d'aller plus loin, vérifiez que vous êtes bien conscient.e qu'il n'y a pas de différence conceptuelle entre un circuit 8 bits et un circuit 16 ou 32 bits.

2.1 Prise en main de Digital

Digital est un simulateur de circuits numériques que nous allons utiliser dans les TP AC pour développer des circuits numériques (des circuits combinatoires puis séquentiels simples dans un premier temps, après quoi nous passerons à des circuits dîts « complexes »). Il sera ensuite réutilisé dans le cadre du cours d'AO (Architecture des Ordinateurs) pour simuler une machine de calcul.

Digital est installé sur les plateformes Linux du département TC [2], ici :

`/opt/Digital/`

Pour lancer Digital, il suffit de lancer `Digital.sh` dans le répertoire ci-dessus. Les instructions d'installation de Digital sur votre machine sont disponibles sur Moodle.

QUESTION 10 ▶ Afin de vous permettre de prendre en main l'outil, suivez le tutoriel de la documentation de Digital (menu `Help→Documentation`, juste les sections *A1.2* et *A1.3* de *First steps*). Notez que la configuration standard de Digital utilise une représentation simplifiée pour les portes logiques. Vous pouvez passer à la version « classique » utilisée en cours via le menu `Edit→Settings→Use IEEE 91-1984 shapes`.
En addition à la section *A1.2*, voici quelques *trucs* à connaître pour utiliser `Digital` :

- Utiliser `Ctrl-z` pour "défaire" une action malencontreuse.
- Utiliser `Ctrl-click` (sous entendu click-gauche) pour selectionner les fils (pour pouvoir les supprimer).
- Pour bouger un nœud, cliquez dessus, relacher le click et bouger le nœud. Si on maintient le click enfoncé, cela déplace tout le design.

2.2 Additionneur 1 bit

Créez quelquepart un répertoire `TP-Digital`. Vous y placerez tous les circuits demandés dans les différentes séances de TP.

QUESTION 11 ▶ Votre premier circuit dans ce répertoire réalisera l'addition 1 bit. Ce circuit possède trois entrées `a`, `b` et `c_in` représentant respectivement les deux valeurs à additionner et la retenue entrante de l'addition. Il possède deux sorties : `s`, dénotant la valeur de la somme, et `c_out`, dénotant la valeur de la retenue de sortie.
On vous rappelle la définition de `s` et `c_out` :

```
s = (a xor b) xor c_in
c_out = (a and b) or (a and c_in) or (b and c_in)
```

Par défaut, les portes ont 2 entrées. Si vous avez besoin d'un `AND` ou d'un `OR` à trois entrée (ou plus) vous pouvez changer l'attribut correspondant d'un clic droit de la souris. Vous pouvez aussi changer l'orientation des portes, complémenter certaines entrées des portes (très utile pour implémenter multiplexeurs et démultiplexeurs), etc.

Sauvegardez ce circuit dans un fichier nommé `add1bit`. Attention : il est important d'utiliser des noms informatifs car vous allez ensuite pouvoir réutiliser vos circuits pour en construire d'autres, plus complexes.

1. Ce qui ne nous fait pas remonter à un passé si lointain : les processeurs 4 bits étaient encore utilisés il n'y a "pas si longtemps" (le 4004 de Intel a été fabriqué jusqu'au début des années 80, et les calculettes HP jusqu'à la HP49 vendue dans les années 2000 utilisaient un autre processeur 4 bits, le Saturn).
2. Digital est disponible ici : `https://github.com/hneemann/Digital`. (la requête Google qui le trouve est "digital logisim"). Cliquez sur "Download latest Release", puis "Digital.zip" ; décompressez le fichier zip. Finalement, lancez `Digital.exe` sous Windows ou tapez dans un terminal `sh Digital.sh` sous Linux.

Au delà de l'exécution pas-à-pas, dirigée par le changement des valeurs d'entrées du circuit, que vous avez utilisée dans le tutoriel ci-dessus, Digital vous propose aussi un menu `Analysis` qui calcule la table de vérité de votre circuit (`Digital` propose plein d'autres outils d'analyse et de test que vous êtes fortement encouragés à essayer).

Une fois votre circuit terminé et analysé, si vous n'êtes pas sûr de vous, vérifiez avec un enseignant que le comportement observé à travers les informations fournies dans la fenêtre d'analyse est bien celui attendu.

2.3 Additionneur 4 bits

Pour construire un additionneur 4 bits, vous n'allez surtout pas copier-coller 4 fois les portes que vous venez d'utiliser pour l'additionneur 1 bit. Plutôt, vous allez encapsuler votre additionneur 1 bit dans un composant que vous pourrez ensuite réutiliser 8 fois pour obtenir l'additionneur 8 bits.

QUESTION 12 ▶ Lisez la partie 1.4 de la doc ("Hierarchical design"). Créez un nouveau fichier Digital par `File→New Embedded Circuit`. Sauvez le tout de suite sous le nom `add8bits` dans le même répertoire que votre `add1bit`. Ce dernier devrait apparaître dans la liste des composants disponibles, sous la rubrique `Components→Custom`.

Placez 4 copies du composant `add1bit` sur le circuit et connectez-les pour réaliser un circuit additionneur 4 bits. Attention, un additionneur 4 bits prend en entrée des mots de 4 bits, vous allez donc être amené à utiliser des *splitter* comme expliqué ci-dessous.

Placez les ports de votre additionneur 4 bits, nommez-les `a`, `b` et `c_in` pour les entrées et `s` et `c_out` pour les sorties et changez leur attribut "Data bits" (4 bits) quand nécessaire pour créer des entrées sortie de plusieurs bits (des "bus"). En effet, un additionneur 4 bits prend en entrée deux mots de 4 bits (`a` et `b`) et une retenue entrante de 1 bit (`c_in`).

Pour pouvoir connecter ces ports à vos composants, vous utiliserez des "**Splitter**" (menu `Components → Wires`). Pour configurer un *Splitter*, il faut faire un click-droit sur le composant. Les paramêtres *input splitting* et *output splitting* servent à configure le *splitter*. La syntaxe de ces deux champs est une suite d'entiers qui indiquent comment on groupe les bits. par exemple la suite 1,2,1 sur *input splitting* signifie que l'on a trois fils en entrée : un fil d'un bit (le bit 0), un fil de 2 bits (les bits 1 à 2) et un fil d'un bit (le bit 3) qui constituent nos 4 bits d'entrée.

Ici, nous avons besoin, par exemple pour l'entrée `a`, d'une configuration d'*input splitting* de 4 (i.e. 4 bits groupés) et une configuration d'*output splitting* de `1,1,1,1` (i.e. 4 fils d'un bit). Une notation résumée pour `1,1,1,1` est `1*4`.

Testez votre additionneur 4 bits et sauvegardez-le.

2.4 Additionneur / Soustracteur 8 bits

Lire la page 20 du Poly et comprendre le codage en complément à 2 des entiers relatifs sur n bits.

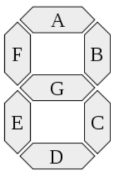
QUESTION 13 ▶ Constatez, Ô miracle que votre additionneur fonctionne aussi sur des entier en complément à deux. Pour cela passez les probes en *signed integer* et testez sur des valeurs comprises entre -128 et 127. Comprenez vous pourquoi ?
On peut profiter de la circuiterie de l'additionneur pour créer un circuit additionneur/soustracteur. Les modifications nécessaires se basent sur le calcul du complément binaire d'un deux des chiffres paramètres de la soustraction. Pour deux entiers dont on veut calculer la différence `A-B`, on calcule le complément du second, $\overline{\texttt{B}}$, chaque bit de $\overline{\texttt{B}}$ étant défini par la négation du bit correspondant dans $\overline{\texttt{B}}$. Puis, on calcule l'addition `A+`$\overline{\texttt{B}}$`+1`.
Pour réaliser cela à partir de votre additionneur 8 bits, vous utiliserez la retenue entrante, qui permettra maintenant de décider si on souhaite une addition (elle vaut alors 0) ou une soustraction (elle vaut alors 1). Cette entrée conditionne la complémentation de la deuxième entrée de l'opération, ainsi que la valeur de sa retenue d'entrée. Le choix entre `B` et $\overline{\texttt{B}}$ se fait à l'aide d'un xor pour chaque bit de `B`, prenant comme première entrée `B` et comme seconde entrée la retenue entrante de votre opération.

QUESTION 14 ▶ *(Si vous êtes en avance, sinon passez à la partie suivante)* Réalisez un additionneur / soustracteur 8 bits en suivant les indications précédentes.

3 Afficheur 7 segments

Les afficheurs 7 segments sont un type d'afficheur numérique très présent sur les calculatrices : les caractères s'écrivent en allumant ou en éteignant des segments, au nombre de sept. Quand les 7 segments sont allumés, on obtient le chiffre 8. Voici les 1e symboles représentés avec l'affichage à 7 segments :

Les segments sont généralement désignés par les lettres allant de A à G :



On dispose de 4 entrées (s_3, s_2, s_1, s_0) qui représenteront les 4 bits d'un nombre exprimé en base 2 et compris entre 0 et 15. On veut que ce nombre en base 2 s'affiche en base 16 sur l'afficheur.

QUESTION 15 ▶ Ecrivez la table de vérité pour le segment A, déduisez la formule logique du segment A.

QUESTION 16 ▶ Entrez cette formule dans Digital, la convention pour les formules ditital est ! pour NOT, $*$ pour AND et $+$ pour OR. Comme il y a plus de 0 que de 1 on pourra éventuellement proposer une forme normale disjonctive.

QUESTION 17 ▶ faites un circuit à partir de cette formule, puis ajouter l'afficheur sept segments de digital (`Components->IO->Display->seven-segment-display`) pour afficher le segment du haut en fonction des quatre entrées (mettre les autres segments à 0 sinon vous ne pourrez pas simuler). Le segment A est le premier à gauche sur le dessus de l'afficheur de digital.

QUESTION 18 ▶ Si vous avez le temps, faite la même chose pour les 6 autres segments et programmez un afficheur 7 segments complet sur digital. Les entrées de l'afficheur 7 segment sont agencée de la manière suivante :