Correction TP

1. Avant de commencer

Pour ces TP, on va utiliser le logiciel Logisim pour simuler facilement des circuits logiques. On part du principe que les TPs sont faits sous Linux, et que vous êtes déjà un peu autonome avec un terminal.

Créez un répertoire pour les TP de LIFASR3, puis, dans votre navigateur web, créez un favori pour la page de Logisim (Google « Logisim ») : `http://www.cburch.com/logisim/`.

Télécharger l'archive `logisim-generic-2.7.1.jar` de Logisim dans votre répertoire de TP : vous lancerez le logiciel en entrant la commande `java -jar logisim-generic-2.7.1.jar` dans un terminal.

2. Premiers exercices

Une des particularités de Logisim est de pouvoir éditer et simuler son circuit en même temps.

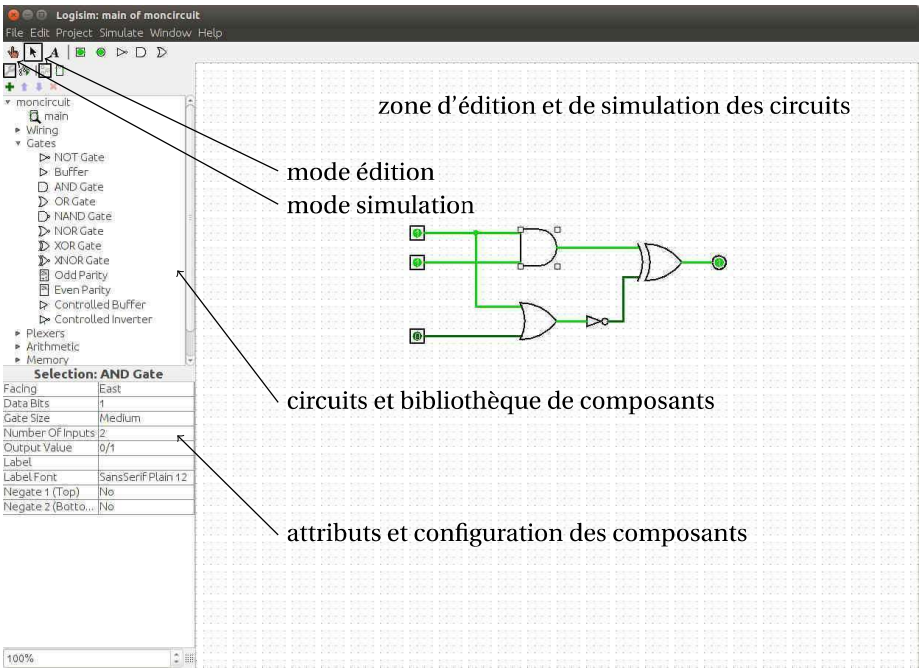


FIGURE 1 – Fenêtre de Logisim.

2.1 Mode édition

Pour utiliser le mode édition, il faut sélectionner la flèche comme indiqué en haut de la figure 1.

1. On peut alors choisir un composant dans la bibliothèque, sur la gauche. Pour l'ajouter dans son schéma, il suffit de cliquer sur le composant désiré, puis de le déposer dans le schéma.
2. Chaque composant que vous utiliserez aura des attributs modifiables dans la zone inférieur gauche de Logisim. Par exemple si l'on pose une porte AND, on peut modifier le nombre de signaux qu'elle prend en entrée (attribut `Number Of Inputs`), ou son orientation (attribut `Facing`).
3. Il est possible de faire des copier/coller d'un ou plusieurs composants (sélectionner, puis `Ctrl+C/V` ou bien `Ctrl+D`). Dans ce cas, les composants conserverons aussi tous les attributs préalablement définis.

Des éléments dont vous aurez besoin rapidement :

— Pour les entrées, l'élément `Pin` de `Wiring` dans la bibliothèque, avec l'attribut `output?=no` (voir aussi le « petit carré » dans la barre d'outils).

— Pour les sorties, l'élément `Pin` avec `output?=yes` ; (voir aussi le « petit rond » dans la barre d'outils).

— Les portes logiques sont présentes dans le répertoire `Gates` de la bibliothèque.

Notes pour plus tard :

— pour tous les composants qui ont cet attribut, vérifiez que **ThreeState=No**,

— méfiez vous « des fils qui se touchent » en particuliers entres les entrées rapprochées des portes logiques !

2.2 Mode simulation

Logisim est capable de simuler le circuit en affichant les valeurs des signaux directement sur le schéma : l'utilisateur peut définir les valeurs des bits en entrée et observer la réaction du circuit.

1. Pour utiliser le mode simulation, il faut sélectionner la main en haut à gauche de Logisim.
2. Il est possible de contrôler l'état des différentes entrées en cliquant directement dessus avec la « petite main » (forcément, uniquement en mode simulation).
3. En cliquant sur une entrée, la valeur doit alterner entre 0 en vert clair, et 1 en vert foncé.
4. Les sorties prennent les valeurs calculées par le circuit en fonction des entrées. Pour les fils qui ne transportent qu'un bit, la convention des couleurs est identique : 0 en vert clair, et 1 en vert foncé.

Exercice 2.1 : Un premier circuit pour apprendre à se servir de l'interface.

1) Ouvrez Logisim, et reproduisez le circuit ci-dessous aussi fidèlement que possible. Indications :

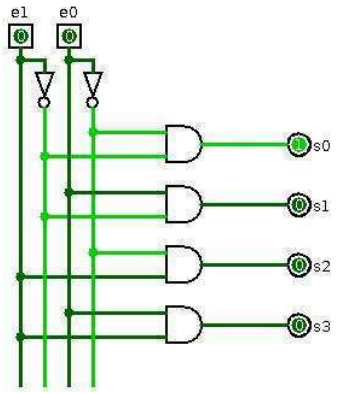
— Pour orienter les entrées (`Pin` avec `output?=no`), utilisez leur attribut `Facing`. Pour nommer les entrées, utilisez l'attribut `Label`, et pour placer le nom au bon endroit, l'attribut `Label Location`.

— Les mêmes remarquent valent pour les sorties (`Pin` avec `output?=yes`).

— Les portent AND doivent présenter deux entrées, donc utilisez `Number Of Inputs=2`.

— Faites des copiers-collers, quitte à modifier les attributs ensuite !

— Pour effacer un composant ou un fil : placez vous dessus, puis faites un click-droit et `Delete`.



2) Enregistrez votre travail dans le fichier `exo11.circ` (`File→Save`).

3) En vous mettant en mode simulation, complétez la table de vérité suivante :

e_1	e_0	s_3	s_2	s_1	s_0
0	0				
0	1				
1	0				
1	1				

e_1	e_0	s_3	s_2	s_1	s_0
0	0	0	0	0	1
0	1	0	0	1	0
1	0	0	1	0	0
1	1	1	0	0	0

4) Comment s'appelle le circuit que l'on vient de réaliser ? .....

.....

Il s'agit d'un décodeur 2 bits : la seule sortie à 1 est $s_{(e_1 e_0)_2}$; toutes les autres sont à 0.

Exercice 2.2 : Un deuxième petit circuit

Commencez par créer un nouveau projet avec `File→New`. Vous enregistrerez votre travail dans le fichier `exo12.circ`. Dans cet exercice, on expérimente autour de multiplexeurs 4 bits vers 1 bit.

1) Notre multiplexeur doit comporter :
- 4 bits e_0, e_1, e_2, e_3 en entrée,
- 1 bit s en sortie,
- les bits de sélection c_1 et c_0 également en entrée.

L'entrée dont l'indice est donné par $(c_1 c_0)_2$ doit être envoyée sur la sortie s. Indiquez quelle valeur doit prendre s en fonction des entrées du circuit dans le tableau suivant.

c_1	c_0	s
0	0	
0	1	
1	0	
1	1	

c_1	c_0	s
0	0	e_0
0	1	e_1
1	0	e_2
1	1	e_3

2) Donnez une formule pour s, en n'utilisant que les fonctions `AND`, `OR` et `NOT`. .....

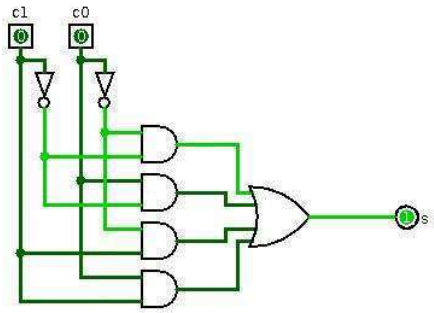
.....

$s = \overline{c_1} \cdot \overline{c_0} \cdot e_0 + \overline{c_1} \cdot c_0 \cdot e_1 + c_1 \cdot \overline{c_0} \cdot e_2 + c_1 \cdot c_0 \cdot e_3$

3) Implantez [1] votre multiplexeur dans Logisim, en n'utilisant que les fonctions `AND`, `OR` et `NOT`. Comment faire pour tester son bon fonctionnement ? .....

.....

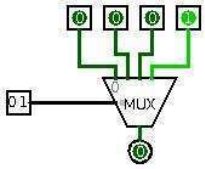
On peux faire un circuit dans ce genre là (attention à « faire propre ») :



Pour tester le bon fonctionnement du circuit, il faut, pour $k = 0,\ldots,3$: mettre $(c_1 c_0)_2 = k$, vérifier que $s = e_k$, et que s n'est pas affecté par la modification des autres entrées.

4) Maintenant, trichons : il y a déjà un composant multiplexeur dans la bibliothèque de Logisim ! Instanciez [2] un composant **Multiplexer** du répertoire `Plexers`, en mettant bien dans les attributs **Include Enable ?=No** et **Disabled Output=Zero**. Implantez ensuite le circuit suivant, et testez le :

1. Le verbe est ici de la même façon que lorsqu'on dit « je vais implanter mon algorithme de tri en C. »
2. Verbe compliqué mais d'usage courant qui veut dire ici « placez dans votre circuit ».

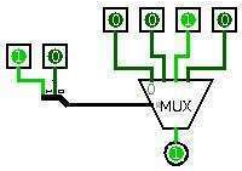


Indications :

— Pour le multiplexeur, il faut mettre **Include Enable ?=No** et **Disabled Output=Zero**, puis choisir judicieusement les attributs `Data Bits` et `Select Bits`.

— Notez que **l'entrée située sur la gauche dans le circuit regroupe 2 bits** : il s'agit d'une entrée `Pin` avec `Three-state?=No` (toujours vérifier !), `Output?=No` (logique) et **DataBits=2**.

5) On modifie le circuit précédent pour faire apparaître individuellement les deux entrées de sélection. Pour cela, on utilise un composant qui permet de séparer le paquet de deux fils de sélection du multiplexeur en deux fils : ce composant est le **Splitter** du répertoire `Wiring`. Implantez le circuit suivant, et testez le :



Un `Splitter` présente deux attributs entiers importants : **Bit Width In** et **Fan Out**. Un `Splitter` prend d'un côté un bus de (`Bit Width In`) bits, et les répartit en (`Fan Out`) petits paquets.

3. Réutiliser ses propres circuits

Tout circuit réalisé dans Logisim peut être réutilisé dans un autre circuit. Ce mécanisme permet de réutiliser le travail déjà fait et de hiérarchiser la conception des circuits (cela correspond un peu à l'idée des fonctions).

Dans cette section, on prend comme exemple celui des additionneurs binaires. Créer un nouveau projet, que vous enregistrerez dans `adder.circ`, puis travaillez toujours dans ce même fichier.

Exercice 3.1 : Un demi-additionneur

1) Commencez par créer un circuit `DA` : pour cela vous pouvez faire `Project→Add circuit`, puis donner son nom au circuit. Assurez vous que vous éditez bien le circuit `DA`, en double-cliquant sur son nom : une loupe doit apparaître dessus. Réalisez le circuit représenté à gauche sur la Figure 2.

2) On crée une instance de `DA` dans le circuit principal : éditer le circuit `main` en double-cliquant sur son nom. Ensuite, déposez une instance de `DA` comme vous l'avez déjà fait pour des composants de la bibliothèque, et complétez le circuit comme indiqué à droite sur la Figure 2. Notez que l'on retrouve les entrées (a, b), et les sorties (r_s, s), « disposition » que lors de la création du circuit : vous pouvez faire s'afficher leurs noms en passant le pointeur dessus.

3) Complétez la table de vérité ci-dessous. Pourquoi appelle-t-on ce circuit un demi-additionneur ?

a	b	r_s	s
0	0		
0	1		
1	0		
1	1		

.....

.....

.....

.....

a	b	r_s	s
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	1

Ce circuit ne permet de calculer que la somme des deux bits a et b : $2 \times r_s + s = a + b$. Il ne permet pas de prendre en compte de retenue entrante r_e : c'est pour cela qu'on l'appelle un demi-additionneur. Les esprits chagrins diront que l'on devrait plutôt appeler cela un « 2/3 d'addition-

neur », car il somme deux bits sur trois : je n'ai pas de bonne réponse à cela…

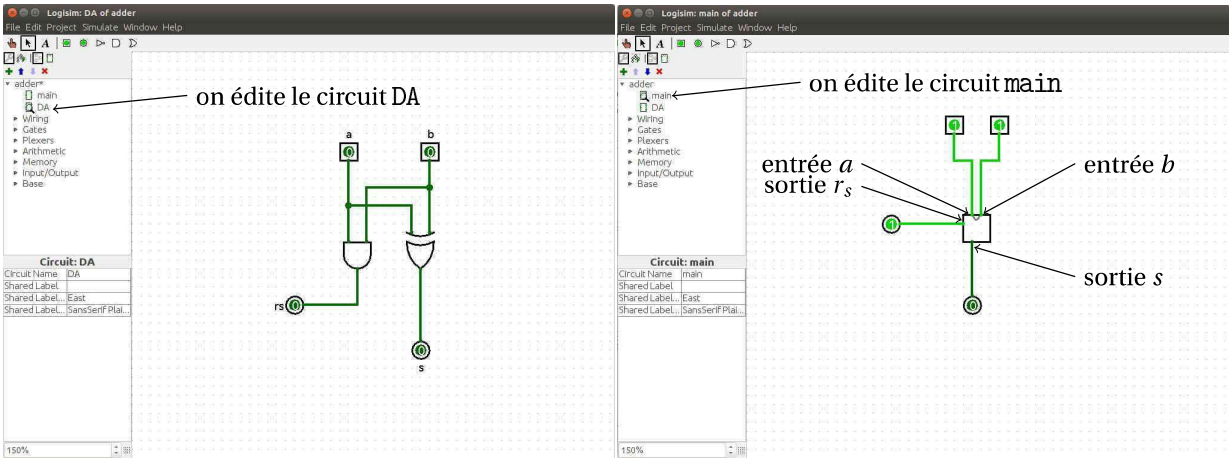


FIGURE 2 – Création (à gauche) et instanciation (à droite) du circuit DA.

Exercice 3.2 : Additionneur 1 bit complet

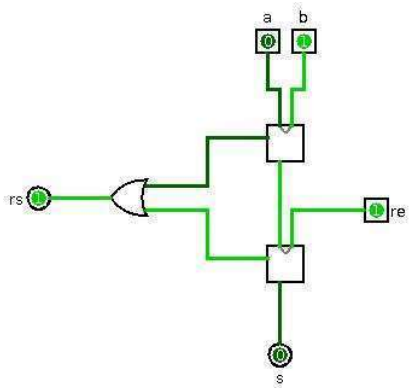
Un additionneur 1 bit complet prend en entrée trois bits a, b et r_e, et produit deux bits de sortie r_s et s tels que $2 \times r_s + s = a + b + r_e$ (au sens des entiers naturels).

1) En faisant la table de vérité d'un additionneur 1 bit complet, trouver comment réaliser ce circuit à l'aide de deux demi-additionneur et d'une porte OR.

On commence par faire la table de vérité :

a	b	r_e	r_s	s
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

On peut vérifier que $s = (a \oplus b) \oplus r_e$. On utilise deux DA :
— le premier va calculer $a \oplus b$ et un bit de retenue,
— le deuxième $(a \oplus b) \oplus r_e$ et un bit de retenue.
On vérifie ensuite dans le table de vérité que $r_s = 1$ ssi l'un ou l'autre des DA produit une retenue sortante. On en arrive donc au circuit suivant :



2) Implantez un circuit AC réalisant un additionneur 1 bit complet.

3) Veillez à bien tester votre circuit en vous basant sur sa table de vérité.

Exercice 3.3 : Additionneur 4 bits

Vous devez réaliser maintenant un additionneur 4 bits, dans un circuit `add4`, qui devra :

— prendre en deux entiers binaires a et b, chacun sur 4 bits, ainsi qu'une retenue entrante r_e sur 1 bit,

— produire en sortie l'entier binaire s sur 4 bits, et une retenue sortante r_s sur 1 bit.

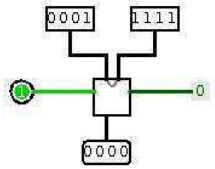
Les entrées et sorties doivent vérifier $2^4 \times r_s + s = a + b + r_e$. Vous utiliserez :

— un splitter pour obtenir les 4 bits qui composent l'entrée a, un autre pour décomposer b,

— un splitter regrouper les 4 bits qui composent la sortie s,

— 4 instances de votre circuit `AC`.

Testez ensuite votre circuit dans le `main`, en réalisant un circuit ressemblant à celui ci-dessous. Pour la retenue entrante, on utilise une constante égale à 0 (`Wiring`, `Constant`). Avec votre circuit, calculez $(0101)_2 + (0011)_2$, $(1111)_2 + (0001)_2$, $(1010)_2 + (1011)_2$. Dans chacun des cas, vérifiez à la main que vous obtenez bien le bon résultat. Si vous deviez tester tous les cas possibles, combien y en aurait-il ?

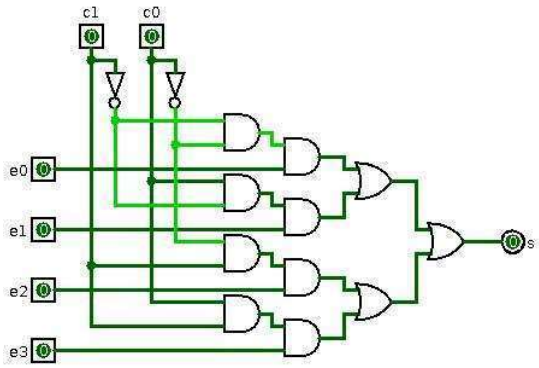


Si on suppose que la retenue entrante reste à 0, cela nous laisse avec 8 entrées de 1 bit : il y a donc en tout 2^8 entrées à tester. Cela fait déjà trop pour toutes les essayer « à la main ». Pour l'instant on ne le fait pas, mais il y a moyen de le faire avec Logisim sans trop de difficulté.

Exercice 4.2 : Autour des multiplexeurs

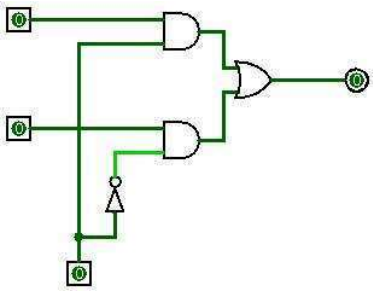
Dans cet exercice, vous n'utiliserez que des portes `AND` ou `OR` à deux entrées, et des portes `NOT`.

1) Implantez dans Logisim un circuit `mux41table` qui réalise un multiplexeur 4 bits vers 1 bits, en utilisant la technique de la table de vérité (comme on l'a déjà vu précédemment). Combien de portes logiques avez-vous utilisées ?



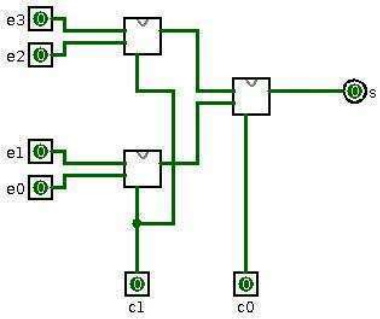
En tout, cela fait 13 portes logiques : 11 portes à deux entrées, 2 inverseurs.

2) Implantez un circuit `mux21` qui réalise un multiplexeur 2 bits vers 1 bits, en utilisant la technique de la table de vérité. Combien de portes logiques avez-vous utilisées ?



En tout, cela fait 4 portes : 3 portes à deux entrées, 1 inverseur.

3) Réalisez un circuit `mux41` en utilisant comme brique de base le circuit `mux21`. En tout, combien de portes logiques comporte votre circuit ?



En tout, cela fait 12 portes : 9 portes à deux entrées, 3 inverseurs (dont un que l'on peut supprimer car il est commun à deux `mux21`).

4) Que peut-on conclure ?

Le circuit issu de la table de vérité pour un multiplexeur 4 bits vers 1 bit semble « très naturel ». Il ne faut pas perdre de vue cependant qu'il peut être « optimisé » : on peut réduire le nombre de portes, par exemple en adoptant un point de vue plus algorithmique sur le probléme. Il doit aussi être possible d'optimiser directement les expressions booléennes... On pourrait aussi essayer de voir ce que cela donne pour un multiplexeur 8 bits vers 1 bit !