

# TP 1

## LOGISIM- Circuits combinatoires part. 1

Fichiers fournis : tp1\_comp2etu.c

### 1.1 Démarrage

Page web de LOGISIM : <http://www.cburch.com/logisim/>.

#### EXERCICE 1 ► Mise en route

On part du principe ici que les TPs sont réalisés sous Linux, même s'il est possible de les faire sous Windows (il faut dans ce cas savoir se débrouiller avec la ligne de commande de Windows, et il suffira d'adapter les commandes utilisées sous Linux).

Créez un répertoire sur votre compte pour les TP d'ARCHI.

Télécharger `logisim-generic-2.7.1.jar` sur

<http://sylvain.brandel.pages.univ-lyon1.fr/archi/logisim-generic-2.7.1.jar> (ou sur la page web de LOGISIM) et mettez-le dans votre répertoire de TP.

Vous lancerez le logiciel en entrant la commande `java -jar logisim-generic-2.7.1.jar` dans un terminal. Il est également possible de double-cliquer sur le fichier `logisim-generic-2.7.1.jar` depuis une fenêtre de navigation dans le système de fichier, mais dans ce cas il faudra préciser le chemin du répertoire dans lequel les fichiers seront sauvegardés.

Créez un répertoire pour le TP1, puis réalisez le tutoriel "Beginner's tutorial" disponible sur la page de LOGISIM

Vous n'oublierez pas *dès la création de la première porte logique* de sauvegarder votre fichier, et de taper `Ctrl+s` régulièrement. Vous ne passerez pas trop de temps sur ce Tutoriel (une demi-heure maximum).

### 1.2 Circuits combinatoires de base

**Testez soigneusement et pas-à-pas tous vos circuits!**

#### EXERCICE 2 ► Multiplexeurs/décodeurs

Dans un nouveau fichier :

- Réalisez un décodeur 3 bits vers 8 bits. Testez. *On utilisera des portes (Gates) And à 3 entrées.*
- Comparez le comportement avec un décodeur de la librairie. Testez avec une source (Pin carré) 3 bits, un afficheur (Pin rond) 8 bits, et un Splitter (Fan Out à 8 et BitWidthIn à 8) pour relier la sortie du décodeur à l'affichage 8 bits.
- Réalisez un multiplexeur 2 bits vers 1 bit. Comparez avec un multiplexeur de la librairie.

#### EXERCICE 3 ► Circuits à construire

En commençant par écrire la table de vérité des fonctions booléennes désirées, construisez les circuits suivants<sup>1</sup> :

- **Encodeur octal** : c'est un circuit à 8 entrées  $e_7, \dots, e_0$  et à trois sorties  $s_2, s_1, s_0$ . Si  $e_i$  est à 1, on veut que  $(s_2 s_1 s_0)_2 = i$ . On suppose qu'un seul des  $e_i$  est à 1.
- **Parité impaire** sur 3 bits : c'est un circuit à 3 entrées et une sortie qui vaut 1 si et seulement si le nombre des entrées à 1 est impair.

---

1. Ces deux exercices sont aussi dans le cahier de TD.

### 1.3 Dépassements en C

#### EXERCICE 4 ► Dépassement de capacité en complément à 2

Récupérez le fichier `tp1_comp2etu.c` sur la page web du cours. En supposant qu'un `char` prend un octet et un `short` 2 octets, prédites le comportement de ce programme à l'exécution. Vérifiez.

```

1  #include <stdio.h>
   #include <stdlib.h>

   int main() {
       unsigned char uc1, uc2, uc3;
6    signed char sc1, sc2, sc3;

       printf("Taille de unsigned char : %u octet(s)\n", sizeof(char));
       printf("Taille de signed char : %u octet(s)\n\n", sizeof(char));

11   uc1 = 200;
       uc2 = 60;
       uc3 = uc1 + uc2;
       printf("(unsigned char) uc1=%4d, uc2=%4d, uc1+uc2=%4d\n",
              uc1, uc2, uc3);

16   sc1 = 100;
       sc2 = 60;
       sc3 = sc1+sc2;
       printf("(signed char) sc1=%4d, sc2=%4d, sc1+sc2=%4d\n",
              sc1, sc2, sc3);

21   sc1 = -100;
       sc2 = -60;
       sc3 = sc1+sc2;
       printf("(signed char) sc1=%4d, sc2=%4d, sc1+sc2=%4d\n",
              sc1, sc2, sc3);

26   return 0;
   }

```

### 1.4 Circuits combinatoires

#### EXERCICE 5 ► Une calculatrice à PGCD

Nous allons construire un circuit qui réalise le calcul du PGCD pour les entiers positifs (8 bits). La figure 1.1 vous fournit une définition et un programme C pour ce calcul.

“En arithmétique élémentaire, le plus grand commun diviseur, abrégé en général PGCD, de deux nombres entiers naturels non nuls est le plus grand entier qui divise simultanément ces deux entiers. Par exemple le PGCD de 20 et 30 est 10. En effet, leurs diviseurs communs sont 1, 2, 5 et 10.”

Listing 1.1 – 'Afficherec.asm'

```

int gcd(int x, int y){
    while (x != y){
        if (x<y) y=y-x;
        else   x=x-y;
    }
    return x;
}

```

FIGURE 1.1 – Documentation pour le PGCD (Wikipédia)

- Quel est le pgcd de 12 et 8?
- Expliquez la différence entre cet algorithme et celui que vous connaissez pour calculer le pgcd?

Pour vous simplifier la vie, nous vous fournissons un circuit (figure 1.2) qui possède déjà les composants de données et de calcul. Nous n'aurez plus qu'à ajouter les composants de contrôle.

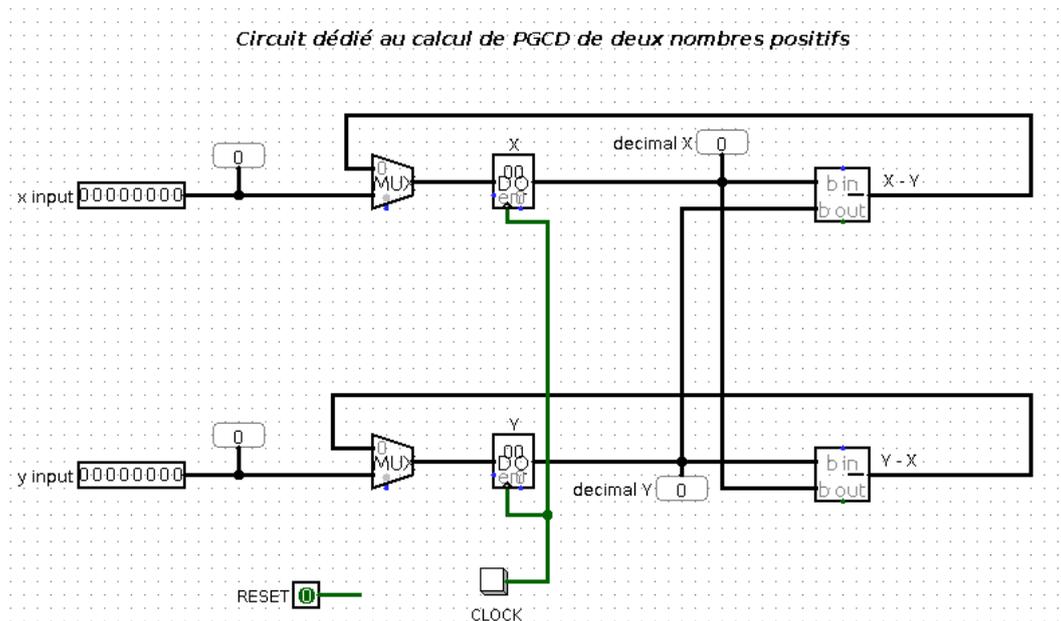


FIGURE 1.2 – Le circuit fourni

- Sur la page web du cours, téléchargez le circuit `tp1_pgcdetu.circ`.
- Observez le circuit : les entrées de gauche seront utilisées pour rentrer les valeurs initiales pour  $x$  et  $y$  (en binaire). Des sondes (Probe) décimales ont été ajoutées pour pouvoir lire ces valeurs en base 10 (ainsi que les valeurs en sortie des registres).
- Remarquez bien qu'un "tick" d'horloge effectue 1 étape dans le calcul, *le calcul n'est pas instantané!*
- Rajoutez les composants de contrôle : quand les entrées sont disponibles, l'entrée (Pin) `reset=1` doit permettre d'initialiser les registres  $X$  et  $Y$  avec ces valeurs (au premier appui sur `clock`). Ensuite, on remettra l'entrée `reset` à 0. Puis chaque appui du bouton `clock` cause l'exécution d'une étape de l'algorithme. Une fois que le PGCD est trouvé, plus aucun changement ne doit arriver. *On pourra judicieusement se poser les deux questions suivantes : quelles sont les conditions de sélection de l'entrée 1 du multiplexeur du haut (resp. du bas)? Quelles sont les conditions d'écriture de chacun des registres (entrée Enable)?*
- Vérifiez avec  $x = 42$  et  $y = 56$ . Le PGCD trouvé est?