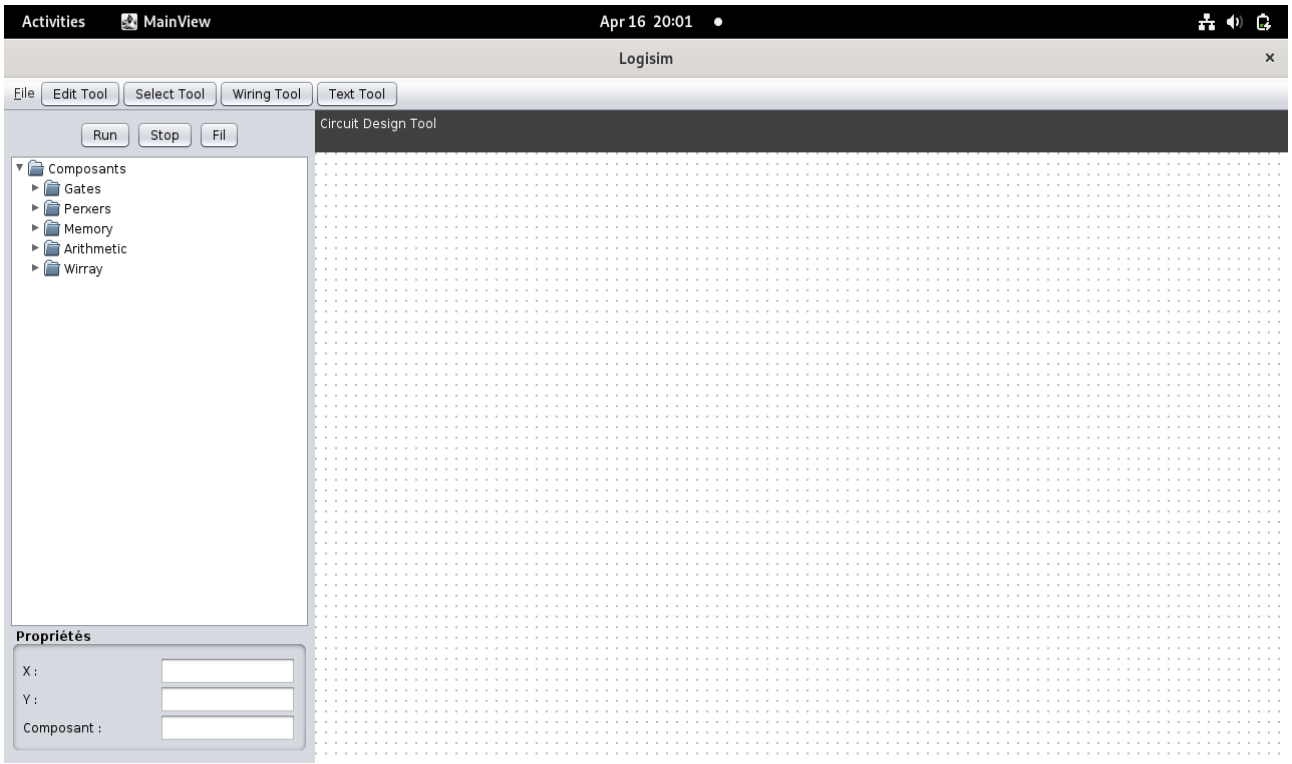
1. Introduction.

Ce document décrit le fonctionnement et l'utilisation d'un circuit logique conçu à l'aide du logiciel « **Logisim »**, par mon groupe.
Le circuit est composé de plusieurs éléments : des portes logiques (NOT, AND, OR, NAND, etc.), des générateurs d'entrée (boutons ON/OFF), ainsi qu'un générateur aléatoire (produisant des états ON, OFF, UNKNOWN ou ERROR) permettant de tester la logique du système. Une « **LED »** est utilisée comme indicateur visuel : elle s'allume ou s'éteint en fonction du résultat logique obtenu.

Nous détaillerons par la suite le fonctionnement de chaque porte et la manière dont elles interagissent au sein du circuit.



2.La présentations des trois buttons principale :

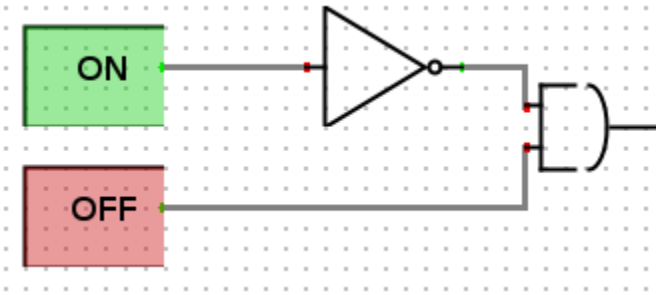
Run : Lance l'exécution, lance une méthode simulate dans le code logique, qui calcule donc lance l'évaluation des portes, pour chaque fil, son état et qui va essayer de trouver un état stable (point fixe) dans le circuit.

Stop :

Fil : Lorsqu'on clique sur le bouton *Créer un fil*, il devient possible de créer un lien (ou fil) entre deux ports. Pour cela :

1. Cliquez d'abord sur le **port de sortie** du composant **source**.
2. Puis cliquez sur le **port d'entrée** du composant **destination**.

Fil à valeur : UNKNOWN



Fil à valeur : FALSE

Fil à valeur : TRUE

Fil à valeur : ERROR

Une fois les deux ports sélectionnés, un fil est automatiquement créé entre eux.

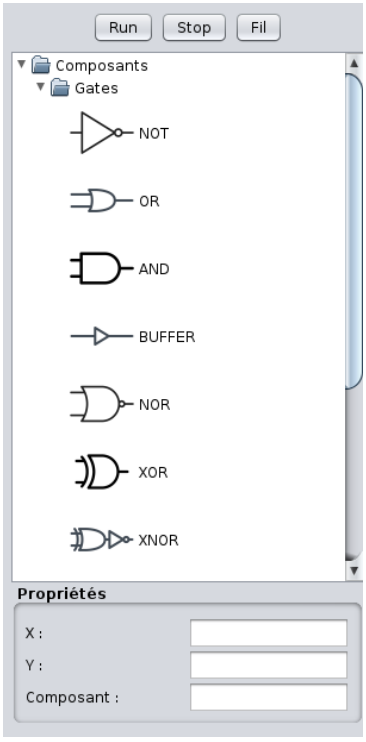
3.Présentation du bloc Priorités :

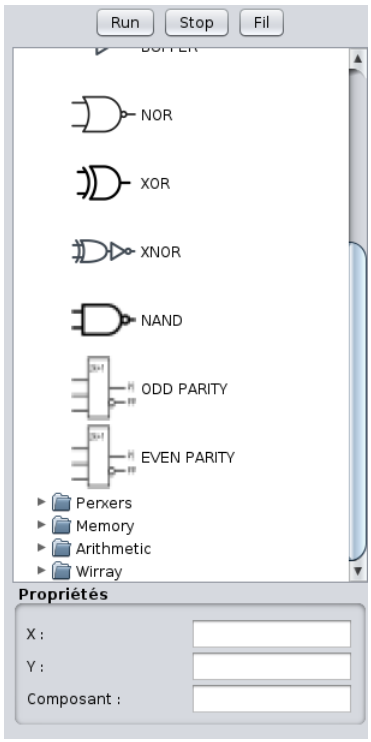
X et Y : Représentent les **coordonnées** de la position d'un composant sur le plan de travail. Ces valeurs permettent de localiser graphiquement chaque composant dans l'espace du circuit.

Composant : Représente l'**identifiant unique (ID)** attribué à chaque composant activé. Même si plusieurs composants sont du même type (ex. : plusieurs portes AND), chacun possède un ID distinct qui permet de le différencier des autres.

4.Representation des composants :

1) Les composants logiques « Gates » :
   a) La liste des portes logiques :
      i. NOT
      ii. OR
      iii. AND
      iv. BUFFER
      v. NOR
      vi. XOR
      vii. XNOR
      viii. NAND
      ix. ODD Parity
      x. EVEN Parity





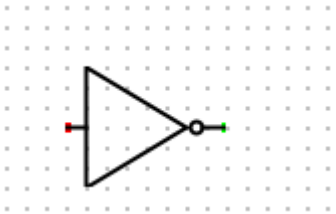
   b) La présentation de chaque porte logique :

---

i. NOT :

Fonction :

La porte **NOT** inverse l'état logique de son **entrée**.



Caractéristiques :

- **1 seule entrée**
- **1 seule sortie**

Fonctionnement :

->Si l'entrée est 0 (faux / OFF), la sortie devient 1 (vrai / ON)

->Si l'entrée est 1 (vrai / ON), la sortie devient 0 (faux / OFF)

Exemple concret :

Si on connecte un bouton ON/OFF à l'entrée :

- Quand le bouton est **désactivé (0)**, la LED connectée à la sortie **s'allume (1)**
- Quand le bouton est **activé (1)**, la LED **s'éteint (0)**

En langage logique :

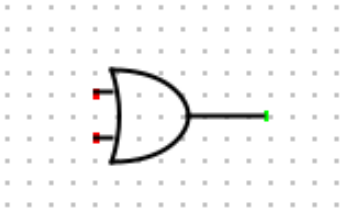
Sortie = NON(Entrée) ⇔ S = ¬E

ii. OR :

Fonction :

La porte **OR** inverse l'état logique de son **entrée**.

La porte **OR** renvoie 1 si au moins une de ses **entrées** est égale à **1**.



Elle permet de dire : "*si l'un OU l'autre est vrai, alors la sortie est vraie*."

Caractéristiques :

- **2 entrées**
- **1 seule sortie**

Fonctionnement :

-> Si aucune des deux entrées n'est active → sortie = 0

-> Si au moins une est active → sortie = 1

Exemple concret :

Quand on branche deux boutons ON/OFF :

- Si **l'un ou l'autre bouton est activé**, la LED s'allume.
- Elle s'éteint uniquement si **les deux boutons sont éteints**.

En langage logique :

Sortie = A OR B ⇔ S = A + B

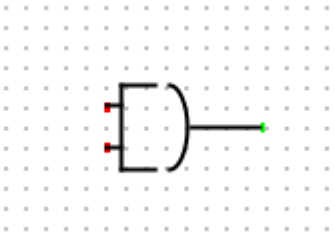
iii. AND :

Fonction :

La porte **AND** renvoie **1 uniquement si les deux entrées sont à 1**.
Elle modélise la logique "**A ET B doivent être vrais**" pour que la sortie soit vraie.

Caractéristiques :

- **2 entrées** (souvent A et B)
- **1 sortie**



Fonctionnement :

-> Si les des deux entrées sont activé → sortie = 1.

-> Si au moins une entrée est désactivée → sortie = 0.

Exemple concret :

Si on a deux interrupteurs (A et B) branchés à une porte AND.

- La LED connectée à la sortie **ne s'allume que si les deux interrupteurs sont activés.**
- Si **l'un des deux est éteint**, la LED reste éteinte.

En langage logique :

Sortie = A AND B ⇔ S = A · B

---

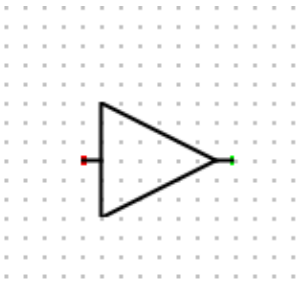
iv. BUFFER :

Fonction :

La porte BUFFER renvoie exactement l'état de son entrée à sa sortie, sans aucune modification. Elle transmet l'information telle quelle, elle modélise la logique "A doit être égal à B" pour que la sortie soit égale à l'entrée.

Caractéristiques :

- 1 seule **entrée.**
- 1 seule **sortie.**



Fonctionnement :

- Si l'entrée est activée (1) → la sortie est également activée (1).
- Si l'entrée est désactivée (0) → la sortie est également désactivée (0).

Exemple concret :

Si on a un interrupteur (A) branché à une porte BUFFER.

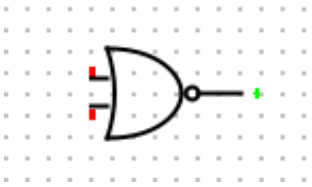
- Si l'interrupteur est **activé (ON)**, la LED connectée à la sortie s'allume (1).
- Si l'interrupteur est **désactivé (OFF)**, la LED reste éteinte (0).

En langage logique :

Sortie = A (Identique à l'entrée) ⇔ S = A.

v. NOR :

Fonction :



La porte **NOR** renvoie **1 uniquement si les deux entrées sont à 0**.
Elle modélise la logique "**A ET B doivent être faux**" pour que la sortie soit vraie.

Caractéristiques :

- **2 entrées** (souvent A et B)
- **1 sortie**

Fonctionnement :

- **Si les deux entrées sont désactivées (0)** → la sortie est activée (1).
- **Si au moins une des entrées est activée (1)** → la sortie est désactivée (0).

Exemple concret :

Si on a deux interrupteurs (A et B) branchés à une porte **NOR**.

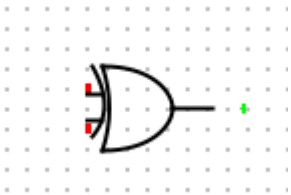
- Si **les deux interrupteurs sont éteints (0)**, la LED connectée à la sortie s'allume (1).
- Si **l'un des deux interrupteurs est allumé (1)**, la LED reste éteinte (0).

En langage logique :

Sortie = NON(A OR B) ⇔ $S = \neg(A + B)$

vi. XOR :

Fonction :



La porte **XOR** (OU exclusif) renvoie **1 uniquement si les deux entrées sont différentes**.
Elle modélise la logique "**A OU B, mais pas les deux à la fois**" pour que la sortie soit vraie.

Caractéristiques :

- **2 entrées** (souvent A et B)

- **1 sortie**

Fonctionnement :

- **Si les deux entrées sont identiques (0, 0 ou 1, 1)** → la sortie est désactivée (0).
- **Si les deux entrées sont différentes (0, 1 ou 1, 0)** → la sortie est activée (1).

Exemple concret :

Si on a deux interrupteurs (A et B) branchés à une porte **XOR**.

- Si **les deux interrupteurs sont dans le même état** (les deux éteints ou les deux allumés), la LED connectée à la sortie reste éteinte (0).
- Si **un interrupteur est activé et l'autre est éteint**, la LED s'allume (1).

En langage logique :

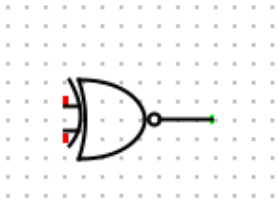
Sortie = A XOR B ⇔ $S = A \oplus B$

---

vii. XNOR :

Fonction :

La porte **XNOR** (OU exclusif négatif) renvoie **1 uniquement si les deux entrées sont identiques**.
Elle modélise la logique "**A ET B doivent être égaux**" pour que la sortie soit vraie.



Caractéristiques :

- **2 entrées** (souvent A et B)
- **1 sortie**

Fonctionnement :

- Si les deux entrées sont identiques (0, 0 ou 1, 1) → la sortie est activée (1).
- Si les deux entrées sont différentes (0, 1 ou 1, 0) → la sortie est désactivée (0).

Exemple concret :

- Si **les deux interrupteurs sont dans le même** état (les deux éteints ou les deux allumés), la LED connectée à la sortie s'allume (1).
- Si **un interrupteur est activé et l'autre est éteint**, la LED reste éteinte (0).

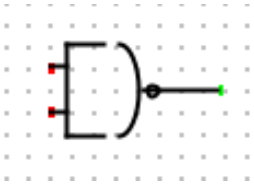
En langage logique :

Sortie = A XNOR B ⇔ $S = A \odot B$.

---

viii. NAND:

Fonction :



La porte **NAND** renvoie **0 uniquement si les deux entrées sont à 1**. Elle modélise la logique "**A ET B doivent être vrais, mais la sortie doit être l'inverse**" pour que la sortie soit vraie.

Caractéristiques :

- **2 entrées** (souvent A et B)
- **1 sortie**

Fonctionnement :

- **Si les deux entrées sont activées (1, 1)** → la sortie est désactivée (0).
- **Si au moins une entrée est désactivée (0, 1 ou 0, 0)** → la sortie est activée (1).

Exemple concret :

Si on a deux interrupteurs (A et B) branchés à une porte **NAND**.

- Si **les deux interrupteurs sont allumés (1)**, la LED connectée à la sortie s'éteint (0).
- Si **au moins un interrupteur est éteint (0)**, la LED reste allumée (1).

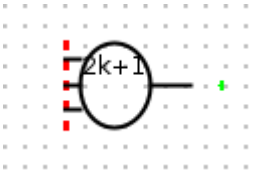
En langage logique :

Sortie = NON(A AND B) ⇔ $S = \neg(A \cdot B)$

---

ix. ODD Parity :

Fonction :



La **parité impaire (ODD Parity)** est un mécanisme de **contrôle d'erreur** utilisé pour détecter les erreurs dans les données binaires. Elle assure que le nombre total de **bits à 1** dans un ensemble de données est toujours **impaire**.

Caractéristiques :

- Vérifie si le nombre de **1** dans un ensemble de bits est **impair**.
- Ajoute un **bit de parité** à la fin de la séquence de données pour garantir cette parité impaire.

Fonctionnement :

1. **Si le nombre de 1 dans les bits est déjà impair** → le bit de parité est **0** (pour conserver l'imparité).

2. **Si le nombre de 1 dans les bits est pair** → le bit de parité est **1** (pour rendre le total impair).

Exemple concret :

Supposons qu'on a les bits suivants : 101010.

- Il y a **3 bits à 1** (ce qui est impair), donc le bit de parité sera **0**.
- La séquence complète devient alors : 1010100.

Si tu avais une séquence comme 110010 (2 bits à 1, c'est pair), le bit de parité serait **1** pour rendre le total impair :

- La séquence devient alors : 1100101.

En langage logique :

Pour une séquence de bits A1, A2, ..., An, la parité impaire est calculée comme suit :

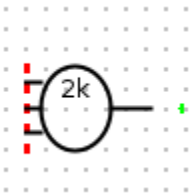
Bit de parité = (Nombre total de 1 dans A1, A2, ..., An) mod 2

=> Si le total est pair, le bit de parité est **1**, sinon il est **0**.

---

x. EVEN Parity :

Fonction :



La **parité paire (Even Parity)** est un mécanisme de **contrôle d'erreur** utilisé pour s'assurer que le nombre total de **bits à 1** dans une séquence de données est toujours **pair**.

Caractéristiques :

- Vérifie si le nombre de **1** dans un ensemble de bits est **pair**.
- Ajoute un **bit de parité** à la fin de la séquence de données pour garantir cette parité paire.

Fonctionnement :

1. **Si le nombre de 1 dans les bits est déjà pair** → le bit de parité est **0** (pour maintenir la parité paire).
2. **Si le nombre de 1 dans les bits est impair** → le bit de parité est **1** (pour rendre le total pair).

Exemple concret :

Supposons qu'on a les bits suivants : 101010.

- Il y a **3 bits à 1** (ce qui est impair), donc le bit de parité sera **1**.
- La séquence complète devient alors : 1010101.

Si tu avais une séquence comme 110010 (2 bits à 1, c'est pair), le bit de parité serait **0** pour conserver la parité paire :

- La séquence devient alors : 1100100.

En langage logique :

Pour une séquence de bits A1, A2, ..., An, la parité paire est calculée comme suit :

Bit de parité = (Nombre total de 1 dans A1, A2, ..., An) mod 2

⇨ Si le total est impair, le bit de parité est **1** (pour rendre le nombre de 1 pair). Sinon, il est **0**.