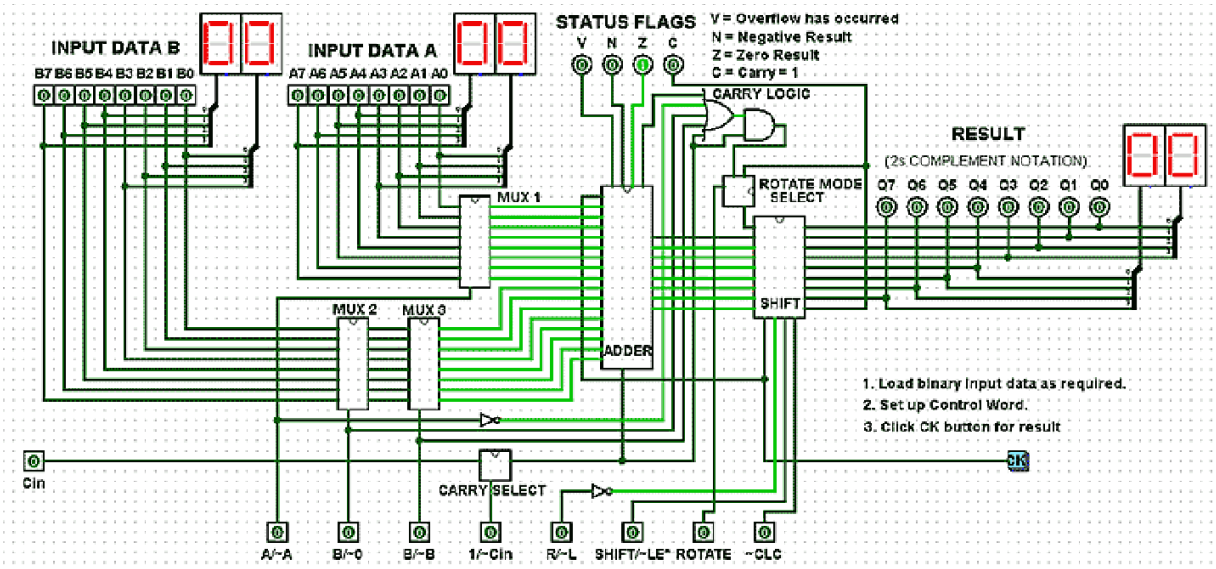
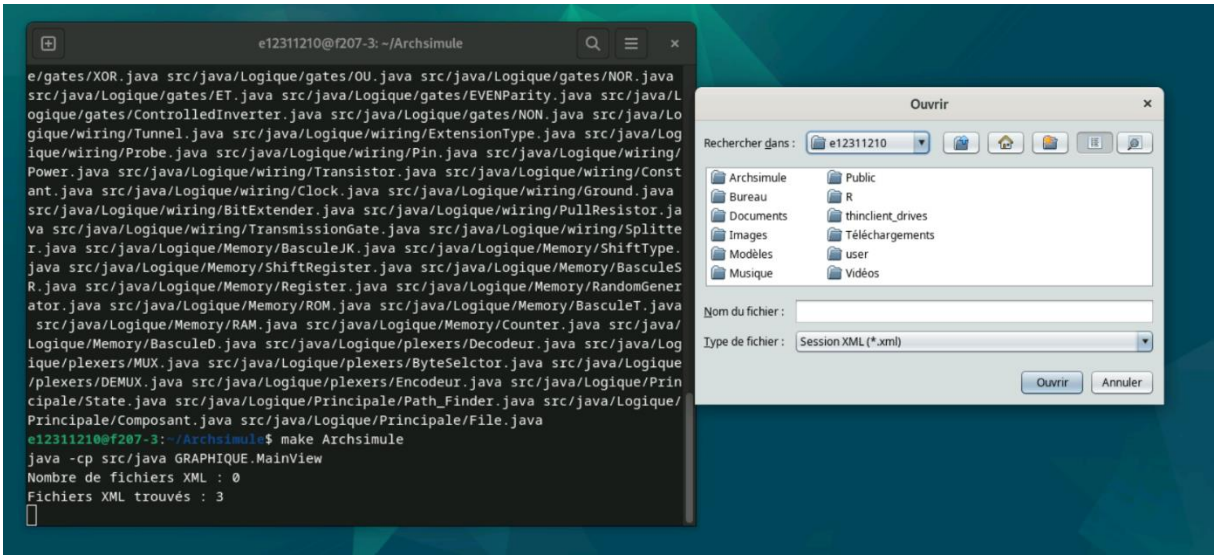
La documentation utilisateur



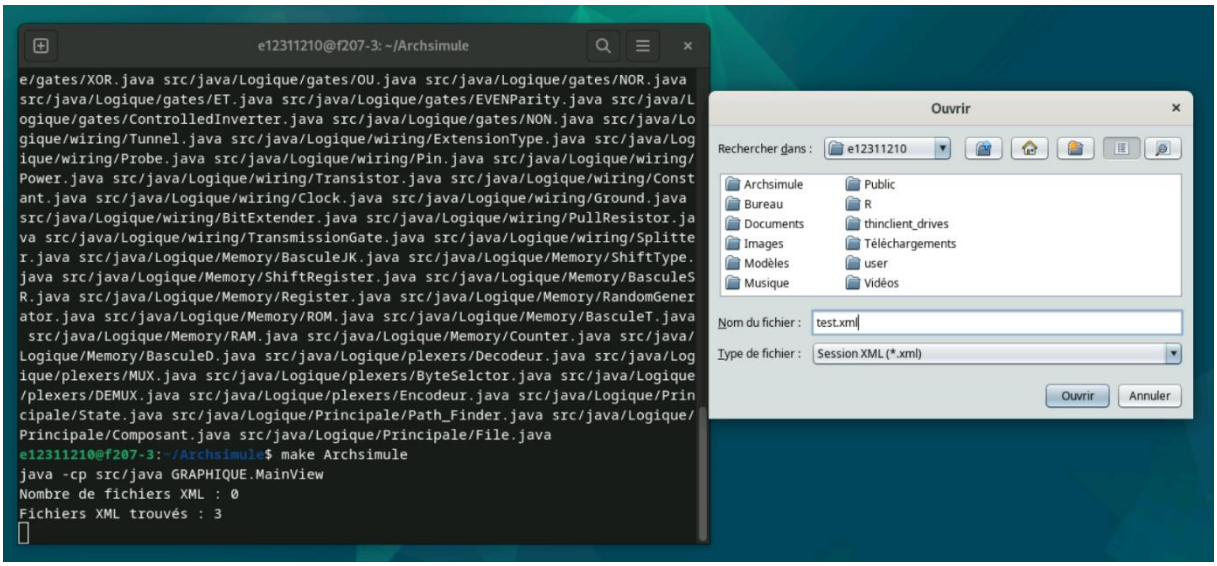
1. Introduction.

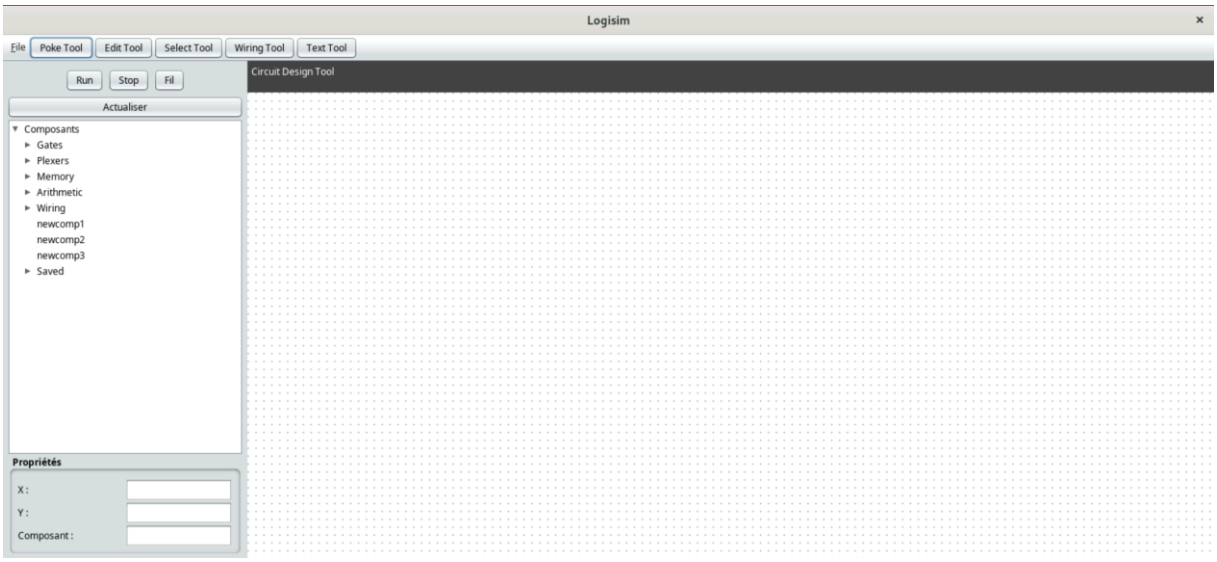
Ce document décrit le fonctionnement et l'utilisation du logiciel ArchiSimule conçu comme copie du logiciel « **Logisim** », par mon groupe.

Lancement du logiciel : en exécutant la commande suivante sur la terminale : make Archisimule



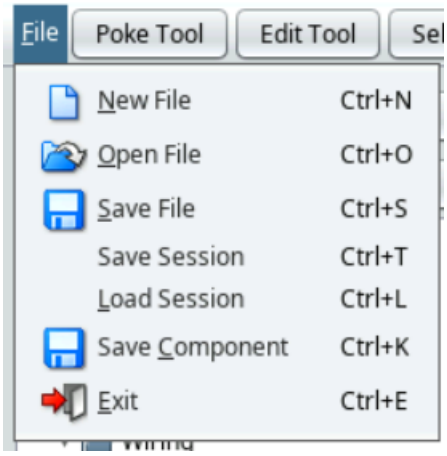
Une fenêtre s'affichera vous demandant d'indiquer ou vous voulez créer un nouveau fichier ou ouvrir un fichier déjà existant, après avoir indiqué cela et cliquer sur Ouvrir la fenêtre principale du fichier s'ouvrira, comme l'exemple suivant :





Présentations des trois buttons principale :

Le menu File :



New File : Créer un nouveau fichier.

Open File : Ouvrir un fichier.

Save File : Enregistrer un fichier.

Save session : Enregistrer une session.

Load session : Charger une session.

Save Component : Sauvegarder un composant.

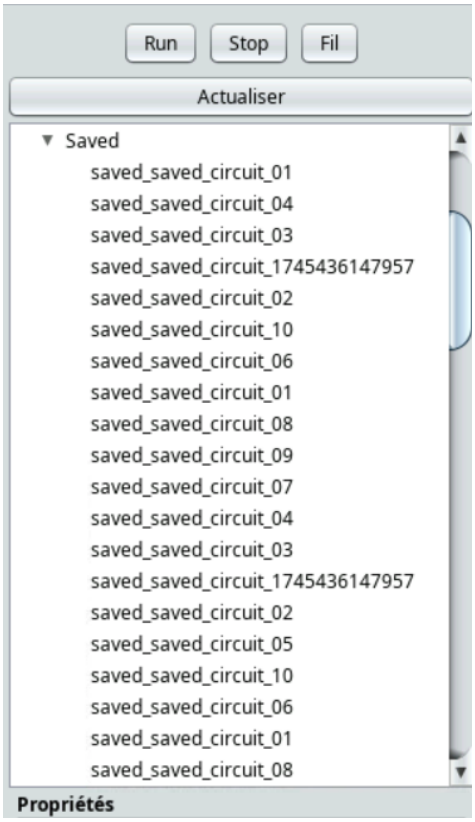
Exit : Quittez la session.

(New File, Open File) : ne font rien juste pour l'esthétique.

- Save File : permet l'enregistrement du circuit construit afin de l'utiliser après comme un composant intégré dans un autre circuit, après avoir cliqué dessus un message vous informons

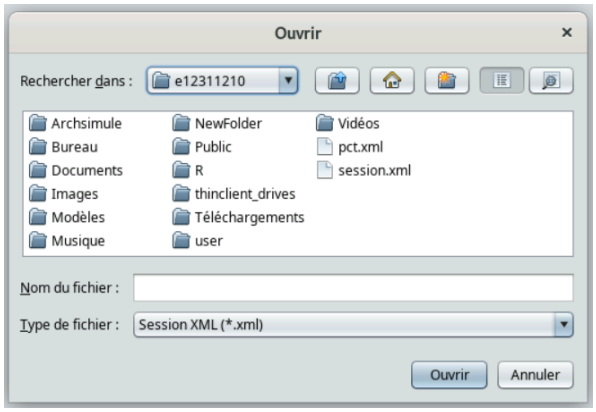
que l'enregistrement a été effectué tout en indiquant le nom attribué au circuit enregistré.

Afin de pouvoir visualiser cela faudrait cliquer sur le bouton Actualiser, et trouver votre fichier dans la catégorie Saved.

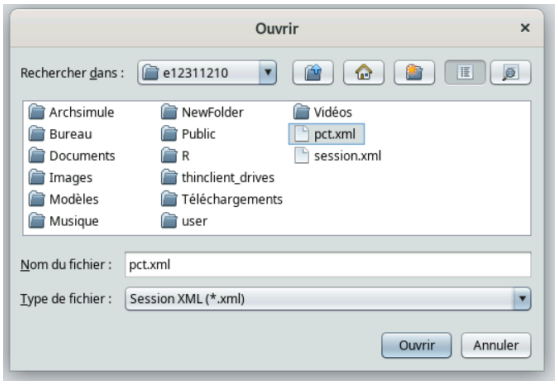


- <u>Save Session :</u> permet de sauvegarder la session entière pas le circuit, en cliquant dessus cela s'affichera sur votre écran : vous demandant de mettre le nom sous lequel vous voulez enregistrer la session et son emplacement après avoir entré ses informations et cliqué sur le button Enregistrer la sauvegarde de la session s'effectuera, et vous trouverez votre session enregistrée sous le nom entrée et dans l'emplacement spécifié.

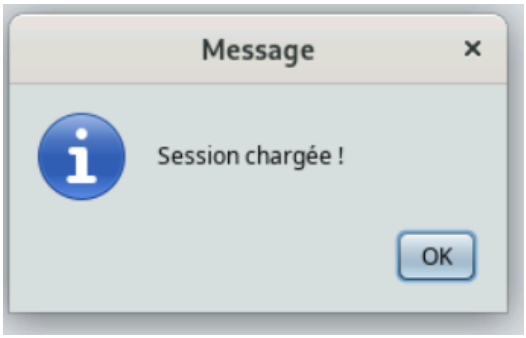
- <u>Load Session :</u> permet d'ouvrir une session déjà enregistrée, en cliquant dessus cette fenêtre s'affichera :



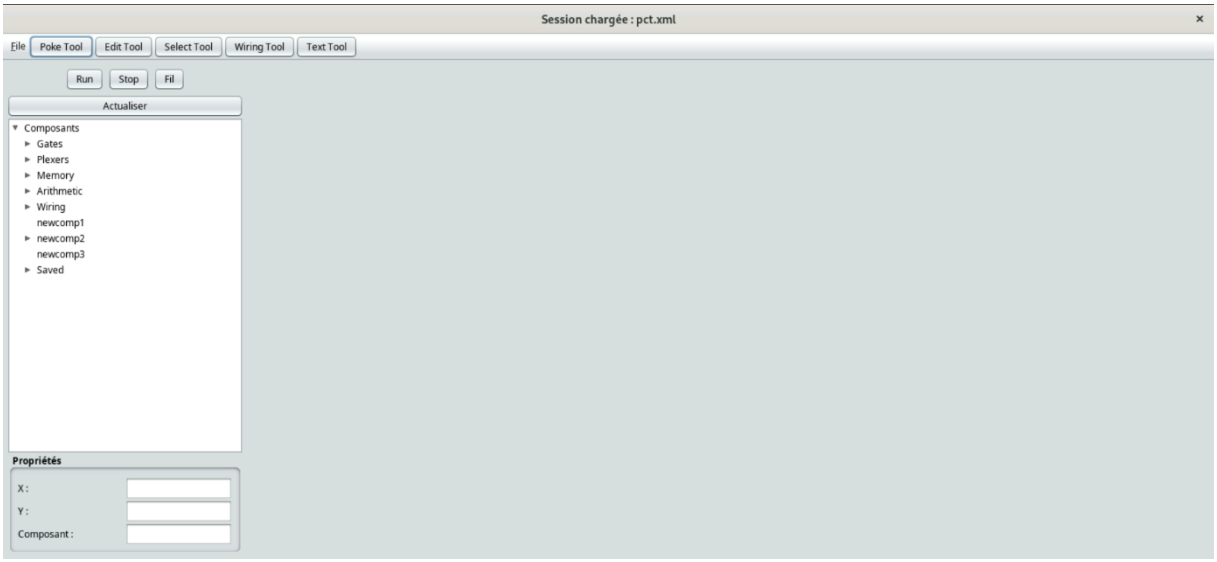
Ou vous indiquerez l'emplacement et le nom de la session à charger, comme cela :



Et après avoir cliqué sur Ouvrir, la session se chargera et un message sur l'écran comme suit vous l'indiquera :

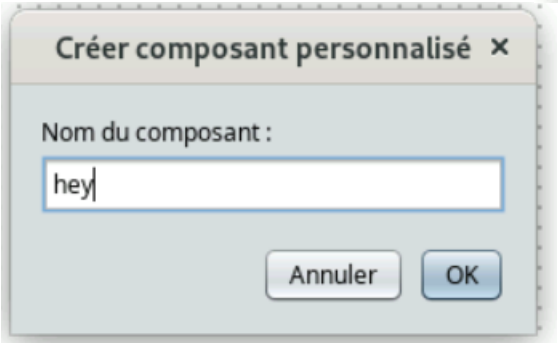


Puis vous verrez la session chargée comme suite :

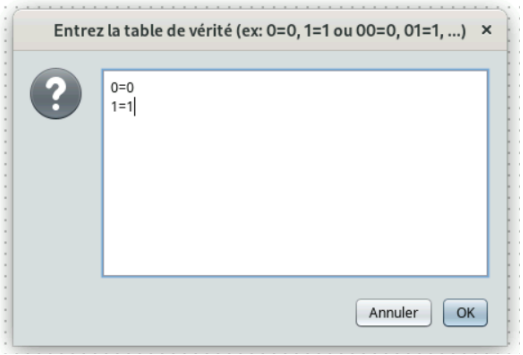


- <u>Save Component :</u> permet de créer un composant selon le nom et la logique que vous voulez en suivant ses étapes : -indiquer le nombre d'entrées de votre composant

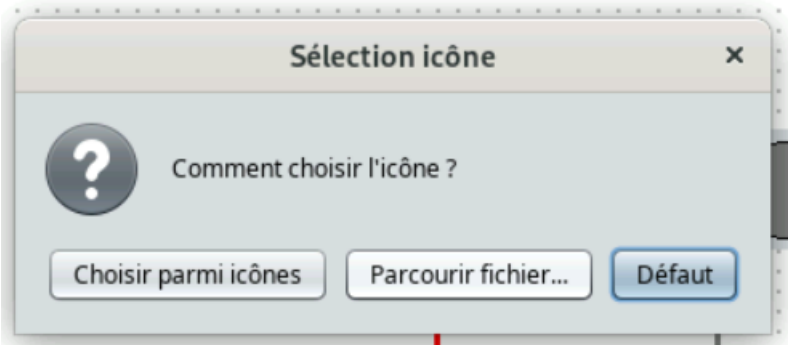
-introduire le nom du composant



-introduire la table de vérité du composant, selon quel logique le composant fonctionne.



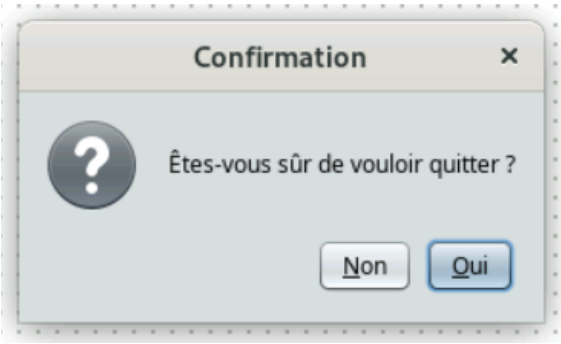
-Insérer une image qui représente le composant.



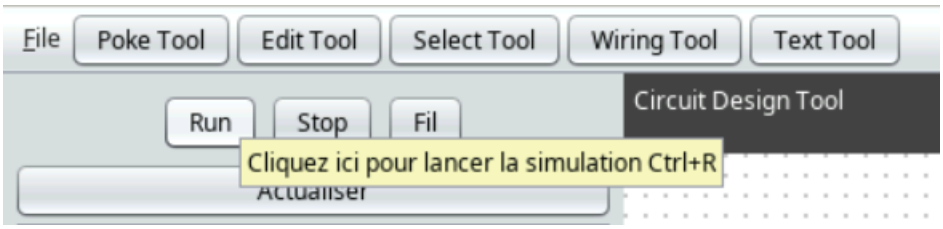
Puis ce message s'affichera afin de vous informer que le composant a bien été créé dans « custom_components2 ».

Vous trouverez le composant créé dans la catégorie newcomp2.

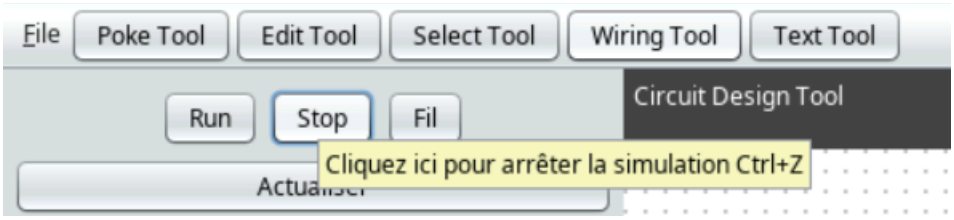
- Exit : permet de quitter la session, après avoir confirmé vouloir quitter.



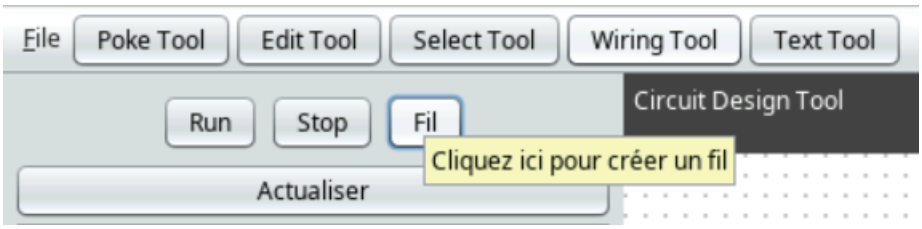
(Poke Tool, Edit Tool, Select Tool, Wiring Tool, Text Tool): ne font rien juste pour l'esthétique.



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



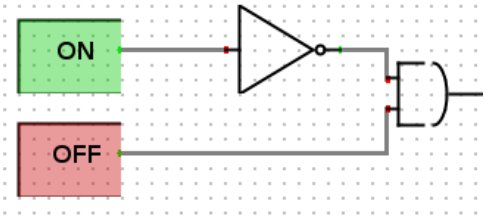
Button Stop : arrête la simulation, modifie un booléen dans le code logique qui force l'arrêt de l'évaluation des composants et arrête la transmission des valeurs par les files.



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**.

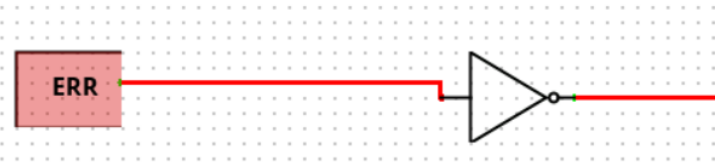
Fil à valeur : UNKNOWN



Fil à valeur : FALSE

Fil à valeur : TRUE

Fil à valeur : ERROR

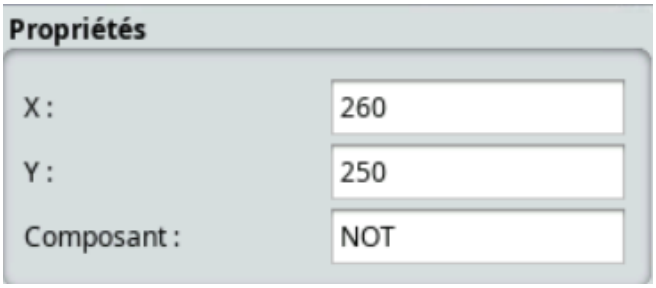


Une fois les deux ports sélectionnés, un fil est automatiquement créé entre eux.

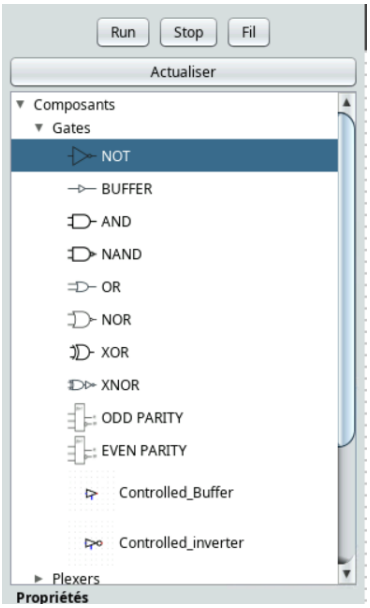
Présentation du bloc Priorités :

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.



Représentation des composants :



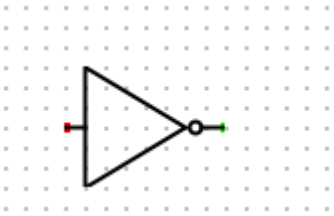
1) Les composants logiques « Gates » :
   a) La liste :
      i. NOT
      ii. OR
      iii. AND
      iv. BUFFER
      v. NOR
      vi. XOR
      vii. XNOR
      viii. NAND
      ix. ODD Parity
      x. EVEN Parity

   b) La présentation de chaque composant logique :

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)** .

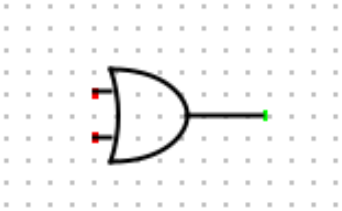
En langage logique :

Sortie = NON(Entrée) ⇔ S = non(Entée).

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 :

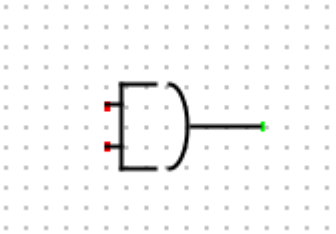
- Si **l'un ou l'autre bouton est activé**, la LED s'allume.
- Elle s'éteint uniquement si **les deux boutons sont éteints**.

En langage logique :

Sortie = A OR B ⇔ S = A + B.

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.

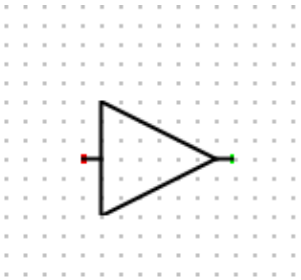
- La LED connectée à la sortie ne s'allume que si les deux interrupteurs sont activés.
- Si l'un des deux est éteint, la LED reste éteinte.

En langage logique : Sortie = A AND B S = A · B

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.

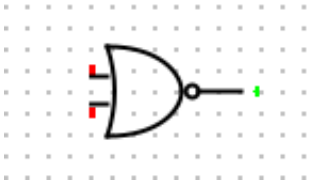
- Si l'interrupteur est **activé (ON)**, la LED connectée à la sortie s'allume (1).
- Si l'interrupteur est **désactivé (OFF)**, la LED reste éteinte (0).

En langage logique : Sortie = A (Identique à l'entrée) ⇔ S = A.

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**.

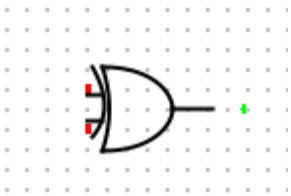
- Si **les deux interrupteurs sont éteints (0)**, la LED connectée à la sortie s'allume (1).
- Si **l'un des deux interrupteurs est allumé (1)**, la LED reste éteinte (0).

En langage logique : Sortie = NON(A OR B) ⇔ S = non(A + B).

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).

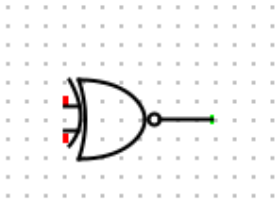
En langage logique :

Sortie = A XOR B ⇔ $S = A \oplus B$

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).

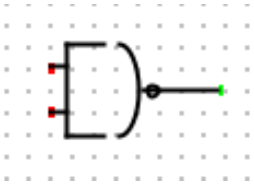
Exemple concret :

- 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 ⇔ $S = A \odot B$.

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 :

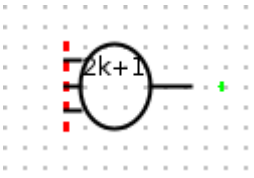
Si on a deux interrupteurs (A et B) branchés à une porte **NAND**.

- 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 = non(A · B).

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 :

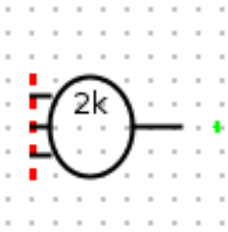
Pour une séquence de bits A1, A2, ..., An, la parité impaire est calculée comme suit :

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**.

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.

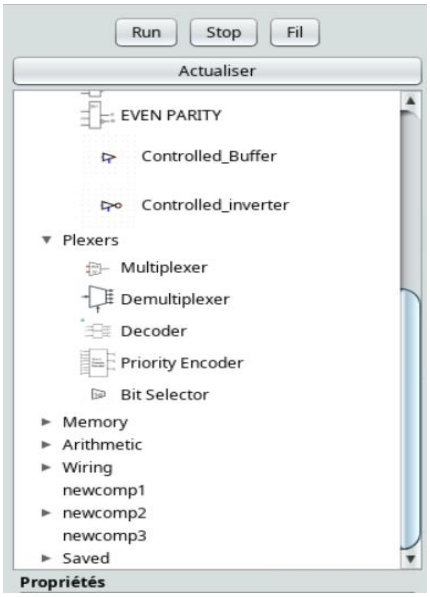
En langage logique :

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**.

2) Les composants combinatoires « Plexers »:



a) La listes :

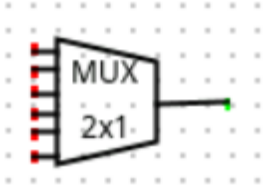
i. Multiplexeur
ii. DeMux
iii. Decoder
iv. Priority Encoder
v. Bit Selector

b) La présentation de chaque composant :

MUX - Multiplexeur

Fonction :

Le **multiplexeur** sélectionne une **seule entrée parmi plusieurs** pour l'envoyer vers **une sortie**, en fonction de **l'état des deux dernières lignes de sélection.**



Caractéristiques :

- **n** entrées de données (D_0, D_1, ..., D_{n-1})

- **1 sortie**
- **$\log_2(n)$** lignes de sélection (S_0, S_1, ..., S_k)

Fonctionnement :

- Les lignes de sélection indiquent quelle entrée est transmise à la sortie.
- Par exemple, dans un **MUX 4→1**, il y a 4 entrées (D_0 à D_3), 2 sélections (S_0, S_1), et 1 sortie.

Expression logique (MUX 4→1) :

Y = (non(S1)·non(S0)·D0) + (non(S1)·S0·D1) + (S1·non(S0)·D2) + (S1·S0·D3).

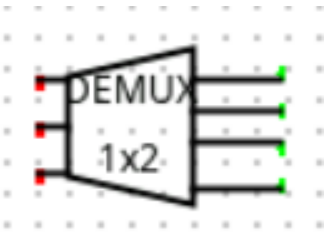
Exemple concret :

Un MUX 4→1 peut choisir entre 4 capteurs et envoyer **seulement un signal** au microcontrôleur, selon les signaux de sélection.

DEMUX - Démultiplexeur

Fonction :

Le **démultiplexeur** fait l'opposé du MUX : il envoie **un seul signal d'entrée** vers **une des nombreuses sorties**, en fonction du **dernier bit de sélection**.



Caractéristiques :

- **1 entrée**
- **n** sorties (Y_0, Y_1, ..., Y_{n-1})
- **$\log_2(n)$** lignes de sélection

Fonctionnement :

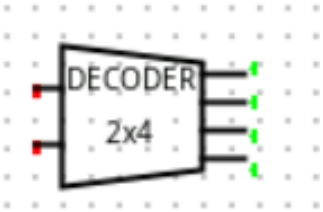
- Une seule sortie est activée selon les lignes de sélection ; les autres sont à 0.

Exemple concret :

Un DEMUX 1→4 peut diriger un signal d'alarme vers **une pièce spécifique** (sortie) en fonction du code envoyé (sélection).

Priority Encoder

Fonction :



L'**encodeur** transforme un signal **sur n lignes** (une seule active à la fois) en un code **binaire sur $\log_2(n)$ bits**.

Caractéristiques :

- **n entrées** (une seule à 1)
- **$\log_2(n)$ sorties**

Fonctionnement :

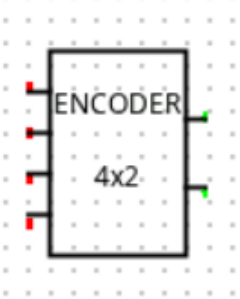
- Identifie la **position de l'entrée active** et la convertit en code binaire.

Exemple concret :

Un encodeur 8→3 convertit un signal d'un des 8 boutons pressés en un code binaire à 3 bits, pour indiquer lequel.

Decoder

Fonction :



Le **décodeur** effectue l'opération inverse de l'encodeur : il transforme un code **binaire de k bits** en un signal activant **une seule des 2^k sorties**.

Caractéristiques :

- **k entrées**
- **2^k sorties**

Fonctionnement :

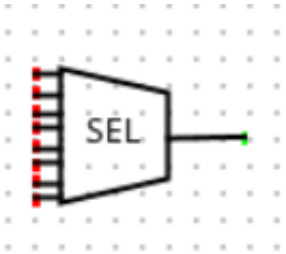
- Une seule sortie est activée, correspondant au **code binaire reçu en entrée**.

Exemple concret :

Un décodeur 3→8 allume une LED spécifique (parmi 8) selon le code binaire (000 à 111) reçu.

Bit Selector

Fonction :



Le **sélecteur de bit** extrait un **bit précis** d'un mot binaire, selon **les trois derniers bits de sélection**.

Caractéristiques :

- **1 mot binaire** d'entrée (par ex. 8 bits)
- **$\log_2(n)$** lignes de sélection
- **1 bit de sortie** (le bit sélectionné)

Fonctionnement :

- L'entrée est un mot binaire (par ex. 10110100)
- La sélection (par ex. 010) permet d'extraire le 2e bit (en partant de la droite ou de la gauche selon la convention)

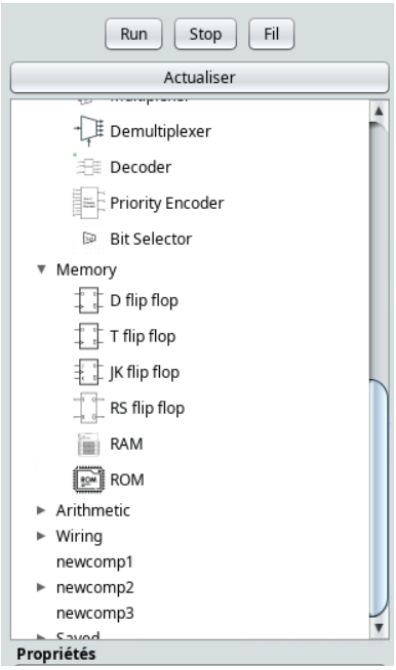
Exemple concret :

Dans un processeur, un bit sélecteur peut extraire **le bit de parité** ou un bit de **statut spécifique** dans un registre.

3) Les composants mémoires :

a) La liste :

i. D flip flop
ii. T flip flop
iii. JK flip flop
iv. RS flip flop
v. Register
vi. Counter
vii. RAM
viii. ROM



b) La présentation de chaque composant :

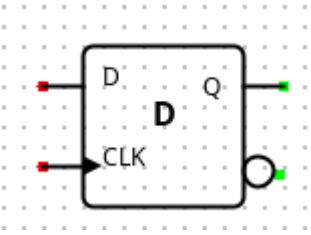
D flip flop

Entrée : D, horloge.

Fonction :

- Sur front d'horloge : Q = D.

Utilisation : Stockage synchrone simple de 1 bit.



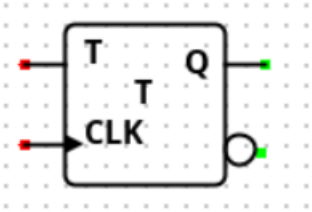
T flip flop

Entrée : T, horloge.

Fonction :

- T = 0 -> mémoire.
- T = 1 -> Q bascule (change d'état).

Utilisation : Compteurs.



RS flip flop

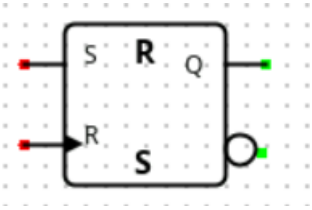
Entrées : S (Set), R (Reset)

Sortie : Q (et souvent non(Q))

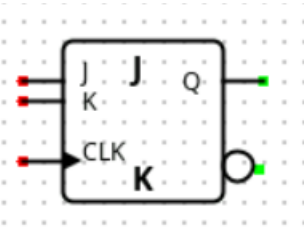
Fonction :

- S = 1, R = 0 → Q = 1 (set).
- S = 0, R = 1 → Q = 0 (reset).
- S = 0, R = 0 → mémoire (pas de changement).
- S = 1, R = 1 → **état interdit.**

Utilisation : bascule simple, pas stable pour toutes combinaisons.



JK flip flop



Entrées : J (Set), K (Reset), Horloge (CLK).
Sortie : Q (et souvent non(Q)).

Fonction :

- J = 0, K = 0 → **Mémoire** (pas de changement).
- J = 1, K = 0 → **Q = 1** (Set).
- J = 0, K = 1 → **Q = 0** (Reset).
- J = 1, K = 1 → **Q bascule** (Toggle : si Q = 0 → 1, si Q = 1 → 0).

Utilisation :

- Plus polyvalent que le RS : **pas d'état interdit.**
- Utilisé pour créer des **compteurs**, **bascules T**, et **diviseurs de fréquence.**
- Comportement **sensible au front d'horloge.**

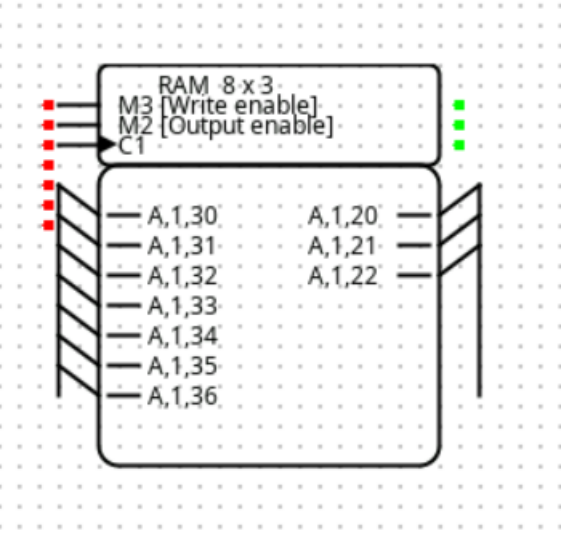
RAM

Entrées :

• Données à écrire (Word) : 3 premiers bits [0-2].

•Adresse : 3 bits secondaires [3-5].

• WriteEnable (1 bit)[6].



Sorties :

• Données lues (Word – même taille que les données) : 3bits [0-2].

Fonction :

• WriteEnable = 1 → le mot placé sur les entrées données est écrit à l'adresse donnée.

• WriteEnable = 0 → la valeur mémorisée à cette adresse est lue et placée en sortie.

• L'adresse est codée en binaire, en partant du bit de poids faible.

• La lecture se fait toujours, même si l'écriture est désactivée.

Utilisation :

• Sert à stocker et lire des données binaires.

• Utilisée dans les processeurs, microcontrôleurs et circuits logiques pour gérer des variables ou des registres.

• Permet la création de mémoires temporaires dans des circuits séquentiels.

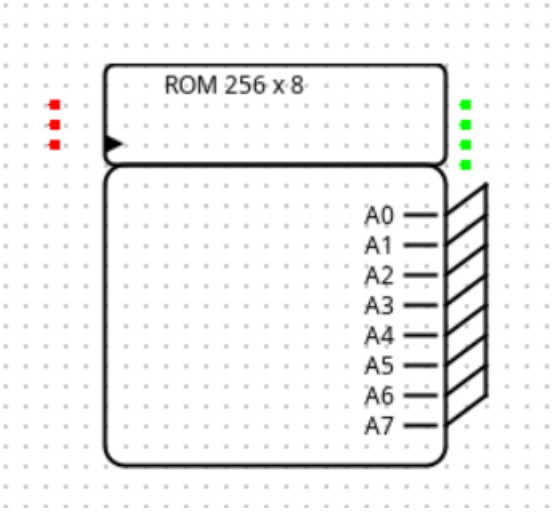
• Comportement contrôlé par un bit WriteEnable (lecture seule ou écriture + lecture).

• Initialisée à UNKNOWN à la création.

ROM

Entrées :

• Adresse mémoire: 3 bits [0-2].



Sorties :

• Donnée lue « mot mémoire » : 4bits [0-3].

Fonction :

• Lecture seulement : les données sont fixes et définies à la création.

• Aucune écriture n'est possible durant l'exécution.

• À chaque changement d'adresse, le mot correspondant est automatiquement lu et envoyé en sortie.

• L'adresse est interprétée en binaire, du bit de poids faible vers le poids fort

Utilisation :

• Sert à stocker des constantes ou des programmes fixes.

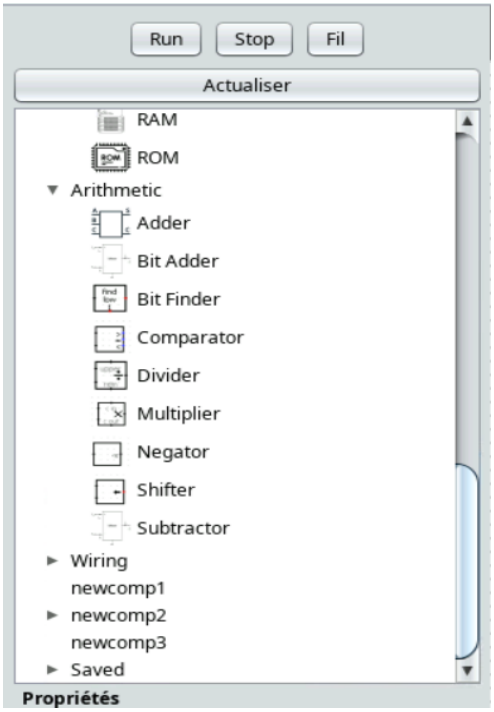
• Typiquement utilisée pour les micrologiciels (firmwares), décodage de données, ou tables de correspondance.

• Elle est remplie à l'initialisation via une matrice (State[][]) passée en paramètre du constructeur.

• Ne peut pas être modifiée pendant l'exécution.

• Fiable et stable, aucun risque d'écrasement de données.

4) Les Composants arithmétiques



a) La liste :

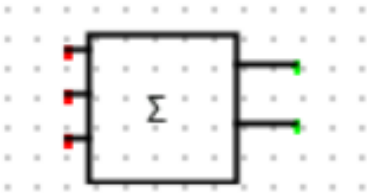
i. Adder
ii. Bit Adder
iii. Bit Finder
iv. Comparator
v. Divider
vi. Multiplier
vii. Negator
viii. Shifter
ix. Substractor

b) La présentation de chaque composant :

Bit ADDER

Fonction :

L'additionneur réalise l'addition de deux nombres binaires de un bit, il renvoie la somme et éventuellement une retenue si nécessaire, comme il permet de dire : “A ajouté à B donne un résultat en sortie”.



Caractéristiques :

- 2 entrées de 1 bit (A et B)
- 1 sortie principale (S = somme)
- 1 sortie de retenue (Cout)

Fonctionnement :

-> A = 0, B = 0 → S = 0, Cout = 0.

-> A = 1, B = 0 ou A = 0, B = 1 → S = 1, Cout = 0.

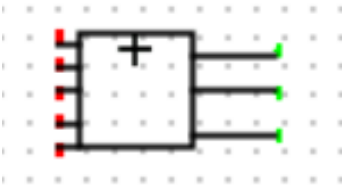
-> A = 1, B = 1 → S = 0, Cout = 1 (retenue).

Exemple concret : Additionner deux bits : 1 + 1 = 10 (S = 0, Cout = 1).

En logique binaire : $S = A \oplus B \Leftrightarrow Cout = A \cdot B$

ADDER

Fonction :



Additionne deux groupes de deux bits, chaque bit est traité par un bit adder, connecté en série.

Caractéristiques :

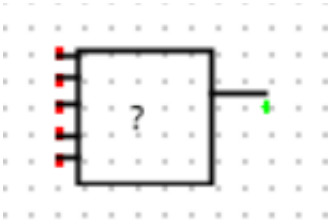
- 1 bits d'entrée A.
- 1 bit d'entrée B.
- 1 bit de retenue d'entrée (Cin).
- 1 bit de somme.
- 1 bit de retenue de sortie (Cout).

Fonctionnement : Addition binaire complète avec propagation de la retenue.

Exemple concret : A = 0011 (3), B = 0101 (5) → Résultat = 1000 (8)

Bit Finder

Fonction : « dans notre version uniquement »



Renvoie le bit a la position 2 d'un mot binaire.

Permet de dire : "Qu'elle est le 2 bit ?".

Caractéristiques :

- 1 entrée de 5 bits.
- 1 sortie indiquant la valeur de la deuxiéme entrée.

Fonctionnement : Analyse chaque bit de gauche à droite ou droite à gauche, jusqu'à trouver un '1'.

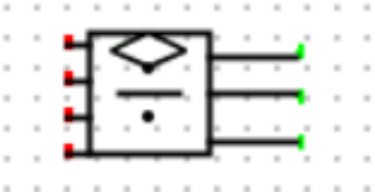
Exemple concret :

Entrée = 01000 → Sortie = 1.

Comparator

Fonction :

Compare deux valeurs de deux bits A et B puis permet de dire : “A est plus grand, plus petit ou égal à B ?”



Caractéristiques :

- 2 entrées (A, B).
- 3 sorties possibles :
    - A = B
    - A < B
    - A > B

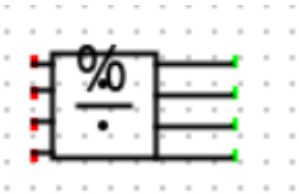
Fonctionnement : Compare bit à bit, à partir du bit le plus significatif.

Exemple concret : A = 01 (1), B = 11 (5) → Sortie : A < B.

Divider

Fonction :

Effectue une division binaire entre deux nombre de deux bits : A ÷ B, produit un quotient et un reste.



Caractéristiques :

- 2 entrées (dividende A, diviseur B).
- 1 sortie de deux bits : quotient.

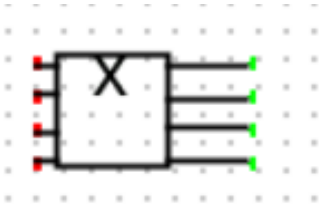
Fonctionnement : Utilise des soustractions successives ou des décalages pour calculer le résultat.

Exemple concret :

A = 10 (2), B = 00 (0) → Quotient = ERREUR.

A=01 (1), B=01 (1) → Quotient = 01(1) .

Multiplier



Fonction : Effectue la multiplication binaire de deux nombres de deux bits : A × B, permet d'additionner plusieurs versions décalées de B.

Caractéristiques :

- 2 entrées binaires A et B.
- 1 sortie sur 2bits (produit).

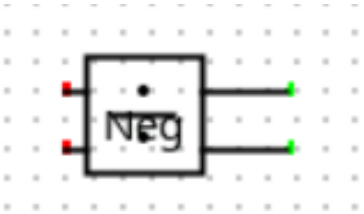
Fonctionnement :

Addition conditionnelle de B décalé selon les bits de A.

Exemple concret : A = 00 (0), B = 01 (1) → Sortie = 00 (0)

Negator

Fonction :



Produit l'inverse binaire d'un mot de deux bits (bit par bit).

Caractéristiques :

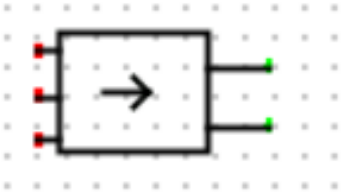
- 2 entrées binaires.
- 2 sorties de l'inversée du mot.

Fonctionnement : Chaque bit devient son opposé (0 ↔ 1).

Exemple concret : Entrée = 10 → Sortie = 01.

Shifter

Fonction :



Décale le bit d'un mot à gauche ou à droite, utilisé pour multiplier/diviser par 2, ou faire des opérations logiques.

Caractéristiques :

• 1 mot d'entrée.

• 1 signal de direction (gauche ou droite).

• 1 sortie décalée.

Fonctionnement :

Décalage à gauche = multiplication par 2.

Décalage à droite = division entière par 2.

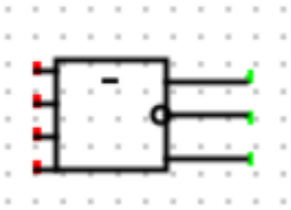
Exemple concret :

Entrée = 011 → 10.

Entrée= 010 → 00.

Substractor

Fonction :



Effectue une soustraction binaire : A – B, utilise le complément à deux pour inverser B.

Caractéristiques :

• 4 entrées « des mots de deux bits » : A et B.

• 2 sortie « un mot de deux bits » : différence.

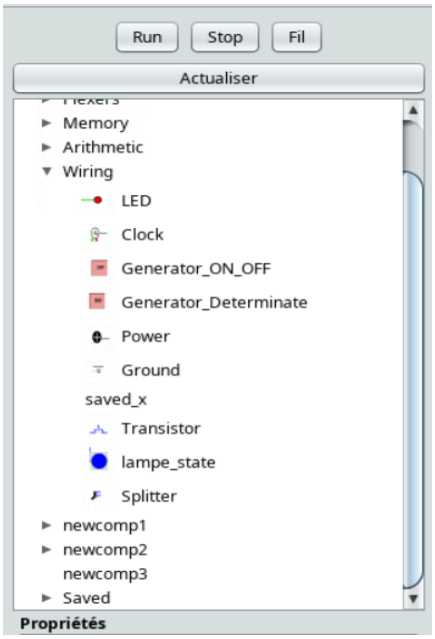
• retenue de sortie.

Fonctionnement : Calcule A + (non(B) + 1).

Exemple concret : A = 11 , B = 01 → Sortie = 10 1.

11 - 01=11+(non(01)+1)=11+(10+1)=11+01=10 1.

5) Les composants d'alimentation et de référence « Wiring » :



a) La liste :

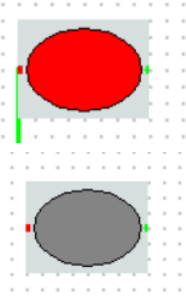
i. LED
ii. Clock
iii. Generator_ON_OFF
iv. Generator_Determinate
v. Power
vi. Ground

vii. Transistor
viii. Lampe_state
ix. Splitter

b) La représentation de chaque composant :

LED

Rôle : Affiche visuellement l'état d'un signal logique.



Fonctionnement :

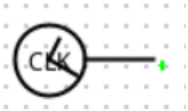
- Si l'entrée = 1 → la LED s'allume (rouge ou verte selon le thème).
- Si l'entrée = 0 → la LED reste éteinte.

Utilisation : Pour observer le résultat d'une sortie ou tester un état logique.

Clock

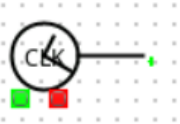
Rôle : Fournit un signal d'horloge périodique (alternance de 0 et 1 à intervalles réguliers).

Interaction utilisateur : Cliquer sur le composant fait apparaître deux boutons :



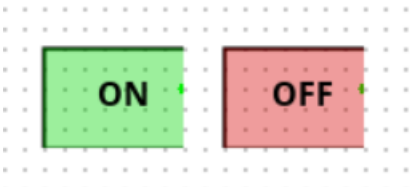
- Run : démarre l'oscillation du signal d'horloge.
- Stop : arrête le signal.

Utilisation : Cadencer des circuits séquentiels (flip-flops, compteurs, registres, etc.).



Generator_ON_OFF

Rôle : Génère une valeur binaire fixe (1 ou 0), activable manuellement.



Contrôle : Clic gauche + Ctrl : change l'état :

0 → 1 → 0 → etc.

Utilisation : Fournir un signal simple pour tester des circuits.

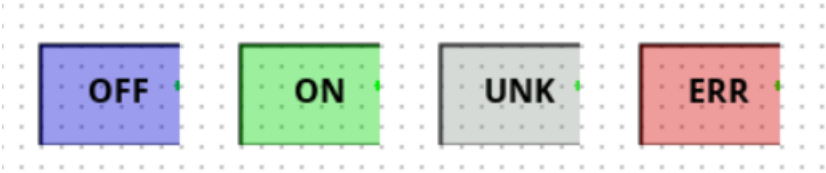
Generator_Determinate

Rôle : Génère une valeur parmi quatre états logiques déterminés.

États possibles : TRUE, FALSE, ERROR, UNKNOWN.

Contrôle : Clic gauche + Ctrl : passe à l'état suivant dans l'ordre.

Utilisation : Simuler différentes conditions logiques, y compris des cas anormaux.

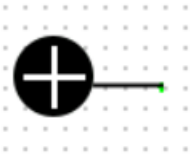


Power

Rôle : Source d'alimentation logique constante.

Valeur fournie : 1 logique (tension haute).

Utilisation : Alimenter ou forcer un composant à rester à 1.



Ground

Rôle : Masse logique.

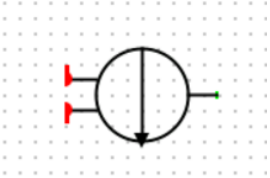
Valeur fournie : 0 logique (tension basse).

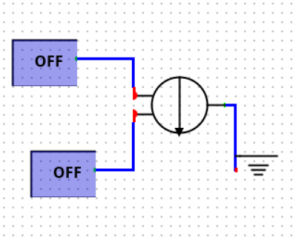
Utilisation : Point de référence ou de retour pour les signaux logiques.

Transistor

Entrées :

- Signal (entrée principale)
- Contrôle (commande ou "gate")

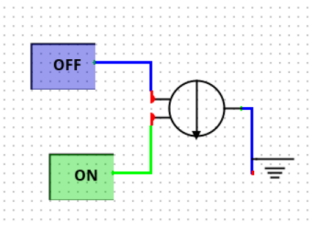




Sortie :

• Signal de sortie (la sortie suit l'entrée si le contrôle est activé)

Fonction :



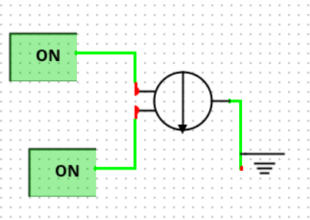
• Le transistor agit comme un interrupteur contrôlé.

• Si Contrôle = 1 (ou TRUE), alors la sortie suit l'entrée (Signal).

• Si Contrôle = 0 (ou FALSE), la sortie est coupée (souvent mise à UNKNOWN ou FALSE selon implémentation).

• Permet de bloquer ou transmettre un signal selon une condition.

Utilisation :



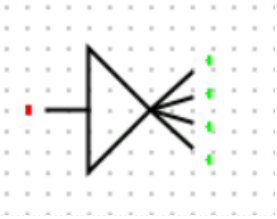
• Utilisé dans tous les circuits pour créer des portes logiques, mémoires, et bascules.

• Essentiel dans la commande de flux de courant ou de données.

• Représente le composant fondamental de tout microprocesseur.

Splitter

Entrée :

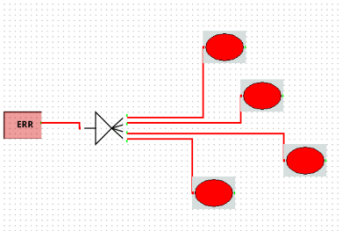


• Signal unique (1 bit)

Sorties :

• Plusieurs sorties (copie du signal d'entrée sur chacune)

Fonction :

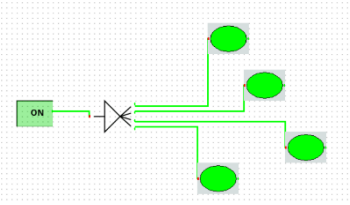


• Reçoit un seul signal en entrée et le duplique sur plusieurs sorties.

• Tous les fils de sortie reçoivent exactement le même état que l'entrée.

• Ne modifie pas la valeur du signal, il la réplique simplement.

Utilisation :



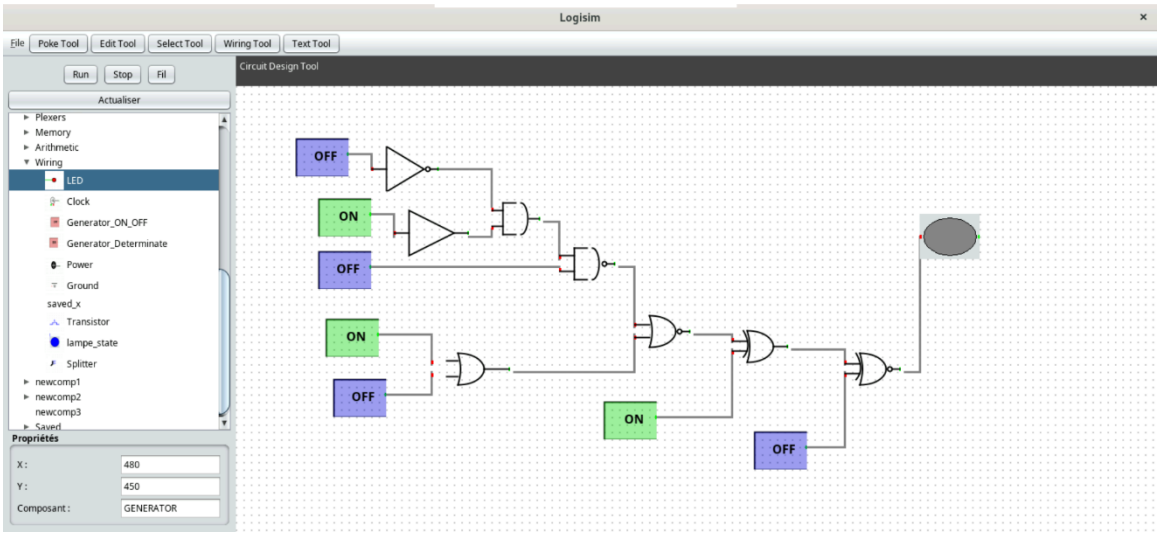
• Sert à répartir un même signal vers différents composants (comme un fil qui se divise).

• Très utile pour connecter une même horloge, un même bit de contrôle ou une tension à plusieurs endroits du circuit.

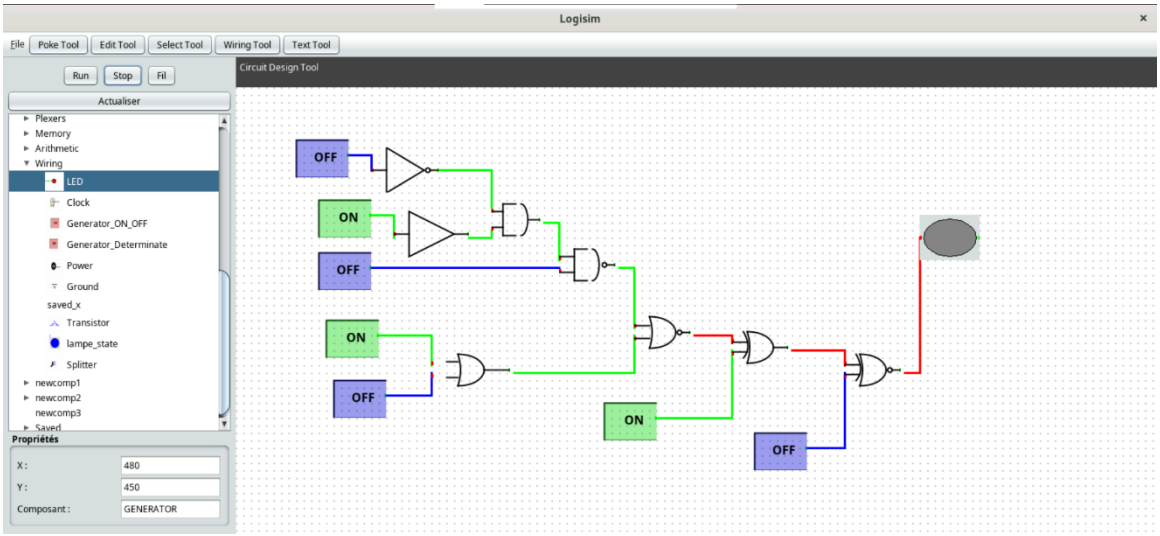
• Permet de réduire le câblage visuel en séparant un fil unique en plusieurs branches.

Exemple d'un circuit construit dans le logiciel :

1. Avant de lancer la simulation :



2. Après la simulation :



On espère que ce guide vous a permis de comprendre l'utilisation des composants logiques du logiciel et leur configuration dans différents contextes.

Merci d'avoir consulté la documentation utilisateur de ArchiSimul, nous espérons qu'elle vous a été utile pour prendre en main les différents composants et fonctionnalités disponibles.

Pour toute question, suggestion ou problème rencontré, contacté notre responsable IHM qui reste à votre disposition : syndia.toutou@edu.univ-paris13.fr

Consultez régulièrement notre dépôt git pour : Mises à jour logicielles, nouveaux tutoriels, aides visuelles et vidéos et pour pour lire le README : https://github.com/Anis-Bouda/the_crew .

Bon travail avec notre ArchiSimul !