Table des matières

Table des figures

Introduction

L'objectif de ce travail est de réaliser un simulateur logique de circuits digitaux. qui va permettre le calcul des valeurs de sorties en fonction de celles d'entrées en simulant le fonctionnement des différentes portes logiques, choisit par l'utilisateur sous forme d'un fichier texte.

Le simulateur est constitué de quatre parties :

- Lecture du fichier texte qui contiens la description du circuit au niveau des portes logiques.
- Lecture du fichier texte qui contient les valeurs en entrée du circuit.
- Après la lecture du fichier on obtiens un modèle du circuit.
- Simulation logique : on calcul les valeurs de chaque sortie du circuit en fonction des entrées.

La figure suivante décrit l'architecture de notre simulateur.

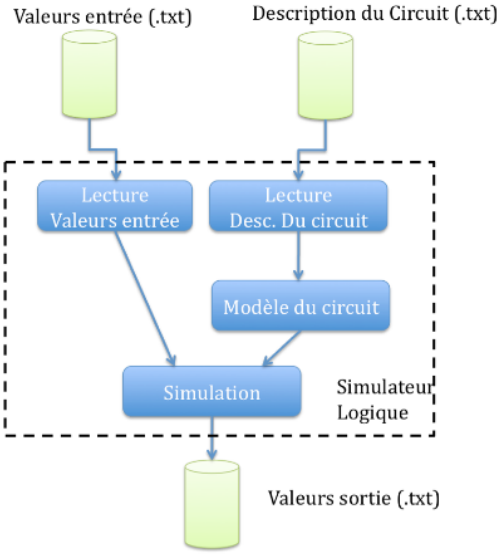

FIGURE 1.1: Schéma explicatif du simulateur logique

Modélisation des portes logiques

Dans cette partie on va créer une hiérarchie de classes permettant de modéliser l'ensemble des portes logiques, sachant que chaque porte logique peut avoir une ou plusieurs entrées et au maximum une seul sortie. ce qui va représenter un modèle du circuit qu'on veut simuler.

Pour cela on va créer une classe appelée **portesGen** et elle est constituée des fonctions suivantes :

- **void Set_input(int val, int indice)** :pour affecter une valeur à une entrée de la porte d'indice '**indice**'.
- **int Calculate_output()** :pour calculer la valeur de la sortie.
- **void print_info()** : afficher à l'écran le nom de la porte plus les valeurs courantes de chaque entrée et de la sortie.

Ces trois fonction vont être obligatoirement précédées par le mot clé **virtual**, pour permettre aux classes filles de les utiliser chacune d'une façons approprié.

les classes filles vont hériter de la classe mère **portesGen**, vont représenter les différents types de portes logique possibles (And, Or, Xor, Not). chaque porte va avoir comme variables :

- **int m_nbreEntree** : représente le nombre d'entrées de chaque porte.
- **int *m_entrees** : c'est le vecteur d'entrées.
- **int m_s** : la sortie correspondante.

Modèle du circuit

Le modèle du qu'on veut circuit est composé d'un certain nombre d'entrée primaire **ep**, un nombre de porte logique **sp** et enfin d'un nombre de sortie primaire **pl**. (fig.3.1).

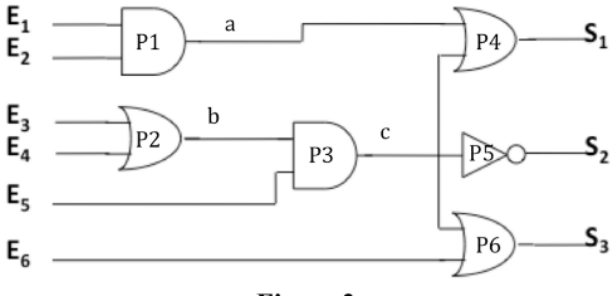

FIGURE 3.1: Exemple d'un circuit logique.

On peut représenter ce circuit par un graph, tel que chaque sommet du graphe peut être une entrée primaire, une porte logique ou une sortie primaire. Les arcs du graphe correspondent aux connections entre entrée primaire/sortie d'une porte vers l'entrée d'une porte logique/sortie primaire. (fig.3.2).

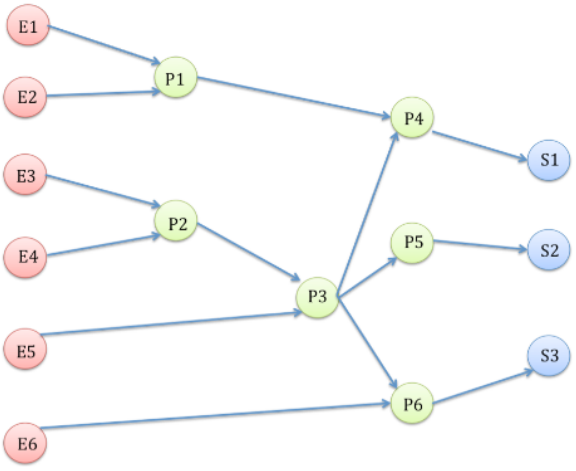

FIGURE 3.2: Le Graphe correspondant du circuit logique.

Pour cela, et a fin de calculer la matrice d'adjacence du graph correspondant, on va créer une classe appelées **Graph**, les différentes variables que contient cette classe sont les suivants :

- **int m_ep** : nombre d'entrées primaires.
- **int m_sp** : nombre de sorties.
- **int m_pl** : nombre de portes logiques.
- **string *m_NodeName** : vecteur qui contient les sommets du graph.
- **string *m_OutName** : vecteur qui stocke les entrées secondaires.
- **string **m_LL** : matrice qui contient la description du circuit logique (les informations de chaque porte, ses entrées et la sorties correspondantes) elle est similaire au fichier texte qui contient la description du circuit.
- **int m_LL_Ncol** : nombre de colonnes de la matrice m_LL.
- **int **MAdj** : matrice d'adjacence du graph.
- **int *valSorties** : vecteur des sorties calculées (s1 s2 ... sN).
- **portesGen **m_ptr** : est un vecteur de pointeurs vers des objets de types portesGen.
- **Vecteur *m_vect** : un objet de type Vecteur pour la lecture des valeurs d'entrées.

quant aux fonctions utilisées dans cette classe :

- **calcule_ep(), calcule_sp(), calcule_pl(), calcul_largeur()** : pour calculer des variables correspondantes. et les autres pour le remplissages des vecteurs.

Lecture du circuit

Le fichier texte qui contient la description du circuit a la syntaxe suivante :

```
INPUT E1 E2 … EN ;
OUTPUT S1 S2 … SK ;
Type PorteLogique1 SP EP1 EP2 … EPZ1 ;
Type PorteLogique2 SP EP1 EP2 … EPZ2 ;
…
Type PorteLogiqueW SP EP1 EP2 … EPZW ;
```

FIGURE 4.1: syntaxe du fichier texte qui contient la description du circuit.

si on applique ce principe au circuit précédent on aura le fichier suivant :

```
INPUT E1 E2 E3 E4 E5 E6 ;
OUTPUT S1 S2 S3 ;
and P1 a E1 E2 ;
or P2 b E3 E4 ;
and P3 c b E5 ;
or P4 S1 a c ;
not P5 S2 c ;
or P6 S3 c E6 ;
```

FIGURE 4.2: Exemple fichier texte qui contient la description du circuit.

On doit écrire une ou plusieurs fonctions membres de la classe graphe, qui sont capable de lire le fichier texte qui contient le circuit afin de construire le graphe équivalent.

ses fonctions sont les suivants :

- **calcule_ep(), calcule_sp(), calcule_pl(), calcul_largeur()**

Lecture valeurs en entrées

Le fichier texte, qui contient les valeurs logiques à appliquer en entrée au circuit, est défini avec la syntaxe suivante :

```
INPUT E1 E2 … EN ;
OUTPUT S1 S2 … SK ;
Type PorteLogique1 SP EP1 EP2 … EPZ1 ;
Type PorteLogique2 SP EP1 EP2 … EPZ2 ;
…
Type PorteLogiqueW SP EP1 EP2 … EPZW ;
```

FIGURE 5.1: syntaxe du fichier texte qui contient les valeurs logiques.

Voici un exemple du fichier texte qui contient les valeurs logiques.

```
INPUT E1 E2 E3 E4 E5 E6 ;
OUTPUT S1 S2 S3 ;
and P1 a E1 E2 ;
or P2 b E3 E4 ;
and P3 c b E5 ;
or P4 S1 a c ;
not P5 S2 c ;
or P6 S3 c E6 ;
```

FIGURE 5.2: Exemple du fichier texte qui contient les valeurs logiques.

On doit écrire une classe pour la lecture est le stockage de l'ensemble des vecteurs en mémoire. cette classe est appelée **Vecteur**, cette classe est celle qui s'occupe du calcul des vecteurs d'entrées pour les les stocke dans la mémoire dans la matrice ¡¡m_v¿¿, elle contient les variables suivantes :

- **int **m_v** : matrice où le fichier texte est convertit en entier et stocké en mémoire.
- **int m_nbreDeVecteurs** : nombre de lignes de cette matrice.
- **int m_nbreDentrees** : nombre de colonnes de cette matrice.

les fonctions de cette classe sont les suivantes :

- **void calcul_nbreDeVecteurs()** : pour calculer le nombre de vecteurs.
- **void set_nbreDentrees(int value)** : donner le nombre d'entrées a cette fonction (c'est le mème nombre d'entrées de la classe Graph 'm_ep').
- **int get_value(int i,int j)** : est pour faire sortir le nombre de vecteurs et l'utiliser a l'extérieur des fonctions de la classe.
- **int get_nvect()** : est faite pour obtenir la valeur de n'importe quelle entrées de l'extérieur de la classe.
- **void remplissage_m_v(void)** : pour remplir la matrice m_v.

Simulation Logique

Le principe de la simulation logique. s'agit d'affecter chaque vecteur aux entrées primaires de notre circuit et de calculer les sorties correspondantes. Dans ce cas, on appliquera l'algorithme DFS partir de chaque entrée primaire. Pendant l'application de la DFS on traverse plusieurs sommets jusqu'aux sorties primaires. Chaque fois qu'on traverse un sommet on calcule la valeur de la sortie (en utilisant la fonction Calculate_output) et on propage la valeur de la sortie vers les portes auxquelles la sortie est connecté (en utilisant la fonction Set_input). (fig.6.1).

```
Pour chaque vecteur V {
  Affecter V aux entrées primaires du circuit
  Pour chaque entrée primaire EP {
    DFS (EP)
  }
  Affichage de la valeur des sorties
}
```

FIGURE 6.1: Principe de l'algorithme de DFS.

Pour cela on doit combiner tout les travaux faites dans les parties précédentes (donc tout les classes : portesGen, Graph et Vecteur), cependant tout le calcul va se faire dans la classe **Graph** car c'est dans cette classe où on calcul toutes les variable nécéssaires pour la céation du simulateur logique, on aura juste besoin de récupérer des informations de la classe **Vecteur** et la classe **porteGen**, pour cela on doit juste déclarer deux pointeurs sur les deux classes précédentes dans la classe **Graph**.