C++

Qucs-RFlayout
Générateur de typons pour les schémas hyperfréquences de Qucs

Auteur :
Thomas LEPOIX

MASTER 2 Systèmes Embarqués

E.S.T.E.I.
École Supérieure des Technologies Électronique, Informatique, et Infographie
Département Systèmes Embarqués

2 mai 2019

Table des matières

Chapitre 1

Introduction

1.1 Circuits imprimés hyperfréquence

En haute fréquence les composants analogiques discrets utilisés en électronique classique sont inappropriés du fait des réactances linéiques qui ne sont alors plus négligeables et qui faussent complètement le fonctionnement des circuits. Il existe plusieurs technologies alternatives appropriées aux hautes fréquences qui tirent parti de ces réactances linéiques. Il s'agit de concevoir des composants dits "répartis" à partir de pistes de circuit imprimé aux dimensions particulières et calculées pour leur conférer une impédance donnée.

Cela donne aux circuits imprimés des appareils hyperfréquence des allures tout à fait caractéristiques, aux pistes très géométriques.

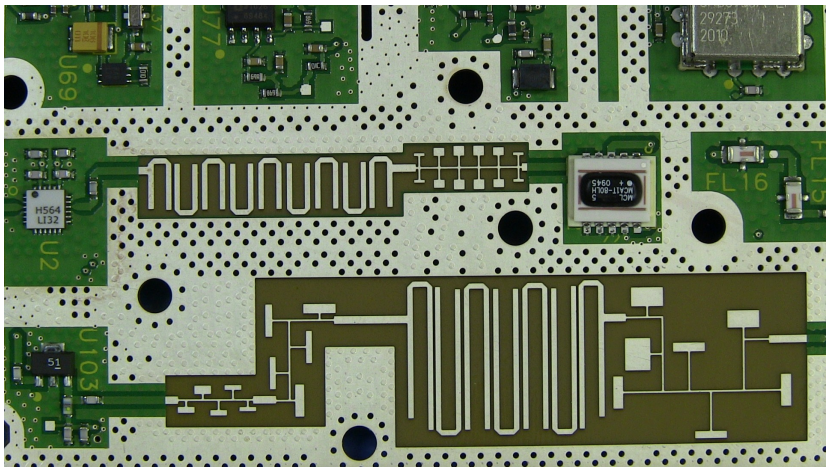


Figure 1.1 – Exemple d'un circuit imprimé hyperfréquence

Ci-dessous un aperçu des technologies les plus courantes pour concevoir des circuits imprimés hyperfréquences :

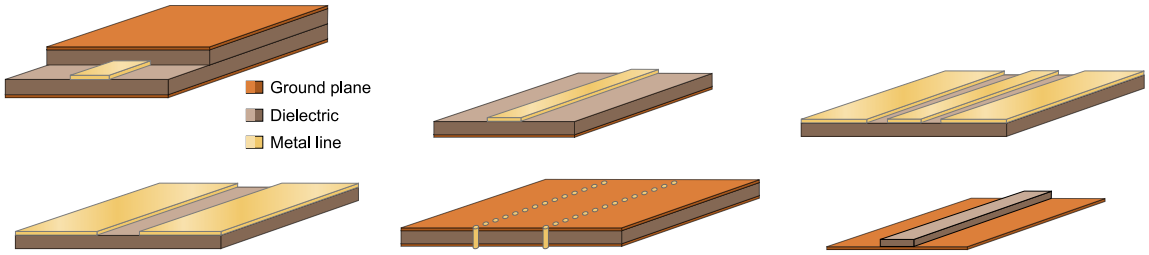


FIGURE 1.2 – Différentes technologies de circuit imprimé hyperfréquence :
Stripline, Microstrip, Coplanar waveguide (CPW)
Slotline, Substrate integrated waveguide (SIW), Imageline

La technologie microruban ("microstrip" en anglais) est l'une des plus utilisée car très simple à mettre en œuvre. Un circuit imprimé microruban est double face : une face sur laquelle se trouve le circuit et une sur laquelle se trouve un plan de masse, rien de plus.

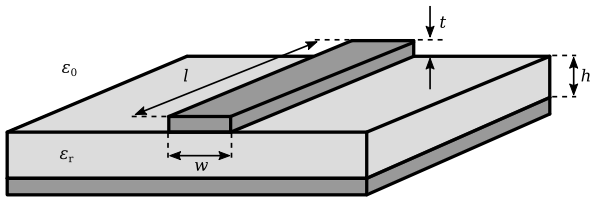


FIGURE 1.3 – Technologie microruban

Le schéma ci-dessus représente une coupe transversale de circuit imprimé avec en gris clair le substrat et en gris foncé le plan de masse sur le dessous et une piste sur le dessus.

De par sa nature sensible à de nombreux paramètres, la conception de circuits imprimés hyperfréquences est quasiment indissociable de l'utilisation d'un logiciel de simulation.

1.2 Qucs

Qucs est un logiciel libre de simulation électronique ne disposant pas d'interface pour produire de typon.

Le but de ce projet est de développer une interface permettant de convertir les schémas hyperfréquences de Qucs (prenant actuellement en charge les technologies microruban et coplanaire) vers un logiciel d'édition de typon, par exemple PcbNew de la suite KiCad ou encore pcb-rnd qui a l'air intéressant et adapté à ce genre d'application.

Qucs et PcbNew utilisent tous les deux des formats textes (façon XML) pour stocker leurs données. respectivement `.sch` et `.kicad_pcb`.

Le cœur du logiciel consistera donc en un convertisseur d'un format texte vers un autre. Un prototype bash fonctionnel a déjà été réalisé et est disponible à l'adresse suivante :
`https://github.com/thomaslepoix/QucstoKicad`.

Ensuite il faudra développer une interface graphique permettant un certain contrôle de l'opération. Par exemple :

- Choisir les fichiers d'entrée et de sortie.
- Choisir si l'on exporte le schéma comme un typon entier prêt à être imprimé ou comme un composant hyperfréquence (filtre, antenne, etc) prêt à être utilisé au sein d'un projet.
- Prévisualiser l'allure du typon élaboré à partir du schéma de simulation.
- Éventuellement effectuer des actions élémentaires sur la structure (rotation, symétrie d'une piste, taille du plan de masse, pourquoi pas l'ajout de vias espacées régulièrement, etc.).

À ce stade le logiciel serait fonctionnel. Il serait ensuite intéressant de l'intégrer au projet Qucs au même titre que d'autres outils comme le calculateur de lignes ou le synthétiseur de filtres.

Ci-dessous quelques documents permettant de comprendre le résultat attendu du logiciel. Dans l'ordre de déroulement du processus de production. Le logiciel à produire est donc l'outil permettant de passer de l'étape 1 à l'étape 2 :

- **`01_Filter_Qucs.pdf`** : Schéma de simulation d'un filtre sous Qucs.
- **`02_Filtre_Kicad.png`** : Ouverture avec PcbNew du fichier produit par le script prototype.
- **`03_Filter_Typon.pdf`** : Typon produit par PcbNew.
- **`04_Filter_Photo.png`** : Filtre produit avec le typon.

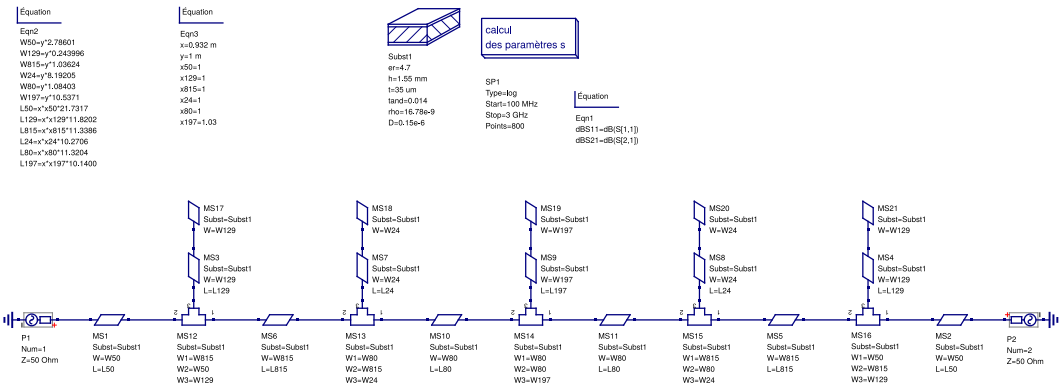


FIGURE 1.4 – 01_Filter_Qucs.pdf : Schéma de simulation d'un filtre sous Qucs

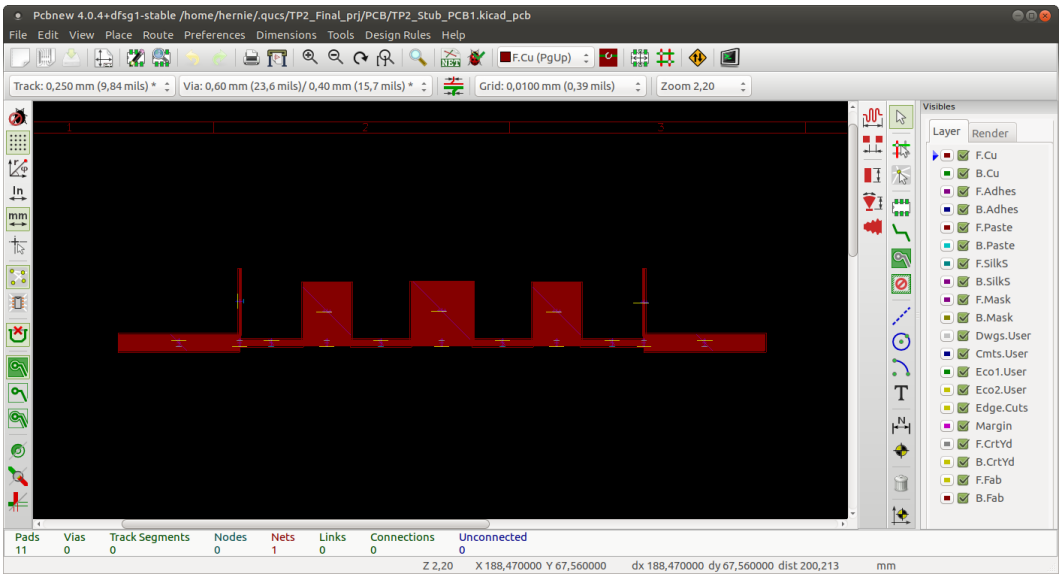


FIGURE 1.5 – 02_Filter_Kicad.png : Ouverture avec PcbNew du fichier produit par le script prototype

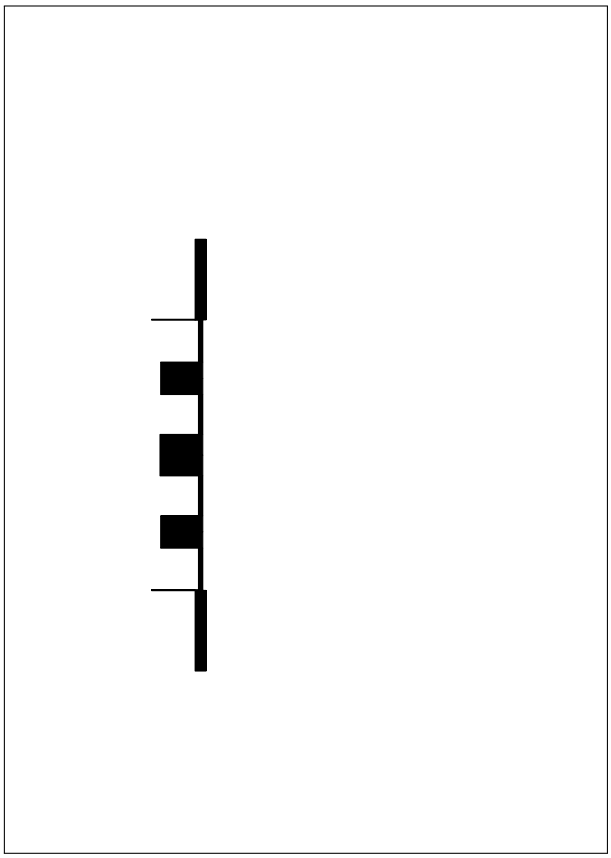


FIGURE 1.6 – 03_Filter_Typon.pdf : Typon sur feuille A5 produit avec PcbNew

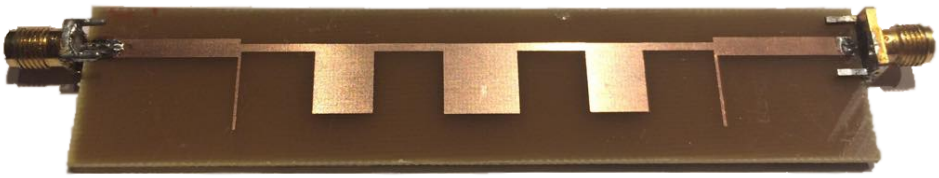


FIGURE 1.7 – 04_Filter_Photo.png : Circuit imprimé avec le typon

Lors du développement, le logiciel sera scindé en deux parties distinctes :

- Le moteur de conversion des formats dont le script existant est un prototype.
- L'interface graphique.

Chapitre 2

Étude et réalisation technique

2.1 Architecture

En analysant le format d'enregistrement des schémas de qucs, les données suivantes sont exploitables :

- La netlist, liste des interconnexions de tous les éléments du schéma. Par exemple "connexion entre le port 1 du composant 1 et le port 3 du composant 2".
- Le type de chaque élément, c'est-à-dire sa forme. Par exemple piste rectangulaire, point de jonction en T ou en X, angle, etc.
- Les dimensions de chaque élément.
- L'orientation de chaque élément (0°,90°,180°,270°).
- La présence ou non d'une symétrie sur l'axe horizontal.

La position absolue (en $\overrightarrow{x}$ et $\overrightarrow{y}$) ne fait pas partie de ces données car elle désigne la position des composants sur le schéma et non leur position réelle, ceux-ci étant reliés entre eux par des "fils" théoriques sans dimensions physiques.

La position des éléments est cependant nécessaire pour pouvoir dessiner le typon. Elle devra donc être déterminée à partir des autres données ainsi que de la position du premier élément analysé, fixée arbitrairement.

Le moteur du logiciel comportera donc plusieurs organes :

- Un parseur de texte pour extraire du schéma les données intéressantes.
- Un algorithme permettant de déterminer la position de chaque élément.
- Une fonction d'écriture dans le fichier de sortie.

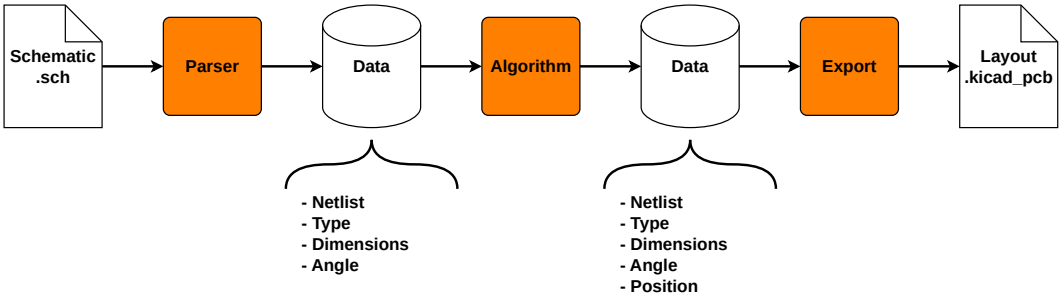


FIGURE 2.1 – Schéma du moteur de conversion

2.2 Analyse du format de données de Qucs

La documentation de Qucs fournit des explications sur le format `.sch`. La ligne décrivant les composants nous intéresse :

```
<type name active x y xtext ytext mirrorX rotate "Value1" visible "Value2" visible ...>
```

- `type` : Identifie le type du composant, par exemple `R` pour une résistance ou `MLIN` pour une ligne microruban.
- `name` : L'identificateur unique du composant, par exemple `R1`, `R2`, etc. Les composants microruban sont tous notés `MS1`, `MS2`, etc.
- `x`, `y`, `xtext` et `ytext` : Données relatives au placement du composant sur la feuille de travail de Qucs. Ces données ne nous intéressent pas.
- `mirrorX` : Indique si le composant a subi une symétrie sur l'axe x. Prend la valeur `1` en cas de symétrie, `0` sinon.
- `rotate` : Indique si le composant a subi des rotations antihoraires. Peut prendre les valeurs `0`, `1`, `2`, `3`, qui sont à interpréter comme des multiples de 90°.
- `"ValueX"` : Représente une donnée du composant, seules les dimensions nous intéressent.
- `visible` : Indique si la donnée précédente est visible sur la feuille de travail de Qucs. Ces données ne nous intéressent pas.

La commande suivante permet de générer la netlist associée au schéma choisi :

```
~$ qucs -n -i schematic.sch -o netlist.net
```

Celle-ci contient des lignes à l'allure suivante :

```
MLIN:MS1 _net0 _net1 Subst="Subst1" W="1 mm" L="10 mm" Model="Hammerstad"
↪  DispModel="Kirschning" Temp="26.85"
MCORN:MS2 _net1 _net2 Subst="Subst1" W="1.1 mm"
```

- On reconnaît d'abord le type et l'identificateur du composant.
- `_netX` sont les données qui nous intéressent dans ce fichier, elles désignent le nom du "fil" connecté à chaque port du composant. Si un autre composant est connecté à celui-ci, l'un de ses ports sera "relié" à l'un de ces "fils". Par exemple ici le second port du composant `MS1` est connecté au premier port du composant `MS2`.
- Certaines données sont redondantes avec celles stockées dans le schéma, les dimensions notamment.
- D'autres présentes dans le schéma, la symétrie et la rotation, ne figurent pas ici.

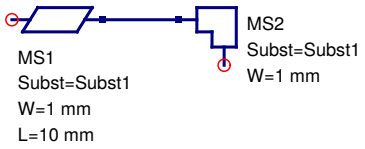


FIGURE 2.2 – Configuration décrite par l'exemple de netlist précédent

Finalement les données intéressantes sont le type, l'identificateur, la symétrie, la rotation et les dimensions pour le fichier schéma. Et les connexions entre ports pour le fichier netlist.

2.3 Parseur

Le parseur a pour rôle d'extraire des fichiers schéma et netlist les données intéressantes et de les transcrire dans la mémoire interne du logiciel.

Les éléments seront représentés par des classes, une par type d'élément. Il s'agit ici d'initialiser un objet pour chaque élément du schéma avec toutes les valeurs disponibles, à savoir type, identificateur, symétrie, rotation et dimensions.

Ensuite, lors de la lecture de la netlist, d'affecter une valeur aux attributs correspondants aux ports de connexion.

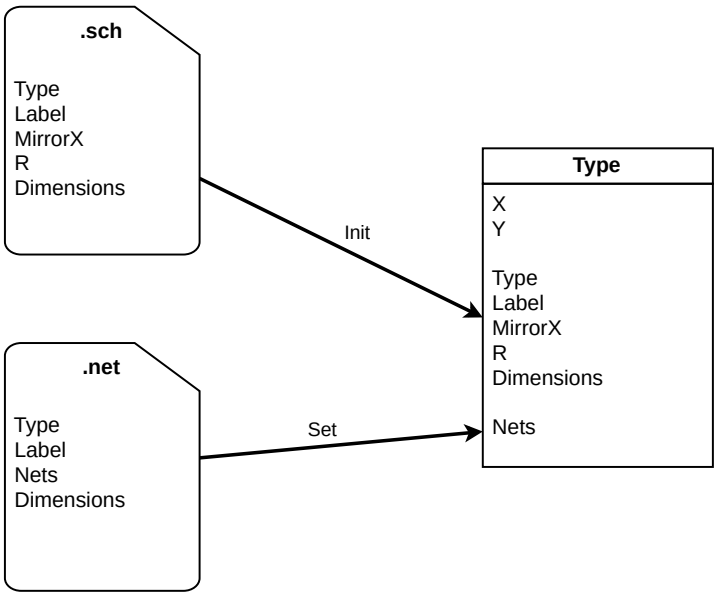


FIGURE 2.3 – Schéma de l'extraction des données dans le schéma et la netlist par le parseur

La lecture des fichiers se fait par l'utilisation des expressions régulières. Le parseur dispose de plus d'une fonction permettant de convertir en multiplicateur les éventuels suffixes présents dans les valeurs des dimensions, en écriture scientifique ou notation d'ingénieur.

2.4 Principe de l'algorithme de calcul de position

- La position d'un élément dépend de son type, généralement il s'agit de la position de son centre.

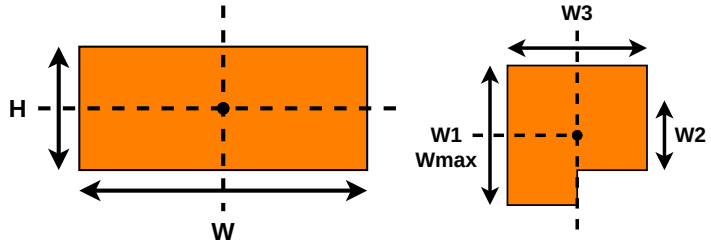
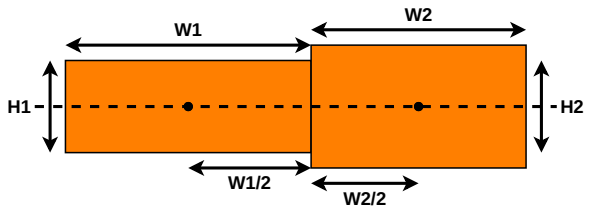


FIGURE 2.4 – Schéma du centre d'une piste rectangulaire (**MLIN**) et d'un point de jonction en T (**MTEE**)

- La position d'un élément est déterminée par la position de l'élément précédent, leurs orientations et leurs propriétés géométriques.



$$P2 = P1 + \frac{W_1}{2} + \frac{W_2}{2}$$

FIGURE 2.5 – Schéma de la position d'un élément relative à un autre

- Point de départ de l'algorithme : L'élément d'indice minimal.
- Passe d'élément en élément par le port non traité d'indice minimal.
- Un élément est traité lorsque tous ses ports ont été traités.
- Un élément partiellement traité est mis en attente dans une liste de type LIFO (Last In First Out).

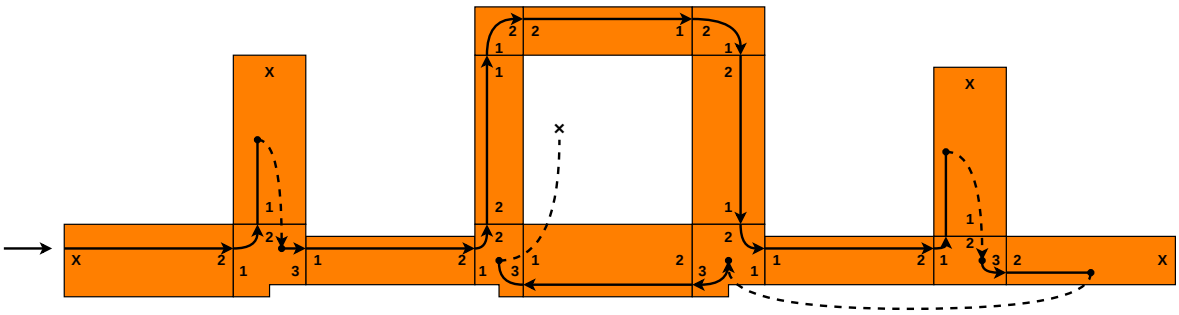


FIGURE 2.6 – Déroulement de l'algorithme sur un schéma microruban quelconque

- Une fois toutes les positions connues, un décalage peut être nécessaire afin qu'aucun élément n'ait de coordonnées négatives, ce qui le ferait déborder du cadre de travail sur le logiciel de sortie.

L'algorithme peut être modélisé de la façon suivante. Figure en légende une représentation des structures mémoires utilisées, notamment des trois pointeurs **`current`**, **`prev`**, **`next`**.

- **`current`** est un curseur de l'élément en cours d'analyse, il évolue sur le schéma comme décrit sur la figure précédente.
- **`next`** sert uniquement de tampon pour faire pointer **`next->prev`** vers **`current`** avant de faire pointer **`current`** vers **`next`**.
- **`prev`** représente l'élément précédent l'élément **`current`** dans le sens où la position de **`prev`** est utilisée pour déterminer la position de **`current`**. D'ou le fait que **`prev`** soit un attribut propre à chaque élément et non un curseur libre de déplacement comme le sont **`next`** et **`current`**.

La ligne de pointillés marque l'instant de la boucle où le pointeur **`current`** passe d'un élément à un autre. De plus les variables **`prev`** et **`XY`** sans préfixe sont relatives à **`current`**. Enfin, **`prev_XYstep`** et **`current_XYstep`** sont deux variables libres et non des attributs des éléments **`prev`** et **`current`**.

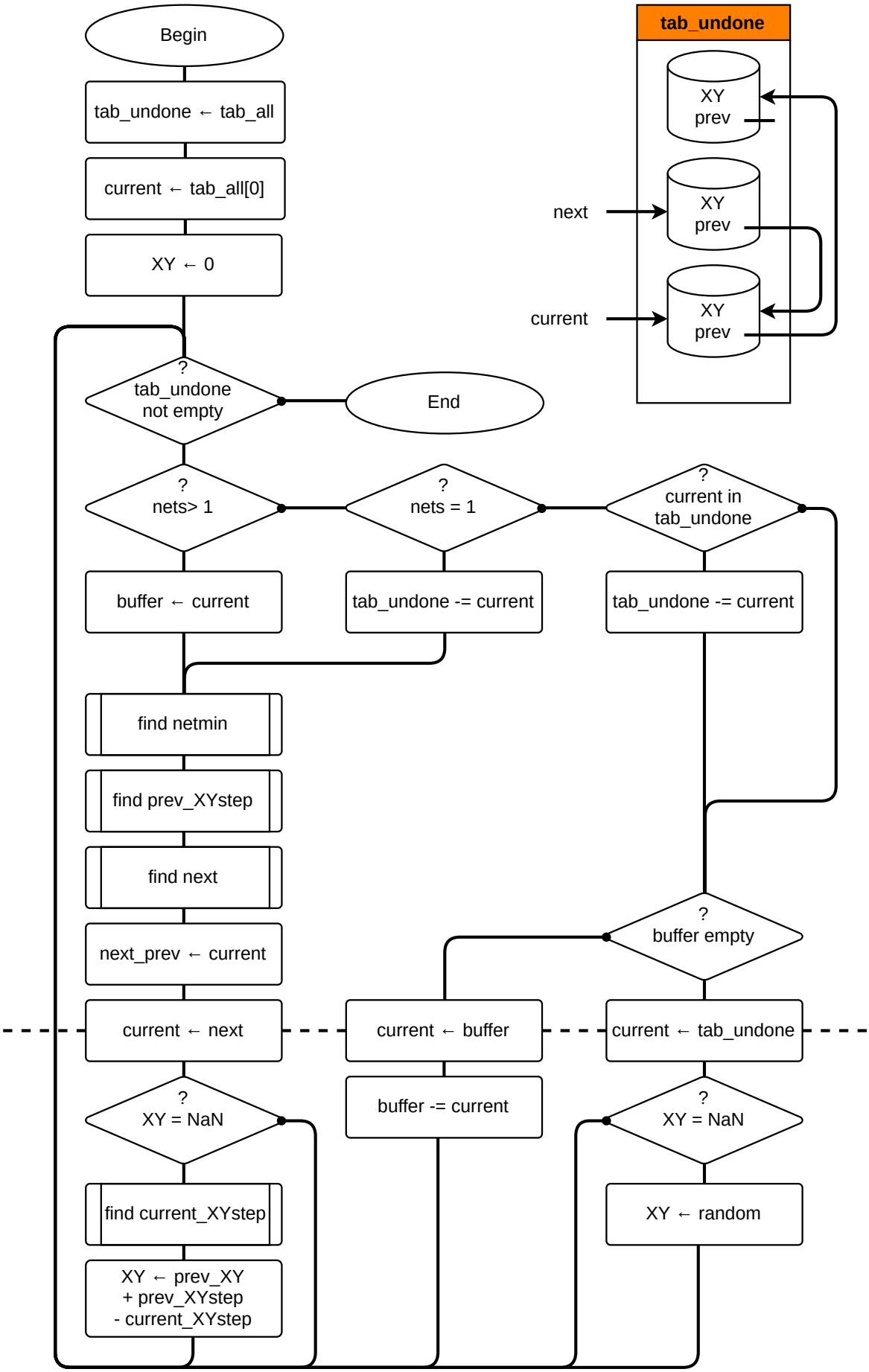


FIGURE 2.7 – Algorigramme du calculateur de position

2.5 Modélisation des éléments

Les éléments du schéma sont modélisés par des classes, toutes héritant de la classe mère **`Element`** qui regroupe les attributs et méthodes communs à tous les éléments.

L'on retrouve ainsi stockés dans la classe **`Element`** :

- Le label de l'élément.
- Son type.
- La présence d'une symétrie horizontale.
- L'angle d'orientation.
- Le nombre de ports de l'élément.
- La coordonnée en $\overrightarrow{x}$ de sa position.
- La coordonnée en $\overrightarrow{y}$ de sa position.
- Les **`getters`** de tous ces attributs.
- Les **`setters`** des attributs n'étant pas initialisés au moment de la création d'un objet, c'est à dire les coordonnées en $\overrightarrow{x}$ et en $\overrightarrow{y}$ de l'élément.
- Un pointeur vers l'élément "précédent". Il s'agit là de satisfaire un besoin technique de l'algorithme de parcours du schéma.
- Les méthodes **`rotateX()`** et **`rotateY()`**, qui seront utilisées par les méthodes **`getP()`** des classes filles.

Et dans les classes filles décrivant les éléments :

- Une ligne de description.
- Tous les attributs relatifs à la forme géométrique de l'élément en question.
- Pour chaque port, le nom du "fil" connecté audit port.
- Le nombre de sommets que comporte la forme géométrique de l'élément.
- Les **`getters`** de tous ces attributs.
- Les **`setters`** des attributs n'étant pas initialisés au moment de la création d'un objet, c'est à dire les "fils" connectés à chaque port.
- Un tableau à deux dimensions **`tab_p[][2]`**contenant les coordonnées en $\overrightarrow{x}$ et en $\overrightarrow{y}$ de chaque sommet.
- Une méthode **`setP()`** permettant de calculer la position de chaque sommet à partir de la position de l'élément.
- Une méthode **`getP()`** permettant d'obtenir les coordonnées de chaque sommet selon plusieurs paramètres.

Ci-dessous un diagramme de classe UML représentant la classe mère **`Element`** et une classe fille quelconque, **`Mlin`** qui modélise une ligne microruban. Ce diagramme peut être étendu à l'ensemble des classes décrivant des éléments. Les attributs décrivant la géométrie de l'élément ainsi que leurs **`getters`** et **`setters`** ne seront bien entendu pas les mêmes.

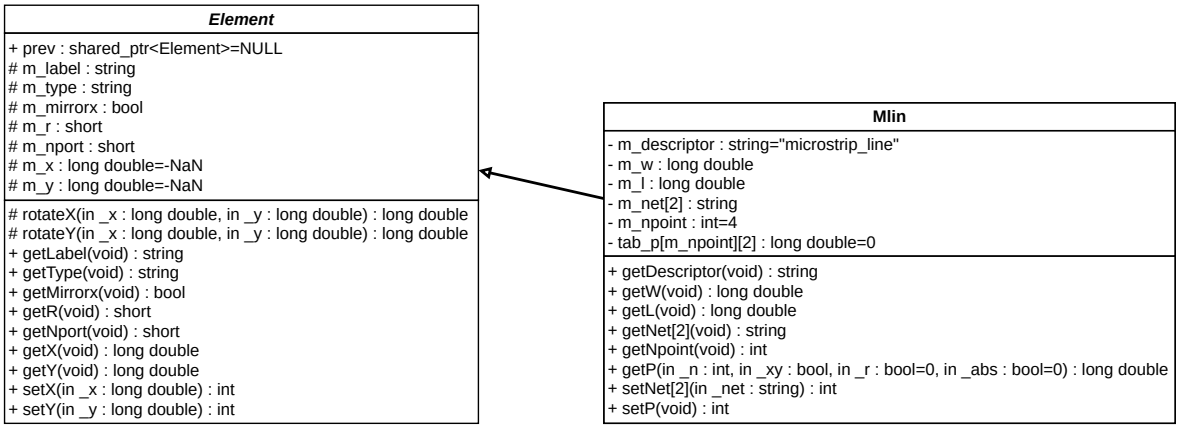


FIGURE 2.8 – Diagramme de classe UML
Classes mère virtuelle Element et classe fille quelconque Mlin

2.5.1 Les méthodes setP() et getP()

Le format **`.kicad_pcb`** fonctionne d'une façon proche du format de Qucs **`.sch`** puisqu'il permet de décrire les éléments comme suit :

- La position de l'élément (souvent de son centre) sur le typon.
- L'angle de rotation appliqué à celui-ci.
- Ses dimensions, dans le cas où le format **`.kicad_pcb`** propose une forme géométrique correspondante. Par exemple la largeur et la longueur d'une ligne microruban **`MLIN`** modélisée par un rectangle. Ou encore l'angle d'ouverture, le rayon externe et le rayon interne d'un stub radial **`MRSTUB`**.

Cependant tous les formats ne fonctionnent pas d'une façon aussi proche du format **`.sch`**. La façon la plus universelle de dessiner les formes géométriques est d'utiliser la forme primitive du polygone, qui n'est autre qu'une succession de sommets. Certains formats prennent en compte les rotations, d'autres pas. Certains formats proposent une position centrale pour le polygone et utilisent des positions relatives au centre pour placer les sommets du polygone, d'autres pas et utilisent des coordonnées absolues pour chaque sommet, c'est-à-dire relatives à l'origine du typon.

Pour maximiser la compatibilité du logiciel à différents formats de sortie, il est donc nécessaire de disposer d'un système assez souple pour obtenir les coordonnées de chaque sommet d'une forme :

- Absolues ou relatives à la position de la forme.
- Avec ou sans l'application de la rotation.

Dans ce but, chaque classe décrivant un élément dispose d'un tableau à deux dimensions **`tab_p[][2]`** contenant les coordonnées $\overrightarrow{x}$ et $\overrightarrow{y}$ de chaque sommet de la forme de l'élément. Ces coordonnées sont relatives à la position de l'élément et l'angle de rotation n'est pas appliqué.

La méthode **`setP()`** permet de calculer toutes ces coordonnées à partir de la position de l'élément et de ses dimensions. Elle est donc totalement propre à chaque type d'élément. À titre d'exemple, voici le code de la méthode **`setP()`** de la classe **`Mlin`**, qui modélise un rectangle doté d'une largeur et d'une longueur :

```
int Mlin::setP(void) {
    tab_p[0][_X]=-m_l/2;
    tab_p[0][_Y]= m_w/2;
    tab_p[1][_X]= m_l/2;
    tab_p[1][_Y]= m_w/2;
    tab_p[2][_X]= m_l/2;
    tab_p[2][_Y]=-m_w/2;
    tab_p[3][_X]=-m_l/2;
    tab_p[3][_Y]=-m_w/2;
    return(0);
    }
```

Quant à la méthode **`getP()`**, elle permet par ses arguments d'obtenir pour chaque sommet une coordonnée :

- $\overrightarrow{x}$ ou $\overrightarrow{y}$.
- Avec ou sans l'application de la rotation. Les méthodes **`rotateX()`** et **`rotateY()`** de la classe **`Element`** sont prévues à cet effet.
- Absolue ou relative à la position de l'élément.

Cette méthode dépend dans une moindre mesure de la classe dans laquelle elle figure. Voici son code pour la classe **`Mlin`** :

```
long double Mlin::getP(int _n, bool _xy, bool _r, bool _abs) {
    long double coord;
    if(_r) {
        coord= _xy ? rotateY(tab_p[_n][_X], tab_p[_n][_Y])
                   : rotateX(tab_p[_n][_X], tab_p[_n][_Y]);
    } else {
        coord=tab_p[_n][_xy];
        }
    return(_abs ? coord+(_xy ? m_y : m_x) : coord);
    }
```

2.5.2 Implémentation des formes géométriques des éléments

Il est pertinent de dresser une table de vérité de l'orientation des éléments en fonction de leur angle de rotation (multiples de 90°) et de la symétrie sur l'axe des $\overrightarrow{x}$:

MX	R	MLIN	MGAP	MSTEP	MOPEN	MRSTUB	MCORN	MMBEND	MTEE	MCROSS	MCOUPLED
0	0	1 2	1 2	1 2	1	1	1 2	1 2	1 2 3	2 1 3 4	1 2 4 3
0	1	2 1	2 1	2 1	1	1	2 1	2 1	2 3 1	3 2 4 1	2 3 1 4
0	2	2 1	2 1	2 1	1	1	2 1	2 1	3 2 1	4 3 1 2	3 4 2 1
0	3	1 2	1 2	1 2	1	1	1 2	1 2	1 3 2	1 4 2 3	4 1 3 2
1	0	X Symetry independant				1	2 1	2 1	3 1 2	4 1 3 2	4 3 1 2
1	1					1	2 1	2 1	2 3 1	3 4 2 1	3 2 4 1
1	2					1	1 2	1 2	2 1 3	2 3 1 4	2 1 3 4
1	3					1	1 2	1 2	1 3 2	1 2 4 3	1 4 2 3

FIGURE 2.9 – Table de l'orientation des éléments en fonction de leurs attributs "rotate" et "mirrorX"

Ensuite, concernant le dessin des formes, cela est directement lié à la théorie des lignes en hyperfréquences. Le plus sage est de consulter la documentation d'autres logiciels de simulation pour observer la géométrie des formes les plus complexes. Par exemple :

- **`https://awrcorp.com/download/faq/english/docs/Elements/ELEMENTS.htm`**
- **`http://literature.cdn.keysight.com/litweb/pdf/ads2002c/dgpas/index.html`**

Parmi les éléments microruban pris en charge, **`MOPEN`** et **`MSTEP`** n'ont aucune signification géométrique, il ne correspondent qu'à un effet à prendre en compte lors de la simulation. Pour les autres, ci-dessous les schémas de leur topologie, sans rotation et sans symétrie :

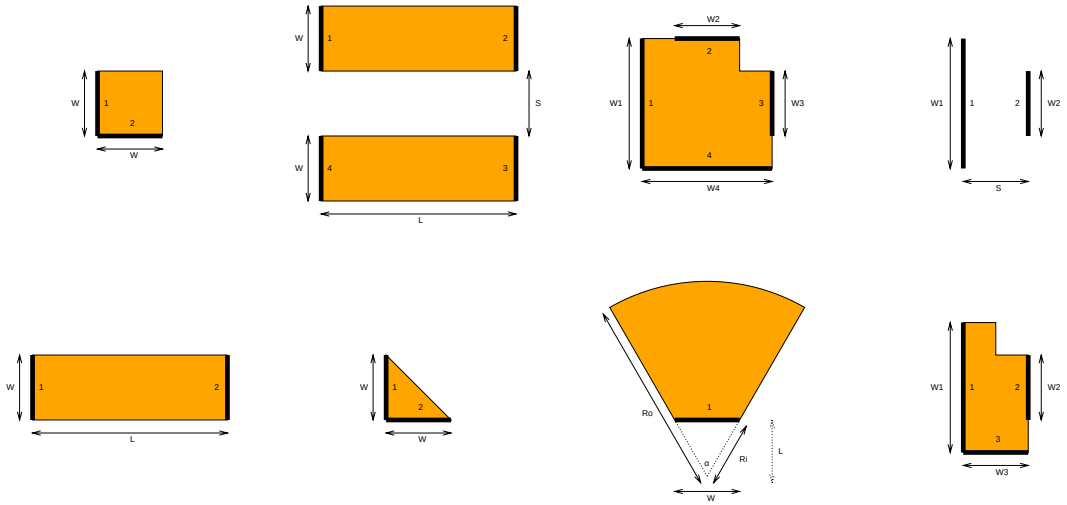


FIGURE 2.10 – Topologie des éléments microruban pris en charge et ayant une géométrie
MCORN, **MCOUPED**, **MCROSS**, **MGAP**
MLIN, **MMBEND**, **MRSTUB**, **MTEE**

2.6 Export vers un logiciel de sortie

Pour implémenter le support d'un format de sortie, la méthode repose sur les quelques points suivants :

- Créer un typon vierge et le sauvegarder. Cela constitue généralement le squelette du fichier à produire.
- Créer des polygones et leur appliquer des transformations (symétrie, rotation, translation) pour observer comment le fichier évolue.
- De même pour les autres formes disponible s'il en existe, notamment les vias.
- Une lecture de la documentation du format est vivement recommandée si elle est disponible.

Les logiciels de sortie cibles prioritaires du projet sont les suivants :

- PcbNew de la suite KiCad : L'un des logiciels libres de conception électronique les plus aboutis et les plus utilisés. Il enregistre les données au format **`S-expression`**. Il permet de sauvegarder un typon complet avec l'extension **`.kicad_pcb`** ou une empreinte de composant avec l'extension **`.kicad_mod`**. La façon de coder les données est légèrement différente dans les deux cas.
- pcb-rnd, fork de PCB de la suite gEDA, semble approprié au design hyperfréquence puisqu'à la différence de PcbNew conçu pour fonctionner par projets, pcb-rnd permet un design beaucoup plus libre, de "dessiner" un circuit imprimé sans nécessité d'utiliser des composants associés à des empreintes. Il enregistre les données au format **`lihata`**. Il permet de sauvegarder un typon complet avec l'extension **`.lht`** ou aussi une empreinte avec l'extension **`.lht`** ou **`.fp`**.

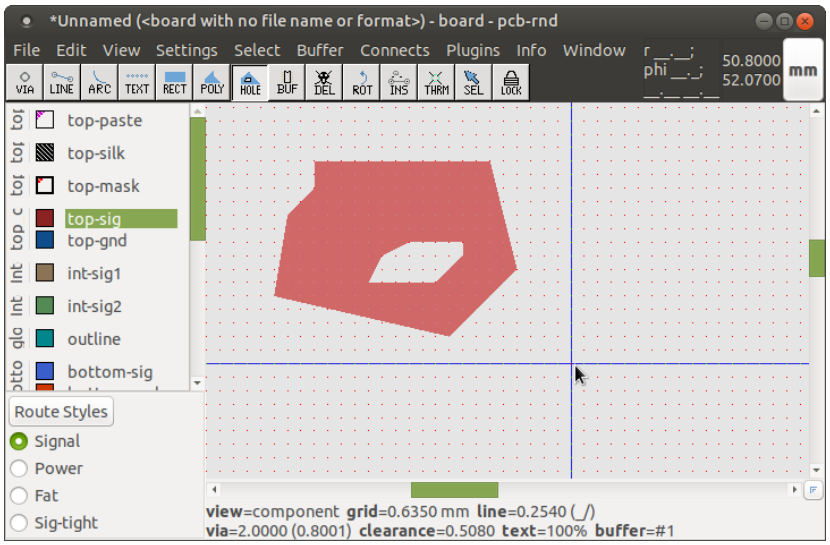


FIGURE 2.11 – Aperçu de pcb-rnd

- OpenEMS n'est pas un logiciel d'édition de typon, il s'agit d'un logiciel de simulation électromagnétique, Qucs étant un logiciel de simulation électronique. La différence est que Qucs n'est capable de simuler un circuit microruban que par le modèle des lignes de transmission, tout à fait valable dans le cas d'un filtre par exemple mais qui montre ses limites lorsqu'il s'agit de simuler un circuit rayonnant, comme une antenne patch. D'autres méthodes beaucoup plus lourdes et robustes comme MOM (résolution d'intégrales) et FDTD (résolution d'équations différentielles) permettent de simuler des comportements électromagnétiques en 3 dimensions. OpenEMS repose sur la méthode FDTD.
  C'est un logiciel qui utilise des scripts Matlab ou son équivalent libre GNU Octave comme interface utilisateur. Ces scripts peuvent être fastidieux à écrire puisqu'il faut décrire "à la main" les formes géométriques 3D que l'on souhaite simuler ainsi que le maillage correspondant. Le présent projet pourrait donc également être une porte vers un moteur de simulation électromagnétique jusqu'à présent absent de Qucs. À noter qu'une interface entre les deux logiciels fait partie des ambitions de l'équipe de développement de Qucs et qu'il utilise d'ores et déjà Octave pour exécuter des scripts post-simulation. Les scripts Matlab / Octave portent l'extension `.m`.

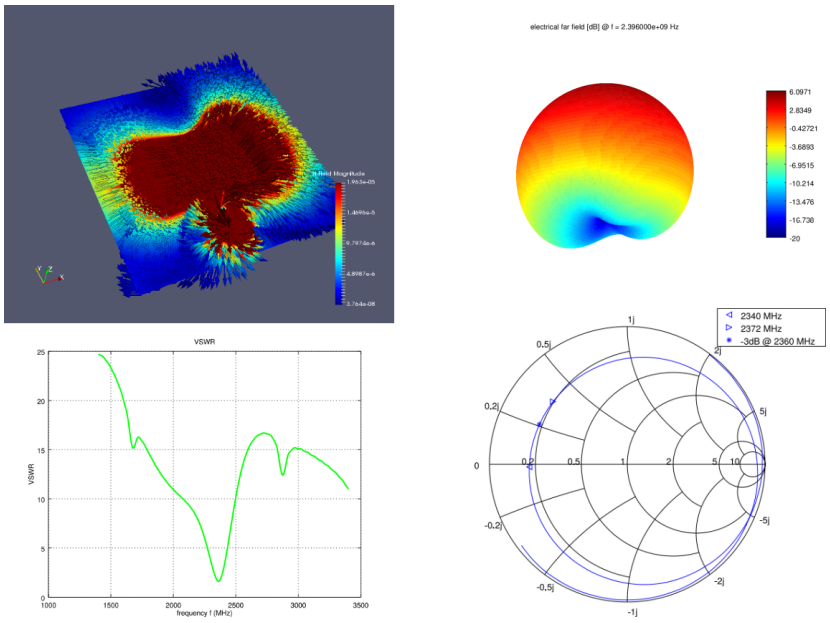


FIGURE 2.12 – Aperçu de résultats de simulation d'une antenne patch avec OpenEMS

- Les formats `.pdf` et `gerber` sont également des options intéressantes qui dispenseraient même de l'utilisation d'un logiciel d'édition de typon pour produire un circuit.

2.7 Interface graphique

Ci-dessous l'allure globale de l'interface graphique. Elle doit avant tout permettre le choix des fichiers d'entrée et de sortie et le choix du format de sortie. Ainsi que l'aperçu du typon tel que reconstitué à partir du schéma de simulation.

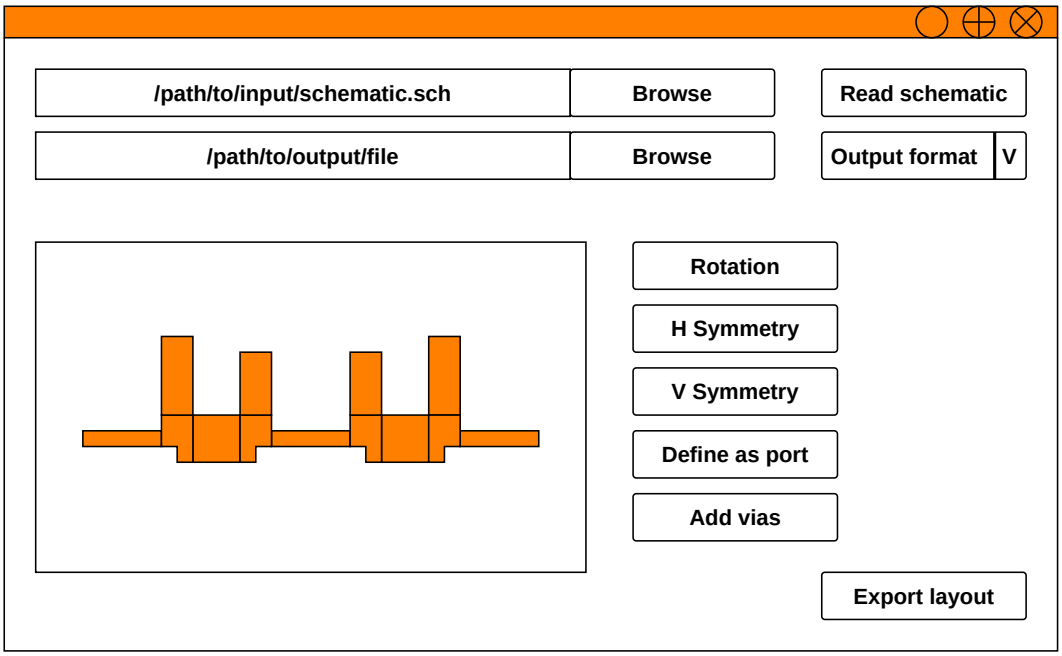


FIGURE 2.13 – Schéma de l'allure de l'interface graphique

L'interface doit également permettre d'agir de façon rudimentaire sur le schéma. Par exemple de définir des ports dans le cas où le schéma sera exporté comme empreinte de composant. Ou d'effectuer quelques actions géométriques.

Chaque action sera bridée afin de ne pas être impertinente. Par exemple, il n'est pas pertinent d'utiliser une symétrie qui modifierait les caractéristiques électromagnétiques de la structure schématisée. Ni de définir comme port un élément n'en étant pas un lors de la simulation.

Une étude de la pertinence de chaque action dans tous les cas de figure possibles sera donc à réaliser. Cette fonctionnalité peut s'avérer longue et compliquée à élaborer puisque la pertinence d'une modification géométrique est déterminée par la théorie des transmissions hyperfréquence. Études et analyses seront donc nécessaires.

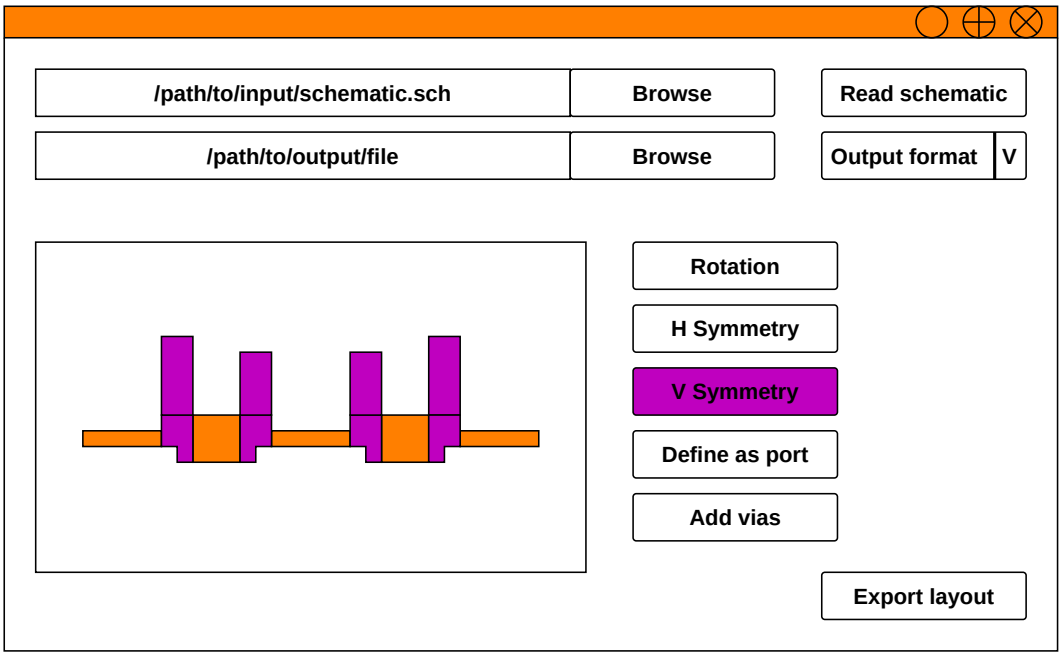


FIGURE 2.14 – Exemple : Éléments sujets à la symétrie verticale

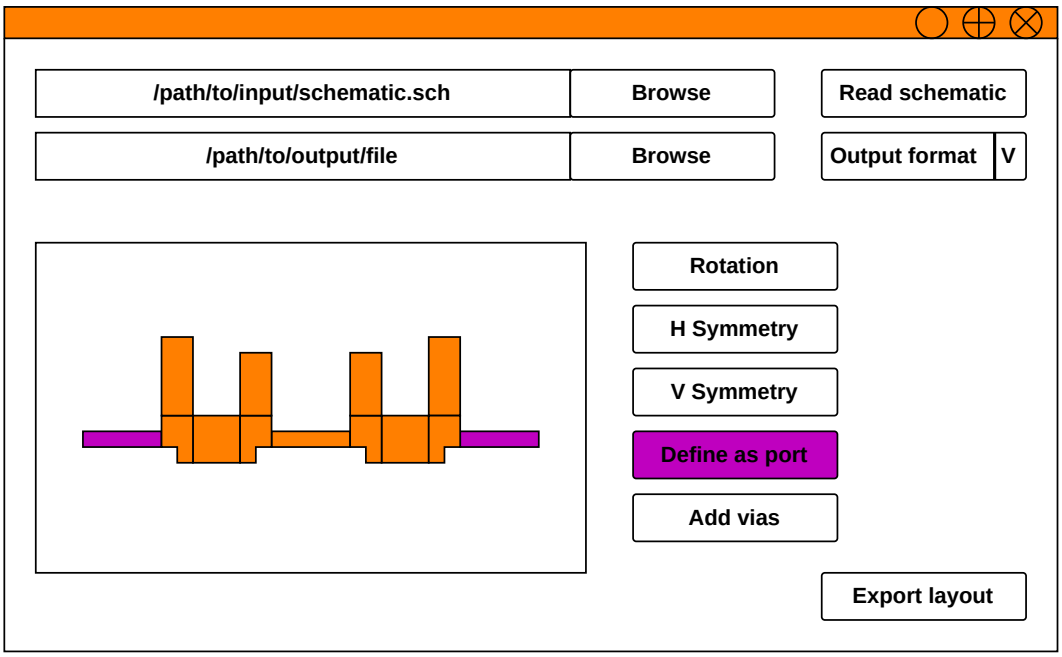


FIGURE 2.15 – Exemple : Éléments sujets à la définition comme port

2.8 Prévisualisation du typon élaboré

La prévisualisation du typon doit permettre quelques actions rudimentaires d'observation, à savoir le zoom, la translation verticale et horizontale. Le scroll de la souris combiné à une touche de sélection est une interface appropriée pour cela.

Concernant la touche de sélection, deux conventions ont été rencontrées : La première utilisée par KiCad, pcb-rnd et partiellement OpenEMS, et la seconde utilisée par Qucs, Altium-designer, ainsi que GIMP et Inkscape (listes non-exhaustives).

- Aucune : Zoom
- `[CTRL]` : Translation horizontale
- `[SHIFT]` : Translation verticale

- Aucune : Translation verticale
- `[CTRL]` : Zoom
- `[SHIFT]` : Translation horizontale

La convention utilisée par Qucs semble préférable bien que celle utilisée par les logiciels de sortie KiCad et pcb-rnd semble également pertinente. Une paramétrabilité de cela est toutefois envisageable.

Concernant le rendu graphique, en prévision de l'implémentation de l'export vers un logiciel comme OpenEMS qui tient compte de l'épaisseur du substrat, des pistes de cuivres, etc. un rendu tridimensionnel a été choisi dès le départ. D'où l'utilisation d'OpenGL. Un clic (droit ou gauche) glissé permet de faire pivoter le typon sur les trois axes de l'espace.

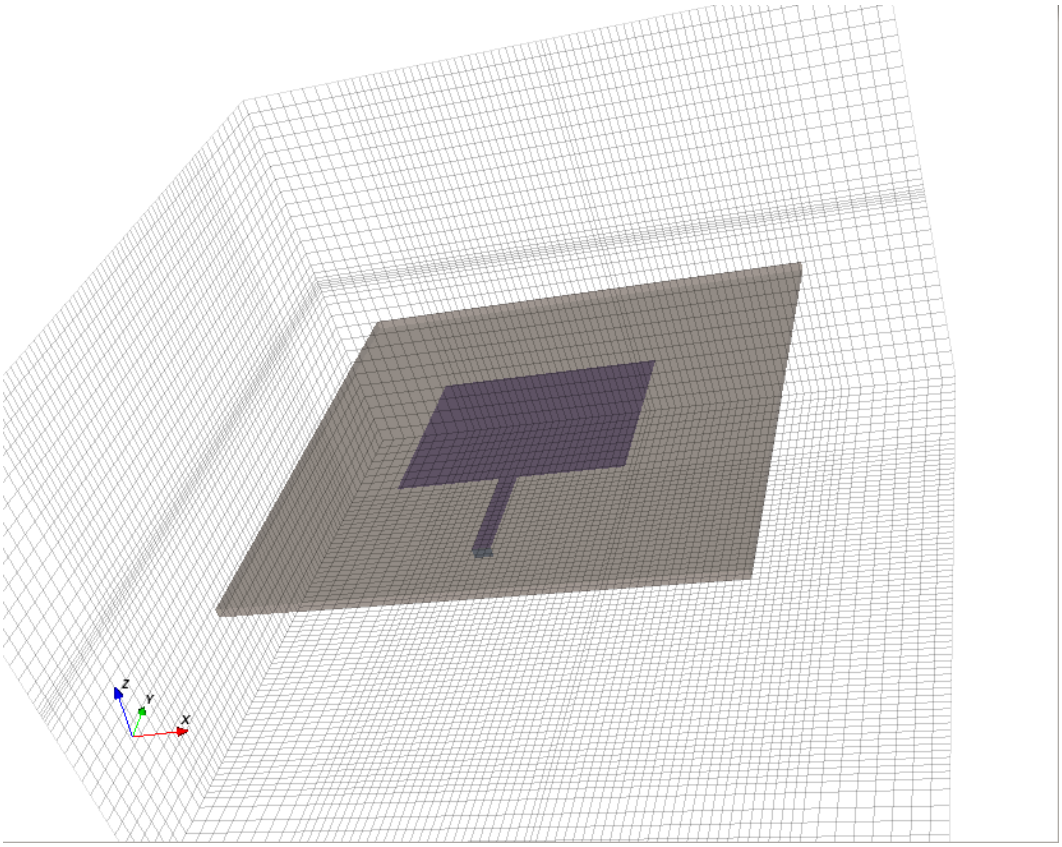


FIGURE 2.16 – Aperçu d'une antenne patch à simuler sur OpenEMS

Chapitre 3

Éléments techniques intéressants

3.1 Stockage des objets

Au départ le programme a été envisagé selon l'approche du mode texte, où le moteur de conversion n'est appelé qu'une fois par exécution du programme. Stocker les objets générés par le parseur dans un tableau de pointeur d'objets a donc été une solution naturelle.

Cependant l'implémentation de l'interface graphique telle qu'imaginée implique que le parseur puisse être appelé plusieurs fois durant l'exécution du programme. Et donc que le cycle de création, stockage, destruction des objets ait lieu plusieurs fois. Le tableau de pointeurs d'objets comme moyen de stockage des objets devient alors problématique puisqu'il n'est pas extensible.

Des tentatives de créer un mécanisme de réallocation mémoire du tableau se sont avérées infructueuses puisque le comportement du programme devenait relativement aléatoire lorsque le parseur était appelé plus d'une fois.

La solution retenue pour régler le problème a été de stocker les objets dans un conteneur beaucoup plus ++ que le tableau de pointeurs hérité du langage C. Il s'agit du couple vecteur - pointeurs intelligents.

3.1.1 Les pointeurs intelligents

L'allocation dynamique (avec `new`) requiert une attention toute particulière puisque la désallocation n'est pas automatique et doit être faite manuellement (avec `delete`). De plus une entité allouée dynamiquement n'est accessible que par un pointeur.

Les principaux risques liés a l'utilisation des pointeurs classiques pour gérer les entités dynamiques sont les suivants :

- Le risque de ne pas libérer la mémoire allouée. En particulier quand différents éléments du programme peuvent lever des exceptions qui sont autant de chemins possibles que peut prendre le programme et qui peuvent faire qu'un `delete` ne sera jamais appelé. Créant ainsi une fuite mémoire.
- Le risque de désallouer la mémoire d'un pointeur ayant déjà été désallouée ou n'ayant pas encore été allouée.
- Le risque de désallouer la mémoire d'un pointeur tandis que d'autres pointeurs pointent sur la même case mémoire. Ce qui peut donner lieu a des comportements difficiles à prévoir.

Les pointeurs intelligents ont été créés dans le but de pallier ces problèmes. Il s'agit de templates se comportant comme des pointeurs classiques, à la différence qu'ils disposent d'un mécanisme de surveillance mémoire faisant que lorsqu'ils cessent de vivre, la mémoire pointée est automatiquement désallouée si aucun autre pointeur ne pointe sur la même case mémoire.

Il en existe trois :

- **`unique_ptr`** : Ce pointeur est le seul à pouvoir accéder à la mémoire qu'il pointe.
- **`shared_ptr`** : Ce pointeur tolère que d'autres pointeurs pointent sur la mémoire pointée. Il dispose d'un compteur permettant de savoir combien de pointeurs pointent sur cette mémoire, lorsque le compteur tombe à zéro, la mémoire est désallouée.
- **`weak_ptr`** : Ce pointeur peut pointer sur une mémoire allouée par un **`shared_ptr`** mais n'incrémente pas le compteur des pointeurs associés à la mémoire. En d'autres termes, il ne peut pas être le seul pointeur associé à une mémoire et n'est pas capable de prolonger la durée de vie d'une entité.

Ainsi l'utilisation des pointeurs intelligents dispense de la nécessité de **`delete`**.

À titre de d'illustration, voici la syntaxe du même programme sans fuite mémoire, écrit avec un pointeur classique et avec un pointeur intelligent :

```
#include <iostream>
using namespace std;

int main(void) {
    int* a=new int(5);
    cout << "a : " << a << endl;
    cout << "*a : " << *a << endl;
    delete a;

    return(0);
    }
```

```
#include <iostream>
#include <memory>
using namespace std;

int main(void) {
    unique_ptr<int> a=unique_ptr<int>(new int(5));
    cout << "a : " << a.get() << endl;
    cout << "*a : " << a << endl;

    return(0);
    }
```

Pour plus de détails, un cours sur les pointeurs intelligents se trouve à l'adresse suivante : **`https://loic-joly.developpez.com/tutoriels/cpp/smart-pointers/`**

3.1.2 Les vecteurs

La librairie standard propose de nombreux conteneurs. Le vecteur est l'un des plus élémentaire, il se substitue bien au tableau du langage C puisqu'il permet d'accéder à ses éléments par l'opérateur `[]`, cependant il a l'avantage que n'a pas le tableau d'être extensible et rétractable.

Les méthodes les plus utilisées sont :

- **`.push_back()`** qui permet d'ajouter un élément à la fin du vecteur.
- **`.clear()`** qui permet de vider le vecteur, c'est à dire de désallouer en mémoire les éléments qui s'y trouvent.
- **`.shrink_to_fit()`** qui permet de réduire la taille occupée par le vecteur à ce qu'il contient actuellement.

Pour stocker des objets alloués dynamiquement dans un vecteur, il est particulièrement conseillé d'utiliser des pointeurs intelligents. Les syntaxes mises en œuvre sont alors les suivantes :

```
#include <vector>
#include <memory>
#include "mlin.h"
using namespace std;

int main(void) {
    vector<shared_ptr<Element>> tab_all;
    tab_all.push_back(shared_ptr<Element>(new Mlin("MS1", "MLIN", 1, 0, 2, 1.0, 10.0)));

    return(0);
    }
```

En plus de résoudre le problème posé par l'utilisation d'un tableau, le vecteur a le bon goût de permettre d'utiliser des boucles **`foreach`** avec des itérateurs, cela permet d'alléger significativement la syntaxe des accès aux éléments du tableau.

```
for(int i=0;i<n_elem;i++) {
    tab_all[i]->setP();
    }
```

```
for(shared_ptr<Element> it : tab_all) {
    it->setP();
    }
```

3.2 Les expressions régulières

Les expressions régulières sont un outil extrêmement puissant de description de syntaxe. Elles sont donc tout à fait adaptées à la réalisation d'un parseur de texte. Deux sites sont particulièrement utiles pour les utiliser. Le premier détaille leurs syntaxes et les façons de les utiliser de manière assez exhaustive, le second est un outil facilitant leur création.

`http://www.rexegg.com/regex-quickstart.html`
`https://regex101.com/`

Ci-dessous l'allure du site. Il propose une ligne où entrer l'expression régulière à tester, une boite où entrer du texte auquel elle sera appliquée, un volet d'explication de chacun des caractères et des motifs présents dans l'expression régulière, ainsi qu'un volet affichant le contenu de chacun des groupes de correspondances de l'expression pour chaque portion du texte qu'elle décrit.

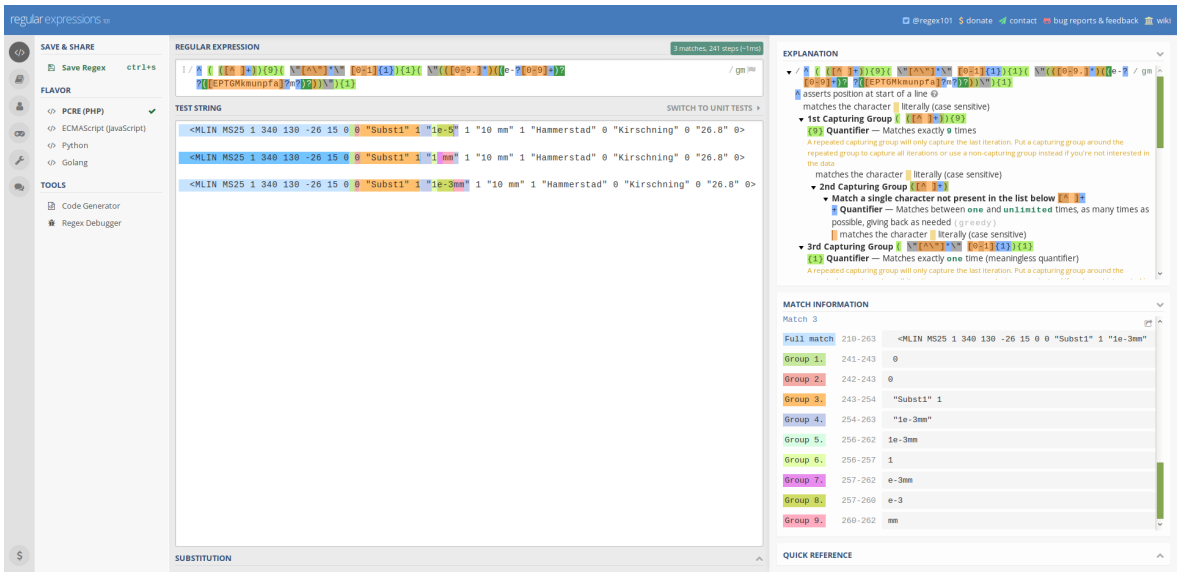


FIGURE 3.1 – Capture d'écran du site regex101.com lors de l'élaboration de l'une des expressions régulières utilisées par le parseur

Globalement les expressions régulières permettent d'isoler des motifs (par l'utilisations de parenthèses) appelés "matching group", groupes de correspondances en français, auxquels on peut ensuite accéder grâce à leur indice. Par exemple, l'expression régulière ci-dessous est utilisée par le parseur pour récupérer le premier champ contenant la valeur d'une dimension d'un élément.

```
^ ( ([^ ]+)){9}( \"[^\"]*\" [0-1]{1}){1}( \"(([0-9.]*)((e-?[0-9]+)?
↪  ?([EPTGMkmunpfa]?m?)?))\"){1}
```

- Le groupe `6` permet d'accéder à la valeur absolue du champ.
- Le groupe `8` permet d'accéder à la portion en notation scientifique du suffixe.
- Le groupe `9` permet d'accéder à la portion en notation d'ingénieur du suffixe.

```
<MLIN MS25 1 340 130 -26 15 0 0 "Subst1" 1 "1e-3mm" 1 "10 mm" 1 "Hammerstad" 0
↪  "Kirschning" 0 "26.8" 0>
```

Appliquée à la ligne ci-dessus, on obtient :

- `1` en valeur absolue
- `e-3` en suffixe scientifique
- `mm` en suffixe d'ingénieur

3.3 Tracé d'un polygone concave avec OpenGL

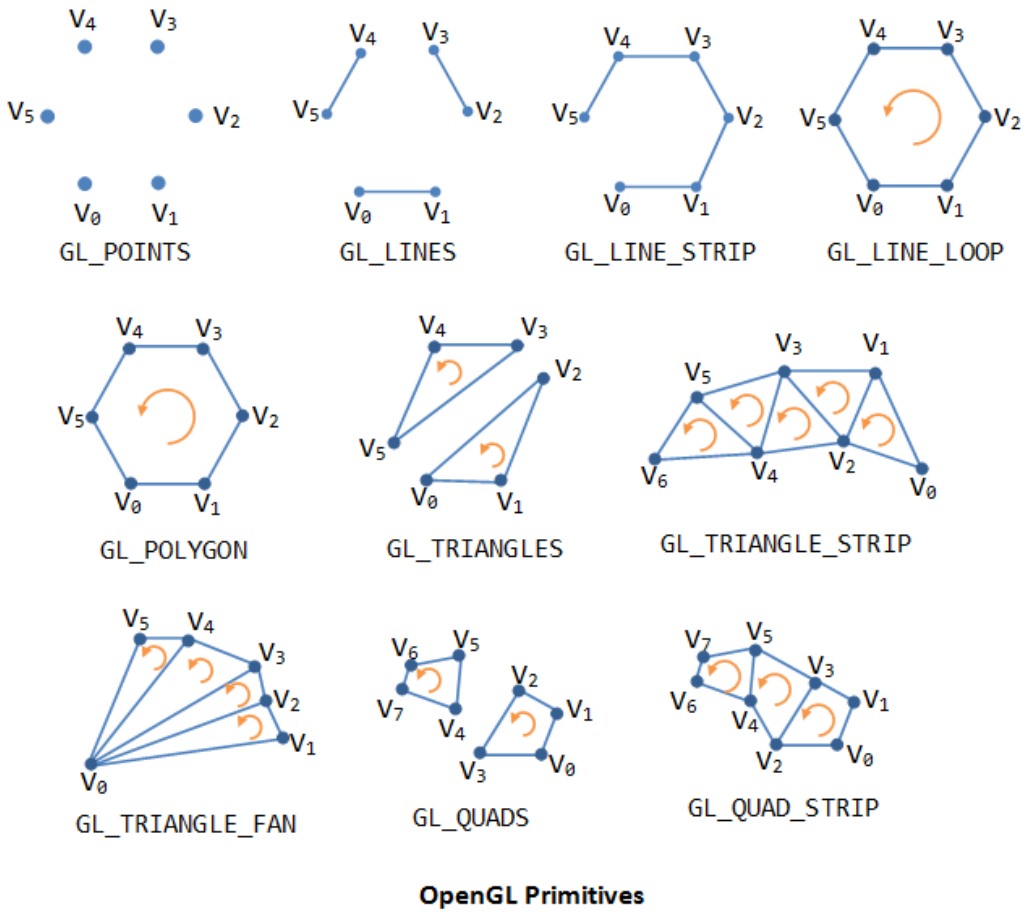


FIGURE 3.2 – Formes primitives d'OpenGL

OpenGL ne propose aucune forme primitive permettant de tracer un polygone concave. En effet la forme **GL_POLYGON** fonctionne de la même manière que **GL_TRIANGLE_FAN**, c'est-à-dire qu'il existe une arête entre le point d'origine et tous les autres points, ce qui fausse totalement la figure si l'on essaie de l'utiliser pour tracer un polygone concave.

Une technique consiste à utiliser le "stencil buffer", pochoir en français, qui permet de filtrer les contours d'un dessin. L'on trace une première fois le polygone concave dans le buffer, sans l'afficher sur l'image, puis on le trace une seconde fois, sur l'image cette fois ci, en utilisant le contenu du buffer pour délimiter les contours du polygone et empêcher qu'une "arête interne" ne morde l'angle concave du polygone.

Cela implique de dessiner deux fois au lieu d'une chacune des formes.

En terme de code, cela se traduit par le passage du premier code au second pour tracer le polygone concave désiré.

```
    glBegin(GL_POLYGON);
        glColor3f(0.5f, 1.0f, 0.0f);
        glVertex3f(-1.0f, -1.0f, 0.0f);
        glVertex3f(1.0f, -1.0f, 0.0f);
        glVertex3f(1.0f, 1.0f, 0.0f);
        glVertex3f(0.0f, 1.0f, 0.0f);
        glVertex3f(0.0f, 0.0f, 0.0f);
        glVertex3f(-1.0f, 0.0f, 0.0f);
    glEnd();
```

```
glEnable(GL_STENCIL_TEST);

glClear(GL_STENCIL_BUFFER_BIT);
glColorMask(GL_FALSE, GL_FALSE, GL_FALSE, GL_FALSE);
glStencilFunc(GL_ALWAYS, 0x1, 0x1);
glStencilOp(GL_KEEP, GL_INVERT, GL_INVERT);

    glBegin(GL_POLYGON);
        glColor3f(0.5f, 1.0f, 0.0f);
        glVertex3f(-1.0f, -1.0f, 0.0f);
        glVertex3f(1.0f, -1.0f, 0.0f);
        glVertex3f(1.0f, 1.0f, 0.0f);
        glVertex3f(0.0f, 1.0f, 0.0f);
        glVertex3f(0.0f, 0.0f, 0.0f);
        glVertex3f(-1.0f, 0.0f, 0.0f);
    glEnd();

glColorMask(GL_TRUE, GL_TRUE, GL_TRUE, GL_TRUE);
glStencilFunc(GL_EQUAL, 0x1, 0x1);
glStencilOp(GL_KEEP, GL_KEEP, GL_KEEP);

    glBegin(GL_POLYGON);
        glColor3f(0.5f, 1.0f, 0.0f);
        glVertex3f(-1.0f, -1.0f, 0.0f);
        glVertex3f(1.0f, -1.0f, 0.0f);
        glVertex3f(1.0f, 1.0f, 0.0f);
        glVertex3f(0.0f, 1.0f, 0.0f);
        glVertex3f(0.0f, 0.0f, 0.0f);
        glVertex3f(-1.0f, 0.0f, 0.0f);
    glEnd();
```

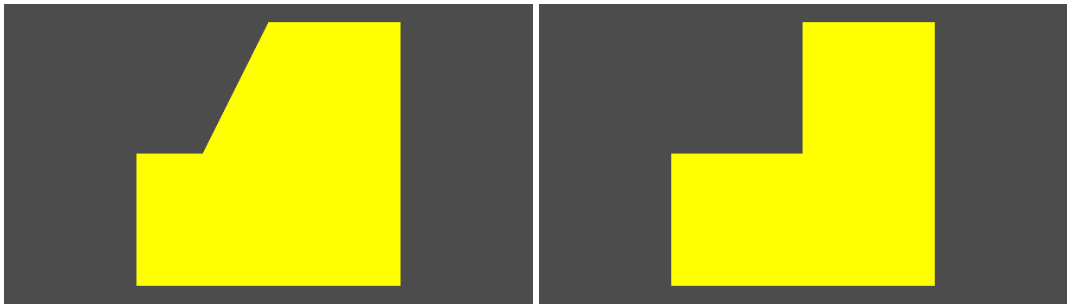


FIGURE 3.3 – Polygone concave sans et avec utilisation du stencil buffer

Chapitre 4

État actuel du projet

4.1 Travail effectué

À l'heure actuelle, le projet a atteint un état satisfaisant compte tenu de l'objectif de départ.

Le moteur de conversion fonctionne pour tous les éléments microruban à l'exception du coupleur à franges ("lange coupler" en anglais). En effet l'allure ce celui-ci sur son symbole ne semble pas correspondre à une topologie courante. L'élément n'étant pas documenté, un échange avec les développeurs va être nécessaire.

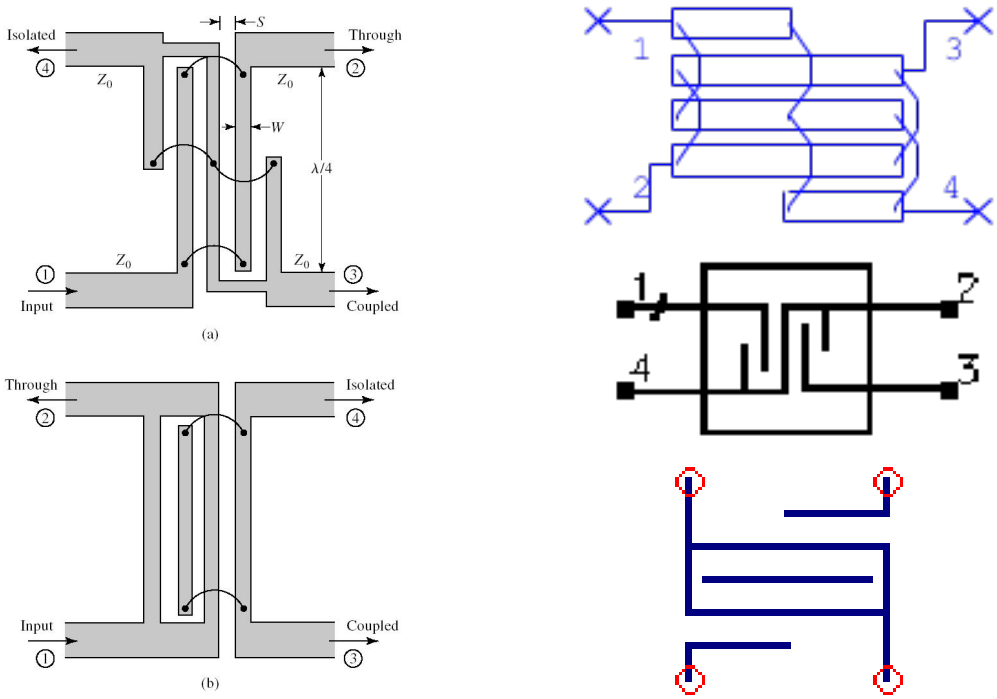


FIGURE 4.1 – Exemples de topologies de coupleur à frange à gauche et symboles de l'élément dans les logiciels AWR, ADS et Qucs à droite

Il supporte actuellement les formats de sortie suivants :

- PcbNew `.kicad_pcb` : Circuit imprimé complet.
- PcbNew `.kicad_mod` : Empreinte de composant.
- pcb-rnd `.lht` : Circuit imprimé complet.

L'interface graphique est opérationnelle ainsi que l'aperçu du typon qui supporte :

- Un typon en 3D.
- Des déplacements verticaux et horizontaux (respectivement scroll et **[SHIFT]** + scroll).
- Un zoom (**[CTRL]** + Scroll).
- Des rotation autour des axes $\overrightarrow{x}$; $\overrightarrow{y}$ ou $\overrightarrow{x}$; $\overrightarrow{z}$ (respectivement clic gauche glissé et clic droit glissé).

Ci-dessous des captures d'écran en guise de preuve de fonctionnement :

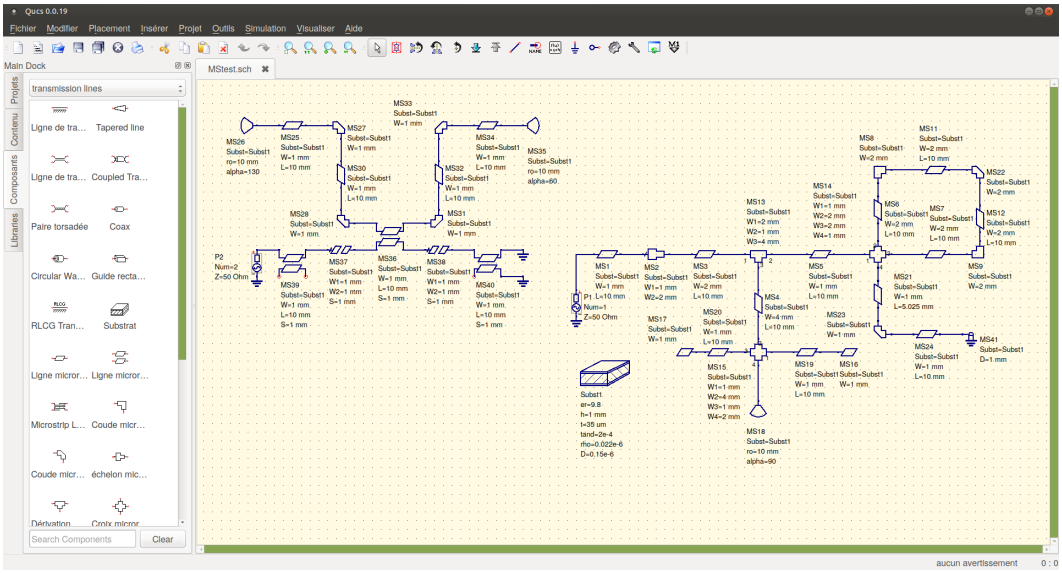


FIGURE 4.2 – Qucs : Schéma **.sch**

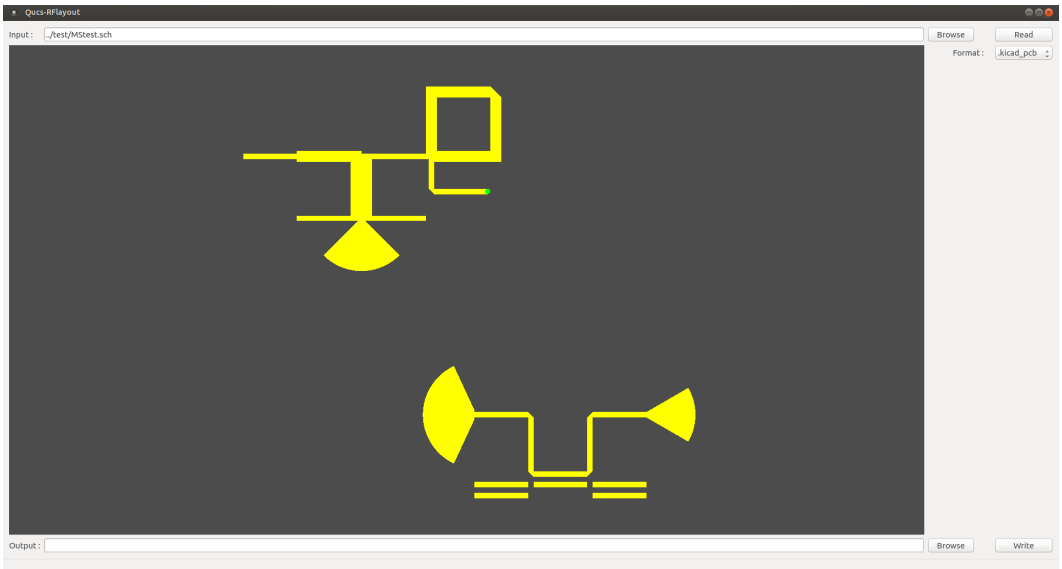


FIGURE 4.3 – Qucs-RFlayout : Interface graphique et aperçu du typon

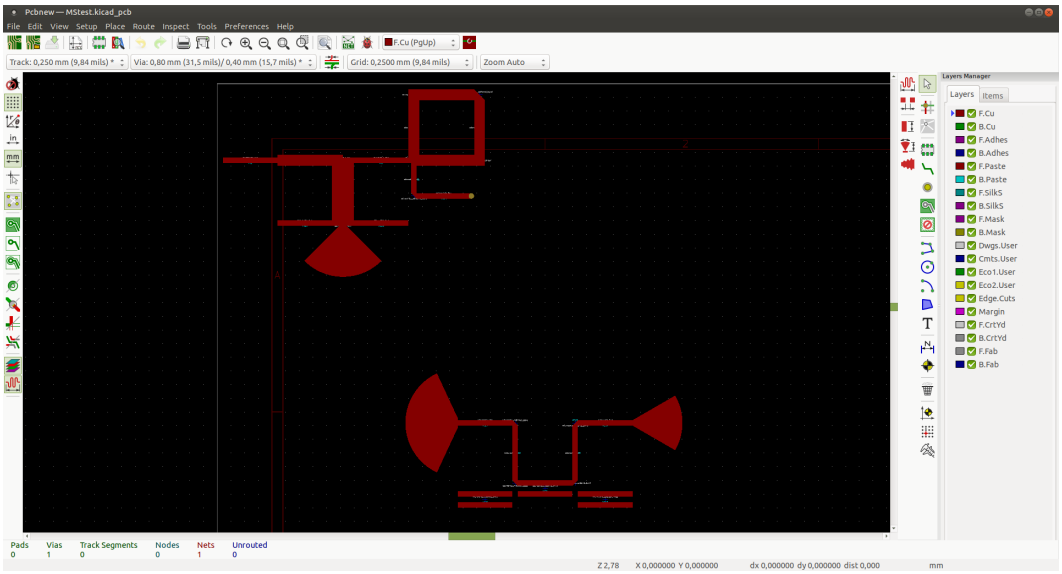


FIGURE 4.4 – PcbNew : Typon `.kicad_pcb`

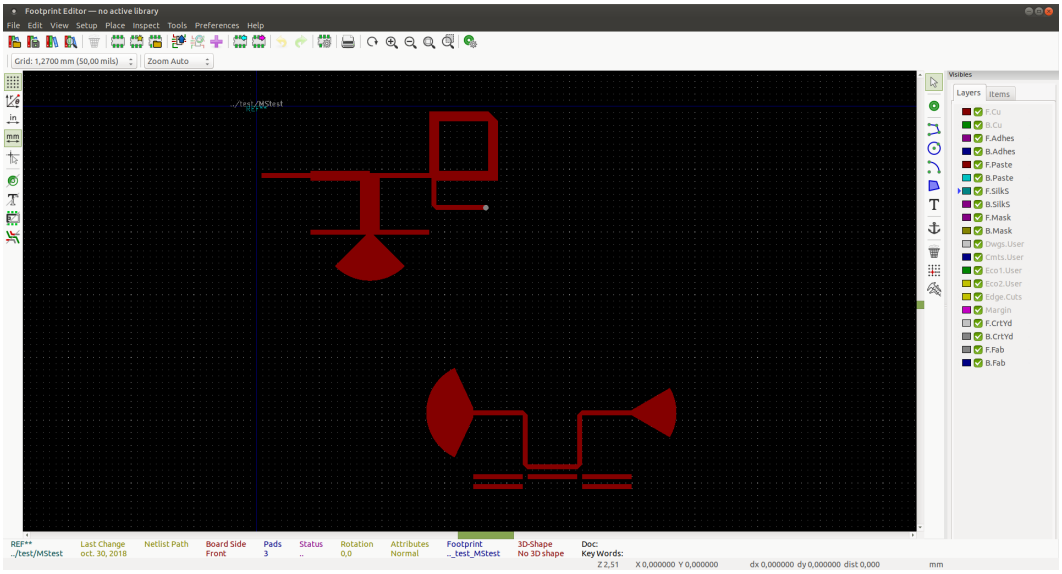


FIGURE 4.5 – PcbNew : Empreinte `.kicad_mod`

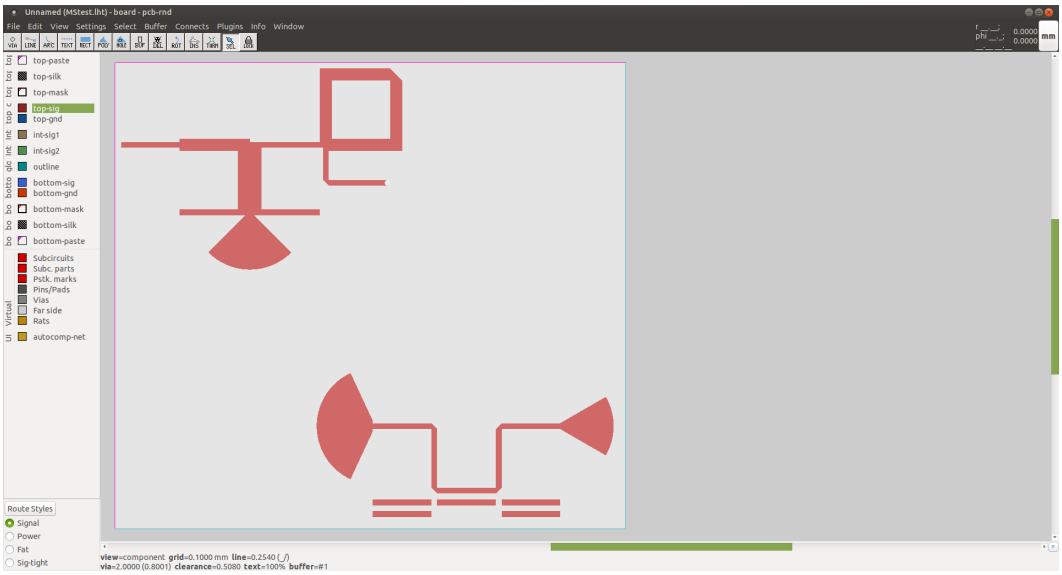


FIGURE 4.6 – pcb-rnd : Typon `.lht`

Le projet est disponible sur Github sous licence GPL-2.0 (licence utilisée pour Qucs) à l'adresse suivante : **`https://github.com/thomaslepoix/Qucs-RFlayout`**.

4.2 Travail à venir

4.2.1 Nouvelles fonctionnalités

- L'ajout du support des équations dans les valeurs que prennent les éléments est une fonctionnalité à forte valeur ajoutée puisque celles-ci font partie de Qucs et qu'il est très contraignant de devoir les résoudre manuellement avant de pouvoir convertir le schéma en typon. Cette fonctionnalité peut cependant être compliquée à programmer étant donné que de nombreuses fonctions mathématiques sont disponibles. De la documentation au sujet des équations est disponible à ces adresses :
  `http://qucs.sourceforge.net/docs/tutorial/functions.pdf`
  `http://qucs.sourceforge.net/docs/tutorial/equations.pdf`
  Il est probable que la meilleure solution d'ajouter cette fonctionnalité soit de réutiliser le résolveur d'équation de Qucs. Il faudra donc attendre que le projet atteigne un état plus avancé pour implémenter cela.
- Le support de l'élément substrat **`SUBST`** qui contient des informations comme l'épaisseur du diélectrique du PCB utilisé h, l'épaisseur du conducteur t, la permittivité relative du diélectrique ϵ_r, le facteur de perte $tan\delta$, la résistivité du conducteur ρ ou encore la rugosité quadratique du conducteur D. Cela est un prérequis à l'ajout des scripts OpenEMS `.m` comme format de sortie puisque ces données (géométriques ou physiques) sont utilisées pour les simulations.
- Le support des scripts OpenEMS comme format de sortie demandera de plus une étude plus poussée des techniques de maillage automatique que propose le logiciel, les essais faits lors des TPs d'hyperfréquence s'étant avérés plutôt hasardeux.
- L'ajout à l'interface graphique du système permettant de modifier la géométrie du typon dans la limite de la pertinence. Comme expliqué précédemment, cette condition

imposera un long travail de recherches avant de pouvoir envisager de l'implémenter. Chose qui pourra également s'avérer compliquée.
Cette fonctionnalité étant de faible valeur ajoutée pour le logiciel, elle ne fait clairement pas partie des priorités.

- L'ajout à l'interface graphique d'une ligne ou d'une console permettant d'afficher des messages témoins de l'activité du logiciel (par exemple "Écriture dans le fichier <...> : OK")
- L'ajout du support des éléments en technologie guide d'onde coplanaire. Ainsi que du coupleur à franges microruban.

4.2.2 Documentation

- L'utilisation de Doxygen pour générer une documentation technique du programme au format web ou `.pdf`.
- La rédaction d'une documentation utilisateur, aussi bien dans un menu d'aide du programme, que dans un document `.pdf` indépendant, que dans la page de documentation du site web de Qucs. Les pages propres à chaque composant semblent des endroits appropriés :
  `http://qucs.github.io/tech/technical.html`
  Ou bien une page dédiée à Qucs-RFlayout conviendrait mieux. La procédure de modification du site web ayant déjà été mise en œuvre pour ajouter un rapport de TP d'hyperfréquence sur la conception de filtres microruban à la page recensant les documentations écrites par des tierces personnes :

Qucs
Quite universal circuit simulator

home screenshots support download roadmap development FAQ

[user manual - third party - technical papers - components - related - press/education - contact]

Documentation provided by third parties

The following list contains hypertext links to documents provided by people not directly involved in the development of Qucs. The authors of Qucs do not feel responsible for the documents' content.

- French student practical work about the design of microstrip filters by Thomas Lepoix, 2017 -- Qucs - Design of microstrip filters
- Online tutorial by Ozgur Cobanoglu -- Qucs Tutorial - DC Parameter Sweep
- Quite Universal Circuit Simulator White Paper by Brian Clearwaters, 2006 -- White Paper
- Turkish online tutorial by Mustafa Baser -- Qucs for beginner physicists
- Japanese online introduction -- Introduction to Qucs
- Wikipedia entry for Qucs -- Qucs - Wikipedia
- Publication by Mustafa Baser, 2006 -- Promoting Conceptual Change with Free Open Source Software through Active Learning
- Publication by Mustafa Baser -- Effects of Conceptual Change and Traditional Confirmatory Simulations on Pre-Service Teachersí Understanding of Direct Current Circuits
- Publication by Mike Brinson -- GNU Simulators Supporting Verilog-A Compact Model Standardization
- Russian tutorial about Qucs by Vladimir Gololobov -- Qucs and FlowCode

© 2004 - 2017 Qucs team

FIGURE 4.7 – Capture d'écran de la page **`https://qucs.github.io/thirdparty.html`**

Cette documentation devra contenir avant tout les schémas topologiques des éléments tels qu'implémentés ainsi que les tables d'orientation de ceux-ci en fonction des variables rotation et symétrie en $\overrightarrow{x}$.

- L'écriture, au minimum dans un fichier **`CONTRIBUTE`** parmi les sources, d'une méthode permettant d'implémenter le support d'un nouveau format de sortie.

4.2.3 Refactoring

- Gérer les cas d'erreur en utilisant des exceptions, c'est-à-dire avec les mots clés **`try`**, **`catch`** et **`throw`**. Cela est une façon de faire plus acceptable que l'utilisation des conditions.
- Une révision de la cohérence des noms de variable, de fonction et des conventions de nommage. L'utilisation des _ au sein de noms en minuscules est une préférence personnelle face au CamelCase et sera utilisée en priorité.
- L'utilisation de la fonction Qt **`tr()`**, utilisée dans Qucs pour traduire les chaînes de caractères. Ainsi que la traduction en français du logiciel.

4.2.4 Intégration à Qucs

- L'utilisation de CMake comme moteur de production puisqu'il est utilisé par le projet Qucs.

- Une phase d'étude de la façon dont les logiciels du menu `Outils` sont intégrés à Qucs. Il est probable qu'il convienne de s'inscrire sur la mailing list du logiciel et de s'entretenir avec les développeurs à ce sujet.
- Intégrer Qucs-RFlayout à Qucs sur un fork de celui-ci avant d'ouvrir une pull-request vers le dépôt officiel du projet. Il est impératif que le logiciel ait une maturité suffisante et que les éléments listés plus haut soient terminés.

Annexe A

Liens et références

A.1 Sites des projets connexes

- Qucs-RFlayout : **https://github.com/thomaslepoix/Qucs-RFlayout**
- Qucs : **http://qucs.github.io/**
- PcbNew : **http://kicad-pcb.org/discover/pcbnew/**
- pcb-rnd : **http://repo.hu/projects/pcb-rnd/**
- OpenEMS : **http://openems.de/start/**
- Octave : **https://www.gnu.org/software/octave/**

A.2 Bibliographie

- Cours sur les pointeurs intelligents :
  https://loic-joly.developpez.com/tutoriels/cpp/smart-pointers/
- Cheat sheet et cours sur les expressions régulières :
  http://www.rexegg.com/regex-quickstart.html
- Outil de debug des expressions régulières :
  https://regex101.com/
- Exemple d'utilisation d'OpenGL :
  https://bogotobogo.com/Qt/Qt5_OpenGL_QGLWidget.php
- Format de donnée des schémas Qucs :
  https://qucs-help.readthedocs.io/en/latest/internal.html
- Format de donnée S-expression :
  http://kicad-pcb.org/help/file-formats/
- Format de donnée lihata :
  http://repo.hu/projects/pcb-rnd/developer/lihata_format/
- Documentation de National Instruments - AWR :
  https://awrcorp.com/download/faq/english/docs/Elements/ELEMENTS.htm
- Documentation de Keysight - ADS :
  http://literature.cdn.keysight.com/litweb/pdf/ads2002c/dgpas/index.html