Modélisation Objet -- MOBJ

Contents

TME 6 -- Parser XML

Les deux TME précédents (4 & 5) nous ont permis de créer la structure de données NETLIST, ainsi qu'un *driver* XML. Le driver XML est réalisé par l'assemblage des méthodes membre **`toXml()`** des différentes classes. Il suffit juste d'utiliser un fichier comme flux de sortie.

L'objectif du TME 6 est d'effectuer l'opération inverse: lire un fichier XML et recréer la structure de données en mémoire.

Pour cela nous allons utiliser la librairie **`libxml2`**, formellement partie de GNOME, mais pouvant être utilisée de façon complètement autonome. Cette librairie est écrite en C et non pas en C++.

Pour utiliser la bibliothèque, il vous faut ajouter l'*include* suivant à vos fichiers:

```
#include  <libxml/xmlreader.h>
```

La documentation complète de l'interface XML *stream* est accessible ici:

`http://xmlsoft.org/html/libxml-xmlreader.html`

Note

La librairie **`libxml2`** utilise le type `xmlChar` (`xmlChar*`). Dans le cadre des TME on pourra faire des *cast* directs entre `xmlChar` et `char`.

Arborescence du Code & `cmake`

Modifications à apporter au fichier de configuration `CMakeLists.txt` de `cmake` du TME45. On ajoute la détection de la **`libxml2`**, les chemins pour ses *includes* et la bibliothèque à l'édition de lien. Ne pas oublier de renommer le binaire de `tme45` à `tme6`.

```
# Detection de LibXml2.
find_package(LibXml2)

# Trouver les includes de LibXml2.
   include_directories( ${SCHEMATIC_SOURCE_DIR} ${LIBXML2_INCLUDE_DIR} )

# [...]

# Faire l'edition de liens avec la LibXml2.
       add_executable ( tme6 ${cpps} )
target_link_libraries ( tme6 ${LIBXML2_LIBRARIES} )
```

Note

Rappel la méthode pour organiser votre code et compiler est décrite dans procédure de compilation.

Note

Pour que ce changement prenne effet au niveau du `Makefile`, n'oubliez pas de relancer la commande `cmake`.

Parseur XML d'une Netlist

L'organisation du parseur XML a été décrite en cours, en particulier son organisation répartie dans les différentes classes. Le code de ces méthodes statiques vous est fourni pour la classe `Cell`.

Fichier Utilitaires `XmlUtil`

Les fonctions `xmlCharToString()` et `xmlGetIntAttribute()` vous sont fournies dans les fichiers:

- XmlUtil.h
- XmlUtil.cpp

Méthode `Cell::fromXml()`

Code de la fonction membre statique `Cell::fromXml()`:

```
Cell* Cell::fromXml ( xmlTextReaderPtr reader )
{
  enum  State { Init           = 0
              , BeginCell
              , BeginTerms
              , EndTerms
              , BeginInstances
              , EndInstances
              , BeginNets
              , EndNets
              , EndCell
              };
```

```
const xmlChar* cellTag      = xmlTextReaderConstString( reader, (const xmlChar*
const xmlChar* netsTag      = xmlTextReaderConstString( reader, (const xmlChar*
const xmlChar* termsTag     = xmlTextReaderConstString( reader, (const xmlChar*
const xmlChar* instancesTag = xmlTextReaderConstString( reader, (const xmlChar*

Cell* cell   = NULL;
State state  = Init;

while ( true ) {
  int status = xmlTextReaderRead(reader);
  if (status != 1) {
    if (status != 0) {
      cerr << "[ERROR] Cell::fromXml(): Unexpected termination of the XML parse
    }
    break;
  }

  switch ( xmlTextReaderNodeType(reader) ) {
    case XML_READER_TYPE_COMMENT:
    case XML_READER_TYPE_WHITESPACE:
    case XML_READER_TYPE_SIGNIFICANT_WHITESPACE:
      continue;
  }

  const xmlChar* nodeName = xmlTextReaderConstLocalName( reader );

  switch ( state ) {
    case Init:
      if (cellTag == nodeName) {
        state = BeginCell;
        string cellName = xmlCharToString( xmlTextReaderGetAttribute( reader,
        if (not cellName.empty()) {
          cell = new Cell ( cellName );
          state = BeginTerms;
          continue;
        }
      }
      break;
    case BeginTerms:
      if ( (nodeName == termsTag) and (xmlTextReaderNodeType(reader) == XML_REA
        state = EndTerms;
        continue;
      }
      break;
    case EndTerms:
      if ( (nodeName == termsTag) and (xmlTextReaderNodeType(reader) == XML_REA
        state = BeginInstances;
        continue;
      } else {
        if (Term::fromXml(cell,reader)) continue;
      }
      break;
    case BeginInstances:
      if ( (nodeName == instancesTag) and (xmlTextReaderNodeType(reader) == XMI
```

```
            state = EndInstances;
            continue;
          }
          break;
        case EndInstances:
          if ( (nodeName == instancesTag) and (xmlTextReaderNodeType(reader) == XMI
            state = BeginNets;
            continue;
          } else {
            if (Instance::fromXml(cell,reader)) continue;
          }
          break;
        case BeginNets:
          if ( (nodeName == netsTag) and (xmlTextReaderNodeType(reader) == XML_READ
            state = EndNets;
            continue;
          }
          break;
        case EndNets:
          if ( (nodeName == netsTag) and (xmlTextReaderNodeType(reader) == XML_READ
            state = EndCell;
            continue;
          } else {
            if (Net::fromXml(cell,reader)) continue;
          }
          break;
        case EndCell:
          if ( (nodeName == cellTag) and (xmlTextReaderNodeType(reader) == XML_READ
            continue;
          }
          break;
        default:
          break;
      }

      cerr << "[ERROR] Cell::fromXml(): Unknown or misplaced tag <" << nodeName
           << "> (line:" << xmlTextReaderGetParserLineNumber(reader) << ")." << end
      break;
    }

    return cell;
  }
```

Lecture & Écriture sur Disque

Code de la méthode statique `Cell::load()`, pour lire une netlist depuis le disque. Elle essaye de lire un fichier de même nom que la *netlist* (avec une extension `.xml`) dans un sous répertoire `./cells` *par rapport au répertoire courant*:

```
Cell* Cell::load ( const string& cellName )
{
  string           cellFile = "./cells/" + cellName + ".xml";
  xmlTextReaderPtr reader;

  reader = xmlNewTextReaderFilename( cellFile.c_str() );
  if (reader == NULL) {
    cerr << "[ERROR] Cell::load() unable to open file <" << cellFile << ">." << e
    return NULL;
  }

  Cell* cell = Cell::fromXml( reader );
  xmlFreeTextReader( reader );

  return cell;
}
```

Code de la méthode `Cell::save()`, pour écrire une netlist sur le disque:

```
void  Cell::save () const
{
  string  fileName   = getName() + ".xml";
  fstream fileStream ( fileName.c_str(), ios_base::out|ios_base::trunc );
  if (not fileStream.good()) {
    cerr << "[ERROR] Cell::save() unable to open file <" << fileName << ">." << e
    return;
  }

  cerr << "Saving <Cell " << getName() << "> in <" << fileName << ">" << endl;
  toXml( fileStream );

  fileStream.close();
}
```

Fichiers d'exemples

Vous sont fournis les fichiers `xml` suivants:

- and2.xml
- xor2.xml
- or2.xml
- halfadder.xml

Ils sont à copier dans le répertoire `<>/work/cells/`.

Travail à Réaliser

Déboguage progressif: pour tester les fonctions de chargement XML on pourra partir d'un fichier XML où les balises correspondant aux parties non encore supportées du parseur seront en commentaires. Puis au fur et à mesure, on active les morceaux commentés.

Fonction `main()` pour ce TME:

```
int main ( int argc, char* argv[] )
{
  cout << "Chargement des modeles:" << endl;
  cout << "- <and2> ..." << endl;
  Cell::load( "and2" );

  cout << "- <or2> ..." << endl;
  Cell::load( "or2" );

  cout << "- <xor2> ..." << endl;
  Cell::load( "xor2" );

  cout << "- <halfadder> ..." << endl;
  Cell* halfadder = Cell::load( "halfadder" );

  cout << "\nContenu du <halfadder>:" << endl;
  halfadder->toXml( cout );

  return 0;
}
```

Question 1

Implanter `Term* Term::fromXml(Cell*, xmlTextReaderPtr)`. En cas d'erreur elle renverra un pointeur `NULL`.

Question 2

Implanter `Instance* Instance::fromXml(Cell*, xmlTextReaderPtr)`, doit renvoyer `NULL` en cas d'erreur.

Question 3

Implanter `Net* Net::fromXml(Cell*, xmlTextReaderPtr)`, doit renvoyer `NULL` en cas d'erreur.

Question 4

Implanter `bool Node::fromXml(Net*, xmlTextReaderPtr)`, doit renvoyer `false` en cas d'erreur.

Note

La méthode `Node::fromXml()` ne crée pas de nouvel objet. Elle réalise la connexion entre un `Net` et un `Term`. Deux cas se présentent:

1. L'attribut `instance` n'est pas présent, alors il s'agit d'une connexion à un terminal de la `Cell`.
2. L'attribut `instance` est présent, alors il s'agit d'un terminal de ladite instance.

Question 5

Test en boucle: une fois le parseur terminé, afficher le résultat avec `Cell::toXml()` et comparer au fichier original avec la commande UNIX `diff`. Si tout s'est bien passé, les fichiers doivent être identiques. Comparez avec la commande UNIX `diff -w` pour ignorer les espaces.

Petit Memento de `libxml2`

Allocation, Désallocation & Parcours de l'Arbre
`xmlTextReaderPtr xmlNewTextReaderFilename(const char*)` Crée un parseur `xmlstream` à partir du contenu d'un fichier.
`void xmlFreeTextReader(xmlTextReaderPtr)` Libère un parseur `xmlstream`
`void xmlTextReaderRead(xmlTextReaderPtr)` Passe au noeud suivant dans le `xmlstream`. Retourne 1 si nous sommes sur le noeud suivant, 0 si le `xmlstream` est terminé et -1 en cas d'erreur.
Fonctions Utilitaires
`int xmlTextReaderGetParserLineNumber(xmlTextReaderPtr)` Retourne le numéro de ligne dans le fichier où se trouve le noeud
`const xmlChar* xmlTextReaderConstString (xmlTextReaderPtr,const xmlChar*)` Alloue une chaîne de caractère dans le `reader` qui pourra ensuite être comparée par pointeur
`string xmlCharToString(xmlChar*)` Traduit une chaîne de caractères `xmlChar` en `string`. La chaîne `xmlChar` est désallouée par cette fonction
Accès aux Caractéristiques du Noeud Courant
`int xmlTextReaderNodeType(xmlTextReaderPtr)` Retourne le type du noeud. Une valeur de l'`enum xmlReaderTypes`. Nous gèrerons: • `XMLREADERTYPEELEMENT` --- `<mytag>`. • `XMLREADERTYPEENDELEMENT` --- `</mytag>` ou `<mytag/>`. • `XMLREADERTYPEWHITESPACE` ou `XMLREADERTYPESGNIFICANTWHITESPACE`.
`const xmlChar* xmlTextReaderConstLocalName (xmlTextReaderPtr)` Retourne le nom du *tag* du noeud courant
`xmlChar* xmlTextReaderGetAttribute (xmlTextReaderPtr, const xmlChar*)` Retourne la valeur (chaîne de caractères) de l'attribut dont le nom est donné en argument