UE MOBJ [4L103]

Jean-Paul CHAPUT
Jean-Paul.Chaput@lip6.fr

SESI

2018-2019

V.1

```
ENTITY halfadder IS
PORT ( a    : IN  std_logic;
       b    : IN  std_logic;
       sout : OUT std_logic;
       cout : OUT std_logic;
     );
END halfadder;
ARCHITECTURE structural OF component IS
  xor_1 : xor port map (
    i0 => a,
    i1 => b,
    q  => sout
  );
  and_1 : and2 port map (
    i0 => a,
    i1 => b,
    q  => cout
  );
BEGIN
END structural;
```

V.2

```
<?xml version="1.0"?>
<cell name="halfadder">
  <terms>
    <term name="a"    direction="In"  x="0"   y="120"/>
    <term name="b"    direction="In"  x="0"   y="200"/>
    <term name="sout" direction="Out" x="300" y="80" />
    <term name="cout" direction="Out" x="300" y="230"/>
  </terms>
  <instances>
    <instance name="xor_1" masterCell="xor2" x="150" y="200"/>
    <instance name="and_1" masterCell="and2" x="150" y="50" />
  </instances>
  <nets>
    <net name="a" type="External">
      <node name="a"                    id="0"/>
      <node name="i0" instance="xor_1" id="1"/>
      <node name="i0" instance="and_1" id="2"/>
    </net>
    <!-- To be continued -->
  </nets>
</cell>
```

V.3

```
Cell* Cell::fromXml ( xmlTextReaderPtr reader ) {
  while ( true ) {
    int status = xmlTextReaderRead(reader);
    if (status != 1) {
      if (status != 0) {
        cerr << "[ERROR] Cell::fromXml():"
                "Unexpected termination of the XML parser." << endl;
      }
      break;
    }
    switch ( xmlTextReaderNodeType(reader) ) {
      case XML_READER_TYPE_WHITESPACE:
      case XML_READER_TYPE_SIGNIFICANT_WHITESPACE:
        continue;
    }
    // [...] Traitement des noeuds --> appel a continue.

    // Fin de boucle atteinte --> erreur.
  } // end while(true)
}
```

V.3

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

    const xmlChar* cellTag
      = xmlTextReaderConstString(reader,(const xmlChar*)"cell");
    const xmlChar* netsTag
      = xmlTextReaderConstString(reader,(const xmlChar*)"nets");
    const xmlChar* termsTag
      = xmlTextReaderConstString(reader,(const xmlChar*)"terms");
    const xmlChar* instancesTag
      = xmlTextReaderConstString(reader,(const xmlChar*)"instances");
}
```

V.3

```
Cell* cell   = NULL;
State state  = Init;
while(true) {
  const xmlChar* nodeName = xmlTextReaderConstLocalName(reader);
  switch( state) {
    case Init:
      if (cellTag == nodeName) {
        state = BeginCell;
        string cellName = xmlCharToString
          (xmlTextReaderGetAttribute(reader,(const xmlChar*)"name"));
        if (not cellName.empty()) {
          cell = new Cell(cellName);
          state = BeginNets;
          continue; // OK, on passe au noeud suivant.
        } else
          state = ParseError; // KO, pas de continue.
      }
   // [...] to be continued.
  } // end switch(state).
}
```

V.3

```
while(true) {
  switch( state) {
    // [...] Traitement des etats precedents.
    case BeginTerms:
      if ((nodeName == termsTag) and
          (xmlTextReaderNodeType(reader)==XML_READER_TYPE_ELEMENT)) {
        state = EndTerms;
        continue; // OK, transition <cell> --> <terms>.
      }
    case EndTerms:
      if ((nodeName == termsTag) and
          (xmlTextReaderNodeType(reader)==XML_READER_TYPE_END_ELEMENT))
        state = BeginInstances;
        continue; // OK, transition </terms> --> <instances>.
      } else {
        if (Term::fromXml(cell,reader)) continue; // OK, <term/>.
      }
    // [...] Traitement des etats suivants.
  }
}
```

V.3

```
while(true) {
  switch(state) {
    // [...] Traitement des etats precedents.
    case EndCell:
      if ((nodeName == cellTag) and
          (xmlTextReaderNodeType(reader)==XML_READER_TYPE_END_ELEMENT))
        continue; // OK, </cell>.
      }
    default:
      break;
  } // End switch(state).

  cerr << "[ERROR] Cell::fromXml(): Unknown or misplaced tag <"
       << nodeName << "> (line:"
       << xmlTextReaderGetParserLineNumber(reader)
       << ")." << endl;
  break;
} // End while(true).
```