NAME

seadif - language for sea-of-gates data representation

DESCRIPTION

Seadif is a language that is specially suited for data representation in a sea-of-gates database. It essentially is a subset of the E
sion 2.0.0, but it has a number of enhancements. These enhancements mainly deal with convenient layout representation an
consists of four levels. At the top are the *libraries*, at the second level are the *functions*, at the third level are the *circuits* and
the meta hierarchy is to provide a selection mechanism: each function lists a number of functional equivalent circuits and each

The Seadif language is an "open" language. It can be extended while remaining compatibility with the older version. The sead
ments and processes only the elements that it recognizes. The following five subsections discuss the Seadif statements curr
mentioned in *sealib (3SDF)*. Meaning of the --rather informal-- syntax used below to describe the syntax of the Seadif langua

<statement> is a statement or a terminal symbol like
a <string> or a <number>, see below;
<statement>? means zero or one time <statement>;
<statement>* means zero, one or more times <statement>;
<aa> | <bb> means either <aa> or <bb>, not both;
{ <aa> <bb> } groups two statements <aa> and <bb> to
syntacticly form a single statement.

All other symbols represent themselves. Spaces, tabs and newlines are all equivalent. Left and right parenthesis do not need
language element need either spaces or parenthesis surrounding them. For most statements the order in which they appear is
the LaySlice statement (see below). It also is not true for terminal symbols of type <string> and type <number>.

Terminal symbols of type <number> can be octal, decimal or hex and the format corresponds to the C-syntax, that is, leading
19 = 0x13. Terminal symbols of type <string> should be quoted with double-quotes (") and they may contain any character e
require quotes but they do require a leading percent (%) if the first position is numeric. For instance, "2towers" is equivalent to
dition...)

The following terminal symbols are <strings>s: <libraryname>, <functionname>, <circuitname>, <cirportname>, <cirins
tributestring> and <layoutname>.

The following terminal symbols are <number>s: <layernumber>, <xposition>, <yposition>, <xlength>, <ylength>, <xoffset>,
<mtx5>, <xleft>, <xright>, <ybottom> and <ytop>.

SEADIF

The Seadif statement is the root of the meta hierarchy. It serves no other purpose than grouping a set of libraries and image des

```
<Seadif> ::= (Seadif <string>
             <Status>?
             <SeadifImage>*
             <Library>* )
```

For a description of <SeadifImage> refer to *sdfimage (4SDF)*. The <Status>, <SeadifImage> and <Library> statements may o

LIBRARY

A library groups a set of related functions that are available in a certain technology. Synopsis:

```
<Library> ::= (Library <libraryname>
              <Technology>?
              <Status>?
              <Function>* )

<Technology> ::= (Technology <string> )
```

The <Technology>, <Status> and <Function> statements may occur in any order.

FUNCTION

A function groups a set of functional equivalent circuits. Synopsis:

```
<Function> ::= (Function <functionname>
                <FunSimulate>?
                <FunType>?
                <Status>?
                <Circuit>* )

<FunType> ::= (FunType <string> )

<FunSimulate> ::= (FunSimulate <string> )
```

Normally, FunSimulate refers to a procedure that simulates the behavior of the function. FunType provides additional info
Type>, <Status> and <Circuit> statements may occur in any order.

CIRCUIT

A circuit describes a network of (yet other) circuits that implement the function. A circuit also groups layouts that have identi

```
<Circuit> ::= (Circuit <circuitname>
                <Status>?
                <CirPortList>?
                <CirInstList>?
                <NetList>?
                <BusList>?
                <Attribute>?
                <Layout>* )

<CirPortList> ::= (CirPortList <CirPort>* )

<CirPort> ::= (CirPort <cirportname> )

<CirInstList> ::= (CirInstList <CirInst>* )

<CirInst> ::= (CirInst <cirinstname> <CirCellRef>
                  <attributestring>? )

<CirCellRef> ::= (CirCellRef <circuitname>
                     <CirFunRef>? )

<CirFunRef> ::= (CirFunRef <functionname> <CirLibRef>? )

<CirLibRef> ::= (CirLibRef <libraryname> )

<NetList> ::= (NetList <Net>* )

<Net> ::= (Net <netname> <Joined> )

<Joined> ::= (Joined <NetPortRef>* )

<NetPortRef> ::= (NetPortRef <cirportname>
                     <NetInstRef>? )

<NetInstRef> ::= (NetInstRef <cirinstname> )

<BusList> ::= (BusList <bus>* )

<Bus> ::= (Bus <busname> <NetRef>* )
```

<Attribute> ::= (Attribute <attributestring>?)

<NetRef> ::= (NetRef <netname> <NetPortRef>∗)

The CirPortList is the list of i/o ports ("terminals") through which the circuit comunicates with other circuits. The CirInstList
(note: this is *not* the meta hierarchy). The NetList specifies the connections between the CirPorts on this (parent) circuit and
Sometimes it is convenient to think of a set of Nets as a bus. The Bus statement provides a means for net grouping. WARNI
<NetPortRef>s in a <NetRef>. The Attribute string specifies miscellaneous information about the circuit. For instance, the at
the channel length and width. The attributes of a circuit instance usually override the (default) attributes of the instantiated circ

LAYOUT

A layout statement describes the geometry of a circuit implementation. Synopsis:

```
<Layout> ::= (Layout <layoutname>
             <Status>?
             <LayPortList>?
             <LayBbx>?
             <LayOffset>?
             {<LayInstList> | <LaySlice>}?
             <WireList>? )

<LayPortList> ::= (LayPortList <LayPort>∗ )

<LayPort> ::= (LayPort <cirportname>
              <PortLayer>? <LayPort>? )

<PortLayer> ::= (PortLayer <layernumber> )

<PortPos> ::= (PortPos <xposition> <yposition> )

<LayBbx> ::= (LayBbx <xlength> <ylength> )

<LayOffset> ::= (LayOffset <xoffset> <yoffset> )

<LayInstList> ::= (LayInstList
                   <LayInst>∗
                   <LaySlice>∗
                   <LayInstList>∗ )

<LayInst> ::= (LayInst <layinstname> <LayCellRef>
              <Orient>? )

<Orient> ::= (Orient <mtx0> <mtx1> <mtx2>
                <mtx3> <mtx4> <mtx5> )

<LayCellRef> ::= (LayCellRef <layoutname> <LayCirRef>? )

<LayCirRef> ::= (LayCirRef <circuitname> <LayFunRef>? )

<LayFunRef> ::= (LayFunRef <functionname> <LayLibRef>? )

<LayLibRef> ::= (LayLibRef <libraryname> )

<LaySlice> ::= (LaySlice {vertical | horizontal | chaos}
               <LayInst>∗
```

<LaySlice>*
<layInstList>*)

<WireList> ::= (WireList <Wire>*)

<Wire> ::= (Wire <layernumber> <xleft> <xright>
<ybottom> <ytop>)

The LayPortList lists for each CirPort one or more geometrical representations (LayPorts). A LayPort occupies exactly one gr
PortLayer (and also of the Wire statement) is interpreted as follows. If <layernumber> is positive then the indicated layer is
the layer 0 - <layernumber> is removed from the position. Currently, the Oceanic sea-of-gates tools interpret the layers 1, 2 ar
between the image and first metal have <layernumber> 100, contacts between first and second metal have <layernumber> 101
ernumber> 102. The LayBbx statement declares the bounding box of the layout. The LayOffset specifies a shift with respe
LayInstList names the layout instances used. The LaySlice statement does exactly the same, but it also specifies a slicing pla
the arguments of LaySlice are placed from left to right. The modifier "vertical" specifies ordering from bottom to top. The
equivalent to LayInstList. The WireList lists all the rectangles that make up the layout. Interpretation of <layernumber> is as

AUTHORS

Patrick Groeneveld and Paul Stravers, Delft University of Technology.

SEE ALSO

sealib(3SDF), sdfread(3SDF), sdfwrite(3SDF), sdfopen(3SDF), sdfclose(3SDF), sdfimage(4SDF).