A converter from IDaSS 0.09 to synthesizable VHDL

Dassen, M.N.M.A.

Award date:
1996

Technische Universiteit Eindhoven

Faculty of Electrical Engineering
Section of Digital Information Systems

Master's Thesis

A converter from IDaSS 0.09 to synthesizable VHDL

M.N.M.A. Dassen

Supervisor : Prof. Ir. M.P.J. Stevens

Coach : Dr. Ir. A.C. Verschueren

February 1996

Summary

Modern integrated circuits are described is VHDL code. This code is translated to a set of masks by a silicon compiler.
The section Digital Information Systems at the Technical University Eindhoven systems is developing a complete digital system design path from specification to implementation (VHDL). IDaSS is the design and simulation tool.

IDaSS designs are written in a specific IDaSS language. The conversion from IDaSS code to VHDL code is performed by the IDaSS to VHDL converter.
For IDaSS version 0.08 a converter partly was designed at the moment IDaSS 0.09 was introduced. The new IDaSS version with new features creates a different language then the old version. With the structure of the IDaSS 0.08 to VHDL converter it isn't possible to convert a specific IDaSS 0.09 design.

In this report a new converter is described. This converter allows IDaSS 0.09 designs with the new features. The IDaSS 0.09 converter is creating a complete different data structure during the parsing of the design file. The different blocks aren't stored anymore in a hierarchical tree. The reason is the splitting of schematics in contents and boundaries.

In the new converter also several routines are improved. The most important change in the new converter is the handling of names. The names in VHDL are more like the original names. New in the IDaSS 0.09 converter is the use of parameters. Values or strings can replaced by parameters that are defined on a higher level in the design. The converter is connecting parameters to special signals that get on a certain level a value.

Table of contents

1 Introduction

The section Digital Information Processing systems is developing a complete digital system design path from specification to implementation. The tool which is developed to design and simulate a digital system is IDaSS. IDaSS is an **I**nteractive **D**esign **a**nd **S**imulation **S**ystem that can handle Ultra Large Scale Integration designs [VER 90].

The design IDaSS creates, is written in a specific IDaSS language [VER 95]. The common used silicon compilers (next step in process) need VHDL (**V**HSIC (very **H**igh **S**peed **I**ntegrated **C**ircuits) **H**ardware **D**escription **L**anguage).

Shortening the design time for digital circuits with IDaSS is only possible the IDaSS language can be converted to VHDL. This step is performed by the IDaSS to VHDL converter. The converter has to generate standard VHDL [STD 87] (usable for most silicon-compilers).

In 1994 W.M. Kruijtzer started to implement this converter [KRU 94]. The first step was a converter which only converted structural VHDL, which means that only the components and the connections between these were converted. In 1995 W.M. Kruijtzer implemented a converter that handled the behaviour of the components [KRU 95]. J.F. Pont extended the converter in 1995 [PON 95]. This converter is able to convert design from IDaSS version 0.08 to VHDL for the most common used blocks. In 1995 also, A.C. Verschueren completed the new IDaSS 0.09. This version contains a lot of new futures. The file format of this new IDaSS is also changed.

Adapting the existing converter to the new IDaSS version and implementing the new IDaSS features is described in this report.
This report is started with an overview of IDaSS with the new features. Second the changed structure of the compiler is described. In chapter 4 the tools are explained that complete the data structure of the converter to write VHDL. In the last part of the report, the structure of the VHDL code from an IDaSS design is shown.

2 The IDaSS design tool version 0.09

IDaSS is an interactive design and simulation system that is used to design digital circuits. The designs has to be described in a hierarchy of schematics, that contains registers, memories state machines buffers etc [VER 90].
With IDaSS it is possible to simulate the behaviour of the design. With viewers on buses, connectors and blocks, every design function can be monitored and every value can be

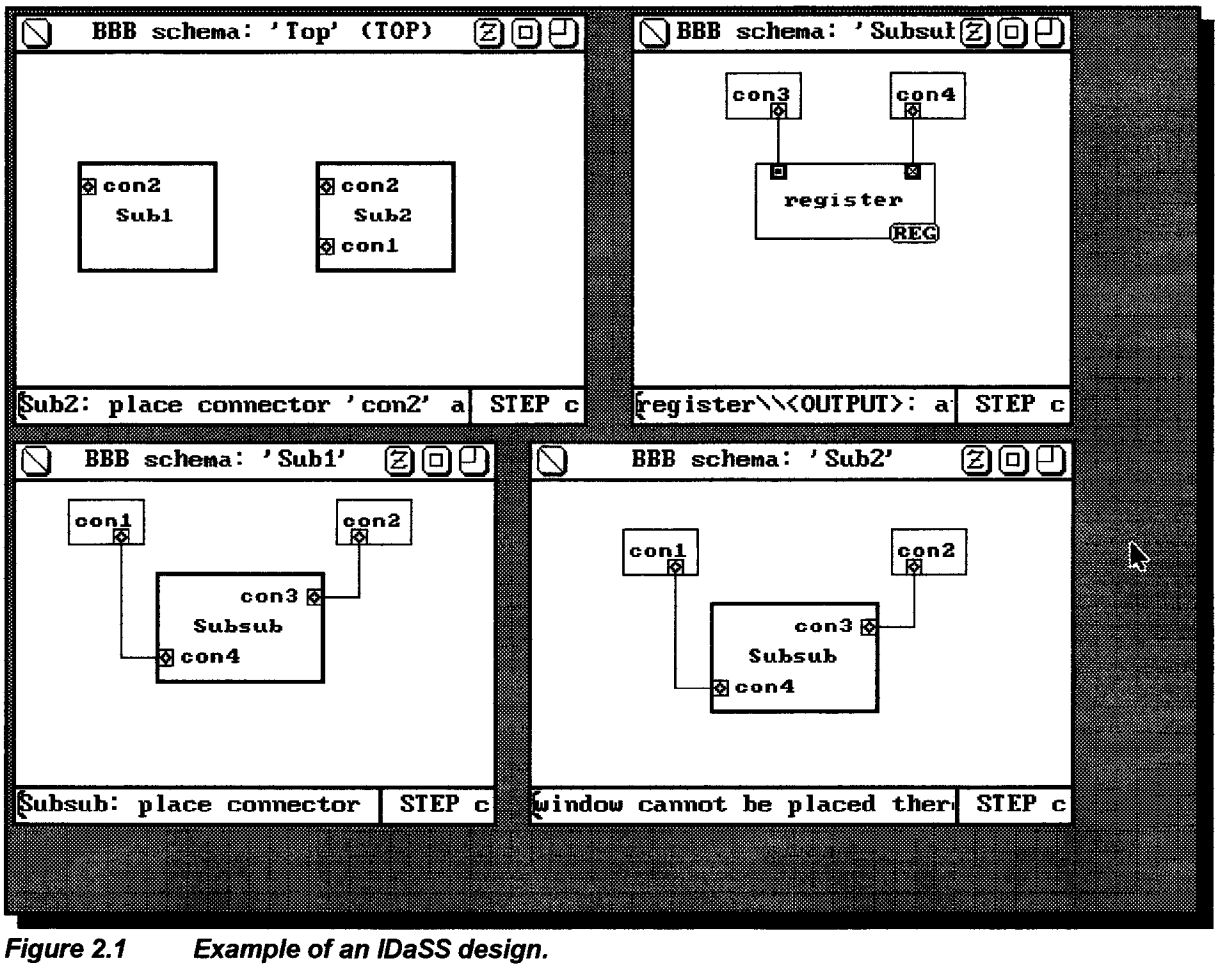

Figure 2.1 Example of an IDaSS design.

checked. The new IDaSS version has a couple of new possibilities. Schematics can be used several times in a design (specified once in the description file). Parameters can be used in behaviour descriptions, to specify reset values or file names. In section 2.3 a further explanation is given. This new version has a completely reviewed description file (DES file). In figure 2.1 an example of a design is given. This design is used in this report as example. In chapter 5.3 the VHDL code of this design is printed.

2.1 IDaSS schematics

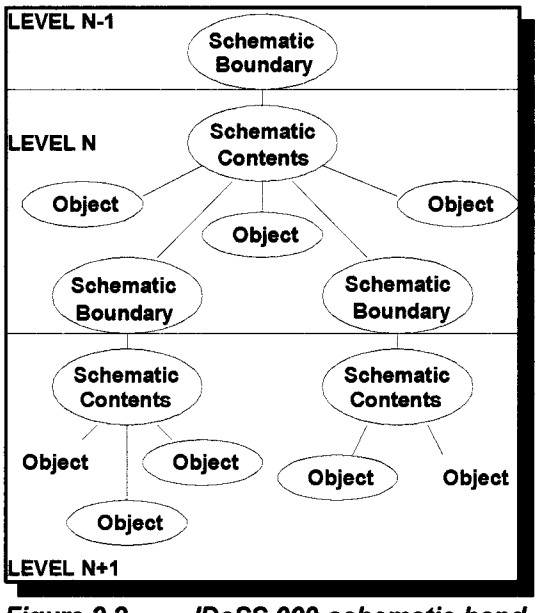

Figure 2.2 IDaSS 009 schematic handling.

Schematic in IDaSS 0.09 are split into two parts:

- The contents of the schematic.
- The boundary of the schematic.

In the contents specification the blocks with their behaviour and connections inside the schematic are described. If specified, also contents parameters are defined in this part of a schematic definition. The contents specification describes every possible contact on the schematic. For reused schematics there is just one contents definition. For every schematic (also for reused ones) there is a boundary specification. In this specification the connected contacts and the boundary parameters are described. In figure 2.2 the structure of the design from figure 2.1 is shown.

2.2 IDaSS parameters

In IDaSS 0.09 parameters are introduced. Their are four types of parameters which can be defined:

- Boundary specification parameters: Attached to a schematic boundary. Defined in design file.
- Contents specification parameters: Attached to a schematic contents. Defined in design file.
- Boundary instance parameters: Attached to a schematic boundary, are not saved in design file (used for simulation).
- Contents instance parameters: Attached to a schematics contents, are not saved in design file (used for simulation).

Parameters can be used in behaviour descriptions of operators, state-machines and control connectors. Also reset values and file names can be defined with parameters. At every schematic in IDaSS parameters can be defined. The contents parameters are equal for every schematic with the same contents (copies). If a parameter is defined at a schematic it automatically will be used for all the copies of this schematic. The boundary parameters are unique for every schematic. Copies of schematic can have different boundary parameters, or parameters with different definitions.
The sequence in which IDaSS is searching for a definition of a parameter is drawn in figure 2.3. Only the specified parameters (Specified in the Des-file) are used. If a parameter isn't defined in the current schematic (boundary or contents), IDaSS is searching a level higher for the parameter definition until the parameter is found.

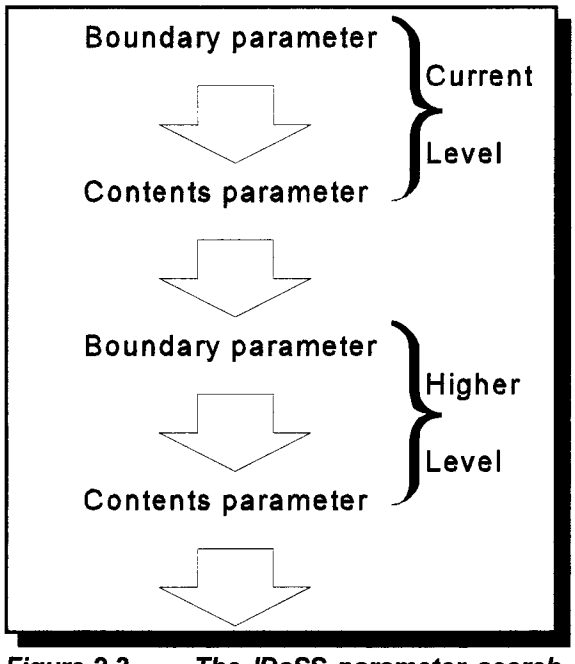

Figure 2.3 ***The IDaSS parameter searching sequence.***

The advantage of parameters is the possibility to change a complete set of values by changing one parameter. The second advantage of parameters is to use copies of schematics which behave not completely identical (using boundary parameters).

2.3 The IDaSS description file format

The complete description of the IDaSS 009 file format can be found in [VER 95]. In this section a brief overview is given to introduce a couple of important terms used in this report. Each line in a DES file starts with a switch character. The first line always starts with a header. This header begins with the IDaSS version. Next are the date and time of saving the design. After the header the first block is defined. Blocks are defined on the following way:

#Block blockname		#Schematic PentiumPro
.....	→Example:	.....
.Block blockname		.Schematic PentiumPro

Only description files that start with #TopSchematic (complete design saved) and #Schematic (Single schematic saved) are allowed to be converted with the converter.
The most important switch characters are:

#	Start description of a block
.	End description of a block
/	Line that contains graphical information of the IDaSS drawing: information not used in compiler.
‘	Line containing behaviour information of an object.
“	Line containing comment. In future this comment has to be copied into the VHDL files.

PB A definition of a boundary parameter follows

PC A definition of a contents parameter follows.

R A contents number follows. Every contents has a unique number. If the number is positive, the contents description follows (first use of the contents). If negative, no description follows (copies).

The complete range of switch characters can be found in [VER 95].
Next, the first lines of the Des-file from the example design are printed.

```
"IDaSS V0.09 02/14/96 10:04:36
#TopSchematic Top
/P? ?
R1
#Schematic Sub1
/P8@14 18@14
CBcon2 ?
/P0@2 3@0
R2
#SuperConnector con1
/P12@6 10@6
CBcon1 ?
/P6@4 ?
.SuperConnector con1
#SuperConnector con2
/P48@6 .....
```

3 Overview of the converter

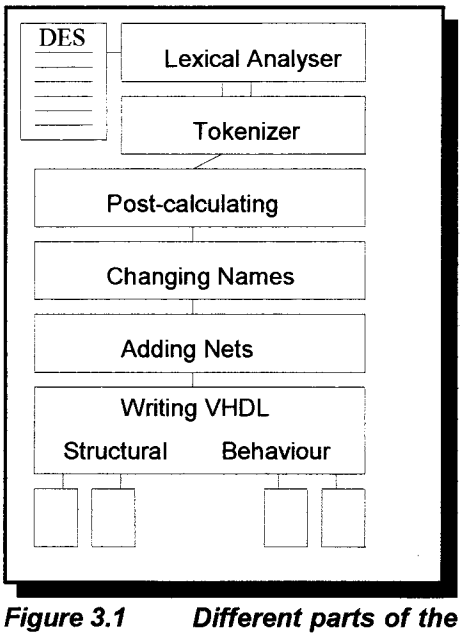

Figure 3.1 ***Different parts of the converter.***

In this chapter the design of the IDaSS 00.9 to VHDL converter is explained. The first step of converting a IDaSS file is reading the design file and storing the required design information into the memory of the converter. In the first paragraph of this chapter this is explained. After parsing the design file, the created data structure has to be completed and corrected, which is described in the next chapter. In the third paragraph new data structures of IDaSS elements are explained. In the last paragraph of this chapter the data structure of a design that contains re-used schematics is drawn. In figure 3.1 the different parts of the converter and their connections are drawn.

3.1 Parsing a DES-file

Parsing a design file means reading the file and saving the required information into the data structure of the converter. For the parsing of the IDaSS description file a lexical analyser (Lex) and a parser (Yacc) [LMB 92][PON 95][KRU 94] are used. Lex and Yacc together produce the source code to parse the design file.
Lex is a lexical analyser that recognizes all basic data items and switch characters used in an IDaSS description file. Every time lex recognizes an implemented string it returns a token. If a string doesn't match any rule, lex creates an error message. Yacc is reading one by one tokens from lex. If this token is permitted in the Yacc description (in a certain position Yacc is programmed to accept a range of tokens) the next token is requested. During the parsing of the description file, instructions can be executed. (Example: After finding a register in an IDaSS file, the name, width, reset value etc. are stored in a data structure). When an undefined token is offered to Yacc, the parser generates an error message.

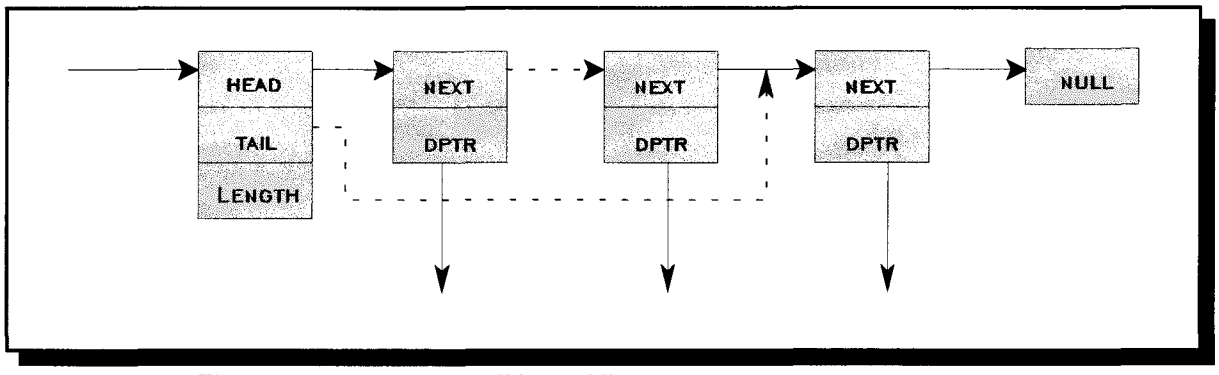

Figure 3.2 The used structure for all linked lists.

3.2 General linked-list structure

In the previous section the parsing of the design file is briefly described. During the parsing design information has to be stored. This information will be stored in a hierarchical data structure. In this structure much of elements have to be stored in linked lists. Instead of using a simple linked list, the converter uses the kind of lists described in figure 3.1.. In this kind of lists it is easy to find the length of the list or to find the last list element. The fields **dptr** point to the stored data. In the converter several routines are designed {**list.h**} to handle this kind of lists.

3.3 Important IDaSS object structures

In the converter a lot of objects are stored in specific data structures. In figure 3.3 the structures for a contents, a boundary and a parameter are described. A complete overview of used structures is given in the file **{types.h}.**

A short explanation of the contents of the **boundary** structure is given below:

- Name: Name from object in IDaSS
- Newname: Name from object in VHDL
- Contents: Contents description of the boundary
- Upper: Contents the boundary is part of
- Bparams: List of defined boundary parameters
- Contacts: List of used contacts on boundary

The **contents** structure:

- Ref_nr: Contents number
- Boundary: List of boundaries which use the contents definition
- Lower: List of boundaries that are part of the contents
- Contact: List of contacts on the contents

- Cparams: List of contents parameters
- Systems: List of systems that are part of contents
- Busses: List of busses that are part of contents
- Status: Field used by converter to control precalculating

The **Parameter** structure:
- Name: Name used in VHDL
- Value: Corresponding value of parameter when integer
- Anystring: Corresponding value of parameter when string

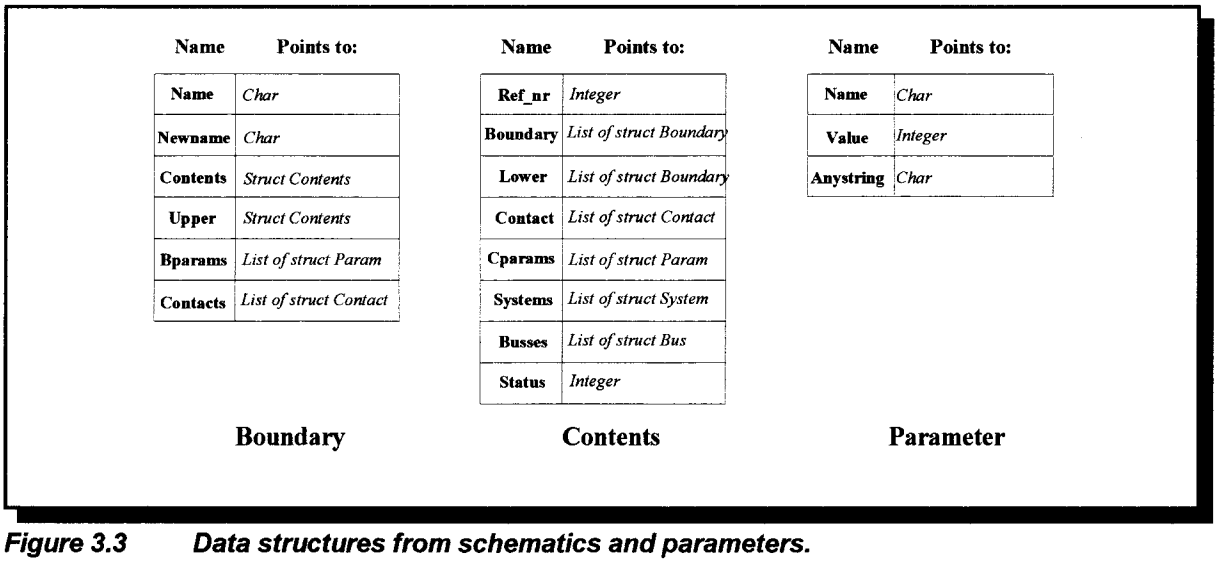

Figure 3.3 Data structures from schematics and parameters.

In the next section the hierarchy of a design is drawn.

3.4 Hierarchy structure

An IDaSS design is a collection of objects with certain connections. The starting point of the datastructure is the list of contents named **design**.
In this list the contents are stored in order of reference number. In figure 3.5 an example of a data tree from the design in figure 3.4 is drawn (the design from chapter 2). The schematics Sub1 and Sub2 are copies. They use the same contents definition. SubSub is defined just once in the tree (a lower schematic of contents 2) as boundary. In hardware it will be implemented twice.
The elements of the linked list are grey. The lines describe pointers. The data fields marked with a "*" contain a linked list. In this lists the contacts, systems, buses and parameters are

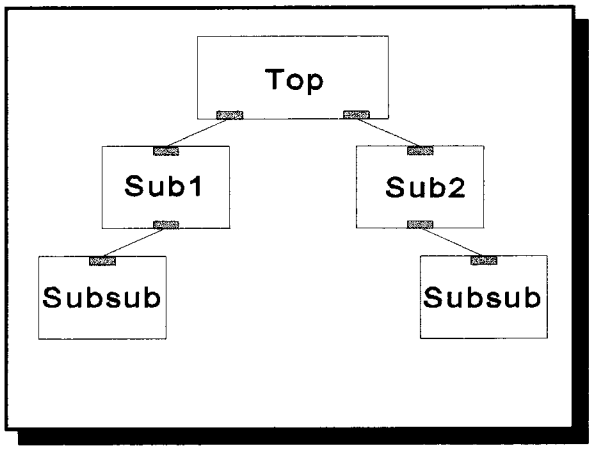

Figure 2.1 The hierarchy of the example design.

stored. In this design an example is given from a re-used design (contents 2).

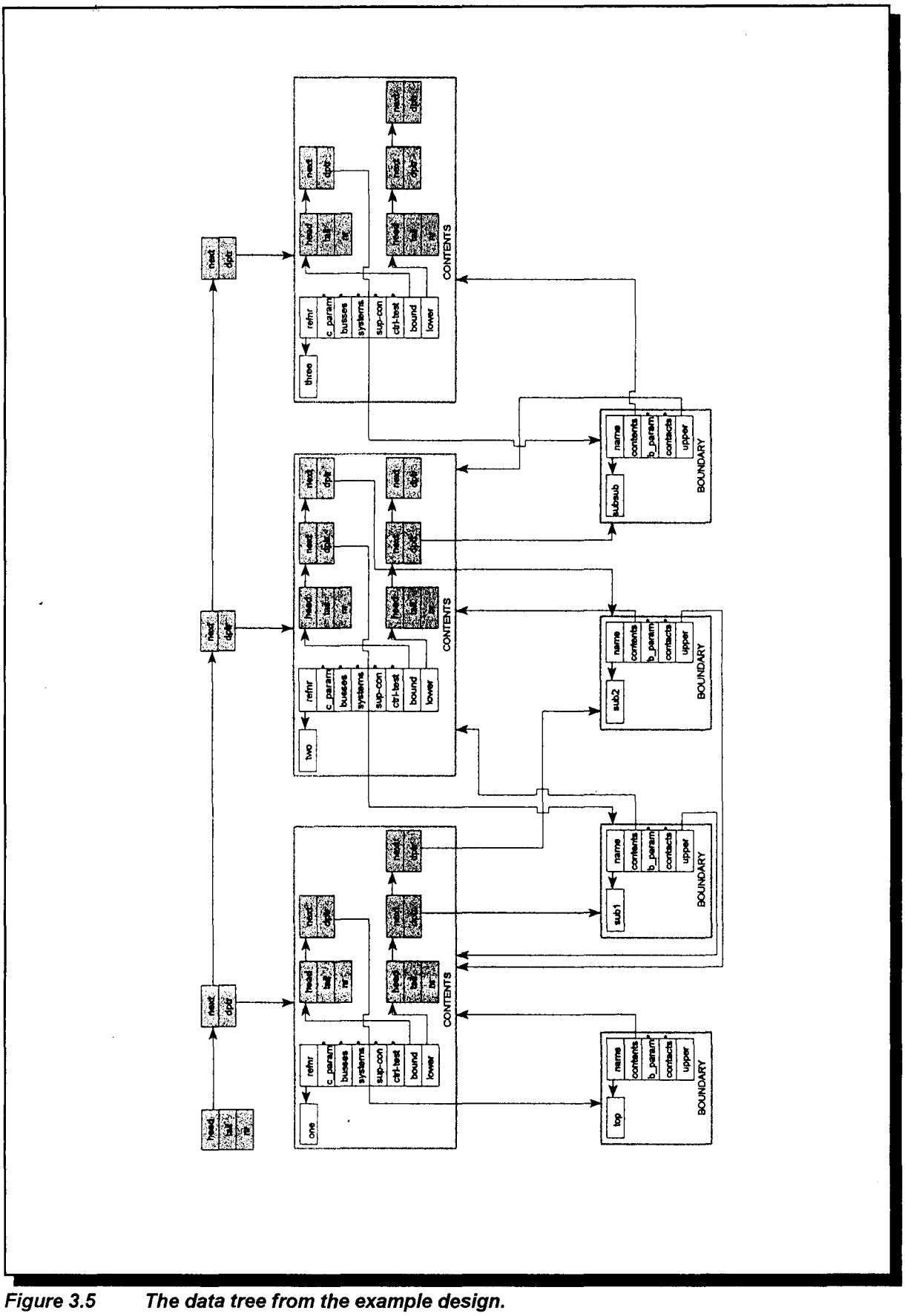

Figure 3.5 The data tree from the example design.

4 Completing the converter data tree

After parsing, the created data-structure doesn't contain all needed data to write the design in VHDL. A couple of operations have to complete the data structure before VHDL (figure 3.1). Another problem of the IDaSS design is the naming of blocks. The restrictions of VHDL concerning names differ from the restrictions in IDaSS. Before writing VHDL every name has to be checked and possibly corrected.
The above mentioned functions have to regard the complete design.
There are two ways used of walking through a design.

- The first method is called walking depth-first through the design. First the lowest schematics are visited followed by the corresponding higher schematic. In figure 4.1a this is shown.
- The second method sequences all contents. The list **design**, which contains all contents, can

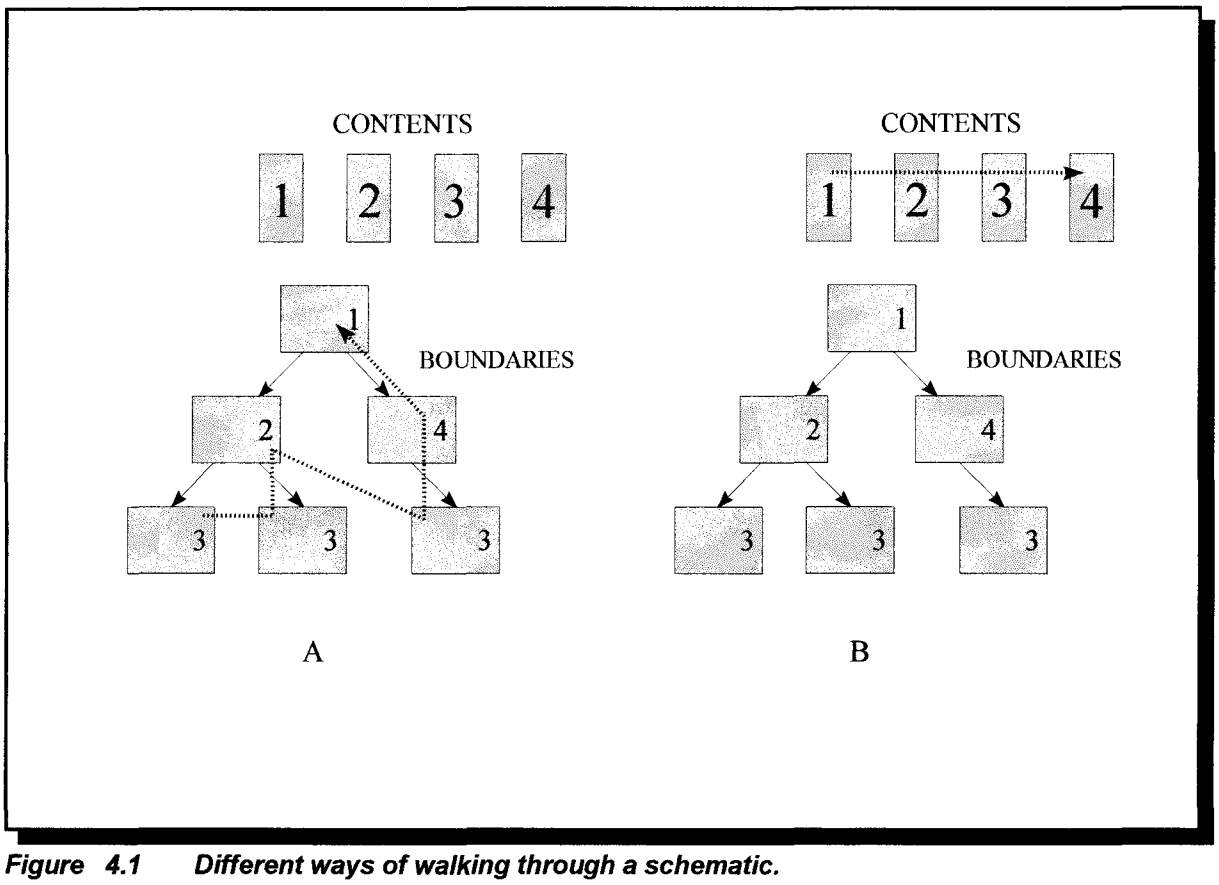

Figure 4.1 Different ways of walking through a schematic.

be used for visiting contents after contents. Not every function can be completed successful with this easy method. This method is shown in figure 4.1b.

In the following routines the two methods are both used. If possible, the second method is used for its simplicity.

When walking depth-first through the design, contents that are used several times will be visited several times. If not necessary a data field called "status" in the contents definition will be used. When the contents is visited this field will be checked. If not set, the function starts for the contents and the status is set. If set, the schematic will be passed without calculations.
The syntax of walking depth-first is given below:

```
Function Calculating(boundarylist)              *1
        foreach_boundary(boundarylist)
        if (LOWER_SCHEMATICS)
                Calculating(boundarylist) *2
        if (STATUS)==0)
                DO calculation
                STATUS = 1
```

*1 The list of boundaries that use contents '1' is given for input.
*2 The list of lower boundaries of the current contents is given.

4.1 Searching the width of busses

The width of a bus isn't described in the description file of a design. Most buses have to be declared with their width in the VHDL code. The width can be found by interrogating an input or output connected to the bus. The width of an in- or output from systems (all blocks except lower schematics) is available.
By searching depth first the width of the buses from lower schematics will have been calculated already. If there is only a schematic connected to the bus the corresponding bus into that lower schematic will be searched. The width from that bus is calculated already.
The status field will be used to avoid double calculation of the width for buses in a reused schematic.
The algorithm used to calculate the width is given below.

```
"Visiting all schematics depht-first with use of the status field.
foreach_bus(buslist)
        "Search in the connectionlist of the bus if there is a system (no schematic)
        "connected to the bus.
        if (no system found)
                "Search in the connectionlist of the bus if there is a lower level schematic
                "connected to the bus.
                if (no lower schematic)
                        "Bus only connected to bidirectional contact(s). We only give a
                        "warning that the width can not be calculated.
                else
                        "Search in the contactlist of that schematic for the matching contact
                        "the bus is connected with.
                        "Search in the buslist of the lower level schematic for the
                        "bus where that contact is part of and take the width.
        else
                "We found a real system, so the width of the bus can be found
                "in the input or outputlist of the system.
                if (input found)
```

```
                "We found an input that is connected to the bus.
                "The width of the bus is the same has the width of this input.
else
                "We found an output that is connected to the bus.
                "The width of the bus is the same has the width of this output.
```

4.2 Calculating the width and direction of schematic contacts

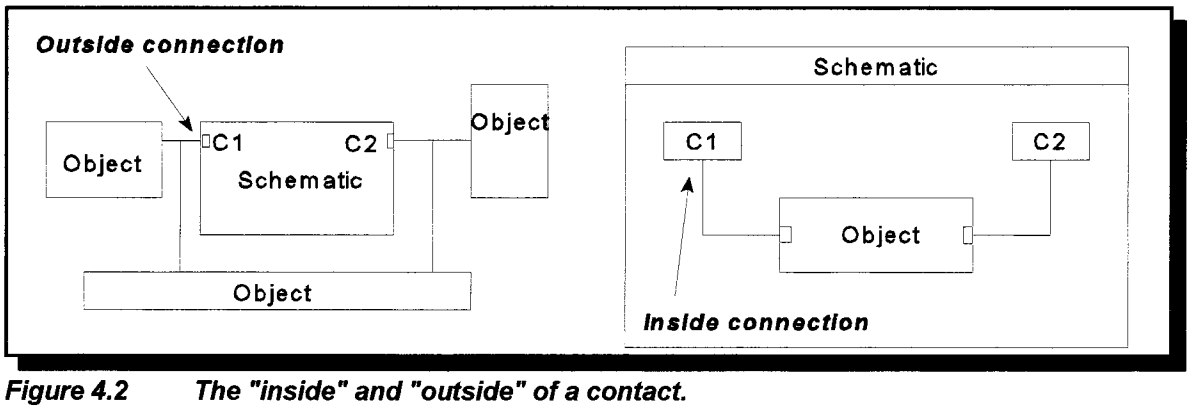

Figure 4.2 The "inside" and "outside" of a contact.

In the IDaSS design file, the width and direction of lower schematic contacts aren't defined. Because some converters don't handle bidirectional contacts correctly, the direction has to be calculated further if necessary.
The width of a schematic contact can be found on the connected buses (calculated in an earlier pass). The direction is a more difficult problem.

The direction of a contact can be calculated by checking the connections of the connected busses. The inside bus is connected to the contact at the schematic definition. The outside bus connects the schematic on the level in which it's located (figure 4.2).

In three steps the direction will be calculated:
1 Taking the inside direction.
2 If inside direction is bidirectional, take the outside direction.
3 If the outside direction differs from the inside direction, watch if the schematic is re-used. For re-used schematics, the outside directions are compared to define a final direction.

The main routine in which the three steps are called is Recalculate_bidirs.

```
Recalculate_bidirs(design)
        Recalculate_contacts(boundarylist)
        Post_proces_contacts(boundarylist)
        "If there are still unresolved contacts, then they can be found in the
        "linked-list pointed to by 'previous_saved'. We declare these contacts
        "bidirectional and inform the user.
        Calculate_final_direction(design);
```

The first step is called recalculate_contacts. This function tries to find the width and the direction of all contacts for every schematic in the design.

```
recalculate_contacts
    "Visit each schematic (contents) of the design (depth-first).
    foreach_boundary(boundarylist)
        foreach_contact
        "Search corresponding superconnector description in contents
        "Checking if this contact isn't already calculated for the first pass
        if (still unknown direction)
            Search_connected_bus
            Get_contacttype(bus)
            "Calculate the direction of the current contact. If the direction is
            "bidirectional the direction is only implemented in the superconnector
            "desciption.
            "Else the direction is also stored in the boundary description
            "The contact has the same width as the bus it is connected with. The width
            "is stored in the contact and superconnector description.
            "When the contact only connected with superconnectors internal, the
            "direction is still bidirectional at this moment. For further processing it's
            "defined bidirectional.
            if (superconnector direction == unknown)
                superconnector->type = c_bidir;
            "Contacts not bidirectional are saved in the superconnector field and
            "in the contact description. Their direction won't change anymore
```

In the above algorithm is shown that the routine handles two data-fields:

1 Superconnector field in contents description.
2 Contact field in boundary description.

The contact field will only be changed is there isn't another direction possible after "inside" calculation. Else, the direction will only be stored in the superconnector field.
The next step, only passed for contacts which aren't still definite is post_poces_contacts. This function tries to resolve the direction by looking "outside" the schematic.

```
post_proces_contacts
    "Visit each schematic (boundary) of the design (depth-first).
    foreach_boundary
        "Only do postprocessing if not at topboundary.
        if (NO previousboundary )
            "Search corresponding superconnector
            "For every contact examine if bidirectional or needs postprocessing.
            foreach_contact
                if ((contact == bidirectional) && (superconnector== bidirectional))
                    "We found a contact we have to post-proces. Search for the
                    "bus on the previous boundary! where the contact is part of.
                    get_contacttype(previousboundary)
                    "Recalculate the direction of the current contact.
                    if (direction == bidirectional)
                        "Minimal one input and one TS-output on the bus
                        contact = bidirectional
                        update_saved_bidirs(boundary->name);
```

```
else
        if (direction == output)
                "There's an output connected to the external
                "bus. The only situation in which this is possible
                "is a lower bus connected
                "with only superconnector(s).
                contact->direction = c_input;
                update_saved_bidirs(boundary->name);
if (higher boundary external contact part of bus)
        "The external contact is also derived.
        if (direction == c_output)
                tmpcontact->direction = c_output;
        else
                "We can not determine the direction of the
                "current contact unless we know the direction of
                "the higher boundary contact.
                "We save the current contact.
if (No extern_contact)
        if (direction = Three State)
                "Only TS-outputs on this bus
                "If this is the first time we run the second pass for
                "this contact (unknown) or if in earlier passes the
                "contact was also input it becomes input.
                "Else it becomes bidirectional.
        if (direction == c_input)
                "Only inputs on this bus
                "Contact becomes output if in an earlier pass
                "also output is found (or no earlier pass)
                "else the contact becomes bidirectional.
```

In the above routine only the contact field in the boundary definitions is edited. After post_proces_contacts the superconnector fields of schematic contacts contains the "inside" direction of the contacts. When bidirectional, the contacts fields exist the "outside" direction (more than one for re-used schematics).
The next step is comparing the outside directions from different schematics (boundaries) with the same contents, and determining a final direction that's stored in the superconnector field.

```
Calculate_final_direction
    foreach_contents
        foreach_superconnector
            direction = unknown
            "Searching the corresponding contact on the boundary
            foreach_boundary   "That uses current contents"
                if (contact is used in boundary)
                    if (direction==unknown)
                        direction = direction of corresponding contact
                        "first connected boundary
                    else
                        if (direction != direction of
                                corresponding_contact)
                            direction=c_bidir;
                            "contact already used with different
                            "direction
            if (direction==c_bidir)
```

```
                "Stop list_foreach_boundary, direction can't change anymore
        if (direction==unknown)
                currentcontact->direction=c_bidir
        else
                currentcontact->direction=direction
```

In the above functions there are two routines used:

- get_contacttype: This function searches for the real direction of contacts by looking to the connected bus with all his connections.

- update_saved_bidirs: This function is used to update contacts that have been saved by the post_proces_contacts function.

4.3 Adapting the IDaSS design names to VHDL

The limitations of names in IDaSS are not the same as the limitations in VHDL. In VHDL schematic and system names need to be unique in a design. In IDaSS they just need to be unique in a certain contents. A second difference is the use of underscores. Names in VHDL may not end with an underscore or contain a double underscore. The last restriction in VHDL is a list of reserved words.
To avoid that block names that are used in different contents have equal names, the converter is attaching the contents number to a name (Example: alu -> alu_3). This is done for: Schematics, systems, buses and superconnectors. Input, output, function and state names have to be unique for an object in VHDL. They don't get a postfix. To check and change names the following routine is designed.

```
rename_schematic_ifneeded
        "The double underscores are separated by an "x"and if ending on a underscore an "x" is
        "adapted to the name in the routine correct_name.
        Schematic->newname = correct_name(block->newname,"__",insert,postlen)
        "tmp becomes the name with contentsnumber
        tmp = strsav(block->newname)
        tmp = attach contentsnumber
        "The name of a boundary may not be equal to another boundary. This can only
        "be a boundary with the same contents (same postfix). The name of a boundary
        "without postfix may not be equal to behaviour or structural file names.
        if (!( CMP_BOUNDARY && CMP_FILE_NAMES))                                    *1
                "If the complete name isn't unique in the selected lists, the following steps are
                "taken:         1 - remove postfix
                "               2 - if name is (MAX-LENGHT - postfix) remove last 2 chars
                "               3 - attach "post,post" and the postfix again
                "               4 - check for unique =>if not increase aa->zz
        return(newname)
```

*1 In this line the name is checked for uniqueness.

This function is implemented for each type of name, with different restrictions. In the next overview the different names and the restrictions are written down.

Table 4.1, restrictions for block names.

Block	**restrictions**
Schematic	other lower boundaries and file_names
System	lower boundaries, other systems and file_names
Contact	other schematic contacts
Bus	other busses, connectors, boundaries, systems and file_names
Input	other inputs, keywords, clock name and reset name
Output	other outputs, inputs, keywords, clock name and reset name
Function	other functions, inputs and outputs
State	boundaries, systems, other states and file_names

The last category of names are parameter names. Because they become a signal and sometimes a contact they have to be unique with system, schematic, bus and contact names for the complete design. All these names are ending on a positive (except zero) value. By attaching a zero after each parameter the names becomes unique (example parameter__test ⇨ parameter_x_test_0). Also parameter names are checked for a correct syntax with the standard routine.

4.4 Adding clock, reset and parameter nets

The last correction of the data structure is the implementation of the "invisible" buses. Clock and reset buses aren't drawn in an IDaSS design and aren't included in the design file. Also the nets of the state machines aren't included in a design.
The clock and reset nets are implemented in one pass. In the first pass nets are created in every contents, and objects are connected. During the second pass the different levels are connected. Also in the second pass is the connection from lower schematics to the nets. Before adding reset inputs, there is a check if the reset value is used. The implementation of these nets is done in the file **{extra.c}**.
The control and test nets of the state machines are created with a couple of functions. For re-used schematics has to be controlled if there aren't unused inputs (in one boundary an object is controlled, in a copy it isn't). The implementation of state machine nets is done in the file {fsm_io.c}.

Another type of invisible buses, are the buses used for parameters.
When a parameter is found during the parsing of the design in an object, an input with the parameter-name (after correction) is inserted into the object.
The further processing of the input follows in **{extra.c}**. The main routine is called make_param_net that completes the design depth-first and uses the status field to avoid double

calculation.

```
make_param_net
    foreach_boundary
        foreach_system
            foreach_input
                if(input == parameter)
                    param_input_found
        foreach_lowerboundary
            foreach_contact
                if (contact == parameter)
                    " The contact only has to be connected
                    " to a net if the parameter isn't already
                    " defined in the lower schematic itself.
                    if(not defined in lower boundary)
                        param_input_found
```

The function param_input_found is connecting the contact or input to the net with the same name. If the net doesn't exist the function creates the net and a contact to a higher level.

In the algorithm the restriction "not defined in lower schematic" is used before connecting an parameter input on a lower schematic to a bus. If a parameter is defined on a schematic it is possible that the parameter is defined in the lower schematic. In this case we connect a new signal to the parameter input. In the next chapter this is explained in VHDL.

5 Writing VHDL

The goal of the converter is writing VHDL. After building the data tree all the information is available to write VHDL. Re-using schematics and parameters are important changes in the converter. The VHDL code is adapted to this feature.

5.1 Conversion of IDaSS hierarchy to VHDL

When writing a design in VHDL, re-used schematics have to be defined also in VHDL just once. The entity name of a contents definition can be defined by the user. This name will be followed be the contents number. Example: structural definition - str_1
Behaviour definition - beh_1
When the converter is writing different schematic definition different files, this names are also the file names. When a schematic is use more than once in one contents, it will have one component declaration. In paragraph 5.3 the structural VHDL files of the example design are printed with explanations.

5.2 Mapping parameters in VHDL

Parameters become inputs on an block. There are two possibilities for an input. An input can be an input from an object (register, operator, memory, etc).
In this case the input will be connected to a signal with the same name as the input.
The input also can be part of a lower schematic. Now we have two possibilities:

- ⇨ The input will be connected to a net with the same name as the input if the parameter is not defined in the lower schematics list of boundary parameters or in the lower schematics list of contents parameters.
- ⇨ A new net will be created and connected to the input if the parameter is defined into the lower schematic. The signal (net) gets the value the parameter has to be.

Extra created nets never get a contact to a higher level. They always get a value in the signal definition.
Normal parameter nets get a contact in two cases:

- ⇨ The parameter isn't defined in the current contents.
- ⇨ The parameter is defined in one of the surrounding boundaries of the contents.

If for a re-used schematic a parameter is defined once as a boundary parameter and also as contents parameter, the boundary parameter overrules in one boundary the contents parameter. This is the reason a contact is necessary. The different cases are shown in paragraph 5.3

5.3 VHDL code

The design of chapter 2 is converted to VHDL. The code is printed next. The design includes three contents. Also parameters are used. The behaviour code isn't printed.

```
-- --  >> Schematic Top renamed to Top_1 in VHDL.
Entity definition of first contents

ENTITY str_1 IS
 PORT (
  clock : IN std_ulogic;
  reset : IN std_ulogic;
 );
END str_1;

ARCHITECTURE structure OF str_1 IS
Architecture of first contents

 COMPONENT str_2          Declaration of schematics sub1 and sub2 with the same
  PORT (                  Contents
   con1_2 : OUT std_ulogic_vector(7 DOWNTO 0);
   con2_2 : IN std_ulogic_vector(7 DOWNTO 0);
   clock : IN std_ulogic;
   reset : IN std_ulogic;
  );
 END COMPONENT;


BEGIN

 Sub1_2 : str_2
  PORT MAP (
   OPEN,       Con 1 that isn't used on sub1_2 is not connected to a signal.
   OPEN,
   clock,
   reset,
  );

 Sub2_2 : str_2
  PORT MAP (
   OPEN,
   OPEN,
   clock,
   reset,
  );

END structure;


-- --  >> Schematic Sub1 renamed to Sub1_2 in VHDL.
-- --  >> Schematic Sub2 renamed to Sub2_2 in VHDL.

ENTITY str_2 IS
 PORT (
```

```
  con1_2 : OUT std_ulogic_vector(7 DOWNTO 0);
  con2_2 : IN std_ulogic_vector(7 DOWNTO 0);
  clock : IN std_ulogic;
  reset : IN std_ulogic
 );
END str_2;

ARCHITECTURE structure OF str_2 IS

 COMPONENT str_3
  PORT (
   con3_3 : IN std_ulogic_vector(7 DOWNTO 0);
   con4_3 : OUT std_ulogic_vector(7 DOWNTO 0);
   clock : IN std_ulogic;
   reset : IN std_ulogic;
   par2_0 : IN std_ulogic_vector(7 DOWNTO 0)        Contact on contents 3 for parameter.
  );
 END COMPONENT;

   The signal  for parameter par2_0 gets its value (defined in schematic subsub).
 SIGNAL Subsub_3par2_0local : std_ulogic_vector(7 DOWNTO 0) := "00000100";

BEGIN

 Subsub_3 : str_3
  PORT MAP (
   con2_2,
   con1_2,
   clock,
   reset,
   Subsub_3par2_0local          Parameter input par2_0 on schematic Subsub_3 is connected
  );                            to a specially created signal.

END structure;



-- --  >> Schematic Subsub renamed to Subsub_3 in VHDL.

ENTITY str_3 IS
 PORT (                                  Ports of contents 3
  con3_3 : IN std_ulogic_vector(7 DOWNTO 0);
  con4_3 : OUT std_ulogic_vector(7 DOWNTO 0);
  clock : IN std_ulogic;
  reset : IN std_ulogic;
  par2_0 : IN std_ulogic_vector(7 DOWNTO 0)        Parameter contact to a higher level
 );                                                The parameter is defined as boundary
END str_3;                                         parameter.

ARCHITECTURE structure OF str_3 IS

 COMPONENT register_3
  PORT (
   reset : IN std_ulogic;
   clock : IN std_ulogic;
```

```
    par1_0 : IN std_ulogic_vector(7 DOWNTO 0);      parameter input
    par2_0 : IN std_ulogic_vector(7 DOWNTO 0);      parameter input
    xi : IN std_ulogic_vector(7 DOWNTO 0);
    xo : OUT std_ulogic_vector(7 DOWNTO 0)
   );
 END COMPONENT;

    par1_0 is defined as contents parameter and not as boundary parameter.
    There isn't a contact to a higher level. The connected bus is declared here and gets
    the right value.
 SIGNAL par1_0 : std_ulogic_vector(7 DOWNTO 0) := "00000011";

BEGIN

 i_register_3 : register_3
   PORT MAP (
    reset,
    clock,
    par1_0,                          Net that parameter input is connected to
    par2_0,                          Net that parameter input is connected to
    con3_3,
    con4_3
   );

END structure;
```

6 CONCLUSIONS AND RECOMMENDATIONS

The new data structure created for IDaSS 0.09 designs satisfies for the implementation of IDaSS 0.09 designs. With the new data structure it's much easier to step through a design. The new function for name correction is much easier with guaranteed unique names. The only names in VHDL that have to be changed are the names of the signals, used to define the value of a contact from a lower level schematic (from a parameter). These names are correct and unique, but they become very long (schematic name + parameter name + "local"). The use of parameters in IDaSS is supported by the new converter. The only problem with parameters in the converter is the width of the signals for parameters. Only for parameters used in registers, the converter reduces the width of the signal. For other parameters, the width of the signal will be 64 bits. This can't be accepted, so in future a the width has to be calculated more precisely.

Literature

[KR 88] Kernighan, B.W. and D.M. Ritchie
C HANDBOEK
Schoonhoven: Prentice Hall, 1990.
Translated from the English: The C Programming Language.
London: Prentice Hall, 1988.

[KRU94] Kruijtzer, W.M.
GENERATING A VHDL FRAMEWORK FROM IDASS DESIGN FILES
Digital Information Systems Group, Faculty of Electrical Engineering, Eindhoven, University of Technology, 1991.
Report.

[KRU95] Kruijtzer, W.M.
A CONVERTER FROM IDASS TO SYNTHESIZABLE VHDL
Digital Information Systems Group, Faculty of Electrical Engineering, Eindhoven, University of Technology, 1991.
Graduation Report.

[LMB 92] Levine, J.R. and T. Mason, D. Brown
LEX AND YACC. 2nd ed.
Sebastopol: O'Reilly & Associates, Inc., 1992.

[STD 87] IEEE STANDARD VHDL LANGUAGE REFERENCE MANUAL
New York: IEEE, 1988.
IEEE Std. 1076-1987.

[PON 95] Pont, J.F
A CONVERTER FROM IDASS DESIGN FILES TO SYNTHESIZABLE VHDL
Digital Information Systems Group, Faculty of Electrical Engineering, Eindhoven, University of Technology, 1995.
Graduation Report.

[VER 90] Verschueren, A.C.
IDASS FOR ULSI (IDASS MANUAL)
Digital Information Systems Group, Faculty of Electrical Engineering, Eindhoven, University of Technology, 1990.

[VER 95] Verschueren, A.C.
IDASS V0.09 .DES FILE FORMAT
Digital Information Systems Group, Faculty of Electrical Engineering, Eindhoven, University of Technology, 1995.