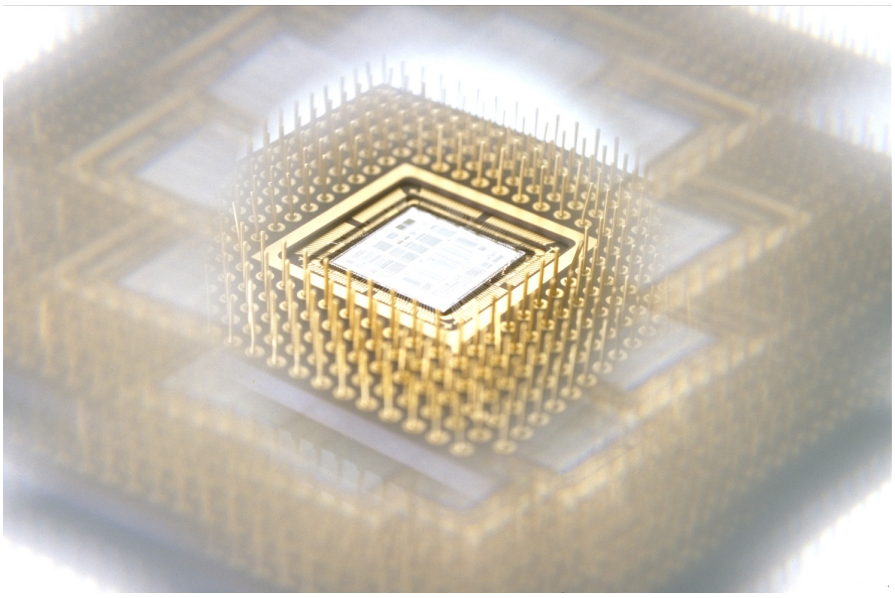
Avertec Tools

Yagle
Reference Guide



Software Release 3.4p5

June 7th, 2010

Avertec Copyright (c) 1998-2006 All Rights Reserved

About this Document

This document explains:

- The input formats supported
- How to perform the functional abstraction
- The VHDL generated
- User constraints
- Pattern recognition for analog blocks
- The Primary Data Structure

Documentation issued and compliant with Avertec Tools Release 3.4p5.

Please contact support@avertec.com for comments relating to this manual.

Table of Contents

1. Input Files 7
1.1. Netlist 7
1.1.1. SPICE 7
Expressions and Values 7
User-defined Functions 8
MOSFET 8
MOSFET Models 9
JFET 11
Junction Diode 12
Resistance 12
Capacitance 13
Subcircuit Instance 13
Independant Voltage Source 13
Supported Voltage Sources: Scenario 1 14
Supported Voltage Sources: Scenario 2 15
Supported Voltage Sources: Scenario 3 15
Supported Voltage Sources: Scenario 4 15
Supported Voltage Sources: Scenario 5 15
Supported Voltage Sources: Scenario 6 16
File Inclusion 16
Subcircuit 16
Parameters 17
Temperature 17
Scale Factor 17
Global Nodes 17
1.1.2. VERILOG 18
1.1.3. VHDL 18
1.2. Parasitics 18
1.2.1. DSPF Used for Connectivity 18
1.2.2. DSPF Used for Back-Annotation 18
1.2.3. SPEF 18
1.3. INF - Design Specific Configuration 18
1.3.1. Description 18
SDC input file 19
User-defined INF file 19
1.3.2. General 19
1.3.3. Disassembly Directives 19
IGNORE 19
CONSTRAINT 20
MUTEX 20

INPUTS 20
STOP 20
DIROUT 21
DLATCH 21
CKLATCH 22
PRECHARGE 22
NOTLATCH 22
MARKSIG 23
MARKTRANS 23
1.3.4. Behavioral Model Directives 24
SUPPRESS 24
SENSITIVE 24
1.3.5. Correspondencies INF / Tcl 25
2. Output Files 26
2.1. VHDL - Generated Behavior 26
2.1.1. Description 26
2.1.2. Latches and Registers 26
2.1.3. High impedance or Conflictual Nodes 27
2.1.4. Vectorization 28
2.1.5. Example 28
2.2. Verilog - Generated Behavior 29
2.3. CNS - Cone Netlist Structure 29
2.3.1. Reason for CNS 29
2.3.2. CNS in Circuit Disassembly 30
2.3.3. CNS Terminology 31
The Global CNS Figure 31
A Cone and its Elements 31
Grouping of Cones 32
The CNS Figure Hierarchy 32
2.4. CNS - Data Structures 33
2.4.1. The CNS Figure 33
2.4.2. The Link List 33
Link Structure Fields 33
Standard Link Types 34
2.4.3. The Branch List 35
Branch Structure Fields 35
Standard Branch Types 35
2.4.4. The Link List 36
Link Structure Fields 36
Standard Link Types 36
2.4.5. The Edge List 37
Edge Structure Fields 37
Standard Branch Types 38
2.4.6. The Transistor List 38
2.4.7. The Connector List 39
2.4.8. The Cell List 39

Cell Structure Fields 39
Standard Cell Types 40
3. Log Files 41
3.1. REP - Report File 41
3.1.1. Warning Messages 41
3.1.2. Error Messages 42
3.1.3. Fatal Errors 43
3.2. User-defined Log File 43
4. Configuration Variables 45
4.1. License Server 45
4.2. Environment 45
4.3. Names 45
4.4. Technology 47
4.5. Input Netlist and Parasitics 48
4.6. SPICE Parser 49
4.7. SPICE Driver 54
4.8. VHDL Parser/Driver 56
4.9. VERILOG Parser/Driver 57
4.10. DSPF/SPEF Parser 57
4.11. General Configuration 59
4.12. Disassembly 61
4.12.1. Functional Analysis 61
4.12.2. Transistor Orientation 62
4.12.3. Latch Recognition 63
4.12.4. Pattern Matching 67
4.12.5. Behavioral Model Generation 67
4.12.6. Cone Output Files 70
4.13. Output Configuration 71
4.14. Pattern Matching 72
4.15. Hierarchical Pattern Matching 73
4.16. API Specific 74
4.17. GUI 75
5. Tcl Interface 76
5.1. General 76
5.1.1. Configuration 76
avt_Config 76
avt_GetConfig 76
5.1.2. File Loading 76
avt_SetBlackBoxes 76
avt_LoadBehavior 77
avt_DriveBehavior 77
avt_LoadFile 77
avt_EncryptSpice 78
avt_SetCatalog 78
avt_GetCatalog 78
avt_CheckTechno 78

5.1.3. Netlist Modification 79
avt_GetNetlist 79
avt_FlattenNetlist 79
avt_DriveNetlist 79
avt_DisplayNetlistHierarchy 80
avt_DisplayResistivePath 80
avt_RemoveResistances 80
avt_RemoveCapacitances 81
5.1.4. Statistics 81
avt_StartWatch 81
avt_StopWatch 81
avt_PrintWatch 81
avt_GetMemoryUsage 82
avt_RegexIsMatching 82
5.2. Design Specific Configuration 82
5.2.1. General 82
inf_SetFigureName 82
inf_AddFile 82
inf_Drive 83
inf_ExportSections 83
inf_CleanFigure 83
5.2.2. Netlist 84
inf_DefineIgnore 84
5.2.3. Disassembly 84
inf_DefineMutex 84
inf_DefineInputs 84
inf_DefineDirout 85
inf_DefineDLatch 85
inf_DefineNotDLatch 85
inf_DefineNotLatch 85
inf_DefineKeepTristateBehaviour 85
inf_DefinePrecharge 86
inf_DefineNotPrecharge 86
inf_DefineModelLatchLoop 86
inf_DefineMemsym 86
inf_DefineRS 87
inf_MarkSignal 87
inf_MarkTransistor 87
inf_DefineSensitive 88
inf_DefineSuppress 88
5.3. Disassembling 88
5.3.1. yagle 88
6. Error Codes 89
6.1. API 89
6.2. AVT 92
6.3. BEF 96

6.4. BEG 96
6.5. BEH 97
6.6. BHL 99
6.7. BGL 99
6.8. BVL 101
6.9. CBH 103
6.10. CGV 103
6.11. CNS 103
6.12. GNS 104
6.13. INF 111
6.14. LOG 113
6.15. MBK 113
6.16. MCC 117
6.17. MGL 119
6.18. SLIB 120
6.19. SPF 120
6.20. STM 122
6.21. TAS 124
6.22. TRC 128
6.23. VAL 132
6.24. YAG 132
Index 134

Chapter 1. Input Files

1.1. Netlist

1.1.1. SPICE

The syntax of the SPICE subset supported by Yagle is given here in Backus-Naur Form. The meta-symbols of BNF are:

`::=`	meaning "is defined as"
`|`	meaning "or"
`<>`	angle brackets used to surround category names. The angle brackets distinguish syntax rules names (also called non-terminal symbols) from terminal symbols which are written exactly as they are to be represented.
`[]`	Enclose optional items
`{}`	Enclose repetitive items (zero or more times)

Expressions and Values

A value is refered to as `<val>`. A value can be associated with the following units, and is scaled accordingly:

`ff`	1e-15
`pf`	1e-12
`f`	1e-15
`p`	1e-12
`n`	1e-9
`u`	1e-6
`m`	1e-3
`k`	1e+3
`meg`	1e+6
`mi`	25.4e+6
`g`	1e+9

v	1
ns	1e-9
ps	1e-12
s	1

An expression is refered to as `<expr>`, and should appear enclosed in `''`, `()` or `{}`. Carriage returns are ignored within expressions and treated as white spaces, which means that an expression can be continued on subsequent lines without using the `+` sign.

The following mathematical functions supported within `<expr>` are `valif`, `max`, `dmax`, `min`, `dmin`, `trunc`, `int`, `sqrt`, `exp`, `log`, `sin`, `cos`, `tan`, `asin`, `acos`, `atan`, `atan2`, `sinh`, `cosh`, `tanh`, `log10`, `ceil`, `floor`, `fabs`, `abs`, `pow` and `pwr`

User-defined Functions

Yagle supports user-defined functions when specified through the `.FUNC` card. The example below illustrates the syntax supported:

```
.FUNC my_func(a,b) a*b+pow(2,a)
```

MOSFET

```
Mxx <ND> <NG> <NS> <NB> <MNAME> [L=<val>] [W=<val>]
+ [AD=<val>] [AS=<val>] [PD=<val>] [PS=<val>]
+ {[<param>=<val>|<expr>]} [$X=<val>] [$Y=<val>]
```

Parameters:

xx	MOS transistor name
<ND>	Drain node
<NG>	Gate node
<NS>	Source node
<NB>	Bulk node
<MNAME>	Model name, described in a .MODEL card

Optional parameters:

L=<val>	Channel length in meters (unless specified unit)
W=<val>	Channel width in meters (unless specified unit)
AD=<val>	Drain area in sq. meters (unless specified unit)
AS=<val>	Source area in sq. meters (unless specified unit)

`PD=<val>`	Drain Perimeter in meters (unless specified unit)
`PS=<val>`	Source Perimeter in meters (unless specified unit)
`$X=<val>`	X coordinate
`$Y=<val>`	Y coordinate
`<param>=<val>|<expr>`	Instantiation specific parameters, for example `nrs`, `nrd`, `mulu0`, `delvt0`, `sa`, `sb`, `sd`, `nf`, `nfing`, `m`

If `AD`, `AS`, `PD` or `PS` are not specified, they are calculated with the `GEOMOD` parameter (only BSIM4, otherwise value is 0).

MOSFET Models

```
.MODEL <MNAME> <nmostype>|<pmostype>
+ [(] [<param>=<val>|<expr>] [)]

<nmostype> ::= NMOS|NMOSBSIM3|NMOSBS32|NMOSBS4|
+ NMOSBS41|NMOSBS42|NMOSBS43|NMOSBS44|NMOSBS45|NMOSBS46

<pmostype> ::= PMOS|PMOSBSIM3|PMOSBS32|
+ PMOSBS4|PMOSBS41|PMOSBS42|PMOSBS43|PMOSBS44|PMOSBS45|PMOSBS46
```

MOS types

`NMOS,PMOS`	N-Channel, P-Channel MOSFET model
`NMOSBSIM3,PMOSBSIM3`	N-Channel, P-Channel BSIM3v3.0 Berkeley MOSFET model
`NMOSBS32,PMOSBS32`	N-Channel, P-Channel BSIM3v3.2.4 Berkeley MOSFET model
`NMOSBS4,PMOSBS4`	N-Channel, P-Channel BSIM4.0 Berkeley MOSFET model
`NMOSBS41,PMOSBS41`	N-Channel, P-Channel BSIM4.1 Berkeley MOSFET model
`NMOSBS42,PMOSBS42`	N-Channel, P-Channel BSIM4.2 Berkeley MOSFET model
`NMOSBS43,PMOSBS43`	N-Channel, P-Channel BSIM4.3 Berkeley MOSFET model
`NMOSBS44,PMOSBS44`	N-Channel, P-Channel BSIM4.4 Berkeley MOSFET model
`NMOSBS45,PMOSBS45`	N-Channel, P-Channel BSIM4.5 Berkeley MOSFET model
`NMOSBS46,PMOSBS46`	N-Channel, P-Channel BSIM4.6 Berkeley MOSFET model

Supported BSIM and PSP levels:

`LEVEL=8`	NGSPICE Berkeley BSIM3v3 model, up to BSIM3v3.2.4 (`VERSION=32`)
`LEVEL=49`	HSPICE BSIM3v3 model, up to BSIM3v3.2.4 (`VERSION=32`)
`LEVEL=53`	ELDO BSIM3v3 model, up to BSIM3v3.2.4 (`VERSION=32`)
`LEVEL=14`	NGSPICE Berkeley BSIM4 model, up to BSIM4.3 (`VERSION=43`)
`LEVEL=54`	HSPICE BSIM4 model, up to BSIM4.3 (`VERSION=43`)
`LEVEL=60`	ELDO BSIM4 model, up to BSIM4.3 (`VERSION=43`)

```
TOOL hspice
BSIM3V3  param level   49
BSIM3V3  param level   53
BSIM4    param level   54
PSP      param level 1020
PSPB     param level 1021

TOOL eldo
BSIM3V3  param level   49
BSIM3V3  param level   53
BSIM4    param level   60
PSP      param level 1020
PSPB     param level 1021

TOOL ngspice
BSIM3V3  param level    8
BSIM4    param level   14


TOOL titan
BSIM3V3  model BSM3 setdefault version 3.0
BSIM3V3  model BS32 setdefault version 3.24
BSIM4    model BS4  setdefault version 4.2
BSIM4    model BS41 setdefault version 4.1
BSIM4    model BS42 setdefault version 4.21
```

Different industry-standard electrical simulators have different interpretations of the parameters of .MODEL statement, which also deviate from the Berkeley model (see Berkeley's BSIM3v3.2.4 or BSIM4.3.0 MOSFET Model User's Manual). This can lead to significant differences in the results given by different simulators.

With no `simToolModel` variable set, Yagle uses the HSPICE model. Otherwise, Yagle interprets the parameters in the .MODEL statement with regard to the value of the `simToolModel` variable, and uses the model of the corresponding simulator.

The HSPICE BSIM3v3 model (`LEVEL=49`, `simToolModel = HSPICE`) used by Yagle deviates from the Berkeley BSIM3v3 model with regard to the following parameters (only if parameter `ACM=0-3`):

`CJSWG`	ignored, `CJGATE used instead`

MJSWG	ignored, MJSW used instead
PBSW	ignored, PHP used instead
PBSWG	ignored, PHP used instead
NF	the W of the is divided by NF to choose the appropriate model in the techno file

The ELDO BSIM4 (LEVEL=60, simToolModel = ELDO) model used by Yagle deviates from the Berkeley BSIM4 model with regard to the initialization of the binning parameters of LPEB (lateral non uniform doping on K1):

LLEPB=0	instead of LLEPB=LLPE0
WLEPB=0	instead of WLEPB=WLPE0
PLEPB=0	instead of PLEPB=PLPE0
NF	the W of the is not divided by NF to choose the appropriate model in the techno file

The TITAN BSIM models used by Yagle are fully compliant with Berkeley BSIM models. The only special behavior relates to NF

NF	the W of the is not divided by NF to choose the appropriate model in the techno file

JFET

```
Jxx <ND> <NG> <NS> <MNAME> {[<param>=<val>|<expr>]}
+ [$X=<val>] [$Y=<val>]
```

Parameters:

xx	JFET transistor name
<ND>	Drain node
<NG>	Gate node
<NS>	Source node
<MNAME>	Model name, described in a .MODEL card

Optional parameters:

$X=<val>	X coordinate
$Y=<val>	Y coordinate

`<param>=<val>|<expr>`	Instantiation specific parameters

Warning: JFETs are parsed but are not suuported as transistors. They can only be interpreted as resistances. See `avtSpiJFETisResistance`.

Junction Diode

```
Dxx NP NN MNAME [AREA=<val>] [PJ|PERI=<val>]
+ {[<param>=<val>|<expr>]} [$X=<val>] [$Y=<val>]
```

Parameters:

`xx`	Diode name
`<NP>`	Positive node
`<NN>`	Negative node
`<MNAME>`	Model name, described in a .MODEL card

Optional parameters:

`$X=<val>`	X coordinate
`$Y=<val>`	Y coordinate
`<param>=<val>|<expr>`	Instantiation specific parameters

Resistance

```
Rxx N1 N2 [R=]<val>|<expr> [TC1=<val>|<expr>]
+ [TC2=<val>|<expr>] {[<param>=<val>|<expr>]}
```

Parameters:

`xx`	Resistance name
`<N1>, <N2>`	Resistance nodes
`[R=]<val>|<expr>`	Value of resistance in Ohm

Optional parameters:

`TC1=<val>|<expr>`	Parsed but not supported
`TC2=<val>|<expr>`	Parsed but not supported
`<param>=<val>|<expr>`	Parsed but not supported

Capacitance

```
Cxx <N1> <N2> [C|VALUE=]<val>|<expr> [POLY=<val>|<expr>]
+ {[<param>=<val>|<expr>]}
```

Parameters:

`xx`	Capacitance name
`<N1>, <N2>`	Capacitance nodes
`[C|VALUE=]<val>|<expr>`	Value of capacitance in Farads

Optional parameters:

`POLY=<val>|<expr>`	Parsed but not supported
`<param>=<val>|<expr>`	Parsed but not supported

Subcircuit Instance

```
Xxx {<NN>} <MNAME> {[<param>=<val>|<expr>]}
+ [$X=<val>] [$Y=<val>] [$T=<Tx> <Ty> <R> <A>]
```

Parameters:

`xx`	Instance name
`{<NN>}`	list of nodes. Number must be the same as the subcircuit being instantiated
`<MNAME>`	Subcircuit being instantiated

Optional parameters:

`$X=<val>`	X coordinate
`$Y=<val>`	Y coordinate
`$T=<Tx> <Ty> <R> <A>`	Transform of the placement (X translation, Y translation, reflexion and rotation). Parsed but not supported
`<param>=<val>|<expr>`	Instantiation specific parameters, updating subcircuit parameters

Independant Voltage Source

```
Vxx <NP> <NN> DC [=]
+ <expr>|<pwl_function>|<pulse_function>

<pwl_function> ::= PWL (<TN> <VN> {<TN> <VN>} [TD] [SHIFT=<val>])

<pulse_function> ::= PULSE (<V0> <V1> <TD> <TR> <TF> <PW> <PER>)
```

Parameters:

`xx`	Voltage source name
`<NP>`	Positive node. The node may be hierarchical, up to one level of hierarchy
`<NN>`	Negative node. The node may be hierarchical, up to one level of hierarchy

Piece Wise Linear function parameters:

`<TN>`	Time Ti in seconds (unless specified unit)
`<VN>`	Value Vi of the source in volts at time Ti
`<TD>`	Negative node
`SHIFT=<val>`	Value added to all time values specified in the PWL card

Pulse function parameters:

`<VO>`	Initial value in volts of DC voltage
`<V1>`	Pulse magnitude in volts
`<TD>`	Delay time in seconds (unless specified unit)
`<TR>`	Rise time in seconds (unless specified unit)
`<PW>`	Pulse width in seconds (unless specified unit)
`<PER>`	Pulse period in seconds (unless specified unit)

The `PWL` and `PULSE` functions can be used to define clocks as an alternative to the INF or SDC constraint files. However, care should be taken to ensure that enough of the waveform is specified for the parser to be able to deduce the rise/fall clock instants and the period.

The `DC` function can be used to specify power supply values. If the specified negative node is the node `0`, or a node for which a supply value has been associated, then the supply value given by the sum of the DC value and the negative node supply value is associated to the positive node. Fairly complex multi-voltage configurations are possible, since multiple Vcards are possible and they can be resolved in any order.

The `DC` function, especially in combination with the `.GLOBAL` directive is a powerful mechanism for specifying which nodes are power supplies. Supplies can be completely determined using these cards without using any configuration variables. A node for which the supply value is superior to `avtVddVssThreshold` is considered to be a VDD node, else the node is considered to be a VSS node.

Supported Voltage Sources: Scenario 1

```
.GLOBAL inh_VDD inh_GND
```

```
Vsup inh_VDD inh_GND 1.2V
Vgnd inh_GND 0 0V

.SUBCKT inv A B inh_VDD inh_GND
MP0 B A inh_VDD inh_VDD PCH
MN0 B A inh_GND inh_GND NCH
.ENDS
```

Supported Voltage Sources: Scenario 2

```
Vgnd GND 0 0V
Vsup VDD GND 1.2V

.SUBCKT INV A B inh_VDD inh_GND
MP0 B A inh_VDD inh_VDD PCH
MN0 B A inh_GND inh_GND NCH
.ENDS

Xinv0 A B VDD GND INV
```

Supported Voltage Sources: Scenario 3

```
Vgnd GND 0 0V

.SUBCKT INV A B inh_VDD inh_GND
Vsup inh_VDD inh_GND 1.2V
MP0 B A inh_VDD inh_VDD PCH
MN0 B A inh_GND inh_GND NCH
.ENDS

Xinv0 A B VDD GND INV
```

Supported Voltage Sources: Scenario 4

```
Vgnd GND 0 0V
Vsup12 VDD GND 1.2V

.SUBCKT INV A B inh_VDD inh_GND
Vsup14 inh_VDD inh_GND 1.4V
MP0 B A inh_VDD inh_VDD PCH
MN0 B A inh_GND inh_GND NCH
.ENDS

Xinv0 A B VDD GND INV
```

`Vsup14` is ignored

Supported Voltage Sources: Scenario 5

```
Vgnd GND 0 0V
Vsup12 VDD GND 1.2V
Vsup14 VDD GND 1.4V

.SUBCKT INV A B inh_VDD inh_GND
MP0 B A inh_VDD inh_VDD PCH
MN0 B A inh_GND inh_GND NCH
.ENDS

Xinv0 A B VDD GND INV
```

`Vsup12` is ignored

Supported Voltage Sources: Scenario 6

```
Vgnd Xinv0.inh_GND 0 0V
Vsup12 Xinv0.inh_VDD 0 1.2V

.SUBCKT INV A B
MP0 B A inh_VDD inh_VDD PCH
MN0 B A inh_GND inh_GND NCH
.ENDS

Xinv0 A B INV
```

The following syntax is not supported:

Vsup12 Xinv0.inh_VDD Xinv1.inh_GND 1.2V

File Inclusion

```
.LIB|.LIBRARY LNAME [<LIBTYPE>]
.INCLUDE <FILENAME>
```

Yagle has a limited support of relative paths: when the path is not absolute, the path is assumed to be relative to the working directory of Yagle (the directory where it has been invoked from). Contrary to other simulators, it is not assumed to be relative to the directory of the file which makes the inclusion. This limitation can be overwhelmed by the variable `avtLibraryDirs`

Subcircuit

```
.SUBCKT <NAME> <NN> {<NN>} [PARAM:]{[<param>=<val>|<expr>]}

{<component>}

.ENDS [<NAME>]

<component> ::= M|J|D|R|C|X|V|.SUBCKT|.LIB|.INCLUDE|.MODEL|.PARAM
```

Parameters:

`<NAME>`	Name of the subcircuit
`<NN>`	Node name. Nodes with the same name followed by period and number are considered to be on the same net, even if they are not connected in the subcircuit. For example, in `.SUBCKT nand2 in out out.1 out.2 vss vdd out.3` the nodes `out`, `out.1`, `out.2` and `out.3` are considered to be the same signal. See also `avtSpiMergeConnector` and `avtSpiConnectorSeparator`.
`<param>=<val>|<expr>`	Default parameters

Yagle supports the declaration of subcircuits within subcircuits. However, if subcircuit A is defined within a subcircuit, instantiations of subcircuit A must not occur before in the file. This is not true if subcircuit A is defined at top-level.

Parameters

```
.PARAM {<param>=<val>|<expr>}
```

Temperature

```
.TEMP <val>|<expr>
.OPTION TEMP <val>|<expr>
```

Scale Factor

```
.SCALE <val>|<expr>
```

Scales MOSFET parameters.

```
L=L*<val>

W=W*<val>

PD=PD*<val>

PS=PS*<val>

SA=SA*<val>

SB=SB*<val>

SD=SD*<val>

AD=AD*<val>*<val>

AS=AS*<val>*<val>
```

Global Nodes

```
.GLOBAL {node}
```

When using the TCL interface, one should take care that the validity of `.GLOBAL` statement is limited to the context of the `avt_LoadFile` function call. For example, let's suppose a `.GLOBAL` statement defined in `globals.spi`, and a netlist defined in `netlist.spi`:

```
avt_LoadFile globals.spi spice
avt_LoadFile netlist.spi spice
```

With such a script, the `.GLOBAL` statement will not be available in `netlist.spi`. If it is not the intended behavior, prefer `.INCLUDE globals.spi` in `netlist.spi`.

1.1.2. VERILOG

Yagle supports the structural subset of the Verilog Hardware Description Language. For more information see the IEEE P1364 standard

1.1.3. VHDL

Yagle supports the structural subset of the VHDL Hardware Description Language. For more information see the IEEE P1076 standard.

1.2. Parasitics

1.2.1. DSPF Used for Connectivity

If the DSPF is used for connectivity purpose (the SPICE netlist is not connected without the DSPF, connectivity is ensured by the R elements), use .INCLUDE parasitics.dspf inside .SUBCKT

1.2.2. DSPF Used for Back-Annotation

If there is no BUSBIT construct, only identifiers containing [] or < > are considered as vectors.

1.2.3. SPEF

Yagle supports the Standard Prasitic Exchange Format Language, used for parasitic back-annotation putpose. For more information see the IEEE 1481-1999 standard.

1.3. INF - Design Specific Configuration

1.3.1. Description

The `.inf` file is an ASCII file with different sections. Blank lines, lines starting with '#' or any character between '/*' and '*/' are considered as comments. Most of the sections, except the header, are defined with a section name and the information related to this section are enclosed with `Begin` and `End;`. The syntax for each section depends on the section itself. The sections are not depending on each other so they can be declared out of order and several times.

The syntax is case insensitive for the keywords. The signal names can be given without quotes but to enable the use of signal names with special characters or unfortunately matching a keyword, they are required.

It is also possible to use units. For timing values with no specified unit, the Pico-second will be used. Valid units are: ps, ns, ms. For the capacitance values with no unit is specified, the Femto-farad will be used. Valid units are: pf, ff.

The information file gives information for the abstraction, database construction and static timing analysis tools. The information file name does not need to be related to a subcircuit name. In fact, the first token that should be set in the information file is "name" followed by the subcircuit name the information are given for. If this information is not given, the information file parser will guess the name of the subcircuit from the file name.

For a subcircuit, several information files can be given. They will be loaded in a particular order and the information will be merged. Actually, there are 2 different information files that can be generated by our tools and loaded, by default, with the following precedence:

SDC input file

The SDC command will be translated in the appropriate information file sections.

User-defined INF file

The order of loading and the files to load can be set by modifying the variable `avtReadInformationFile`. The files are separated by commas and the special character '$' will be replaced by the subcircuit name. Regular expressions can be used there to handle more complicated naming. Its default value is `$.spice.inf,$.sdc.inf,$.inf`. The '\' before the '.' simply indicates to the regular expression matching algorithm not to interpret '.' as any characters but as the character '.'. The names are from the less priority to the more priority. If the value is set to `no`, no information file will be loaded.

1.3.2. General

It is necessary to specify the design name. This is done by using the section "name". Optionally, a version for the information file can be given.

```
NAME mysubckt;
VERSION 1.2;
```

1.3.3. Disassembly Directives

IGNORE

If some components must be removed from the original netlist for any reason, it is possible to specify the component type and names in the IGNORE section. Th ere are 4 component types: Instances, Transistors, Resistances, Capacitances. Th e component names can be given using regular expressions.

```
IGNORE
BEGIN
  Instances: *fake, top.instancetoremove;
  Capacitances: toolowcapa*;
  ...
END;
```

At the moment, it is possible to remove resistances and capacitances only in the top level figure. To overcome this limitation, an information file can be wr itten for the sub-circuits.

CONSTRAINT

To perform analysis in a specific case, the user can apply static logic levels on the input connectors with the CONSTRAINT section. It contains internal or external signals constrained by "one" or "zero". The static logic levels are propagated through the netlist before databse construction.

```
CONSTRAINT
BEGIN
    sig1: 0;
    sig2: 1;
    sig3: 0;
    ...
END;
```

MUTEX

If some of the input connectors are mutually exclusive, this should be indicated in the MUTEX section using:

`muxUP{term1, term2, ...}`	to express that one port at most in the list is "one".
`muxDN{term1, term2, ...}`	to express that one port at most in the list is "zero".
`cmpUP{term1, term2, ...}`	to express that one and only one port in the list is "one".
`cmpDN{term1, term2, ...}`	to express that one and only one port in the list is "zero".

```
MUTEX
BEGIN
    muxUP{a,...,d};
    muxDN{m,...,p};
    cmpUP{i,...,l};
    cmpDN{x,...,z};
    ...
END;
```

INPUTS

User can specify connectors which should be considered as inputs. The disassembling process uses this information to inhibit the construction of cone branches from these connectors. This directive is essentially useful when dealing with RAMs, when the user wants to inhibit the construction of the reading bus.

```
INPUTS
BEGIN
    connector0;
    connector1;
    ...
END;
```

STOP

User can specify a list of signals which should be considered as stop points for the functional analysis phase of the disassembly. This means that any logic preceding the stop points will not be used in order to analyze the behavior of any gate following the stop point.

```
STOP
BEGIN
    sig1;
    sig2;
    sigs*;
    ...
END;
```

It is possible to use the wildcard '*' as for signal renaming.

DIROUT

In order to orient transistors, user can specify a list of signals, each one identifying the source or drain of a transistor. Transistors are then oriented towards these signals. Orientation of successive transistors is done by associating a level with each signal identifying a source or drain. Transistors are then oriented from higher level signals to lower level signals.

Transistor orientation is useful to avoid false branches construction, especially when dealing with pass-transistors.

```
DIROUT
BEGIN
    sig2: "1";
    sig3: "2";
    ...
END;
```

If no DIROUT level is specified, default level is `-1`.

It is possible to use the wildcard '*' as for signal renaming.

The DIROUT directive is equivalent to the NETOUTPUT directive in FCL. If a signal name is preceded by the '~' character then this signal will not be treated as an output, this is to deal with the case of signals whose names end in "_s" and must not be considered as output terminals.

DLATCH

Some internal tri-state nodes have to be considered to be dynamic latches for functional modeling and timing analysis purposes.

If only a few number of that tri-state nodes have to be taken into account, specify the list of corresponding signals into a DLATCH section. On the contrary, if the variable `yagleMarkTristateMemory` is set to `yes` in the configuration file and a few of tri-state nodes must not be taken into account, specify the list of corresponding signals into a DLATCH section, preceding each signal by a '~' character.

```
DLATCH
BEGIN
    sig1;
    ~ sig2;
    ...
END;
```

It is possible to use the wildcard '*' as for signal renaming.

CKLATCH

By default the latch recognition phase performed during the circuit disassembly does not require that external clocks be specified. Latches are identified either by structure (yagleStandardLatchDetection) or by Boolean analysis of combinatorial loop stability (yagleAutomaticLatchDetection).

However, it is sometimes necessary to constrain the latch recognition to identify only those latches for which the local clocks lie on a genuine clock path. To do this, specify the list of external connectors (or internal signals) in a CKLATCH section.

It is also possible to specify that an external connector (or internal signal) is definitely not a clock by preceding the name by a '~' character. In this case, any latch input at the end of a timing path originating from this connector (or signal) is considered to be a data input.

```
CKLATCH
BEGIN
    connector1;
    sig1;
    ~ sig2;
    ...
END;
```

It is possible to use the wildcard '*' as for signal renaming.

PRECHARGE

Signals whose names end in "_p" are considered to be precharged and therefore dealt with differently by the tool. If any other signals should be considered precharged these can be specified in the PRECHARGE section.

```
PRECHARGE
BEGIN
    sig1;
    ~ sig2;
    ...
END;
```

If a signal name is preceded by the '~' character then this signal will not be treated as a precharge, this is to deal with the case of non-precharge signals whose names end in "_p".

NOTLATCH

Disables the detection of latch on given signal.

```
NOTLATCH
BEGIN
    sig1;
    ...
END;
```

If a signal name is preceded by the '~' character then this signal will not be treated as a precharge, this is to deal with the case of non-precharge signals whose names end in "_p".

MARKSIG

Allows to set special markings on signals (nets). Especially useful for custom latch recognition. Available markings are:

LATCH	Signal corresponds to a latch memory-point.
FLIPFLOP	Signal corresponds to a flip-flop memory-point.
MASTER	Signal corresponds to the master memory-point of a flip-flop.
SLAVE	Signal corresponds to the slave memory-point of a flip-flop.
MEMSYM	Signal corresponds to one side of a symmetric memory.
RS	Signal corresponds to one side of an RS bistable.
VDD	Signal corresponds to an alimentation.
VSS	Signal corresponds to ground.
BLOCKER	No branch of a cone can go through the signal.
STOP	Cannot exploit logic beyond this point for functional analysis in the disassembler.
SENSITIVE	Marks the signal as a particularly sensitive signal. If a timed behavioral model of this signal is produced then the most precise (but cumbersome) model will be generated.

Example:

```
MARKSIG
BEGIN
    sig1: LATCH+MASTER
    sig2: STOP
    ...
END;
```

MARKTRANS

Allows to set special markings on transistors. Especially useful for custom latch recognition. Available markings are:

BLEEDER	Transistor corresponds to a bleeder.
FEEDBACK	Transistor corresponds to a feedback transistor of a memory-point.
COMMAND	Transistor corresponds to a command transistor of a memory-point, i.e driven by command signal.

NOT_FUNCTIONAL	Transistor should be ignored when calculating gate functionality.
BLOCKER	No branch of a cone can contain this transistor unless it is the first transistor of the branch.
UNUSED	No branch of a cone can contain this transistor.
SHORT	The transistor is considered short-circuited, the gate signal no longer contributes to the list of inputs.

Example:

```
MARKTARNS
BEGIN
    trans1: FEEDBACK+NOT_FUNCTIONAL
    trans2: COMMAND
    ...
END;
```

1.3.4. Behavioral Model Directives

SUPPRESS

The user can specify a list of signals which to be eliminated by Boolean simplification of the final behavioral description.

```
SUPPRESS
BEGIN
    sig1;
    sig2;
    sigs*;
    ...
END;
```

It is possible to use the wildcard '*' as for signal renaming.

SENSITIVE

This directive is useful in the case of functional abstraction with timing information. The user can specify a list of bussed signals for which the most precise timed behavioral model will be used. This is only required for certain critical signals.

```
SENSITIVE
BEGIN
    sig1;
    sig2;
    sigs*;
    ...
END;
```

It is possible to use the wildcard '*' as for signal renaming.

1.3.5. Correspondencies INF / Tcl

INF FILE	INF API	SDC
IGNORE	inf_DefineIgnore	
CONSTRAINT		set_case_analysis 0 | 1
MUTEX	inf_DefineMutex	
INPUTS	inf_DefineInputs	
STOP	inf_DefineStop	
DIROUT	inf_DefineDirout	
DIROUT ~	inf_DefineNotDirout	
DLATCH	inf_DefineDLatch	
DLATCH ~	inf_DefineNotDLatch	
CKLATCH	inf_DefineCkLatch	
CKLATCH ~	inf_DefineNotCkLatch	
PRECHARGE	inf_DefinePrecharge	
PRECHARGE ~	inf_DefineNotPrecharge	
NOTLATCH	inf_DefineNotLatch	
BREAK	inf_DefineBreak	
BYPASS		set_disable_timing / set_false_path -from | -through | -to
FALSEPATH		set_false_path -from -through -to
INTER	inf_DefineInter	
PINSLOPE / PINSLEW	inf_DefineSlope	set_input_transition
PINSLOPE / PINSLEW	inf_DefineConnectorSwing	
PATH DELAY MARGIN	inf_DefinePathDelayMargin	
NOFALLING		set_case_analysis fall / set_false_path -fall
NORISING		set_case_analysis rise / set_false_path -rise
CONNECTOR DIRECTIONS	inf_DefineConnectorDirections	
RC	inf_DefineRC	
NORC	inf_DefineNORC	
NOCHECK		set_false_path -setup | -hold
DONOTCROSS	inf_DefineDoNotCross	
SLOPEIN	inf_DefineSlopeRange	
CAPAOUT	inf_DefineCapacitanceRange	
CROSSTALK MUTEX	inf_DefineCrosstalkMutex	
SETUP	inf_SetSetupMargin	
HOLD	inf_SetHoldMargin	
CONDITIONED COMMAND STATES	inf_DefineConditionedCommandStates	
CLOCK CONNECTORS / ASYNCHRONOUS CLOCK GROUP	inf_DefineClock	create_clock / create_generated_clock / set_clock_latency
EQUIVALENT CLOCK GROUP	inf_DefineEquivalentClockGroup	
MULTIPLE CLOCK PRIORITY	inf_DefineClockPriority	
SPECIFY INPUT CONNECTORS		set_input_delay
VERIFY OUTPUT CONNECTORS		set_output_delay
MULTICYCLE PATH		set_multicycle_path

Chapter 2. Output Files

2.1. VHDL - Generated Behavior

2.1.1. Description

The VHDL which is automatically generated by Yagle is an RTL level zero delay data-flow description. It contains a list of concurrent assignment statements which represent the behavior of the simple gates which are neither high impedance nor conflictual. Distinct VHDL process blocks are used to describe those gates which represent identified memory elements as well as those gates for which Yagle was unable to verify the duality.

The declarations within the VHDL file conform to those of the IEEE STD_LOGIC package. This makes the VHDL readily exploitable by most commercial simulation, formal proof and synthesis tools.

The VHDL produced if FCL is used remains similar since a given behavioral description is parsed to generate an internal behavioral model, however this model can contain delay information, specified through the use of the AFTER attribute. However, if the full hierarchical pattern recognition is used, then any kind of behavioral description can be produced. Here we can only describe the VHDL which is automatically generated.

2.1.2. Latches and Registers

Latches and registers are described using separate VHDL processes, to allow easy latch inference by any subsequent parser. A cascaded "IF ... THEN ... ELSIF ... ... END IF" statement is used to represent separate driving conditions. For example a latch with two clocks would be represented by :

```
REG1: PROCESS (ck1,ck2)
BEGIN
  IF ck1 = '1' THEN
    latch <= data1;
  ELSIF ck2 = '1' THEN
    latch <= data2;
  END IF;
END PROCESS;
```

It should be noted that the lexical construction guarantees that the latch is not conflictual, and that the memory signal is incompletely assigned. This allows straightforward latch inference.

Registers are differentiated from latches through the use of the EVENT attribute on one (and only one) of the clock signals. For example, a positive-edge triggered flip-flop with an asynchronous reset would be represented by :

```
REG1: PROCESS (ck,reset)
BEGIN
  IF reset = '1' THEN
    reg <= '0';
```

```
    ELSIF (ck = '1' and ck'EVENT) THEN
      reg <= data;
    END IF;
  END PROCESS;
```

2.1.3. High impedance or Conflictual Nodes

Both high impedance and conflictual nodes are modeled by two process blocks. One process describes the condition for the assignment of logic '1', whilst the other describes the condition for the assignment of logic '0'. For example a simple tri-state gate would be described as follows :

```
BUS0: PROCESS (enable,data)
BEGIN
  IF (enable = '1' and data = '1') THEN
    out <= '1';
  ELSE
    out <= 'Z';
  END IF;
END PROCESS;

BUS1: PROCESS (enable, data)
BEGIN
  IF (enable = '1' and data = '0') THEN
    out <= '0';
  ELSE
    out <= 'Z';
  END IF;
END PROCESS;
```

The "IF ... THEN ... ELSE ... END IF" configuration is used to ensure that the tri-state signal is completely defined. This is necessary to avoid the tri-state node being mistaken for a latch.

2.1.4. Vectorization

No additional vectorization is deduced by Yagle. Connector and signal declarations can be optionally vectorized or devectorized. All assignment expressions are given on a bit by bit basis.

2.1.5. Example

The following example of a VHDL file generated by Yagle illustrates the typical structure :

```
-- VHDL data flow description generated from `addaccu`
--  date : Tue Jul 18 20:08:28 2000

library IEEE;
use IEEE.std_logic_1164.all;

-- Entity Declaration

ENTITY addaccu IS
  PORT (
  a : in std_logic_vector (3 DOWNTO 0);
  b : in std_logic_vector (3 DOWNTO 0);
  ck : in std_logic;
  s : inout std_logic_vector (3 DOWNTO 0);
  sel : in std_logic;
  vdd : in std_logic;
  vss : in std_logic
  );
END addaccu;

-- Architecture Declaration

ARCHITECTURE RTL OF addaccu IS
  SIGNAL accu : STD_LOGIC_VECTOR (3 DOWNTO 0);
  SIGNAL oper : STD_LOGIC_VECTOR (3 DOWNTO 0);
  SIGNAL carry : STD_LOGIC_VECTOR (2 DOWNTO 1);

BEGIN
  carry (1) <= ((oper (1) and ((oper (0) and (a (1) or a (0))) or a (1))) or
(oper (0) and a (1) and a (0)));
  carry (2) <= ((a (2) and (carry (1) or oper (2))) or (carry (1) and oper (2)));
  oper (0) <= ((b (0) and (accu (0) or not (sel))) or (accu (0) and sel));
  oper (1) <= ((b (1) and (accu (1) or not (sel))) or (accu (1) and sel));
  oper (2) <= ((b (2) and (accu (2) or not (sel))) or (accu (2) and sel));
  oper (3) <= ((b (3) and (accu (3) or not (sel))) or (accu (3) and sel));

REG0: PROCESS (ck)
BEGIN
  IF (ck = '0' and ck'EVENT) THEN
    accu (0) <= s (0);
  END IF;
END PROCESS;

REG1: PROCESS (ck)
BEGIN
  IF (ck = '0' and ck'EVENT) THEN
    accu (1) <= s (1);
  END IF;
END PROCESS;

REG2: PROCESS (ck)
BEGIN
  IF (ck = '0' and ck'EVENT) THEN
    accu (2) <= s (2);
  END IF;
END PROCESS;
```

```
REG3: PROCESS (ck)
BEGIN
  IF (ck = '0' and ck'EVENT) THEN
    accu (3) <= s (3);
  END IF;
END PROCESS;

  s (0) <= not ((a (0) xor oper (0)));

  s (1) <= (a (1) xor oper (1) xor (not (oper (0)) or not (a (0))));

  s (2) <= (a (2) xor oper (2) xor not (carry (1)));

  s (3) <= (a (3) xor oper (3) xor not (carry (2)));

END;
```

2.2. Verilog - Generated Behavior

2.3. CNS - Cone Netlist Structure

In this section, we provide details of the main underlying data structure manipulated by Yagle. The heart of the disassembly procedure is, in fact the transformation of a transistor netlist into the disassembled gates represented by this so-called CNS data structure. It is this data structure which is functionally characterized in order to automatically generate the VHDL from a transistor net-list.

These details are provided to help the reader understand the task performed by Yagle and to aid in the comprehension of the information provided in the CNS output file.

2.3.1. Reason for CNS

The CNS (acronym for Cone Net-list Structure) data structure is designed to represent extracted gate net-lists. It has evolved out of the need for a common data structure for CAD-VLSI verification tools such as: formal verification, timing and power analysis, and logico-temporal simulation.

These tools require efficient data structures which can directly support fast algorithms, since the volume of data to be treated is generally very high. Yet they also require enough structural and electrical details of the circuit for the verification results to accurately reflect the final circuit. An extracted transistor net-list effectively contains all the electrical characteristics required, however, the lack of orientation of the net-list renders unfeasible the verification of circuits of any reasonable size.

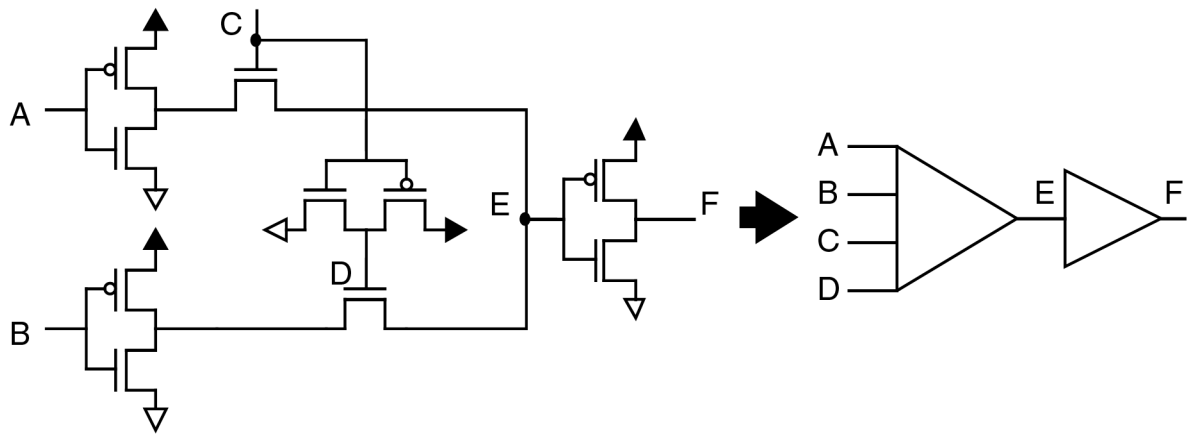
The CNS data structure attempts to combine the precision advantage of a transistor net-list with the speed advantage of a logical gate net-list. This is achieved by representing the circuit as a directed acyclic graph representing signal-flow within the circuit. Each node of this graph is a type of pseudo logical gate known as a cone. A graph representation of the circuit allows the direct implementation of rapid traversal algorithms useful in simulation and timing analysis.

The fundamental element of this data structure is the Cone. The Cone represents a means of cutting-up the transistor net-list such that the influences on every transistor gate are well-defined. In effect, a cone contains the set of current paths to the power supplies or external connectors for the gate of every transistor. Since transistor gates represent the cone to cone boundaries, and the gate current of MOS transistors is negligible, there is no current transfer between cones. This characteristic makes the cone conceptually ideal for the analysis of any kind of circuit behavior which depends on charge transfer, for example timing and power consumption.

2.3.2. CNS in Circuit Disassembly

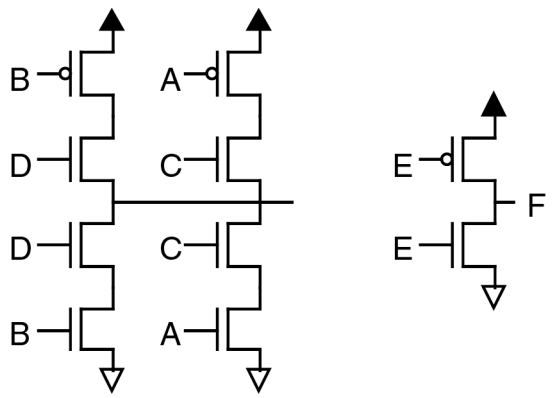
Since the most common means of obtaining a CNS representation of a circuit is through the disassembly of a transistor net-list, an understanding of the principles of disassembly is useful in the comprehension of the CNS data structure. This section, therefore, briefly explains the basics of the disassembly process.

The disassembly of a circuit is based on the principle of obtaining the equations which define the state of each transistor gate. In order to obtain these equations, the circuit representation is converted from a transistor net-list to a cone net-list (see figure 9.1), it is this representation which is stored in the CNS data structure.



A cone is defined as being, for each circuit node which is connected to at least one transistor gate, the set of branches which, from this node, attain a power supply or an external connector on the traversal of transistor source-drain junctions. Each branch consists of links which correspond to the transistors traversed. These branches therefore reveal the signals which govern the state of the transistor gate(s) for which the cone is being constructed.

A set of cones is therefore obtained (completely defining the state of all transistor gates and drivable external connectors), each of which contain a set of branches. For the example of figure 9.1, the two cones E and F are made up of the branches shown in figure 9.2.



This set of branches allows us to express the behavior of the cone and hence generate a Boolean expression for the state of the corresponding transistor gate. This expression is in fact composed of two parts: the function which represents the conditions necessary for Vdd to impose (Sup), and the equivalent for Vss (Sdn).

In reality these conditions have to verified globally, this means that Sup and Sdn are expressed in terms of the logic surrounding the cone. The depth, in terms of logic gates, used for the expansion is defined by the user.

2.3.3. CNS Terminology

The Global CNS Figure

The top-level data structure of CNS is called the CNS figure and is declared in 'C' as a cnsfig_list. This, in common with all the other CNS objects, is a linked-list structure, the first element of the structure being a pointer to the next element in the list. This is provided mainly for memory management purposes, since it allows block allocation of cnsfig_list structures.

The CNS figure is the global description of the entire disassembled circuit. In common with the MBK lofig_list logical net-list representation, it contains the list of external connectors, and the list of transistors. However, the internal structure of the circuit is represented by the list of cones as opposed to a list of instances.

In addition to the above, the CNS figure also contains a number of optional fields. The first of these is a list of cells, each cell corresponding to a grouping of cones. The next two fields are filled in by the circuit disassembler if requested, but are currently not supported by the parser of the CNS figure. These are: a pointer to a global behavioral figure for the circuit, and a logical net-list for the circuit. The logical net-list is hierarchical since each instance of the logical figure corresponds to a cone, and cones of similar physical structure are grouped into identical models for which the disassembler generates separate behavioral descriptions. The final field is the USER field, which allows the addition of user-defined information to the structure.

A Cone and its Elements

The fundamental object of CNS is the cone, this is the disassembled equivalent of a logical gate. It is made up (as described in §9.3.2) of branches, a branch corresponding to a path from the node on which the cone is built to an external port across transistor source-drain junctions.

Each cone contains up to four sets of branches but at least one. These sets correspond to the type of external port on which the branch ends. The four types are: VDD, VSS, EXT and GND, corresponding to branches terminating on Vdd or Vss power supplies, external connectors, or ground (for GaAs compatibility) respectively. Note that for external connector branches, the final link of the branch points to the corresponding connector.

The connectivity between cones is represented by edges. Each cone contains two lists of edges: one for the inputs, and one for the outputs. An edge contains a pointer to an object to which the cone is connected (cone or external connector) and a type indicating the type of object and certain characteristics of the connection.

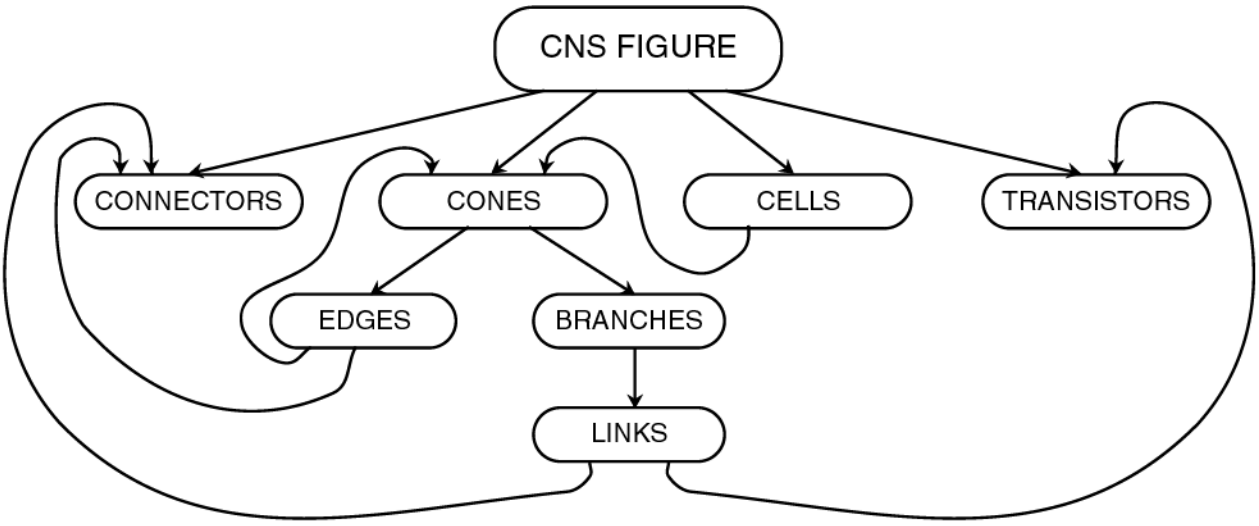
Grouping of Cones

CNS contains a mechanism for the grouping of cones, this is useful in the detection of complex gates by pattern recognition. This mechanism is accommodated by means of the list of cells in the CNS figure. Each cell contains the list of cones contained within the cell, a type indicating what the grouping correspond to, and an optional behavioral figure.

The CNS Figure Hierarchy

The CNS data structure is an inhomogeneous, hierarchical data structure, that is each level of the hierarchy contains specific types of objects, which are different to the types found on other levels. The complete hierarchy is shown in figure 9.3.

The most notable feature revealed in figure 3 is the looped nature of the hierarchy. It is this characteristic which gives CNS its flexibility in traversal, hence allowing implementation of efficient algorithms.



2.4. CNS - Data Structures

2.4.1. The CNS Figure

A detailed explanation of the various parts of the CNS figure is given in §9.4.1. The figure is defined as a cnsfig_list structure which is summarized in table 9.1.

Field Name	Description
NEXT	link to next CNS figure in list
NAME	name of the figure
LOCON	list of external connectors
LOTRS	list of transistors
CONE	list of cones
CELL	list of cells (cone groups)
LOFIG	hierarchical logical net-list
BEFIG	global behavioral figure
USER	user-defined

Table 9.1: Summary of the CNS figure

This, in common with all the other CNS objects is a linked-list data structure, for reasons of memory management. The last four fields can be NULL.

2.4.2. The Link List

Link Structure Fields

The link object, of type link_list, is made up of the following fields:

struct link *NEXT	Pointer to the following link in the branch's list of links.
long TYPE	The logical sum of masks indicating the type and nature of the link.
union ulink ULINK	Pointer to the object to which the link refers, LOTRS, LOCON or PTR.
float CAPA	The capacitance of the node at which the link is built.
struct ptype *USER	User defined information.

Note that the ULINK field is a union since it can point to a transistor (field LOTRS of type lotrs_list*) or a connector (field LOCON of type locon_list*). In addition the union contains a field for a generic pointer (PTR of type void*) to facilitate pointer comparison.

Standard Link Types

CNS_IN	An external connector link corresponding to an input only connector.
CNS_INOUT	An external connector link corresponding to a bidirectional connector.
CNS_2EQUIP	A generic type corresponding to devices with only two equipotentials, e.g. diode or resistance links.
CNS_3EQUIP	A generic type corresponding to devices with three equipotentials, e.g. MOS transistors.
CNS_SWITCH	A link corresponding to part of a CMOS transmission gate.
CNS_COMMAND	A link corresponding to a transistor, within a latch cone, whose gate is driven by the command signal of the latch.
CNS_ACTIVE	A generic type corresponding to any active device.
CNS_PASSIVE	A generic type corresponding to any passive device.
CNS_DOWN CNS_UP	Generic types indicating the orientation of non-symmetric devices links, e.g. diodes. UP corresponding to towards the power supply, and DOWN corresponding to away from the power supply.
CNS_SW	A generic type indicating any active switching device, usually a transistor.
CNS_PULL	A link corresponding to a passive pull up or pull down resistance.
CNS_DRIV_PULL	A link corresponding to an active pull-up or pull-down resistance.
CNS_DIODE_UP **CNS_DIODE_DOWN**	A link corresponding to a diode oriented according to the generic orientation masks.
CNS_RESIST	A link corresponding to a passive resistance, e.g. an MOS transistor whose gate is connected to a power supply.
CNS_CAPA	A link corresponding to a capacitance, e.g. an MOS transistor whose source and drain are connected to the same equipotential.

CNS_DIPOLE A link corresponding to a dipole.

A large number of the above masks are generic types, they are rarely used in the affectation of a type to a link since the are included in the non-generic types. For example the type CNS_RESIST includes the masks CNS_PASSIVE and CNS_2EQUIP.

The generic types are included to facilitate the testing of links, since they allow certain type groups of links to be tested for using a comparison with a single mask.

2.4.3. The Branch List

Branch Structure Fields

The branch object, of type branch_list, is made up of the following fields:

struct branch *NEXT Pointer to the following branch in the cone's list of branches.

long TYPE The logical sum of masks indicating the type and nature of the branch.

struct link *LINK The list of links which make up the branch.

struct ptype *USER User defined information.

Standard Branch Types

CNS_VSS CNS_VDD
A branch corresponding to a path from the cone output node to the Vdd(Vss) power supply.

CNS_VDD
A branch corresponding to a path from the cone output node to the Vdd(Vss) power supply.

CNS_GND
A branch corresponding to a path from the cone output node to ground (exists only in GaAs).

CNS_EXT
An external connector branch, i.e. a path from the cone output node to an external connector. Note that the final link of the branch is the external connector.

CNS_NOT_FUNCTIONAL
A branch which does not contribute to the functionality of the cone, for example: a pull-up resistance or a bleeder.

CNS_BLEEDER
A branch corresponding to one of the forms of figure 5a.

CNS_DEGRADED
A branch which degrades the output level of the cone, i.e. a Vdd branch containing an N-type transistor or a Vss branch containing a P-type transistor.

CNS_PARALLEL
A branch for which there exists one or more parallel branches within the cone. See §9.3.2 for the definition of parallel branches

CNS_PARALLEL_INS
For any given set of parallel branches, all but one are marked with the type PARALLEL_INS. This allows algorithms which traverse the list of branches to consider only one of the set of parallel branches by ignoring those of type PARALLEL_INS.

CNS_FEEDBACK
A branch which corresponds to part of the feedback loop in a latch cone.

2.4.4. The Link List

Link Structure Fields

The link object, of type link_list, is made up of the following fields:

struct link *NEXT	Pointer to the following link in the branch's list of links.
long TYPE	The logical sum of masks indicating the type and nature of the link.
union ulink ULINK	Pointer to the object to which the link refers, LOTRS, LOCON or PTR.
float CAPA	The capacitance of the node at which the link is built.
struct ptype *USER	User defined information.

Note that the ULINK field is a union since it can point to a transistor (field LOTRS of type lotrs_list*) or a connector (field LOCON of type locon_list*). In addition the union contains a field for a generic pointer (PTR of type void*) to facilitate pointer comparison.

Standard Link Types

CNS_IN	An external connector link corresponding to an input only connector.
CNS_INOUT	An external connector link corresponding to a bidirectional connector.
CNS_2EQUIP	A generic type corresponding to devices with only two equipotentials, e.g. diode or resistance links.

CNS_3EQUIP	A generic type corresponding to devices with three equipotentials, e.g. MOS transistors.
CNS_SWITCH	A link corresponding to part of a CMOS transmission gate.
CNS_COMMAND	A link corresponding to a transistor, within a latch cone, whose gate is driven by the command signal of the latch.
CNS_ACTIVE	A generic type corresponding to any active device.
CNS_PASSIVE	A generic type corresponding to any passive device.
CNS_DOWN CNS_UP	Generic types indicating the orientation of non-symmetric devices links, e.g. diodes. UP corresponding to towards the power supply, and DOWN corresponding to away from the power supply.
CNS_SW	A generic type indicating any actice switching device, usually a transistor.
CNS_PULL	A link corresponding to a passive pull up or pull down resistance.
CNS_DRIV_PULL	A link corresponding to an active pull-up or pull-down resistance.
CNS_DIODE_UP **CNS_DIODE_DOWN**	A link corresponding to a diode oriented according to the generic orientation masks.
CNS_RESIST	A link corresponding to a passive resistance, e.g. an MOS transistor whose gate is connected to a power supply.
CNS_CAPA	A link corresponding to a capacitance, e.g. an MOS transistor whose source and drain are connected to the same equipotential.
CNS_DIPOLE	A link corresponding to a dipole.

A large number of the above masks are generic types, they are rarely used in the affectation of a type to a link since the are included in the non-generic types. For example the type CNS_RESIST includes the masks CNS_PASSIVE and CNS_2EQUIP.

The generic types are included to facilitate the testing of links, since they allow certain type groups of links to be tested for using a comparison with a single mask.

2.4.5. The Edge List

Edge Structure Fields

The edge object, of type edge_list, is made up of the following fields:

struct link *NEXT	Pointer to the following edge in the cone's list of edges.
long TYPE	The logical sum of masks indicating the type and nature of the edge.
union uedge UEDGE	Pointer to the object to which the edge refers, CONE, LOCON or PTR.
struct ptype *USER	User defined information.

Note that the UEDGE field is a union since it can point to a transistor (field CONE of type cone_list*) or a connector (field LOCON type locon_list).

Standard Branch Types

CNS_VSS CNS_VDD	An edge corresponding to a cone built on a Vdd(Vss) power supply node.
CNS_GND	An edge corresponding to a cone built on a ground node (exists only in GaAs).
CNS_EXT	Indicates that the edge is an external connector and hence that the pointer in the UEDGE union is of type locon_list*.
CNS_CONE	Indicates that the edge is a cone and hence that the pointer in the UEDGE union is of type cone_list*.
CNS_BLEEDER	Indicates that the edge corresponds to the input, or corresponding output of a bleeder loop (see figure 5a).
CNS_COMMAND	Indicates that the edge corresponds to a command input of a latch cone.
CNS_LOOP	Indicates that the edge forms part of a two cone loop.
CNS_FEEDBACK	Indicates that the edge corresponds to the input, or corresponding output of a latch feedback loop

2.4.6. The Transistor List

The CNS transistor object uses the same data structure as MBK, the lotrs_list structure, hence the reader is referred to the MBK documentation for details. CNS does however define its own TYPE masks as well as additional USER types for the transistors.

The TYPE masks are:

CNS_TN	defined as TRANSN
CNS_TP	defined as TRANSP

The additional USER types are:

CNS_INDEX
A unique number for the transistor wihin the circuit, used by the parser/driver in order to refer to individual transistors.

CNS_LINKTYPE
Contains the TYPE affected to any links referring to the transisitor.

CNS_DRIVINGCONE
Contains a pointer to the cone built on the equipotential to which the transistor gate is connected.

CNS_CONE
Contains a chain_list of the cones containing links referring to the transistor. This field is rarely created due to reasons of memory efficiency.

2.4.7. The Connector List

The CNS connector object uses the same data structure as MBK, the locon_list structure, hence the reader is referred to the MBK documentation for details. CNS does, however, define additional USER types for the connectors.

The additional USER types are:

CNS_INDEX	A unique number for the connector within the circuit, used by the parser/driver in order to refer to individual connectors.
CNS_EXT	Contains a pointer to the cone built on the equipotential connected to the external connector.
CNS_CONE	Contains a chain_list of the cones containing links referring to the external connector. This field is rarely created due to reasons of memory efficiency.

2.4.8. The Cell List

Cell Structure Fields

The cell object, of type cell_list, is made up of the following fields:

struct link *NEXT	Pointer to the following cell in the figure's list of cells.
long TYPE	The logical sum of masks indicating the type of the cell.
chain_list *CONES	The list of cones contained within the cell.
struct befig *BEFIG	The behavioral description of the cell.
struct ptype *USER	User defined information.

Standard Cell Types

A number of standard cell types have been defined in the CNS header to deal with the recognition of GaAs cone configurations, it is possible that this list will be extended to include certain standard CMOS forms.

The convention for adding a user defined cell type so that it is recognized as such by the CNS driver is to sum a desired type reference number with the constant CNS_UNKNOWN. This allows the driver to identify that the cell is not defined within CNS but is nonetheless a legal type, hence the cell is identified within the CNS file by the index and not by a name as is the case for the predefined types.

Chapter 3. Log Files

3.1. REP - Report File

Each execution of Yagle results in the generation of a report file. This file is given the name <input_name>.rep. It contains a list of diagnostics (warnings and error messages) attributed to particular signals or transistors within the input net-list. Here we explain in more detail the particular messages which you may come across in this report file.

3.1.1. Warning Messages

"[WAR] Possible unconnected supply ?"
Means that an internal signal whose name contains `avtVddName` or `avtVssName` has been found. Verify if this signal should be connected to an external supply, or if `avtGlobalVddName` and `avtGlobalVssName` should be positioned.

"[WAR] Transistor used as a resistance"
Indicates that a transistor P-channel (resp. N-channel) with gate connected to the ground (resp. power supply) has been found in the circuit.

"[WAR] Transistor used as a diode"
Indicates that a transistor with drain (or source) connected to gate has been found in the circuit, and the signal connecting them is neither power supply nor ground.

"[WAR] Transistor is always off"
Indicates that a transistor P-channel (resp. N-channel) with gate connected to power supply (resp. ground) has been found in the circuit.

"[WAR] Transistor used as a capacitance"
Indicates that a transistor with drain and source connected together has been found in the circuit.

"[WAR] Gate of transistor is not connected"
Indicates that a transistor gate which is connected to nothing has been found in the circuit.

"[WAR] Drain of transistor is not connected"
Indicates that a transistor drain which is connected to nothing has been found in the circuit.

"[WAR] Source of transistor is not connected"
Indicates that a transistor source which is connected to nothing has been found in the circuit.

"[WAR] Transistors are not used in the circuit"
This means that these transistors are not used to pull up or pull down any transistor gate in the circuit, or any external connector. This occurs for example if the output of a gate does not drive anything: In this case Yagle considers the transistors of the gate to be unused.

"[WAR] Loop between 2 gates (bleeder found)"
This means that a loop corresponding to a bleeder has been found in the circuit.

"[WAR] Loop between 2 gates (latch found)"
This means that a loop corresponding to a latch has been found in the circuit.

"[WAR] Loop between 2 gates (bi-stable found)"
This means that a loop corresponding to a bi-stable has been found in the circuit.

"[WAR] Loop between 2 gates (nothing found)"
This means that a two gate loop which does not correspond to a latch, bleeder or bi-stable has been found in the circuit.

"[WAR] Conflict may occur on signal"
This means that the signal may be pulled-up and pulled-down simultaneously. This is a warning since this message may disappear with a greater depth for the functional analysis process. Or it may not be possible to resolve the conflict given the logic within the circuit.

"[WAR] HZ state may occur on signal"
This means that the signal is not pulled up or pulled down for any set of input stimuli on the cone entries. This is a warning for the same reason as a conflict.

"[WAR] Signal does not drive anything"
This means that the signal is not used as the input to any gate or used to drive any external connector.

"[WAR] Connector unused"
This means that the external connector is neither the input nor the output of any of the extracted transistor gates.

3.1.2. Error Messages

The presence of any of the following errors will disable the generation of the VHDL or Verilog description. If this behavior is not desired then Yagle must be executed with the `yagleNotStrict` variable.

"[ERR] Bad direction on connector"
Indicates that the orientation of an external connector after disassembly does not correspond to that specified in the input netlist.

"[ERR] Transistor gate signal is not driven"
Indicates that a transistor gate can not be pulled up or down.

3.1.3. Fatal Errors

The following error messages will not be found in the report file. These errors are fatal and will abruptly stop the execution of Yagle.

"[FATAL] No VDD/VSS connector in the circuit"
This means that Yagle did not find any external ports whose name is the name of the power supply in the circuit. Do `avtVddName` and `avtVssName` have the right value?

"[FATAL] Connector is power supply and ground"
This means that Yagle found a connector whose name includes `avtVddName` and `avtVssName`.

"[FATAL] No VDD/VSS signal in the circuit"
This means that Yagle did not find any signal whose name is the name of the power supply in the circuit.

"[FATAL] Several external connectors on signal"
This means that Yagle found several external connectors connected to the same equipotential, a configuration which Yagle considers illegal.

3.2. User-defined Log File

A global log file can be generated, logging the proccessing of all the components of the software. This file is customizable, and user can choose which component to log, and the level of log to apply.

Each line in the log file is beginning with the code related to the logged software component:

`FAC`	file access tracing
`MCH`	disk cache tracing (used for `.stm`, `.rcx` and `.spef` files)
`MCC`	MOSFET characterization
`RCN`	RC networks construction
`TRC`	RC networks characterization
`YAG`	transistor netlist disassembly
`TAS`	information related to delay calculation
`STM`	information related to delay models
`EFG`	spice deck generation

`GSP`	automatic stimuli generation
`TLF`	`.tlf` file generation
`LIB`	`.lib` file generation
`ERR`	error redirection in log file
`PRS`	statistics related to netlist parsing
`SPI`	detailed logging of the spice netlist and technology file parser

The `avtLogFile` variable activates the creation of the log file. The `avtLogEnable` variable selects the software components to log and the level of log. Please refer to the 'Configuration Variables' chapter for more details.

Chapter 4. Configuration Variables

4.1. License Server

avtLicenseServer
`<string>` Hostname of the machine running the license server

avtLicenseProject
`<string>` Project name. Used in license logging.

4.2. Environment

avtLibraryDirs
`<string>` The set of library directories which are scanned for required subcircuits.

avtBlackboxFile
`<string>` Name of the file containing the cells to exclude of analysis.

avtCatalogueName
`<string>` File containing a list of subcircuits to be considered as leaf cells when flattening a design. Each line in this file refers to a single subcircuit, with the format `<subcircuit> C`. The default value is `CATAL`.

4.3. Names

avtVddName
`<string>` Name of any signal or connector which is to be considered as power supply (a `*` in the name matches any string). Several names, separated by `:`, may be specified.

avtVssName
`<string>` Name of any signal or connector which is to be considered as ground (a `*` in the name matches any string). Several names, separated by `:`, may be specified.

avtGlobalVddName

`<string>` Name of an internal signal to be considered as power supply (a `*` in the name matches any string). Signals in different subcircuits of a hierarchical netlist with a name given here will be considered as equipotential and this name will be used in the flattened netlist. This is identical to the use of the .GLOBAL directive in a spice netlist. Several names, separated by `:`, may be specified.

avtGlobalVssName

`<string>` Name of an internal signal to be considered as ground (a `*` in the name matches any string). Signals in different subcircuits of a hierarchical netlist with a name given here will be considered as equipotential and this name will be used in the flattened netlist. This is identical to the use of the .GLOBAL directive in a spice netlist. Several names, separated by `:`, may be specified.

avtCaseSensitive

`yes` Upper and lower case characters are distinct

`no` Upper and lower case characters are seen as identical

`preserve` Default, upper and lower case characters are seen as identical but the original case is preserved

avtInstanceSeparator

`<char>` Character used to separate instance names in a hierarchical description. Default value is `.`

avtFlattenKeepsAllSignalNames

`yes` When flattening a netlist, each signal keeps all its names through the hierarchy.

`no` Default, only one name (the shortest) is kept per signal.

avtVectorize

Controls the internal representation of vector-signals.

`yes` Default, vector-signals are represented internally as vectors, as far as the vector indexation is one of [], <>, _. For example, if both `foo[1]`, `foo<1>` and `foo_1` appear in the source file, they will all be represented internally as `foo 1`

`no` Vector signals are represented internally as they appear in the source file.

`<string>` Explicits the vector-signals indexations that will be interpreted as vectors, and the represented internally as vectors. `string` is a comma-separated list of single or paired delimiters. For example, if `string` is set to `"[],_"`, only `foo[1]` and `foo_1` will be represented internally as `foo 1`.

Special attention should be paid to the Verilog case. Verilog only accepts `[]` as legal vector indexation. Legal verilog vectors are represented internally as vectors if `avtVectorize` is different to `no`.

Illegal Verilog vectors are supported and controlled by `avtVectorize` as far as they are escaped and `avtStructuralVerilogVectors` is set to `yes`. For example, `\foo<1>` is represented internally as a vector if `avtStructuralVerilogVectors` is set to `yes` and `avtVectorize` is set to `<>`.

4.4. Technology

avtElpCapaLevel

Allows the user to compute differents kind of input capacitance.

`0` Input capacitance is the average between up and down capacitance

`1` Default, nominal up and nominal down capa are used to compute timing

`2` Same behavior as if set to 1 but also minimal and maximal capacitances are computed for both transitions (6 capacitances at all).

avtTechnoModelSeparator

`<char>` Character that will be used as a separator between the model name and the model index. Default value is `.`

avtElpDriveFile

`yes`	A ELP file specified by `avtElpGenTechnoName` will be printed after transistor electrical characterization.
`no`	default

avtElpGenTechnoName

`<string>`	Name of the generated ELP file. Default is `techno.elp`.

4.5. Input Netlist and Parasitics

avtInputFilter

`<string>`	Shell command line used to decompress an input netlist

avtOutputFilter

`<string>`	Shell command used to compress an output file

avtFilterSuffix

`<string>`	Suffix of the compressed files

avtDisableCompression

`<string>`	Space separated filename list for which compression must be ignored. EXAMPLE: "*.rcx *.rep"

avtAnnotationKeepCards

`transistor`	`M` character is kept before the transistor name
`diode`	`D` character is kept before the diode name
`resistance`	`R` character is kept before the resistance name
`instance`	`X` character is kept before the instance name
`capacitance`	`C` character is kept before the capacitor name
`none`	No character is kept
`all`	`M`, `R`, `X`, `C`, `D` characters are kept

avtMaxCacheFile

`<int>`	If cache mechanisms are used, sets the maximum number of files that can be opened at the same time. Larger the value is, faster is the disk access. Default value is 128. Maximum value depend on your system (see UNIX command `limit`).

avtParasiticCacheSize

`<int>[Kb|Mb|Gb]`	Size (bytes) of the memory cache for all applications dealing with parasitics. Value represents the maximum amount of information stored in memory. Increase this value to lessen disk access and speed-up application. `avtParasiticCacheSize` cannot be used together with compressed files.
`10Mb`	Default
`0`	Disable cache and load all the parasitic information

avtFlattenForParasitic

`yes`	Yagle flattens a hierarchical netlist in order to annotate the netlist with SPEF or DSPF parasitics. To be used together with `avtCatalogueName`
`no`	default

avtVddVssThreshold

`<float>`	Value (in volts) defining the absolute voltage value level above which a node is considered to be a power supply node. Default value is 0.5.

4.6. SPICE Parser

avtSpiCreateTopFigure

`yes`	Default, parser automatically creates a top-level for all elements outside of SUBCKT definition. All equipotentials are made into external connectors. The name of the top-level is the same as the filename without the extension unless a subcircuit of this name exists, in which case the name is prefixed by `top_`
`no`	No top-level is created

avtSpiParseFirstLine

`yes`	First line of all SPICE files are taken into account, unlike the behavior in standard SPICE
`no`	First line of all SPICE files are ignored
`include`	Default, first line of the top-level SPICE file is ignored, but the first line of included files are parsed normally

avtSpiReplaceTensionInExpressions

`yes`	Avoids expression evaluation errors due to unhandled dynamic tensions in expression. The voltage is considered to be 0.
`no`	Default.

avtEnableMultipleConnectorsOnNet

`yes`	By default, there can only be one external connector per net after a netlist parse. If multiple connectors are found, they are merged into one. This can have a big drawback. Connectors required on the interface of a top level netlist can be missing. There can also be issues for ignoring instances with transparencies using hierarchical names as transparences are analysed to build nets prior to check ignored instance resistors. Setting this variable to yes allows multiple external connectors on nets so transparences are analysed during the resistor removal step without the nets being shorted already. This has an effect on ignored instances containing transparences. It affects Yagle behaviour and may make it not work in hierarchical mode.
`no`	Default.

avtSpiMergeConnector

`yes`	Default, connectors with the same radical, but different node indexes, will be merged (they are supposed to belong to the equipotential outside the subcircuit). The separator between the radical and the index is given by `avtSpiConnectorSeparator`.
`no`	Connectors are not merged

avtSpiConnectorSeparator

`<char>`	Character used to separate a connector radical name from its node index (`ck.1`, `ck.2` ... for example).

avtSpiKeepNames

`transistor`	Transistor name is kept in the database
`diode`	Diode name is kept in the database
`resistance`	Resistance name is kept in the database
`allnodes`	All node names are kept for signals in the database
`none`	No name is kept in the database
`all`	All names are kept in the database

avtSpiKeepCards

`transistor`	`M` character is kept before the transistor name
`diode`	`D` character is kept before the diode name
`resistance`	`R` character is kept before the resistance name
`instance`	`X` character is kept before the instance name
`capacitance`	`C` character is kept before the capacitor name
`none`	No character is kept
`all`	`M`, `R`, `X`, `C`, `D` characters are kept

avtSpiNameNodes

`yes`	Default, nodes names are used rather than the node numbers
`no`	Only node numbers are used

avtSpiNodeSeparator

`<char>`	Character that will be used as a separator between the node name and the node number. The default value is _

avtSpiInstanceMultiNode

`yes`	Default, allows two or more identical nodes to be declared in a subckt interface
`no`	Only the first node declared is taken into account

avtSpiIgnoreDiode

`yes`	Diodes are ignored by the SPICE parser.
`no`	Diodes are characterized.

avtSpiMergeDiodes

`yes`	Diodes are merged with neighboring transistors if the transistor is of the same type and area of the connected source or drain is 0.
`no`	Diodes are characterized independantly.

avtSpiIgnoreVoltage

`yes`	Voltage sources are ignored by the SPICE parser.
`no`	Voltage sources are not ignored.

avtSpiIgnoreModel

`yes`	Model directives are ignored by the SPICE parser.
`no`	Model directives are not ignored.

avtSpiIgnoreCrypt

`yes`	Encryption directives (used to indicate encrypted data) are ignored.
`no`	The default. Encryption directives must surround encrypted test obtained by `avt_EncryptSpice` function.

avtSpiJFETisResistance

`yes`	JFETs are considered to be resistances. Values are resolved by the SPICE parser.
`no`	

avtSpiShortCircuitZeroVolts

`yes`	Voltage sources with a value of 0 are modeled by the SPICE parser as resistances of 0 Ohms.
`no`	

avtSpiMaxResistance

`<float>`	If a resistance's value is greater than `float` (in Ohms), then the resistance is considered to be open circuit.

avtSpiMinResistance

`<float>`	If a resistance's value is less than `float` (in Ohms), then the resistance is considered to be short circuit.

avtSpiMinCapa

`<float>`	If a capacitance's value is less than `float` (in Ohms), then the capacitance is ignored

avtSpiOneNodeNoRc

`no`	Removes on all nets containing only one node all parasitics information at the end of the parse.
`yes`	

avtSpiOrderPinPower

`yes`	Uses the name (in the same manner as `avtSpiDspfBuildPower`) of the instance nodes to ensure a correct order for power supply connectors.
`no`	

avtSpiFlags

This configuration is used to control the behavior of the spice parser/driver. The values (flags) are added separated with commas.

`DriveInstanceParameters`	Enables the drive of the instances with all their parameters
`IgnoreGlobalParameters`	Works with `DriveInstanceParameters` and removes all the global parameters from the instance parameters to drive. Useful when the netlist has been flattened and the parameters inherited by the leaf instances.
`KeepBBOXContent`	Will keep the content of the figures set as blackboxes whereas by default only the interfaces are kept.
`TransfertTopLevelVcards`	Will transfert voltage sources connected to instances, who are defined out of a subckt in the spice file, in their corresponding circuit subckt so the Vcards can be taken into account when working on one of this instance circuit. This option is enabled by default. It can be unset by adding '!' in front of the option: '! TransfertTopLevelVcards'.
`ExplicitInstanceNames`	If enabled then instance names specified in the netlist are prefixed by the subckt name in order to create the internally used name.

avtSpiTolerance

This variable tunes the tolerance of the SPICE parser regarding unrecognized syntaxes for R (resistances) and C (capacitances) devices.

`low`	Parser exits when encountering unknown syntax
`medium`	Parser continues and tries to keep only the nominal value of the device, issuing a warning message
`high`	Same as in the `medium` configuration, but no warning message is issued

avtSpiHandleGlobalNodes

`yes`	Default, global nodes defined in spice netlist without resistances will be considered equipotential.
`no`	

4.7. SPICE Driver

avtSpiVector

`_`	Default, vectors are of the shape `foo_1` in output spice files
`[]`	Vectors are of the shape `foo[1]` in output spice files
`()`	Vectors are of the shape `foo(1)` in output spice files
`<>`	Vectors are of the shape `foo<1>` in output spice files

avtSpiDriveDefaultUnits

`<string>`	Its behavior is to indicate the parameter units to be used when instantiating a transistor. For instance, `avtSpiDriveDefaultUnits = W:1e-6;L:1` will set the spice driver to drive parameter W value in micron and parameter L in meter.

avtSpiUseUnits

`yes`	Allows the use of units in driven spice files. This is the default.
`no`	

avtSpiDriveParasitics

`yes` A SPEF file will be generated while parsing a SPICE file. The loaded file will be stripped of all resistors and capacitors. The SPEF file can be used as a parasitic cache file.

`no`

avtSpiDriveTrsInstanceParams

`no` Specifics instances parameters for the models of transistors will not be driven.

`yes`

avtSpiDriveCapaMini

`<float>` When driving a Spice netlist, doesn't drive capacitances below `float` (in Pico-farads). Default is 10-6 pF.

avtSpiDriveResiMini

`<float>` When driving a Spice netlist, fix minimum value for resistances to `float` (in Ohms). Default is 10e-3 Ohms.

avtSpiRCMemoryLimit

`<int>` Amount of memory in MB allowed to creating a .SPEF file from a spice file. This option influences `avtSpiDriveParasitics` speed. The default value is 100.

avtSpiFlags

This configuration is used to control the behavior of the spice parser/driver. The values (flags) are added separated with commas.

`DriveInstanceParameters`	Enables the drive of the instances with all their parameters
`IgnoreGlobalParameters`	Works with `DriveInstanceParameters` and removes all the global parameters from the instance parameters to drive. Useful when the netlist has been flattened and the parameters inherited by the leaf instances.
`KeepBBOXContent`	Will keep the content of the figures set as blackboxes whereas by default only the interfaces are kept.
`TransfertTopLevelVcards`	Will transfert voltage sources connected to instances, who are defined out of a subckt in the spice file, in their corresponding circuit subckt so the Vcards can be taken into account when working on one of this instance circuit. This option is enabled by default. It can be unset by adding '!' in front of the option: '! TransfertTopLevelVcards'.

4.8. VHDL Parser/Driver

avtVhdlMaxError

`<int>`	Maximum number of errors before the VHDL structural parser abandons.

avtStructuralVhdlConfigure

`yes`	VHDL structural driver generates the appropriate configuration statement to allow simulation.
`no`	Default

avtStructuralVhdlSuffix

`<string>`	Suffix of VHDL structural (netlist) file. The default is `vhd`

avtBehavioralVhdlSuffix

`<string>`	Suffix of VHDL behavioral file. The default is `vhd`

4.9. VERILOG Parser/Driver

avtVerilogKeepNames

`yes`	When generating Verilog output, any internal names which are not legal verilog names are preceded by a double backslash.
`no`	Default. Illegal names are modified to create a legal name.

avtStructuralVerilogVectors

Affects the parsing of illegal Verilog vector-signals in a netlist, i.e. vector-signals that are not indexed using the `[]` characters. Illegal Verilog vector-signals are supported as long as they are preceded by `\`, otherwise the Verilog parser issues a syntax error. Legal Verilog vector-signals are controlled by `avtVectorize`.

`yes`	Force illegal Verilog vector-signals to be represented as vectors in the internal database, with regard to the value of `avtVectorize`. For example, `\foo<1>` is represented internally as `foo 1` if `avtVectorize` is set to `<1>`
`no`	Default, illegal Verilog vector-signals are represented in the internal database as they appear in the file. For exemple, `\foo<1>` is represented internally as `foo<1>`

avtStructuralVerilogSuffix

`<string>`	Suffix of Verilog structural (netlist) file. The default is `v`

avtBehavioralVerilogSuffix

`<string>`	Suffix of Verilog behavioral file. The default is `v`

avtVerilogMaxError

`<int>`	Maximum number of errors before the Verilog parser abandons.

4.10. DSPF/SPEF Parser

avtAnnotationPreserveExistingParasitics

`yes`	Existing parasitics on nets annotated in a DSPF/SPEF file won't be overridden by the parasitics in the DSPF/SPEF file. The DSPF/SPEF information will rather be added to the existing ones.
`no`	Default

avtAnnotationDeviceConnectorSetting

`<string>` Overrides the internal tool known device connector names used in DSPF/SPEF annotation. The string must contain 10 items in the following order: transistor source name, transistor gate name, transistor drain name, transistor bulk name, resistor positive connector, resistor negative connector, capacitor positive connector, capacitor negative connector, diode positive connector, diode negative connector. By default, the tool knows of "s g d b 1 2 1 2 1 2" and "s g d b pos neg 1 2 1 2".
Example:

```
avt_config avtAnnotationDeviceConnectorSetting "src gate drn blk 1 2 1 2 1 2"
```

avtSpiDspfBuildPower

`yes` Only used for DSPF annotation. When creating a figure from DSPF information, use the avtGlobalVddName, avtGlobalVssName, avtVddName and avtVssName to detect power connections on the instance, so they are created on the boundary of it instead of being merged with all unknown connectors.

`no`

avtSpiDspfLinkExternal

`yes` Only used for DSPF annotation. When an external connector is not connected to anything, and if there is an internal signal with the same name, then the connector is assumed to be on this signal.

`no`

avtSpiPinDspfOrder

`yes` Only used for DSPF annotation. Order of connector of an instance is the one described in the DSPF instead of the one described for the instance interface.

`no`

4.11. General Configuration

yagHierarchicalMode

`yes`	Activates the hierarchical disassembly mode.
`no`	Default

yagWriteStatistics

`yes`	Statistics of detected power supplies and particular transistor configurations are saved in a file with the suffix `.stat`.
`no`	Default

yagMutexHelp

`yes`	An algorithm which attempts to guess any misssing MUTEX constraints is activated. Groups of signals which could have constraints are reported in a file of suffix `.mutex`.
`no`	Default

yagSearchLoops

`yes`	An algorithm to detect combinatorial loops of more than two gates is activated. Any loops detected are reported in a file of suffix `.loop`.
`no`	Default

yagDebugCone

`<string>`	If set to the name of a cone, then additional debug information for that cone is displayed.

yagNotStrict

`yes`	Certain aspects of the net-list coherency are not verified, such as un-driven transistor gates.
`no`	Default

yagElpCorrection

`yes`	Updates the capacitances to take into account technology dependant factors, such as diffusion capacitance, gate capacitance and shrink.
`no`	Default

yagSuppressBlackboxes

`yes` Reads a hierarchical net-list in which some of the instances are considered to be black boxes (i.e. their internal structure is unavailable). The list of these instances is given by the user in a file whose name is given by `avtBlackBoxFile`. Yagle creates a new intermediate netlist containing only the non-black box instances, and modifies the original net-list to instantiate this new figure and the black box instances. The modified original netlist is saved to disk, and the functional abstraction is performed on the intermediate figure.

`no` Default

yagIgnoreBlackboxes

`yes` Reads a hierarchical netlist in which some of the instances are considered to be black boxes. The name of these instances is given in a file whose name is given by `avtBlackBoxFile`. The hierarchical netlist is then flattened to the transistor level apart from the black box instances to generate a hybrid transistor and instance netlist. The functional abstraction is performed on this hybrid netlist.

`no` Default

yagRemoveInterconnects

`yes` Deletes parasitic information before performing desassembly and functional abstraction.

`no` Default

yagSilentMode

`yes` Yagle does not write anything to the standard output.

`no` Default

4.12. Disassembly

4.12.1. Functional Analysis

yagAnalysisDepth

`<int>`	Allows the user to set the depth for the functional analysis. This is the number of gates that will be taken into account for the functional analysis, so that Yagle can detect re-convergence in the circuit. Default is 7.
`0`	Functional analysis process is disabled

yagHzAnalysis

`yes`	Allows functional analysis through high impedance nodes.

yagMaxBranchLinks

`<int>`	Maximum number of links in a cone branch.

yagRelaxationMaxBranchLinks

`<int>`	Used to limit the maximum number of links for the difficult gates for which functional dependencies could not be resolved.

yagBddCeiling

`<int>`	Limits the maximum number of BDD nodes which are allowed to be created for the resolution of any Boolean expression. If this limit is exceeded the operation is abandoned. Default is 10 000.

yagElectricalThreshold

`<float>`	Used in electrical resolution of conflicts to determine the zones corresponding to the high, low and conflictual states. Default is 4, implying that the high and low states are represented by zones 1/4 of the zone Vss-Vdd.

yagUseStmSolver

`yes`	Precise current calculations using technology files are used in electrical conflict resolution.
`no`	Default, basic transistor dimensions are used in electrical conflict resolution.

yagRelaxationAnalysis

During the gate construction phase, Yagle attempts to resolve all functional dependencies before building a particular gate. However, in particular cases of looped dependencies, this may not be possible for all gates.

`yes`	Functional dependencies are ignored to resolve these gates.
`no`	Default, Yagle tries to use as much information as possible.

yagDetectGlitchers

`yes`	A branch containing two transistor with mutually exclusive gate drivers and which cannot be part of another gate are assumed to exist dynamically. They are therefore not removed by the functional analysis. This is the default.
`no`	

yagKeepRedundantBranches

For any CMOS dual cones extracted, if supplementary branches are added at a later stage of the disassembly and the gate remains non-conflictual, then these branches are considered to be functionally redundant.

`yes`	The branches are kept.
`no`	Default, the branches are removed.

yagPullupRatio

`<float>`	Used in the detection of pull-up or pull-down resistance transistors. Default is 10, implying that a transistor is a pull-up if an estimation of its resistance is greater than 10 times the resistance of the most resistive current path to ground. Similarly for pull-pown resistances.

4.12.2. Transistor Orientation

yagSimpleOrientation

`yes`	Activates a simple transistor orientation heuristic. Can sometimes accelerate the disassembly, however, it is more robust to rely exclusively upon the functional analysis.
`no`	Default.

yagUseNameOrientation

`yes`	Exploits the _s naming convention for transistor orientation.
`no`	Default.

yagBlockBidirectional

`yes`	Bidirectional transistors are not allowed.
`no`	Default.

yagCapacitanceCones

`yes`	Default. Build cones on nodes with only capacitances. Necessary to calculate the out of path capacitance.
`no`	Disables construction of cones on only capacitance nodes.

yagTestTransistorDiodes

`yes`	Default. Any transistor with the gate shorted to a source or a drain is considered as a diode.
`no`	Disables diode detection.

yagMutexHelp

`yes`	A file with extension '.mutex' is generated containing help for MUTEX settings on external pins or memory nodes necessary to correctly orient the transistors.
`no`	Default.

4.12.3. Latch Recognition

yagSimpleLatchDetection

A simple structure based recognition algorithm which handles the various cases of double inverter loops. This approach is not usually required and is not guaranteed to be formally correct but can sometimes help in cases where the automatic approach is too CPU intensive. The following values can be given for this variable:

`memsym` Double inverter loops are also analyzed to see if they correspond to a simple symmetric bitcell. In this case the the command of the bitcell is the input of the pass transister or transfer gate connected directly to the loop.

`levelhold` Double inverter loops are considered to be level-hold or buskeeper structures (i.e. not latches).

`strictlevelhold` Double inverter loops are considered to be level-hold or buskeeper structures (i.e. not latches), but only if only one side of the inverter loop is connected.

`latch` Double invertor loops are treated as latches without any anlysis, unless a level-hold or memsym option is also activated and these forms match. Commands are guessed without analysis. This option helps if double inverter loops are used to latch the output of complex multiplexors.

The above options can be concatenated by separating the individual options with a '+' character. However the combinations "levelhold+strictlevelhold" and "levelhold +latch" make no sense. The search options are applied in the order specified above. By default all the options are disabled.

yagAutomaticLatchDetection

`yes` Advanced latch detection algorithm based on Boolean loop analysis is activated. Default.

`no` Advanced latch detection is disabled.

yagSetResetDetection

`yes` Only works with `yagAutomaticLatchDetection` set to `yes`. Asynchronous latch commands are marked asynchronous instead of being marked commands. False timing arcs (corresponding to the conditioning of data by an asyn or the conditioning of an async by a clock) are disabled.

`remove` Does the same as the `yes` mode. In addition marks as non-functional any branches corresponding to an asynchronous write.

`no` Default.

yagAutomaticRSDetection

`mark`	Default. Only works with `yagAutomaticLatchDetection` set to `yes`. Supplementary RS bistable detection algorithm is applied to automatically detected latches. Only NAND/NOR types are accepted. Any detected RS bistable loops will be marked and reported but not treated as latches.
`no`	Automatic RS detection is disabled. Recognition depends upon `yagAutomaticLatchDetection`
`mark+latch`	One of the gates of the loop is considered a latch. The latch is the gate with the largest number of outputs.
`mark+legal`	The algorithm assumes that an RS structure always remains in its legal states. Timing arcs are suppressed accordingly. For NOR-based RS, the following timing arcs are suppressed: S(f) to QB(r), R(f) to Q(r), QB(r) to Q(f) and Q(r) to QB(f). For NAND-based RS, the following timing arcs are suppressed: S(r) to QB(f), R(r) to Q(f), QB(f) to Q(r) and Q(f) to QB(r).
`mark+illegal`	The algorithm assumes that an RS structure may enter an illegal state. Less timing arcs are suppressed than when the tool assumes that an RS structure always remains in its legal states. For NOR-based RS, the following timing arcs are suppressed: Q(r) to QB(f) and QB(r) to Q(f). For NAND-based RS, the following timing arcs are suppressed: Q(f) to QB(r) and QB(f) to Q(r).

yagAutomaticMemsymDetection

`yes`	Only works with `yagAutomaticLatchDetection` set to `yes`. Supplementary symmetric memory detection algorithm is applied to automatically detected latches. Symmetric memories are memorizing elements such as bitcells for which data is written in both or either side of the memorizing loop. Both sides of the loop are marked as latches is order to verify all cases.
`no`	Default.

yagDetectDynamicLatch

`yes`	Internal tri-state nodes are considered to be dynamic latches for functional modeling and timing analysis purposes. A special algorithm, similar to that used in the automatic latch detection, is used to identify the latch commands and generate an accurate latch model.
`no`	Default.

yagDetectPrecharge

`yes`	An algorithm designed to detect automatically most kinds of precharge nodes is activated. The algorithm is particularly designed for domino precharge style designs.
`no`	Default.

yagBleederStrictness

A level between 0 and 2 defining the strictness of the bleeder detection algorithm. The value determines the kind of gate which can be tolerated in the bleeder loop.

`0`	any CMOS gate is acceptable
`1`	default, any CMOS dual gate is acceptable
`2`	it must be an inverter

yagStandardLatchDetection

Deprecated. This structure based latch recognition technique is activated by default as a catch-all. It will probably be removed in a future version.

`yes`	Default, standard latch detection algorithm is activated.
`no`	Standard latch detection algorithm is disabled.

yagLatchesRequireClocks

`yes`	Any latch which does not have a command which is at the end of a path from a specified clock is not considered to be a latch. If this option is used, then extreme care should be taken to specify the clocks to avoid problems in any subsequent analysis.
`no`	Default.

yagDetectClockGating

`yes`	If clocks are configured before the disassembly phase then reconvergence between clock and data will be detected, appropriate timing check and data filtering directives are automatically generated.
`check`	Same as above except only the timing checks are added.
`filter`	Same as above except only the data filtering directives are added.
`no`	Default.

yagDetectDelayedRS

`yes`	Detect special type of NAND/NOR bistable loop structure containing additional inverters to add delay in the loop. Results in the same handling as the `legal` setting for RS detection. This type of structure is commonly used to generate non-overlapping clocks.
`no`	Default.

4.12.4. Pattern Matching

yagUseGenius

`yes`	Extends the simple pattern recognition of FCL to allow the recognition of hierarchically defined structures of generic size.
`no`	Default

yagUseOnlyGenius

`yes`	Same as `yagUseGenius` but Yagle stops the execution after the hierarchical pattern recognition phase.
`no`	Default

4.12.5. Behavioral Model Generation

yagTasTiming

`max`	delay information is calculated for annotation of the data flow description using timing characterization with worst case timings
`med`	delay information is calculated for annotation of the data flow description using timing characterization with average timings
`min`	delay information is calculated for annotation of the data flow description using timing characterization with best case timings

yagSplitTimingRatio

`<float>` Used if a timed behavioral model is generated. Models for some auxiliary signals will be enhanced to differentiate up and down transitions. This operation is performed if one of the transitions has a delay greater than `float` times the other. If the value is less than 1 then this operation is never performed. Note that this option should not be used if a Verilog behavioral model is to be generated sice verilog timings always contain this differentiation. The default value is 0.

yagSensitiveTimingRatio

`<float>` Used if a timed behavioral model is generated. Models for some auxiliary signals will be enhanced to differentiate the timing according to the input which actually changes. `float` corresponds to the minimum ratio between the greatest and the least timing value above which the operation is performed. If the value is less than 1 then this operation is never performed. The default value is 0.

yagMaxSplitCmdTiming

`<int>` Used if a timed behavioral model is generated. Models for some busses or register signals will be enhanced to differentiate the timing according to each input combination which can change the value. `int` corresponds to maximum number of combinations under which the differentiation is applied. The default value is 0 (disabled).

yagSensitiveTimingDriverLimit

`<float>` Used to set an upper limit to the number of expression inputs for which `yagSensitiveTimingRatio` has an effect. If expression depends on mare variables than this limit the sensitive timing expression is not generated. Used to avoid unwieldy models for complex multiplexor structures.

yagOneSupply

`yes` Only one power supply and ground connector is defined in the interface of the behavioral description.

`no` Default

yagNoSupply

`yes`	Disables dumping of power supplies declaration into generated behavior.
`no`	Default

yagReorderInterfaceVectors

`yes`	All bussed connectors on the interface of the design being modelled are re-ordered such that they are defined as vectors with most significant bit first.
`no`	Default. Interface connectors are left in the order of the original design.

yagBleederIsPrecharge

`yes`	Bleeders are modeled as nodes which maintain their value.
`no`	Default

yagTristateIsMemory

`yes`	Internal high impedance nodes are modeled as nodes which maintain their value.
`no`	Default

yagAssumeExpressionPrecedence

`yes`	Signals with multiple drivers are modeled using a single cascaded IF statement, hence a precedence is assumed.
`no`	Default

yagSimplifyExpressions

`yes`	Boolean expression simplification is performed on the final model.
`no`	Default

yagSimplifyProcesses

`yes`	Simplifies the expressions of the behavioral data flow processes.
`no`	Default

yagMinimizeInvertors

`yes`	Chains of invertors are reduced in the final model.
`no`	Default

yagCompactBehavior

`yes`	A compaction algorithm is applied on the generated model capable of generating vectorized and looped assignations in order to reduce the size of the code.
`no`	Default

yagBusAnalysis

`yes`	Uses a functional analysis algorithm to distinguish individual drivers of bussed signals.
`no`	Default

yagDriveConflictCondition

`yes`	Latches and bussed signals for which a conflict condition is detected after all analysis are modeled with this conflict condition.
`no`	Default. The conflictual condition is ignored.

yagDriveAliases

`yes`	Drives a file with the extension .aliases with information on the circuit hierarchy. This file is used when using the tool avt_vcd2hvcd.tcl that rebuild a hierarchical .vcd from from a flat .vcd file.
`no`	Default. No file generated.

4.12.6. Cone Output Files

avtVerboseConeFile

`yes`	Generating a `.cnv` cone file result in a more readable version but which is not suited for GUI vizualisation.
`no`	Default.

avtNormalConeFile

`yes`	A normal `.cns` cone file is produced.
`no`	Default.

avtFullConeFile

`yes`	`.cnv` or `.cns` cone files are generated with parasitic information.
`no`	Default.

yagGenSignature

`yes`	Signatures are generated for each cone which are used to associate icons with the cones.
`no`	Default.

4.13. Output Configuration

yagOutputName

`<string>`	Name given to the generated behavioral data flow description
`no`	Default

yagGeniusTopName

`<string>`	Name given to the root structural figure of a hierarchical netlist generated as a result of GNS

yagGenerateBehavior

`yes`	Default, generates the behavioral data flow description.

yagGenerateConeFile

`yes`	A `.cns` file is generated, giving details of the disassembled gates
`no`	Default

yagGenerateConeNetList

`yes`	Generates a structural description of the disassembled gates together with a behavioral model for each distinct gate type
`no`	Default

yagHierarchyGroupTransistors

`yes`	In hierarchical abstraction mode, any transistors left in an otherwise fully modeled design hierarchy, are grouped together and abstracted as if they were in their own subckt.
`no`	Default. Any transistors left over will be driven as such (in verilog) or ignored (in vhdl).

avtOutputBehaviorFormat

`vhd`	Output behavior format is VHDL
`vlg`	Output behavior format is Verilog

avtOutputBehaviorVectorDirection

`TO`	Vector signals generated by BEG functions will be expressed from the lower bound to the higher bound
`DOWNTO`	Default, vector signals generated by BEG functions will be expressed from the higher bound to the lower bound

4.14. Pattern Matching

fclLibraryName

`<string>`	Name of the file containing the list of cells in the user-defined cell library used. The default is `LIBRARY`.

fclLibraryDir

`<string>`	Access path to the directory containing the user-defined cell library used. Default is a directory `/cells` in `avtWorkDir`.

fclGenericNMOS

`<string>`	A colon separated list of transistor model names which the FCL pattern-matching engine considers will match to any N-type transistor. If a pattern netlist contains non-generic N-channel transistors then these transistors will only match to transistors with an identical model. Default is `tn:TN`.

fclGenericPMOS

`<string>` A colon separated list of transistor model names which the FCL pattern-matching engine considers will match to any PMOS transistor. If a pattern netlist contains non-generic P-channel transistors then these transistors will only match to transistors with an identical model. Default is `tp:TP`.

fclWriteReport

`yes` A correspondence file is created if the -fcl option is used. This file details all the recognized instances.

`no` Default

fclAllowSharing

`yes` Matched cells are allowed to share transistors.

`no` Default

fclCutMatchedTransistors

`yes` Matched transistors are eliminated from the transistor netlist. Results in a strict partitioning of the cones and the matched cells.

`no` Default

fclMatchSizeTolerance

`<int>` Percentage tolerance for matching transistor sizes.

fclTraceLevel

`<int>` Number greater than 0. Trace information is displayed during the pattern-matching phase.

fclDebugMode

`<int>` Number greater than 0. Additional debugging information is displayed during the pattern-matching phase.

4.15. Hierarchical Pattern Matching

gnsLibraryName

`<string>` Name of the file (recognition library) containing the list of cells to recognize. Default is `LIBRARY`.

gnsLibraryDir

`<string>` Access path to the directory containing the recognition library. Default is directory `cells/` in `avtWorkDir`.

gnsKeepAllCells

`yes` All matched structures are extracted from the netlist.

`no` Default

gnsTemplateDir

`<string>` Directory where to find the GNS templates. Default is `$AVT_TOOLS_DIR/gns_templates`.

gnsTraceLevel

`<int>` From 0 to 6. Indicates the level of trace displayed during the recognition phase. Default is 0.

gnsTraceFile

`<string>` Name of the output trace file. Default is stdout.

gnsTraceModel

`<string>` When tracing the recognition, indicates the name of the recognized model to trace. If not specified, traces all models.

gnsFlags

This configuration controls the behavior of GNS. The values (flags) are added separated with commas. Available flags are:

`EnableCore` Enable the generation of a core file for a crash in a user compiled API.

`NoGns` Disables the generation of the `.gns` file

`VerboseGns` Produces a more readable `.gns` file.

`NoOrdering` Disables the top-level instance connectors reordering. Should not be set if using the BEG functions.

4.16. API Specific

apiFlags

Controls the behavior of the GNS API. The values (flags) are added separated with commas. Available flags are:

`ttvUseInstanceMode`	Sets the TTV functions to generate/use one timing view per instance of the same matched subcircuit.
`ttvDriveDTX`	Enables the drive of the `.dtx` and `.stm` files for timing views created with the TTV functions

apiDriveCorrespondenceTable

`yes`	Correspondence table between behavioral names and electrical names has to be driven (`.cor` file)
`no`	Default

apiUseCorrespondenceTable

`yes`	Use the signal correspondance to revert the driven signals to their corresponding name in the original netlist. It removes the artificially created GNS hierarchical names.
`no`	Default

apiDriveAllBehavior

`yes`	All the generated behaviors will be driven.
`onlymodel`	Only the first instance of each model will be driven
`no`	Default

4.17. GUI

xyagIconLibrary

`<string>`	Full path of a specific .slib icon library

xyagMakeCells

`yes`	Complex gates are represented using a single icon
`no`	Default

Chapter 5. Tcl Interface

5.1. General

5.1.1. Configuration

avt_Config

`void avt_Config(char *var, char *val)`

Main way to configure the tool. Affects a value to one of the variables listed in the Configuration Variables section

`var` Configuration variable to be set

`val` New value

`EXAMPLE` `avt_Config tasGenerateConeFile yes`

---

avt_GetConfig

`char *avt_GetConfig(char *var)`

returns the configurated value for configuration variable `var`

`var` Configuration variable to be set

`EXAMPLE` `set cone_cfg [avt_GetConfig tasGenerateConeFile]`

5.1.2. File Loading

avt_SetBlackBoxes

`void avt_SetBlackBoxes(List *list)`

Allows the user to blackbox subcircuits. Blackboxed subcircuits will not be analyzed. Instead, the tool will let a hole. Whether this hole should be filled up or not by a timing description depends on configuration variables `tasIgnoreBlackbox` *and* `tasTreatBlackboxHierarchically`*. If a blackbox name is prefixed with "unused:", no hole will be created but instead all transistors in the blackbox will be marked as unused. Those*

blackboxes can still be retreived with GNS if the recognition rule uses the same transistor names as in the blackbox. This command is equivalent to and overrides the creation of a `BLACKBOX` *file.*

`list` List of subcircuits to be blackboxed. All intended blackboxed subcircuits should present as only one `avt_SetBlackBoxes` command is allowed.

`EXAMPLE` `avt_SetBlackBoxes [list "sense_amp"]`

---

avt_LoadBehavior

```
BehavioralFigure *avt_LoadBehavior(char *filename, char *format)
```

Loads behavioral descriptions and construct internal representation according to the file format

`filename` File to be loaded

`format` Available formats are `vhdl` and `verilog`

`EXAMPLE` `avt_LoadFile model.v verilog`

---

avt_DriveBehavior

```
void avt_DriveBehavior(BehavioralFigure *befig, char *format)
```

Drives a behavioral description according to the file format from the given internal representation

`befig` Behavior to be driven

`format` Available formats are `vhdl` and `verilog`

`EXAMPLE` `avt_DriveBehavior $befig output.v verilog`

---

avt_LoadFile

```
void avt_LoadFile(char *filename, char *format)
```

Loads files and construct internal representation according to the file format

`filename` File to be loaded

`format` Available formats are `spice`, `tlf4`, `tlf3`, `lib`, `verilog`, `vhdl`, `spf`, `dspf`, `inf`, `spef` and `ttv`

EXAMPLE `avt_LoadFile design.hsp spice`

avt_EncryptSpice

`void avt_EncryptSpice(char *inputname, char *outputname)`

Encrypts all sections of a Spice file (netlist or technology file) which are encapsulated by the `.protect` *and* `.unprotect` *spice cards.*

`inputname` File to be encrypted

`outputname` Destination for encrypted output

EXAMPLE `avt_EncryptSpice techno.hsp techno.hsp.enc`

avt_SetCatalog

`void avt_SetCatalog(List *argv)`

Sets the leaves when flattening a netlist to catal level; equivalent to create a CATAL file

`argv` List of subcircuits that will be used as leaves

EXAMPLE `avt_SetCatalog [list "nand2" "inv" ]`

avt_GetCatalog

`StringList *avt_GetCatalog()`

Returns the current list of cells set as leaves for a catal-level flatten

EXAMPLE `set catal [avt_GetCatalog]`

avt_CheckTechno

`void avt_CheckTechno(char *label, char *tn, char *tp)`

Runs a set of benchs to findout possible technology errors

`label` A prefix label for the output result files

`tn` NMOS transistor characteristics. It's a space separated string with coming first the NMOS transistor name followed by the parameters. Authorized parameters are: l, w, delvt0, mulu0, sa, sb, sd, nf, nrs, nrd, sc, sca, scb, scc.

`tp`	same as `tn` for PMOS transistor.
`EXAMPLE`	avt_CheckTechno check1 "nmos l=0.4u w=0.8u" "pmos l=0.4u w=1.6u"

5.1.3. Netlist Modification

avt_GetNetlist

`Netlist *avt_GetNetlist(char *name)`

Retrieves a netlist from memory and returns its pointer

`name`	Name of the netlist to get in the program's memory
`EXAMPLE`	`set netlist [avt_GetNetlist "my_design"]`

avt_FlattenNetlist

`void avt_FlattenNetlist(Netlist *lf, char *level)`

Flattens a netlist to a given level.

`lf`	Pointer on the netlist to be flattened
`level`	Hierarchical level (coming from top-level) the nelist will be flattened to. Available levels are `trs`, `catal` or `bbox` (transistor, catalog or blackbox). If none of those levels are used, `level` will be considered an instance name, to which the netlist will be flattened.
`EXAMPLE`	`avt_FlattenNetlist $netlist trs`

avt_DriveNetlist

`void avt_DriveNetlist(Netlist *lf, char *filename, char *format)`

Saves the netlist on disk according to the given format

`lf`	Pointer on the netlist to be saved
`filename`	Name of the file to be created
`format`	Available formats are `spice`, `verilog`, `vhdl` and `spef`
`EXAMPLE`	`avt_DriveNetlist $netlist design.spi spice`

avt_DisplayNetlistHierarchy

```
void avt_DisplayNetlistHierarchy(FILE *f, char *netlistname, int
maxdepth)
```

Displays hierarchy information of a given netlist, and other info such as number of transistors

`f`	Pointer on the file where to save information, for standard output set `stdout`
`netlistname`	Pointer on the netlist
`maxdepth`	Maximum hierarchical depth coming from top level; can be set to 0 for infinite depth
EXAMPLE	`avt_DisplayNetlistHierarchy stdout "my_design" 3`

avt_DisplayResistivePath

```
void avt_DisplayResistivePath(FILE *f, Netlist *lf, char *connector1,
char *connector2)
```

Displays one resistive path between two connectors at the interface of a netlist.

`f`	Pointer on the file where to save information, for standard output set `stdout`
`lf`	Pointer on the netlist
`connector1`	first connector name
`connector2`	second connector name
EXAMPLE	`avt_DisplayResistivePath stdout [avt_GetNetlist "mynetlistname"] vdd_0 vdd_1`

avt_RemoveResistances

```
void avt_RemoveResistances(Netlist *lf, char *nameregex)
```

Removes all resistances on signals matching a regular expression

`lf`	Pointer on the netlist where to remove resistances
`nameregex`	Regular expression to be matched, for all signals use `*`

EXAMPLE	`avt_RemoveResistances $netlist "cpu.*.sig3*"`

avt_RemoveCapacitances

`void avt_RemoveCapacitances(Netlist *lf, char *nameregex)`

Removes all capacitances on signals matching a regular expression

`lf`	Pointer on the netlist where to remove capacitances
`nameregx`	Regular expression to be matched, for all signals use `*`
EXAMPLE	`avt_RemoveCapacitances $netlist "cpu.*.sig3*"`

5.1.4. Statistics

avt_StartWatch

`void avt_StartWatch(char *name)`

Starts a timer; if the timer already exixts it'll be reset to 0.

`name`	Timer name
EXAMPLE	`avt_StartWatch "CPU_TIME"`

avt_StopWatch

`void avt_StopWatch(char *name)`

Stops a timer; the timer must be started for the function to work

`name`	Name of the timer to stop
EXAMPLE	`avt_StopWatch "CPU_TIME"`

avt_PrintWatch

`char *avt_PrintWatch(char *name)`

Returns a string with the value of a timer; the timer must have been started

`name`	Name of the timer to print

EXAMPLE `avt_PrintWatch "CPU_TIME"`

avt_GetMemoryUsage

`long avt_GetMemoryUsage()`

Returns an integer with the memory usage of the program in bytes

EXAMPLE `set memory [avt_GetMemoryUsage]`

avt_RegexIsMatching

`int avt_RegexIsMatching(char *nametocheck, char *template)`

Returs 1 if `nametocheck` *matches the regular expression* `template`*, 0 otherwise.*

`nametocheck` name to check.

`template` regular expression to use.

EXAMPLE `set match [avt_RegexIsMatching tatoo5 *too*]`

5.2. Design Specific Configuration

5.2.1. General

inf_SetFigureName

`void inf_SetFigureName(char *name)`

Sets the target figure on which to apply the INF functions

`name` Name of the target figure

EXAMPLE `inf_SetFigureName cpu`

inf_AddFile

`void inf_AddFile(char *filename, char *figname)`

Loads an INF file and applies included statements on a figure (this function does not invoke `inf_SetFigureName`*).*

`filename`	INF file to load
`figname`	Figure on which to apply INF statements. Those statements will be added to the ones that my be already present.
EXAMPLE	`inf_AddFile cpu.inf cpu`

inf_Drive

`void inf_Drive(char *outputname)`

Saves applied INF statements on disk

`outputname`	File where to save INF statements (the `.inf` suffix is not automatically added)
EXAMPLE	`inf_Drive cpu.inf`

inf_ExportSections

`void inf_ExportSections(char *outputname, char *section)`

Saves on disk applied INF statements related to specific INF sections

`outputname`	File where to save INF statements
`section`	`OperatingCondition`, `PinSlew`, `Rename`, `Stop`, `Sensitive`, `Suppress`, `Inputs`, `NotLatch`, `CkLatch`, `Ckprech`, `Precharge`, `Dirout`, `Mutex`, `CrosstalkMutex`, `Constraint`, `ConnectorDirections`, `PathIN`, `PathOUT`, `PathDelayMargin`, `MulticyclePath`, `Ignore`, `NoCheck`, `Bypass`, `NoRising`, `NoFalling`, `Break`, `Inter`, `Asynchron`, `DoNotCross`, `Transparent`, `RC`, `NORC`, `SIGLIST`, `Falsepath`, `Delay`, `Dlatch`, `FlipFlop`, `Slopein`, `Capaout`, `OutputCapacitance`, `SwitchingProbability`, `Directives`, `Stb` and `Stuck`.
EXAMPLE	`inf_ExportSections cpu.inf "Dirout CrosstalkMutex"`

inf_CleanFigure

`void inf_CleanFigure()`

Removes all INF statements on current figure

5.2.2. Netlist

inf_DefineIgnore

```
void inf_DefineIgnore(char *type, List *list)
```

The tool ignores specified components. Equivalent to commenting out elements in a SPICE netlist.

`list` Pointer on the list of components to ignore. An component name can be a regular expression.

`type` Supported types are `Instances`, `Transistors`, `Resistances`, `Capacitances`, `Diodes`, `Parasitics` and `SignalNames`. `Parasitics` affects only DSPF files. `SignalNames` affects only the flattening of a hierarchical netlist, by ignoring the given name if several hierarchical names are available for one net.

EXAMPLE `inf_DefineIgnore Transistors *.M23*`

5.2.3. Disassembly

inf_DefineMutex

```
void inf_DefineMutex(char *type, List *list)
```

Adds mutual exclusion constraints on signals, in order to help the disassembly process. May be especially usefull when dealing with shifters or multiplexors, in case mutual exclusion constraints can not be directly derived from internal combinational circuitry (if the mutual exclusions constraints come from latched values or come from constraints on external pins).

`type` Mutual exclusion constraints, legal values for are `muxup`, `muxdn`, `cmpup` and `cmpdn` (see INF file description)

`list` List of signals mutual exclusions constraints should be applied on

EXAMPLE `inf_DefineMutex cmpup [list a_0 a_1 a_2 a_3]`

inf_DefineInputs

```
void inf_DefineInputs(char *name)
```

Sets a signal as a circuit input, in order to help the disassembly process.

`name`	Signal's name

inf_DefineDirout

`void inf_DefineDirout(char *name, int level)`

Defines the level of a signal for transistor orientation, in order to help the disassembly process.

`name`	Signal's name
`level`	Signal's level; transistors are oriented (the sense of the current is) from high-level to low-level signals.

inf_DefineDLatch

`void inf_DefineDLatch(char *name)`

Sets a signal as a dynamic latch. Works only if the surrounding circuitry permits a HZ state on the signal. Commands are then identified automatically.

`name`	Signal's name

inf_DefineNotDLatch

`void inf_DefineNotDLatch(char *name)`

Disables a dynamic latch directive on a signal. To be used together with `yagMarkTristateMemory`

`name`	Signal's name

inf_DefineNotLatch

`void inf_DefineNotLatch(char *name)`

Disables the identification of a latch on a signal

`name`	Signal's name

inf_DefineKeepTristateBehaviour

`void inf_DefineKeepTristateBehaviour(char *name)`

Disables the transformation of bus into register when configurations 'avtVerilogTristateIsMemory' or 'yagleTristateIsMemory' is used to drive a behavioural model.

`name` Signal's name

---

inf_DefinePrecharge

```
void inf_DefinePrecharge(char *name)
```

Sets a signal as a precharge.

`name` Signal's name

---

inf_DefineNotPrecharge

```
void inf_DefineNotPrecharge(char *name)
```

Disables the identification of a precharge on a signal

`name` Signal's name

---

inf_DefineModelLatchLoop

```
void inf_DefineModelLatchLoop(char *name)
```

Feedback loop is explicitly modeled in behavioural model if signal is a static latch.

`name` Signal's name

---

inf_DefineMemsym

```
void inf_DefineMemsym(char *name0, char *name1)
```

Sets a pair of signals to be a symmetric memory so long as there is a loop between the two signals.

`name0` name of first memsym signal.

`name1` name of second memsym signal.

`EXAMPLE` `inf_DefineMemsym memsym0 memsym1`

---

inf_DefineRS

```
void inf_DefineRS(char *name, char *type)
```

Allows control of how individual RS are handled. Overrides the global setting in yagAutomaticRSDetection.

`name` Signal's name, either the set or the reset one is enough.

`type` LEGAL, ILLEGAL or MARK_ONLY.

EXAMPLE `inf_DefineRS rsnode "LEGAL"`

---

inf_MarkSignal

```
void inf_MarkSignal(char *name, char *marks)
```

Allows application of special signal markings, such as latch identification.

`name` Signal's name

`marks` For a complete list of markings please refer to the INF section of this manual, MARKSIG subsection.

EXAMPLE `inf_MarkSignal dff_m "LATCH+MASTER"`

---

inf_MarkTransistor

```
void inf_MarkTransistor(char *name, char *marks)
```

Allows application of special transistor markings, such as latch commands identification.

`name` Signal's name

`marks` Legal markings are "Bleeder", "Feedback", "Command", "NonFunctional", "Blocker", "Short", "Unused". Types may be concatenated with the '+' character and are case-insensitive. For a description of the types please refer to the INF section of this manual, MARKTRANS subsection.

EXAMPLE `inf_MarkTrans m0 "FEEDBACK+NOT_FUNCTIONAL"`

---

inf_DefineSensitive

`void inf_DefineSensitive(char *name)`

Sets a signal as a timing sensitive net.

`name`	Signal's name

inf_DefineSuppress

`void inf_DefineSuppress(char *name)`

Sets an auxiliary signal to be suppressed.

`name`	Signal's name

5.3. Disassembling

5.3.1. yagle

void yagle(char *figname) yagle (<figname>) generates a RTL behavioral description from the transistor-level netlist given by <figname>. The netlist should exist in the program's memory (loaded with avt_LoadFile for example).

Chapter 6. Error Codes

6.1. API

API-001	Error executing <string>_AtLoad_Initialize(): <string>
API-002	Could not open dynamic library '<string>' reason: <string>
API-003	Cannot read file <string>
API-004	Internal error #<decimal>
API-005	<string>:<decimal>: '<string>' not defined
API-006	<string>:<decimal>: '<string>' already used, primary declaration line <decimal>, file <string>
API-007	<string>:<decimal>: Only int*, void*, double *, char* and FILE* are accepted
API-008	<string>:<decimal>: illegal format string '%%<character>'
API-009	<string>:<decimal>: not enough arguments for format
API-010	<string>:<decimal>: only const char* accepted
API-011	<string>:<decimal>: too many arguments for format
API-012	<string>:<decimal>: exclude can't be in a conditional block
API-013	<string>:<decimal>: '*' doesn't match with type
API-014	<string>:<decimal>: function type FILE* doesn't match
API-015	<string>:<decimal>: type of variable '<string>' doesn't match
API-016	<string>:<decimal>: '<string>' might be used uninitialized
API-017	<string>:<decimal>: digit '<decimal>' doesn't match
API-018	<string>:<decimal>: string '<string>' doesn't match
API-019	<string>:<decimal>: flow '<string>' doesn't match
API-020	<string>:<decimal>: undefined type '<string>'

API-021	<string>:<decimal>: sizeof(<string>) is unknown
API-022	<string>:<decimal>: assignment from <string><string> to <string><string> without a cast
API-023	<string>:<decimal>: incompatible type assignment <string><string> != <string><string>
API-024	<string>:<decimal>: variable '<string>' can not be indexed
API-025	<string>:<decimal>: forbiden operation on this variable type
API-026	<string>:<decimal>: '<string>' has an unexpected type
API-027	<string>:<decimal>: '<string>' should not be pointers
API-028	<string>:<decimal>: type of '<string>' and '<string>' mismatch
API-029	<string>:<decimal>: '<string>' and '<string>' should not be pointers
API-030	<string>:<decimal>: '<string>' has an unexpected type
API-031	<string>:<decimal>: unauthorized test on type '<string>'
API-032	<string>:<decimal>: interpreter can not cast <string> to <string>
API-033	<string>:<decimal>: too few arguments for function 'malloc'
API-034	<string>:<decimal>: too many arguments in function 'malloc'
API-035	<string>:<decimal>: too few arguments for function 'callfunc'
API-036	<string>:<decimal>: callfunc: only 'char *' pointer type can be used in function call
API-037	<string>:<decimal>: can not assign <string><string> to <string><string>
API-038	<string>:<decimal>: unknown variable '<string>'
API-039	<string>:<decimal>: unautorized operation on pointers
API-040	<string>:<decimal>: unautorized operation on type <string><string>
API-041	<string>:<decimal>: division by zero
API-042	<string>:<decimal>: unautorized operation on type '<string>'
API-043	<string>:<decimal>: can not make the requested dereference of '<string><string>'

API-044	<string>:<decimal>: can not make the dereference of '<string>'
API-045	<string>:<decimal>: Can not call functions with more then <decimal> arguments
API-046	<string>:<decimal>: function '<string>' can't be found in the dynamic libraries
API-047	<string>:<decimal>: <string>
API-048	error happens at <string>:<decimal>
API-049	<string>:<decimal>: variable '<string>' already declared in this scope
API-050	<string>:<decimal>: type '<string>' must be used as pointer
API-051	<string>:<decimal>: type of '<string>' must be 'FILE *'
API-052	<string>:<decimal>: Fatal error while executing program
API-053	<decimal> errors. Cannot execute
API-054	<string>:<decimal>: parameter '<string>' is uninitialised
API-055	<string>:<decimal>: conflicting type for parameter '<string>' : '<string><string>'!='<string><string>'
API-056	<string>:<decimal>: return value for void function
API-057	<string>:<decimal>: return value for void function
API-058	<string>:<decimal>: conflicting type for return value : '<string><string>'!='<string><string>'
API-059	Function '<string>' used in action was not found Location(s):
API-060	<string>:<decimal>: too many arguments in function '<string>'
API-061	<string>:<decimal>: too few arguments for function '<string>'
API-062	Somewhere: function '<string>' can't neither be found in interpreter nor in the dynamic libraries
API-063	Somewhere executing <string>
API-064	Error happens at somewhere when executing '<string>'
API-065	Error executing TCL function '<string>'

6.2. AVT

AVT-000	`Usage: avtdeltoken toolname servername hostname hostid hosttoken pid` When using the AVT license server (avtld), avtdeltoken allows to delete a token. Be careful, deleting a token under use will crash the corresponding process.
AVT-001	`Character out of [0-9A-Za-z] range in avertecbanner` An illegal value has been used within an internal function. Please, report internal errors to Avertec Support.
AVT-002	`Resulting size bigger than <decimal> columns not allowed in avertecbanner` The formatting size of an object or a text does not fit the avertec banner dimensions. Please, report internal errors to Avertec Support.
AVT-003	`'<string>' is not a valid variable` Invalid variable in avttools.conf file. Check variable name or suppress wrong variable definition.
AVT-004	`File avttools.conf, syntax error line <decimal>` File avttools.conf cannot be printed. Correct synthax error (see documentation)
AVT-005	`'<string> = <string>' with no effect. Value '<string>' already set with '<string>'` A variable has been set twice within avttools.conf, suppress one if possible.
AVT-006	`File avttools.conf, multiple declaration of '<string>'` A variable has been set several times. Conflicts may occur. Check the declarations of the variable within avttools.conf file and keep only one.
AVT-007	`Word <string> too long` The string provided to the license server is too long. Use avtinfo to ensure valid value or license file are provided.
AVT-008	`String too long` The line or message provided to the license server is too long.
AVT-009	`Impossible to delete this token` The token cannot be deleted. Check token name and user rights.
AVT-010	`Usage: avtgenkey toolname vendor server hostid date license type` The license server avt (avtld) failed to handle the license key of avtlicense file.
AVT-011	`Usage: avtinfo tool_name` avtinfo displays information for the avt license server (avt). Ensure a valid tool_name is used.

AVT-012	`Bad Format For Date` The date in the license file is not valid. The license file may be invalid. Check the license file in use.
AVT-013	`Bad Token: <string>` An invalid token has been requested by the license server. Check license file is valid and up to date.
AVT-014	`Option -<character> requires an operand` The specified option is not properly used. Check tool's usage.
AVT-015	`Unrecognised option: -<character>` The specified option is unknown. Check tool's usage.
AVT-016	`Unrecognized Command line` Command line synthax invalid. Check tool's usage.
AVT-017	`Error in opening file <string>` The specified file cannot be open. Check user's rights, and file path.
AVT-018	`Usage: avtreserve tool_name minutes [nb_token]` Display the usage of binary avtreserve. avtreserve is used by the avt license server (avtld).
AVT-019	`Unable to reserve token` Token reservation failed. Check token name and reservation command synthax.
AVT-020	`Usage: avttool tools_name_list` Display usage for monitoring tokens information.
AVT-021	`Run failed` The binary ends with an error. Other messages should bring more information.
AVT-022	`Impossible to give token` The license server was unable to get a pid for the token process.
AVT-023	`License server <string> not responding` The license server is unreachable. Check a valid license server is running.
AVT-024	`Environment variable <string> is not defined` A variable needs to be set in the environment. Avertec documentation supplies the user with the variables legal values.
AVT-025	`...try with server <string>` Another license server has been found and will be tryed. This can occur if the original license server specified by the user was not found or returned an error.

AVT-026	`Bad license server <string>` The license server is not valid or was not responding. Check a valid license server is running.
AVT-027	`<string>` The running tool encountered an error. Specific information supplied by the tool is displayed.
AVT-028	`Token never taken` No token information available for the license server. If a binary (avtinfo, avtdeltoken, or other avt license server utility) has been used, please check the relevant usage.
AVT-029	`Bad token description` The command line synthax used is not valid. Check for relevant avt license server utility usage.
AVT-030	`Bad file name, check autorisation for <string>` The utility failed to open the specified license key or log file.
AVT-031	`Missing part of log file` The avt license server returns an error with the license log file. Usually not a critical error for the running process if avtld is used rather than flexnet.

AVT-033	`AVT-_LICENSE_FILE not set and $AVT_TOOLS_DIR/etc/avtlicense.lic missing.`
	`If errors occur, please check your Flexlm license paths.`
	The flex license file was not found. Use the flexnet license utilities to get information about flexnet license status.

AVT-034	`Flexlm returned error '<decimal>' when <string>.` A generic message displaying error numbers returned by flexnet. Flexnet (or FlexLM) documentation provide some clue to handle surch error number.
AVT-035	`Unable to find Flexlm job corresponding to feature <string>` The specified feature is not valid or no job use it. Flexnet license utilities can provide information about the feature, and the jobs running.
AVT-036	`Please, resolve Flexlm errors before running Avertec programs.` Flexnet encounters an error. As long as Flexnet license will not runs properly no licensed Avertec tool can be run. Dealling with license errors is a priority. Check if license file path and license daemon are valid and reachable.
AVT-037	`Log level of variable '<string>' is not a number. It is '<string>'.` Please refer to the configuration of log report.
AVT-038	`Log level of variable '<string>' must be a positive number less or equal to 9. It is '<string>'.`

	Please refer to the configuration of log report.
AVT-039	Log variable '<string>' is unknown. Please refer to the configuration of log report.
AVT-040	Invalid value '<string>' for configuration variable '<string>', should be 'yes' or 'no' Please refer to the HiTas Reference Guide.
AVT-041	Invalid value '<string>' for configuration variable '<string>', please refer to the Reference Guide for correct values Please refer to the HiTas Reference Guide.
AVT-042	Invalid file format '<string>' for '<string>', please refer to the Reference Guide for valid formats Valid formats are 'spice', 'lib', 'tlf3', 'tlf4', 'ttv', 'vhdl', 'verilog', 'dspf', 'spef' and 'inf'.
AVT-043	Could not find a file matching '<string>' The specified filename filter does not match any files in accessible directories.
AVT-044	Multiple settings for configuration variable '<string>': '<string>' is overwritten by '<string>'
AVT-045	could not open file '<string>'
AVT-046	could not create file '<string>'
AVT-047	'avtWarningFilter' has been set to '<string>'
AVT-048	Subsequent call to 'avt_LoadFile' will not retain previous definitions of global parameters The scope of .SCALE, .GLOBAL or other global parameters is limited to the files loaded by a single call of 'avt_LoadFile'. The values of those global parameters are not retained for subsequent calls of 'avt_LoadFile'.
AVT-049	Incorrect unit specified in '<string>'
AVT-050	Incorrect number of jobs <string>
AVT-051	received answer from job <decimal>, but is not running !
AVT-052	job <decimal> is not in state STAT_EXEC_WAIT when received begin packet
AVT-053	job <decimal> is not in state STAT_EXEC_RUN when received data
AVT-054	job <decimal> is not in state STAT_EXEC_RUN when received end
AVT-055	Job <decimal> aborted [<string>]

AVT-056	A system error occured when runing a new job
AVT-057	Statistical result file '<string>' already exist. Using '<string>'.
AVT-058	Error in <string> table. Entry <decimal> correspond to index <decimal>.
AVT-059	Received unhandled command '<string>' from job <decimal>
AVT-061	Invalid binary transfer
AVT-062	Failed to create directory '<string>'
AVT-063	Invalid value '<string>' for configuration variable '<string>' Valid value are '1mW', '100uW', '10uW', '1uW', '100nW', '10nW', '1nW', '100pW', '10pW' and '1pW'.

6.3. BEF

BEF-000	Behavior out format <string> is not a legal format! A Verilog file will be dumped. To describe the format for the generated behavior, you must set 'avtOutputBehaviorFormat' either to 'vhd' for VHDL or to 'vlg' for Verilog.\n
BEF-001	Behavior in format <string> is not a legal format! To describe the format of input behavior, you must set 'avtInputBehaviorFormat' either to 'vhd' for VHDL or to 'vlg' for Verilog.\n

6.4. BEG

BEG-001	<string> Attempt to merge bit vector and single bit <string>
BEG-002	<string> Internal error <string>
BEG-003	<string> Connector declared in, used as out <string>
BEG-004	<string> Null condition <string>
BEG-005	<string> Conflicting vector <string>
BEG-006	<string> Expression and variable has different size <string>
BEG-007	<string> Parser Failure <string>
BEG-008	<string> Conflicting declaration and use <string>

BEG-009	<string> Conflicting bus use <string>
BEG-010	<string> Direction of declaration conflicts with use <string>
BEG-011	<string> Selected signal, vector expression not allowed <string>
BEG-012	<string> Attempt to insert a signal into a defined signal <string>
BEG-013	<string> Vector incompletely defined, made external <string>
BEG-014	<string> Connector declared out, used as in <string>
BEG-015	<string> Bad value for 'BEG_USER_WAY', accepted 'to' or 'downto' <string>
BEG-016	Unknown
BEG-017	<string> Trace: <string>
BEG-018	<string> Convert an In to Out <string>

6.5. BEH

BEH-000	syntax error
BEH-001	combinatory loop: `<string>`
BEH-002	cannot make bdd of empty expression
BEH-003	cannot find terminal `<string>`
BEH-004	illegal use of STABLE attribute
BEH-005	cannot simplify internal signals
BEH-006	cannot make derivatives of expressions
BEH-040	signal `<string>` never assigned
BEH-041	`<string>` has not an empty architecture
BEH-068	port `<string>` has unknwon type
BEH-069	port `<string>` has unknwon mode
BEH-070	unknown time unit
BEH-100	cannot find `<string>`

BEH-107	cannot open result file
BEH-101	<string>: unknown operator
BEH-102	<string>: cannot create empty atom
BEH-103	<string>: cannot build NOT of empty expression
BEH-104	<string>: cannot combine empty expressions
BEH-105	<string>: cannot find terminal
BEH-110	<string>: decompiler called on empty figure
BEH-115	<string>: illegal bit string value : `<character>`
BEH-116	<string>: the same expression cannot be used twice
BEH-119	<string>: empty guard expression: `<string>`
BEH-120	<string>: empty waveform expression: `<string>`
BEH-200	<string>: illegal use of attribute STABLE
BEH-201	<string>: unknown terminal operand `<string>`
BEH-202	<string>: unknown operator `<decimal>`
BEH-203	<string>: empty expression
BEH-199	<string>: Please contact Avertec support
BEH-300	beaux `<string>` not empty
BEH-301	bebus `<string>` not empty
BEH-302	bebux `<string>` not empty
BEH-303	beder not empty
BEH-304	befig `<string>` not empty
BEH-305	begen `<string>` not empty
BEH-306	bemsg `<string>` not empty
BEH-307	beout `<string>` not empty
BEH-308	bequad not empty

BEH-309	bereg `<string>` not empty
BEH-310	biabl not empty
BEH-311	binode not empty
BEH-312	%20s -> <string>
BEH-313	%23s <string>
BEH-315	bevectaux `<string>` not empty
BEH-316	bevectout `<string>` not empty
BEH-317	bevectbux `<string>` not empty
BEH-318	bevectbus `<string>` not empty
BEH-319	bevectreg `<string>` not empty
BEH-320	vectbiabl not empty

6.6. BHL

BHL-000	Internal error <string>

6.7. BGL

BGL-000	Internal error <string>
BGL-001	`<string>` line <decimal> :`<string>` is incompatible with the entity name
BGL-002	`<string>` line <decimal> :bad entity declaration
BGL-003	`<string>` line <decimal> :bad port clause declaration
BGL-004	`<string>` line <decimal> :port `<string>` already declared
BGL-005	`<string>` line <decimal> :illegal port declaration `<string>` (mode, type, guard mark)
BGL-006	`<string>` line <decimal> :bad port declaration
BGL-007	`<string>` line <decimal> :`<string>` is incompatible with the architecture name

BGL-008	`<string>` line <decimal> :bad architecture declaration
BGL-009	`<string>` line <decimal> :illegal declaration
BGL-010	`<string>` line <decimal> :signal `<string>` already declared
BGL-011	`<string>` line <decimal> :illegal signal declaration `<string>` (type, guard mark)
BGL-012	`<string>` line <decimal> :component `<string>` already declared
BGL-013	`<string>` line <decimal> :instance `<string>` already declared
BGL-014	`<string>` line <decimal> :`<string>` unknown component
BGL-015	`<string>` line <decimal> :illegal usage of implicit port map description
BGL-016	`<string>` line <decimal> :`<string>` unknown local port
BGL-017	`<string>` line <decimal> :`<string>` unknown port or signal
BGL-018	`<string>` line <decimal> :illegal concurrent statement
BGL-019	`<string>` line <decimal> :bad signal association
BGL-020	`<string>` line <decimal> :null array not supported
BGL-021	`<string>` line <decimal> :illegal constraint in declaration of type
BGL-022	`<string>` line <decimal> :signal `<string>` used out of declared range
BGL-023	`<string>` line <decimal> :width or/and type mismatch
BGL-024	`<string>` line <decimal> :port `<string>` connected to more than one signal
BGL-025	`<string>` line <decimal> :can only assign to/from an external connector
BGL-026	`<string>` line <decimal> :instance <string> mismatch with the model
BGL-027	`<string>` line <decimal> :Unhandled feature
BGL-028	`<string>` line <decimal> :<string>
BGL-029	Cannot open result file
BGL-030	Cannot continue further more.

BGL-031	`<string>` line <decimal> :Syntax error
BGL-032	Too many errors. Cannot continue further more
BGL-033	File does not exist : <string>
BGL-034	Abnormal parsing for : <string>
BGL-035	Connection missing on port `<string>`
BGL-036	Consistency checks will be disabled
BGL-038	Internal error <string> while executing <string>
BGL-037	`<string>` line <decimal> :<string>

6.8. BVL

BVL-000	Internal error <string>
BVL-001	`<string>` line <decimal> `<string>` is incompatible with the entity name
BVL-002	`<string>` line <decimal> Bad entity declaration
BVL-003	`<string>` line <decimal> Bad port clause declaration
BVL-004	`<string>` line <decimal> Port `<string>` already declared
BVL-005	`<string>` line <decimal> Illegal port declaration (mode, type, kind)
BVL-006	`<string>` line <decimal> Bad port declaration
BVL-007	`<string>` line <decimal> `<string>` is incompatible with the architecture name
BVL-008	`<string>` line <decimal> Bad architecture declaration
BVL-009	`<string>` line <decimal> Illegal declaration
BVL-010	`<string>` line <decimal> Signal `<string>` already declared
BVL-011	`<string>` line <decimal> Illegal signal declaration (type, kind)
BVL-012	`<string>` line <decimal> `<string>` unknown port or signal
BVL-013	`<string>` line <decimal> Illegal concurrent statement

BVL-014	`<string>` line <decimal> Label `<string>` already declared
BVL-015	`<string>` line <decimal> `<string>` is incompatible with the block's label
BVL-016	`<string>` line <decimal> Input port `<string>` cannot be assigned
BVL-017	`<string>` line <decimal> Illegal unguarded signal assignment for `<string>`
BVL-018	`<string>` line <decimal> Illegal guarded signal assignment `<string>`
BVL-019	`<string>` line <decimal> Some choices missing in the selected signal assignment
BVL-020	`<string>` line <decimal> Output port `<string>` cannot be read
BVL-021	`<string>` line <decimal> Duplicate choice in selected signal assignment
BVL-022	`<string>` line <decimal> Illegal use of OTHERS in selected signal assignment
BVL-023	`<string>` line <decimal> Null array not supported
BVL-024	`<string>` line <decimal> Incompatible constraint and type
BVL-025	`<string>` line <decimal> Illegal assignment of `<string>` (widths mismatch)
BVL-026	`<string>` line <decimal> Signal `<string>` used out of declared range
BVL-027	`<string>` line <decimal> Width or/and type mismatch
BVL-028	`<string>` line <decimal> Signal `<string>` assigned more than once
BVL-029	`<string>` line <decimal> Signal `<string>` never assigned
BVL-030	`<string>` line <decimal> Illegal condition on signal `<string>`
BVL-031	`<string>` line <decimal> BEPOR type is unknown
BVL-032	`<string>` line <decimal> `<string>` is not a bit string litteral
BVL-033	`<string>` line <decimal> Bad generic declaration
BVL-034	`<string>` line <decimal> Bad generic element

BVL-035	`<string>` line <decimal> `<string>`: when expression must be a constant
BVL-036	`<string>` line <decimal> Illegal generic declaration (type, kind)
BVL-037	`<string>` line <decimal> Illegal constant declaration (type, kind)
BVL-038	`<string>` line <decimal> Illegal use of attribute STABLE on `<string>`
BVL-039	`<string>` line <decimal> Different delays not supported on waveforms
BVL-040	`<string>` line <decimal> Syntax error
BVL-041	Too many errors. Cannot continue further more
BVL-042	`<string>` Error line <decimal> : <string>

6.9. CBH

CBH-000	Internal Error <string>
CBH-001	<string>: Possible cause library not charged
CBH-002	<string> needs a file as argument
CBH-003	<string>

6.10. CGV

CGV-001	Internal error #<decimal>
CGV-002	Internal warning #<decimal>
CGV-003	<string>:%ld: unknown internal error <decimal>
CGV-004	could not open file '<string>'
CGV-005	could not parse file '<string>'

6.11. CNS

Error messages description not available yet.

6.12. GNS

GNS-001	instance <string> already exist in figure <string>
GNS-002	instance model is the figure <string> itself
GNS-003	connector number discrepancy between figure <string> and instance <string> in figure <string>
GNS-004	Internal error <decimal>
GNS-005	Internal warning <decimal>
GNS-006	<string>: can't find transistor '<string>' in model
GNS-007	Invalid symmetry detected for instance '<string>' (<string>) on connector '<string>' and ? (<string> and <string>)
GNS-008	no FCL match for `<string>' (instance:'<string>' model:`<string>')
GNS-009	Vector ordering failed on instance '<string>' (<string>) who might be used as \"exclude\" . Try to add 'NoOrdering' in the 'GnsFlags' variable
GNS-010	Found an alim linked to a 'not' alim : <string> and <string> in <string>
GNS-011	can't retrieve blackbox instance `<string>' (`<string>')
GNS-012	can't retrieve blackbox connector `<string>.<string>' (`<string>')
GNS-013	can not find generic variable '<string>', assumed value 0
GNS-014	could not find black box '<string>' in circuit
GNS-015	single connector '<string>' (instance '<string>') is linked to a vector connector <string>
GNS-016	vector connector '<string>' (instance '<string>') is linked to a single connector <string>
GNS-017	could not find correspondance for transistor '<string>'
GNS-018	found a transistor with no name
GNS-019	could not find correspondance for signal '<string>'
GNS-020	can't drive '<string>' type in function call
GNS-021	can't drive pointer type in function call

GNS-022	can't find figure '<string>'
GNS-023	Could not write file <string>.gns
GNS-024	Cannot create file <string>.gen
GNS-025	Can't redirect GENIUS output to '/dev/null'
GNS-026	<string>:<decimal>: division by zero
GNS-027	forbidden operators 'mod', 'rem', '**'
GNS-028	variable <string> not found
GNS-029	<string>:<decimal>: IF forbidden for GNS
GNS-030	a variable name was expected for instance '<string>', found a number
GNS-031	generic variable '<string>' not define
GNS-032	Value of '<string>' for instance '<string>' must be <decimal>, actually %ld
GNS-033	There should be at least one instance of model '<string>' with <string>=<decimal>
GNS-034	More than 1 unknown generic variable
GNS-035	variable '<string>' is not defined yet
GNS-036	<string>: can't go thru '<string>'
GNS-037	<string>: can't find instance '<string>' in model
GNS-038	no corresponding transistor for <string>
GNS-039	*** <decimal> error(s) detected, I can't get farther!! ***
GNS-040	for model instance '<string>' can not evaluate left or right bound for connector '<string>' l=<decimal> r=<decimal>
GNS-041	no correspondance found for signal '<string>(<decimal>)
GNS-042	Error: <string>
GNS-043	No model file in library
GNS-044	Cannot open model file <string>

GNS-045	no model <string> found
GNS-046	unknown connector (<string>) declared in symmetric connector list
GNS-047	invalid mix of vector and bit
GNS-048	unknown connector (<string>) declared in coupled connector list
GNS-049	Could not find subfigure '<string>' in file '<string>'
GNS-050	Spice file <string> should be a flat transistor netlist
GNS-060	other errors follow...
GNS-061	in model '<string>', connector '<string>' of unexistant instance '<string>' must not be linked the model interface
GNS-062	in model '<string>', if connector '<string>' of unexistant instance '<string>' is not used, it must be linked to a supply
GNS-063	in model '<string>', connector '<string>' of unexistant instance '<string>' must not be linked to another unexistant instance connector
GNS-064	Inconsistancies found for instance '<string>' of model '<string>'
GNS-065	Connector '<string>' of instance mismatched with connector '<string>' of model
GNS-066	Inconsistancies found for connector '<string>' of instance '<string>' with model '<string>'
GNS-067	Connector number mismatched
GNS-068	While parsing correspondance tables, line <decimal>, transistor out of context
GNS-069	While parsing correspondance tables, line <decimal>, could not find transistor '<string>' in original netlist
GNS-070	While parsing correspondance tables, line <decimal>, signal out of context
GNS-071	While parsing correspondance tables, line <decimal>, could not find signal '<string>' in original netlist
GNS-072	While parsing correspondance tables, line <decimal>, instance out of context
GNS-073	While parsing correspondance tables, line <decimal>, could not find instance correspondance '<string>' table

GNS-074	While parsing correspondance tables, line <decimal>, could not find instance correspondance table
GNS-075	While parsing correspondance tables, line <decimal>, variables out of context
GNS-076	While parsing correspondance tables, line <decimal>, dictionary entry without dictionary mode
GNS-077	While parsing correspondance tables, line <decimal>, too many entries in dictionary
GNS-078	While parsing correspondance tables, line <decimal> ignored
GNS-079	Could not find instance in model to start search with There should be at least one real instance in the model
GNS-080	could not find instance '<string>' in GNS toplevel instance
GNS-081	out of bounds with <string> and <string> started from <string>
GNS-082	computing error for index=%ld end=<decimal>
GNS-083	parameter discrepancy between <string> and <string>
GNS-084	infinite loop on <string>
GNS-085	transistor in loop is forbidden
GNS-086	<string>:<decimal>: GNS ignored expansed 'FOR' driven by variable '<string>'
GNS-087	several signals connected to connector <string> of instance '<string>'
GNS-088	vector connector <string> connected to single signal <string>
GNS-089	connector '<string>' with several signals
GNS-090	can not compute destination connector index for '<string>', the high bound of signal '<string>' is not known yet
GNS-091	vector connector '<string>' connected to one bit signal
GNS-092	bit number <decimal> is out of bounds for signal <string>
GNS-093	bit number <decimal> is out of bounds for connector <string>
GNS-094	transistor type/parameter discrepancy (<string>)

GNS-095	<string>:<decimal>: too many parameters for transistor '<string>'
GNS-096	<string>:<decimal>: a number was expected for instance '<string>', found a variable name
GNS-097	<string>:<decimal>: A positive non nul number was expected for instance '<string>'
GNS-098	<string>:<decimal>: unknown transistor parameter '<string>'
GNS-099	instance type/parameter discrepancy (<string>)
GNS-100	too many connectors in instance '<string>'
GNS-101	connectors <string> and <string> mismatch for instances <string> and <string>
GNS-102	not enough connectors in instance '<string>'
GNS-103	parameter discrepancy between instances <string> and <string>
GNS-104	<string>:<decimal>: a variable name was expected for instance '<string>', found a number
GNS-105	could not find generic variable '<string>' in entity variable list
GNS-106	several signals connected to connector <string> of instance <string>
GNS-107	no search done on connector '<string>' signal '<string>', model must be a connexe graph
GNS-108	no search done on signal '<string>(%ld)', model must be a connexe graph
GNS-109	can't compute connector bound for connector '<string>' certainly while building a fake instance or transistor
GNS-110	width mismatch between connector '<string>(<decimal>..<decimal>)' and signal '<string>(<decimal>..<decimal>)'
GNS-111	Index <decimal> is out of range for signal <string> (<decimal>..<decimal>)
GNS-112	Index <decimal> is out of range for signal <string>
GNS-113	<string>:<decimal>: negative vector bound computed for expression, values are <decimal> and <decimal>
GNS-114	no search done on connector '<string>(%ld)', model '<string>' must be a connexe graph

GNS-115	connector <string>.<string> is in coupled list but has no symetric
GNS-116	no symmetry found for connector <string> in coupled connector list
GNS-117	coupling won't work with vectors ... yet...
GNS-118	can not find coupled connector for '<string>'
GNS-119	could not find connector '<string>(<decimal>)' for instance '<string>'
GNS-120	while swapping <string> and <string>, one of the connector did not have coupled connector list while the other has
GNS-121	Internal limitation. too much symmetric informations. Actual limit is <decimal>
GNS-122	same signal in different symmetry list
GNS-123	same signal in different coupled list
GNS-124	<string>:<decimal>: array of signal '<string>' out of bounds with model line <decimal>
GNS-125	<string>:<decimal>: array doesn't match for '<string>' (line model <decimal>)
GNS-126	<string>:<decimal>: connector '<string>' is missing in left side of instance
GNS-127	<string>:<decimal>: too many connections in Port Map. Component line <decimal>
GNS-128	<string>:<decimal>: not enough connections in Port Map. Component line <decimal>
GNS-129	<string>:<decimal>: <string> already excluded
GNS-130	<string>:<decimal>: instance <string> doesn't exist in architecture <string> of <string>
GNS-131	<string>:<decimal>: INPUT '<string>' cannot be connected with OUTPUT '<string>'
GNS-132	<string>:<decimal>: OUTPUT '<string>' cannot be connected with INPUT '<string>'
GNS-133	<string>:<decimal>: OUTPUT '<string>' cannot be connected with INPUT '<string>'

GNS-134	<string>:<decimal>: only one variable authorized in a 'for' expression. Use Hierarchy!
GNS-135	<string>:<decimal>: forbidden operator '<string>' on variable
GNS-136	<string>:<decimal>: division by zero could appear
GNS-137	<string>:<decimal>: a generic isn't needed by model line <decimal>
GNS-138	<string>:<decimal>: a generic is needed by model line <decimal>
GNS-139	<string>:<decimal>: not enough variables in component. model ends at line <decimal> with '<string>'
GNS-140	<string>:<decimal>: type '<string>' doesn't match with model line <decimal>
GNS-141	<string>:<decimal>: too many variables in component. model ends at line <decimal> with '<string>'
GNS-142	<string>:<decimal>: a port isn't needed by model line <decimal>
GNS-143	<string>:<decimal>: a port is needed by model line <decimal>
GNS-144	<string>:<decimal>: a bit is expected for '<string>' line <decimal> of model
GNS-145	<string>:<decimal>: size of '<string>' mismatches with model line <decimal>
GNS-146	<string>:<decimal>: a vector is expected for '<string>' line <decimal> of model
GNS-147	<string>:<decimal>: predefined rule '<string>': IN Grid, INOUT Source, INOUT Drain, IN Bulk
GNS-148	<string>:<decimal>: There is no vector in predefined rule '<string>'
GNS-149	<string>:<decimal>: The predefined entity '<string>' don't have a generic
GNS-150	<string>:<decimal>: '<string>' must be an external connector
GNS-151	<string>:<decimal>: predefined rule '<string>' impossible to use as a transistor name as an entity name
GNS-152	<string>:<decimal>: more than one action defined for entity '<string>'
GNS-153	<string>:<decimal>: no architecture defined for entity '<string>'

GNS-154	<string>:<decimal>: too many (><decimal>) transistors in architecture '<string>' to start FCL
GNS-155	<string>:<decimal>: Generate forbidden with transistor in model <string>
GNS-156	<string>:<decimal>: Blackbox with generic variables not implemented yet
GNS-157	<string>:<decimal>: GNS can't start on a pure transistor netlist '<string>'
GNS-158	you must explicitly specify the vector range for '<string>'
GNS-159	<string>:<decimal>: negative vector index
GNS-160	<string>:<decimal>: vector direction error
GNS-161	no figure model <string> found
GNS-162	<string>:<decimal>: <string> instance '<string>' can't be found
GNS-163	<string>:<decimal>: '<string>' used several times, primary use line <decimal> in file <string>
GNS-164	<string>:<decimal>: variable '<string>' not defined
GNS-165	<string>:<decimal>: <string>
GNS-166	Cannot read file <string>
GNS-167	Error executing TCL funcion '<string>': <string>
GNS-168	internal: unknown transistor model '<string>'

6.13. INF

INF-001	<string>L and W not required The parameters L and W won't be use because they have no meaning in this declaration.
INF-002	<string>unrecognized token '<string>', should be <string> The expected token was not found.
INF-003	<string>unrecognized token '<string>' The token is unknown and can't be handled.
INF-004	<string>invalid value '<string>', should be <string>

An invalid value has been encountred.

INF-005	`<string>unknown section '<string>', ignoring section` The section is unknown and can't be handled.
INF-006	`<string>unknown direction '<string>', ignoring section` The connector direction is unknown and can't be handled.
INF-007	`<string>clock '<string>' is not delared yet` A clock signal is referenced but has not been declared yet.
INF-008	`<string>no period specified for clock '<string>' or no default period` It's impossible to find a period to associate with the clock. Either the default period or the clock period should be defined.
INF-009	`<string>:<decimal>: syntax error near '<string>'` A syntax error occured when parsing file. Either a signal name matchs a syntax token or the token is unknown. In the first case, consider enclosing the signal name with quote.
INF-010	`<string>:<decimal>: syntax error in regular expression` An invalid regular expression have been detected.
INF-011	`no figure name given, guessing it is '<string>'` The information file should begin with the figure name associated with the informations. If the name is not specified, a figure name will be guessed from the file name. This could lead to errors if the guessed name is wrong.
INF-012	`can not open '<string>' information file` An error occured when trying to read the information file.
INF-013	`<string>information on signal <string> already read -- ignored`
INF-014	`<string>information on signal '<string>' has already been set elsewhere -- overriding with inf values` A default value has been set by the netlist spice deck and will be shadowed by a new value who has more priority.
INF-015	`<string>information '<string>' for '<string>' has already been set to '<string>' elsewhere -- overriding with inf value '<string>'` A default value has been set by the netlist spice deck and will be shadowed by a new value who has more priority.
INF-016	`<string>information '<string>' for '<string>' has already been set elsewhere -- overriding with inf values` A default value has been set by the netlist spice deck and will be shadowed by a new value who has more priority.

INF-017	<string>information '<string>' for '<string>' has already been set to <decimal> elsewhere -- overriding with inf value <decimal> A default value has been set by the netlist spice deck and will be shadowed by a new value who has more priority.
INF-018	could not create file '<string>' An error occured when trying the open the information file in write mode.
INF-019	<string>slope defined for pin '<string>' <string> is too low, set to <string>
INF-020	<string>invalid check type ('<string>') for 'NoCheck' section
INF-021	unknown inf section '<string>'
INF-022	<string>unknown characteristic '<string>', ignoring this entry The signal characteristic is unknown and can't be handled.
INF-023	name '<string>' doesn't match any <string> in circuit
INF-024	<string>incompatible tokens given for 'directives' section Using "... with sig rising|falling" or "... before|after sig up|down" is forbidden.
INF-025	<string>invalid type '<string>', ignoring this entry The given type does not apply to this section and can't be handled.
INF-026	<string>out of range probability value: %g The probability value must be a value ranging from 0 to 1.
INF-027	invalid marking <string>

6.14. LOG

LOG-000	Internal Error <string>
LOG-001	Maximum number of variables for BDDs reached
LOG-002	BDD's system not enough memory...

6.15. MBK

MBK-000	connector <string> not found in instance <string>
MBK-001	connector number mismatch beetwen instance <string> and figure <string>

MBK-002	can't evaluate '<string>', assuming 0
MBK-003	transistor length is null
MBK-004	transistor width is null
MBK-005	conflicting power supply on node '<string>' keeping %gv (other is %gv)
MBK-006	Can't flatten figure <string> because RC cache is active
MBK-007	lofigchain is missing on lofig <string>
MBK-008	Null CTC on signal %ld (1) in instance <string>
MBK-009	flattenlofig: connector <string> exists only in instance <string>
MBK-010	flattenlofig: connector <string> in instance <string>: number of physical nodes differ
MBK-011	Null CTC on signal %ld (2) in instance <string>
MBK-012	figure <string> not empty (type=%ld)
MBK-013	unflat error: no supply ground
MBK-014	unflattenlofig: connector number inconsistency between model '<string>' and instance '<string>'
MBK-015	duplosig impossible: signal %ld already exist
MBK-016	the radical <string> is already used in a vector
MBK-017	the radical <string> has a spurious vectorized value <string> (<string>)
MBK-018	figure '<string>': transistor '<string>' appears several times
MBK-019	can't open file <string>
MBK-020	addlofig impossible: figure <string> already exists
MBK-021	addlomodel impossible: model <string> already exists
MBK-022	illegal transistor type: %ld
MBK-023	addloins impossible: instance <string> already exist in figure <string>
MBK-024	addloins impossible: instance model is the figure <string> itself

MBK-025	addloins impossible: connector number discrepancy between figure <string> and instance <string> in figure <string>
MBK-026	addlocon impossible: connector <string> already exists in figure <string>
MBK-027	addlocon impossible: bad direction <character> in figure <string>
MBK-028	addlosig impossible: signal %ld already exist in figure <string>
MBK-029	getloins impossible: instance <string> doesn't exist in figure <string>
MBK-030	getlotrs impossible: transistor <string> doesn't exist in figure <string>
MBK-031	getlocon impossible: connector <string> doesn't exist in figure <string>
MBK-032	getlosig impossible: signal %ld doesn't exist in figure <string>
MBK-033	viewlo: empty list of figure
MBK-034	setsigsize() impossible: BKSIG not NULL
MBK-035	Conflict power supply on signal: <string>
MBK-036	u is not a square matrix
MBK-037	u and l matrix size mismatch
MBK-038	could not find a pivot
MBK-039	singular matrix given
MBK-040	matrix a and b size can not allow '*' operation
MBK-041	matrix a and b size can not allow '-' operation
MBK-042	matrix solve order has not been computed yet
MBK-043	matrix a and sol size can not allow solve operation
MBK-044	unreducable matrix given
MBK-045	fatal mbkalloc error: not enough memory when trying to allocate %lu bytes, top= %luKb
MBK-046	fatal mbkrealloc error: not enough memory
MBK-047	Can't open file <string> because too big

MBK-048	mbksysfopen: bad value for access
MBK-049	file <string> opened, file <string> ignored
MBK-050	file <string> opened for writting, file <string> is deleted
MBK-051	Cannot evaluate parameter <string>='<string>'<string>
MBK-052	Cannot evaluate expression '<string>=<string>' in figure '<string>': variable<string> <string> <string> unknown, assuming 0
MBK-053	Undefined <string> parameter '<string>' in subcircuit '<string>'
MBK-054	Error #<decimal> in avt communication protocol (<string>)
MBK-055	IP port <decimal> already in use. Waiting...
MBK-056	Can't activate master process in a slave process
MBK-057	not enought communication slot to handle new input connection !
MBK-058	Non slot to handle new input connection !
MBK-060	some communication slot are still active, but all process are finished !
MBK-061	a lofig is not allowed in the slave process !
MBK-062	a cnsfig is not allowed in the slave process !
MBK-063	a ttvfig is not allowed in the slave process !
MBK-064	a abnormal end-of-file was encountered !
MBK-065	error '<string>' while reading file '<string>'
MBK-066	error '<string>' while writing to file '<string>'
MBK-067	missing end of encryption marker in file '<string>'
MBK-068	Failed to transfer file '<string>'
MBK-069	Environment variable '<string>' is not set
MBK-070	Expression '<string>' returned negative value for <string>
MBK-071	Diode model evaluation returned <string> capacitance value

MBK-072	Resistive paths found between power supplies. If resistor names do not appear, consider using 'avt_config avtSpiKeepNames resistance'.

6.16. MCC

MCC-000	Unknown transition in mcc_generatesimgate
MCC-001	Default values assumed for vbs
MCC-002	Can't generate params for <string> (L = %ldn, W = %ldn)
MCC-003	Can't find vds for a degraded transistor
MCC-004	Hspice bsim3v3 model used with invalid ACM value (<decimal>)
MCC-005	Level = 53, default ACM=10 used!!!
MCC-006	Level = 49, default ACM=0 used!!!
MCC-007	Invalid value of parameter MOBMOD, default value used!
MCC-008	Computation of Vdsat failed! Default computation of Vdsat used!
MCC-009	Negative Leff for <string> (L = %ldn, W = %ldn)! default value assumed: Leff = %ldn
MCC-010	Negative Weff for <string> (L = %ldn, W = %ldn)! default value assumed : Weff = %ldn
MCC-011	Computation of Vdsat failed! Bad value for RDSMOD = 0 assumed
MCC-012	Technofile <string> to get doesn't exist!
MCC-013	Technofile <string> to delete doesn't exist
MCC-014	No model <string> in technofile <string>, can't get index!
MCC-015	No model <string> in technofile <string>, can't get model name!
MCC-016	No model <string> in technofile <string>, can't get XL!
MCC-017	No model <string> in technofile <string>, can't get XW!
MCC-018	Can't initialise diode model '<string>' parameters, unknown diode's level: <decimal>
MCC-019	Computation of VDDdeg for model <string> which is a PMOS transistor!

MCC-020	Issue occured while computing VDDdeg for model <string> (L=%g,W=%g), default value 'VDD (%g) - VTH (%g)' assumed (%g)
MCC-021	Computation of VSSdeg for model <string> which is a PMOS transistor!
MCC-022	Issue occured while computing VSSdeg for model <string> (L=%g,W=%g), default value 'VTH' assumed (%g)
MCC-023	mcc_dio_calcCDEP invalid value for TLEV! TLEV = 0 assumed
MCC-024	Issue occured while computing VTI_nmos
MCC-025	Issue occured while computing VTi_pmos
MCC-026	Technofile <string> doesn't exist, can't addmodel!
MCC-027	Model <string> doesn't exist!
MCC-028	No model <string> (<string> case L = %ldn, W = %ldn) for <string> The technology file does not contain the model or the device size is out of the model ranges. It may also be necessary to load the techonology file before using CPE.
MCC-029	No model for diode model <string>!
MCC-030	Closest model assumed <string> (lmin = %ldn, lmax = %ldn, wmin = %ldn, wmax = %ldn)
MCC-031	Positive vbs (%g) for transistor <string> (model <string> L=%gu W=%gu)
MCC-032	Negative vbs (%g) for transistor <string> (model <string> L=%gu W=%gu)
MCC-033	<string> is used for best corner!
MCC-034	<string> is used for worst corner!
MCC-035	Could not characterize transistor model '<string>' with given PVT: vdd=%gv temp=%g Check if the PVT suits your transistor model
MCC-036	Unknown transistor instance specific parameter '<string>'
MCC-037	Failed to evaluate <string>=<string>
MCC-038	Cannot evaluate model parameter <string>='<string>'<string>
MCC-039	Extra transistor instance specific parameter <string>=%g found in subckt '<string>', found value used

MCC-040	Extra transistor instance specific parameter M=<decimal> : technology check won't work properly
MCC-041	<string> found on transistor <string> in technology file subckt '<string>'
MCC-042	Can't determine transistor model type for '<string>' using simToolModel setting '<string>'. Guessing '<string>'. Please set correct simToolModel ('<string>'?).
MCC-043	Can't open transistor model definition file <string>.
MCC-044	Parse error line <decimal> file <string>.
MCC-045	Can't parse transistor definition file. Search in <string> : <string>

6.17. MGL

MGL-001	syntax error line <decimal>
MGL-002	illegal vector range specification at line <decimal> The bounds specified for a bussed signal are not numerical values.
MGL-003	port '<string>' already declared at line <decimal> An external connector with the same name has already been declared in the module.
MGL-004	net '<string>' already declared at line <decimal> A net with the same name has already been declared in the module.
MGL-005	net '<string>' used out of declared range at line <decimal> A bit a range of bits is refered to for the given net despite being beyond the bounds specified in the net declaration.
MGL-006	width or/and type mismatch at line <decimal> A port connection in an instantiation does not match the port specification of the module being instantiated.
MGL-007	escaped vector '<string>' not declared in strict bit ascending/descending order at line <decimal> Only occurs if avtStructuralVerilogVector is set to yes. Escaped vectors declared as individual bits can be handled correctly only if the bits are declared in a strict ascending or descending order without any gaps.
MGL-008	escaped vector '<string>' not declared with all bits together at line <decimal> Only occurs if avtStructuralVerilogVector is set to yes. Escaped vectors declared as individual bits can be handled correctly only if the bits are declared together.

MGL-009	Missing external connector for external signal '<string>' driving verilog netlist '<string>' Occurs when driving an incoherent verilog netlist.
MGL-010	Cannot open file to drive verilog netlist '<string>' File I/O error, check disk space and privileges.

6.18. SLIB

Error messages description not available yet.

6.19. SPF

SPF-001	Invalid pin type '<string>' at line <decimal> The pintype field does not contain a valid [IiOoSsBbJjXx] character.
SPF-002	Syntax error at line <decimal>, token '<string>' The given line does not conform to legal SPF syntax. If token is 'CR', there are probably missing elements on the line.
SPF-003	Undeclared node '<string>' (line <decimal>) The given node is not declared in the current net.
SPF-004	Net '<string>' with mismatching total capacitance (total=%gpf, sum=%gpf) The total capacitance already associated with a net in a design does not match the value given for the total capacitance for the net in the SPF file.
SPF-005	Unsupported divider, using '/' (line <decimal>) A missing or invalid character used to specify the hierarchy divider.
SPF-006	Unsupported delimiter, using ':' (line <decimal>) A missing or invalid character used to specify the name delimiter.
SPF-007	Unknown syntax for BUSBIT section (line <decimal>) Illegal syntax used to specify bus delimiters in the BUSBIT section.
SPF-008	Undefined design entity name The name of the design to be annotated is not specified in the SPF file.
SPF-009	Entity '<string>' does not exist The design entity specified for annotation does not exist. Either the netlist has not been loaded yet or the names do not match.
SPF-010	Unknown capacitance unit '<string>', assuming 'pf' (line <decimal>)

Illegal specification of capacitance unit. Legal values are ff, pf, nf, uf, mf, kf. The final 'f' is optional and case is irrelevant.

SPF-011 `Unknown resistance unit '<string>', assuming none (line <decimal>)`
Illegal specification of resistance unit. Legal values are f, p, n, u, m, k. The case is irrelevant.

SPF-012 `Device or instance '<string>' cannot be found (line <decimal>)`
The object specified for connection to the parasitic network does not exist.

SPF-013 `Connector of transistor '<string>' named '<string>' cannot be found (line <decimal>)`
The name used to specify the transistor connector is invalid. Legal names are g, d, s, b.

SPF-014 `Connector of transistor '<string>' named '<string>' might be connected to the wrong signal (line <decimal>)`
There may be an incoherance in the parasitic file.

SPF-015 `Instance '<string>' cannot be found (line <decimal>)`
The instance specified for connection to the parasitic network does not exist.

SPF-016 `Connector of instance '<string>' named '<string>' cannot be found (line <decimal>)`
The name used to specify the instance connector is invalid.

SPF-017 `Connector '<string>' cannot be found (line <decimal>)`
The external pin specified for connection to the parasitic network does not exist.

SPF-018 `Connector named '<string>' is on the wrong equipotential (line <decimal>)`
There may be an incoherance in the parasitic file.

SPF-019 `Negative capacitance found (line <decimal>)`
A negative value was used to specify a capacitance, this is ignored.

SPF-020 `Negative resistance found (line <decimal>)`
A negative value was used to specify a resistance, this is ignored.

SPF-021 `<decimal> signal(s) missing in the netlist`
Signals specified in the parasitic file do not exist in the netlist.

SPF-022 `Don't know how to handle connector '<string>' for resistor annotation (line <decimal>)`
The handled names for resistor connectors are 'pos', 'neg' , '1' and '2'.

SPF-023 `Signal '<string>' missing in the netlist (line <decimal>)`
Signal can not be found in netlist to annotate.

SPF-024	Failed to open file '<string>' The dspf file was not found.
SPF-025	RC Network declaration for signal '<string>' continued at line <decimal> Multiple consecutive declarations for the signal have been detected. They will be merged.
SPF-026	Don't know how to handle capacitance/diode connector '<string>' for capacitance/diode annotation (line <decimal>) The handled names for capacitance and diode connectors are '1' and '2'.
SPF-027	No declaration of crosstalk capacitance node '<string>' (line <decimal>), associated with net '<string>'
SPF-028	Node '<string>' (line <decimal>) could not be created, coupling capacitance to this node will be ignored A coupling capacitance to a non existant signal in the netlist will be ignored.

6.20. STM

STM-000	Array bound write in model '<string>'
STM-001	Parse error file '<string>' line <decimal>
STM-002	solvepi (): non convergence
STM-003	Negative capacitance found on signal <string>
STM-004	Deleting more models than allocted
STM-005	stm_cell_delmodel: null cell
STM-006	stm_cell_delmodel: null model
STM-007	Can't open file '<string>'
STM-008	stm_addht impossible: hash table size is '0'
STM-009	stm_addhtitem impossible: value is STM_EMPTYHT or STM_DELETEHT
STM-010	Constraint calculation meaningless in SCM model
STM-011	Cannot resolve pi tree with tables: taking c1 + c2
STM-012	Cannot resolve pi tree with polynoms: taking c1 + c2
STM-013	Load parameter not yet implemented for polynoms

STM-014	Clock slew parameter meaningless in SCM models
STM-015	Clock slew parameter not yet implemented for polynoms
STM-016	Data slew parameter meaningless in SCM models
STM-017	Data slew parameter not yet implemented for polynoms
STM-018	imax not yet implemented for tables
STM-019	imax not yet implemented for polynoms
STM-020	Slew parameter not yet implemented for polynoms
STM-021	Merge of polynom models not yet implemented
STM-022	Reduction of polynoms not yet implemented
STM-023	Shift of polynoms not yet implemented
STM-024	Negation of polynoms not yet implemented
STM-025	No model type for signature
STM-026	STM cannot compute imax from constant model
STM-027	STM cannot compute vth from constant model
STM-028	STM cannot compute slope from constant model
STM-029	NULL model
STM-030	Lost memory consistency
STM-031	Shift of SCM models meaningless
STM-032	Negation of SCM models meaningless
STM-033	Table extrapolation
STM-034	Noise created twice
STM-035	Cannot get current file position
STM-036	Cannot set file position
STM-037	Bad unit for STM_CACHESIZE
STM-038	invth created twice

STM-039	Default capacitance range used for function model
STM-040	Could not find timing model file for figure '<string>' The .stm file might not be accessible to the tool.
STM-041	Constant and table cannot be both valid
STM-042	CALL_SIMULATION called out of context
STM-043	CALL_SIMULATION can't find association for %p '<string>'
STM-044	CALL_SIMULATION_ENV called out of context
STM-045	CALL_SIMULATION_ENV can't find association for %p '<string>'
STM-046	CALL_CTK_ENV called out of context
STM-047	CALL_CTK_ENV can't find association for '<string>'
STM-048	GNS information not loaded, can't compute
STM-049	Could not retreive instance <string>
STM-050	Round overflow
STM-051	Parse error while reading TLF file line <decimal>
STM-052	Imprecision risk for delay '<string>(<character>)' to '<string>(<character>)', slope = <decimal>ps, load = <decimal>fF.

6.21. TAS

TAS-001	slope computing error : circuit not extracted or wrong extraction
TAS-002	netlist error: the connector <string> is not an input
TAS-003	netlist error: the connector <string> is not an output
TAS-004	the connector <string> is not used
TAS-005	the connector <string> can not reach the high level
TAS-006	the connector <string> can not reach the low level
TAS-007	can not force the level of connector <string>
TAS-008	can not open the file <string>

TAS-009	the signal <string> doesn't exist in the circuit or is not a transistor gate or is already declared
TAS-010	can not close the file <string>
TAS-011	no control signal in the latch %ld (<string>)
TAS-012	incomplete input list for the cone %ld (<string>)
TAS-013	the latch <string> is not a differential latch
TAS-014	error in technology file <string>
TAS-015	block without diffusion capacitance. can not measure the drain or source capacitance
TAS-016	block without diffusion capacitance drain and source capacitance will be measured with ACM=1 SPICE method
TAS-017	floating exception
TAS-018	no value or edge can be set for signal <string>. set to 0
TAS-019	can not find the slopes of the cone %ld (<string>)
TAS-020	no active branch found in the cone %ld (<string>)
TAS-021	error in the truth table %ld (<string>)
TAS-022	unknown state <string>
TAS-023	unknown column type <string>
TAS-024	non external branch containing connector cone %ld (<string>)
TAS-025	no slope on the node <string>
TAS-026	usage of the new delay switch model requiere to set the TAS_DELAY_SWITCH to NO.
TAS-027	tasalloc() error: not enough memory. only the critical path will be retained. freeing memory space
TAS-028	fatal error: not enough memory
TAS-029	unknown event %ld
TAS-030	no inout or output connector
TAS-031	loop detected: see the <string>.loop file for more informations

TAS-032	monolatch chain %ld (<string>)
TAS-033	the cone %ld (<string>) does not contain feedback pathes
TAS-034	non-external branch in the cone %ld (<string>)
TAS-035	latch on output connector <string>
TAS-036	precharge on output connector <string>
TAS-037	negative or zero delay between <string>
TAS-038	transistor link losed
TAS-039	too small resistance for signal <string>. set to 1
TAS-041	the precharge processing must be used with the 'ttv' file. the '-n' option will be set
TAS-042	there is no extractible path in the figure <string>
TAS-043	bad environment variable <string>
TAS-044	the cell characterization option is incompatible with the 'deb' file
TAS-045	errors detected in the netlist. see <string>.rep file for more details
TAS-046	latch on input connector <string> ignored
TAS-047	precharge on input connector <string> ignored
TAS-048	memory on input connector <string> ignored
TAS-050	transistors detected in figure <string> impossible in hierarchical mode or instances do not have timing files
TAS-051	impossible to find in figure <string>
TAS-052	bad connector direction on signal <string>
TAS-053	internal connector <string> not found in RCX view
TAS-054	only one connector on internal signal <string>
TAS-055	only input connectors or constante connectors on internal signal <string>
TAS-056	no input on internal signal <string>

TAS-057	conflict detected on internal signal <string>
TAS-058	no rc delay on signal <string>. may be the rc tree are not connected or there is no slope on the driver
TAS-059	more than one latch on signal <string>
TAS-060	more than one precharge on signal <string>
TAS-061	fatal error check your input files
TAS-062	figure <string> already exists
TAS-063	no rc delay on signal <string>. may be loop detected
TAS-064	no or bad rcx file for figure <string>
TAS-065	name <string> already exists
TAS-066	It is better to simulate the signal <string>
TAS-067	No timing model for instance <string>: flattened
TAS-068	No execution context (missing tas_setenv)
TAS-069	Conflicting temperature: <string>
TAS-070	PVT conditions prevent signal propagation on cone <string>, timing arc suppressed Vin under static threshold (Increasing power supply may solves)
TAS-071	PVT conditions lead to out of bound parameters on cone <string>, timing arc suppressed Delay and slope calculations failed on pass-gate (Increasing power supply may solves)
TAS-072	Signal '<string>' connected on power connector '<string>' of instance '<string>'
TAS-073	Could not find connector '<string>' in circuit '<string>'
TAS-074	can't extract path to simulate
TAS-075	can't get patterns to simulate the path

6.22. TRC

TRC-000	`[AWE:<decimal>] Internal error :`
	`net <string>`
	`driver <string>`
	`receiver <string>`
	The internal data representation for parasitic element is not consistent. Please contact Avertec Support.

TRC-001	`[AWE] Negative delay computed on net <string>` Parasitics on the net present some particularities, leading to an error when computing delays with AWE algorithm.
TRC-002	`[AWE] All order fail when computing delay on net <string>` Parasitics on the net present some particularities, leading to an error when computing delays with AWE algorithm.
TRC-003	`Can't open file <string> for direct access. Cache is not used.` The .rcx file is compressed, so it can't be opened for direct access. The cache is not used, the file is entirely loaded in memory, leading to more memory consumption.
TRC-004	`avtElpCapaLevel mismatch #<decimal>.` The value of the avtElpCapaLevel variable is not consistent with previous setting.
TRC-005	`Internal error #<decimal> on net <string>.` This is an internal error. Please contact Avertec Support
TRC-006	`Internal error #<decimal>.` This is an internal error. Please contact Avertec Support
TRC-007	`Negative ground capacitance found on signal <string>. code=<decimal>` The total capacitance on a net is negative. If it's a power supply net, it should be identified, and no operation on this signal would occur. Check configuration file.
TRC-008	`Can't determine equivalent load on signal <string>.` The equivalent gate output load can't be computed. The gate output is probably not connected to anything.
TRC-009	`The pi load for the equivalent gate load on signal <string> can't be determined.` The equivalent gate output load can't be computed. The gate output is probably not connected to anything, or parasitics on the net present some others particularities.
TRC-010	`The equivalent gate output load gives a negative capacitance for signal <string>. 0 value is retained` This problem occurs when there is a very strong coupling between nets.
TRC-011	`Can't open file <string>.<string> for writting.`

Check usual Unix specification for file access: permission mode on file or on directory, disk space,...

TRC-012 `An error occurs when writting in file <string>.`
Check usual Unix specification for file access: permission mode on file or on directory, disk space,...

TRC-013 `Try to load an RCX file version #<decimal>. Only 4th version is supported.`
The RCX file is too old to be supported. It must be created with newer version of the tool.

TRC-014 `File parse error : Line too long.`
The length of a line in the RCX file is too long. This case occurs if there are very long signal or instance names, too many nodes on a connector, or if the file is corrupted. The last line of an RCX file is the word END.

TRC-015 `File parse error line <decimal> in file <string>.<string>.`
An element is not recognized in the RCX file. The file may be corrupted. The last line of an RCX file is the word END. If this is not the case, there has been a problem during the creation of the RCX file.

TRC-016 `Error while getting environment variable : <string>. It is <string>`
Check the Avertec Documentation to set proper values.

TRC-017 `could not modify connector '<string>' origin group`
This is an internal error in rc delay calculation with GNS.

TRC-018 `rcx timing function does not exist`
This is an internal error in rc delay calculation with GNS.

TRC-019 `could not get connector '<string>' origin group`
This is an internal error in rc delay calculation with GNS.

TRC-020 `origin index of connector '<string>' is out of range`
This is an internal error in rc delay calculation with GNS.

TRC-021 `could not get connector '<string>' destination group`
This is an internal error in rc delay calculation with GNS.

TRC-022 `destination index of connector '<string>' is out of range`
This is an internal error in rc delay calculation with GNS.

TRC-023 `rcx timing function does not exist`
This is an internal error in rc delay calculation with GNS.

TRC-024 `rcx timing function already exist`
This is an internal error in rc delay calculation with GNS.

TRC-025	missing rcx timing origin or destination group This is an internal error in rc delay calculation with GNS.
TRC-026	could not find timings \"<character><character>\" between connectors '<string>' and '<string>' for signal '<string>' This is an internal error in rc delay calculation with GNS.
TRC-027	could not find timings \"<character><character>\" for an input slope of %gps between connectors '<string>' and '<string>' for signal '<string>' This is an internal error in rc delay calculation with GNS.
TRC-028	can't compute awe delay because a connector is missing : start='<string>' end='<string>' This is an internal error in rc delay calculation.
TRC-029	can't compute awe delay because input slope is negative : start='<string>' end='<string>' The input slope for rc delay calculation is negative. There is a problem with the gate model. Check technological parameters and power supply.
TRC-030	can't compute awe delay because power supply is less than vt : start='<string>' end='<string>' Check power supply and technological parameter of the driver.
TRC-031	can't compute awe delay because threshold is greater than power supply : start='<string>' end='<string>' The voltage on the rc net can't reach the threshold delay measure.
TRC-032	can't compute awe delay because because there is no rcx view or rcx view is typed RCXERROR : start='<string>' end='<string>' This is an internal error in rc delay calculation.
TRC-033	can't compute effective load on signal <string> because there is not a valid rcx view. This is an internal error in rc delay calculation.
TRC-034	can't get the file position : <string> When using rc cache, can't get the file offset. Try to disable file compression if needed, or remove rc cache. Check unix file so.
TRC-035	can't set the file position : <string> When using rc cache, can't change the position in file. Try to disable file compression if needed, or remove rc cache. Check unix file access so.
TRC-036	an error occured when reading file : <string> This rcx file may be corrupted or truncated. The last line of the rcx file is EOF. Check disk space.

TRC-037	`an inconsistant line has been read : <string>` The rcx file is corrupted or truncated. The last line of the rcx file is EOF. Check disk space.
TRC-038	`No aggressor <string> found : <string>` The rcx file is corrupted or truncated. The last line of the rcx file is EOF. Check disk space.
TRC-039	`Signal <string> is defined more than one time : <string>` The rcx file is corrupted.
TRC-040	`Signal <string> is not defined : <string>` The rcx file is corrupted.
TRC-041	`Problem in cache mechanism : the signal <string> can't be released`
TRC-042	`[AWE] can't compute the switching instant of the input for rc net <string>` There is a problem when computing the rc input slope on a net. In this case, the rc delay is equal to the switching instant of the rc output.
TRC-043	`[AWE] can't compute the switching instant of the output for rc net <string>` There is a problem when computing the rc output slope on a net. In this case, the rc delay is equal to 0 and output rc slope is equal to input rc slope.
TRC-044	`[AWE] can't determine the shape of the input for rc net <string>` The representation for the rc driver can't be determined. In this case, the rc delay is equal to 0 and output rc slope is equal to input rc slope.
TRC-045	`internal problem in rcdelay cache : signal <string> is yet in cache` There is a proble in rcdelay cache mechanism. Desactivate rcdelay cache.
TRC-046	`internal problem in rcdelay cache : no signal` There is a proble in rcdelay cache mechanism. Desactivate rcdelay cache.
TRC-047	`internal problem in rcdelay cache : signal <string> is not in cache` There is a proble in rcdelay cache mechanism. Desactivate rcdelay cache.
TRC-048	`internal problem in rcdelay cache : unknown data structure in cache for signal <string>` There is a proble in rcdelay cache mechanism. Desactivate rcdelay cache.
TRC-049	`internal problem in rcdelay cache : no awe data for signal <string>` There is a problem in rcdelay cache mechanism. Desactivate rcdelay cache.
TRC-050	`Net <string> is not connex` The rc description of a net doesn't connect all connectors\n

TRC-051	`Disabling compression for <string>.rcx file as RC cache is active.` There is a request to write an .rcx file with the simultaneous uses of output filter and rc cache functionality activated. Since this last functionality requiers direct access on file, the output filter is not used for this file.
TRC-051	`Net <string> is not connex` The rc description of a net doesn't connect all connectors\n
TRC-052	`Can't solve the LU hybrid matrix on net <string> from locon <string>` There is a problem in awe computation with matrix.\n
TRC-053	`Can't handle UTD for circuit '<string>' generated with avtEnableMultipleConnectorsOnNet=yes in hierarchical mode` avtEnableMultipleConnectorsOnNet=yes is supported only for transistor level netlists.\n

6.23. VAL

Error messages description not available yet.

6.24. YAG

YAG-001	`Circuit not found` Attempt to disassemble a circuit entity which does not exist.
YAG-002	`RC cache must be deactivated in this mode` Disassembly modes which involve manipulation of hierarchy require that the RC be deactivated.
YAG-003	`Connector '<string>' is power supply AND ground` The given external connector has been detected as both Vdd and Vss. This error is fatal.
YAG-004	`No VDD signal in the circuit` No Vdd power supply has been identified in the circuit to disassemble. This error is not fatal, however, it is recommended to check the power supply configuration.
YAG-005	`No VSS signal in the circuit` No Vss power supply has been identified in the circuit to disassemble. This error is not fatal, however, it is recommended to check the power supply configuration.
YAG-006	`Can't open file '<string>'` The specified file cannot be opened. Check disk space and access privileges. This error may be fatal.

YAG-007	`Can't close file '<string>'` The specified file cannot be closed. Check disk space and access privileges.
YAG-008	`Unrecognized contraint '<string>' in the circuit` The specified mutual exclusion constraint has no correspondance in the circuit to disassemble.
YAG-009	`Can't name toplevel <string> in Blackbox mode` Attempt to rename the top-level figure using a name which is already in use.
YAG-010	`Figure '<string>' has no transistors and no instances` Circuit contains no elements to disassemble.
YAG-011	`Transistors in non-leaf figure '<string>'` In hierarchical disassembly mode, if a no-leaf figure contains a mixture of transistors and instances then the result may not be as expected.
YAG-012	`Signal '<string>' is not driven in behavioural figure`

Index

apiDriveAllBehavior **75**
apiDriveCorrespondenceTable **75**
apiFlags **74**
apiUseCorrespondenceTable **75**
avtAnnotationDeviceConnectorSetting **58**
avtAnnotationKeepCards **48**
avtAnnotationPreserveExistingParasitics **57**
avtBehavioralVerilogSuffix **57**
avtBehavioralVhdlSuffix **56**
avtBlackboxFile **45**
avtCaseSensitive **46**
avtCatalogueName **45**
avtDisableCompression **48**
avtElpCapaLevel **47**
avtElpDriveFile **48**
avtElpGenTechnoName **48**
avtEnableMultipleConnectorsOnNet **50**
avtFilterSuffix **48**
avtFlattenForParasitic **49**
avtFlattenKeepsAllSignalNames **46**
avtFullConeFile **71**
avtGlobalVddName **46**
avtGlobalVssName **46**
avtInputFilter **48**

avtInstanceSeparator **46**
avtLibraryDirs **45**
avtLicenseProject **45**
avtLicenseServer **45**
avtMaxCacheFile **49**
avtNormalConeFile **70**
avtOutputBehaviorFormat **72**
avtOutputBehaviorVectorDirection **72**
avtOutputFilter **48**
avtParasiticCacheSize **49**
avtSpiConnectorSeparator **50**
avtSpiCreateTopFigure **49**
avtSpiDriveCapaMini **55**
avtSpiDriveDefaultUnits **54**
avtSpiDriveParasitics **55**
avtSpiDriveResiMini **55**
avtSpiDriveTrsInstanceParams **55**
avtSpiDspfBuildPower **58**
avtSpiDspfLinkExternal **58**
avtSpiFlags **53**
avtSpiFlags **55**
avtSpiHandleGlobalNodes **54**
avtSpiIgnoreCrypt **52**
avtSpiIgnoreDiode **51**
avtSpiIgnoreModel **52**
avtSpiIgnoreVoltage **52**
avtSpiInstanceMultiNode **51**

avtSpiJFETisResistance **52**
avtSpiKeepCards **51**
avtSpiKeepNames **51**
avtSpiMaxResistance **52**
avtSpiMergeConnector **50**
avtSpiMergeDiodes **52**
avtSpiMinCapa **53**
avtSpiMinResistance **52**
avtSpiNameNodes **51**
avtSpiNodeSeparator **51**
avtSpiOneNodeNoRc **53**
avtSpiOrderPinPower **53**
avtSpiParseFirstLine **50**
avtSpiPinDspfOrder **58**
avtSpiRCMemoryLimit **55**
avtSpiReplaceTensionInExpressions **50**
avtSpiShortCircuitZeroVolts **52**
avtSpiTolerance **54**
avtSpiUseUnits **54**
avtSpiVector **54**
avtStructuralVerilogSuffix **57**
avtStructuralVerilogVectors **57**
avtStructuralVhdlConfigure **56**
avtStructuralVhdlSuffix **56**
avtTechnoModelSeparator **47**
avtVddName **45**
avtVddVssThreshold **49**

avtVectorize **46**
avtVerboseConeFile **70**
avtVerilogKeepNames **57**
avtVerilogMaxError **57**
avtVhdlMaxError **56**
avtVssName **45**
fclAllowSharing **73**
fclCutMatchedTransistors **73**
fclDebugMode **73**
fclGenericNMOS **72**
fclGenericPMOS **73**
fclLibraryDir **72**
fclLibraryName **72**
fclMatchSizeTolerance **73**
fclTraceLevel **73**
fclWriteReport **73**
gnsFlags **74**
gnsKeepAllCells **74**
gnsLibraryDir **74**
gnsLibraryName **73**
gnsTemplateDir **74**
gnsTraceFile **74**
gnsTraceLevel **74**
gnsTraceModel **74**
xyagIconLibrary **75**
xyagMakeCells **75**
yagAnalysisDepth **61**

yagAssumeExpressionPrecedence **69**
yagAutomaticLatchDetection **64**
yagAutomaticMemsymDetection **65**
yagAutomaticRSDetection **65**
yagBddCeiling **61**
yagBleederIsPrecharge **69**
yagBleederStrictness **66**
yagBlockBidirectional **63**
yagBusAnalysis **70**
yagCapacitanceCones **63**
yagCompactBehavior **70**
yagDebugCone **59**
yagDetectClockGating **66**
yagDetectDelayedRS **67**
yagDetectDynamicLatch **65**
yagDetectGlitchers **62**
yagDetectPrecharge **66**
yagDriveAliases **70**
yagDriveConflictCondition **70**
yagElectricalThreshold **61**
yagElpCorrection **59**
yagGenerateBehavior **71**
yagGenerateConeFile **71**
yagGenerateConeNetList **71**
yagGeniusTopName **71**
yagGenSignature **71**
yagHierarchicalMode **59**

yagHierarchyGroupTransistors **72**
yagHzAnalysis **61**
yagIgnoreBlackboxes **60**
yagKeepRedundantBranches **62**
yagLatchesRequireClocks **66**
yagMaxBranchLinks **61**
yagMaxSplitCmdTiming **68**
yagMinimizeInvertors **70**
yagMutexHelp **59**
yagMutexHelp **63**
yagNoSupply **69**
yagNotStrict **59**
yagOneSupply **68**
yagOutputName **71**
yagPullupRatio **62**
yagRelaxationAnalysis **62**
yagRelaxationMaxBranchLinks **61**
yagRemoveInterconnects **60**
yagReorderInterfaceVectors **69**
yagSearchLoops **59**
yagSensitiveTimingDriverLimit **68**
yagSensitiveTimingRatio **68**
yagSetResetDetection **64**
yagSilentMode **60**
yagSimpleLatchDetection **63**
yagSimpleOrientation **62**
yagSimplifyExpressions **69**

yagSimplifyProcesses **69**

yagSplitTimingRatio **68**

yagStandardLatchDetection **66**

yagSuppressBlackboxes **60**

yagTasTiming **67**

yagTestTransistorDiodes **63**

yagTristateIsMemory **69**

yagUseGenius **67**

yagUseNameOrientation **63**

yagUseOnlyGenius **67**

yagUseStmSolver **61**

yagWriteStatistics **59**