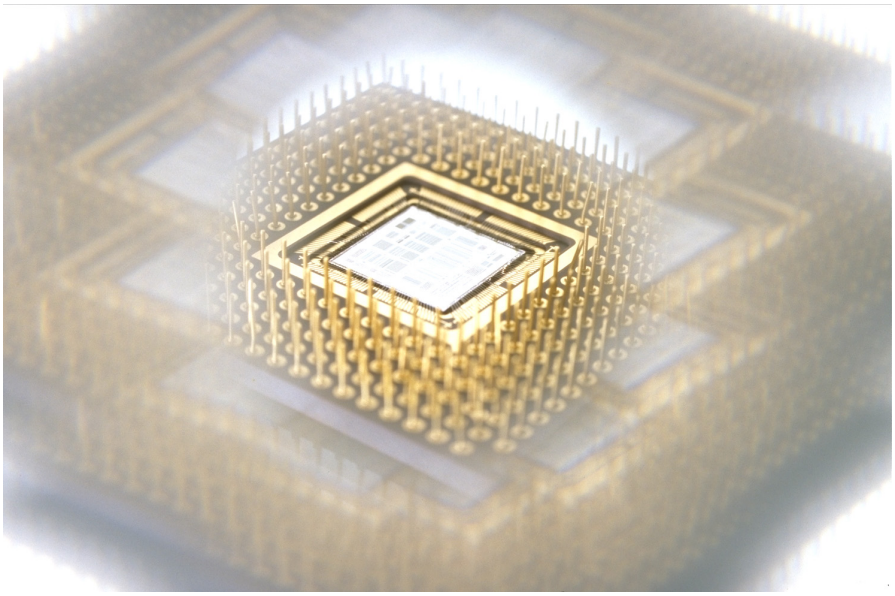
Avertec Tools

Yagle
User Guide



Software Release 3.4p5

June 7th, 2010

About this Document

This document explains:

- Software installation
- The input formats supported
- How to perform the functional abstraction
- The VHDL/Verilog generated
- User constraints
- The Graphical User Interface

Documentation issued and compliant with Avertec Tools Release 3.4p5.

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

Table of Contents

Chapter 1. Software Installation

1.1. System Requirements

The complete installation requires approximately 650Mb disk space. If you wish to execute all the examples, you will need 700Mb of free disk space.

The following platforms are supported:

Solaris	8, 9, 10 (32bit and 64bit for each)
Linux	RedHat Enterprise Linux 3.0 (32bit and 64bit)

1.2. What the Distribution Provides

The distribution provides all the relevant files required to install and operate the Avertec tools. This includes:

- Installation script
- End-user license agreement
- Binary executables
- License server data
- Manual pages
- Documentation in PDF and HTML format
- Tutorials
- Environment configuration files

1.3. Scope of the Installation

The distribution can be installed onto any part of a file system so long as the person performing the installation has write access privileges. You may, for example, choose to install all the tools in a user's home directory. Alternatively, you may install the tools on an NFS file server for multi-user access. In both cases, the installation process is the same, apart from the location on the file system. The only requirements for the execution of the binaries are appropriate access privileges together with a network connection to the machine chosen to act as the license server.

1.4. Performing the Installation

If starting from a CD-ROM, you must first perform the necessary commands to mount it.

You should then open a terminal and change directory to the place on the file system you want the tools to be installed. Launch the installation script as follow.

```
> /cdrom/AvtTools/Install (Solaris)
> /mnt/cdrom/Install (Linux)
```

If starting from a TAR archive file, you must first untar it, and change directory to the place on you want the tools to be installed

```
> cd /users/me/tar/
> tar -xvf AvtTools_2.8.tar
> cd /users/me/work/
> /users/me/tar/AvtTools_2.8/Install
```

The installation script present you with the installation choices detailed in the subsequent sections. For each choice you will be given a default reply (in square brackets) which you can accept by simply pressing the <RETURN> or <ENTER> key. Unless the choice requires a file or a directory path in response, you will also be given the list of possible replies. An invalid response will result in an error message and will take you straight back to the original question.

```
Enter the source directory [/users/me/tar/AvtTools_2.8]:
```

Root directory the distribution is installed from. If installation is done from a CD-ROM, default is the root directory of the CD-ROM. If installation is done from an archive, default is the root directory of the archive.

```
You must accept the following license agreement before installation

Press return to continue
```

Text of a license agreement. Press <SPACE> to advance one screen at a time, or <ENTER> to advance one line at a time. Please read carefully all the terms of this agreement.

```
Do you accept the terms and conditions? [accept]:
```

You must accept the terms of this license agreement before being able to continue with the installation.

```
Enter the destination directory [/users/me/work/AvtTools]:
```

Full path of the directory you wish to install the software in. By default this is a subdirectory named AvtTools of the current directory.

```
Directory /users/me/work/AvtTools does not exist...

Do you want to create it now y/n? [y]:
```

```
Creating installation directory...
```

If specifying a destination directory that does not exist, you will be asked to confirm its creation. If you type `n` then you will be asked to specify an alternative directory.

```
Enter the OS to install

        S2.6       : Solaris 2.6
        S2.8       : Solaris 2.8
        S2.8_64    : Solaris 2.8 64bits
        S2.9       : Solaris 2.9
        S2.9_64    : Solaris 2.9 64bits
        RHEL3.0    : Red Hat Enterprise Linux 3.0
        RHEL3.0_64 : Red Hat Enterprise Linux 3.0 64bits
        RHL8.0     : Red Hat Linux 8.0

OS [S2.6 S2.8 S2.8_64 S2.9 S2.9_64 RHEL3.0 RHL8.0]:
```

By default executables for all supported platforms are installed. However, you may wish to install only those which you require.

Hit `<ENTER>` to accept the default, or type the name of the platform for you wish to install.

```
Enter the license server name [cardiff]:
```

Name of the machine you intend to run the license server on. By default, it is the name of the current machine.

1.5. Setting-up the Environment

The installation process creates a CSH environment file setting environment variables for tool access:

```
source $AVT_TOOLS_DIR/etc/avt_env.csh
```

On 64bit systems, one can choose to use either 32bit or 64bit software version. To use 64bit-software version, add the following argument:

```
source $AVT_TOOLS_DIR/etc/avt_env.csh 64
```

Where `$AVT_TOOLS_DIR` is the destination directory of the installation.

You can either source this file or set explicitly the appopriate environment variables in a startup script such as the .cshrc.

The variables to set are:

`AVT_TOOLS_DIR`	Full path of the Avertec tools root directory.
`PATH`	Access paths for the appropriate binaries, e.g. `$AVT_TOOLS_DIR/tools/Solaris_2.8/bin`

`LD_LIBRARY_PATH`	Access paths for the appropriate shared object (`.so`) libraries. `e.g. $AVT_TOOLS_DIR/tools/Solaris_2.8/api_lib`
`MANPATH`	Access paths for the Avertec man pages, e.g. `$AVT_TOOLS_DIR/man`
`AVT_LICENSE_SERVER`	Name of the machine hosting the licence server.
`AVT_LICENSE_FILE`	Full path of the licence file.

1.6. The FLEXLM Licence Server

Yagle license control is done through the standard FLEXLM license server. Avertec's license server daemon is `avtlicd`.

The command:

```
> lmgrd -c <avertec_license_key_file>
```

sets `AVTLICD_LICENSE_FILE` to `avertec_license_key_file`

starts `avtlicd` (provided it is in `$PATH`)

creates `~/.flexlmrc`

Chapter 2. Overview

2.1. Functional Abstraction with Yagle

Yagle performs the automatic generation of HDL descriptions, in Verilog or VHDL, from transistor-level netlists, by partitioning and analyzing the network of transistors. Tri-state nodes of the circuit are expressed as VHDL Bus. Latches and registers are expressed as conditioned statements within separate VHDL processes. The process is totally free of user intervention and does not require any pre-defined library. Nevertheless, a user-defined gate library can be provided in order to handle complex latches or analog circuitry.

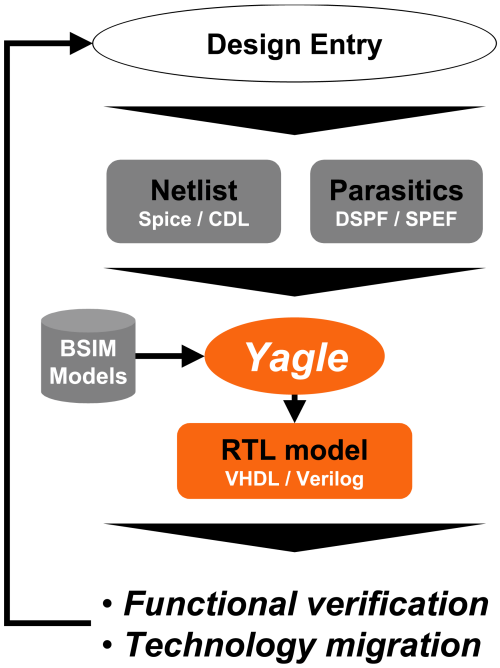
The generated HDL descriptions can be used by common verification tools. Yagle allows functional modeling and verification of full and semi-custom designs, by logical simulation or equivalence checking.

Yagle's HDL descriptions are also compliant with synthesis tools requirements, and allow easy technology migration.

Yagle's unique ability to provide timing back-annotated HDL descriptions, close to physical implementation, enables the setup of solutions based on signal activity, such as power consumption or IR-drop analysis.

A hierarchical pattern-matching engine allows genuine treatment of analog cells. Its memory-array recognition capability enables Yagle to abstract Mbytes SRAMs in a matter of minutes.

The following diagram illustrates Yagle's integration in the design flow.



The most important features of Yagle are:

- Handles complex CMOS and NMOS circuitry including pass-transistors, precharge logic, and domino logic.
- Works on complex blocks such as microprocessor cores, PCI components, routing components and multimedia systems.
- Automatically detects and models latches and registers
- Mixed Analog/Digital components handled by an optional user-defined device library.
- Pattern-matching engine for memory array recognition, built-in analog library
- Flat or hierarchical SPICE/CDL input transistor netlist.
- DSPF, SPEF parasitics support
- Industry-standard VHDL or Verilog behavioral output, compatible with commercial simulation and synthesis tools.
- Structural VHDL or Verilog output suitable for automatic test pattern generators.
- SPICE accurate timing annotation, with BSIM3 and BSIM4 transistor models support
- GUI and Tcl interface

Yagle is able two generate HDL behavioral descriptions at different levels of abstraction.

Closest to physical implementation is a low-level HDL. Associated with this description, Yagle generates a correspondence table, that link electrical and logical names.

A high-level HDL is obtained by expression simplification and intermediary signals suppression.

A compact HDL can also be obtained by vectorization. The pattern-matching engine identifies the repetitive structures, such SRAM arrays, and vectorizes the HDL descriptions according to them.

2.2. Applications

The main applications of Yagle are:

- Functional verification of digital custom designs through simulation or equivalence checking
- ROM content verification
- RAM / CAM formal verification
- IP-reuse and technology migration
- Accurate signal activity obtaining, thanks to timing-annotated low-level HDL descriptions, enabling power consumption computation or IR-drop analysis
- BIST routines validation. The correspondence table between logical and electrical names enables easy identification of the physical nodes activated by the BIST routines.

2.3. Description

2.3.1. Functional Abstraction

The Yagle tool offers designers a revolutionary new strategy for the functional verification of their digital custom circuits, known as Functional Abstraction. Previous strategies for functional verification at the lowest level relied upon SPICE-like electrical simulations and were therefore limited to small circuit blocks. Functional Abstraction takes the task of behavioral verification to a higher level by directly extracting a simulatable RTL description from the transistor netlist by disassembly of the circuit.

2.3.2. Automatic Gate Model

The Yagle approach to circuit disassembly for functional abstraction can be defined as a partitioning of the transistor net-list, according to a limited number of generic rules. Each partition represents an extracted gate for which a behavioral description can be deduced. The result is a totally generic approach with a minimum of user intervention.

In the first phase, Yagle extracts the dual CMOS circuitry. In the second phase, Yagle builds the gate net-list for the remaining circuitry whilst performing functional analysis in parallel, in order to prevent the fabrication of false branches within a gate and to verify the behavior of the gate. This procedure allows Yagle to take into account the functional correlation in the surrounding circuitry. The depth of surrounding circuitry taken into account is adapted automatically within a maximum bound which can be specified by the user.

Chapter 3. Using Tcl Interface

3.1. Script Launch

All functionalities of the Yagle platform can be accessed with the `avt_shell` Tcl scripting interface. `avt_shell` can be used the same way as any `.tcl` script.

`avt_shell` can be used in interactive mode or in script mode. In interactive mode, it is invoked as follow:

```
> avt_shell
```

In script mode, the first line of the script file should look like:

```
#!/usr/bin/env avt_shell
```

3.2. Tools Configuration

The configuration of all the timing tools of the Yagle platform is done in the same way, by the mean of configuration variables. The value given to the variable determines the specific behavior of the tool. When using the Tcl interface, the setting of the values for the configuration variables can be done in two ways:

- In the special file `avttools.conf` in the working directory, with the syntax `variable = value`. Old way, not recommended. Only kept for backward compatibilty.
- In the `avt_shell` script, through the `avt_config` function, taking the variable for first parameter and its affected value for second parameter (`avt_config variable value`).

There is a precedence of the values set in the `avttools.conf` file on the values set in the `avt_shell script`.

3.3. Functions

Here is a list of the families of Tcl functions that can be found within the `avt_shell` interface. For more information, see Yagle Reference Guide.

General	Global configuration, file loading, netlist manipulation, statistics
INF Configuration	Configuration though the INF functions
HDL Construction	Automatic or manual generation of the VHDL/Verilog

3.4. INF and SDC Configuration

For tool configuration needing more than the specification of a simple value (as it is done through the `avt_config` function), Yagle uses the INF mechanism, which is a set of Tcl configuration functions.

SDC commands are grouped together with the INF functions and share the same mechanisms.

All INF functions begin with the `inf_` prefix, except of the SDC commands, which respect their standard naming.

Within a Tcl script, the target sub-circuit must be defined before using INF or SDC commands. Following example is given for a sub-circuit named `my_design`.

```
inf_SetFigureName my_design

set_case_analysis 1 reset
inf_DefineMutex muxup {i0 i1 i2}
```

It is possible to check the INF and SDC functions by driving a `.inf` file. Adding the line:

```
inf_Drive my_design
```

at the end of the previous Tcl script generates the `my_design.inf` file. Each INF and SDC function has a corresponding section in this file (see Yagle Reference Guide).

Chapter 4. Performing the Abstraction

4.1. File Loading

The purpose of this section is to show how to load files containing:

- Transistor technology models
- Design netlist
- Parasitic back-annotation

File loading is done with the Tcl command `avt_LoadFile`. Depending on the file format being read, and on the netlist specificities (such as vectors, connector order,...), additional configuration is sometimes required. Additional configuration should be set with `avt_config` Tcl commands, before invoking `avt_LoadFile`.

4.1.1. Transistor Technology Models

Transistor technology models are necessary to compute timings. If those transistor models appear in a separate file, they should be loaded in the Tcl script with the `avt_LoadFile` function. The `avt_LoadFile` function takes as first argument the name of the file to load, and as second argument its format. A typical loading of a technology file will be such as:

```
avt_LoadFile ../models/bsim3.tech spice
```

If the technology file makes inclusions of other files then inclusion paths should be absolute. If paths are relative, further configuration will be needed to specify the location of those files:

```
avt_config avtLibraryDirs .:../models
```

Technology file can also appear as an inclusion (`.INCLUDE` or `.LIB`) in a Spice netlist. In such a case, it will be loaded at the time the Spice netlist is loaded.

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.

Besides, the LEVEL parameter which appears in the model files is not discriminant enough. Different simulators may interpret differently a same LEVEL value (as it is the case for LEVEL 49, differently interpreted by HSPICE and ELDO). Therefore, it is necessary to specify the targetted simulator of the transistor model. It should be done with the following variable:

```
avt_Config simToolModel ELDO
```

If the `simToolModel` variable is not specified, Yagle will interpret the transistor model as HSPICE does (default value), and check the LEVEL against the following list:

```
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
```

If there is a conflict, for example if LEVEL=60 is given and `simToolModel` is not specified (defaulted to HSPICE), the tool will exit. User needs to properly set the `simToolModel` value.

4.1.2. Input Netlist

In a way or another, one must always provide a transistor-level description of the design. If impossible to give a transistor description for some parts of the netlist, Yagle can also take `.lib` files as input, but it should be understood that Yagle is primarily designed for digital transistor-level analysis, and that providing `.lib` files should only apply to parts of the netlist where Yagle does not apply, e.g. analog parts. Integration of `.lib` files will be discussed later.

A transistor level description can be provided within the following formats:

- Flat-transistor extracted Spice netlist
- Hierarchical Spice netlist, with Spice transistor-level leaf cells
- Hierarchical Verilog netlist, with Spice transistor-level leaf cells
- Hierarchical VHDL netlist, with Spice transistor-level leaf cells

Flat-transistor Spice netlist

A flat-transistor extracted Spice netlist is simply loaded with the following command:

```
avt_LoadFile my_design.spi spice
```

The file can contain parasitics, and preferably contains a `.SUBCKT` statement. If not, an implicit top-level is created, with all the nodes in the netlist reported on the interface. This can lead to computational explosion in further steps of the analysis.

Hierarchical Spice netlist

A hierarchical Spice netlist can be represented by several files. Those files can be loaded either through possibly recursive `.INCLUDE` statements, or through several `avt_LoadFile` commands. However, at least one `avt_LoadFile` command must appear in the Tcl script. The netlist is automatically flattened to the transistor-level, when all the dependancies have been resolved, e.g. when all instanciated sub-circuits correspond to a sub-circuit definition.

In a separate `avt_LoadFile` command, sub-circuit definition can appear after its instanciation, the order is not relevant. For example, the following file can be loaded by `avt_LoadFile my_design.spi spice`:

```
.SUBCKT my_design ...
...
.ENDS my_design

.INCLUDE ../leaf_cells/n1_y.spi
.INCLUDE ../leaf_cells/o3_y.spi
.INCLUDE ../leaf_cells/mx2_y.spi
```

Order is relevant if sub-circuit definitions appear in files read by separate `avt_LoadFile` commands. In that case reading the files containing sub-circuit definitions must be done before reading the files containing their instanciation, as shown in the follwing example:

```
avt_LoadFile leaf_cells/n1_y.spi spice
avt_LoadFile leaf_cells/o3_y.spi spice
avt_LoadFile leaf_cells/mx2_y.spi spice
avt_LoadFile my_design.spi spice
```

Hierarchical Verilog/VHDL netlist

The same example applies to a Verilog netlist and Spice transistor-level leaf-cells:

```
avt_LoadFile leaf_cells/n1_y.spi spice
avt_LoadFile leaf_cells/o3_y.spi spice
avt_LoadFile leaf_cells/mx2_y.spi spice
avt_LoadFile my_design.v verilog
```

or

```
avt_LoadFile my_design.vhd vhdl
```

4.1.3. Parasitics

Yagle treats parasitics files of two kinds:

- Parasitics used as a back-annotation of schematic netlists. In such as case, the connectivity of the schematic netlist is ensured without the parasitics file, which just brings additionnal information. The formats supported for back-annotation are DSPF and SPEF.
- Parasitics used to complete the description of the netlist. In such a case, the netlist is not connected without the parasitic information. Typically, the RC networks make the connectivity. The formats supported for connectivity description are Spice and DSPF (in this case the DSPF is used as a Spice file).

Back-annotation

When a parasitic file is used to back-annotate a schematic netlist, the schematic netlist must be loaded first, through a separate `avt_LoadFile` command. Just invoking the load of the parasitic file afterwards is enough to perform the back-annotation:

```
avt_loadfile my_design.spi spice
avt_loadfile parasitics.spef spef
```

or

```
avt_loadfile my_design.spi spice
avt_loadfile parasitics.spf dspf
```

When using back-annotation, special attention should be paid to name consistency between netlist and parasitics, especially regarding vectors (see next chapter).

Connectivity

If the parasitics file is necessary to ensure the connectivity of the netlist, the parasitics and netlist files should be loaded through a single `avt_LoadFile` command. Parasitic files should be included at appropriate levels of hierarchy with `.INCLUDE` statements.

4.1.4. Vectorization

Yagle has two operating modes regarding vectors. One can choose between a mode where vectors are represented internally as they appear in the source file, and a mode where they are identified as special signals and represented internally accordingly. When a vector is identified as a special signal, the internal representation is a string containing the radical and the index separated by a space character. For example the vector `dummy[0]` is represented as `dummy 0`.

Different delimiters can be used to represent vectors. Configuration of legal delimiters, as well as the choice to treat vectors as special, should be done with the `avtVectorize` configuration variable:

```
avt_config avtVectorize "[],<>"
```

Treating vectors as special signals is usefull when the same vectors can appear with different delimiters in different files. For example if a vector is referred to as `dummy[0]` in a Verilog file, and as `dummy<0>` in a SPEF file, the previous configuration is necessary to make the correspondance between the two names.

4.1.5. Ignoring Elements

For a reason or another, some elements in the source files may be unsupported by Yagle or may not respect standard format syntax. To work around those elements, Yagle provides the means to ignore them during the parse of the source netlist. The elements that can be ignored are instances, transistors, resistances and capacitances. For further information please refer to the `inf_DefineIgnore` command documentation.

4.2. General Configuration

4.2.1. Defining Power Supplies

Special attention should be paid to the definition of power supply and ground nodes (`avtVddName`, `avtVssName` and `simPowerSupply` variables). Indeed, the disassembly process is heavily dependant on the naming of those nodes, as the algorithm is looking for current paths towards power supply and ground. Bad specification of these nodes can lead to the construction of an exponential number of wrong current paths. Power supply and ground definition is the first thing to check if the disassembly process seems to loop infinitely.

Yagle also supports V cards for the definition of power supply and ground nodes. One can distinguish between two cases:

The power supply and ground node appear on the interface of the `.SUBCKT`, and the subcircuit is instanciated. The V cards should refer to the names used in the instanciation:

```
Vsupply vdd gnd DC 1.2V
Vground gnd 0   DC 0V

.SUBCKT my_design a b c vdd_int gnd_int
...
.ENDS my_design

X0 a b c vdd gnd my_design
```

The power supply and ground node does not appear on the interface of the `.SUBCKT`, or the subcircuit is not instanciated. The V cards should refer to the names used within the subcircuit, or appearing on the interface of the `.SUBCKT`, together with `.GLOBAL` statements:

```
.GLOBAL vdd gnd

Vsupply vdd gnd DC 1.2V
Vground gnd 0   DC 0V

.SUBCKT my_design a b c vdd gnd
...
.ENDS my_design
```

4.3. Invoking Functional Abstraction

The functional abstraction routine is invoked by the `yagle` command, which takes as argument the name of a sub-circuit. The sub-circuit must be among the previously loaded netlists. If the sub-circuit contains instances it will be flattened to the transistor-level. In such a case, signal naming respects the hierarchical paths. The name of a signal is the concatenation of the names of the successive instances that appear in the hierarchical path leading to the physical node the signal is associated with. The typical Tcl command for invoking functional abstraction is:

```
yagle my_design
```

where `my_design` is the name of the `.SUBCKT` to treat. If flatten is impossible (i.e. transistor level sub-circuits are missing for leaf cells), with no further configuration, the tool will issue an error and exit.

The default configuration creates a VHDL description.

4.4. Timing Back-Annotation

4.4.1. Defining Simulation Temperature

Temperature can be deined either with the `simTemperature` configuration variable or through a `.TEMP` statement in the Spice file.

4.4.2. Back-Annotation Level

The level of accuracy used for the back-annotation is `yagleTasTiming`

4.5. Output Files

4.5.1. CNS, CNV files

The `.cns` file describes the partitions (cones), and their interconnections, resulting from the disassembly process. This file is very useful for debugging purposes, and necessary for the spice deck generation of timing paths. The file can be generated with the following configuration:

```
avt_config yagleGenerateConeFile yes
```

The `.cns` file is intented to be re-read by Yagle and therefore is not very human-readable. A more friendly version can be generated by setting:

```
avt_config avtVerboseConeFile yes
```

4.5.2. VHDL and Verilog files

VHDL file is generated in default mode. a Verilog file can be generated instead by using the following configuration:

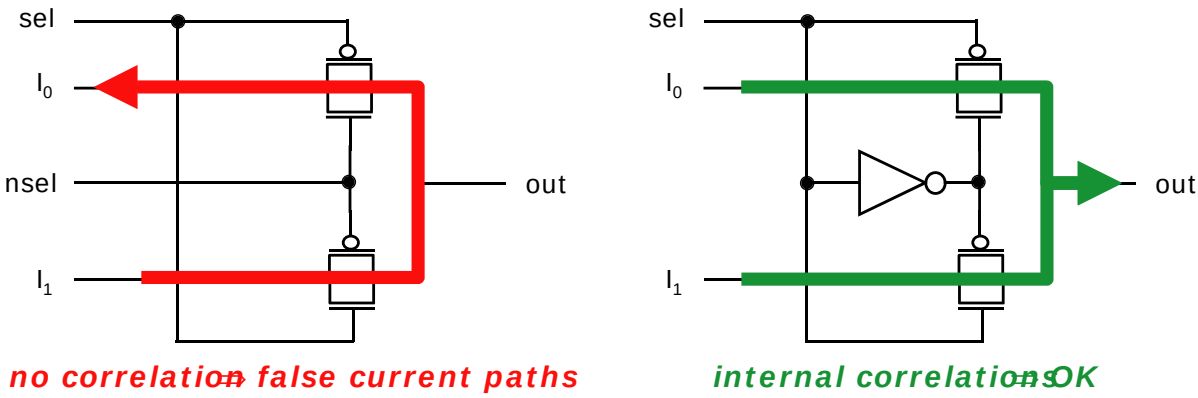
```
avt_config avtOutputBehaviorFormat vlg
```

4.6. Special Elements

4.6.1. Transmission Gate Multiplexers

The detection of multiplexors is done purely algorithmically. The cone partitioning strategy implemented in Yagle perfectly fits with the detection and modeling of transmission-gate based multiplexers, provided that the correlations between the commands can be resolved within the design. The only reason why detection may fail, is because the schematic of the design itself prevents to identify those correlations, for example when commands are input pins. In such a case, correlations (mutual exclusion) should be set externally with INF commands.

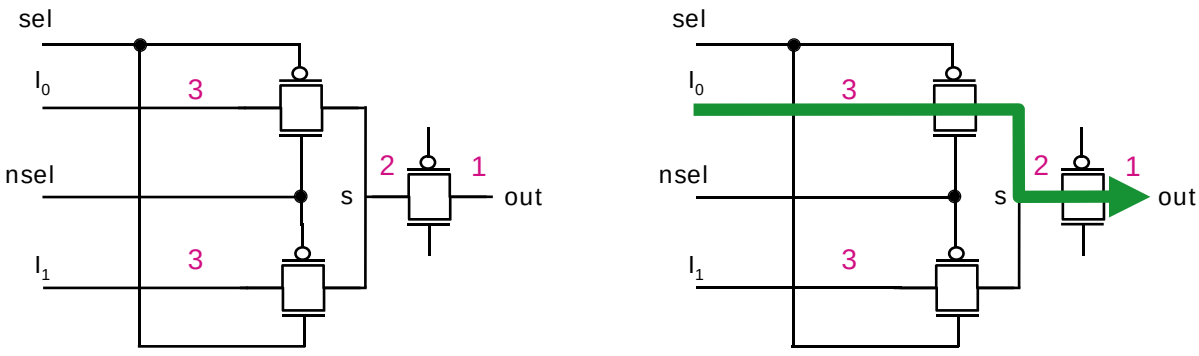
The following diagram shows two situations. In the left-hand design, the mutual exclusion between `sel` and `nsel` is not ensured by the design. There is no way for the tool to identify inputs and outputs, and it constructs false current paths. In the right-hand design, the mutual exclusion between `sel` and `nsel` is ensured by the invertor, and therefore the tool correctly models the multiplexer.



To avoid the construction of false current paths in the left-hand design, the following mutual exclusion configuration should be set:

```
inf_DefineMutex cmpUP {sel nsel}
```

If the transmission gate toppology is more complex, and setting of mutual exclusion constraints become too much difficult, another orientation mechanism is available. Let's consider the next design:



Here orientation can be done by setting levels on signals `i0`, `i1`, `s` and `out`. The transistors are oriented by assuming the current is going from the signals with the higher level to the signals with the lower level. Levels should be set as follow:

```
inf_DefineDirout i0  3
inf_DefineDirout i1  3
inf_DefineDirout s   2
inf_DefineDirout out 1
```

The default orientation value of signals is `-1`.

4.6.2. Latches

Two algorithms exist in Yagle to detect latches. The first one is based on pattern-matching. The tool tries to match in the design built-in latch patterns. This algorithm is enabled with the `yagleStandardLatchDetection` (default behavior is enabled). The second algorithm is based upon the Boolean analysis of gate loops and an electrical analysis of conflicts. This algorithm is enabled with the `yagleAutomaticLatchDetection` variable (default behavior is not enabled). The two algorithms can be enabled together, in such a case the standard latch detection is performed before the automatic latch detection.

4.6.3. Dynamic Latches

Dynamic latches are typically tristate nodes followed by a capacitance. In default mode, tristate nodes are not marked as latches. This behavior can be changed with the `yagleMarkTristateMemory` configuration variable.

Dynamic latches can also be identified with the INF commands `inf_DefineDLatch` and `inf_DefineNotDLatch`

4.7. Case Analysis

Case analysis, such as Scan Mode analysis or Functional mode analysis, is available in the Yagle platform. It is performed by sticking input connectors or internal signals to logical low or logical high values. It is done by adding in the Tcl script the SDC command `set_case_analysis`.

The logical value stuck on the input connector or logical signal is propagated through the design, with regard to the behavior of the gates it crosses. A report of the stuck signals is available in the `.rep` file:

```
[WRN 30] Signal 'ram_na3' is stuck at Zero
[WRN 31] Signal 'ram_a43r_net6' is stuck at One
```

Chapter 5. Using The XYagle GUI

5.1. Presentation of the XYagle Interface

XYagle contains five pull down menus:

- File
- Edit
- View
- Windows
- Options

These menus grant accesss to the XYagle functionality.

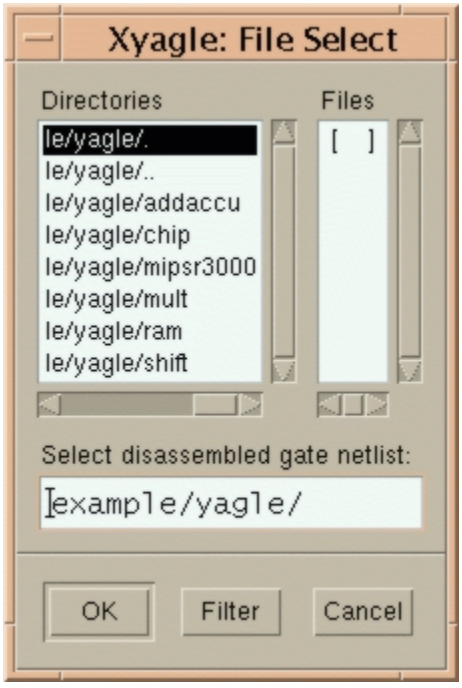
Each menu can be tear of from the main XYagle window in order to allow the user to accesss it directly.

5.1.1. The File Menu

The `File` menu provides accesss to the "Open" and "Quit" functionalities of the XYagle interface. The functionalities of the `File` menu are detailed in the following sections.

Open...

By selecting the `Open...` item of the `File` menu you will be presented with the `XYagle File Select` dialog. Choose which schematic to load by selecting it in this dialog.



Disassemble...

The circuit disassembling can be launched from within XYagle. You will need to read the Yagle User Guide before using this option.

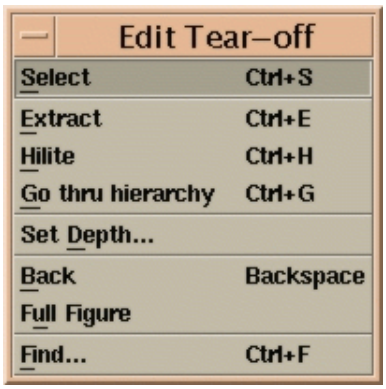
More information about performing the disassembling can be found in the `Disassembling with XYagle` Chapter of the current documentation.

Quit

Selecting this item in the `File` menu will exit XYagle.

5.1.2. The Edit Menu

The `Edit` menu provides accesss to the most current functionnalities of XYagle.



Most of these options will be detailled in following chapters. A quick presentation are given in the following sections:

Extract

Select the `Extract` option in order to switch the extract sub-netlist mode on. More information on modes can be found in the `Schematic Browsing with XYagle` chapter

Highlight

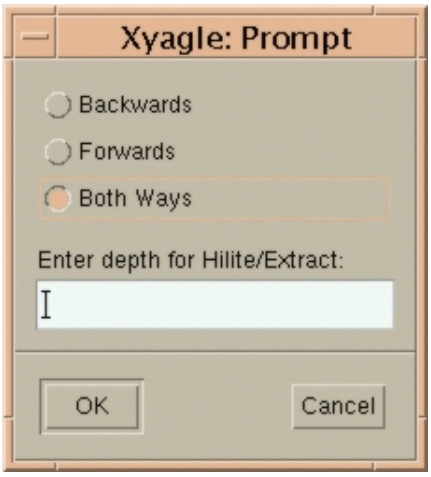
Select the `Highlight` option in order to switch the hiliting gate dependences mode on. More information on modes can be found in the `Schematic Browsing with XYagle` chapter

Go thru hierarchy

Select the `Go thru hierarchy` option in order to switch the traversing hierarchy mode on. More information on modes can be found in the `Schematic Browsing with XYagle` chapter

Set Depth...

By selecting the `Set Depth...` item of the `Edit` menu you will be presented with the `XYagle: Prompt` dialog.



Set the depth value before using the Extract or Highlight modes.

Back

By selecting the `Back` item of the `Edit` menu, you will return to the previous display configuration.

Full Figure

By selecting the `Full Figure` item of the `Edit` menu, you will return to the main figure. This option is used to return to the figure after using options that display sub-netlist of the full figure.

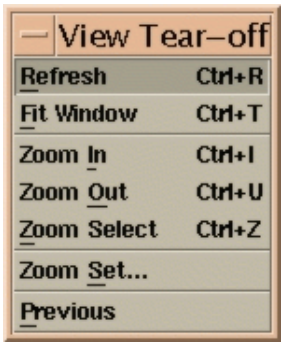
Find...

By selecting the `Find...` item of the `Edit` menu you will be presented with the `XYagle: Prompt` dialog.

This option allows you to search object by name in the current schematic.

5.1.3. The View Menu

The `View` menu provides accesss to a number of options affecting the display of the selected schematic.



The items of this menu have explicit names:

Refresh Allows the user to refresh the display.

Fit window Allows the user to display all the schematic in the XYagle main window at the more effective scale.

Zoom In Allows the user to zoom in and to take a closer look to the schematic elements.

Zoom Out Allows the user to zoom out and to see the surrounding elements of the schematics.

Zoom Select Allows the user to

Zoom Set Allows the user to set the zoom factor. the value is a percentage between 5 and 95%.

Previous Allows the user to

Applications of these options will be used in the following chapters.

5.1.4. The Windows Menu

The `Windows` menu provides access to more specific XYagle functionalities.

These options grant access to information about the schematic and the elements of a disassembled netlist. This section provide an overview of these options. complete information can be found in the `XYagle Basics` and `Disassembled Netlist Information` chapters.

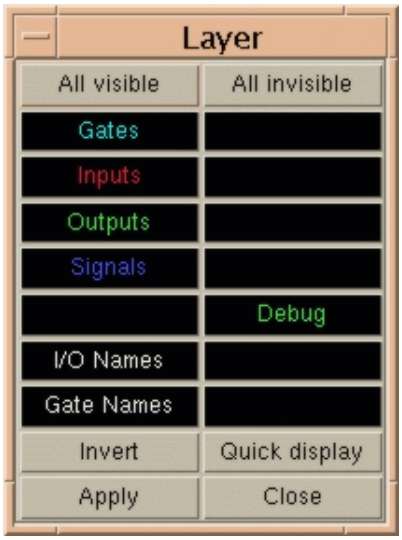
Show Structure Allows the user to see the structure of the next selected gate.

Show Behavior	Allows the user to see the behavior of the next selected gate.
Show Message	Allows the user to view XYagle messages.
Show Info	Allows the user to display general information about the schematic such as the figure name.

5.1.5. the Options Menu

The `Options` menu provides accesss to some basic configuration of the visibility of elements displayed in the XYagle interface.

By selecting the `Layer` item of the `Options` menu, you will then be presented with the `Layer` dialog:



The `Layer` dialog allows the user to configure the visibility of the schematic.

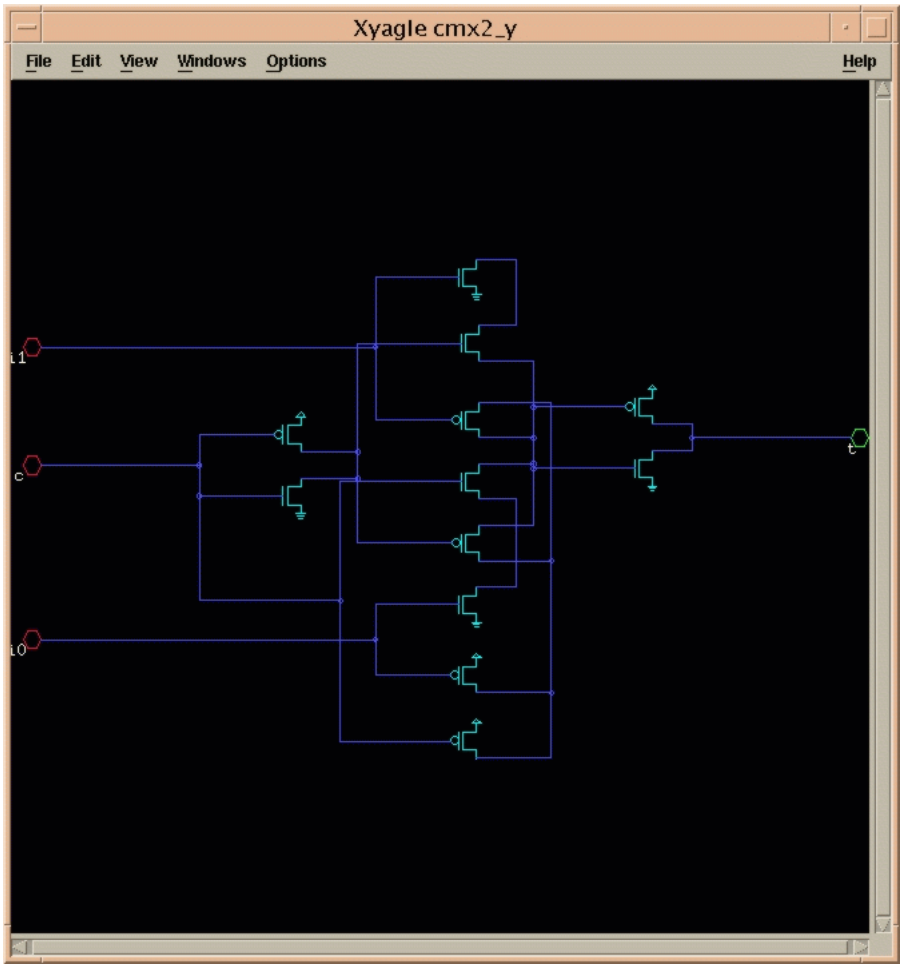
5.2. Loading the Schematic

The XYagle graphical interface allows the user to displays different type of circuit, from transistor level schematic to hierarchical gate netlist.

To load a schematic use the `Open` item in the `File` menu.

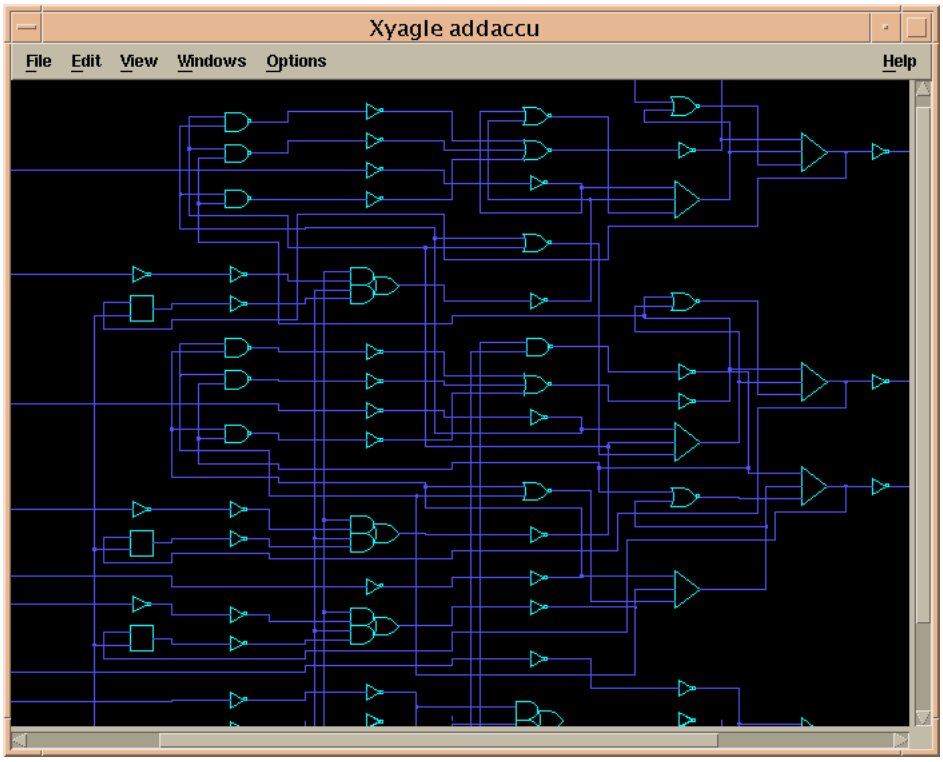
5.2.1. transistor Level Schematic

XYagle displays transistor Level Schematic as a network of symbols representing transistors.



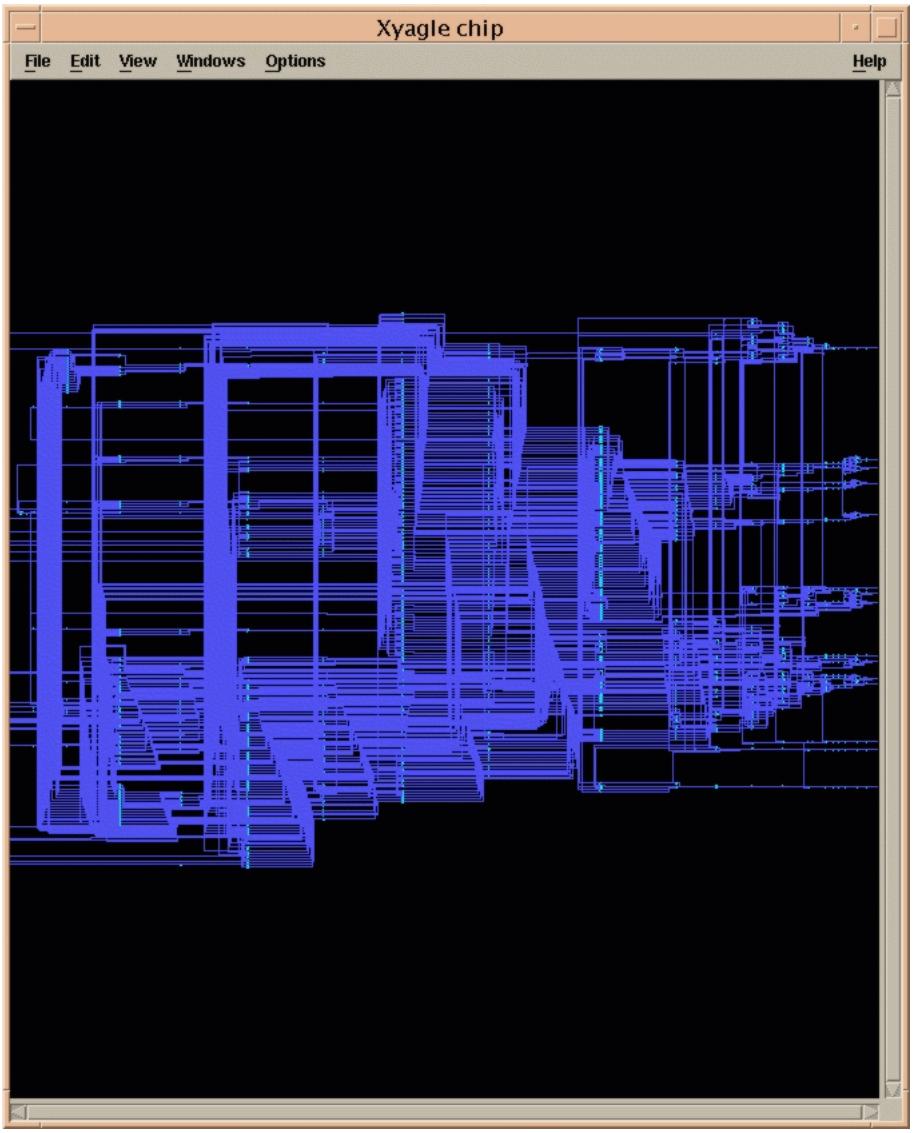
5.2.2. Gate Netlist

XYagle displays gate netlist as network of logical gates.



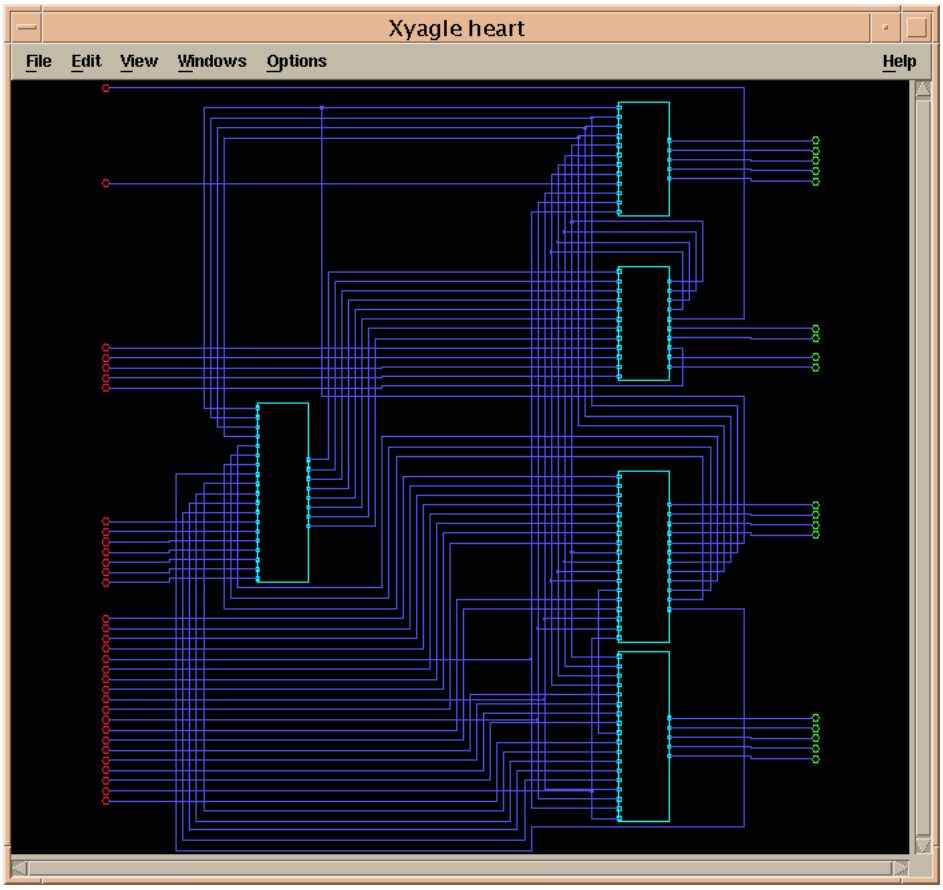
5.2.3. Disassembled Gate Netlist

XYagle display disassembled gate netlist as



5.2.4. Hierarchical Gate Netlist

XYagle displays hierarchical gate netlist as a network of gates and blocs.



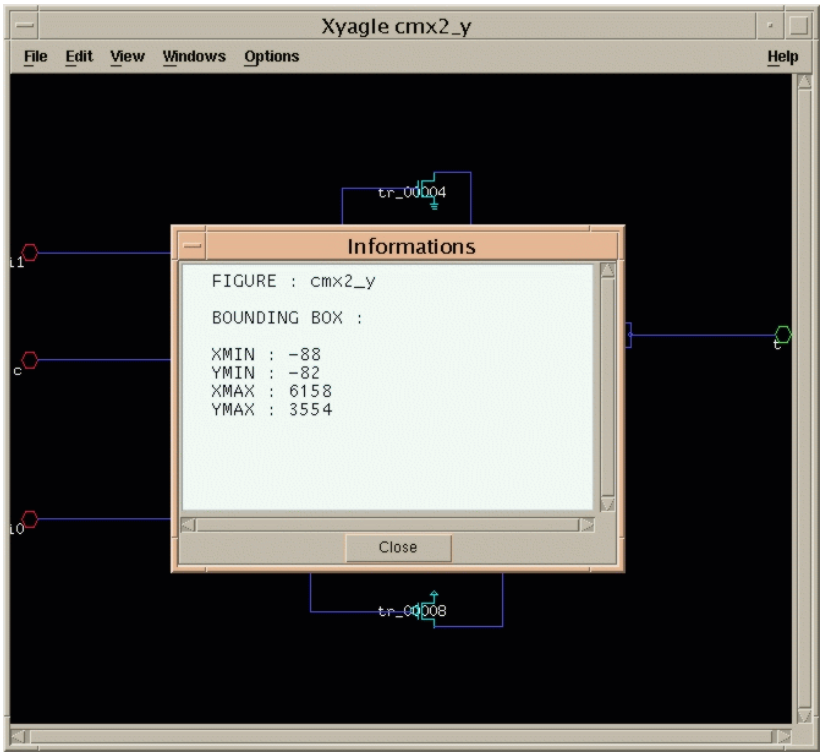
5.3. XYagle Basics

This section explains how to obtain information about the figure loaded in XYagle and how to use the schematic in good conditions of visibility.

5.3.1. Viewing General Information

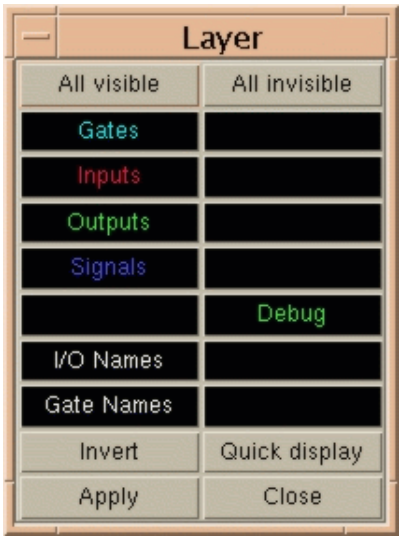
To view the general information use the `Show Info` item in the `Windows` menu.

This opens a window that display the name of the loaded figure and the dimensions of the schematic.



5.3.2. Configuring Visibility

To configure the visibility use the `Layer` item in the `Options` menu.

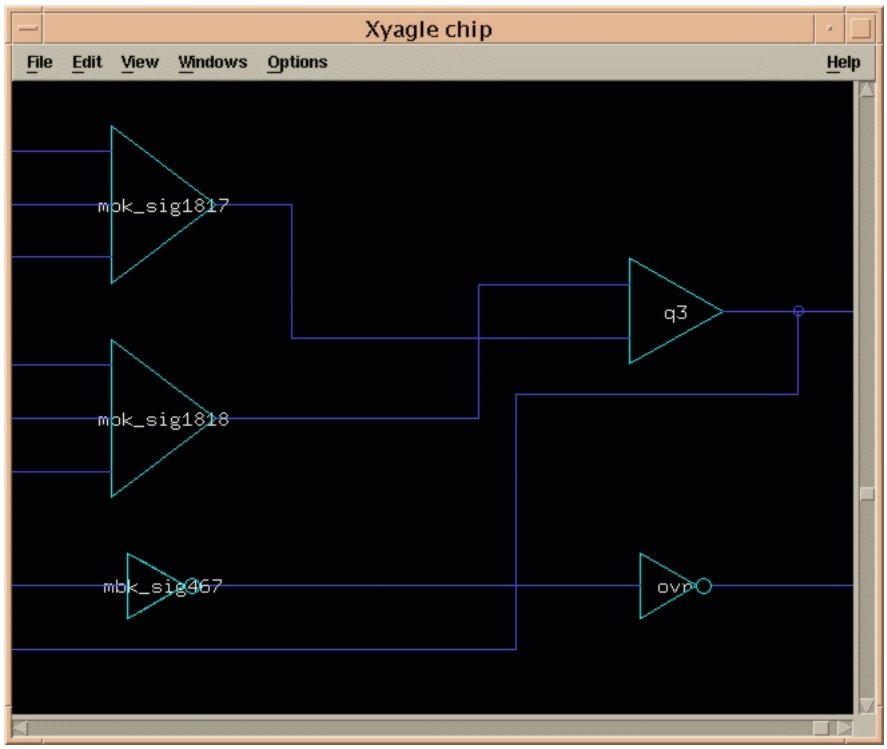


XYagle display objects on layers, select on the left column the layers you want to be visible, and select on the right column the layers you want to be invisible. Then valid your choice with `Apply`.

5.3.3. Navigation in XYagle

To navigate in XYagle, use the `View` menu.

Use he `Zoom In` item of this menu to have a good visibility of the schematic.



Use he `Zoom Out` item of this menu to have a larger visibility of the schematic or to `Fit Window` display the full schematic on the display.

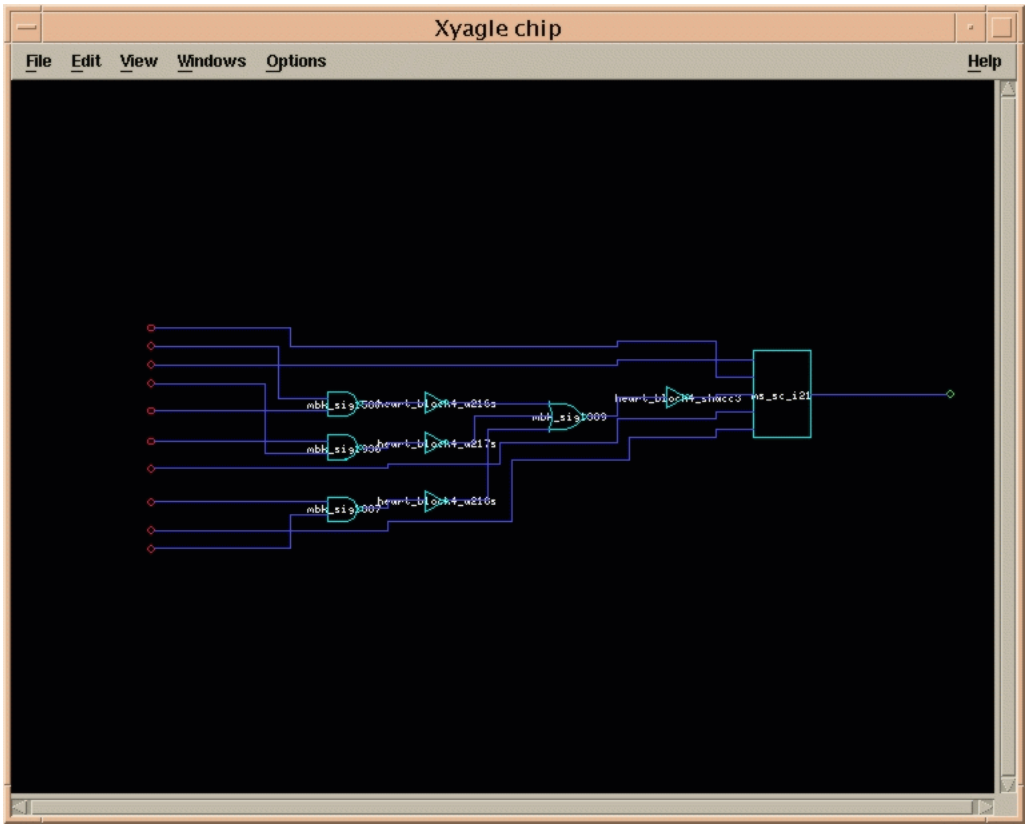
5.4. Schematic Browsing with XYagle

5.4.1. XYagle Browsing Modes

To unselect a XYagle browsing mode use the `Select` item in the `Edit` menu.

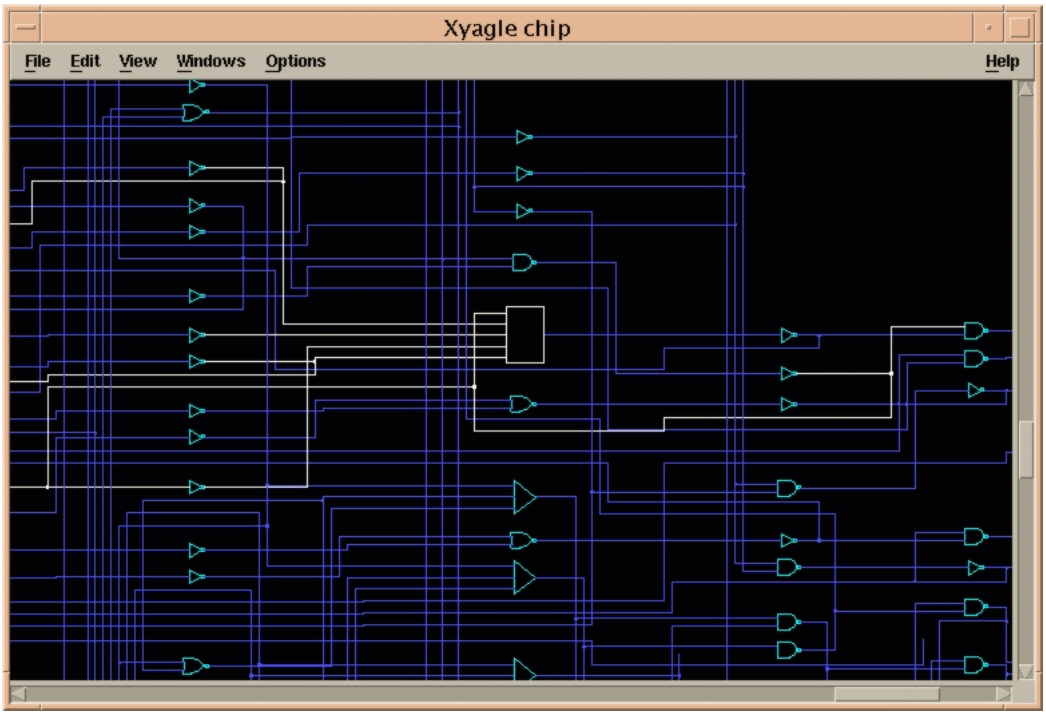
5.4.2. Extracting Sub-Netlists

To extract sub-netlists use the `Extract` item in the `Edit` menu.



5.4.3. Highlighting Gate Dependences

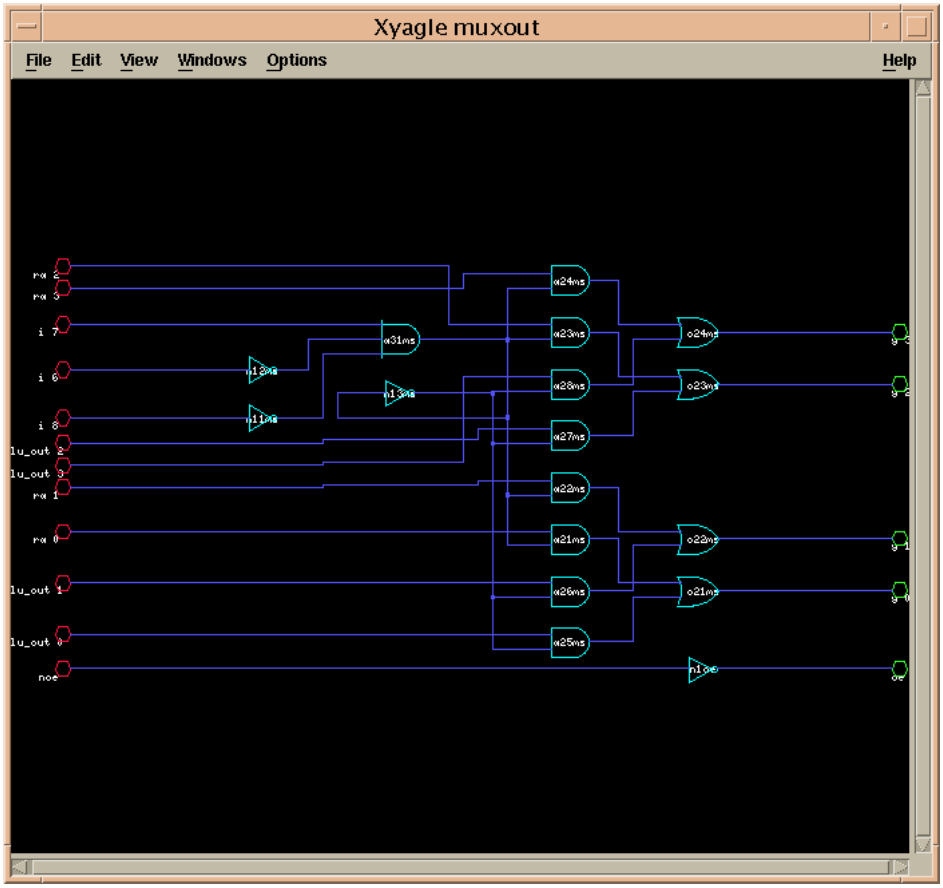
To highlight gate dependences use the `Highlight` item in the `Edit` menu.



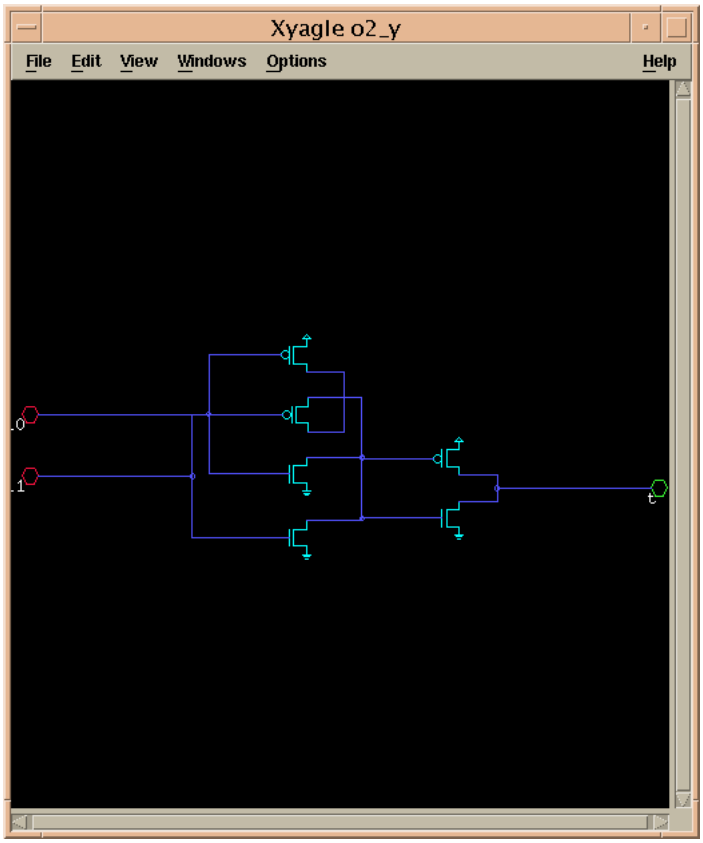
5.4.4. Traversing Hierarchy

To traverse hierarchy use the `Go thru hierarchy` item in the `Edit` menu

.
traversing hierarchy allows the user to display the bloc sub-netlist. for example going thru the hierarchy of the muxout bloc from the previously loaded hierarchical gate netlist display the gates of the bloc.



It is possible to go thru the hierarchy of a gate to display the transistor level of the gate. For exemple, going thru the hierarchy of a OR gate from the previous muxout display the following network of transistor:



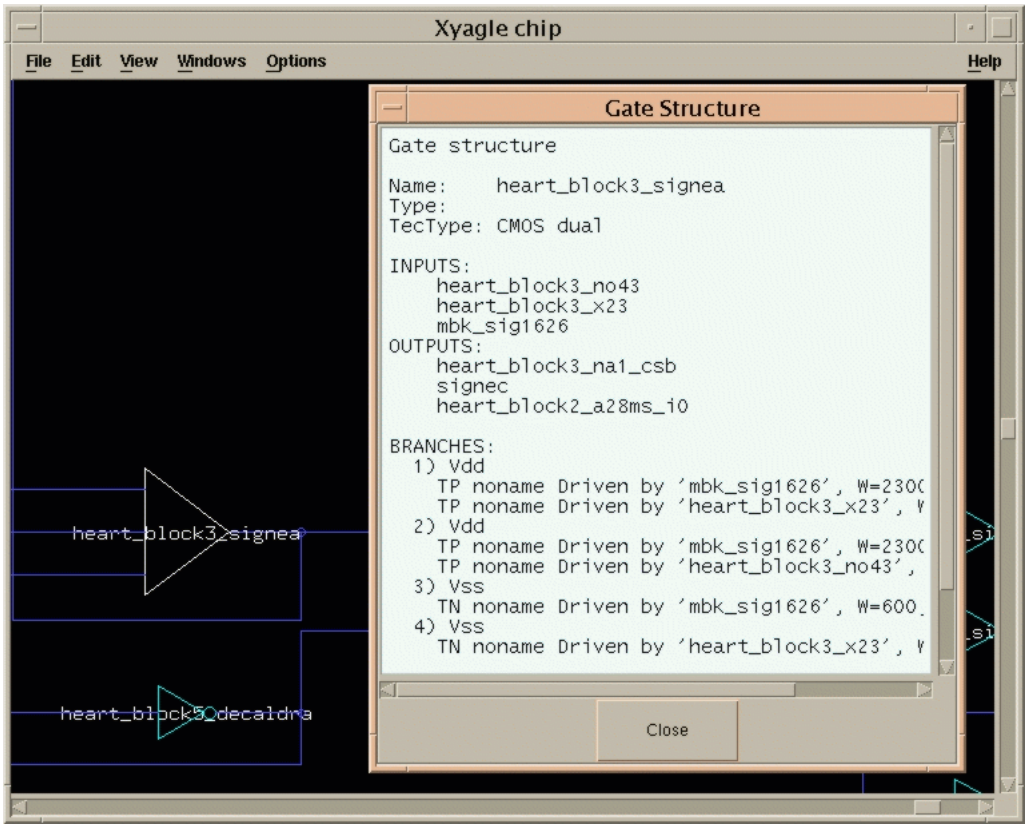
5.4.5. Searching Object by Name

To search object by name use the `Find...` item in the `Edit` menu.

5.5. Disassembled Netlist Information

5.5.1. Viewing the Gate Structure

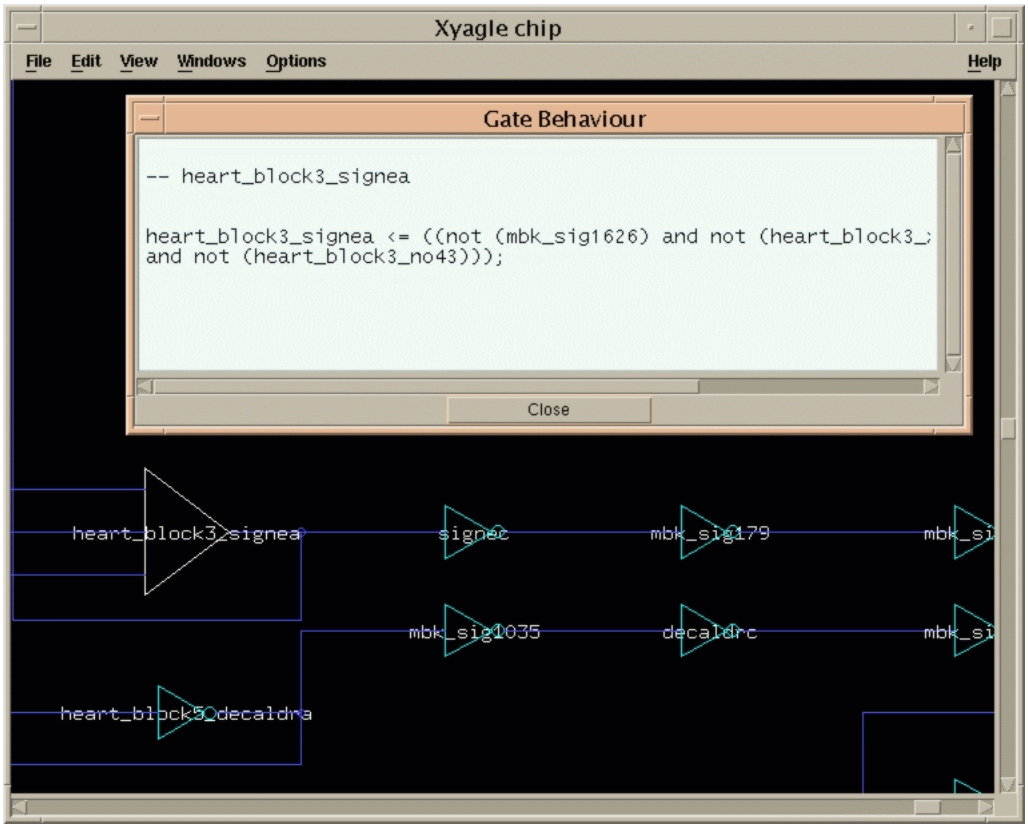
To view the gate structure use the `Show Structure` in the `Windows` menu



The information provided by `Show Structure` on a selected gate structure contains the name of the gate, the inputs and output signals.

5.5.2. Viewing the Gate Behavior

To view the gate behavior use the `Show Behavior` in the `Windows` menu



`Show Behavior` provides the logical dependence between input and output signals of the selected gate.

Index

No index for this document.