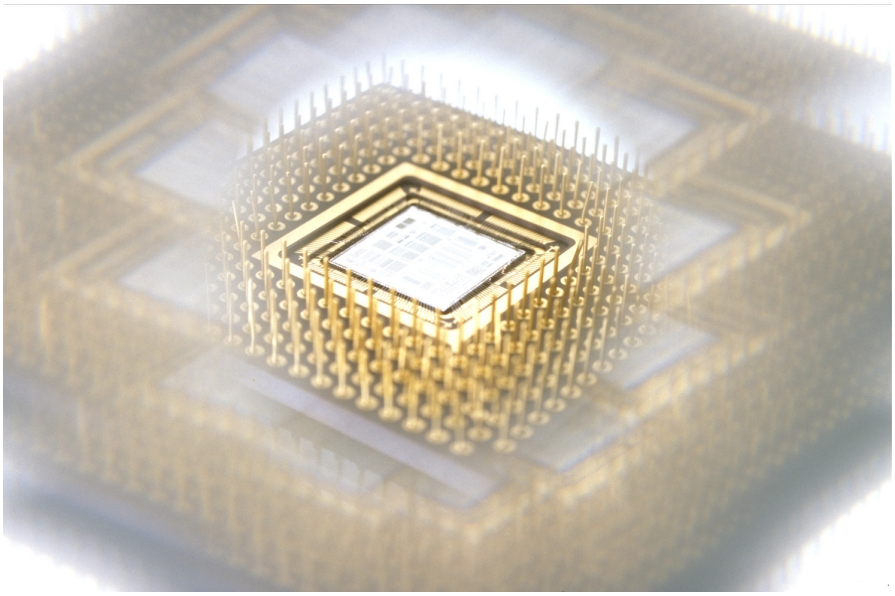
Avertec Tools

HITAS
User Guide



Software Release 3.4p5

June 7th, 2010

About this document

This document explains:

- The tool capabilities and typical applications
- The internal structure of the tool
- How the tool integrates in custom design flows
- How to perform static timing analysis at transistor-level
- How to perform crosstalk analysis at transistor-level
- How to perform timing abstraction at transistor-level
- How to deal with big designs

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. Software Installation 9
1.1. System Requirements 9
1.2. What the Distribution Provides 9
1.3. Scope of the Installation 9
1.4. Performing the Installation 9
1.5. Setting-up the Environment 11
1.6. The FLEXLM Licence Server 12
2. Overview 13
2.1. Static Timing Analysis 13
2.2. Signal Integrity Analysis 13
2.3. Applications 13
2.4. Key Features 14
3. Theory Understanding 15
3.1. Principles 15
3.2. Timing Database Generation 16
3.2.1. MOS Characterization 16
3.2.2. Netlist Disassembly 16
3.2.3. Timing Arcs 17
3.2.4. Timing Models 18
3.2.5. Timing Paths 18
3.3. Timing Database Analysis 18
3.3.1. Database Analysis Flow 18
3.3.2. Static Timing Analysis 19
3.3.3. Crosstalk Analysis 19
3.3.4. Path Searching 20
3.3.5. Timing Abstraction 20
4. Scope of Usage 21
4.1. Introduction 21
4.2. HITAS Basic Assumptions 21
4.2.1. Circuit Partitioning 21
4.2.2. Timing Arcs 22
4.2.3. Current Characterization 22
4.2.4. Algorithm Assumptions 22
4.3. HITAS Digital Structures 23
4.3.1. CMOS Gates 23
4.3.2. Pass-Transistor and Transmission Gate Logic 23
4.3.3. Clocked CMOS Logic 23
4.3.4. Static Latches and Flip-Flops 24
4.3.5. Dynamic Latches 25
4.4. HITAS Analog Structures 25

4.4.1. Sense Amplifier 25
4.4.2. Differential Amplifier 26
4.4.3. Voltage Generator 26
4.4.4. Typical Analog Devices 26
5. Design Flow Integration 27
5.1. Transistor-Level Analysis 27
5.2. Full-Chip Analysis 27
5.3. Input Files 28
5.3.1. Netlist 28
5.3.2. Parasitics 28
5.3.3. Technology 28
5.3.4. Timing characterizations 29
5.3.5. Timing Constraints 29
5.4. Output Files 29
5.4.1. Disassembly 29
5.4.2. Timing Database Generation 29
5.4.3. Static Timing Analysis 29
5.4.4. Crosstalk Analysis 30
5.4.5. Abstraction 30
6. Using Tcl Interface 31
6.1. Script Launch 31
6.2. Tools Configuration 31
6.3. Error Policy 31
6.4. Objects 32
6.5. Functions 33
6.6. INF Configuration and SDC Support 33
7. Timing DB Construction 35
7.1. File Loading 35
7.1.1. Transistor Technology Models 35
7.1.2. Input Netlist 36
7.1.3. Parasitics 37
7.1.4. Vectorization 38
7.1.5. Ignoring Elements 38
7.2. DB Construction 39
7.2.1. Defining Power Supplies 39
7.2.2. Defining Simulation Thresholds 39
7.2.3. Defining Simulation Temperature 39
7.2.4. Invoking DB Construction 40
7.3. Output Files 40
7.3.1. REP file 40
7.3.2. LOOP file 40
7.3.3. CNS, CNV files 40
7.3.4. DTX and STM files 41
7.4. Latch Detection and Modeling 41
7.4.1. Detection Sequences 41
Manual Identification 41

Simple Detection 41
Automatic Detection 41
Dynamic Latches Detection 42
7.4.2. Enabling Detection Sequences 42
7.5. Static Latch Modeling 42
7.5.1. Asynchronous Set and Reset 43
7.5.2. Manual Configuration 43
7.5.3. Intrinsic Setup and Hold 44
Intrinsic Setup 44
Intrinsic Hold 44
7.6. RS-Latches 45
7.6.1. Modeling of NOR-based structures 47
All States Allowed 47
Legal States Only 48
7.6.2. Modeling of NAND-based structures 48
All States Allowed 48
Legal States Only 48
7.6.3. Fine Tuning 49
7.6.4. Manual Tuning 49
7.7. Symmetric Latches 49
7.7.1. Symmetric Pulldown 50
Typical Structure 50
Latch Nodes and Commands 50
Timing Arcs 50
7.7.2. Symmetric Bitcell 51
Typical Structure 51
Latch Nodes and Commands 51
Timing Arcs 51
7.7.3. Asymmetric Pulldown 52
Typical Structure 52
Latch Nodes and Commands 52
Timing Arcs 52
7.8. Dynamic Latches 53
7.9. Special Elements 53
7.9.1. Transmission Gates 53
7.9.2. Transmission Gate Multiplexers 54
7.9.3. Domino Precharge 55
7.10. Case Analysis 56
7.11. Integrating External Timing Abstractions 56
8. Timing DB Browsing 57
8.1. Timing DB 57
8.2. Details Browsing 58
8.3. STA Browsing 58
9. Static Timing Analysis 60
9.1. Performing the Analysis 60
9.2. Output Files 61

9.3. Tcl Reports 61
9.4. Timing Checks 61
9.4.1. Input to Latch 61
Inputs Specifications 61
Timing Checks Description 62
Setup Slack 62
Hold Slack 63
9.4.2. Latch to Latch 63
Timing Checks Description 63
Setup Slack 64
Hold Slack 64
9.4.3. Latch to Output 65
Output Constraints 65
Setup Slack 66
Hold Slack 66
9.5. Skew Compensation 67
9.6. Multicycle Paths 68
9.7. Tips 69
9.7.1. Disabling Master-to-Slave Timing Checks 69
9.8. On-Chip Variation 70
9.9. Clock Schemes Handling 70
9.9.1. Clock Dividers 70
9.9.2. Pulse Generators 70
9.9.3. RS-based Clock Generators 70
10. Crosstalk Analysis 71
10.1. Requirements 71
10.2. Understanding Crosstalk in STA 71
10.2.1. The Issues Involved 71
10.2.2. Algorithm 71
10.2.3. Delay Calculation 73
10.2.4. Noise Calculation 74
10.3. Running the Crosstalk Analysis 75
10.4. Output Files 76
10.5. Browsing Crosstalk Analysis Results 76
10.5.1. Crosstalk Impact on Delays 76
10.5.2. Crosstalk Noise 77
10.5.3. Browsing Information on Event 77
10.5.4. Browsing Local Crosstalk Impact on Delay 78
10.5.5. Browsing Aggressor 78
10.6. Score-Based Result Analysis 78
11. Spice Deck Generation 79
11.1. Simulator Configuration 79
11.2. Spice Deck Generation 80
11.3. Spice Deck Simulation 80
11.4. Out-of-path Transistors 81
12. Analog Sub-circuit Characterization 82

12.1. Objective 82
12.2. Pre-Characterization 82
12.2.1. Database Construction 83
12.2.2. Simulator Linking 83
12.2.3. Hierarchical Netlist Integration (Pre-Layout) 83
12.2.4. Flat Netlist Integration (Post-Layout) 84
12.2.5. Netlists Consistency 84
12.3. On-the-Fly Characterization 84
12.3.1. Database Construction 85
12.3.2. Hierarchical Netlist Integration (Pre-Layout) 85
12.3.3. Flat Netlist Integration (Post-Layout) 86
13. Timing Characterization (.lib) 87
13.1. Setup and Hold Constraints Formulas 87
13.1.1. Setup Correction 88
13.1.2. Hold Correction 88
13.2. Performing the Characterization 88
13.3. Advanced Configuration 89
13.3.1. Input Slope and Output Load Axis 89
13.3.2. Capacitances in the .lib file 89
13.4. Cell Library 89
14. Using the GUI 91
14.1. Timing Database Browsing with XTAS 91
14.1.1. Overview 91
14.1.2. Description 91
14.1.3. Execution 91
XTAS Command 91
XTAS Resource File 92
XTAS Splash Screen 92
14.2. Configuration 92
14.2.1. Memory Size 92
14.2.2. Toolbar Buttons 93
14.2.3. Display Type 93
14.3. Loading the Timing Database 94
14.3.1. Timing Database 94
14.3.2. Crosstalk Timing Database 94
14.4. Accessing XTAS Features 95
14.4.1. Exiting XTAS 95
14.4.2. Browsing Connectors 95
14.4.3. Browsing Registers 95
14.4.4. Browsing Commands 95
14.4.5. Browsing Precharges 96
14.4.6. Browsing Break Points 96
14.4.7. Browsing Internal Signals 96
14.4.8. Browsing Paths 96
14.4.9. Browsing Delays 97
14.4.10. Stability Analysis 97

14.4.11. Common applications 97
14.5. Browsing Timing Signals 98
14.6. Browsing Critical Paths 99
14.6.1. General Procedure 99
14.6.2. Options 100
14.6.3. Critical Paths Display 101
14.7. Browsing Near Critical Paths 102
14.7.1. Overview 102
14.7.2. Options 102
14.7.3. Near Critical Paths Display 103
14.8. Browsing Path Details 103
14.8.1. Overview 103
14.8.2. Options 104
14.8.3. Delay Display 105
14.9. CPE Path Simulation 105
14.9.1. Overview 105
14.9.2. Options 106
14.9.3. Simulation Path Display 107
14.10. Path Visualization 108
14.10.1. Overview 108
14.10.2. Options 108
14.10.3. Path Visualization Display 108
14.11. Browsing Delays 109
14.11.1. Overview 109
14.11.2. Options 109
14.11.3. Delay Display 110
14.12. Static Timing Analysis and Signal Integrity 111
14.12.1. Overview 111
14.12.2. Static Timing Analysis Results 111
Launching the Analysis 111
Loading Switching Windows 112
Crosstalk Analysis Parameterization 112
Stability Parameterization 113
The Crosstalk Analysis Results 115
The Violating Signals Display 115
14.12.3. Noise Analysis 116
Overview 116
Analysis 116
Noise Analysis Results 117
Scores Configuration 119
Crosstalk Information 119
15. Managing Big Designs 122
15.1. File Compression and Disk Caching 122
15.2. Information removal 122
15.3. Object sharing 122
15.4. Execution Speed-up 123

16. Glossary 124
16.1. Logical Description 124
16.2. Physical Description 124
16.3. Timing Description 124
Index 126

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

HITAS 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. Static Timing Analysis

The advent of semiconductor fabrication technologies now allows high performance in complex integrated circuits.

With the increasing complexity of these circuits, static timing analysis (STA) has revealed itself as the only feasible method ensuring that expected performances are actually obtained.

In addition, signal integrity (SI) issues due to crosstalk play a crucial role in performance and reliability of these systems, and must be taken into account during the timing analysis.

However, performance achievement not only lies in fabrication technologies, but also in the way circuits are designed. Very high performance designs are obtained with semi or full-custom designs techniques.

The HITAS platform provides advanced STA and SI solutions at transistor level. It has been built-up in order to allow engineers to ensure complete timing and SI coverage on their digital custom designs, as well as IP-reuse through timing abstraction.

Furthermore, hierarchy handling through transparent timing views allows full-chip verification, with virtually no limit of capacity in design size.

2.2. Signal Integrity Analysis

HITAS crosstalk engine is coupled with static timing analysis (STA) engine and is based upon a multi-switching windows refinement algorithm.

Crosstalk effects are then fully handled in timing checks. Precise delay update is done thanks to current source models for gate, and non-linear charge transfer models for effective wire load computation.

Detailed SI report provides:

- Delta-delays in gates and interconnects
- Effective noise contribution of each aggressor
- Overshoot and undershoot peaks, together with sensitivity of the fan-out gates
- Statistical noise classification allowing Engineering Change Orders

The graphical user interface (GUI) allows easy execution of the crosstalk analysis.

2.3. Applications

HITAS can be used on a wide range of ICs, such as Micro-Processors, Micro-Controllers, Memory Controllers, Custom IPs or Standard-Cell Libraries (Arithmetic Units, Data-Paths...). It provides the following benefits:

Timing sign-off

- Digital custom macros STA sign-off
- Full-chip STA sign-off

Signal Integrity sign-off

- Digital custom macros crosstalk analysis
- Full-chip crosstalk analysis

Design and debug

- Circuit timing analysis during design phase
- Early detection of timing bottlenecks

IP Reuse (.lib files generation)

- Digital custom macros characterization
- Standard cells re-characterization

2.4. Key Features

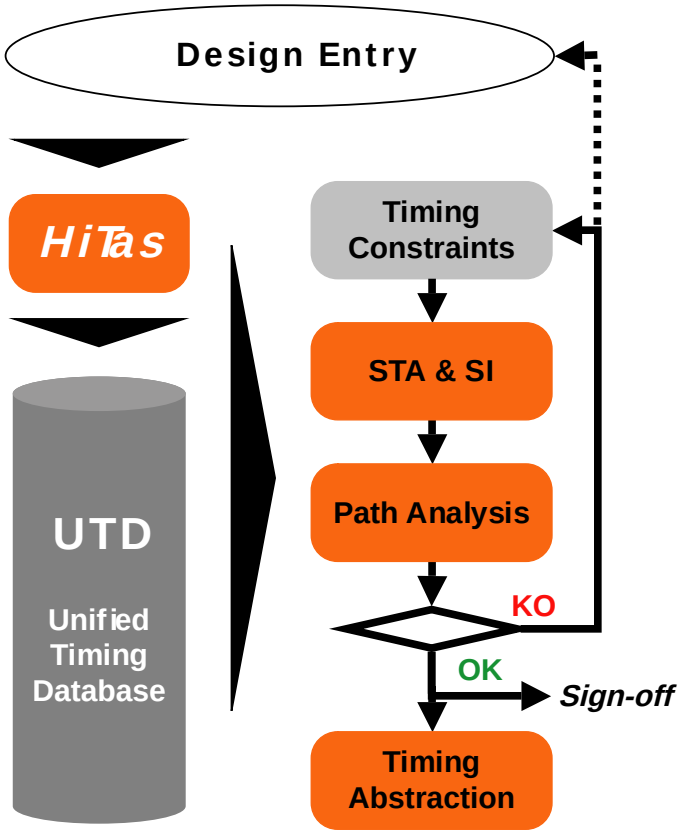
HITAS provides the following features:

- Vector-free fast simulation engine, accuracy within 5% of SPICE
- Current Source Modeling (CSM) of drivers, AWE modeling of interconnects
- SPICE, VHDL or VERILOG netlist support, SPICE, SPEF or DSPF parasitics support, BSIM3 and BSIM4 support
- Design-style support: latch-based, domino logic, barrel shifters, multipliers
- Advanced timing checks: cycle sharing, cycle stealing, multicycle paths
- Full-Chip analysis through transparent hierarchy
- Cross corner analysis
- STA and SI coupled analysis
- Non-linear noise models
- Statistical noise classification
- GUI and Tcl interface
- Critical path automatic spice deck
- SDC timing constraints

Chapter 3. Theory Understanding

3.1. Principles

The way HITAS performs timing analysis of integrated circuits can be described in two main steps. The first step is the generation of a Unified Timing Database (UTD) from the design entry (left-hand side of the following diagram). The second step is the exploitation of the database, in order to perform Static Timing Analysis, Signal Integrity Analysis, and Timing abstraction (right-hand side of the following diagram).



The timing database represents the intrinsic timing characteristics of the design, for a given corner, or multi-corner configuration (processes, voltages and temperatures). Those characteristics are such as gate timing arcs, gate and interconnect delay models, timing paths.

3.2. Timing Database Generation

3.2.1. MOS Characterization

At transistor-level, HITAS works either on a flat transistor netlist obtained by a standard extractor, or on a hierarchical netlist, together with the transistor description of the leaf cells. In the last case, HITAS flattens the netlist to the transistor level.

The first phase of the database generation consists in the electrical characterization of the MOS transistors building up the design. The process of characterization is the reduction and optimization of the generic BSIM equations for each instance of a transistor, with regard to local and global parameters.

The local parameters are the instance-specific parameters, such as length (L), width (W), geometry (NF), stress effect (SA, SB, SD), well proximity effect (MULU0, DELVT0), local drain/source resistances (NRD, NRS) or local power supply voltage value.

The global parameters are the Process, Voltage (nominal) and Temperature conditions.

For given PVT conditions, each transistor is then associated with an instance-specific electrical model, which in turn contains instance-specific optimized equations:

- Ids = f (Vgs, Vds)
- Qd = f (Vgs, Vds)

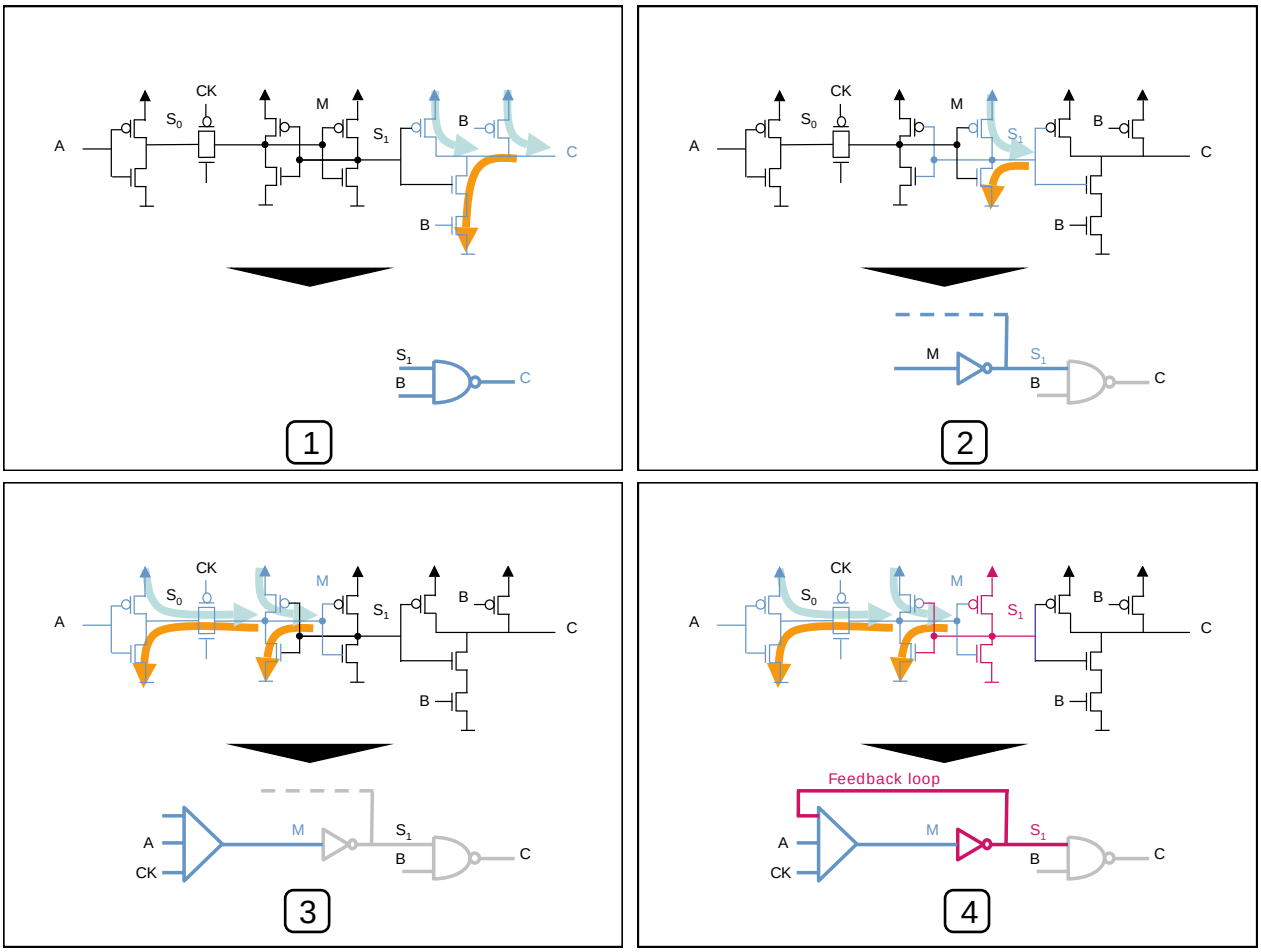
The model is only valid for the transistor referring to it. However, in order to save memory, a sharing mechanism allows transistors that have identical (or close) instance-specific parameters to refer to the same electrical model. Note that a transistor may be characterized for several PVT conditions, and then may be associated to several models.

3.2.2. Netlist Disassembly

The second phase of the database generation consists in partitioning the transistor netlist. This phase uses a procedure called circuit disassembly in order to automatically extract an oriented gate netlist from the transistor netlist, using a strict minimum of a priori knowledge of the circuit structures.

The starting point of the partitioning strategy is the identification of the nodes on which to build a sub-network. The innovation of HITAS disassembly is to build sub-networks between which there is no charge transfer. Therefore, the frontier of a sub-network is the set of nodes that control the gates (insulating polysilicon) of its transistors. A sub-network is then extracted for all the nodes in the netlist that control at less one transistor gate, by following source-drain connections.

The extracted sub-networks are called cones. The construction of a cone on a node N consists in identifying all the current paths between the node N and a voltage source (Vdd or Vss), as illustrated in the following diagram.

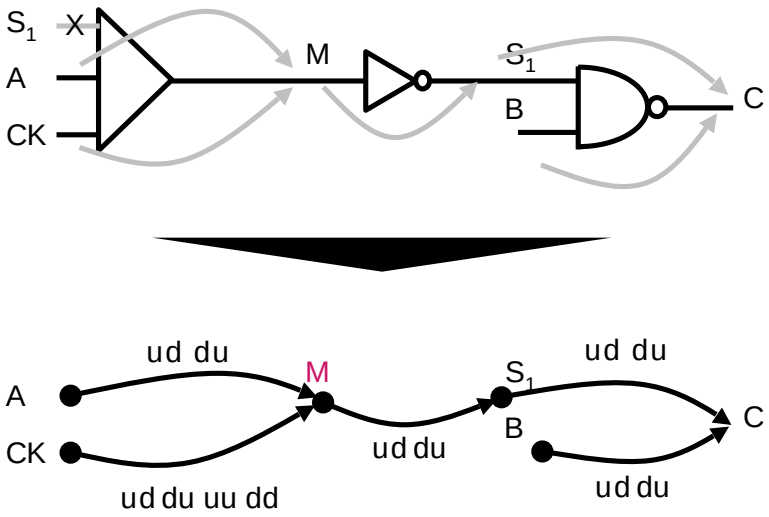


Each cone has a unique output and a certain number of inputs: the nodes controlling the gates of the cone's transistors.

The tool also automatically identifies the memory elements such as memory-cells, latches, pre-charged nodes.

3.2.3. Timing Arcs

The third phase consists of mapping a timing graph on the cone netlist obtained through the disassembly phase (partitioning phase). Timing graph also includes RC networks. The timing graph is defined as follow: edges are rising or falling events (logic transitions) on an input or output of a cone. Arcs are possible causality relations between events. Causality relations are also called timing arcs.



3.2.4. Timing Models

The fourth phase consists of the valuation of the timing arcs. A major innovation in HITAS methodology is the valuation of the timing arcs by delay models.

When a timing arc refers to a cone, it is associated with analytical current source models (CSM), which enable to compute the effective delay for any input slope or output load (CSM present the significant advantage to be independent from input slope and output load). It ensures high precision in slope propagation and crosstalk analysis (where effective load can change during analysis).

The current source models are based on analytical equations, and take into account the following factors:

- The input slope, modeled as a non-linear curve
- The output load, modeled as a pi-network
- The transient short circuit current during the commutation of a gate

When a timing arc refers to an RC network, it is associated with the RC network. Effective delay is computed with the AWE algorithm for any input slope or output load.

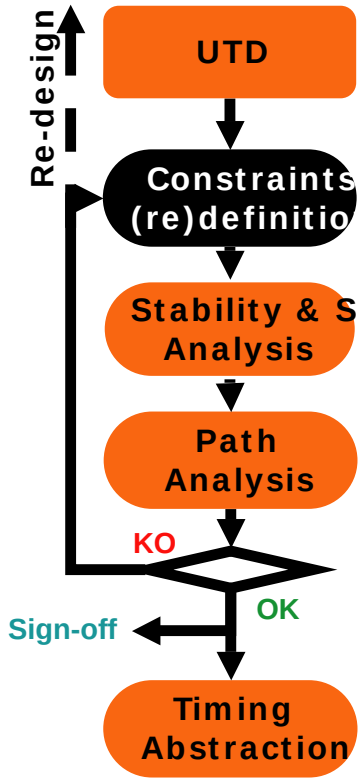
3.2.5. Timing Paths

The last phase consists of the creation of all the timing paths: the successive timing arcs between connectors and memory elements are merged to create timing paths. All the possible timing paths in the design are saved into the database.

3.3. Timing Database Analysis

3.3.1. Database Analysis Flow

The diagram below presents the set of functionalities of the HITAS platform, for complete timing and SI analysis, allowing to performing frequency optimization and violations repairs.



Timing Constraints	Input connectors arrival times, output connectors departure times, clocks specifications
STA and SI	STA is optionally coupled with the SI analysis
Path Analysis	ECO to repair violations or optimize frequencies

3.3.2. Static Timing Analysis

The static timing analyzer engine of HITAS calculates setup and hold slacks for all reference points in a circuit (output connectors, latch data inputs, latch commands, and precharged nodes). The algorithm is based upon the propagation of switching windows throughout the design. The engine needs the specification of the external clocks and the arrival times of the input connectors (which define switching windows). Together with the timing database, the tool propagates the switching windows and the clocks throughout the circuit, in order to obtain switching windows for the points which require verification (the reference points).

The tool calculates setup and hold slacks for each of the reference points by comparing the switching windows obtained at the point with the propagated clocks according to specific timing checks. Any violation of the timing checks translates into a negative value calculated for the setup or hold slacks.

3.3.3. Crosstalk Analysis

Coupling capacitances influence depends on the relative activity on nets. The crosstalk analysis engine should be used to calculate the timing information based upon relative net activity using the current-source models from the timing database.

From the switching windows obtained through the static timing analysis, it is possible to determine whether two capacitance-coupled nets may present simultaneous switching. If this occurs, the mutual influence between the nets is modeled by altering the effective value of the coupling capacitance in accordance with the relative slope values of the signals propagated on the two nets. When the effective value of a capacitance on a net is changed, the associated propagation delay of its driver must be reevaluated. Even the altering of a single propagation delay leads to switching window changes on all subsequent nets. As a result, a new static timing analysis must be performed on the circuit, potentially leading to the detection of new aggression. This loop is then repeated until no new simultaneous switching is detected.

The crosstalk analysis engine calculates all propagation delays for all the instances of a subcircuit, according to their context. When all of these delays are computed, setup and hold margins for each of the reference points are calculated and verified, as for the static timing analysis. In addition, the crosstalk analysis engine generates a report file for the circuit containing: details of aggression, the modification of delays according to the detailed behavior of aggressors, together with an evaluation of peak noise voltage.

3.3.4. Path Searching

Path searching is done between reference points, i.e. input and output connectors, latch data inputs, latch commands, and precharged nodes. The information from the static timing analysis are taken into account in order to determine if paths can go through the latches that may have a transparent behavior.

Together with the propagated clocks, the switching windows define the state of the latches and precharged nodes at the arrival of the input data. If the latch or precharged node is in a transparent state, then the path goes through it.

3.3.5. Timing Abstraction

The timing abstraction engine generates timing models of macro-cells and IP-cores, consisting of lookup tables for all timing paths. In a timing model, timing arcs are given as constraints with respect to interface connectors of the block to be abstracted.

Each instantiation context of a timing model, in terms of input slope and output load, will be different. Lookup tables are therefore necessary for each situation to be handled. A lookup table is generated for each of timing paths and timing constraints. Timing models are given in Cadence TLF3/4 format or in Synopsys Liberty format.

Chapter 4. Scope of Usage

4.1. Introduction

The purpose of this chapter is to provide a user of HITAS with guidelines as to the type of circuits on which tool can be used.

In essence, the HITAS static timing analysis platform is designed for digital custom designs and can handle most techniques used in very high speed or low power designs. HITAS is not, however, designed to cope automatically with analog or structures.

For many designers and CAD teams using advanced design techniques, the distinction between analog and digital is not always clearly defined. The role of this chapter is to define what is digital and what is analog for HITAS.

In order to achieve this we first describe the basic assumptions made by HITAS. If these assumptions are not valid for a circuit structure then this is probably analog. In the next section we provide illustrations of a number of different Digital and Custom Digital structures that HITAS is capable of handling. In the final section we present a selection of typical analog structures for which HITAS, in its native mode, is not suited. However, the tool provides means to link with analog simulators, in order to handle those analog structures.

4.2. HITAS Basic Assumptions

4.2.1. Circuit Partitioning

The circuit partitioning in HITAS is based on the identification of all current paths which define the state of each transistor gate. In order to obtain these paths, the circuit representation is converted from a transistor net-list to a cone net-list.

A cone is defined as being, for each circuit node 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 corresponding to the transistors traversed. These branches therefore reveal the signals governing 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.

This set of branches allows us to express the behaviour 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 be verified globally, this means that Sup and Sdn are expressed in terms of the logic surrounding the cone. The user defines the depth, in terms of logic gates, used for the expansion.

4.2.2. Timing Arcs

After circuit partitioning, each individual cone is characterized in terms of delay. First of all, a causality graph is obtained for each cone.

This cone causality graph is deduced directly from the structure of the cone. If an input drives a transistor in a branch to Vdd then the input generates an up transition on the output. If the transistor is NMOS then an up transition on the input creates the up on the output, so the timing arc is uu. A similar logic is applied for inputs driving PMOS transistors and for transistors in Vss branches. Up to four possible timing arcs can exist for each cone input.

Each timing arc can have a maximum and a minimum value. The aim is to obtain the maximum and minimum delays between all possible transitions of all cone inputs and the possible transitions of the output of the cone.

Each maximum and minimum timing arc is associated with a particular cone branch. This association is made by an initial calculation to obtain the most resistive (or least resistive for minimum) branch giving the timing arc transition.

The analysis of this branch provides the timing model (delay and slope) for the timing arc.

4.2.3. Current Characterization

A branch is made up of a number of transistors (PMOS and NMOS) connected in series. The characterization of the timing arc is made by an analysis of the current characteristics of each transistor in the branch. Two special cases, however, should be mentioned:

a) A cone may contain identical parallel branches. If the branch to characterize is part of a set of parallel branches then the current through all branches of the set is used to characterize the timing arc.

b) A transistor in a branch may be part of a transfer gate transistor pair. In this case the current characteristic is calculated using both NMOS and PMOS transistors of the transfer gate. HITAS assumes that both these transistors conduct simultaneously.

4.2.4. Algorithm Assumptions

The basic assumptions made by HITAS are:

a) A full swing (Vdd to Vss or Vss to Vdd) on the cone output occurs as a result of a full swing on a single cone input.

b) Separate cone inputs do not switch simultaneously.

c) For each branch characterization, only one transistor is considered to switch. Transfer gates and series connected transistors with coupled gates are handled as special cases.

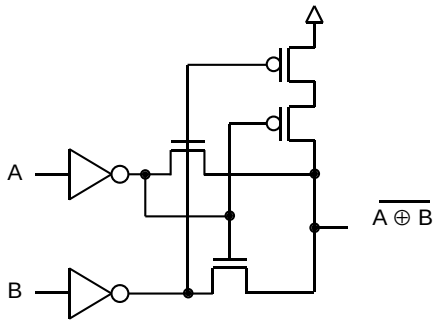
These assumptions are necessary for the partitioning and characterization algorithm to provide a valid result. If these general assumptions hold for a particular circuit structure, then HITAS is applicable.

In the next section we present a selection of design structures for which these assumptions hold.

4.3. HITAS Digital Structures

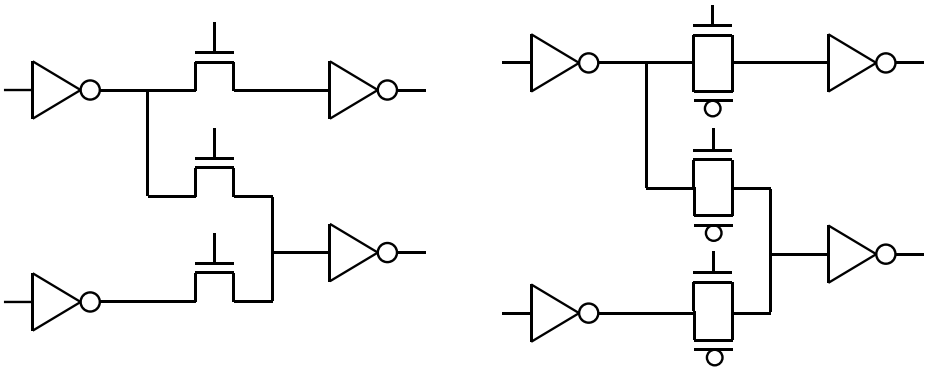
4.3.1. CMOS Gates

HITAS is applicable to all basic CMOS gates regardless of the implementation (invertors, buffers, NAND, NOR, etc). All kinds of exclusive or gates, including the implementation shown in the following figure, can be handled directly using HITAS. No special configuration is necessary.



4.3.2. Pass-Transistor and Transmission Gate Logic

HITAS is applicable to custom digital designs using pass-transistor or transmission gate logic. This often occurs in multiplexer implementations such as those shown in the following figure.

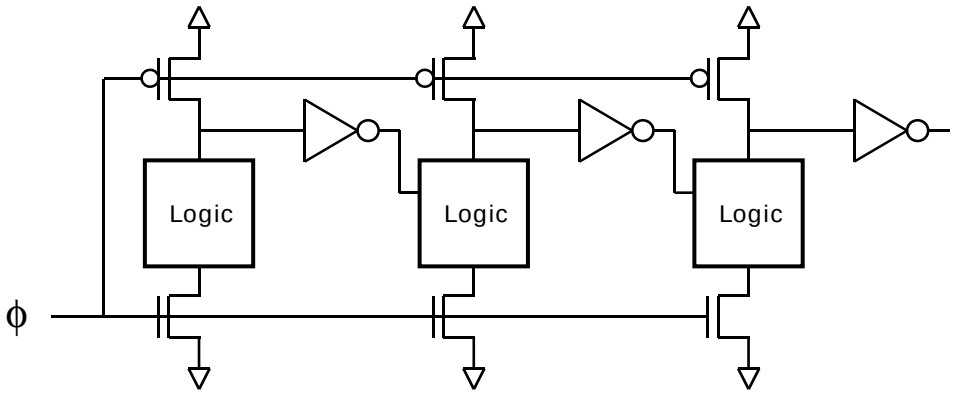


a) Pass-transistor b) Transfer-gate

These structures are handled automatically by HITAS. However, care must sometimes be taken that the partitioning is performed correctly. HITAS may require information about the correlation between selector inputs if this correlation is not present in the block under analysis.

4.3.3. Clocked CMOS Logic

HITAS is applicable to high-speed custom techniques such as clocked CMOS logic. In particular HITAS is well suited to the analysis of Domino-Precharge based designs. Figure 6 shows a typical Domino precharge architecture handled by HITAS.



Each precharge stage is sometimes followed by a keeper or level hold structure. These pose no problem for HITAS.

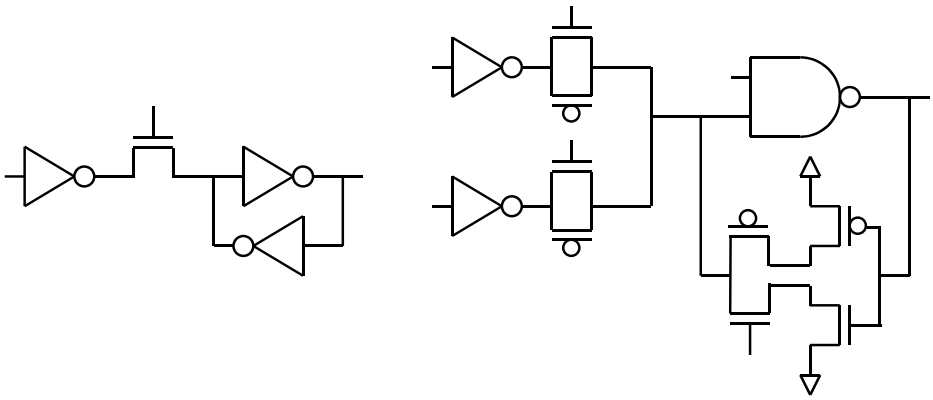
For full handling of this kind of logic it is necessary to activate the automatic precharge detection for the partitioning phase as well as precharge verification during the static timing analysis phase.

4.3.4. Static Latches and Flip-Flops

HITAS incorporates, during the partitioning phase, an advanced algorithm to automatically detect any kind of fully static latch designed using an active feedback loop.

Both conflictual (e.g. inverter feedback) and non-conflictual (e.g. tristate feedback) latches are handled.

Latches can contain any number of clock inputs as well as asynchronous set and reset inputs. All these input types are identified automatically. Following figure shows a number of different latch types handled by HITAS.



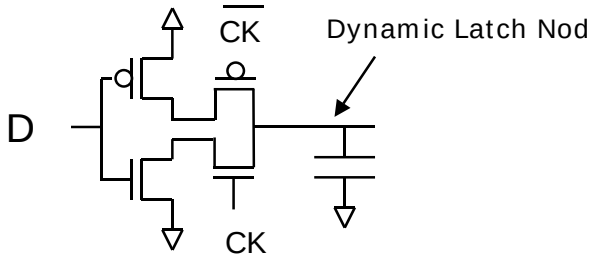
a) Simple conflictual latch

b) Multi-clock latch

Flip-flops are treated as two separate latches (master and slave). HITAS has an option to automatically detect flip-flops. This option can be used simply to report to the user the nodes recognised as the master nodes and the slave nodes. Alternatively, it is possible, for some flip-flops, to group the master and slave and perform a simplified flip-flop timing verification.

4.3.5. Dynamic Latches

HITAS can also be used to recognise dynamic latches such as the simple example shown in the following figure.



Recognition of dynamic latches is a configuration option of HITAS. Some care should be taken as there is can be ambiguity with a tristate bus.

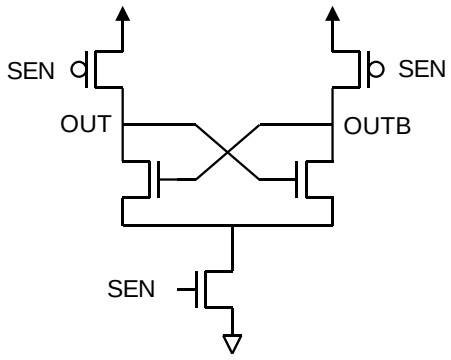
HITAS uses the functional analysis of the partitioning phase to determine whether a node can be a dynamic latch. An internal node for which all drivers can be deactivated is considered to be a dynamic latch unless HITAS is told explicitly otherwise.

4.4. HITAS Analog Structures

In this section we review a selection of typical analog structures which HITAS cannot handle directly.

4.4.1. Sense Amplifier

Following figure shows a typical sense amplifier building block. This structure, typically found in memories, senses a difference in potential between two inputs and provides an output of 1 or 0 depending which is the greater.

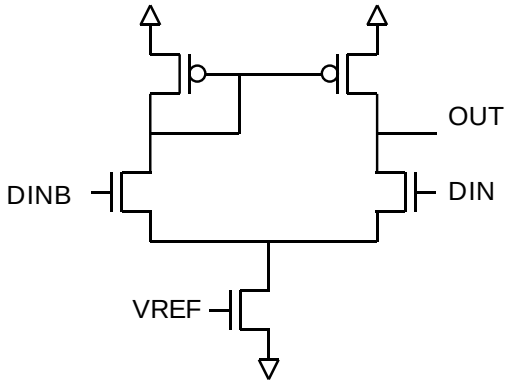


It should be quite clear from the HITAS assumptions that this structure cannot be handled. In effect it requires two input to switch simultaneously and neither of these inputs are full-swing.

4.4.2. Differential Amplifier

Following figure shows a typical long-tailed pair implementation of a differential amplifier. The operation is very similar to the sense amplifier. This is a basic analog amplifier building block but can also be found in high-speed digital logic styles such as CML.

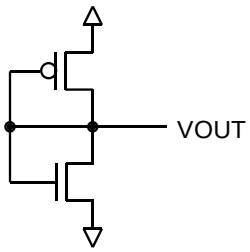
Again the basic HITAS assumptions are not respected here since two inputs switch simultaneously even though they may be full-swing (for differential digital logic).



4.4.3. Voltage Generator

HITAS cannot calculate static power supply values produced by internal voltage generator circuits. The values of the voltages must be provided explicitly to HITAS using Vcard directives in the spice netlist.

For example, following figure shows a simple voltage divider circuit. This may occur in a netlist to provide a secondary lower power supply value to reduce power consumption. Here, the user is required to specify a Vcard at the output of this voltage divider to specify its effect.



4.4.4. Typical Analog Devices

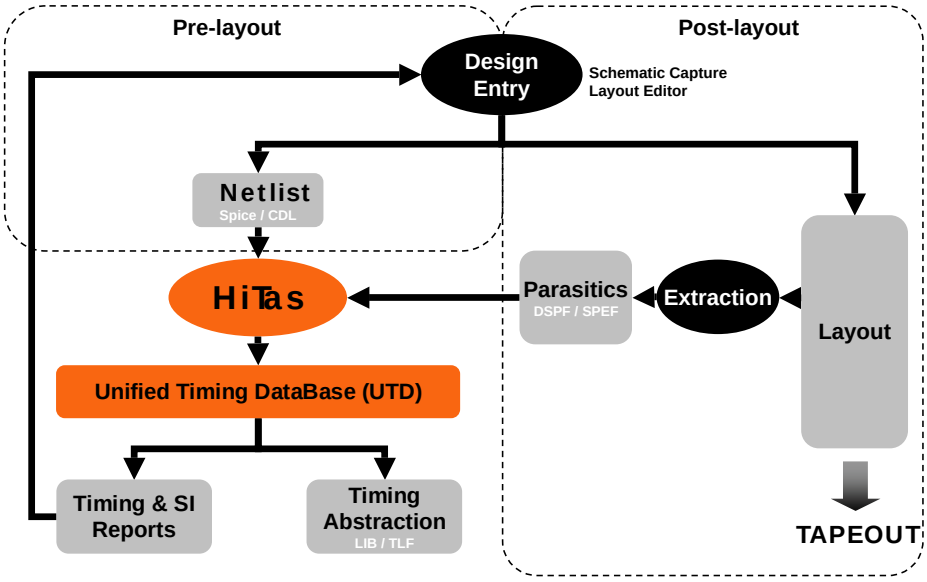
HITAS is not directly applicable to common analog building blocks found in mixed-signal designs. These include:

- Analog to Digital and Digital to Analog Converters
- Voltage Controlled Oscillators
- Phase Locked Loops
- Charge Pumps

Chapter 5. Design Flow Integration

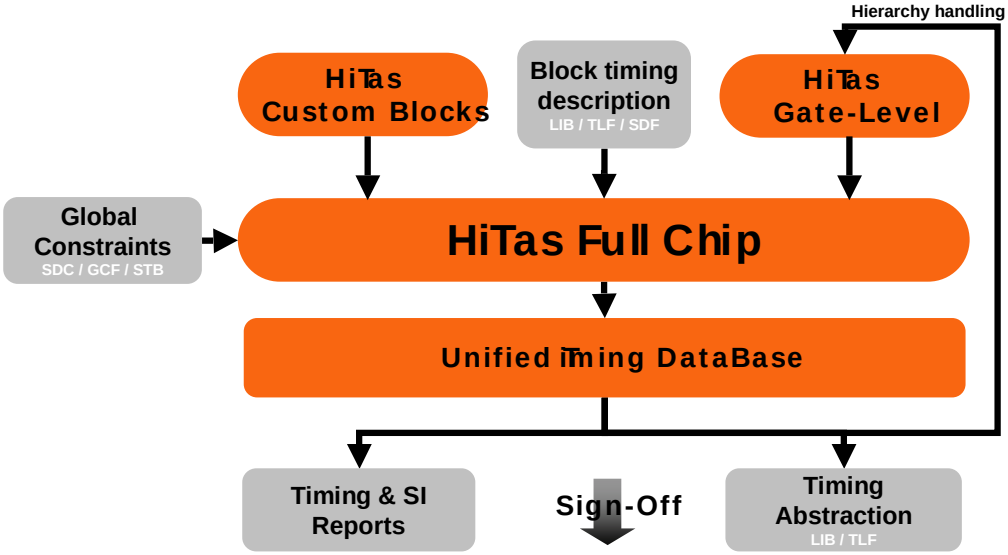
5.1. Transistor-Level Analysis

For a block up to 1M transistors, HITAS performs a flat transistor-level analysis, and generates a flat timing database of the block, taking into account the interconnect parasitics (RC networks).



5.2. Full-Chip Analysis

In the hierarchical analysis mode, HITAS uses existing timing views of instantiated blocks to work out the timing database of the whole circuit, taking into account the interconnects between blocks.



5.3. Input Files

5.3.1. Netlist

.spi	flat transistor extraction from the layout in SPICE format, possibly with interconnect parasitics and coupling capacitances
.cdl	hierarchical schematic CDL/SPICE
.spi	hierarchical netlist in SPICE format
.vhdl	hierarchical netlist in structural VHDL format
.vlg, .v	hierarchical netlist in structural Verilog format

5.3.2. Parasitics

.dspf, .spf	interconnect parasitics and coupling capacitances back-annotation in DSPF/SPF format
.spef	interconnect parasitics and coupling capacitances back-annotation in SPEF format

5.3.3. Technology

bsim3	SPICE format, BSIM3 level
bsim4	SPICE format, BSIM4 level

5.3.4. Timing characterizations

.lib	Synopsys Liberty Format
.tlf	Cadence TLF format

5.3.5. Timing Constraints

.sdc	Synopsys Design Constraints
.gcf	Cadence Global Constraints Format
.inf	Avertec Proprietary Constraints Format

5.4. Output Files

5.4.1. Disassembly

.rep	Contains a list of diagnostics (warnings and error messages) attributed to particular signals or transistors
.cns	Contains the cone view of the circuit. Used for debugging
.cnv	Contains the cone view of the circuit. Used for debugging

5.4.2. Timing Database Generation

The standard output of HITAS is the entire timing view of the circuit, called Unified Timing Database (UTD). It consists of text files suitable for the static timing analysis (timing constraints check), crosstalk analysis and timing abstraction. The UTD is made of the following files:

.dtx	cone and RC timing arcs
.stm	timing models relative to timing arcs (both cone and paths)
.rcx	RC networks relative to RC timing arcs
.loop	combinational loops detected in the circuit

5.4.3. Static Timing Analysis

.str	slack report
.sto	switching windows calculated for all reference points

.ste	warnings encountered during static timing analysis (such as latches with no clock)

5.4.4. Crosstalk Analysis

.ctk	human readable file containing crosstalk related information, such as noise levels and aggressors contributions
.ctx	file containing all the delays calculated with crosstalk effects of a complete design hierarchy

5.4.5. Abstraction

.lib	Timing abstraction in Synopsys Liberty format
.tlf3	Timing abstraction in Cadence TLF3 format
.tlf4	Timing abstraction in Cadence TLF4 format

Chapter 6. Using Tcl Interface

6.1. Script Launch

All functionalities of the HITAS platform can be accessed with the `avt_shell` Tcl scripting interface. `avt_shell` can be used the same way as any `.tcl` script. It is statically linked with all HITAS libraries, and thus contains all HITAS functions.

`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
```

6.2. Tools Configuration

The configuration of all the timing tools of the HITAS 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`
- 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 `avt_shell script` file on the values set in the `avttools.conf`.

6.3. Error Policy

HITAS distinguishes three levels of error:

`level 0`	WARNING, unrecognized or redundant configuration
`level 1`	ERROR, inconsistencies in the inputs possibly workaround by defaulted values. Results are not guaranteed to be valid
`level 2`	FATAL ERROR, inconsistencies in the inputs with no possible workaround. Results will not be valid.

Errors of level 0 and 2 lead to a predefined behavior of the tool, which cannot be tuned.

When encountering an error of level 0, the tool always print a `[Warning]` message. The tool does not abort. Example:

```
[Warning AVT-044] Multiple settings for variable 'simSlope'
```

When encountering an error of level 2, the tool always print an `[Error]` message, and aborts. Example:

```
[Error SPI-009] Can't open file ./inv.spi
```

The user can tune the strictness the tool treats errors of level 1 with. This is done through the configuration variable `avtErrorPolicy`, which can take values `strict` or `lenient`. When set to `lenient` (default setting), errors of level 1 are treated as errors of level 0, and the tool issues a `[Warning]` message, for example:

```
[Warning SPI-070] Conflicting power supply on node 'gnd' keeping 1.62v
```

When `avtErrorPolicy` is set to `strict`, errors of level 1 are treated as errors of level 2. The tool issues an `[Error]` message and aborts, for example:

```
[Error SPI-070] Conflicting power supply on node 'gnd' keeping 1.62v
```

6.4. Objects

HITAS platform functionalities, such as database construction, static timing analysis, path searching..., can be accessed through Tcl functions. Those functions may either display results on standard output, create files, or return pointers on objects. Returned pointers on objects can in turn become arguments of other Tcl functions.

Here is a list of the objects returned by the Tcl functions. For more information about objects, see HITAS Reference Guide.

Netlist	Electrical view of a Subcircuit, which may be either flat or hierarchical, and contain components of the files it originates from: mosfets, resistances capacitances and instances.
TimingFigure	Unified Timing Database
TimingSignal	Node of the subcircuit on which one a timing event can occur. Timing Signals are built on outputs of cones (partitions), on outputs of RC networks, and on input connectors. They can be typed as latches, precharges, latch or precharge commands, connectors, user-defined breakpoints or simple signals.
TimingEvent	Rising or Falling transition on a timing signal
TimingPath	Concatenation of propagation delays through cones and RC networks, between reference points (latches, precharges,

	latch or precharge commands, connectors, user-defined breakpoints)
TimingDetail	Detail about a propagation delay within a timing path
StabilityFigure	Back-annotation of a TimingFigure. Contains the switching windows associated to the TimingSignal objects of the TimingFigure.
StabilityPath	Object associated with each TimingSignal requiring setup/ hold verification (latch, latch command, precharge, output connector). Contains timing information about the input logical cone (do not mismatch with cone as a partition) of the TimingSignal object.

6.5. Functions

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

General	Global configuration, file loading, netlist manipulation, statistics
INF Configuration	Configuration though the INF functions
SDC Support	SDC functions supported for STA configuration
DB Construction	Automatic or manual generation of the timing database
DB Browsing	Functions for retrieving timing paths, propagation delays, signal properties...
STA	STA launch and results analysis
Timing Abstraction	Functions for generating .lib files from the timing database

6.6. INF Configuration and SDC Support

For tool configuration needing more than the specification of a simple value (as it is done through the `avt_config` function), HITAS uses the INF mechanism, which is a set of Tcl configuration functions. INF functions are available for all the phases of the timing analysis process (disassembly, timing database construction, static timing analysis, crosstalk analysis and timing abstraction).

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}
create_clock -name ck -period 1000 [get_ports {ck}]
```

Outside of a Tcl script (in the Xtas GUI), the INF and SDC functions are used through the `.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 HITAS Reference Guide).

Chapter 7. Timing DB Construction

7.1. 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`.

7.1.1. 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, HITAS 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.

7.1.2. 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, HITAS can also take `.lib` files as input, but it should be understood that HITAS is primarily designed for digital transistor-level analysis, and that providing `.lib` files should only apply to parts of the netlist where HITAS 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
```

7.1.3. Parasitics

HITAS 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.

7.1.4. Vectorization

HITAS 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.

7.1.5. Ignoring Elements

For a reason or another, some elements in the source files may be unsupported by HITAS or may not respect standard format syntax. To work around those elements, HITAS 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.

7.2. DB Construction

7.2.1. 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.

HITAS 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
```

7.2.2. Defining Simulation Thresholds

By default, the slope is defined between 20% and 80% of Vdd. It is possible to change those values with the `simVthLow` and `simVthHigh` variables. Delays are always computed with a threshold of 50%.

7.2.3. Defining Simulation Temperature

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

7.2.4. Invoking DB Construction

The timing DB construction routine are invoked by the `hitas` 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 signalis associated with. The typical Tcl command for invoking timing DB construction is:

```
set fig [hitas 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 timing DB containing DTX, STM, RCX files.

7.3. Output Files

7.3.1. REP file

Once the Timing DB has been constructed, it is important to have a look at the `.rep` file. This file is a report of the disassembly process. The essential thing to check here is that latches have been correctly detected. Ideally, all the latches have been automatically detected by the Boolean analysis of the gate loops. However, the report file lists all the loops between gates that have been identified during the disassembly process, and if they have been associated with a latch. A complete list of the warnings issued in the `.rep` is available in the reference manual.

7.3.2. LOOP file

Ideally the `.loop` file does not exist, as it is created only if the path searching algorithm detects combinational loops in the design. Note that the path searching algorithm is performed after the disassembly process, and therefore the occurrence of combinational loops is heavily dependant on the correct recognition of the latches in the design.

7.3.3. 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 tasGenerateConeFile yes
```

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

```
avt_config avtVerboseConeFile yes
```

7.3.4. DTX and STM files

The default configuration generates the `.dtx` file (timing arcs) and `.stm` file /9timing models).

7.4. Latch Detection and Modeling

HITAS handles the following types of latches:

- D-latches
- Symmetric latches and memory cells
- RS latches
- Dynamic latches

All those types of latches are recognized and modeled automatically during the partitioning phase. If automatic detection were to fail, manual identification is still possible. User-defined tags may be applied on nodes in the design.

7.4.1. Detection Sequences

Manual Identification

If latch/command tags on specific nodes are set in the configuration file, those tags will be applied and will disable any subsequent automatic analysis upon those nodes.

Simple Detection

Simple detection is based upon pattern matching. If activated, this sequence uses simple patterns and rules to quickly detect the simplest latch structures. This sequence is to be used with caution. It permits to quickly detect the most common latches but also uses some heuristics which may not be correct for more complex latches. This sequence is particularly useful (time saving) when dealing with static memories (SRAM) containing a large number of 6T bit-cells. Latches detected at that point will be:

- Memory cells, provided they only consist of looped inverters
- Level-holders, looped inverters also
- Simple D-latches. In that case the transistor controlling the memory node will be the one the closer to the memory point

Automatic Detection

Automatic detection is based upon loop analysis. This sequence recognizes any kind of fully static latch designed using an active feedback loop, and containing any number of clock inputs as well as asynchronous set and reset inputs. Within this sequence, specific algorithms may be further applied to model RS latches, asynchronous set and reset within D-latches and symmetric latches.

- RS latches. Automatic detection and modelling of NAND/NOR based RS bi-stable
- Asynchronous signals: set and reset detection. D-Latches further modelling
- Symmetric latches

Dynamic Latches Detection

7.4.2. Enabling Detection Sequences

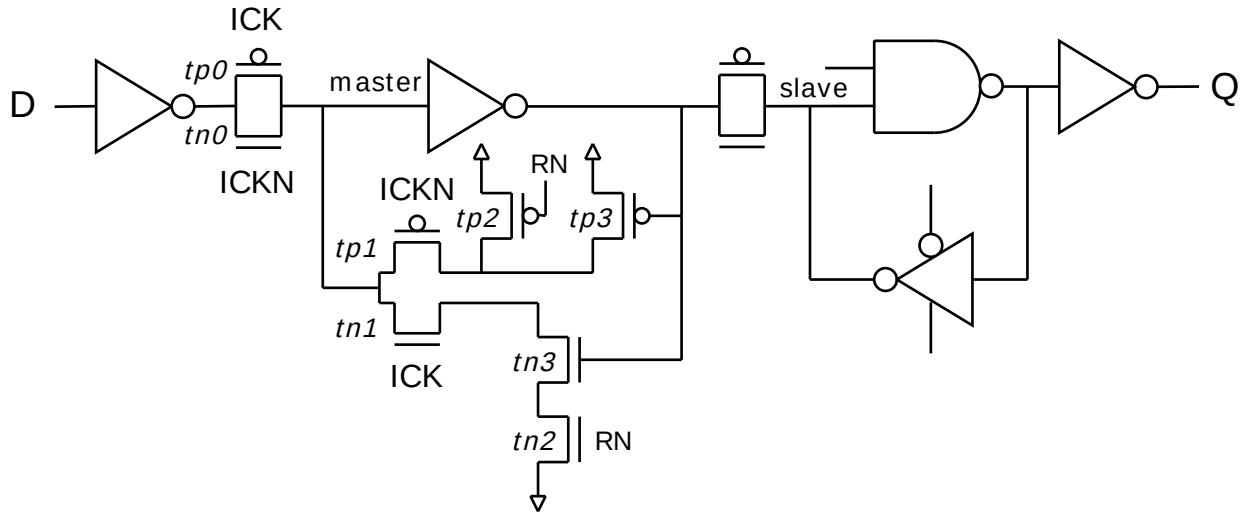
The detection sequences are controlled by the following variables:

Manual Identification	`inf_MarkSignal` `inf_MarkTrans`
Simple Detection	`yagSimpleLatchDetection=memsym|levelhold|latch`
Automatic Detection	`yagAutomaticLatchDetection=yes`
RS Latches	`yagAutomaticLatchDetection=yes` `yagAutomaticRSDetection=mark|latch|legal|illegal`
Asynchronous Signals	`yagAutomaticLatchDetection=yes` `yagSetResetDetection=yes|remove`
Symmetric Latches	`yagAutomaticLatchDetection=yes` `yagAutomaticMemsymDetection=yes|remove`
Dynamic Latches	`yagDynamicLatchDetection=yes`

See HITAS Reference Guide for a more detailed description of those configuration variables.

7.5. Static Latch Modeling

Latches such as the one described below are well recognized with the automatic algorithm controlled by `yagAutomaticLatchDetection`. In this example HITAS recognizes a latch around the node `master` (and another one around the node `slave` - for the sake of simplicity we only discuss here the first latch). This node is tagged in the database as a **Latch** node (see `.cnv` file).



Each Latch node is controlled by **Command** transistors. Here the **Command** transistors for node `master` are `tn0` and `tp0` - they control the writing of the data - and `tp1` and `tp2` - they control the reset of the latch.

7.5.1. Asynchronous Set and Reset

The modeling of static latches can be refined in order to cope with the set and reset signals. This is controlled by `yagSetResetDetection`, and disabled by default (set to `no`). Whwn set to `yes`, the transistor `tp1` is no longer considered to be a command and `tp2` is marked **Async**. This prevents the construction of asynchronous timings arcs between the reset signal and memory node - between `RN` and `master` here - which otherwise clutter the reports.

7.5.2. Manual Configuration

The following manual configuration would lead to the same modeling as the one generated by the automatic detection. We present it as an example of what should be done in order to model latches which would not be automatically recognized. The following lines describe the configuration to be used for tagging the `master` **Latch** node and associated **Command** transistors.

```
inf_MarkSignal master Latch+Master
inf_MarkTransistor tn0 Command
inf_MarkTransistor tp0 Command
inf_MarkTransistor tn1 Feedback+NonFunctional
inf_MarkTransistor tn2 Feedback+NonFunctional
inf_MarkTransistor tn3 Feedback+NonFunctional
inf_MarkTransistor tp1 Command+NonFunctional
inf_MarkTransistor tp2 Command
inf_MarkTransistor tp3 Feedback+NonFunctional
```

HITAS performs latch-based analysis, so the MASTER tag is optional

Timing Arcs

HITAS constructs the following timing arcs:

```
DATA: D (R) -> master (F), enabled by ICKN (R)
```

```
DATA: D (F) -> master (R), enabled by ICK (F)
ACCESS: ICK (F) -> master (R)
ACCESS: ICKN (R) -> master (F)
```

7.5.3. Intrinsic Setup and Hold

Setup and hold timing checks are performed at the memory node itself - the `master` node. For both setup and hold timing checks, HITAS also adds correction margins which further ensure the robustness of the analysis. These margins are called intrinsic setup and intrinsic hold times.

```
SETUP: master (R) -> ICKN(R) - closing event of the command
SETUP: master (F) -> ICK(F)
HOLD: master (R) -> ICKN(R) - closing event of the command
HOLD: master (F) -> ICK(F)
```

Intrinsic Setup

For setup, HITAS checks that the latest possible switching time of the memory node - caused by the switching of the data - occurs before the earliest possible closing time of the latch in the next cycle - here the earliest time between ICK rising and ICKN falling - so that the data is always memorized in the next cycle. This reads:

```
D->mastermax < min(CK->ICKNmin, CK->ICKmin) + cycle_time
```

The setup slack is then defined as:

```
Setup_slack = min(CK->ICKNmin, CK->ICKmin) + cycle_time - D->mastermax
```

Or in a simpler way:

```
Setup_slack = clock_path_min + cycle_time - data_path_max
```

In fact, HITAS does not calculate the latest possible switching time of the memory node, but the latest possible switching time of the feedback node, further ensuring that the data has been properly stored in the latch. This gives:

```
D->mastermax + master->feedback < min(CK->ICKNmin, CK->ICKmin) + cycle_time
```

The setup slack is then defined as:

```
Setup_slack = min(CK->ICKNmin, CK->ICKmin) + cycle_time - D->mastermax - master-
>feedback
```

Or in a simpler way:

```
Setup_slack = clock_path_min + cycle_time - data_path_max - intrinsic_setup
```

From a modeling point of view, this translates to the addition of a setup timing arc, between the memory node and each of the signals driving a gate of a command transistor, whose value is the propagation delay between the memory node and the feedback node.

Intrinsic Hold

For hold, HITAS checks that the earliest possible switching time of the memory node - caused by the switching of the data - occurs after the latest possible closing time of the latch in the same cycle - here the latest time between ICK rising and ICKN falling - so that no other data is accidentally memorized in the same cycle.

```
D->mastermin > max(CK->ICKNmax, CK->ICKmax)
```

The hold slack is defined as:

```
Hold_slack = D->mastermin - max(CK->ICKNmax, CK->ICKmax)
```

Or in a simpler way:

```
Hold_slack = data_path_min - clock_path_max
```

Actually, HITAS does estimates latest possible closing time of the latch not at Vdd/2, as it is done for all other propagation delays, but at the Vt of the Command transistors. This assumes that the data may still be stored, even if the Command transistors are in a low-conducting mode. Preventing this adds robustness to the analysis.

```
D->mastermin > max(CK->ICKNmax, CK->ICKmax) + max(slopeICKN vdd/2->vt, slopeICK vdd/2->vt)
```

The hold slack is defined as:

```
Hold_slack = D->mastermin - max(CK->ICKNmax, CK->ICKmax) - max(slopeICKN vdd/2->vt, slopeICK vdd/2->vt)
```

Or in a simpler way:

```
Hold_slack = data_path_min - clock_path_max - intrinsic_hold
```

From a modeling point of view, this translates in the addition of a hold timing arc, between the memory node and each of the signals driving a gate of a Command transistor, which value is the portion between Vdd/2 and Vt of the slope on the signal driving the gate.

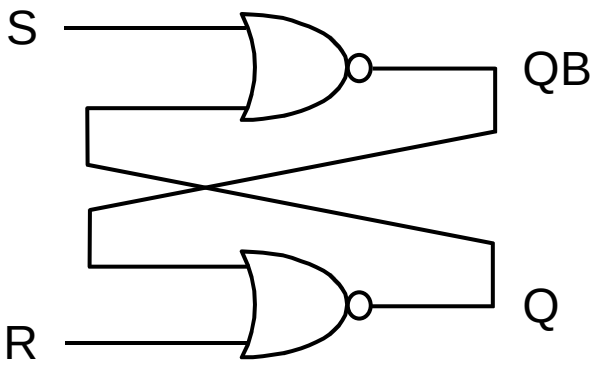
7.6. RS-Latches

HITAS automatically recognizes and models NAND-based and NOR-based RS structures. Only the RS structures where NAND or NOR building gates directly loop on each other (with no intermediary inverter or buffer present) are automatically recognized. However, manual recognition is still possible, and the same range of modeling methods can be applied on manually-defined RS structures.

Automatic recognition of RS structures is enabled by switching on the variable `yagAutomaticRSDetection`:

```
avt_config yagAutomaticRSDetection mark
```

This enables HITAS to detect NAND-based and NOR-based RS structures having an arbitrary number of inputs, providing that the constituent NAND or NOR gates loop directly onto each other. Below is an example of such a structure.



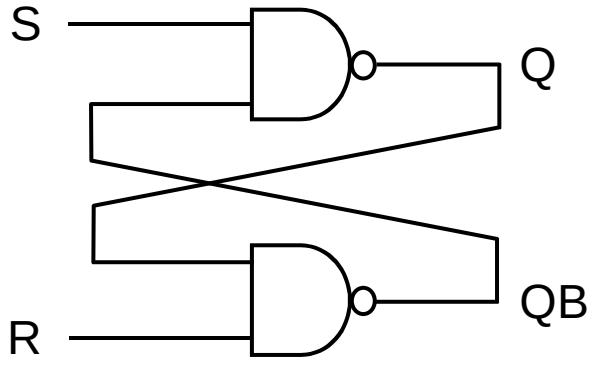
With this configuration, HITAS only marks the looping nodes as `RS` (see `.cnv` file), and issues a message in the `.rep` file. For this NOR-based RS structure, the following timing arcs are then created:

```
S (rising) to QB (falling)
S (falling) to QB (rising)
R (rising) to Q (falling)
R (falling) to Q (rising)
Q (rising) to QB (falling)
Q (falling) to QB (rising)
QB (rising) to Q (falling)
QB (falling) to Q (rising)
```

This leads to combinational loops the tool is not able to handle, for example:

```
S (rising) to QB (falling) to Q (rising) to QB (falling) to ...
```

The study of the truth tables of the NOR gate tells us that such loops actually cannot occur, as some transitions cannot be excited. We will detail this in the next section. Regarding NAND-based RS structures, here is an example of what HITAS properly detects:



Here, the created timing arcs are the following ones:

```
S (rising) to Q (falling)
S (falling) to Q (rising)
R (rising) to QB (falling)
R (falling) to QB (rising)
Q (rising) to QB (falling)
Q (falling) to QB (rising)
```

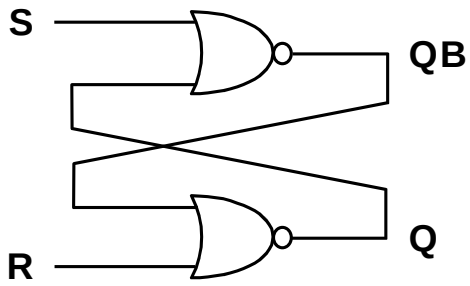
```
QB (rising) to Q (falling)
QB (falling) to Q (rising)
```

They also lead to combinational loops.

7.6.1. Modeling of NOR-based structures

All States Allowed

Let's study the truth table of the NOR-based design:



S	R	Q_{n+1}	QB_{n+1}	Operation
0	0	Q_n	QB_n	hold
1	0	1	0	set
0	1	0	1	reset
1	1	0	0	not allowed

Let's consider the timing arcs from Q to QB. As stated above, we have the following timing arcs, which we can also identify from the truth table:

```
Q (rising) to QB (falling)
Q (falling) to QB (rising)
QB (rising) to Q (falling)
QB (falling) to Q (rising)
```

Now we must keep in mind that we are dealing with NOR gates, and that a NOR propagates a transition from one of its inputs only if the others inputs are all `0`. In a situation where NOR gates are looped on each others, some transitions cannot be excited. If we look at the truth table above, we see that the only way to have `Q (rising)` is to perform the set operation, where `S` is `1` and `R` is `0`. But if `S` is now `1`, no transition can propagate through the NOR gate: `Q (rising) to QB (falling)` can never be excited.

We have the symmetrical situation for `QB (rising) to Q (falling)` in the reset operation. The following timing arcs finally remain, and prevent the occurrence of combinational loops:

```
S (rising) to QB (falling)
S (falling) to QB (rising)
R (rising) to Q (falling)
R (falling) to Q (rising)
Q (falling) to QB (rising)
QB (falling) to Q (rising)
```

This way of modeling is enabled by switching the yagAutomaticRSDetection to `"mark+illegal"`:

```
avt_config  yagAutomaticRSDetection "mark+illegal"
```

Legal States Only

The configuration described above only removes timing arcs that can never be excited. It allows timing arcs used when entering an illegal state: when both `S` and `R` are set to `1`, `Q` and `QB` are both `0`. It is the only situation where the NOR gate can have its `R` (resp. `S`) input set to `1` and its `Q` (resp. `QB`) input is set to `0`, and thus propagate a falling transition from `R` or `S`:

```
S (falling) to QB (rising)
R (falling) to Q (rising)
```

Depending on the surrounding logic of the RS structure, this situation may or may not occur. However, the tool is not able to analyze it. If you desire to consider RS structures always remaining in the legal state, you should state it explicitly:

```
avt_config  yagAutomaticRSDetection "mark+legal"
```

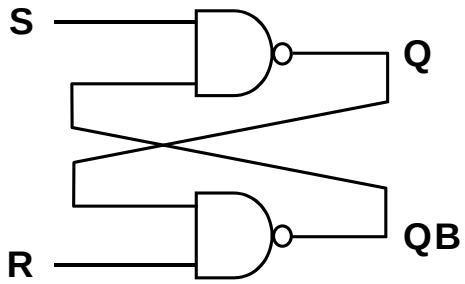
Only the following transitions remain:

```
S (rising) to QB (falling)
R (rising) to Q (falling)
Q (falling) to QB (rising)
QB (falling) to Q (rising)
```

7.6.2. Modeling of NAND-based structures

All States Allowed

The same reasoning applies for a NAND-based structure, based on its truth table:



S	R	Q_{n+1}	QB_{n+1}	Operation
0	0	1	1	not allowed
0	1	1	0	set
1	0	0	1	reset
1	1	Q_n	QB_n	hold

The NAND gate propagates a transition from one of its inputs only if the other inputs are all `1`. Removing the unexcited timing arcs leads to the following arcs remaining:

```
S (rising) to Q (falling)
S (falling) to Q (rising)
R (rising) to QB (falling)
R (falling) to QB (rising)
Q (rising) to QB (falling)
QB (rising) to Q (falling)
```

Legal States Only

If only legal states are allowed, the following timing arcs remain:

```
S (rising) to Q (falling)
R (rising) to QB (falling)
Q (rising) to QB (falling)
QB (rising) to Q (falling)
```

7.6.3. Fine Tuning

The `yagAutomaticRSDetection` variable defines a global behavior for all the RS structures encountered and properly detected. However, one may wish to define a specific behavior for a given RS structure encountered. For example, on 100 RS structures in a design, 94 may be treated as always remaining in legal state, 4 may be treated as possibly entering an illegal state, and the 2 remaining ones may be treated as pulse generators. One should then wish to override a global configuration for given RS structures. It is therefore possible in HITAS with the `inf_DefineRS` Tcl command, as done in the following script, referring to the example above:

```
avt_config yagAutomaticRSDetection "mark+legal"
inf_DefineRS nand1.S illegal
inf_DefineRS nand2.S illegal
inf_DefineRS nand3.S illegal
inf_DefineRS nand4.S illegal
inf_DefineRS pulsegen1.S mark_only
inf_DefineRS pulsegen2.S mark_only
```

7.6.4. Manual Tuning

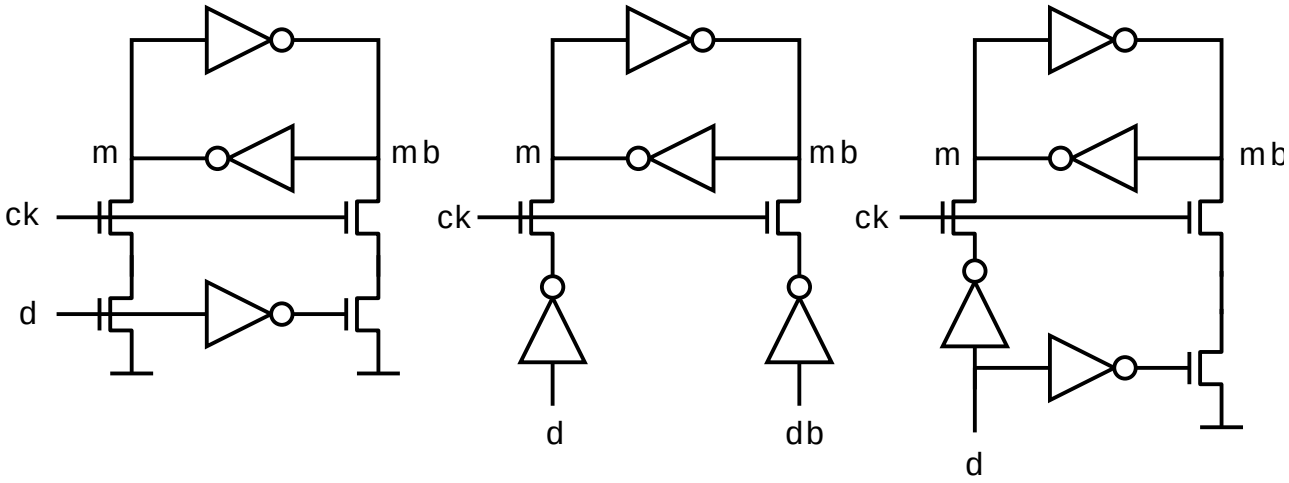
Situations arise where timing arcs should be removed manually: when RS structures are not automatically detected, and when they are used for purposes not covered by the modeling options described above (typically in pulse generators).

When RS structures are not properly detected, timing arcs should be removed in the same way they are removed when automatically handled. Arc removal should be done with inf_DisableTimingArc Tcl function. The following script describes manual removal of timing arcs for the NOR-based RS latch described above, in the "Legal States Only" situation:

```
inf_DisableTimingArc S QB du
inf_DisableTimingArc R Q du
inf_DisableTimingArc Q QB ud
inf_DisableTimingArc QB Q ud
```

7.7. Symmetric Latches

HITAS considers latches symmetric when it is possible to write on both sides of the memorizing loop. Latches which fall into this category are symmetric pulldowns, symmetric bitcells and asymmetric pulldowns, as described below.

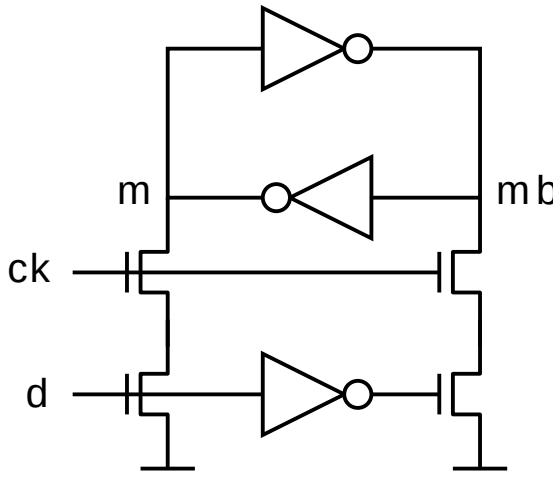


HITAS considers both sides - `m` and `mb` of the symmetric latc being **Latch** nodes. A supplementary **Memsym** tag is added on each **Latch** node. Timing checks - setup and hold - are performed on both sides of the symmetric latch.

Intrinsic setup and hold times are added on each **Latch** node in the same way as they are on a D-Latch **Latch** node. The next sections detail modeling of symmetric latches

7.7.1. Symmetric Pulldown

Typical Structure



Latch Nodes and Commands

The latch nodes are here `m` and `mb`. The **Command** transistors are the ones which gate is controlled by **ck**.

Timing Arcs

Timing arcs are the following:

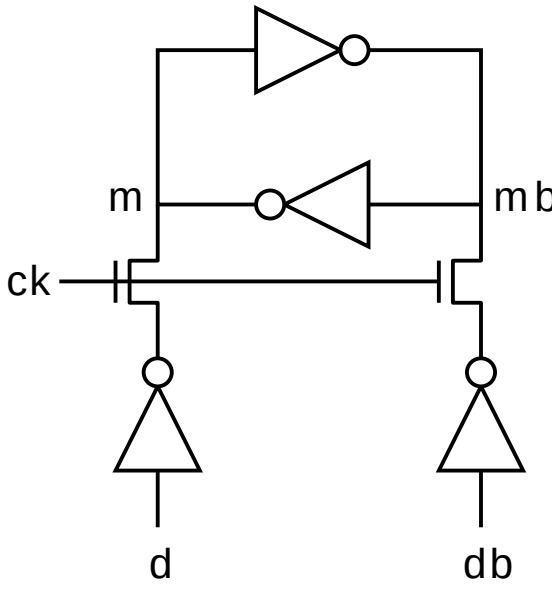
```
DATA: D (R) -> m (F), enabled by CK (R)
DATA: D (F) -> mb (F), enabled by CK (R)
```

```
DATA: m (F) -> mb (R)
DATA: mb (F) -> m (R)
ACCESS: CK (R) -> mb (R) - through the feedback loop
ACCESS: CK (R) -> m (R) - through the feedback loop
SETUP: m (F) -> CK (F)
SETUP: mb (F) -> CK (F)
HOLD: m (F) -> CK (F) - closing event of the command
HOLD: mb (F) -> CK (F)
```

As it is for the D-Latch, the intrinsic setup time is the propagation delay of the inverter between `m` and `mb` and `m` and `mb` - for **Latch** nodes `m` and `mb` respectively.

7.7.2. Symmetric Bitcell

Typical Structure



Latch Nodes and Commands

The latch nodes are here `m` and `mb`. The **Command** transistors are the ones which gate is controlled by **ck**.

Timing Arcs

Timing arcs are the following:

```
DATA: D (R) -> m (F), enabled by CK (R)
DATA: D (F) -> mb (F), enabled by CK (R)
DATA: m (F) -> mb (R)
DATA: mb (F) -> m (R)
ACCESS: CK (R) -> mb (R) - through the feedback loop
ACCESS: CK (R) -> m (R) - through th efeedback loop
SETUP: m (F) -> CK (F)
SETUP: mb (F) -> CK (F)
HOLD: m (F) -> CK (F) - closing event of the command
HOLD: mb (F) -> CK (F)
```

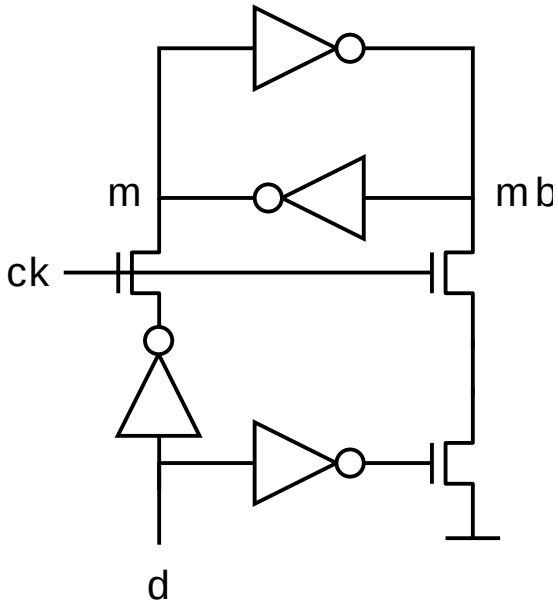
The following timing arcs are disabled: the NMOS transistors drive only a weak current in the latch node - considering a rising transition on the latch node.

```
DATA: D (R) -> m (R), enabled by CK (R)
DATA: D (F) -> mb (R), enabled by CK (R)
```

However, the contribution of the current through the NMOS is taken into account when evaluating the delays of the inverters in the loop. These delays are further used for computing intrinsic setup and hold times. HITAS just assumes that transitions on `d` and `db` are simultaneous.

7.7.3. Asymmetric Pulldown

Typical Structure



Latch Nodes and Commands

The latch nodes are here `m` and `mb`. The **Command** transistors are the ones which gate is controlled by **ck**.

Timing Arcs

Timing arcs are the following:

```
DATA: D (R) -> m (F), enabled by CK (R)
DATA: D (F) -> mb (F), enabled by CK (R)
DATA: m (F) -> mb (R)
DATA: mb (F) -> m (R)
ACCESS: CK (R) -> mb (R) - through the feedback loop
ACCESS: CK (R) -> m (R) - through th efeedback loop
SETUP: m (F) -> CK (F)
SETUP: mb (F) -> CK (F)
HOLD: m (F) -> CK (F)
HOLD: mb (F) -> CK (F)
```

The following timing arc is disabled: the NMOS transistor drive only a weak current in the latch node `m` - considering a rising transition on the latch node.

```
DATA: D (R) -> m (R), enabled by CK (R)
```

However, the contribution of the current through the NMOS is taken into account when evaluating the delay of the inverter between `mb` and `m`. This delay is further used for computing intrinsic setup and hold times.

7.8. Dynamic Latches

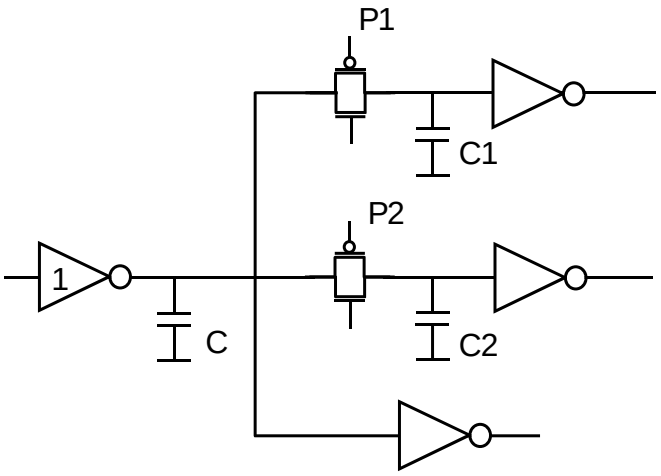
Dynamic latches are typically tristate nodes followed by a capacitance. In default mode, tristate nodes are not marked as latches. This behavior can be changed with the `yagMarkTristateMemory` configuration variable.

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

7.9. Special Elements

7.9.1. Transmission Gates

Transmission gate characterization is natively handled by HITAS, and most of the times it requires no additional configuration. However, there are some cases where correct characterization of the transmission gates depends on the functional behavior of the design, and therefore those cases can not be perfectly handled by static tool. The following diagram illustrates this typical case:



The difficulty here is to properly estimate the amount of capacitance C to be used for the computation of the propagation delay of the inverter 1. The value of this capacitance depends on the logic levels of P1 and P2 driving the transmission gates, while circuit is operating. It is then practically impossible for a static tool to determine exactly the amount of capacitance to be used, as C can change with different combinations of logic levels on P1 and P2. The best thing a static tool can do is to compute min and max capacitances, corresponding to 'all transmission gates closed' and 'all transmission gates opened'.

However, the combinations of logic levels on signals driving the transmission-gates may be limited, and may obviously not lead to the worst case situation, e.g. where all the transmission gates are opened. For example in the case of multiplexers or routing matrices, only one of N transmission gates is opened at a time. HITAS does not provide an automatic mechanism to detect those kinds mutual exclusion situations, but provides two variables to control them manually:

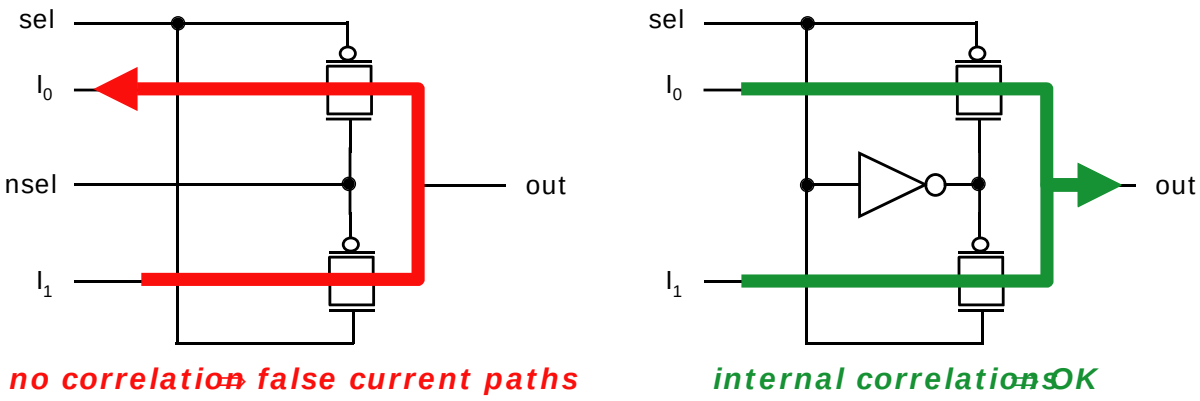
`tasPathCapacitanceFactor` globally controls the ratio of transmission-gates that can be opened at the same time, when determining the capacitance to be used to compute maximum propagation delays. Minimum propagation delays are computed with a ratio of 0.

`tasSwitchCapacitanceFactor` globally controls the ratio of transmission-gates that can be opened at the same time, when determining the capacitance to be associated with input connectors.

7.9.2. Transmission Gate Multiplexers

The detection of multiplexors is done purely algorithmically. The cone partitioning strategy implemented in HITAS 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.

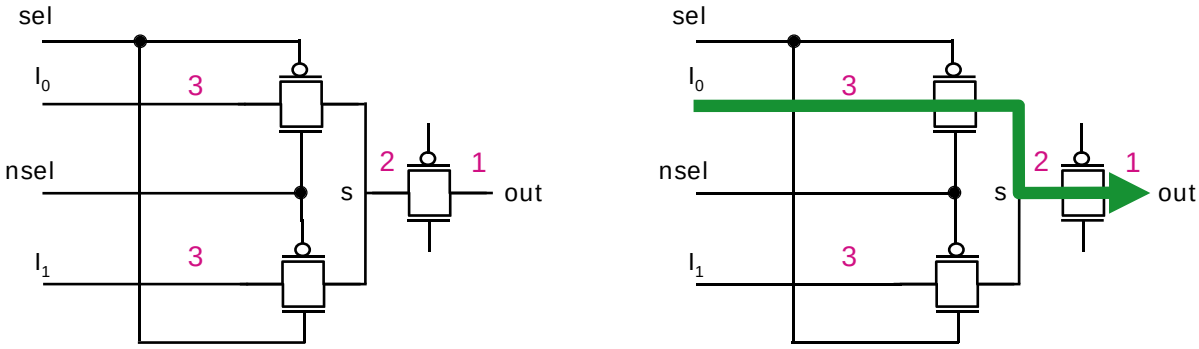
The following diagram shows two situations. In the left-hand design, the mutual exclusion between `sel` and `nsel` is not ensured by the design. There is no way for the tool to identify inputs and outputs, and it constructs false current paths. In the right-hand design, the mutual exclusion between `sel`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

7.9.3. Domino Precharge

The following configuration enables the automatic precharge detection algorithm.

```
avt_config yagDetectPrecharge yes
```

However, precharged elements and latches may present very similar structures, and it is not impossible that the tool mix up between them. In this case, it is more careful to define manually the pre-charged structures. Example is given for a precharged signal named `pre1`, within a design named `my_design`:

```
inf_SetFigureName my_design
inf_DefinePrecharge "pre1"
inf_DefineDirout "pre1"
```

In any case, the following configuration line should be added to enable HITAS to perform special treatment on the precharged signals.

```
avt_config tasTreatPrecharge yes
```

7.10. Case Analysis

Case analysis, such as Scan Mode analysis or Functional mode analysis, is available in the HITAS 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
```

7.11. Integrating External Timing Abstractions

The most straightforward way to integrate 3rd party abstractions is to preload them with `avt_LoadFile` function. In the following example, the transistor-level description of the sub-circuit `mult` is not available. A `.lib` file is used as substitution.

```
avt_LoadFile charac/mult.lib lib
avt_LoadFile cells/*.spi spice
avt_LoadFile my_design.spi spice
set fig [hitas my_design]
```

It is important that the `mult` sub-circuit is not among the sub-circuits loaded in Spice format, i.e. in the `my_design.spi` file. If the `mult` sub-circuit has been loaded in Spice format, it should be blackboxed, using either the BLACKBOX file mechanism or the `avt_SetBlackBoxes` Tcl function.

The BLACKBOX file mechanism works as follow:

As soon as a BLACKBOX file exists in one the directories reachable from HITAS (i.e. the directories specified by the `avtLibraryDirs` variable), HITAS systematically reads it and blackboxes all the sub-circuits it contains. No variable controls the effective blackboxing of the sub-circuits the BLACKBOX file contains. The mere presence of the BLACKBOX file presence is a sufficient condition.

If `avt_SetBlackBoxes` Tcl function is used, the BLACKBOX file mechanism is disabled.

If BLACKBOX file is required, the following line should appear in it:

```
mult
```

In addition, the following line should appear in the script:

```
avt_config tasTreatBlacboxHierarchically yes
```

Chapter 8. Timing DB Browsing

8.1. Timing DB

The following diagram gives the principal Tcl functions for timing database manipulation and path browsing. These are sorted according to their principal argument type.

DESIGN	→	hitas		→	**TimingFigure**
UTD on disk	→	ttv_LoadSpecifiedTimingFigure		→	**TimingFigure**
TimingFigure	→	ttv_GetTimingFigureProperty	TOP_LEVEL	→	**TimingFigure**
			FIGNAME	→	***string***
			TEMP	→	**numerical**
			DEF_SUPPLY	→	**numerical**
			DEF_LOAD	→	**numerical**
			DEF_SLOPE	→	**numerical**
			TH_LOW	→	**numerical**
			TH_HIGH	→	**numerical**
			TECH_NAME	→	***string***
			DATE	→	***string***
			TIME	→	***string***
	→	ttv_GetTimingSignal		→	**TimingSignal**
	→	ttv_GetTimingSignalList		→	**{ TimingSignal}**
	→	ttv_GetClockList		→	**{ TimingSignal}**
	→	ttv_GetPaths		→	**{ TimingPath}**
	→	ttv_DetectFalseClockPath			
	→	ttv_DisplayClockPathReport			
	→	ttv_LoadCrosstalkFile			
	→	ttv_DisplayConnectorToLatchMargin			
	→	stb_LoadSwitchingWindows		→	**StabilityFigure**
	→	stb		→	**StabilityFigure**
	→	stb_DisplaySlackReport			
	→	tma_Abstract		→	**TimingFigure**
{ TimingFigure	→	lib_DriveFile			
		tlf_DriveFile			
TimingPath	→	ttv_GetTimingPathProperty	DELAY	→	**numerical**
			REF_DELAY	→	**numerical**
			DATA_LAG	→	**numerical**
			SLOPE	→	**numerical**
			REF_SLOPE	→	**numerical**
			START_TIME	→	**numerical**
			START_SLOPE	→	**numerical**
			START_SIG	→	**TimingSignal**
			END_SIG	→	**TimingSignal**
			COMMAND	→	**TimingEvent**
			ACCESS_COMMAND	→	**TimingEvent**
			START_TRAN	→	***string***
			END_TRAN	→	***string***
	→	ttv_GetParallelPaths		→	**{ TimingPath}**
	→	ttv_RemoveDuplicatedPath		→	**{ TimingPath}**
	→	ttv_DisplayPathDetail			
	→	ttv_GetPathDetail		→	**TimingDetail**
{ TimingPath}	→	ttv_FreePathList			
	→	ttv_DisplayPathList			
	→	ttv_DisplayPathListDetail			

8.2. Details Browsing

The following diagram gives the principal Tcl functions for detailed browsing of timing database elements. These are sorted according to their principal argument type.

TimingDetail	→	ttv_GetTimingDetailProperty	HZ	→	numerical
			NODE_NAME	→	*string*
			SIGNAL_NAME	→	*string*
			SIGNAL_TYPE	→	*string*
			DELAY	→	numerical
			SLOPE	→	numerical
			REF_DELAY	→	numerical
			REF_SLOPE	→	numerical
			SIM_DELAY	→	numerical
			SIM_SLOPE	→	numerical
			DATA_LAG	→	numerical
			TYPE	→	*string*
	→	ttv_DriveSetupHoldSpiceDeck			
	→	ttv_DriveSpiceDeck			
TimingSignal	→	ttv_GetTimingSignalProperty	CLOCK	→	numerical
			DIR	→	*string*
			CAPA	→	numerical
			LEFT_BOUND	→	numerical
			RIGHT_BOUND	→	numerical
			TOP_LEVEL	→	**TimingFigure**
			NET_NAME	→	*string*
			NAME	→	*string*
			TYPE	→	*string*
			EVENT_UP	→	**TimingEvent**
			EVENT_DOWN	→	**TimingEvent**
			RISING_SLOPE	→	numerical
			FALLING_SLOPE	→	numerical
	→	ttv_GetLatchAccess		→	numerical
	→	ttv_GetLatchSetup		→	numerical
	→	ttv_GetLatchHold		→	numerical
	→	ttv_GetSignalCapaList		→	{numerical}
	→	ttv_GetFullSignalNetName		→	*string*
	→	ttv_GetFullSignalName		→	*string*
	→	ttv_GetLatchCommands		→	**TimingEvent**
	→	stb_GetSignalHold		→	numerical
	→	stb_GetSignalSetup		→	numerical
	→	stb_GetSignalStabilityPaths		→	**{StabilityPath}**
	→	stb_DisplaySignalStabilityReport			
{TimingSignal}	→	stb_DisplayErroneousSignals			
TimingEvent	→	ttv_GetTimingEventProperty	SIGNAL	→	**TimingSignal**
			TRANS	→	*string*

8.3. STA Browsing

The following diagram gives Tcl functions for browsing the results of an STA or crosstalk analysis. These are sorted according to their principal argument type.

StabilityFigur	→	stb_FreeStabilityFigure			
	→	stb_GetErrorList		→	{TimingSignal}
	→	ctk_DriveStatCtk			
	→	ctk_BuildCtkStat			
	→	ctk_LoadAggressionFile			
	→	ctk_GetAggressorList		→	{Aggressor}
StabilityPath	→	stb_GetStabilityPathProperty	SOURCE_SIG	→	TimingSignal
			ERROR_SIG	→	TimingSignal
			CK_NAME	→	*string*
			CK_PERIOD	→	numerical
			SETUP	→	numerical
			HOLD	→	numerical
			SETUP_MARGIN	→	numerical
			HOLD_MARGIN	→	numerical
			CMD_NAME	→	*string*
	→	stb_GetClockTime		→	numerical
	→	stb_GetClockDelta		→	numerical
	→	stb_DisplayClock			
	→	stb_DisplayInfos			
	→	stb_DisplayInputInfos			
	→	stb_GetInputInstabilityRanges		→	{StabilityRange
	→	stb_GetOutputInstabilityRanges		→	{StabilityRange
	→	stb_GetOutputSpecificationRanges		→	{StabilityRange
	→	stb_GetInstabilityRangeStart		→	{StabilityRange
	→	stb_GetInstabilityRangeEnd		→	{StabilityRange
	→	stb_DisplayInputInstabilityRanges			
	→	stb_DisplayOutputInstabilityRanges			
	→	stb_DisplayOutputSpecificationRanges			
{StabilityPath}	→	stb_FreeStabilityPathList			
Aggressor		ctk_GetAggressorProperty	SIGNAL		TimingSignal
			NETNAME		*string*
			CAPA_CTK		numerical
	→		DELAYBESTAGR		*string*
			DELAYWORSTAGR		*string*
			NOISERISE		*string*
			NOISEFALL		*string*
{Aggressor}	→	ctk_FreeAggressorList			

Chapter 9. Static Timing Analysis

9.1. Performing the Analysis

The Static Timing Analysis (sometimes also referred to as Stability Analysis) is performed upon the UTD timing database. The user must provide a timing constraints file, typically an SDC (Synopsys Design Constraints) file, or its INF equivalent, which should at least provide the two following sets of information:

- The definitions of all the external clocks and the global clock period.
- Constraints specifications for the I/O connectors (corresponding to arrival and departures times).

For the first set of information, clock defintions can be specified with functions `create_clock`, `create_generated_clock` and `set_clock_latency`.

For the second set of information, switching windows for the input connectors must be specified with function `set_input_delay`. Switching window definition for the inputs is mandatory to allow their propagation throughout the design. In addition, constraints on the output connectors can be specified with the `set_output_delay` function.

The STA engine is invoked with the function `stb`, taking as argument a pointer on the timing database (`TimingFigure` object). A pointer on a `TimingFigure` object can be obtained as the result of the `hitas` function. If the timing database has already been created and exists on disk (DTX and STM files), then a pointer on the `TimingFigure` object can be obtained with the `ttv_LoadSpecifiedTimingFigure` function. Here is an example of the launch of the STA with a minimal set of constraints:

```
set fig [ttv_LoadSpecifiedTimingFigure cpu]
inf_SetFigureName cpu
create_clock -period 900 -waveform {0 450} clk
set_input_delay -rise 850 -clock clk *
stb $fig
```

The `stb` function first propagates the interface clocks onto the commands of the latches. Then it propagates the switching windows defined on input connectors through the elements of the database, either combinational or sequential, over one clock cycle.

When propagating through combinational elements (gates), `stb` calculates the switching window on the gate's output by just summing up the gate's intrinsic delays to the switching window on its inputs. Depending on the kind of analysis, detailed or not, (`stbDetailedAnalysis`), the tool merges or not disjoint switching windows on the gate's output (see diagram below). Detailed analysis has no impact on setup/hold calculations but is mandatory in subsequent crosstalk analysis

When propagating through sequential elements (latches or precharges), the tool calculates the switching window on the latch's output by propagating the switching window on its input, with regard to the arrival times of the clock on the latch.

Once the `stb` function terminates, the timing database is annotated with stability information, i.e. each reference point of the database (I/O connectors, latches, commands, precharge) is annotated with its propagated switching window.

The comparison of the switching windows on the latch nodes, to the arrival times of the clocks, define the setup/hold slacks on the latches. The comparison of the switching windows on the output connectors, to the `set_output_delay` constraints, define the setup/hold slacks on the output connectors.

9.2. Output Files

With default behavior, the STA engine generates two files:

- `.sto` file: switching windows for each signal of the design under analysis.
- `.str` file: setup and hold slack report computed for all the reference points of the design under analysis.

9.3. Tcl Reports

The slack report is also available through the Tcl function `stb_DisplaySlackReport`. This function generates the same kind of report as the `.str` file, but with more customizable and detailed information.

9.4. Timing Checks

The next sections explain how timing checks are performed. They describe the more common situations one can be faced to, i.e.:

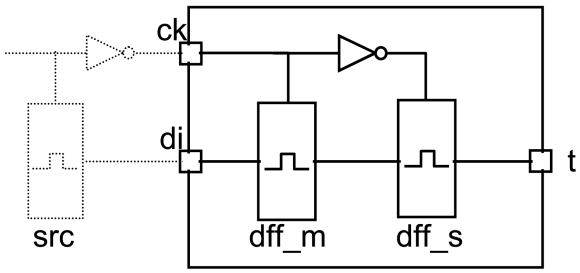
- Input to latch
- Latch to latch
- Latch to output

For each situation, an example of slack report is shown, and we explain the details of the timing checks calculation.

9.4.1. Input to Latch

Inputs Specifications

Regarding input specifications, the STA engine of HITAS makes the assumption that input data is coming from a latch clocked on the opposite phase of the one the data arrives on. In our flip-flop example, `dff_m` is opened on the high state of `ck`, so `di` is supposed to come from a latch opened on the low state of `ck`.

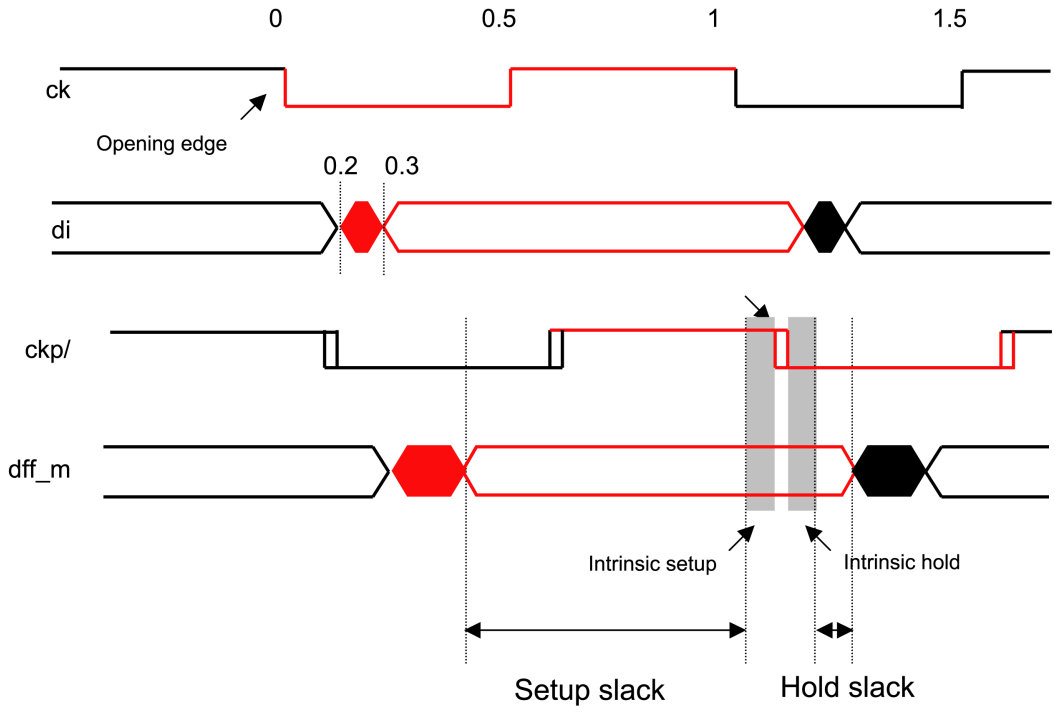


As a result, `di` should be specified as coming from `ck` falling, i.e. when the latch `src` opens. The corresponding SDC commands should look like:

```
create_clock -period 1000 -waveform {500 0} ck
set_input_delay -clock -ck -clock_fall -min 200 di
set_input_delay -clock -ck -clock_fall -max 300 di
```

Timing Checks Description

Diagram below illustrates the way `set_input_delay` directives are propagated throughout the design, and where timing checks are performed.



Setup Slack

Input to latch setup slack report is described in the `slack.rep` file

```
Path (4) :  Slack of 0.762
     DATA VALID:
        Delay
      Acc     Delta     R/F      Cap[pf]    Type    Node_Name          Net_Name       Line
    ------------------------------------------------------------------------------------------
     0.300    0.000 0.200 R       0.034             di                 di
```

```
  0.498   0.198 0.310 F      0.028    (L)   dff_m              dff_m       master
_____________________________________________________________________________
  0.498   0.198            (total)

  DATA REQUIRED:
     Delay
   Acc    Delta    R/F     Cap[pf]   Type   Node_Name          Net_Name    Line
_____________________________________________________________________________
  0.000   0.000 0.200 F      0.016    (C)   ck                 ck
  0.239   0.239 0.258 R      0.046   (CK)   ckn                ckn          inv
  0.340   0.101 0.140 F      0.036   (CK)   ckp                ckp          inv
  0.260  -0.081                             [INTRINSIC SETUP]
  1.260  +1.000                             [NEXT PERIOD]
_____________________________________________________________________________
  1.260   0.260            (total)
```

The value of the setup slack is given by clock_path - data_path = 1260ps - 498ps = 762ps. The intrinsic setup corresponds to an additional delay which models the amount of time required for secure memorization of the data.

Hold Slack

Input to latch hold slack report is described in the `slack.rep` file

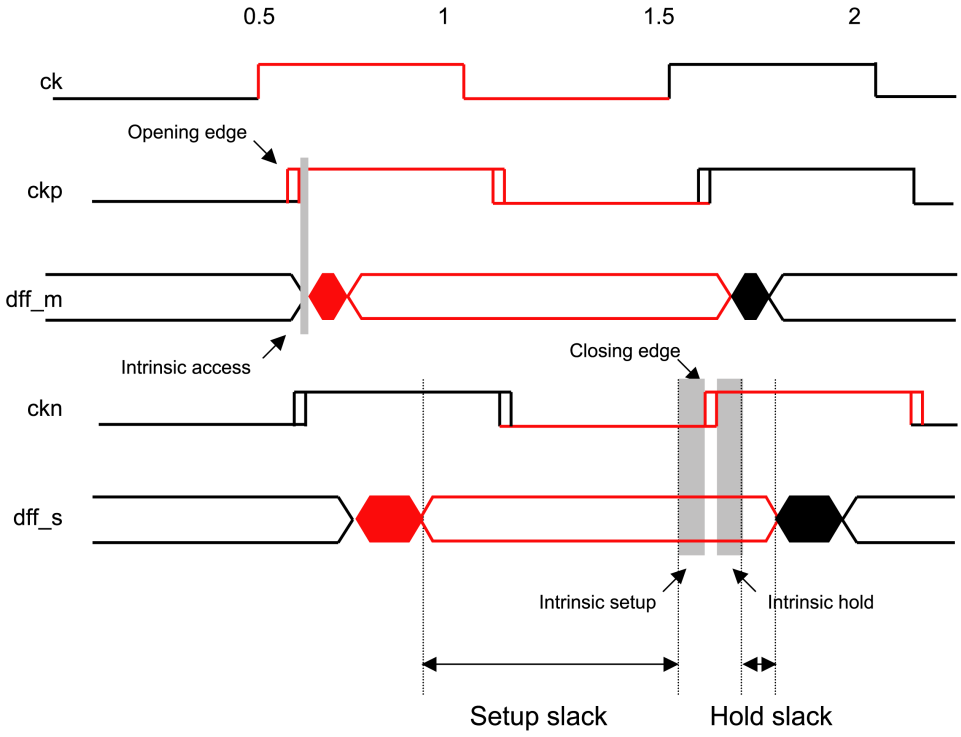
```
Path (2) :  Slack of 0.005
     DATA VALID:
        Delay
      Acc    Delta    R/F     Cap[pf]   Type   Node_Name          Net_Name      Line
_____________________________________________________________________________
     0.200   0.000 0.200 F      0.034          di                 di
     0.542   0.342 0.508 R      0.028    (L)   dff_m              dff_m       master
_____________________________________________________________________________
     0.542   0.342            (total)

     DATA REQUIRED:
        Delay
      Acc    Delta    R/F     Cap[pf]   Type   Node_Name          Net_Name    Line
_____________________________________________________________________________
     0.000   0.000 0.200 F      0.016    (C)   ck                 ck
     0.239   0.239 0.258 R      0.046   (CK)   ckn                ckn          inv
     0.537  +0.298                             [INTRINSIC HOLD]
_____________________________________________________________________________
     0.537   0.537            (total)
```

The value of the hold slack is given by data_path - clock_path = 542ps - 537ps = 5ps. The intrinsic hold corresponds to an additional delay which models the amount of time required for ensuring that the next cycle's data is not memorized in the current cycle.

9.4.2. Latch to Latch

Timing Checks Description

Latch to latch timing checks require no additional configuration, as they are based upon the signals already propagated from inputs, and upon the clock specification. The propagation of the s.w., and corresponding timing checks are described in the following timing diagram:



Setup Slack

Latch to latch setup slack report is described in the `slack.rep` file

```
Path (3) :  Slack of 0.284
     DATA VALID:
        Delay
         Acc    Delta    R/F     Cap[pf]   Type   Node_Name            Net_Name     Line
    ------------------------------------------------------------------------------------
      -0.500   0.000 0.200 R       0.016    (C)   ck                   ck
      -0.399   0.101 0.128 F       0.046   (CK)   ckn                  ckn           inv
      -0.236   0.164 0.169 R       0.036   (CK)   ckp                  ckp           inv
      -0.152   0.083 0.139 F       0.028    (L)   dff_m                dff_m      master
       0.090   0.242 0.189 R       0.040          n11                  n11           inv
       0.321   0.231 0.305 F       0.089    (L)   dff_s                dff_s       slave
    ------------------------------------------------------------------------------------
       0.321   0.821             (total)

       DATA REQUIRED:
          Delay
           Acc    Delta    R/F     Cap[pf]   Type   Node_Name            Net_Name   Line
    ------------------------------------------------------------------------------------
       0.500   0.000 0.200 R       0.016    (C)   ck                   ck
       0.601   0.101 0.128 F       0.046   (CK)   ckn                  ckn         inv
       0.605  +0.005                             [INTRINSIC SETUP]
    ------------------------------------------------------------------------------------
       0.605   0.105             (total)
```

Hold Slack

Latch to latch hold slack report is described in the `slack.rep` file

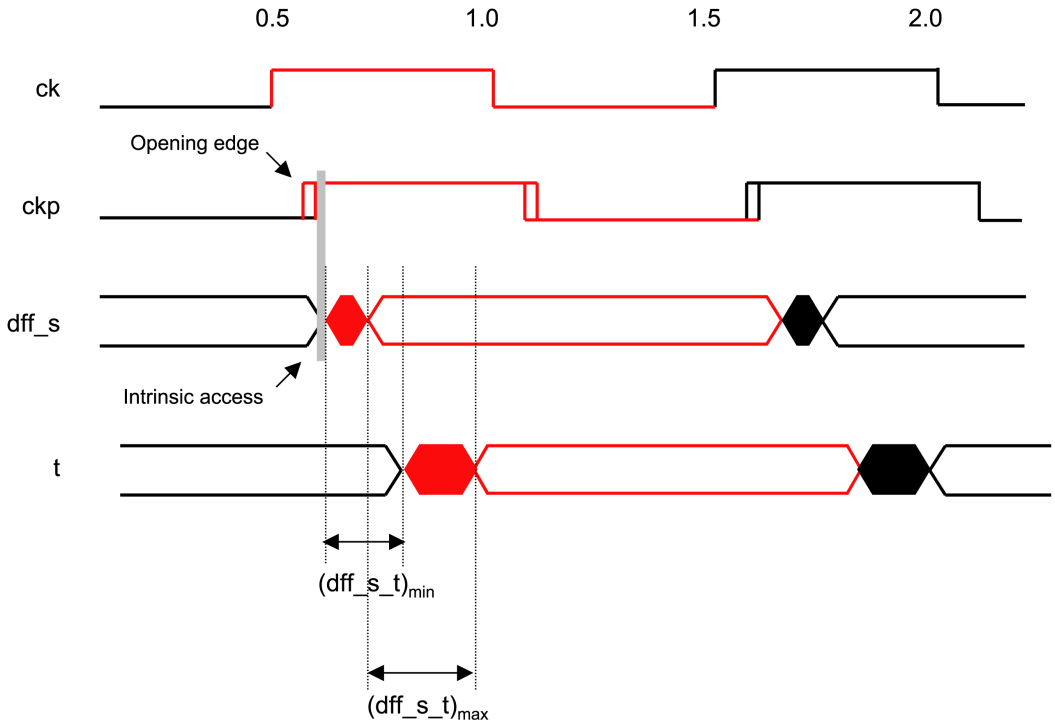
```
Path (3) :  Slack of 0.146
     DATA VALID:
        Delay
         Acc    Delta    R/F     Cap[pf]   Type   Node_Name          Net_Name     Line
```

```
-0.500   0.000 0.200 R      0.016    (C)   ck                  ck
-0.399   0.101 0.128 F      0.046   (CK)   ckn                 ckn           inv
-0.281   0.119 0.177 R      0.028    (L)   dff_m               dff_m      master
-0.223   0.057 0.088 F      0.040          n11                 n11           inv
 0.106   0.329 0.447 R      0.089    (L)   dff_s               dff_s       slave
-------------------------------------------------------------------------------
 0.106   0.606            (total)

 DATA REQUIRED:
    Delay
  Acc    Delta    R/F     Cap[pf]   Type   Node_Name           Net_Name   Line
-------------------------------------------------------------------------------
 0.500   0.000 0.200 R      0.016    (C)   ck                  ck
 0.601   0.101 0.128 F      0.046   (CK)   ckn                 ckn         inv
 0.764   0.164 0.169 R      0.036   (CK)   ckp                 ckp         inv
 0.960  +0.196                             [INTRINSIC HOLD]
-0.040  -1.000                             [PREVIOUS PERIOD]
-------------------------------------------------------------------------------
-0.040   0.460            (total)
```

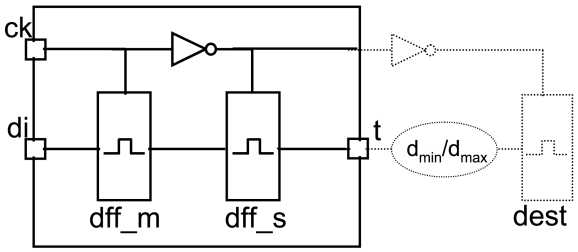
9.4.3. Latch to Output

Output Constraints

Still based on the flip-flop design described above, the timing propagation on output `t` is done as follow:



In order to get setup and hold slacks on the output, one must define timing constraints on `t`. These timing constraints are defined with the `set_output_delay` SDC function. The `set_output_delay` specifies propagation delays from output connector to the next memory element latching the data. As a result, min and max delays are defined as shown in the diagram below.



One must also define the edge the data will be latched by. Here, `dff_s` is closed on the high state of `ck`. The data launched by `t` is supposed to be latched by a memory element clocked on the opposite phase, i.e. closed on low state of `ck`. Therefore, constraints on `t` should be specified relative to falling edge of `ck` (when `dst` latch closes). The `set_output_delay` functions should be used as follow:

```
set_output_delay -clock ck -clock_fall -min 200 t
set_output_delay -clock ck -clock_fall -max 400 t
```

Setup Slack

Latch to output setup slack report is described in the `slack.rep` file

```
Path (1) :  Slack of 0.030
     DATA VALID:
        Delay
      Acc    Delta    R/F     Cap[pf]   Type   Node_Name    Net_Name      Line
    ------------------------------------------------------------------------------
     0.000   0.000 0.200 F      0.016    (C)   ck           ck
     0.239   0.239 0.258 R      0.046   (CK)   ckn          ckn            inv
     0.340   0.101 0.140 F      0.036   (CK)   ckp          ckp            inv
     0.568   0.227 0.327 R      0.089    (L)   dff_s        dff_s        slave
     0.570   0.003 0.118 F      0.011    (S)   t            t              inv
    ------------------------------------------------------------------------------
     0.570   0.570            (total)

    -> Specification: Must be stable after 0.600
```

The setup time is calculated with the maximum set_output_delay value - maximum data path - which is 400ps. As the period is 1000ps, data must arrive before time 1000 - 400 = 600ps. The setup slack is given by 600 - 570 = 30ps.

Hold Slack

Latch to output hold slack report is described in the `slack.rep` file

```
Path (5) :  Slack of 0.635
     DATA VALID:
        Delay
      Acc    Delta    R/F     Cap[pf]   Type   Node_Name    Net_Name     Line
    ------------------------------------------------------------------------------
     0.000   0.000 0.200 F      0.016    (C)   ck           ck
     0.239   0.239 0.258 R      0.046   (CK)   ckn          ckn           inv
     0.385   0.146 0.235 F      0.089    (L)   dff_s        dff_s       slave
     0.435   0.050 0.082 R      0.011    (S)   t            t             inv
    ------------------------------------------------------------------------------
     0.435   0.435            (total)

    -> Specification: Must be stable before -0.200
```

The hold time is calculated with the minimum set_output_delay value - minimum data path - which is 200ps. The hold slack is given by data path - clock path = 435 + 200 - 0 (the clock is ideal in the set_output_delay definition) = 635ps.

9.5. Skew Compensation

When computing hold slack values between two latches, taking into account the clock skew on the full clock tree may lead to excessive pessimism.

The hold slack is the difference between the data arrival time and the clock arrival time on a latch. The data is supposed to remain stable until after the latch has closed, i.e. the data is supposed to arrive after the time the clock arrives. In the case of a master to slave data path, we have:

```
hold_slack = data_time - ck_to_slave
```

The data comes from the opening of the master latch, so we can express the data arrival time as follow:

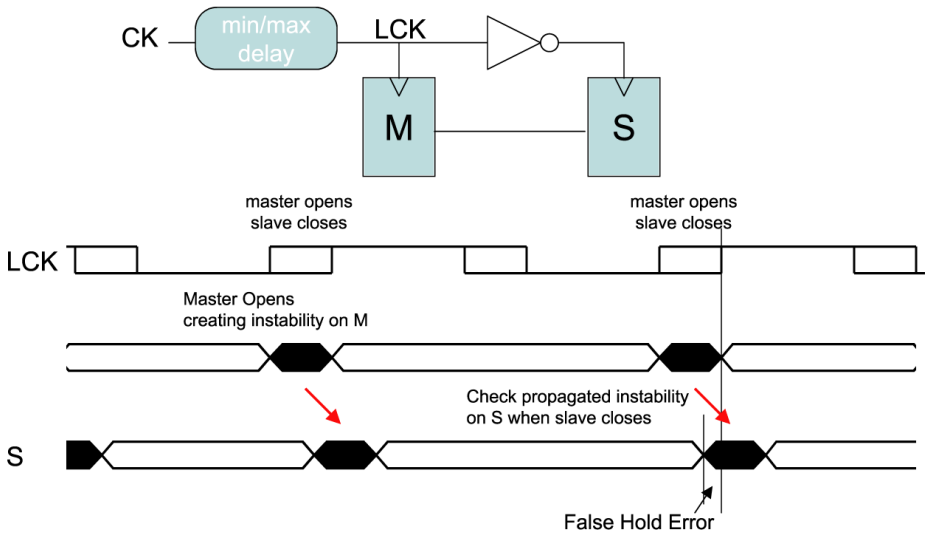
```
data_time = ck_to_master + master_to_slave
```

All in all, we have: (minimized)

```
hold_slack = ck_to_master + master_to_slave - ck_to_slave
hold_slack_min = ck_to_master_min + master_to_slave_min - ck_to_slave_max
```

In this case, paths from clock to master and from clock to slave are almost identical, until the node where they diverge towards master and slave. However, static timing analysis hypothesis may lead to significant differences between min and max propagation delays on a given path (especially when crosstalk effects are taken into account).

In the case of a hold slack, we check a data coming out from a master latch (opening), against the memorizing event of a slave latch (closing). What should be emphasized here is that a single clock edge generates both the opening event of the master latch and the closing event of the slave latch. The signal is propagating through the common part of the clock-to-master and clock-to-slave paths, and diverges to the master and the slave. Therefore, considering min and max propagation delays on the clock paths only makes sense on the parts of the paths which are not common to the clock-to-master and clock-to-slave paths. The following diagram illustrates this situation:



HITAS supports this situation by a skew compensation mechanism. The global difference between clock-to-master (minimum) and clock-to-slave (maximum) is computed. The difference relative to the common part is removed afterwards. It appears in the slack report tagged as `[SKEW COMPENSATION]`.

9.6. Multicycle Paths

HITAS supports multicycle paths through the SDC commands `set_multicycle_path` as described in this section. Multicycle paths specifications typically allow the tool to perform timing checks for data which requires more than one clock cycle to reach its destination. Let's consider the diagram below, which summarizes what can be done with multicycle paths:

```
                ____      ____      ____      ____
ORIGIN_CK  ___|    |____|    |____|    |____|    |____|
           0         1         2         3         4

                ____      ____      ____      ____      ____
DEST_CK    ___|    |____|    |____|    |____|    |____|    |____|
           0         1         2         3         4         5
```

In the 1-cycle default case, checks are done as follow:

- Setup timing check: `ORIGIN(0)` vs. `DEST(1)`
- Hold timing check: `ORIGIN(0)` vs. `DEST(0)`

Let's consider a multicycle path of 5, if one just writes the command:

```
set_multicycle_path 5 -end -to DEST_DATA
```

or its equivalent:

```
set_multicycle_path 5 -end -setup -to DEST_DATA
```

This gives:

- Setup timing check: `ORIGIN(0)` vs. `DEST(5)`

- Hold timing check: `ORIGIN(0)` vs. `DEST(4)`

As you can see, by default the path multiplier (5 here) is applied to the setup check, effectively saying that the data is allowed 5x more clock cycles than the default of 1 to arrive at the destination latch. Note that changing the path multiplier for setup also affects the hold check since, by default, the hold check is 1 cycle before the setup check.

If you want to have the hold checks done as in the 1-cycle case, one must moves the hold check backwards by 4 cycles. This can be done by specifying a path multiplier of 4 for the hold chacks on the same paths as follows:

```
set_multicycle_path 4 -end -hold -to DEST_DATA
```

This gives:

- Setup timing check: `ORIGIN(0)` vs. `DEST(5)`
- Hold timing check: `ORIGIN(0)` vs. `DEST(0)`

9.7. Tips

9.7.1. Disabling Master-to-Slave Timing Checks

Since 2.7p6 release, there is a way to disable the setup/hold checks between master and slave latches, with a semi-manual method.

The SDC command `set_false_path` accepts the `-hold` and `-setup` options, so it is possible to disable setup/hold checks on specific latches through this command, for example:

```
set_false_path -setup -to "*.dff_s"
set_false_path -hold -to "*.dff_s"
```

This will disable setup/hold checks on all the latches matching the pattern `*.dff_s` (assuming the latch node's name is `dff_s`). This method assumes that you know the name of the latch node. Obviously it's not always the case, so HITAS also provides an automatic master/slave detection mechanism. This detection is done during the database construction (the `hitas` command), and is controlled by the following configuration:

```
avt_config yagleAutomaticFlipFlopDetection mark
```

The effect of this configuration is to report master/slave information about the latches in the `.rep` file, as follow:

```
[WRN 33] Loop between 2 gates at 409 'm1.dff_m' (master latch found)
[WRN 34] Loop between 2 gates at 411 'm1.dff_s' (slave latch found)
```

It is possible to extract the SDC commands from the `.rep` file with an AWK script looking like:

```
#!/usr/local/bin/gawk -f
{
  if ($2=="34]") {
    gsub ("'", "", $9);
    printf "set_false_path -setup -to "$9"\n" >"ms_chk.sdc";
```

```
      printf "set_false_path -hold -to "$9"\n" >"ms_chk.sdc";
    }
  }
```

9.8. On-Chip Variation

HITAS handles On-Chip Variation by considering additionnal margins on timing paths. An added margin is associated with a timing path and is defined by an absolute `delta` value and a relative `factor` value. The absolute value is an added or substracted propagation delay. The relative value is a percentage of the propagation delay of the timing path itself.

Both the absolute value and the relative value should be specified by the user, with the `inf_DefinePathDelayMargin` function:

```
path_delay + margin = path_delay * factor + delta
```

Those margins can be specified either path by path or by group of paths (data paths, clock paths, paths arriving on special nodes...).

HITAS also considers positive and negative margins (if `factor > 1`, the margin is positive, else it is negative). Positive margins are used when considering max paths, negative margins are used when considering min paths (for example STA computes setup slacks by considering max data path vs. min clock path, and the other way round for hold slacks).

9.9. Clock Schemes Handling

9.9.1. Clock Dividers

Under construction...

9.9.2. Pulse Generators

Under construction...

9.9.3. RS-based Clock Generators

Under construction...

Chapter 10. Crosstalk Analysis

10.1. Requirements

Crosstalk Analysis is done after the UTD construction performed with the `hitas` function. Only the detailed timing information file of the database is needed (DTX file), together with the interconnect information (RCX file) and the delay models (STM file).

The Crosstalk Analysis is coupled with the Static Timing Analysis, and is therefore performed with the same function `stb`. The inputs needed for crosstalk analysis are the same as the input needed for STA.

10.2. Understanding Crosstalk in STA

10.2.1. The Issues Involved

A coupling capacitance is a capacitance between two nets. Let us consider a signal carried by a net, which we shall call the victim. All other signals carried by nets linked to the victim via a coupling capacitance are called aggressors. A signal is considered to be 'quiet' when no transitions occur on it, and 'active' whenever transitions occur. We define 'quiet' or 'active' to be the state of the aggressor.

The influence of a coupling capacitance depends upon the state of victim and the state of the aggressor. From the point of view of the victim four cases exist:

- Victim and aggressor are both quiet. No coupling effect.
- Victim is active and aggressor is quiet. The coupling capacitance acts like a substrate capacitance.
- Victim is quiet and aggressor is active. Noise is generated on the victim.
- Victim and aggressor are both active. Propagation delay of the driver of the victim signal, and interconnect delay of the victim signal are affected by the aggressor.

In the last case, there are two different effects depending on victim and aggressor transitions:

- The signals switch in same direction. Driver delay and interconnect delay are reduced.
- The signals switch in opposite direction. Driver and interconnect delay are increased.

10.2.2. Algorithm

In order to compute the effect of a coupling capacitance, we need to know whether two signals can switch at the same time, and if they switch in same or opposite direction. The detailed Static Timing Analysis gives switching windows for all the edges (reference points and others) of a timing graph. Given a victim and its set of aggressors, it is therefore possible to determine switching window overlaps, and to detect aggressions inducing noise and delta-delays on the victim.

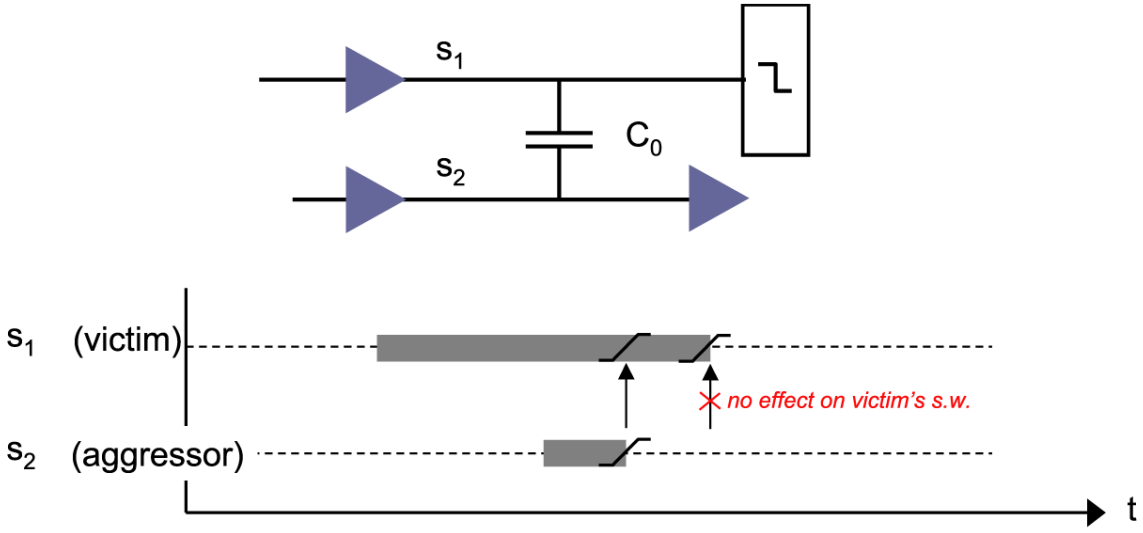
This approach requires that initial detailed switching windows propagation is performed from initial delay values without any knowledge of aggressor behavior. Two options are possible:

- The first is to consider an initial state with no aggression (default mode, can be enhanced with the `stbCtkObservableMode` variable)
- The second is to consider an initial state with all possible aggressions (the `stbCtkWorstBeginCondition` variable)

It is then possible to compute initial values for driver delays and interconnect delays, and perform an initial switching windows propagation. For each victim, the algorithm creates its effective list of aggressors, by analyzing the overlap of the switching windows of its aggressors.

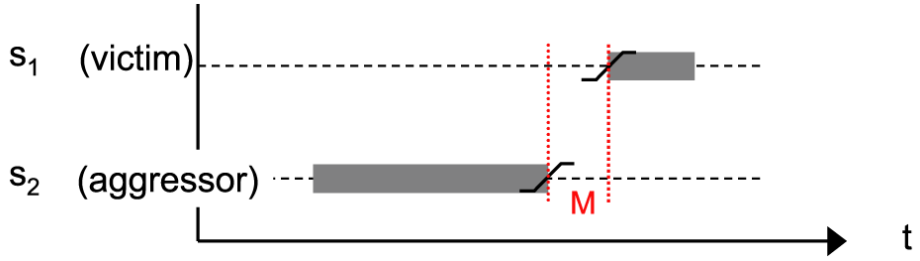
Depending on the chosen initial state, the list of aggressors will either increase or decrease. In both cases, the effective coupling capacitance seen by the victim will change, inducing a re-evaluation of the driver delay and interconnect delay of each victim. Another STA (switching window propagation) is performed with these updated delays. If this second propagation induces no change in the aggressor lists, the algorithm finishes. If this is not the case, further iterations are performed until the algorithm converges.

To render crosstalk analysis less pessimistic, it is possible to refine the modeling of the effect of an aggression. In many cases, an aggression only changes the driver delay of the victim over a short time, switching windows not being significantly affected (victim's earliest and latest switches not being affected). This kind of aggression can then be considered as a non-observable aggression.



Ignoring non-observable aggression results in less pessimistic results, in regard of the setup and hold violations, only affected by the earliest and latest switches. The user can choose to enable analysis with only observable aggression with the `stbCtkObservab1leMode`variable.

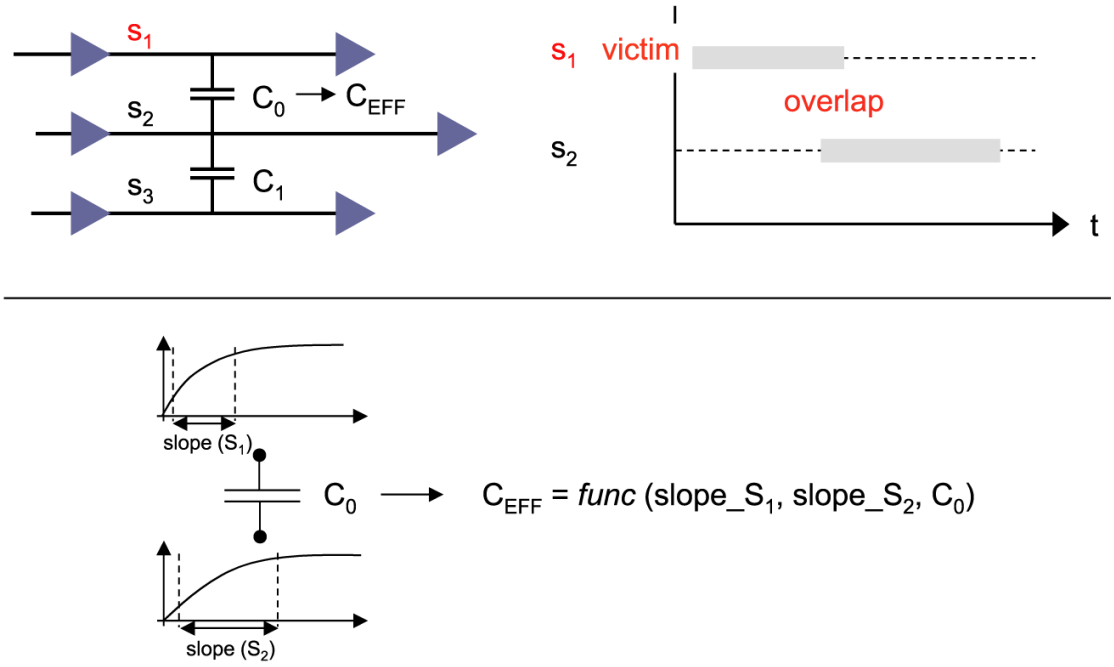
On the other hand, it might be usefull to consider very close - but non overlapping - switching windows as overlapping, as shown in the following example:



The `M` value is controlled by the variable `stbCtkMargin`.

10.2.3. Delay Calculation

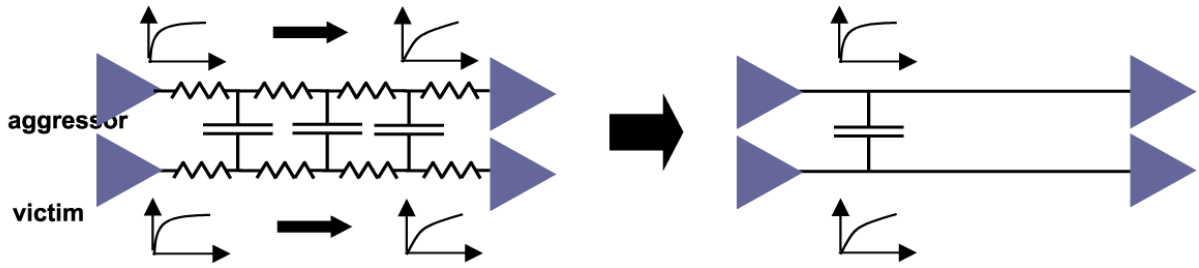
The main factor in computing crosstalk-aware delays is the proper modeling of the effective capacitance. In HITAS, the effective capacitance is computed by an enhanced Miller model, taking into account the relative strengthes of the drivers (actually the slopes at their outputs). The tool computes a crosstalk-aware delay by feeding this updated capacitance to the timing model of a driver.



In order to reduce computational time, HITAS makes the following assumptions:

- Resistances are not taken into account in the coupled network
- The coupling capacitance used in the Miller calculus is the sum of the distributed coupling capacitances on the net
- The Miller calculus uses the slope at the output of the driver for the aggressor, and the slope at the end of the RC network for the victim (maximizing the Miller effect)

The figure below illustrates those assumptions:



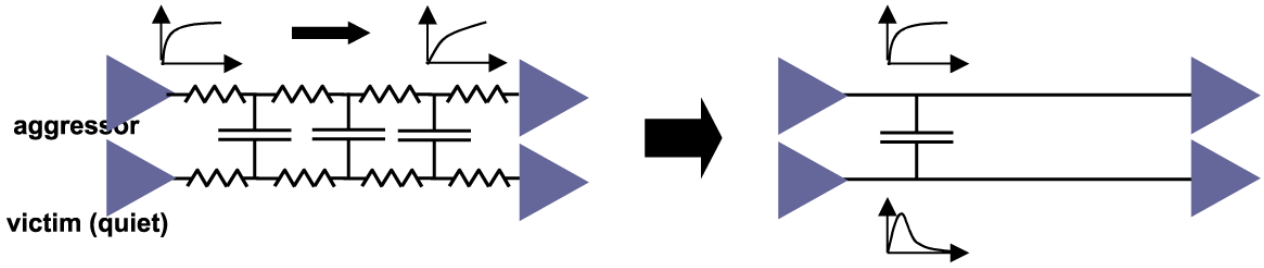
The most accurate delay calculation is done with the following configuration:

```
avt_config rcxCtkSlopeDelay SLOPE_DELAY_ENHANCED
```

10.2.4. Noise Calculation

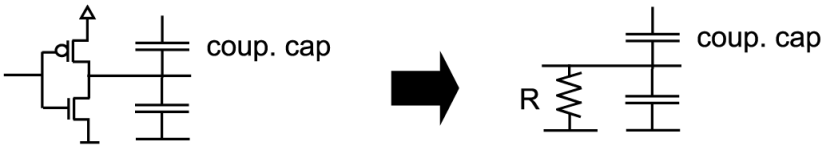
The figure below illustrates those assumptions:

HITAS computes voltage peaks on each net, by modeling the coupled network as illustrated in the figure below:

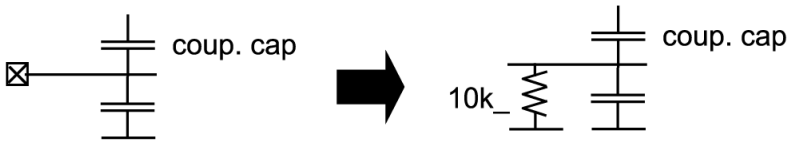


The assumptions made are:

- Resistances are not taken into account in the coupled network
- The coupling capacitance used is the sum of the distributed coupling capacitances on the net
- The slope used is the one at the output of the driver for the aggressor (this maximizes the noise)
- Multiple aggressors are handled by linear superposition of the noises they independantly produce
- The driver of the victim is modeled as a resistance, as shown in the figure below



SCR model: Single Constant Resistance



CC model: Connector Constant

The most accurate noise analysis is done with the following configuration:

```
avt_config rcxCtkSlopeNoise SLOPE_REAL
```

10.3. Running the Crosstalk Analysis

Relevant configuration:

`stbCrosstalkMode`	Activates the crosstalk analyzer within the STA engine. Aggressions are detected according the switching windows intersections. Multiple iterations are automatically performed until no more aggression is detected. This variable should be set to `yes`.
`stbDetailedGraph`	Aggressor detection has a meaning only on a detailed graph, (and not on a path graph). This variable should be set to `yes`.
`stbDetailedAnalysis`	Tunes the building of switching windows either to single or multiple windows per period. Less pessimistic results are obtained with multiple switching windows. It is better to set this variable to `yes`, but it requires more memory.
`stbCtkNoInfoActif`	If the aggressor is not a timing signal (eg: internal signal in a gate), there is no timing information for it. If this variable is set to `yes`, then this aggressor is considered to be always active, else it is considered to be always quiet. It is better to set this variable to `yes` (default).
`stbCtkReportFile`	Activates the generation of the report file (`.ctk`). Since Tcl functions provide an efficient way to browse crosstalk results, and considering the size of this file, this variable is better set to `no` (default).

The crosstalk analysis is done with the `stb` Tcl command, based on a previously generated timing DB. The timing DB may come from the `hitas` Tcl command:

```
set fig [hitas my_design]

avt_config stbCrosstalkMode yes
avt_config stbDetailedGraph yes
avt_config stbDetailedAnalysis yes

stb $fig
```

The timing DB may also come from disk, loaded with the `ttv_LoadSpecifiedTimingFigure` command:

```
set fig [ttv_LoadSpecifiedTimingFigure my_design]

avt_config stbCrosstalkMode yes
avt_config stbDetailedGraph yes
avt_config stbDetailedAnalysis yes

stb $fig
```

10.4. Output Files

Since HITAS crosstalk engine is coupled with the STA engine, it generates the same output files (STO and STR files). In addition, the crosstalk engine creates two or three extra files. The first one contains details of all crosstalk adjusted delays throughout the design hierarchy. This file `.ctx` is intended to be browsed using Xtas or the functions of the Tcl interface. The second file (`.agr`) contains the list of all nodes with the states of their aggressor. This file is intended to realize the report of crosstalk analysis with the Tcl interface. The third file is a human readable file (`.ctk` file) which contains all crosstalk related information. This file is drived if the `stbCtkReportFile` is set

The CTX file is an ASCII text file containing all the delays calculated with crosstalk effects of a complete design hierarchy. This file is associated with all of the original files describing the hierarchy (DTX, STM and RCX). It is intended to be viewed using the timing browser Xtas. This file contains top level delays and instance delays.

10.5. Browsing Crosstalk Analysis Results

10.5.1. Crosstalk Impact on Delays

After the run of the `stb` Tcl command, the timing DB contains nominal propagation delays and crosstalk-aware propagation delays. Therefore, when based upon a crosstalk-annotated timing DB, browsing commands such as `ttv_GetPaths` show crosstalk impact on propagation delays:

```
set fig [ttv_LoadSpecifiedTimingFigure my_design]
avt_config stbCrosstalkMode yes
stb $fig
set clist [ttv_GetPaths $fig * * uu 5 critic path max]
```

This `stb` command (the crosstalk analysis) generates the `.ctx` file for crosstalk impact on delay, suitable for further browsing, in order to dissociate crosstalk analysis and browsing. As crosstalk analysis may be cpu consuming, this will save time. It is then possible to browse crosstalk-annotated timing DBs generated from previous crosstalk analysis runs:

```
set fig [ttv_LoadSpecifiedTimingFigure my_design]
ttv_LoadCrosstalkFile $fig
set clist [ttv_GetPaths $fig * * uu 5 critic path max]
```

10.5.2. Crosstalk Noise

HITAS provides a set of Tcl functions for crosstalk noise analysis. Those functions work on the crosstalk database generated with the `stb` function. If the `stb` function has been launched in the current Tcl script, the crosstalk database is available in memory, and the Tcl noise analysis functions can be used directly:

```
set fig [ttv_LoadSpecifiedTimingFigure my_design]
avt_config stbCrosstalkMode yes
set stbfig [stb $fig]
ctk_DriveStatCtk $stbfig
```

If the `stb` function has been launched in a separate script, then the crosstalk database (`.ctx`, `.sto` and `.agr` files) must be loaded from disk before using noise analysis functions:

```
set fig [ttv_LoadSpecifiedTimingFigure my_design]
ttv_LoadCrosstalkFile $fig
set stbfig [stb_LoadSwitchingWindows $fig my_design.sto]
ctk_LoadAggressionFile $stbfig

ctk_BuildCtkStat $stbfig
ctk_DriveStatCtk $stbfig
```

The `ctk_BuildCtkStat` function is used to re-built consistent information from the crosstalk database files, according to the scoring configuration variables (see later in this chapter).

10.5.3. Browsing Information on Event

For noise and score, the Timing Event stores informations corresponding to the initial state of this transition : the `up` event correspond to the `low` logical level, and the `down` event correspond to the `high` logical level.

Crosstalk information on nodes are stored in an internaly table. This table can be sorted with the command `ctk_SortCtkStatNode`. Information are available through the command `ctk_GetStatNodeProperty`. The `index` parameter is the position in the internal table, from `1` to the value returned by the command `ctk_GetNumberOfCtkStatNode`. To get the position of a particular event, uses command `ctk_GetCtkStatNodeFromEvent`.

10.5.4. Browsing Local Crosstalk Impact on Delay

This information allow to know the elementary delays the most modified by crosstalk effect due to two net switching simultaneously. This information is stored in an internale table. This table can be sorted with the tcl command `ctk_SortCtkStatLine`. Information on delays are available with the tcl command `ctk_GetStatLineProperty`. The `index` parameter is a number from `1` to the value returned by the command `ctk_GetNumberOfCtkStatLine`.

10.5.5. Browsing Aggressor

The aggressor list of a Timing Event is available through the tcl command `ctk_GetAggressorList`. Property for an aggressor in this list are available with the tcl command `ctk_GetAggressorProperty`. This list must be freed with the tcl command `ctk_FreeAggressorList`.

The property `SIGNAL` return the Timing Signal corresponding to the aggressor. If the aggressor is an internal net of a gate, there is no Timing Signal built on it, then the property `SIGNAL` return `NULL`.

10.6. Score-Based Result Analysis

Signals presenting a risk of noise violation are sorted by a score based method. Four scores, ranging from 0 to 10, are reported. Scores assess both the crosstalk impact and the aggression occurrence probability. Scores are reported in the output files, and in the slack and path reports, under the following tags:

`C`	Number of significant aggressors; the closer to 10 is the score, the more significant part of crosstalk is due to a few number of aggressors.
`N`	Noise peak value; 0 means that the noise peak reaches or exceeds the logical threshold of at least one gate in the fanout.
`I`	Switching windows criteria; based upon number of aggressors simultaneously active
`A`	Activity of the aggressor; 10 means that aggressors belong to a clock path, or belong to another clock domain (and assumed to be always active); 0 means that nothing can be gathered about activity, it doesn't mean that aggressors are not active.

A total weighted score is also reported, under the T tag, combining the individual scores. The individual weighting of the C, N, I and A scores can be tuned with the variables `stbCtkCoefCtk`, `stbCtkCoefNoise`, `stbCtkCoefInterval` and `stbCtkCoefActivity`, respectively.

If a signal obtains a mark lower than the value specified in `stbCtkMinNoise`, `stbCtkMinInterval`, `stbCtkMinCtk` or `stbCtkMinActivity`, it is not displayed in the report file.

Chapter 11. Spice Deck Generation

The HITAS platform provides the means to generate a simulatable netlist of any timing path of the timing database. This generated simulatable netlist contains all the transistors that participate to the timing path. Actually, as a timing path runs through a set of cones (partitions), the simulatable netlist contains the transistors belonging to this set of cones. Transistors that do not directly participate to the path (out-of-path transistors) can be either modeled as blocked transistors or as equivalent capacitances.

The path transistor netlist comes with all the stimuli enabling the propagation of the signal through the path. The tool automatically generates the stimuli enabling the appropriate transitions. The path transistor netlist and the stimuli form what is called the Spice Deck.

The HITAS platform also provides the means to link with external Spice simulators for simulating the Spice Deck, and to get back the results for pertinent comparison of HITAS results. The accuracy of most of Spice simulators is heavily dependant on their operating mode (digital/analog) and on convergence configuration. Pertinent comparison of HITAS results should be made with simulators in the mode allowing the most accurate results.

Path simulation is not available for hierarchical timing databases.

11.1. Simulator Configuration

First a few variables need to be set to control spice deck generation. They are related to the simulator being used. Supported simulators are:

- ELDO (Mentor Graphics)
- HSPICE (Synopsys)
- TITAN (Infineon)
- NGSPICE (GEDA)

The minimal required configuration is as follow (example is given for ELDO):

```
# Spice deck target simulator
avt_config SimToolModel eldo

# Simulator invoking (command line)
avt_config avtSpiceString "/tools/eldo $"

# Transistor models (.INCLUDE to be added in spice deck)
avt_config SimTechnologyName bsim3_018.tech
```

Setting the `SimToolModel` variable to a specific simulator also controls the default value of the following variables:

- `avtSpiceOutFile`
- `avtSpiceStdOutFile`

- simSpiceOptions
- simExtractRule
- simMeasCmd

Be aware that setting values for those variable overwrite their default value.

11.2. Spice Deck Generation

A spice deck is related to a timing path, and spice deck generation is based upon the detail of the timing path. Therefore, one should be able to get a timing path from a timing database before generating the spice deck. The following operations must be performed:

```
set fig [ttv_LoadSpecifiedTimingFigure my_design]
set paths [ttv_GetPaths $fig * * ?? 1 critic path max]
set path [lindex $paths 0]

set detail  [ttv_GetPathDetail $path]
```

The spice deck generation is then done as follow:

```
ttv_DriveSpiceDeck $fig $detail "path.spicedeck"
```

This method only generates the spice deck file `path.spicedeck`. It does not allow the reading of the results for comparison. Thsi operation is described in the next section.

11.3. Spice Deck Simulation

Spice deck generation, simulator launch and result reading can be all done together within the function `ttv_DisplayPathListDetail`, given that it has been enabled through the `ttv_DisplayActivateSimulation` function. The `ttv_DisplayPathListDetail` requires a timing path as argument, but no timing detail. It gets it automatically:

```
set fig [ttv_LoadSpecifiedTimingFigure my_design]
set paths [ttv_GetPaths $fig * * ?? 1 critic path max]
set path [lindex $paths 0]

ttv_SetupReport "ps ff"
ttv_DisplayActivateSimulation y
ttv_DisplayPathListDetail stdout $path
```

This script performs the following actions:

- Gets the timing path `$path`
- Activates the simulation engine
- Generates the spice deck: `my_design_ext.spi` and `cmd_my_design_ext.spi`
- Performs the simulation: `/tools/eldo cmd_my_design_ext.spi`
- Reads the simulation result: `cmd_my_design_ext.chi`

- Prints the path detail on standard output, displaying HITAS and simulator delay and slope values

11.4. Out-of-path Transistors

The variable `simOutLoad` controls the way out-of-path transistors are modeled.

If `simOutLoad` is set to `dynamic`, the tool transforms out-of-path transistors into equivalent capacitances in the spicedeck. In such a case transistor models are needed for the grid and source/drain capacitances evaluations.

If `simOutLoad` is set to `transistor`, the tool does not transform out-of-path transistors into capacitances, but just print them in the spicedeck in a blocked configuration. In such a case the transistor models are not needed. One must be careful with this configuration, as the generated spice deck may be very big, for a very little accuracy gain.

The load of a file containing transistor models (`bsim3_018.tech`) at the beginning of the script is necessary when `simOutLoad` is set to `dynamic`.

Chapter 12. Analog Sub-circuit Characterization

12.1. Objective

HITAS is designed to compute propagation delays in digital designs. The advantage of this restrictive target is to enable very fast computing times. The drawback is that non-digital block characterization is not directly handled by HITAS and should be supplied to 3rd-party analog simulators. However, HITAS provides various ways to link with external characterizations.

First, the tool provides the means to choose between integrating a pre-characterization (such as a .lib or .dtx file) and integrating an on-the-fly characterization (by piloting an analog simulator).

The pre-characterization strategy is easier to set-up, but has the drawback to provide only one characterization for (potentially) several instances of an analog sub-circuit. The on-the-fly characterization is a little more difficult to set-up, but provides environment-dependant (PVT, input slopes, output loads) characterizations, and therefore greater accuracy.

Second, the tool is able to integrate the characterizations within either a pre-layout or a post-layout timing analysis.

In a pre-layout timing analysis, the source netlist is hierarchical. With appropriate directives, the tool just ignores the analog sub-circuits during the parse of the netlist. It then constructs the timing database on the remaining only-digital netlist, and fills the "analog holes" with either pre- or on-the-fly characterization.

In a post-layout timing analysis, the situation is a little trickier, as the source netlist is the most often a flat-transistor netlist (if the extracted netlist is hierarchical, the integration process is just the same as in a pre-layout timing analysis). The challenge is here to identify in the top-level netlist the transistors that belong to the analog sub-circuits, in order to ignore them and create the "analog holes" and the remaining only-digital netlist. Assuming that the pre-layout netlist of the analog sub-circuit exists, the identification in the top-level netlist of the transistors that belong to that sub-circuit is done by pattern matching.

12.2. Pre-Characterization

In this section, we present how to create a DTX file within a Tcl script, and how to get timing values from analog simulation results. We then present how to integrate this pre-characterization in the top-level timing database construction, the latest either being based upon a hierarchical or flat transistor netlist.

The pre-characterization approach implies that this DTX file should be created before invoking HITAS on the entire design.

12.2.1. Database Construction

The API functions provided with the `avt_shell` Tcl interface enable the creation of custom timing databases (DTX files). We will just present here a few of those functions , as well as the global construction mechanism.

The `ttv_CreateTimingFigure` function creates the timing database itself. The function takes a netlist as argument, and builds the interface of the new timing database upon the interface of the netlist provided. If the new timing database is intended to replace an analog sub-circuit, it ensures interface consistency between the "hole" in the top-level netlist and the newly created timing database. Note that the Vdd and Vss names must be specified if they appear on the interface.

The `ttv_AddTiming` functions add timing arcs in the database.

The `ttv_FinishTimingFigure` function updates the database.

The `ttv_DriveTimingFigure` function prints the database on disk (DTX file).

See HITAS Tutorial for example.

12.2.2. Simulator Linking

The construction script described above associates "hard" values to the delays and slopes given as parameters of the `ttv_AddTiming` function (timing arc creation). It is the responsibility of the user to associate pertinent values to the parameters of the `ttv_AddTiming` function, by all the ways Tcl scripting provides.

However, the `avt_shell` Tcl interface also provides means to retrieve the values from analog simulations results. Provided functions cover:

- Stimuli description and formatting
- Simulator call
- Delay and slope values retrieving

All these functions require the same configuration as the one required for SPICE deck generation. See HITAS Tutorial for example.

12.2.3. Hierarchical Netlist Integration (Pre-Layout)

The first step to perform when integrating the pre-characterized DTX file within a hierarchical netlist is the blackboxing of the instances of the analog sub-circuits, in order to obtain "analog holes" in the netlist. This is done with the `avt_SetBlackBoxes` function, taking as argument the list of the sub-circuits to blackbox.

The default behavior of HITAS is not to try to fill the "holes". To tell the tool to fill the holes with timing characterizations, the `tasIgnoreBlackbox` variable should be set to `yes`

The timing arcs for the instances `rs_clock_gen` are directly integrated in the new database.

See HITAS Tutorial for example.

12.2.4. Flat Netlist Integration (Post-Layout)

Obtaining the "analog holes" for a flat netlist is a little more difficult than for a hierarchical netlist, as the blackboxing mechanism can not be applied. In order to create these holes, HITAS uses a method based on pattern-matching (the GNS pattern-matching engine). User must provide a transistor pattern (i.e. a transistor level netlist) of the analog sub-circuit which should be substituted by a custom timing database. The most common way is to provide the tool with the schematic SPICE netlist of the analog sub-circuit. The tool then tries to identify the set of transistors in the top-level netlist that matches the provided pattern. The tool then removes those transistors from the top-level netlist, thus creating the "analog holes".

The following steps must be performed to activate the pattern-matching method:

- Create a directory for the transistor patterns. The default one should be `cells`, located in the working directory. However, it is possible to give any name and location to this directory, assuming it is specified in the main script with the `gnsLibraryDir` variable, e.g.:

  ```
  avt_config gnsLibraryDir ../on-the-fly
  ```

- Copy a SPICE netlist of the pattern to match in this directory
- Still in this directory, create the file LIBRARY. This file intends to tell the tool the files in which it will find the patterns. The LIBRARY name can be changed to any name, assuming it is specified in the main script with the `gnsLibraryName` variable.
- Add the following line in the main script to invoke the GNS pattern matching engine:

  ```
  avt_config yagleUseGenius yes
  ```

If HITAS is invoked with the configuration set up to now, it will integrate the timing description (DTX file) of the "analog holes" corresponding to the removed transistors matching the given pattern.

12.2.5. Netlists Consistency

The pre-characterization process is the same in the case of a flat or hierarchical top-level netlist. It just requires the netlist of the analog sub-circuit to simulate, in order to obtain the appropriate values for the timing arcs.

When dealing with a hierarchical netlist, the user should provide a netlist of the analog sub-circuit coming from the same source as the top-level (schematic editor or hierarchical extractor). Thus both netlists contain (or not) parasitics, and the analysis is consistent.

When dealing with a flat netlist, it is most of the times in a post-layout approach. Actually, the only reason for using a flat netlist is that it is sometimes the only output parasitic extractors provide. In such a case, the top-level netlist contains parasitics, and one must take care that the netlist used for simulation of the analog sub-circuit also contains parasitics.

12.3. On-the-Fly Characterization

The on-the-fly characterization follows the same principles as the pre-characterization, i.e.:

- Link with an analog simulator

- Create a timing database of the analog subcircuit
- Integrate the database in a hierarchical or flat top-level netlist

The difference is that the steps one and two are performed for each instance of the analog subcircuit, taking into account instantiation-specific parameters (output load). It then provides greater accuracy but implies a different implementation.

On-the-fly characterization is based upon the GNS pattern-matching engine. The methodology is to associate an "action" with a matching event i.e., each time the tool identifies a pattern in the top-level netlist, it executes the corresponding action. Here the corresponding action will be the steps one and two described above (link with an analog simulator and create a timing database of the analog sub-circuit).

The difference of implementation lies in the description language used to write the action. As the pre-characterization method uses the Tcl language, the writing of the "action" for on-the-fly characterization uses the C language. By the way, the prototypes of the functions used in both cases are identical.

An action is then typically a C file. The association of an "action" with a pattern takes place in the LIBRARY file.

12.3.1. Database Construction

Apart from the syntax, the C file used for simulator link and database creation is very close to the Tcl script used for pre-characterization.

12.3.2. Hierarchical Netlist Integration (Pre-Layout)

The integration of on-the-fly characterization in a hierarchical netlist uses a combination of the blackboxing and pattern-matching mechanisms.

- The blackboxing mechanism should be used in the same way as in the pre-characterization approach. It creates the "analog holes" in the top-level netlist.
- The pattern matching mechanism is used to execute the action described above, each time the tool encounters a pattern matching the "analog hole". As the analog sub-circuit has been removed from the netlist, no information about the transistor structure of the analog sub-circuit remains in the top-level netlist. Therefore, the pattern the tool will try to match with the "analog hole" is just the interface of the provided pattern. The user can provide as a pattern, either a sub-circuit containing transistors, parasitics and so on, or an empty sub-circuit.

The remaining configuration is the same as in the approach of integration of a pre-characterization in a flat netlist.

12.3.3. Flat Netlist Integration (Post-Layout)

The integration of on-the-fly characterization in a flat netlist also uses a combination of the blackboxing and pattern-matching mechanisms. Here, the "analog holes" are created by the pattern matching mechanism itself, as in the integration of a pre-characterization in a flat netlist approach. The blackboxing configuration (`tasIgnoreBlackbox`) should not be used. However, it will have no effect on the netlist itself (as there is no instance to blackbox), but it may have a side effect on the sub-circuit provided for the pattern matching engine (if the name of the latest appears in the `avt_SetBlackBoxes` function, it may be unintentionally blackboxed, making the pattern matching process to fail).

The main advantage of on-the-fly characterization is to provide a specific timing database for each instance of an analog sub-circuit, depending on instantiation context (e.g. output load). Another advantage of the on-the-fly approach is that the analog simulations are performed on the transistors of the sub-circuit matching the pattern provided. Therefore each instance matching the pattern has its own parasitic information.

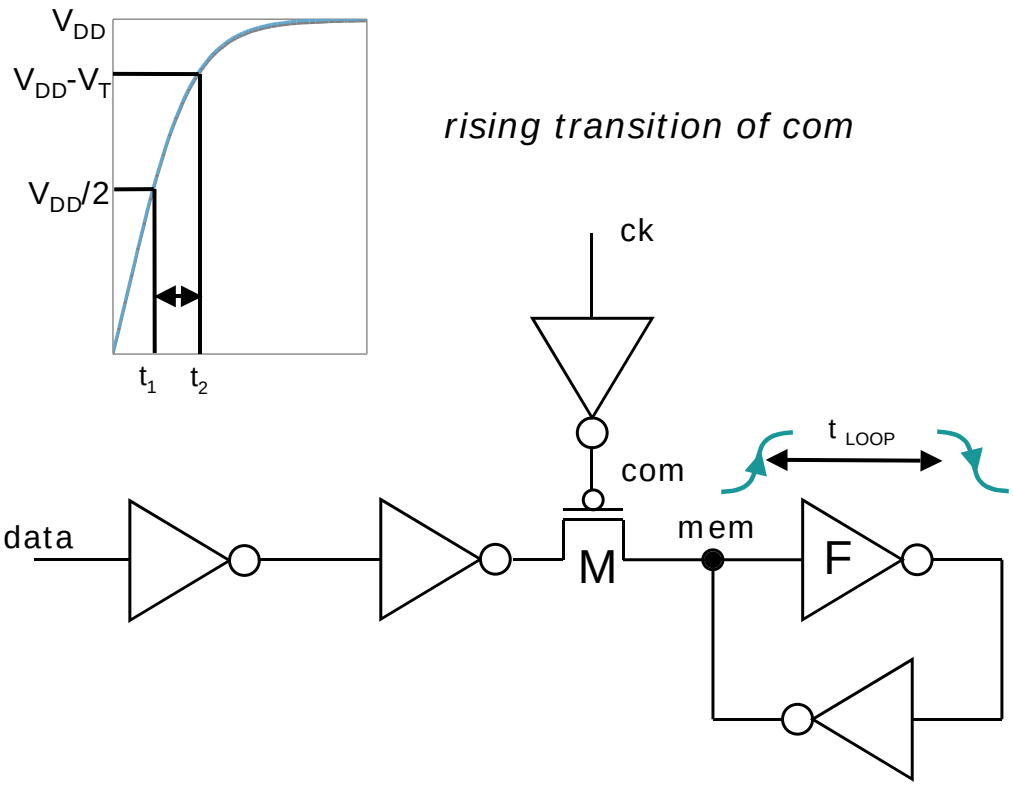
Chapter 13. Timing Characterization (.lib)

13.1. Setup and Hold Constraints Formulas

Interface Setup and hold constraints are computed for each latch and register. They are computed with respect to the following formulas:

```
setup = data_path_max - clock_path_min
hold = clock_path_max - data_path_min
```

The following diagram gives an example of clock path and data path.



- Clock path: from `ck` to `com` (the command of the latch)
- Data path: from `data` to `mem` (the memory point itself).

With such definitions of data and clock paths, the above formulas give optimistic values for setup and hold times. Therefore, a corrective factor is added to those values. The corrective factor for setup is called the "intrinsic setup of the latch"; the corrective factor for hold is called "intrinsic hold of the latch". The formulas now become:

```
setup = data_path_max - clock_path_min + intrinsic_setup
hold = clock_path_max - data_path_min + intrinsic_hold
```

13.1.1. Setup Correction

For setup time, the calculus is as follow:

- `clock_path_min` delay is computed at `com` crossing Vdd/2 (time `t1`), as the transitor `M` closes at time `t2`: `t2 - t1` must be added to `clock_path_min`
- Setup time must ensure that data is correctly written into the latch, i.e. that data crosses the feedback loop. This is modeled by adding `t_loop` to `data_path_max`.

The formula for setup becomes:

```
setup = data_path_max + t_loop - (clock_path_min + t2 - t1)
```

The corrective factor for setup is:

```
intrinsic_setup = t_loop - (t2 - t1)
```

13.1.2. Hold Correction

For hold time, the calculus is as follow:

- `clock_path_max` delay is computed at `com` crossing Vdd/2 (time `t1`), as the transitor `M` closes at time `t2`: `t2 - t1` must be added to `clock_path_max`

The formula for hold becomes:

```
hold = clock_path_max + (t2 - t1) - data_path_min
```

The corrective factor for hold is:

```
intrinsic_hold = t2 - t1
```

13.2. Performing the Characterization

The purpose of timing abstraction is to create a `.lib` file - containing setup, hold and access information of the design - from an already existing timing figure. Within the `avt_shell` Tcl interface, timing abstraction is performed with the function `tmabs`

`BehFigure` is a description of the functionality that can be associated with the design in the `.lib` file. For the moment it takes the `NULL` value.

`TimingFigure` is the database itself, the one the `.lib` file will be created from. Timing abstraction only uses the `.dtx` file. The database can be obtained by two ways. Through the `hitas` function, with the appropriate configuration allowing correct database construction:

```
avt_config ...
avt_config ...

set fig [hitas my_design]
```

If the timing database has already been constructed, the path view can be obtained from the `.dtx` file with the following command:

```
set fig [ttv_LoadSpecifiedTimingFigure my_design]
```

Timing abstraction requires additionnal information concerning clock definition (in order to construct correct setup/hold/access relationships). Clock definition and timing abstraction are then done as follow:

```
inf_SetFigureName my_design
create_clock -period 3000 ck
set abs_fig [tmabs $fig NULL * * *]
```

The `-period` value is irrelevant but is needed to respect SDC syntax. The `.lib` file is generated from the abstracted timing figure `abs_fig` as follow:

```
lib_DriveFile [list $abs_fig] NULL my_design.lib max
```

13.3. Advanced Configuration

13.3.1. Input Slope and Output Load Axis

User-defined input slopes can be defined with the function `inf_DefineSlopeRange`. This function affects the way lookup-table axis are constructed. Be aware that `inf_DefineSlopeRange` should be applied before calling the `hitas` function:

```
inf_SetFigureName my_design
inf_DefineSlopeRange default {100e-12 150e-12 350e-12} custom
set fig [hitas my_design]
```

The same remarks apply to `inf_DefineCapacitanceRange`.

13.3.2. Capacitances in the .lib file

By default, capacitance values are given for input connectors only, as an average value. The given value is the equivalent capacitance allowing to compute the driving gate's delay at vdd/2. Capacitance ranges as well as different rise/fall capacitances can be obtained by tuning the `elpCapaLevel` variable (values `1` or `2`).

Capacitances can also be given for output connectors (set `tmaDriveCapaOut` variable to `yes`). In such a case, the output delay is given WITHOUT taking into account the output connector's capacitance.

13.4. Cell Library

Here is given an example Tcl script performing the timing abstraction of a list of standard cells, into a single `.lib` file:

```
avt_config tasBefig yes
avt_config tmaFunctionalityMode w
```

```
avt_LoadFile ./bsim3_018.tech spice

foreach cell { ao2o22 ff2 inv mux2 na2 } {
    avt_LoadFile $cell.spi spice

    set fig [hitas $cell]
    set beh_fig NULL
    set abs_fig [tma_abstract $fig $beh_fig]

    lappend fig_list $abs_fig
    lappend beh_list $beh_fig
}

lib_drivefile $fig_list $beh_list "stdcells.lib" max
```

Chapter 14. Using the GUI

14.1. Timing Database Browsing with XTAS

14.1.1. Overview

Execution of the HITAS results in the generation of a flat or hierarchical timing database (UTD) describing, in a compressed form, all the timing paths, cone details and interconnect details in the design under analysis. Xtas should be used in order to visualize the information in this timing database. It provides a fast, interactive and user friendly environment in which to perform timing requests. It also provides the means of performing the full static timing analysis and the visualization of accurate timing diagrams for any signals which do not satisfy the constraints.

14.1.2. Description

The XTas tool is used after generation of the UTD. It is used in order to interpret the information in the DTX file. The timing database must first be loaded into XTas. The DTX file contains gate details. In the case of a hierarchical database, it is the top-level DTX file which should be specified.

After loading the database, reference points (connectors, latches, commands, and precharges) can easily be visualized. You can then ask XTas to display critical and near-critical paths based on constraints such as, for example, the display of the ten longest paths ending on a particular point. For any given critical path, it is possible to display instantaneously paths with the same reference points but not the most critical; these are the so-called parallel or near critical paths. Optimization of a critical path is often useless if parallel paths are not similarly optimized.

For any critical or near-critical path, XTas can also instantaneously display the full gate and interconnection details of the path. Unlike most timing analyzers, this is possible even if the path traverses several levels of the hierarchy.

Finally, it is also possible to launch the static timing analysis engine from within XTas. In order to perform this analysis, it is first necessary to create a constraints file (see chapter "Static Timing Analisys" of this Guide). If this file exists alongside the timing database, then a single button launches the analysis after specifying a few options. Any violations are displayed in a new window from which the timing diagrams can be obtained.

14.1.3. Execution

XTAS Command

XTAS is started as follow:

XTAS
Upon entering this command the XTAS main window appears.

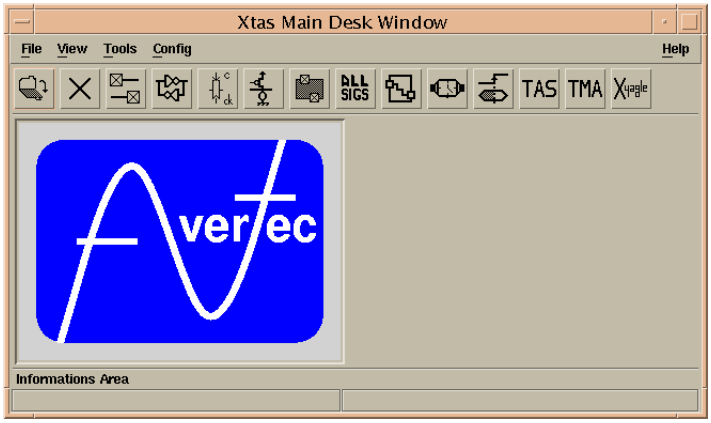
XTAS Resource File

The file AvtTools/etc/Xtas allows modification of certain characteristics of the graphical interface. For example, the color and the size of the windows of xtas can be modified in this file.

XTAS Splash Screen

The XTAS main desk window is the starting point for using XTAS. You can choose to use the menus or the buttons. Buttons are shortcuts giving the same results as a selection in the menus.

The Main window looks like:

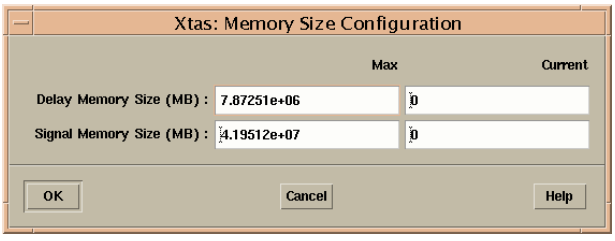


14.2. Configuration

The "Config" menu in XTAS provides access to a number of dialogs containing configuration options affecting the performance and appearance of the graphical database browser. These options are detailed in the following sections.

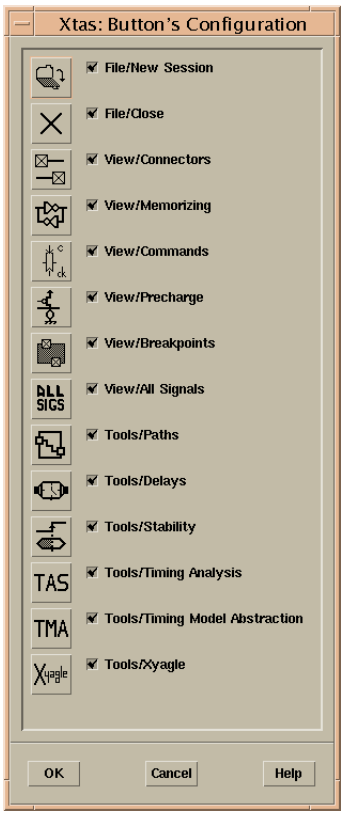
14.2.1. Memory Size

The memory size dialog allows the user to set the maximum and the current amount of memory used as the cache for loading delay and signal details. This dialog looks as follows:



14.2.2. Toolbar Buttons

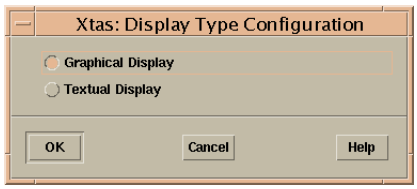
This dialog box allows the user to add or remove Xtas' tool bar buttons. If a button is not checked, it does not appear in the Xtas' tool bar. The dialog also indicates which action corresponds to the button and in which menu this action is also available. In addition, the user can customize the button's bar by checking or not a button. This dialog appears as follows:



A similar dialog is available in the menu "Option" of other windows.

14.2.3. Display Type

This dialog allows the user to choose between a graphical or textual display for the path or delay windows. The graphical display uses static text boxes and icons whereas the textual one displays information in plain text. The textual display is recommended for long signal names and long lists as it has faster display. This dialog appears as follows:



The textual display can be forced as default using the `xtasTextualDisplay` configuration variable.

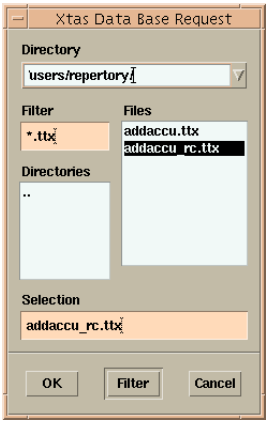
14.3. Loading the Timing Database

14.3.1. Timing Database

After opening the XTAS main window you need to load a DTX format timing database. Choose open in the file menu of the XTAS main window.

This button acts like a shortcut for the opening of the database.

you will then be presented with the XTAS database request dialog. Choose which timing database to load by selecting it in this dialog.

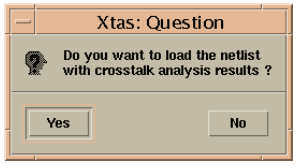


The default filter can be changed using the `xtasDatabaseFilter` configuration variable.

14.3.2. Crosstalk Timing Database

It is also possible to load a CTX format timing database containing information given by crosstalk analysis (See chapter 'Output Files' of the HITAS Reference Guide for further information on this format).

To do this, you must select the timing database in DTX format in the XTAS database request dialog (change Filter to "*.dtx" to have access to the list of DTX files). Then, if a corresponding CTX file exists, the following dialog will appear asking confirmation to load it:



14.4. Accessing XTAS Features

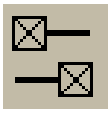
14.4.1. Exiting XTAS

Choose exit in the File menu in order to close XTAS.

Clicking this button in the main window will exit XTAS. In any other window, it will close the current window.

14.4.2. Browsing Connectors

Select Connectors in the View menu of the XTAS main window in order to open the XTAS connectors window



This button opens the XTAS connectors window.

The XTAS connector window allows viewing of the list of the connectors of the Instance. This list can be filtered according to name and/or hierarchical level.

14.4.3. Browsing Registers

Select Registers in the View menu of the XTAS main window in order to open the XTAS registers window

This button opens the XTAS registers window.

The XTAS registers window allows viewing of the list of the registers of the Instance. This list can be filtered according to name and/or hierarchical level.

14.4.4. Browsing Commands

Select Commands in the View menu of the XTAS main window in order to open the XTAS commands window

This button opens the XTAS commands window.

The XTAS commands window allows viewing of the list of the commands of the Instance. This list can be filtered according to name and/or hierarchical level.

14.4.5. Browsing Precharges

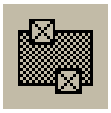
Select Precharges in the View menu of the XTAS main window in order to open the XTAS precharges window

This button opens the XTAS precharges window.

The XTAS precharges window allows viewing of the list of the precharges of the Instance. This list can be filtered according to name and/or hierarchical level.

14.4.6. Browsing Break Points

Select Break Points in the View menu of the XTAS main window in order to open the XTAS break points window



This button opens the XTAS break points window.

The XTAS break points window allows viewing of the list of the break points of the Instance. This list can be filtered according to name and/or hierarchical level.

14.4.7. Browsing Internal Signals

Select Internal Signals in the View menu of the XTAS main window in order to open the XTAS all signals window

This button opens the XTAS all signals window.

The XTAS all signals window allows viewing of the list of all internal signals of the Instance. This list can be filtered according to name and/or hierarchical level.

14.4.8. Browsing Paths

Select Get Paths in the Tools menu of the XTAS main window in order to open the XTAS get paths window

This button opens the XTAS get paths window.

The XTAS get paths window allows the setting of numerous parameters. From these parameters XTAS will display the selected critical paths.

14.4.9. Browsing Delays

Select Get Delays in the Tools menu of the XTAS main window in order to open the XTAS get delay window

This button opens the XTAS get delay window.

The XTAS get delay window allows the setting of numerous parameters. From these parameters XTAS will display the selected max delays.

14.4.10. Stability Analysis

Select Stability in the Tools menu of the XTAS main window in order to open the XTAS stability parameterization window

This button opens the XTAS stability parameterization window.

The stability parameterization window allows you to specify the stability and/or crosstalk analysis options and then performs the analysis.

14.4.11. Common applications

From any window in XTAS it is possible to identify the use and associated menu of any of the Xtas' tool bar buttons. To do this, open the XTAS Button Configuration window from the Options menu.

The following functions are available from most of the window types:

Clicking this button will make the XTAS main window appear at the foreground.

This button closes the current window.

This button opens the XTAS Signals Info window.

This button opens the XTAS Crosstalk Info window.

If the INF file has been modified during a XTAS session (for example for false path information), it is possible to reload it from the File menu.

14.5. Browsing Timing Signals

This section describes timing signals browsing features. Example is given below for memory elements. Browsing of connectors, commands, precharges, break points and internal signals is done the same way.

Open the XTAS Registers window in order to display the memory elements.

Clicking this button opens the XTAS registers window.

The following options are available:

Signal Mask — You can give the name of a particular register in order to display this register. The '*' is a wildcard you can use to select a set of signals whose names will fit the conditions you have given. The default value is '*' and will display all the registers.

Figure name — This is the name of the database you have load in XTAS

Instance name — This is the name of the instance whose registers you are looking for.

List — This option opens the Hierarchy window. It displays the hierarchy of the figure, and allows you to select another instance. It is possible to make the search through all levels. An On-Line help is available for this option in the hierarchy window.

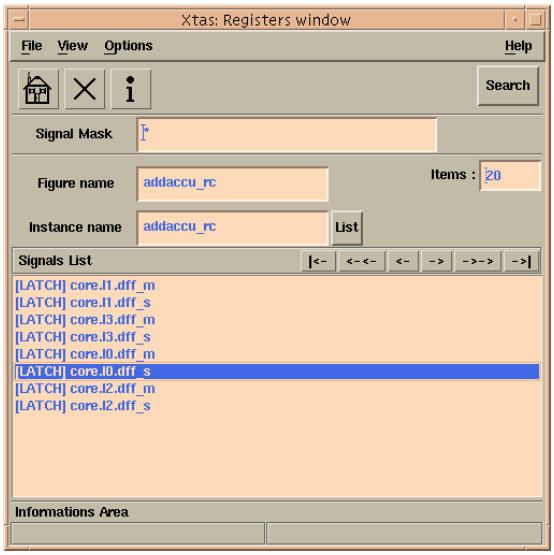
Items — You can set the number of registers you want to display in the Information Area. The resulting list will be given either in a single page or on several pages depending if the number of registers is greater than the value given.

Search — Displays the registers in the Information Area.

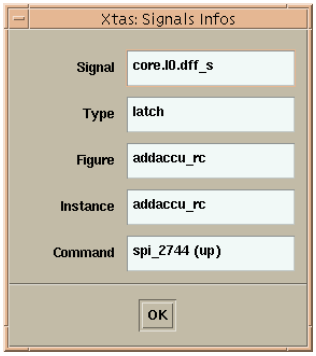
Signals List — If the number of register is greater than the figure for Item XTAS will give the registers list on several pages. The signals list allows you to access all the signals. If the number of register is inferior to the value given, there is only one page.

After Search, this option shows information on a selected signal.

The resulting XTAS Registers window of a Search looks like:



Double clicking on a register displays more information about that register in the XTAS Signals Info Window.



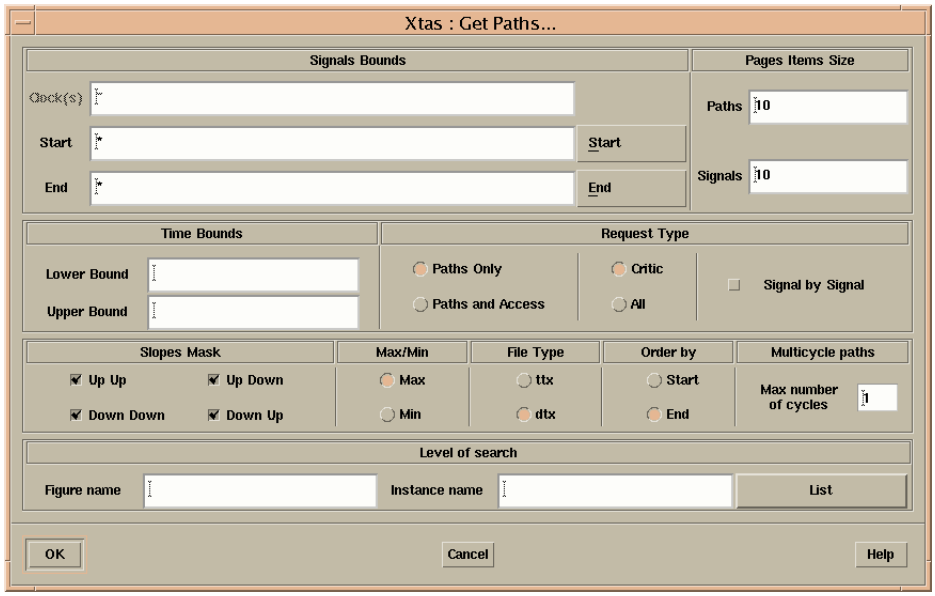
14.6. Browsing Critical Paths

14.6.1. General Procedure

Open the XTAS Get Paths window in order to select options for displaying critical paths.

Clicking this button opens the XTAS Get Paths window.

The XTAS Get Paths window looks like:



This window is the first step to viewing the critical paths of the instance. Use the options to set criteria for displaying critical paths.

14.6.2. Options

You can use the following options in the XTAS Get Paths window:

Start Text Box

This text box describes the regexp (including *) to be matched for the beginning signal of the path. The Start button permits browsing of path extremity signals in the loaded figure.

End Text Box

This text box describes the regexp (including *) to be matched for the terminating signal of the path. The End button permits browsing of path extremity signals in the loaded figure.

Clock(s) Text Box

This text box describes the regexp (including *) to be matched for the clock signal when viewing access.

Time Bounds

These text boxes indicate the time bounds between which paths must be found.

Request Type

Choose the search criteria. When choosing to view "Paths and Access" the "Clock(s)" text box becomes available.

Slopes Mask

Select here the desired types of transitions between the start signal and the end signal.

Max/Min

The Max button performs the search for the longest paths. The Min button performs the search for the shortest paths.

Order by

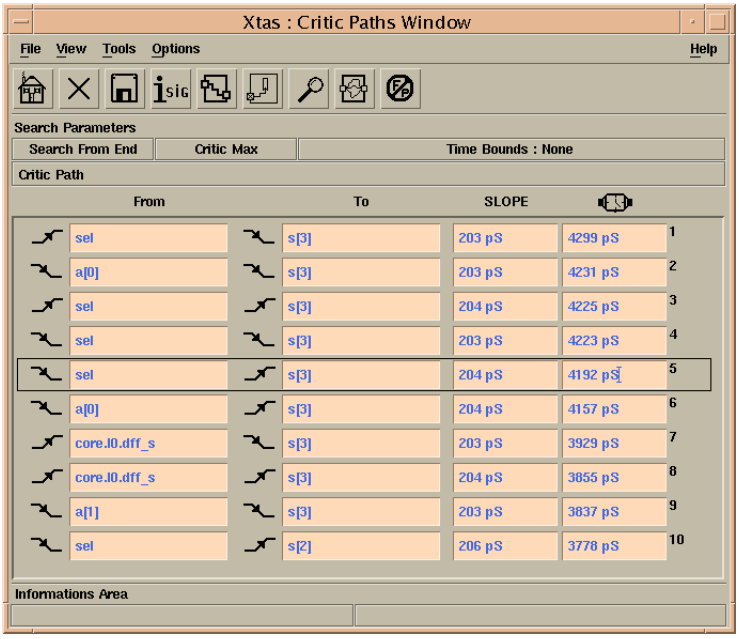
These buttons select the source signal of the paths search. In order to reduce computing time, the signal (start or end) with the most restrictive mask must be chosen.

Search Level

These text boxes allow limiting of the path search to certain levels of hierarchy. The List button allows selecting of instances. By default, search is performed in all levels of hierarchy.

14.6.3. Critical Paths Display

XTAS display the XTAS Critic Paths Window:



Each line in the window corresponds to a critical path. The name and transition of the path terminals are given, together with the path delay and output slope.

Select a register signal in the list to get all the paths that end on the command of this register by clicking on this button. It will open the XTAS Get Paths window with the End Text Box filled with the name of all the commands of that register.

14.7. Browsing Near Critical Paths

14.7.1. Overview

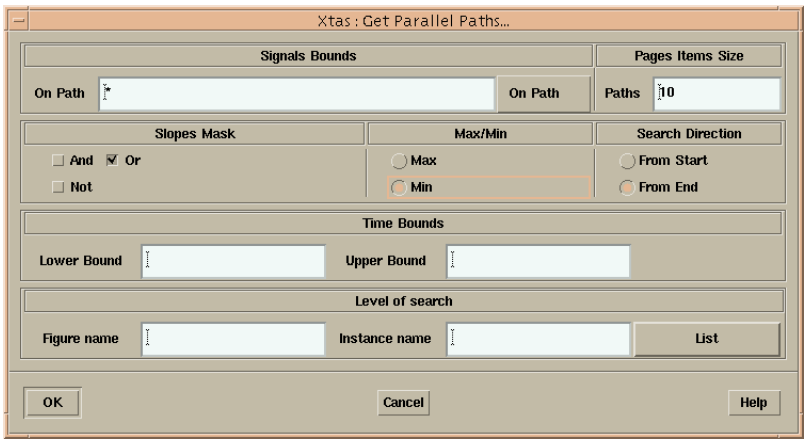
Near critical paths (or parallel paths) are paths with the same terminals as a critical path. Optimization of a critical path is often useless if parallel paths are not similarly optimized.

14.7.2. Options

In the XTAS Critic Paths window open the XTAS Get Parallel Paths in order to view the near critical paths.

Clicking this button opens the XTAS Get Parallel Paths window.

The XTAS Get Parallel Paths window looks like:



This window is the first step to viewing the near critical paths.

The following options are available:

Signals Bounds — The On Path text box describes the regexp (including *) to be matched for intermediary signals of the parallel path. The On Path button permits browsing of signals in the loaded figure.

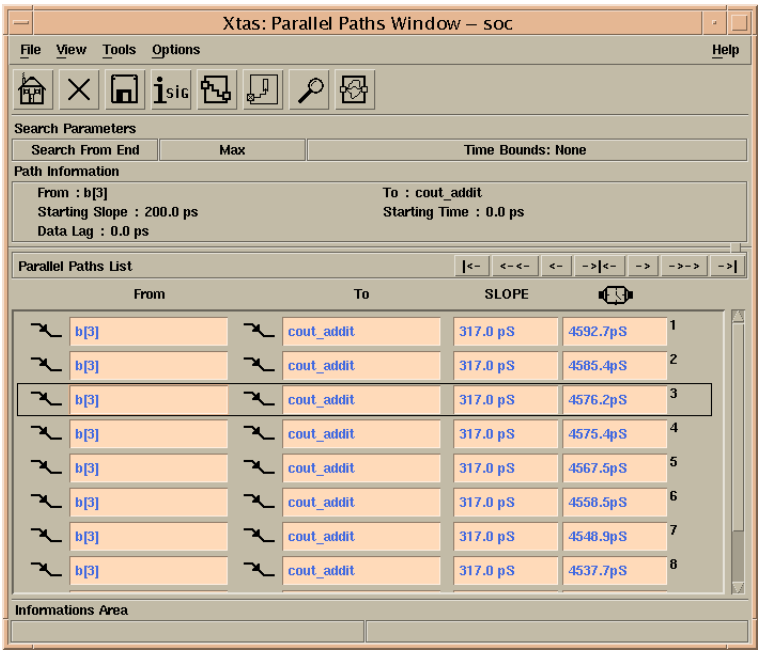
Slopes Mask — Select here the desired type of selection of the intermediary signal. Available types of selection are:

- And: All signals must appear on parallel paths.
- Or: At least one signal must appear parallel paths.
- Not: the signals must not appear on parallel paths.

Max/Min	The Max button performs the search for the longest paths. The Min button performs the search for the shortest paths.
Order by	These buttons select the source signal of the paths search. In order to reduce computing time, the signal (start or end) with the most restrictive mask must be chosen.
Time Bounds	These text boxes indicate the time bounds between which paths must be found. Only critical paths or all paths can be searched, by selecting or not the button Critical Paths.
Search Level	These text boxes allow limiting of the path search to certain levels of hierarchy. The List button allows selecting instances. By default, search is performed in all levels of hierarchy.

14.7.3. Near Critical Paths Display

XTAS display the XTAS Parallel Paths Window:



14.8. Browsing Path Details

14.8.1. Overview

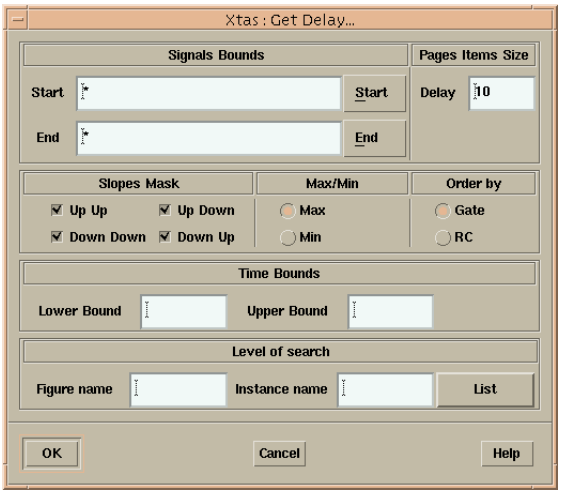
If a DTX file has been generated, it is possible to display elementary gate or RC delays according to certain criteria.

14.8.2. Options

In the XTAS Main window open the XTAS Get Delay window in order to view elementary delays.

Clicking this button opens the XTAS Get delay window.

The XTAS Get Delay window looks like:



The following options are available:

Start Text Box — This text box describes the regexp (including *) to be matched for the beginning signal of the path. The Start button permits browsing of path extremity signals in the loaded figure.

End Text Box — This text box describes the regexp (including *) to be matched for the terminating signal of the path. The End button permits browsing of path extremity signals in the loaded figure.

Slopes Mask — Select here the desired types of transitions between the start signal and the end signal.

Max/Min — The Max button performs the search of the maximum delays. The Min button performs the search of the minimum delays.

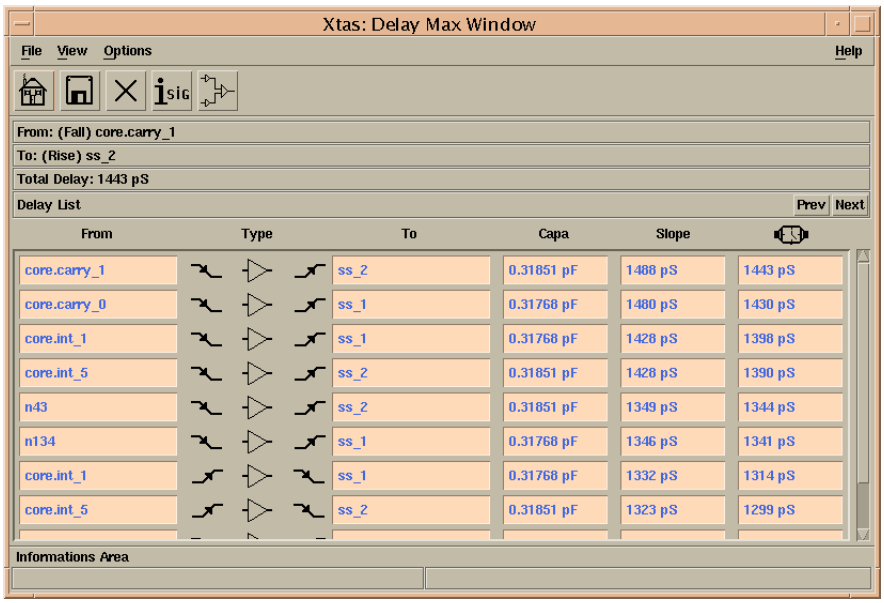
Order by — Those buttons select the type of delay to be searched. RC means interconnect delays, Gate means Gate delays.

Time Bounds — Those text boxes indicate the time bounds between which paths must be found. Only critical paths or all paths can be searched, by selecting or not the button Critical Paths.

Search Level — Those text boxes allow the search of paths in but chosen levels of hierarchy. The List button permits to select instances. By default, search is performed in all levels of hierarchy.

14.8.3. Delay Display

Xtas display the Xtas Delay Max Window:



Each line in the window corresponds to an elementary delay or timing arc. The input and output transitions of the timing arc are given, as well as the delay and the output slope. The type icon identifies the delay as corresponding to a gate or RC interconnection.

14.9. CPE Path Simulation

14.9.1. Overview

When designers use an electrical simulator such as ELDO or HSPICE to obtain path timing information, the path must be extracted manually and appropriate stimuli provided. XTAS can automate this procedure for any identified path.

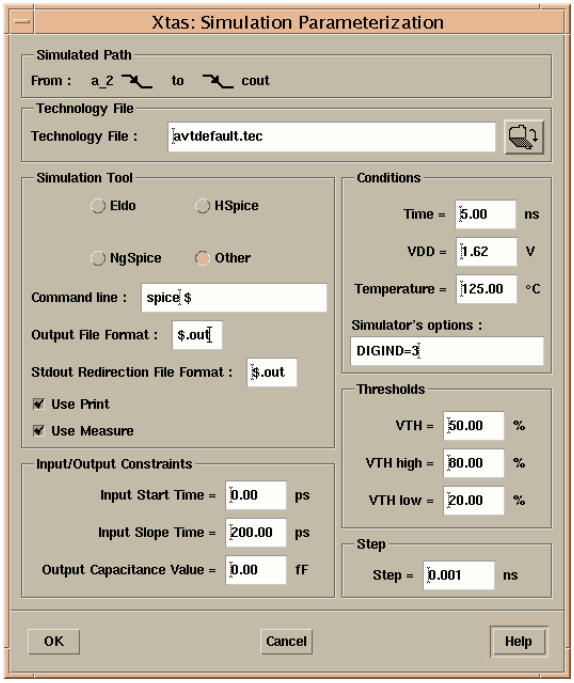
From the XTAS Path Detail window, it is possible to run a simulation with the electrical simulator of your choice. You need only to configure the simulation parameters. Path extraction and stimuli generation are automatic.

In a new XTAS Path Detail window, you can compare results given by HITAS with these given by the simulator.

14.9.2. Options

Clicking this button opens the XTAS Simulation Parameterization window.

The XTAS Simulation Parameterization window looks like:



This window is the first step to simulate a path.

The following options are available:

Technology File — Click on the Open button to choose the directory of the technology file and select it in the list. This is equivalent to the `avtTechnologyName` configuration variable.

Simulation Tool — Allows the user to choose the electrical simulator he wants to use and the corresponding command line. He can also specify the output format generated by the simulator and the file format for stdout redirection. It is equivalent to the `simToolModel`, the `avtSpiceString`, the `avtSpiceOutFile` and the `avtSpiceStdoutFile` configuration variables respectively. Checking the "Use Print" box allows getting an array of value for a node, it is equivalent to setting `simUsePrint` configuration variable to "yes". Checking the "Use Measure" box specify to extract

value from waveform using combination of arithmetic expressions, it is equivalent to setting `simUseMeasure` configuration variable to "yes".

Constraints — Allows the user to set the input/output constraints parameters for simulation. Specify the input slope start time and the transient time of the input slope in picoseconds. Specify the output capacitance value in Femto-farads. It is equivalent to the `simInputStartTime` and the `simSlope` and the `simOutCapaValue` configuration variables.

Conditions — Allows the user to set the simulation conditions like the duration (in nanoseconds), the maximum voltage (in Volts) and the temperature (in degrees Celsius). It is equivalent to the `simTransientTime`, the `simPowerSupply` and the `simTemperature` configuration variables respectively. The user can also specify the spice options to be driven into the spice file for simulation it is equivalent to the `simSpiceOptions` configuration variable.

Thresholds — Allows the user to set the thresholds of a slope as a percentage of VDD. It is equivalent to the `simVth`, the `simVthHigh` and the `simVthLow` configuration variables.

Step — Allows the user to set the transient calculation step for the simulation in nanoseconds. Equivalent to the `simTransientStep` configuration variable.

14.9.3. Simulation Path Display

Xtas displays the Xtas Simulated Critic Max Path Detail Window:

For all timing arc data (Slope, Delay and Total) there are two columns:

These columns give the values obtained with the static timing analysis.

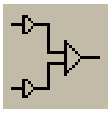
These columns give the values obtained with the selected simulator.

14.10. Path Visualization

14.10.1. Overview

It is possible to view a graphical representation of a path.

14.10.2. Options



Clicking this button opens the XTAS Path Visualization window.

The XTAS Path Visualization window looks like:

This window is the first step to visualize a path.

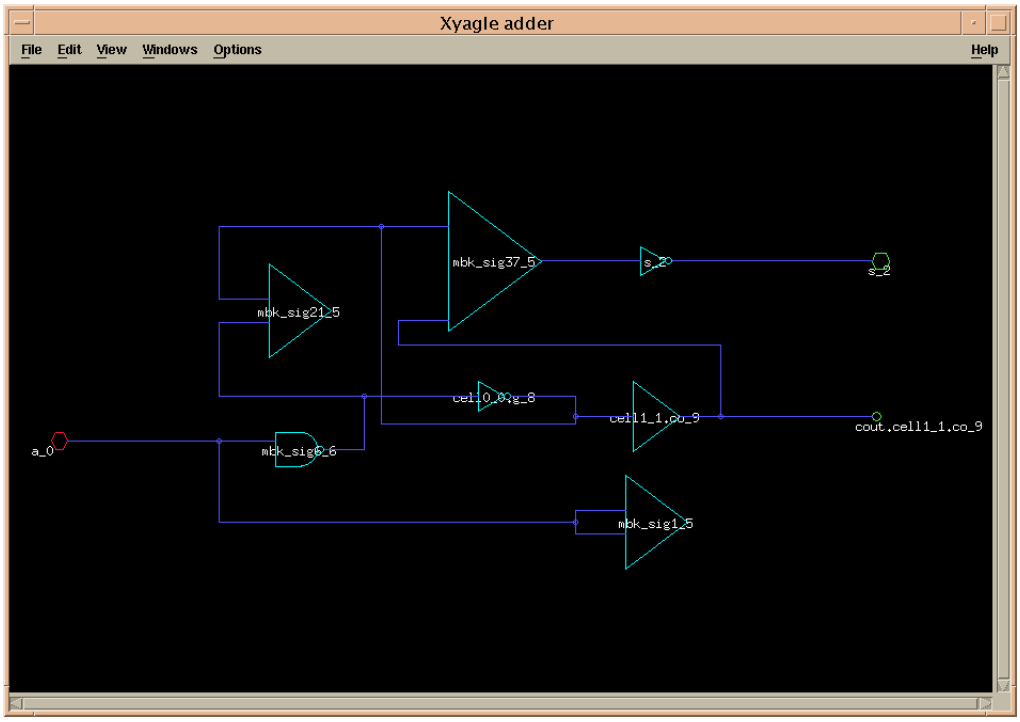
There are two modes of representation:

Highlight displays all the circuit and highlights the selected path.

Extract displays only the selected path.

14.10.3. Path Visualization Display

Xtas displays the following window:



14.11. Browsing Delays

14.11.1. Overview

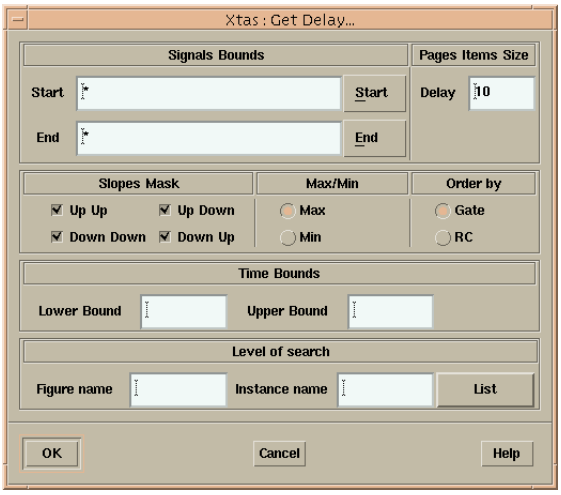
If a DTX file has been generated, it is possible to display elementary gate or RC delays according to certain criteria.

14.11.2. Options

In the XTAS Main window open the XTAS Get Delay window in order to view elementary delays.

Clicking this button opens the XTAS Get delay window.

The XTAS Get Delay window looks like:



The following options are available:

Start Text Box — This text box describes the pattern (including *) to be matched for the beginning signal of the path. The Start button permits browsing of path extremity signals in the loaded figure.

End Text Box — This text box describes the pattern (including *) to be matched for the terminating signal of the path. The End button permits browsing of path extremity signals in the loaded figure.

Slopes Mask — Select here the desired types of transitions between the start signal and the end signal.

Max/Min — The Max button performs the search of the maximum delays. The Min button performs the search of the minimum delays.

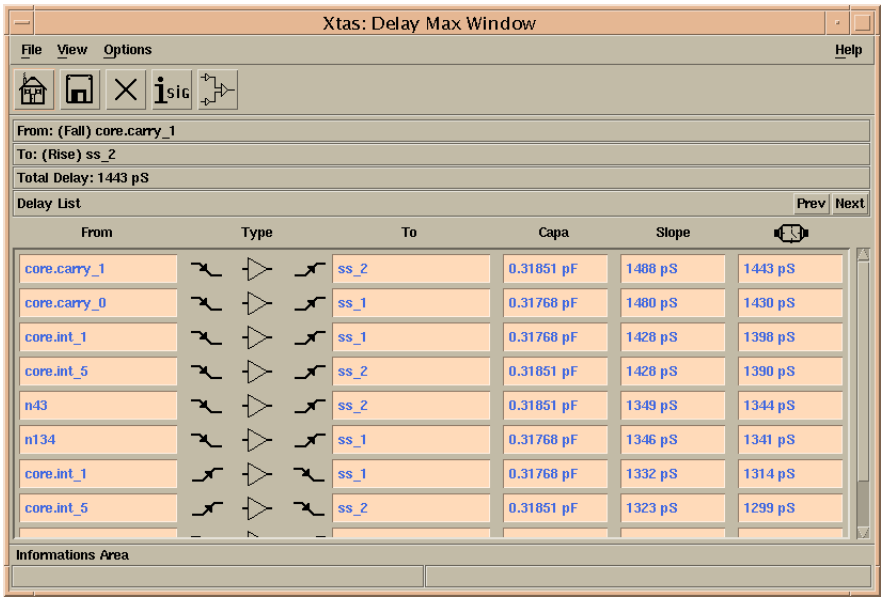
Order by — Those buttons select the type of delay to be searched. RC means interconnect delays, Gate means Gate delays.

Time Bounds — Those text boxes indicate the time bounds between which paths must be found. Only critical paths or all paths can be searched, by selecting or not the button Critical Paths.

Search Level — Those text boxes allow the search of paths in but chosen levels of hierarchy. The List button permits to select instances. By default, search is performed in all levels of hierarchy.

14.11.3. Delay Display

Xtas display the Xtas Delay Max Window:



Each line in the window corresponds to an elementary delay or timing arc. The input and output transitions of the timing arc are given, as well as the delay and the output slope. The type icon identifies the delay as corresponding to a gate or RC interconnection.

14.12. Static Timing Analysis and Signal Integrity

14.12.1. Overview

It is possible to launch the static timing analyzer with or without the crosstalk analysis from within XTas. In order to perform this analysis, it is first necessary to create a constraints file (.inf file). If this file exists alongside the timing database, then a single button launches the analysis after specifying a few options. Any violations are displayed in a new window from which the timing diagrams can be obtained and the noise analysis can be launched.

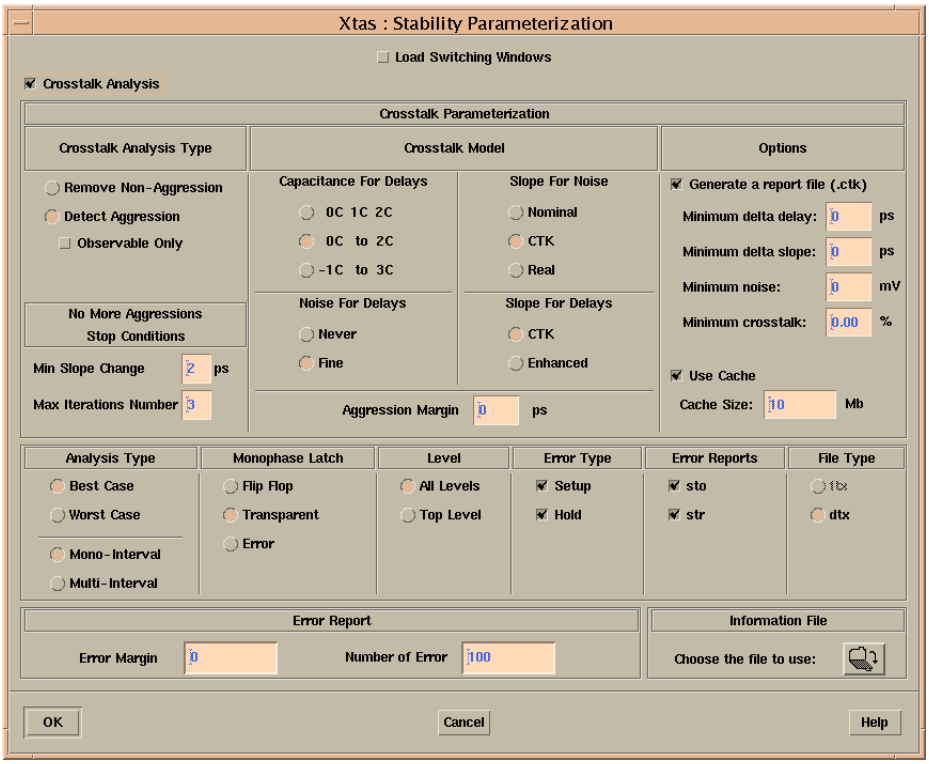
14.12.2. Static Timing Analysis Results

Launching the Analysis

In the XTAS Main window open the XTAS in order to begin the static timing analysis.

Clicking this button opens the XTAS Stability Parameterization window.

The XTAS Stability Parameterization window looks like:



This window is the first step of the static timing analysis.

Loading Switching Windows

If static timing analysis has been performed on the current figure, the user can directly load the results from the .sto file.

Crosstalk Analysis Parameterization

The user can choose to perform the static timing analysis with full handling of crosstalk effects. If this box is checked, the user has access to the crosstalk analysis parameterization area. See the HITAS Reference Manual for more detailed explanations of the configuration options.

Crosstalk Analysis Type

In the "Remove Non-Aggression" mode, all aggression is assumed initially. In the "Detect Aggression" mode, no aggression is assumed initially. In addition to that mode, you can check the box "Observable Only" in order to have less pessimistic results.

No More Aggressions Stop Conditions

These are conditions for stopping slope recalculation when no further aggression is detected or removed: "Min Slope Change" represents the minimum significant slope variation in picoseconds (equivalent to the `stbCtkMinSlopeChange` configuration variable) and "Max Iteration Number" represents the maximum number of recalculation iterations (equivalent to the `stbCtkMaxLastIter` configuration variable).

Crosstalk Model

Capacitance For Delays

Is equivalent to setting the `rcxCtkModel` configuration variable: "0C 1C 2C" corresponds to the MILLER_0C2C value, "OC to 2C" corresponds to the MILLER_NOMINAL value and "-1C to 3C" corresponds to the MILLER_NC3C value.

Noise For Delays

Is equivalent to setting the `rcxCtkNoise` configuration variable.

Aggression Margin

Is equivalent to setting the `stbCtkMargin` configuration variable.

Slope For Noise

Is equivalent to setting the `rcxCtkSlopeNoise` configuration variable.

Slope For Delays

Is equivalent to setting the `rcxCtkSlopeDelay` configuration variable.

Options

The crosstalk analysis can generate a report file (.ctk), which contains delay changes, detailed aggression reports and noise estimation. You can set lower limits to avoid reporting excessive information:

Minimum delta delay

Equivalent to the `ctkDeltaDelayMin` configuration variable.

Minimum delta slope

Equivalent to the `ctkDeltaSlopeMin` configuration variable.

Minimum noise

Equivalent to the `ctkNoiseMin` configuration variable.

Minimum crosstalk

Equivalent to the `ctkCapaMin` configuration variable.

In order to manage the memory more efficiently, you can choose to use a cache and give its size in Megabytes. The bigger the cache, the faster the analysis. It is equivalent to setting the `avtParasiticCacheSize` configuration variable.

Stability Parameterization

Analysis type

The best case analysis is performed by assuming that in the initial conditions, latch transparency is at maximum. The worst case analysis is performed by assuming that in the initial conditions, there is no latch transparency. In Multi-Interval mode, all switching windows are maintained

whereas in Mono-Interval mode, they are merged into a single interval for setup/hold verification. To perform the crosstalk analysis, the Mono-Interval mode is required. See HITAS Reference Guide for further information.

Monophase Latch

In the Flip-Flop mode, a latch clocked on the same phase than the latch generating its input data is assumed to be a flip-flop. In the Transparent mode, a latch clocked on the same phase than the latch generating its input data is always transparent. In the Error mode, a latch clocked on the same phase than the latch generating is input data is not allowed, and an error is reported. See HITAS Reference Guide for further information.

Level

In the All Levels mode, constraints are calculated using all paths throughout the hierarchy. In the Top Level mode, constraints are calculated using only the paths at the top level (i.e. the interconnections at the top level).

Error Type

In the Setup mode, only errors due to setup time violations are reported. In the Hold mode, only errors due to hold time violations are reported

Error Reports

In sto mode, a .sto file is driven. This file contains the switching windows of all the signals in the figure. In str mode, a .str file is driven. This contains the signals on which a setup or hold violation occurs. For each error on a signal, the corresponding origin signals are also reported.

Error Report

The Error Margin is added to the hold and setup constraints of the figure. For example, if there is an error margin of 100ps, a signal with a setup or a hold time below 100ps will be reported as an error.

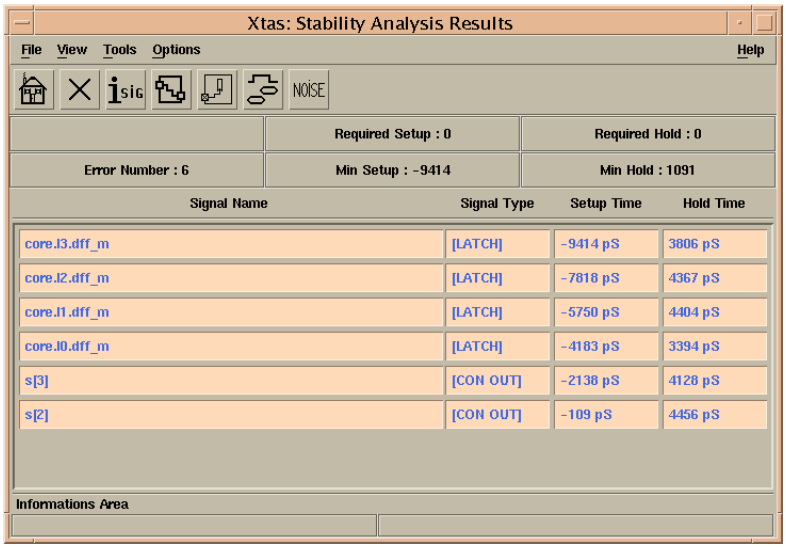
Information File

Specification for the stability analysis can be loaded from information file. Click on the 'Open' button to choose this file. It is possible to merge information from several files. In that way, you have to select files one by one setting the 'Complete' option. If you want to erase all information to load a new one, choose the 'Replace' option. If the `avtReadInformation File` is set, no need to do this operation, unless you want to merge another information. About the information file see the chapter 'Input File' of the HITAS Reference Guide.

The Crosstalk Analysis Results

After performing crosstalk analysis from within XTAS, all delays are subsequently given with crosstalk effect values (see HITAS Reference Guide for further information). To view these new delay values you have to open the critic path detail window (see section 'Viewing Path Details').

The Violating Signals Display



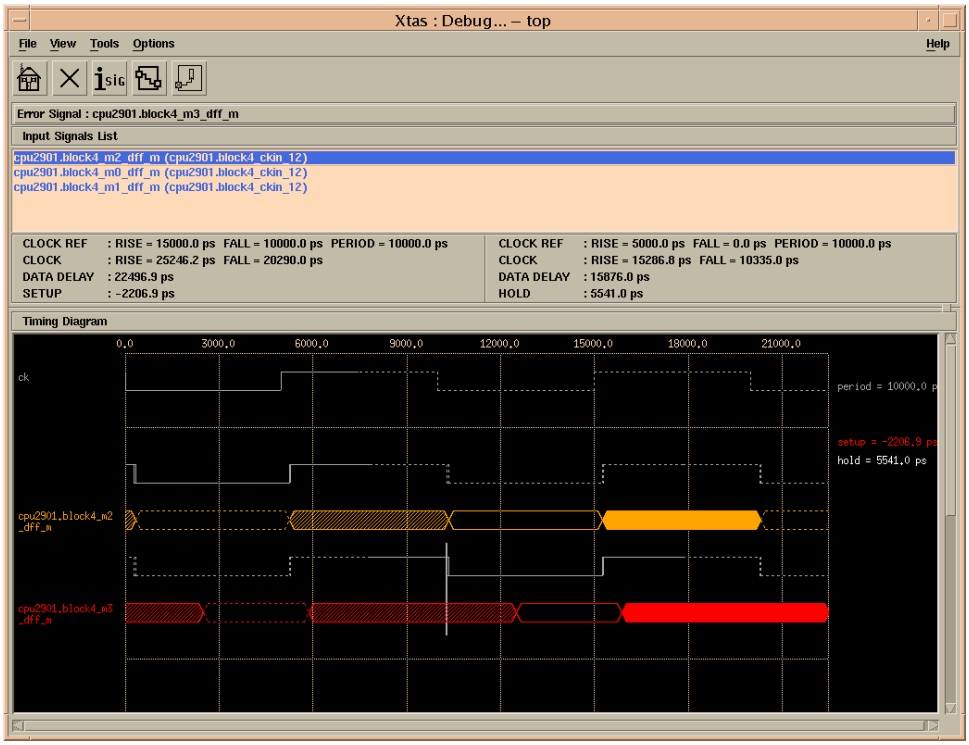
Any reference points for which the timing checks show errors re displayed in the static timing analysis results window.

In order to understand the violations, it is possible to display detailed timing diagrams for any terminal in the circuit.

Click on this button to open the signal Selection dialog.

Enter the name of any terminal in this dialog. If you selected a signal in the list of errors this signal is given by default. When you click on OK then XTAS displays the Debug... window.

The XTAS Debug window looks like:



The Input Signals List displays the input terminals of the paths which give rise to violations at the specified signal.

Selecting an input signal displays a timing diagram giving the signal switching windows of the path terminals as well as illustrating the required timing constraints.

14.12.3. Noise Analysis

Overview

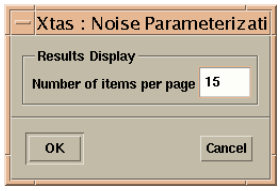
Coupling capacitance has the effect of generating noise on signals. The Noise Analysis calculates upper and lower peak voltages on a signal as a result of any aggression due to crosstalk. Then XTAS displays the results of the analysis as a list of signals sorted according to the peak noise value.

Analysis

The Noise Analysis needs to obtain information on aggressor and victim signals from memory. So Crosstalk Analysis must have been run from XTAS before launching the Noise Analysis. In the Static Timing Analysis Results window open the noise window in order to begin the Noise Analysis.

Clicking this button opens the XTAS Noise Parameterization window.

The XTAS Noise Parameterization window looks like:

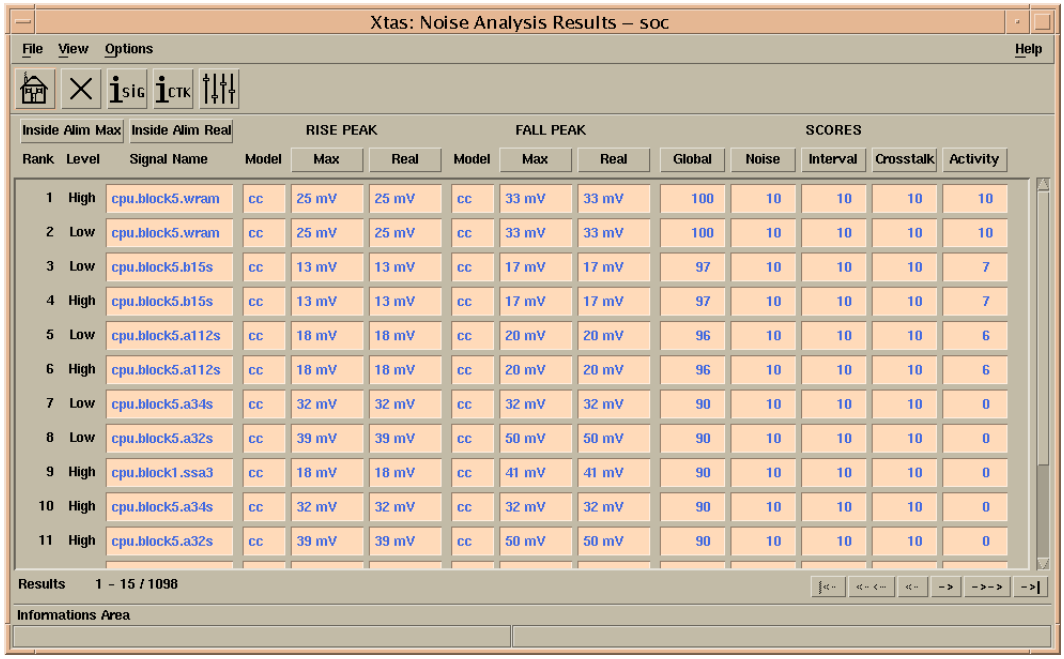


This window is the first step of the noise analysis.

The user can configure the results display by giving the number of item he wants to view per page.

Noise Analysis Results

The Noise analysis results window looks like:



The results are displayed as a list of signals with their noise information. To check which signals present a risk of crosstalk errors, a score based place is performed. There are four criteria: one for the crosstalk impact and three for the probability the aggression occurs. Each of these criteria gives a mark on 10 points.

Each line of the results list gives the following information on the corresponding signal:

General Information

- The rank in descending order of peak noise value
- The static level of the signal (High or Low).
- The signal name.

Rise and Fall Peak Information

- The electrical model used to evaluate peak noise.
- The "real" noise calculated considering possible switching configurations of aggressors.

- The maximum noise calculated with all aggressors considered active.

Scores

- Global: the total score computed with weighting provided by user (see next section).
- Noise: the impact of the noise peak. A mark of 10 means the noise peak reaches or exceeds the static threshold of the following gate.
- Interval: the part of aggressors which crosstalk can be simultaneously active at the same time.
- Crosstalk: the number of significant aggressors. The more the mark is about 10, the more the most significant part of crosstalk is due to a few number of aggressors.
- Activity: the activity of aggressor. For now, only aggressors located on a path clock are supposed to be always active, weighted by the part of crosstalk capacitance to this aggressor.

The user can find the same information in the Noise section of the Crosstalk Report (.ctk file). See HITAS Reference Guide for further information.

The user can sort the results on his favorite criterion by clicking on the corresponding button. The sort is always in descending order. The first five buttons from the right allows to sort on the corresponding score (Global, Noise, Interval, Crosstalk or Activity). The other criteria, based on the noise peak value, are the following:

Inside Alim Max

Signals are sorted on the maximum peak noise value, according to their level. This means that if the level of a signal is High, the selected peak noise value is the maximum fall peak noise value. If the level of a signal is Low, the selected peak noise value is the maximum rise peak noise value.

Inside Alim Real

Signals are sorted on the real peak noise value, according to their level. This means that if the level of a signal is High, the selected peak noise value is the real fall peak noise value. If the level of a signal is Low, the selected peak noise value is the real rise peak noise value.

Rise Peak Max

Signals are sorted on the maximum rise peak noise value, regardless of the static level.

Rise Peak Real

Signals are sorted on the real rise peak noise value, regardless of the static level.

Fall Peak Max

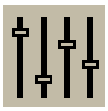
Signals are sorted on the maximum fall peak noise value, regardless of the static level.

Fall Peak Real

Signals are sorted on the real fall peak noise value, regardless of the static level.

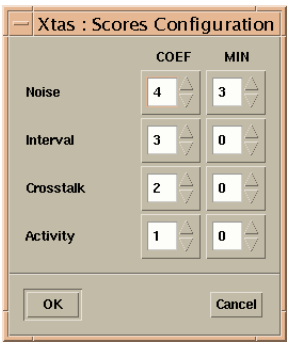
Scores Configuration

In the Noise Analysis Results window the user can configure the scores coefficients and minimum values.



Clicking on this button opens the Scores Configuration window.

The Scores Configuration window looks like:



This window allows the user to set two parameters for each score (Noise, Interval, Crosstalk and Activity). The first one is the weighting to compute the global score obtained by each signal. The second one is the minimum value required for the signal to be reported in the results list. These actions are equivalent to set the following configuration variables:

Noise `stbCtkCoefNoise` and `stbCtkMinNoise`.

Interval `stbCtkCoefInterval` and `stbCtkMinInterval`.

Crosstalk `stbCtkCoefCtk` and `stbCtkMinCtk`.

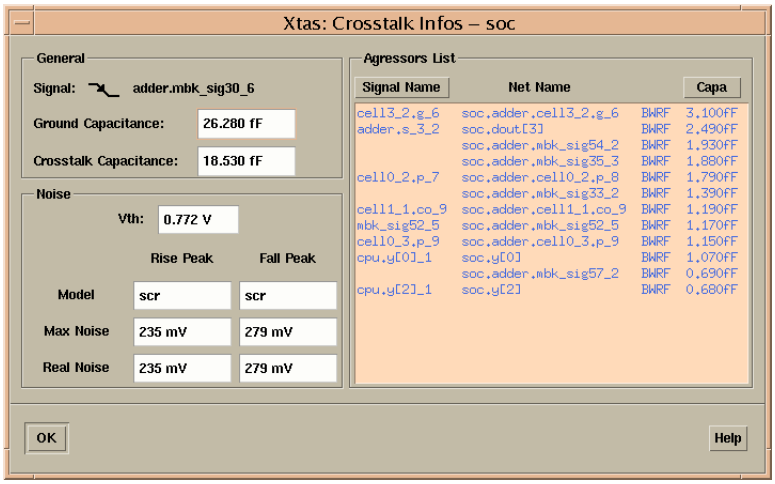
Activity `stbCtkCoefActivity` and `stbCtkMinActivity`.

Crosstalk Information

In the Noise Analysis Results window the user can access crosstalk information.

Selecting a signal and clicking on this button opens the Crosstalk Information window.

The Crosstalk Information window looks like:



General

This part gives the signal state and its name. It gives also the ground capacitance and the total crosstalk capacitance on this signal.

Noise

This part displays the electrical noise on the signal as it is specified in the Noise Analysis Results window (See previous section). Vth is the static threshold of the following gate.

Aggressors List

This part gives the list of the signal aggressors. Each line indicates the aggressor signal Name, its Net Name, the kind of influence it has on the victim and the coupling capacitance on the victim due to this aggressor.
If the signal does not appear on the line, there is no corresponding timing signal. As no stability information is provided for this signal, the crosstalk engine assumes that this signal is always an active aggressor.

If the character 'B' or 'W' or both are present on a line, this means that the aggressor can modify minimum propagation delays (B = Best Case) or maximum propagation delays (W = Worst Case).

If the character 'R' or 'F' or both are present on a line, this means that the aggressor has made a contribution to calculate the real rise (R) peak noise value or the real fall (F) peak noise value.

These characters can appear in lower case ('b','w','r','f') when crosstalk mutex are used. This means the influence of the signal is ignored because of the crosstalk mutex.

Aggressors can be sorted by the signal name or by the capacitance value using the button at the top of the corresponding column. Successive press on the button sort alternatively in ascending or descending order.

The user can find the same information in the Crosstalk section of the Crosstalk Report.

Chapter 15. Managing Big Designs

HITAS offers a set a variables dedicated to speed up execution or to limit the use of memory.

Memory-use strategies can be classified in three categories: the ones that use disk cache, the ones removing non-critical information, and the ones reducing the number of objects to treat by introducing sharing. Disk-caching strategies are accuracy lossless, but can severely impact execution runtimes. Object-sharing strategies try to use the same model for objects having close shapes. It can be really efficient, and is done in such a way that accuracy is preserved.

Execution speeding-up is most of the times done by keeping more objects in memory. However, it can also be correlated with the memory-use sharing strategies, as they limit the number of objects to treat.

15.1. File Compression and Disk Caching

HITAS is able to make a system call to any compression command. To get this mechanism work for input files, one only needs to set two variables: `avtInputFilter` for specifying the compression command to use, and `avtFilterSuffix`, in order to tell the tool which files it should apply the compression command on, for example:

```
avt_config avtInputFilter "gzip -d"
avt_config avtFilterSuffix ".gz"

avt_LoadFile my_design.spi spice
```

This will load seamlessly `my_design.spi` or `my_design.spi.gz`.

If `avtOutputFilter` is used, all output files will be compressed with the specified command and will have the suffix `avtFilterSuffix`.

The `stmCacheSize`, `avtMaxCacheFile` and `avtParasiticCacheSize` allow disk caching for very big design.

15.2. Information removal

Non-critical information can be removed through the variables `avtNoTransistorName` and `tasShortNamesForModels`.

15.3. Object sharing

Object sharing can be applied to the Timing Models, through the `stmShareModels` variable.

15.4. Execution Speed-up

Speeding-up strategies are essentially performed for crosstalk analysis, through variables `stbCtkFastMode` and `rcxFastMode`.

Chapter 16. Glossary

16.1. Logical Description

Subcircuit — Base object of a hierarchical description. The subcircuit is defined by its components, which may be elementary objects (transistors, resistances, capacitances...) or instances of other subcircuits.

Cell — Predefined subcircuit, only containing elementary objects, that may be selected and arranged to create custom or semi-custom integrated circuit.

Instance — Call of a subcircuit in a specific context

(Logical) Signal — Object carrying logical information between instances or elementary objects

Connector — Object carrying the logical information of a signal through the interface of an instance.

16.2. Physical Description

RC network — Connected set of R and C devices

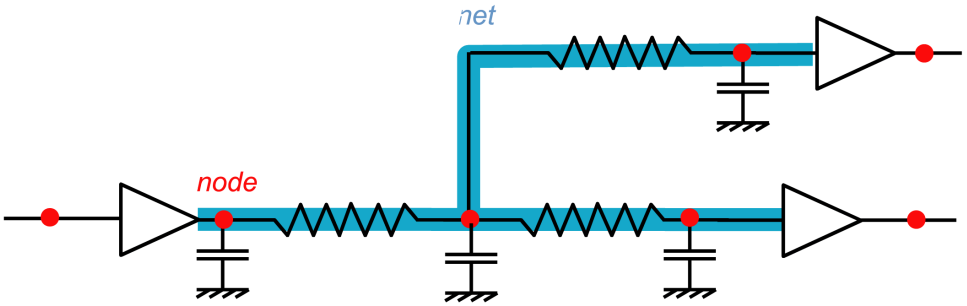
Net — Individual RC network linked to a logical signal. The RC network presents a resistive path between connectors of connected instances. Capacitances are substrate or coupling ones.

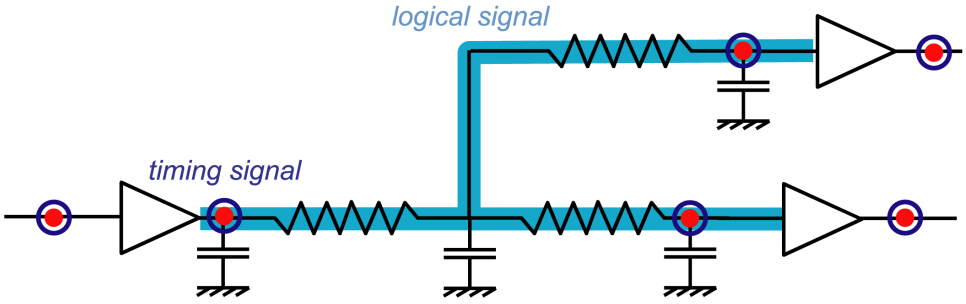
Node — Single point in a RC network

16.3. Timing Description

(Timing) Signal — Point of the circuit where the timing propagation of the logical information is measured. Whereas the logical signal is linked to a net, the timing signal is linked to a node. Indeed, as propagation delays in RC networks must be taken into account, it is necessary to measure the timing propagation of the logical information on several points of the net. The methodology in HITAS is to associate a timing

	signal with each terminal node of a net. As a result, several timing signals may exist where only one logical signal exist.
Event	Rising or falling logical transition on a timing signal
Reference point	Timing signal where timing checks must be performed: input or output connectors, latch, precharge, latch or precharge commands.
Path	List of timing signals, from reference point to reference point, through which logical information is carried
Break point	User defined reference point





Index

No index for this document.