"VERIFICATION OF CONFIGURABLE IOB ACCORDING TO JEDEC STANDARDS"

A dissertation submitted towards the partial fulfillment of the requirement for the Award of the Degree of

Master of Engineering
in

Electronics & Communication

Submitted by

Sanjib Deka
University Roll No: 3313
College Roll No: 08/E&C/03

Under the guidance of

Dr. Asok Bhattacharyya

DEPARTMENT OF ELECTRONICS & COMMUNICATION ENGINEERING
DELHI COLLEGE OF ENGINEERING
UNIVERSITY OF DELHI
2003-2005

CERTIFICATE

This is to certify that the project entitled **"Verification of Configurable IOB According to JEDEC Standards"**, which is being submitted by **Mr. Sanjib Deka,** is a bonafide record of student's own work carried by him under my guidance and supervision in partial fulfillment of requirement for the award of the Degree of **Master of Engineering** in **Electronic & Communication, Electronic & Communication Engineering Department, Delhi College of Engineering, University of Delhi.** The matter embodied in this project has not been submitted for the award of any other degree.

Dr. Asok Bhattacharyya

Project Guide

Professor & Head of Department
Department of Electronics & Communication Engineering
Delhi College of Engineering
Delhi - 110042

ACKNOWLEDGEMENT

It is my pleasure to acknowledge and express my deep sense of gratitude towards my guide, Dr. Asok Bhattacharyya, project guide & HOD (Electronic & Communication Dept, DCE), who supervised the work reported in this thesis report. I sincerely thank him for providing me with opportunity to pursue my internship in ST Microelectronics, Noida & being a constant source of inspiration.

I would like to thank Mr.Hiten Advani, Sectional manager, FPGA dept, for giving me opportunity to work in STMicroelectronics and for his help during the course of this project.

I thank Mr. Puneet Suri and Ms Veena Krishnan, my Project Guide, in STMicroelectronics for providing me with their knowledgeable suggestions and valuable support. Also I thank members of the team Mr. Janit Kumar, Mr. Satinder Singh Malhi, Mr. Arun Mishra, Mr. Prashant Narang, Ms. Shobhna Tiwari, Mr. Varinder Kumar and Mr. Vikrant Singh for their continuous guidance and support for the project. I am greatly thankful to Mr. Rahul Bharti from HR for his help during training.

I also like to thank everyone who has helped me either directly or indirectly in the successful completion of this work.

Sanjib Deka
College Roll No. 08/EC/2K3
University Roll No. 3103

“VERIFICATION OF CONFIGURABLE IOB ACCORDING TO JEDEC STANDARDS”

BY

Sanjib Deka
College Roll no: 08/EC/2K3
University Roll no: 3103

Abstract:
In this thesis Input Output block of FPGA is verified according to JEDEC standards. IOB’s used in FPGA’s are made configurable to support different applications. These IOB comply with electrical standard described by the JEDEC documents. In this thesis the postlay netlist of configurable IOB of FPGA is verified according to JEDEC standard. And the rise to rise delay, fall to fall delay and output duty cycle are also measured. Maximum frequency of operation of IOB was also verified. The simulations were carried out on EDA tools Eldo and Hspice. Measurements were done on the output waveforms. A relationship between rise delay, fall delay, input & output duty cycle was observed. An expression was derived which explains the behaviour observed, & this expression lets us calculate the maximum frequency when output is limited by duty cycle. The theoretical values found by the relation were confirmed with simulation results.

Table Of Contents

Chapter 1

Introduction

1.1 Introduction

Very Large Scale Integration (VLSI) technology has opened the door implementation of powerful digital circuits at low cost. It has become possible to build chips with more than a million transistors, as exemplified by state-of-the-art microprocessors. Such chips are realized using the *full custom* approach, where all parts of a VLSI circuit are carefully tailored to meet a set of specific requirements. Semi-custom approaches such as Standard Cells and Mask-Programmed Gate Arrays (MPGAs) have provided an easier way of designing and manufacturing Application-Specific Integrated Circuits (ASICs).

Each of these techniques, however, requires extensive manufacturing effort, taking several months from beginning to end. This results in a high cost for each unit unless large volumes are produced, because the overhead to begin production of chips ranges from $20,000 to $200,000.

In the electronics industry it is vital to reach the market with new products in the shortest possible time and so reduced development and production time is essential. Furthermore, it is important that the financial risk incurred in the development of a new product be limited so that more new ideas can be prototyped. Field-Programmable gate Arrays (FPGAs) have emerged as the ultimate solution to these time-to-market and risk problems because they provide instant manufacturing and very low-cost prototypes. An FPGA can be manufactured in minutes, and prototype costs are of the order of $100. A field-programmable device is a device in which the final logic structure can be directly configured by the end user, without the use of an integrated circuit fabrication facility.

Here's the general workflow when working with FPGAs:

- You use a computer to describe the "logic function" that you want. You might draw a schematic, or create a text file describing the function, doesn't matter.
- You compile the "logic function" on your computer, using a software provided by the FPGA vendor. That creates a binary file that can be downloaded into the FPGA.
- You connect a cable from your computer to the FPGA, and download the binary file to the FPGA.
- That's it! Your FPGA behaves according to your "logic function".

Keep in mind that

- You can download FPGAs as many time as you want - no limit - with different functionalities every time if you want. If you make a mistake in your design, just fix your "logic function", re-compile and re-download it. No PCB, solder or component to change.
- The designs can run much faster than if you were to design a board with discrete components, since everything runs within the FPGA, on its silicon die.
- FPGAs loose their functionality when the power goes away (like RAM in a computer that looses its content). You have to re-download them when power goes back up to restore the functionality.

1.2 FPGA

FPGAs (Field Programmable Gate Arrays) are arrays of logic blocks which can be linked together to form complex logic implementations. A bit more flexible and complex than CPLDs and PALs, FPGAs can be separated into two categories; Fine Grained and Coarse grained. Fine grained being made up of a sea of gates or transistors or small macro cells, while coarse grained being made up of bigger macro cells which are often made up of flip flops and Look up Tables (LUTs) which make up the combinatorial logic functions. Inside the macro cell are often switches or multiplexers which allow for differing uses of the macro cell. The individual macro cells are connected together with a combination of switch matrixes and metal line matrixes which can be implemented with pass transistors, fuses/antifuses or multiplexers.
A typical FPGA consists of a two-dimensional array of logic blocks that can be connected by general interconnection resources. These arrays of logic block and interconnection resources are surrounded by Input output blocks.

1.3 Input Output Block (IOB)

Any integrated circuit when looked inside package body can be broadly seen as two parts, i.e. core and IOB. Core is the silicon part where input signal provided by the IOB are processed and given back to IOB as output. IOB is the cell that allows interface between the logic inside the chip and external system components. In other words IOB protects the silicon from factors that may either damage the chip or hinder with chips functionality and performance.
I/O Logic Functions

An I/O can have three basic logic functions:
A). *Input*: The I/O receives and adapts the signal to the core. Since core and I/O work at different voltage levels. Signal has to be level shifted to core voltage level. This function can be selected from control bits.

B). *Output*: The IO amplifies and adapts the signal from the core to the outside load. Since core voltage is less than IO voltage IOB amplifies the signal and supply it to the load.

C). *Bidirectional:* IO can both work as input or output path. As shown in figure 3.5 this block has both input and output path. These blocks are selected from control signal from the core. In figure EN signal shown is active low. When EN is high block acts as input path, therefore ZI follows IO. And when EN is low this block acts as output path.

Specific functions of IO
A). *Pull-up and pull-down:* The optional pull-up and pull-down resistors are intended to establish High and Low levels, respectively, at unused I/Os.

B). *Hysteresis*: This feature is included, make core more immune to noise. There are two level of voltage known as trim points.

C). *Analog IO's*: These IO's are designed to carry analog signal e.g. oscillators, ADC. They have different current and voltage requirements thus have different design.

D). *Power Supply IO's*: These IO's are designed to carry supply for core and IO. They have inbuilt ESD protection circuitry.

E). *Latch –up*: The IO structure is more susceptible to latch-up. So the layout are made keeping this mind

1.4 JEDEC standards

JEDEC is the leading developer of standards for the solid-state industry. Almost 2400 participants, appointed by some 270 companies work together in 50 JEDEC committees meet the needs of every segment of the industry, manufacturers and consumers alike. The publications and standards that they generate are accepted throughout the world

JEDEC committees hold frequent meetings throughout the year in domestic and international venues. All standardization work takes place at these meetings and companies must be a member to participate. Surveys are also often taken to find out what companies are doing in important areas that often leads to ballots.

In this thesis the IOB which will be verified follow JEDEC standard.

1.5 Verification

1.5.1 Schematic-Based Simulation

Design simulation involves testing of design using software models. It is most effective when testing the functionality of the design and its performance under worst-case conditions. We can easily probe internal nodes to check your circuit's behaviour, and then use these results to make changes in your schematic.

Simulation is performed using third-party tools that are linked to the Xilinx Development System. The software models provided for the simulation tools are designed to perform detailed characterization of your design. We can perform functional or timing simulation, as described follows:

1.5.2 Functional Simulation

Functional simulation determines if the logic in the design is correct before it is implemented in a device. Functional simulation can take place at the earliest stages of the design flow. Because timing information for the implemented design is not available at this stage, the simulator tests the logic in the design using unit delays.

It is usually faster and easier to correct design errors if you perform functional simulation early in the design flow.

1.5.3 Timing Simulation

Timing simulation verifies that the design runs at the desired speed for the device under worst-case conditions. This process is performed after your design is mapped, placed, and routed for FPGAs. At this time, all design delays are known. Timing simulation is valuable because it can verify timing relationships and determine the critical paths for the design under worst-case conditions. It can also determine whether or not the design contains set-up or hold violations.

1.6 Goals of the Project

This project was done in STMicroelectronics pvt ltd, circuit made by the design team needs to be verified before tape-out. The circuit was designed according to the JEDEC standards, therefore it needs to be verified that it is behaving according to the JEDEC standard

The goals of the project this is to verify that the circuit designed by the design team, behaves electrically according to standard. The following are the things to be verified

- It follows electrical standards as specified by the JEDEC standard.
- Verify working of the circuit under the maximum frequency.
- Check whether the output waveform have distortion or not.
- Give feedback to the design team.
- Suggest changes if there is some problem in the circuit.

The following is the list of parameters measured for verification purpose.

- Input on time.
- Input off time.
- Input time period.
- Output on time.
- Output off time.
- Output time period.
- Rise to rise delay.
- Fall to fall delay.
- Output voltage low level (VOL).
- Output voltage high level (VOH).
- Peak to peak value of output.
- Output Duty cycle.

1.7 Organisation of Thesis

Report title is 'Verification of configurable IOB'. Configurable IOBs are one of the basic blocks of FPGA. These blocks give electrical isolation to the core.

Chapter 2 gives the literature review of the topics which are directly or indirectly related to this project. The function of this chapter is to give reader all the introductory information needed to understand the project. First we discuss about FPGAs, FPGA architecture, building blocks, programming technologies and Applications of FPGA. Then a block of FPGA known as IOB is taken up. This block is explained extensively. Basic blocks of IOB, their importance and protection circuits are discussed. Issues like latch-up, EMC and ESD and method to prevent IOB from these issues are examined. In this project IOB were verified against a standard known as JEDEC standards. Information on JEDEC is provided in the end of the chapter.

Chapter 3 discusses about verification procedure followed. Details of parameter that were measured and cir file are given in this chapter.

Chapter 4 tells how verification was done on the postlay netlist of IOB.

Chapter 5 discusses result and conclusion of the project.

In Appendix syntax of simulators used are added for reference.

Chapter 2

Literature Review

2.1 Introduction

This chapter covers the literature background needed to understand the project. In this chapter FPGA, its architecture, & IOB are discussed. And issues like ESD, Latch-up are also discussed. Reader can skip these issues if the like, this will not hinder in flow of the thesis.

2.2 Field Programmable Gate Array

A programmable logic device is a device whose logic characteristics can be changed and manipulated or stored through programming. The most common simple device which falls into this category is the *PAL (Programmable Array Logic)*. The internals of simple PALs consist simply of an array of AND gates and an array of OR gates. The AND array is programmable while the OR array is relatively fixed. A switch matrix selects which of the PALs inputs will be connected to the AND inputs which are then connect to the fixed OR matrix. The outputs can often times be rerouted to the input matrixes.

FPGAs (Field Programmable Gate Arrays) are arrays of logic blocks which can be linked together to form complex logic implementations. A bit more flexible and complex than CPLDs and PALs, FPGAs can be separated into two categories; Fine Grained and Coarse grained. Fine grained being made up of a sea of gates or transistors or small macro cells, while coarse grained being made up of bigger macro cells which are often made up of flip flops and Look up Tables (LUTs) which make up the combinatorial logic functions. Inside the macro cell are often switches or multiplexers which allow for differing uses of the macro cell. The individual macro cells are connected together with a combination of switch matrixes and metal line matrixes which can be implemented with pass transistors, fuses/antifuses or multiplexers.

2.3 Basic Blocks of FPGA

A typical FPGA consists of a two-dimensional array of logic blocks that can be connected by general interconnection resources. The *interconnection resources* comprise of segments of wire, where the segments may be of various lengths. Programmable switches present in interconnect serve to connect the logic blocks to the wire segments, or one wire segment to another. Logic circuits are implemented in the FPGA by partitioning the logic into individual logic blocks and then interconnecting the blocks as required via the switches.

To facilitate the implementation of a wide variety of circuits, it is important that the FPGA be as versatile as possible. This means that the design of the logic block, coupled with that with of the interconnection resources, should facilitate the implementation of a large number of digital logic circuits.

2.3.1 Logic blocks

Logic block can be designed in many different ways. Some FPGA logic blocks are as simple as 2-input NAND gates, other blocks have more complex structure, such as multiplexer or look-up tables. In some FPGA's, a logic block correspond to an entire PAL like structure. Most logic block also contains some type of flip-flop, to aid in the implementation of sequential circuits.

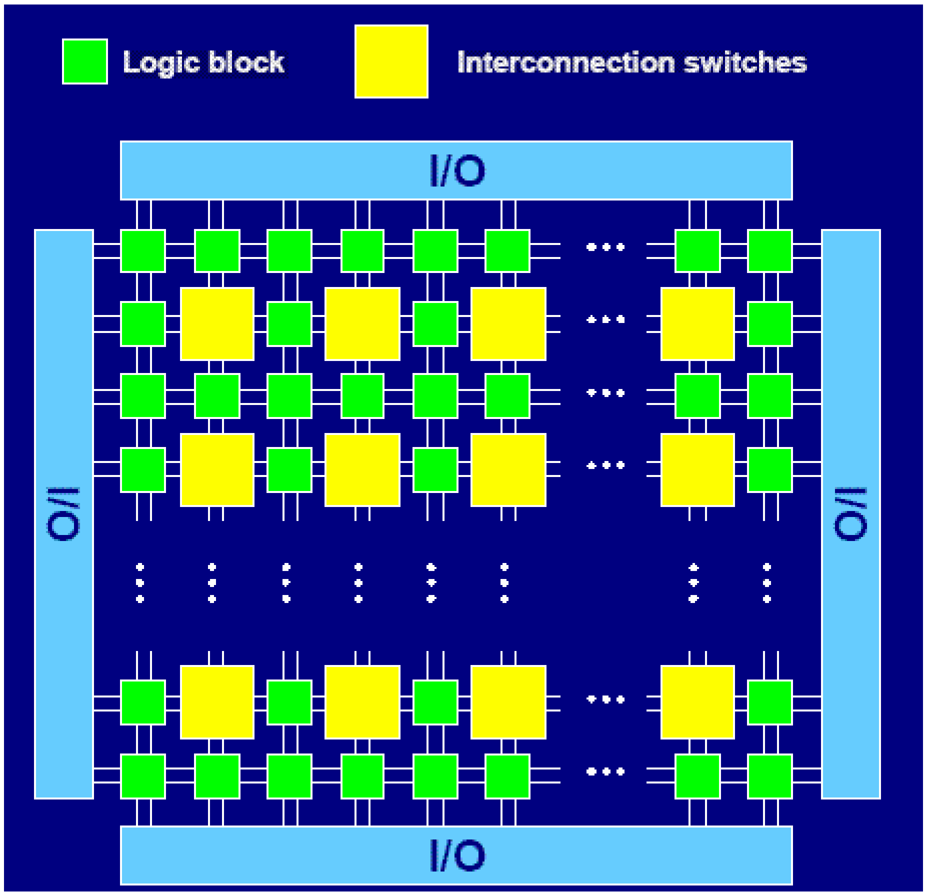


Fig 2.1 Block diagram of FPGA

2.3.2 Interconnection resources

The structure and content of interconnect in a FPGA is called its routing architecture. The routing architecture consists of both wire segments and programmable switches. The programmable switches can be constructed in many ways, including: pass transistor controlled by static RAM cells, anti fuses, EPROM transistors and EEPROM transistors.

2.4 Programming technologies

The term ‘switch’ generally used in FPGA’s refers to the entity that allows programmable connections between wire segments. A more precise term for such an entity is programming element. This programming element can be implemented using different programming technologies. In this section programming technologies will be briefly explained.

Programmable technologies that are currently in use in commercial products are: static RAM cells, anti-fuses, EPROM transistors, and EEPROM transistors. The programming connections are used to implement the programmable connection among the FPGAs logic blocks.

The elements should have following properties:

- The programming element should consume as little chip area as possible.
- The programming element should have a low ON resistance and a very high OFF resistance.
- The programming element should contribute low parasitic capacitance to the wiring resources to which it is attached.
- It should be possible to reliably fabricate a large number of programming elements on a single chip.

2.4.1 Static RAM Programming Technology

The static RAM programming technology is used in FPGA's produced by several companies: Algotronix, Concurrent logic, Plessey Semiconductors, and Xilinx. In these FPGA's, programmable connection are made using pass-transistors, transmission gates, or multiplexer that are all controlled by SRAM cells.

In case of pass-transistor approaches in figure 2.2a and 2.2b RAM cell controls whether RAM cell controls whether the pass gate is on or off. When off, the pass gate present a very high resistance between the two wires to which it is attached. When pass gate is turned on, it forms a relatively low resistance connection between the wires. For multiplexer approach in figure 2.2c, the RAM cells control which of the multiplexer's input should be connected to its output. This option would typically be used to optionally connect one of several wires to a single input of a logic block.

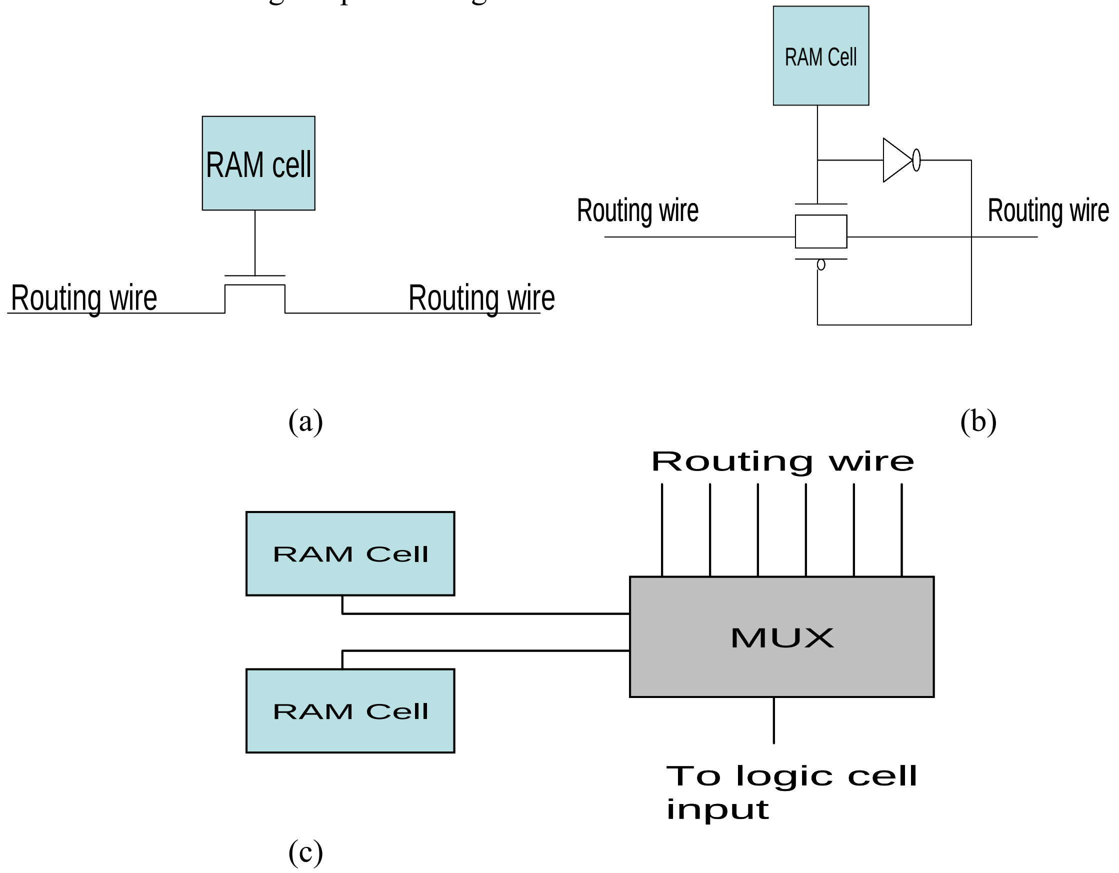


Fig 2.2 Static RAM programming technology

In an FPGA that uses SRAM programming technology, the logic blocks may be interconnected using a combination of pass-gates and multiplexers. Since the static RAM is volatile, these FPGAs must be configured each time the power is applied to the chip.

This requires external permanent memory to provide programming bits such as PROM, EPROM, EEPROM or magnetic disk.

A major disadvantage of SRAM programming technology is its large area. It takes at least five transistors to implement each SRAM cell, plus at least one transistor to serve as a programmable switch. However, SRAM programming technology two major advantages: fast re-programmability and that it require only standard integrated circuit process technology.

2.4.2 Anti-fuse programming technology

Anti-fuse programming technology is used in FPGAs by Actel Corp., Quicklogic, and Crosspoint Solutions. While the anti-fuse used in each of these FPGAs differs in construction, their function is the same. A anti-fuse normally resides in a high-impedance state but can be 'fused' into a low-impedance state when programmed by a high voltage.

The Actel anti-fuse, called PLICE, can be described as a square structure that consists of three layers: the bottom layer is composed of positively-doped silicon (n+ diffusion), the middle layer is a dielectric (oxygen-nitrogen-oxygen insulator), and the top layer is made of poly-silicon. This construction is illustrated in figure 2.3.

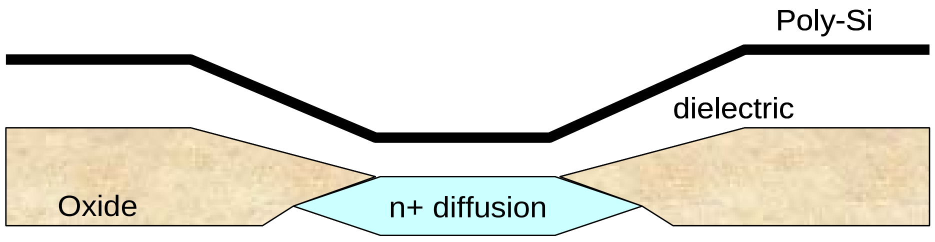


Fig 2.3 Cross section of PLICE Anti-fuse programming technology

The PLICE anti-fuse is programmed by placing a relatively high voltage (18V) across the anti-fuse terminals and driving a current of about 5mA through the device. This procedure generates enough heat I the dielectric to cause it to melt and form a conductive link between the poly-Si and the n+ diffusion. Special high voltage transistors are fabricated within the FPGA to accommodate the necessary large voltages and currents.

Both the bottom layer and the top layer of the anti-fuse are connected to metal wires, so that, when programmed, the anti-fuse forms a low resistance connection (from 300 to 500 ohms) between the two metal wires. This arrangement is depicted in figure 2.4. The PLICE anti-fuse is manufactured by adding three specialized mask to a normal CMOS process.

The anti-fuse used by Quicklogic is called Vialink. It is similar to the PLICE anti-fuse in that it consists of three layers. However, a Vialink anti fuse uses one level of metal for its bottom layer, an alloy of amorphous silicon fir its middle layer, and a second level of metal for the top layer.

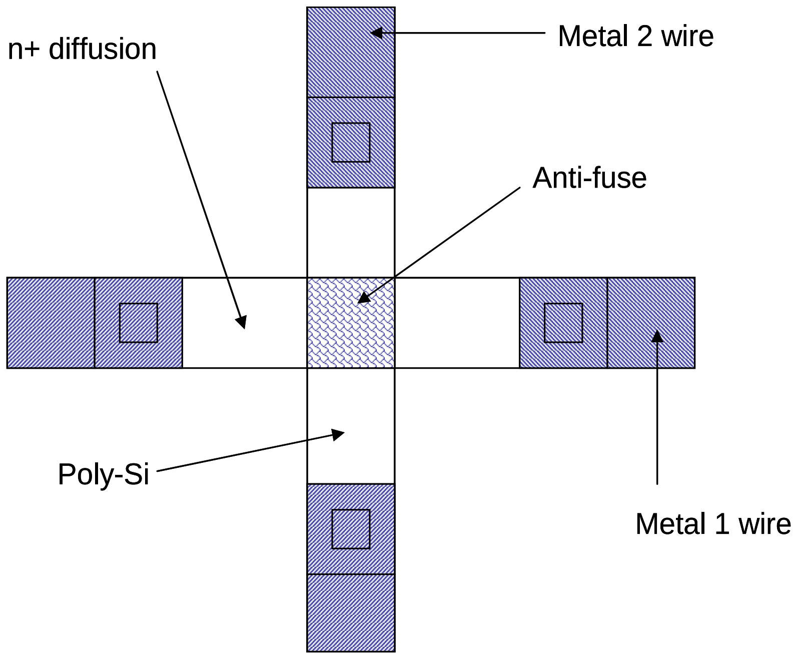


Fig 2.4 Structure of PLICE anti-fuse programming technology

This structure is illustrated I figure 2.5. When in un-programmed state, the anti-fuse presents over a giga ohm of resistance, but when programmed it forms a low resistance path of about 80 ohms between the two metal wires. The anti-fuse is manufactured using three extra masks above a normal CMOS process. Here, a normal via is created for the anti-fuse, but the via si filled with the amorphous silicon alloy instead of metal.

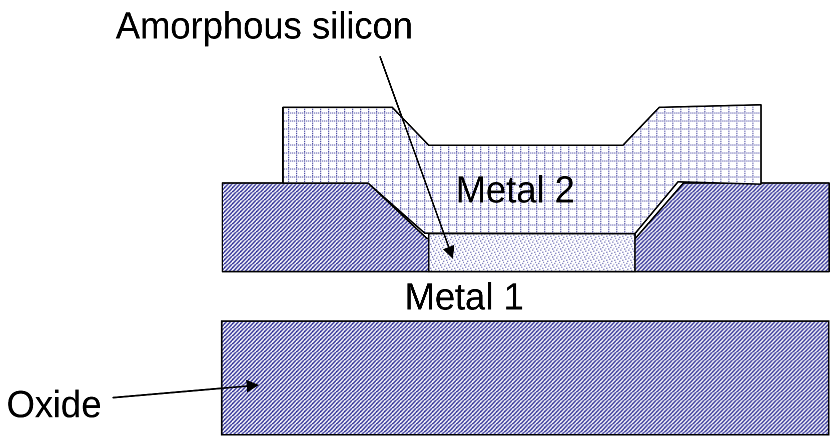


Fig 2.5 Vialink anti-fuse programming technology

The Vialink anti-fuse is programmed by placing about 10 volts across its terminals. When sufficient current is supplied, this results in a change of state in the amorphous silicon and creates a conductive link between the bottom and the top layers of metal.

The chip area required by an anti-fuse (either PLICE or Vialink) is very small compared to the other programming technologies. However, this is somewhat offset by the large space required for the high-voltage transistors that are needed to handle the high programming voltages and currents. A disadvantage of anti-fuses is that their manufacture requires modifications to the basic CMOS process.

2.4.3 EPROM and EEPROM programming technology

EPROM programming technology is used in the FPGAs manufactured by Altera Corp. and Plus logic. This technology is the same as that used in EPROM memories.
Unlike a simple MOS transistor, an EPROM transistor consists of two gates, a floating gate and a select gate. The floating gate, positioned between the select gate and the transistor channel, is so named because it is not electrically connected to any circuitry. In its normal (unprogrammed) state, no charge exists on the floating gate and the transistor can be turned ON in the normal fashion using the select gate.

However when the transistor is programmed by causing a large current to flow between the source and drain, a charge has the effect of permanently turning the transistor OFF. I this way, the EPROM transistor can function as a programmable element. An EPROM transistor can be re-programmed by first removing the trapped charge from the floating gate. Exposing the gate to ultraviolet light excites the trapped electron to the point where they can pass through the gate oxide into the substrate.

EPROM transistors are used in FPGAs in a different manner than are static RAM cells or anti-fuses. That is, rather than serving to programmably connect two wires, EPROM transistors are used as “pull down” devices for logic block inputs. This arrangement is shown in fig 2.5. One wire is called the word line, and is connected to the select gate of the EPROM transistor. As long as the transistor has not been programmed into OFF state, the word line can cause the “bit line”, which is connected to a logic block input to be pulled to logic zero. Since a pull-up resistor is present on the bit line, this scheme allows the EPROM transistors to not only implement connections but also to realize wired-AND logic functions.

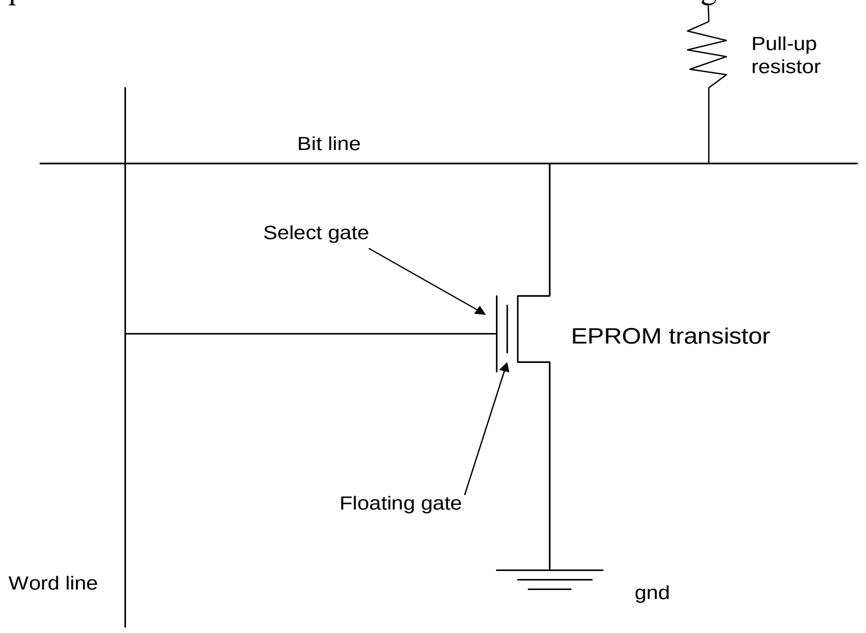


Fig 2.6 EPROM programming technology

A disadvantage of this approach is that the resistor consumes static power. One advantage of EPROM transistors is that they are re-programmable but do not require external voltage.

The EEPROM approach (used in AMD FPGAs) is similar to the EPROM technology except that EEPROM transistors can be re-programmed in-circuit. The disadvantage of using

EEPROM transistors is that they consume about twice chip area as EPROM transistor and they require multiple voltage sources (for re-programming) which might not otherwise be required.

2.5 Comparison of programmable technology

Programming Technology	*Volatile*	*Re-Prog.*	*Chip Area*	*R (ohm)*	*C(ff)*	*Extra FAB Steps*
Static RAM Cells	yes	In-circuit	large	1 – 2 K	10–20 ff	0
PLICE Anti-fuse	No	No	Small anti-fuse, large prog. trans	300-500	3 – 5 ff	3
Via-Link Anti-fuse	No	No	Small anti-fuse, large prog. trans.	50-80	1.3 ff	3
EPROM	No	Out of circuit	Small	2 – 4 K	10–20 ff	3
EEPROM	No	In-circuit	2x EPROM	2 – 4 K	10–20 ff	>5

Table 2.1 characteristic of programming technologies

2.6 Applications of FPGAs

FPGAs can be used in almost all of the applications that currently use Mask-Programmable Gate Arrays, PLDs and small scale integration (SSI) logic chips. Below we present a few categories of such designs.

- Application-Specific Integrated Circuits (ASICs)
- Implementation of Random Logic
- Replacement of SSI Chips for Random Logic
- Prototyping
- FPGA-Based Compute Engines
- On-Site Re-configuration of Hardware

2.7 Design Flow Overview

The standard design flow consists of the following steps:

- ***Design Entry and Synthesis***

In this step of the design flow, you create your design using a Xilinx-supported schematic editor, a Hardware Description Language (HDL) for text-based entry, or both. If you use an HDL for text-based entry, you must synthesize the HDL file into an EDIF or XNF file or, if you are using the Xilinx Synthesis Technology (XST) GUI, into an NGC file.

- ***Design Implementation***

By implementing to a specific Xilinx architecture, you convert the logical design file format, such as EDIF, that you created in the design entry or synthesis stage into a physical file format. The physical information is contained in the Native Circuit Description (NCD) file for FPGAs and the VM6 file for CPLDs. Then you create a bitstream file from these files and optionally program a PROM or EPROM for subsequent programming of your Xilinx device.

- ***Design Verification***

  Using a gate-level simulator or cable, you ensure that your design meets your timing requirements and functions properly.

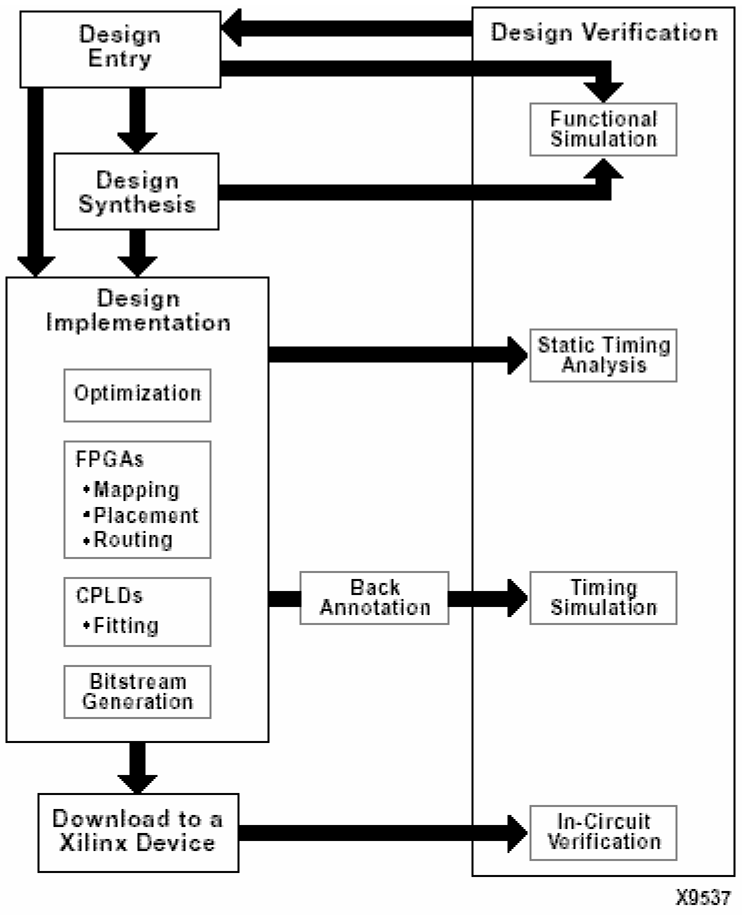


Fig 2.7 Design flow

The full design flow is an iterative process of entering, implementing, and verifying your design until it is correct and complete. The Xilinx Development System allows quick design iterations through the design flow cycle. Because Xilinx devices permit unlimited reprogramming, we do not need to discard devices when debugging the design in circuit.

2.7.1 Design Entry and Synthesis

You can enter a design with a schematic editor or a text-based tool. Design entry begins with a design concept, expressed as a drawing or functional description. From the original design, a netlist is created, then synthesized and translated into a Native Generic Object (NGO) file. This file is fed into a program called NGDBuild, which produces a logical Native Generic Database (NGD) file.

Xilinx Development System Hierarchical Design

Design hierarchy is important in both schematic and HDL entry for the following reasons:

- Helps to conceptualize your design.
- Adds structure to the design.
- Promotes easier design debugging.
- Makes it easier to combine different design entry methods (schematic, HDL, or state editor) for different parts of your design
- Makes it easier to design incrementally, which consists of designing, implementing, and verifying individual parts of a design in stages Design Flow.
- Reduces optimization time.
- Facilitates concurrent design, which is the process of dividing a design among a number of people who develop different parts of the design in parallel, such as in Modular Design.

Schematic Entry Overview

Schematic tools provide a graphic interface for design entry. One can use these tools to connect symbols representing the logic components in your design. We can build our own design with individual gates, or can combine gates to create functional blocks.

Library Elements

Primitives and macros are the building blocks of component libraries. Xilinx libraries provide primitives as well as common high-level macro functions. Primitives are basic circuit elements, such as AND and OR gates. Each primitive has a unique library name, symbol, and description.

HDL Entry and Synthesis

A typical Hardware Description Language (HDL) supports a mixed level description in which gate and netlist constructs are used with functional descriptions. This mixed-level capability enables us to describe system architectures at a high level of abstraction, then incrementally refine a design s detailed gate-level implementation.

HDL descriptions offer the following advantages:

- We can verify design functionality early in the design process. A design written as an HDL description can be simulated immediately. Design simulation at this high level, at the gate level before implementation, allows us to evaluate architectural and design decisions.
- An HDL description is more easily read and understood than a netlist or schematic description. HDL descriptions provide technology-independent documentation of a design and its functionality. Because the initial HDL design description is technology independent, we can use it again to generate the design in a different technology, without having to translate it from the original technology.
- Large designs are easier to handle with HDL tools than schematic tools.

After creating our HDL design, we must synthesize it. During synthesis, behavioural information in the HDL file is translated into a structural netlist, and the design is optimized for a Xilinx device. Xilinx supports HDL synthesis tools for several third-party synthesis vendor partners. In addition, Xilinx offers its own synthesis tool, *Xilinx Synthesis Technology (XST).*

Functional Simulation

After we enter the design, we can simulate it. Functional simulation tests the logic in the design to determine if it works properly.

Constraints

If we want to want to constrain your design within certain timing or placement parameters, we can specify mapping, block placement, and timing specifications. One can also enter constraints by hand or use the Constraints Editor, Floor planner, or FPGA Editor. We can also use the Timing Analyzer Graphical User Interface (GUI).

2.7.2 Design Verification

Design verification is the process of testing the functionality and performance of your design. You can verify Xilinx designs in the following ways:

- Simulation (functional and timing)
- Static timing analysis
- In-circuit verification

Design verification procedures should occur throughout your design process, as shown in the following figures.

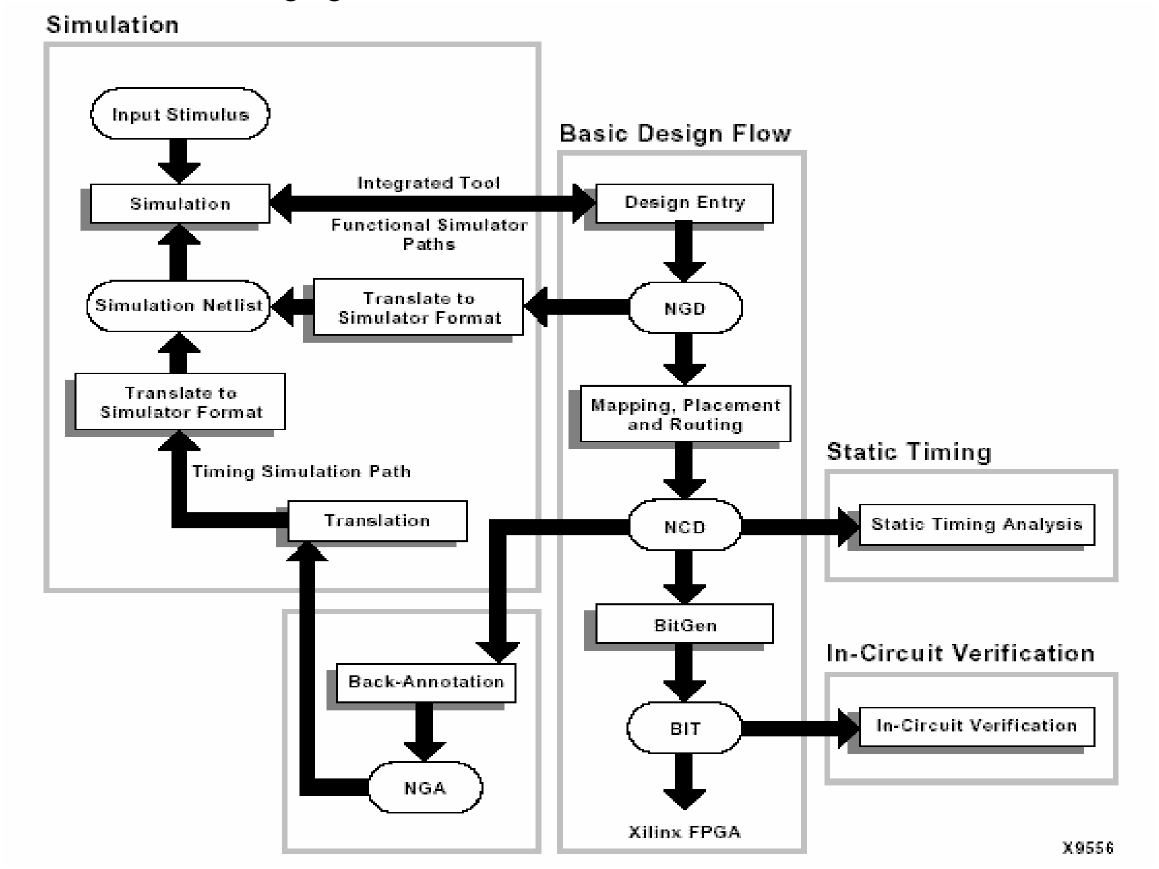


Fig 2.8 Design verification

2.7.2.1. Simulation

We can run functional or timing simulation to verify our design. A back-annotation process must occur prior to timing simulation.

Back-Annotation: Before timing simulation can occur, the physical design information must be translated and distributed back to the logical design. For FPGAs, this back-annotation process is done with a program called NGDAnno. These programs create a database for the netlist writers, which translate the back-annotated information into a netlist format that can be used for timing simulation. The following figures show the back-annotation flows.

NGDAnno: NGDAnno is a command line program that distributes information about delays, setup and hold times, clock to out, and pulse widths found in the physical NCD design file back to the logical NGD file. NGDAnno reads an NCD file as input. The NCD file can be a mapped-only design or a partial or fully placed and routed design.

An NGM file, created by MAP, is an optional source of input. NGDAnno merges mapping information from the NGM file with placement, routing, and timing information from the NCD file. NGDAnno outputs a Native Generic Annotated (NGA) file, which is a back-annotated NGD file. This file is input to the appropriate netlist writer, which converts the binary Xilinx database format back to an ASCII netlist.

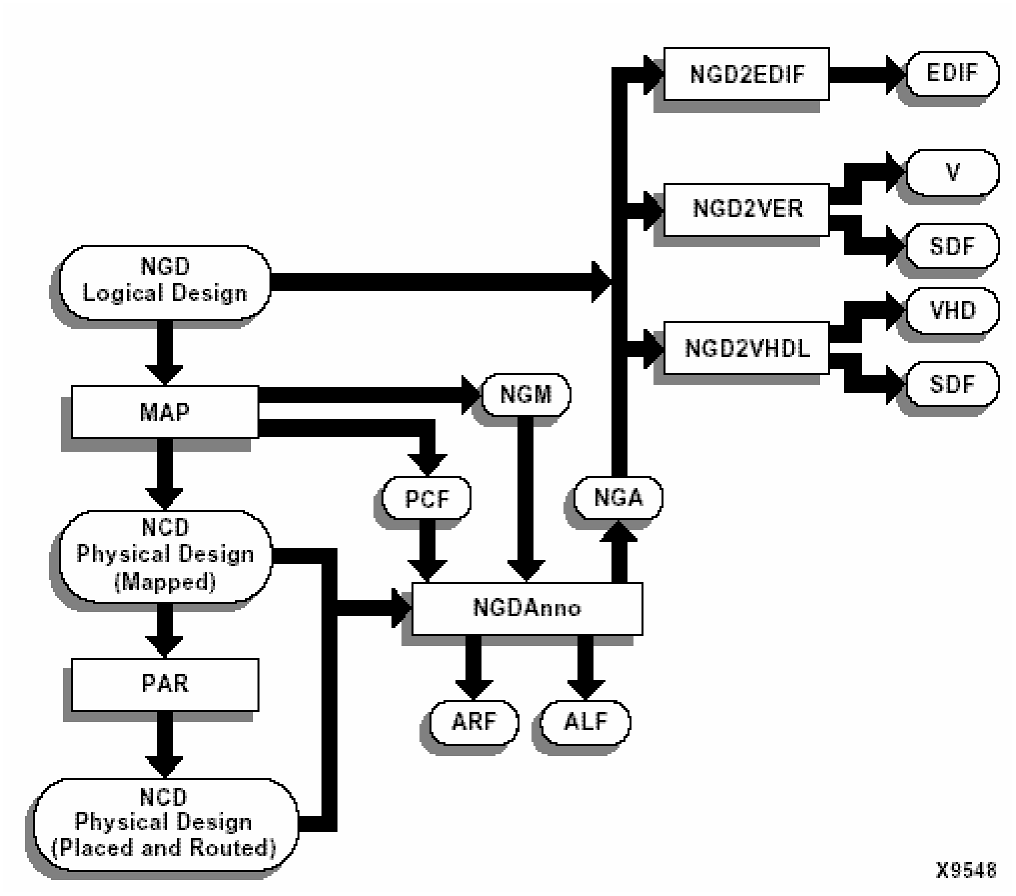


Fig 2.9 Simulation block diagram

Netlist Writers: Netlist writers (NGD2EDIF, NGD2VER, or NGD2VHDL) take the output of NGDAnno and create a simulation netlist in the specified format. An NGD or NGA file is input to each of the netlist writers. The NGD file is a logical design file containing primitive components, while the NGA file is a back-annotated logical design file. NGD2VER translates an NGD or NGA file into a Verilog netlist (V) file. If the input is an NGA file, NGD2VER also generates an SDF (Standard Delay Format) file. The resulting V and SDF files have the same root name as the NGD or NGA file unless you specify otherwise.

Schematic-Based Simulation: Design simulation involves testing of design using software models. It is most effective when testing the functionality of the design and its performance under worst-case conditions. We can easily probe internal nodes to check your circuit's behaviour, and then use these results to make changes in your schematic.

Simulation is performed using third-party tools that are linked to the Xilinx Development System. The software models provided for the simulation tools are designed to perform detailed characterization of your design. We can perform functional or timing simulation, as described follows:

Functional Simulation: Functional simulation determines if the logic in the design is correct before it is implemented in a device. Functional simulation can take place at the earliest stages of the design flow. Because timing information for the implemented design is not available at this stage, the simulator tests the logic in the design using unit delays.

It is usually faster and easier to correct design errors if you perform functional simulation early in the design flow.

Timing Simulation: Timing simulation verifies that the design runs at the desired speed for the device under worst-case conditions. This process is performed after your design is mapped, placed, and routed for FPGAs. At this time, all design delays are known.

Timing simulation is valuable because it can verify timing relationships and determine the critical paths for the design under worst-case conditions. It can also determine whether or not the design contains set-up or hold violations.

2.7.2.2. Static Timing Analysis

Static timing analysis is best for quick timing checks of a design after it is placed and routed. It also allows you to determine path delays in your design. Following are the two major goals of static timing analysis:

Timing verification: This is the process of verifying that the design meets your timing constraints.

Reporting: This is the process of enumerating input constraint violations and placing them into an accessible file. We can analyze partially or completely placed and routed designs. The timing information depends on the placement and routing of the input design.

2.7.2.3. In-Circuit Verification

As a final test, we can verify how the design performs in the target application. In-circuit verification tests the circuit under typical operating conditions. Because we can program our Xilinx devices repeatedly, we can easily load different iterations of the design into the device and test it in-circuit. To verify our design in-circuit, we should download the design bitstream into a device with the Parallel Cable III.

Design Rule Checker: Before generating the final bitstream, it is important to use the DRC option in BitGen to evaluate the NCD file for problems that could prevent the design from functioning properly.

2.8 Input Output Block (IOB)

Any integrated circuit when looked inside package body can be broadly seen as two parts, i.e. core and IOB. Core is the silicon part where input signal provided by the IOB are processed and given back to IOB as output. IOB is the cell that allows interface between the logic inside the chip and external system components. In other words IOB protects the silicon from factors that may either damage the chip or hinder with chips functionality and performance. In this chapter IOB's are discussed in detail.

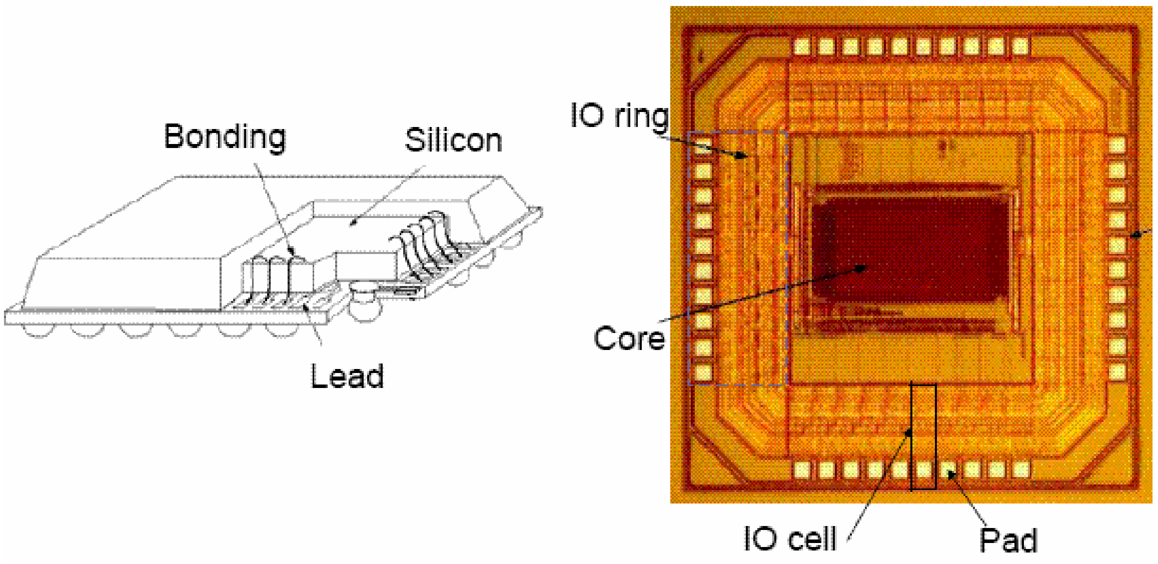


Fig 2.10 Cross section view of an IC at package level and at silicon level

2.8.1 Basic of I/O

2.8.1.1 What is an I/O?

IOB is the cell that allows interface between the logic inside the chip and external system components. As shown in figure this cell has two paths for signal.

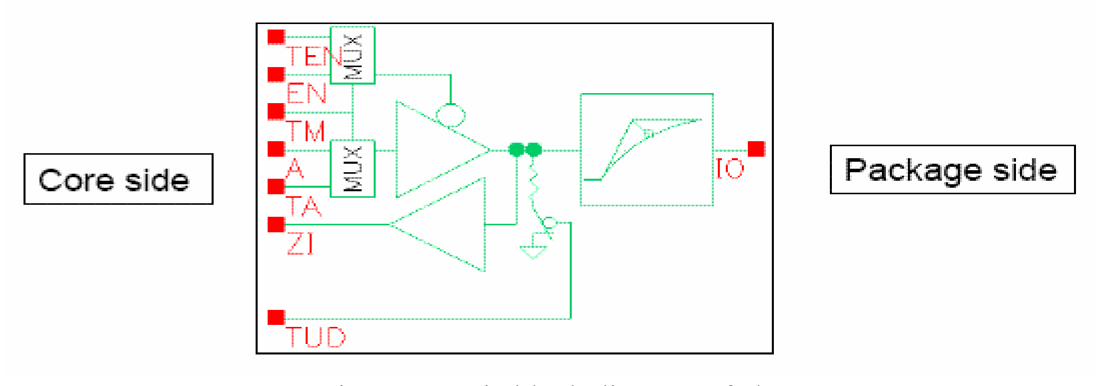


Fig 2.11 Basic block diagram of I/O

One path is for input whereas the other is for output. There are some additional bits known as control bits, which are used by core to configure IOB. IOB and core have different power supply. Core voltage is dependent upon technology being used. I/O voltage complies with IO standard, which in turn is selected according to interface voltage level of external system to which the chip is interfaced.

2.8.1.2 I/O Logic Functions

An I/O can have three basic logic functions:

A). *Input*: The I/O receives and adapts the signal to the core. Since core and I/O work at different voltage levels. Signal has to be level shifted to core voltage level. This function can be selected from control bits.

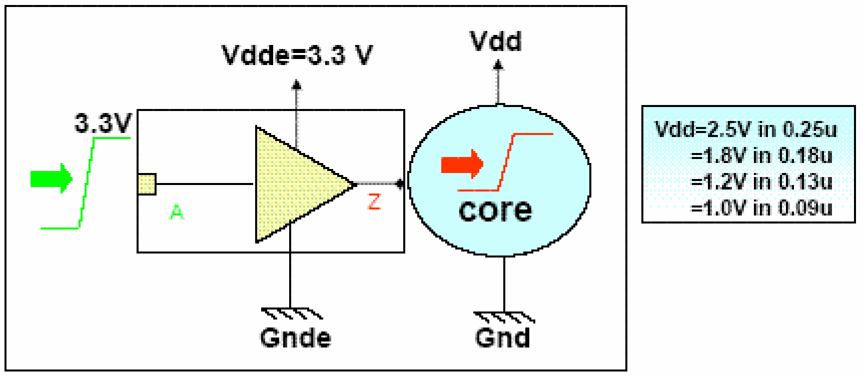


Fig 2.12 Block representation of Input path

B). *Output*: The IO amplifies and adapts the signal from the core to the outside load. Since core voltage is less than IO voltage IOB amplifies the signal and supply it to the load.

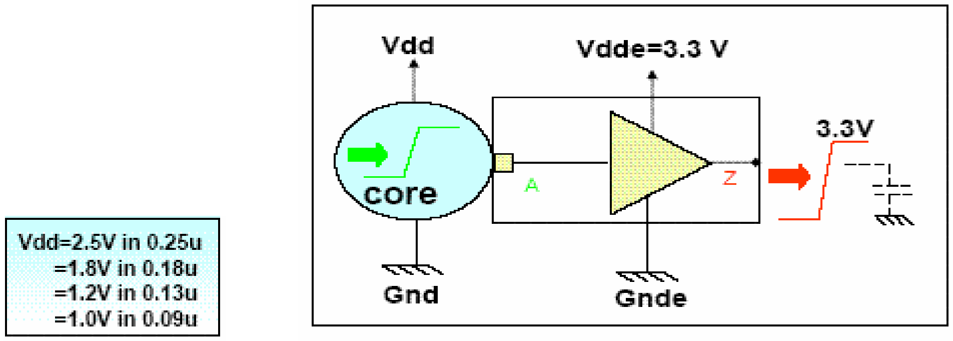
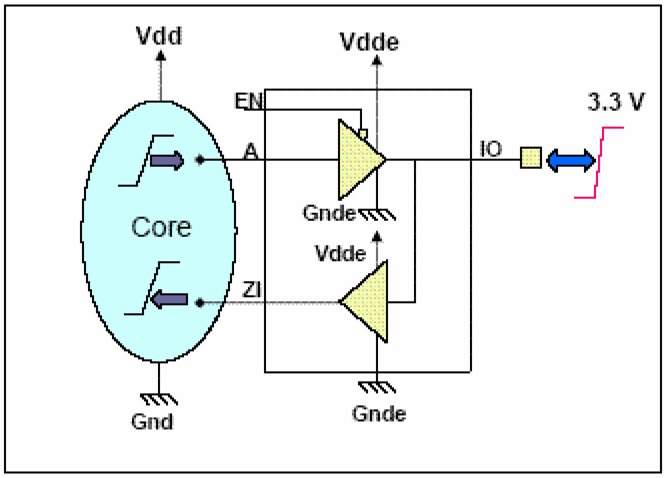


Fig 2.13 Block representation of output path

C). *Bidirectional:* IO can both work as input or output path. As shown in figure 3.5 this block has both input and output path. These blocks are selected from control signal from the core. In figure EN signal shown is active low. When EN is high block acts as input path, therefore ZI follows IO. And when EN is low this block acts as output path.

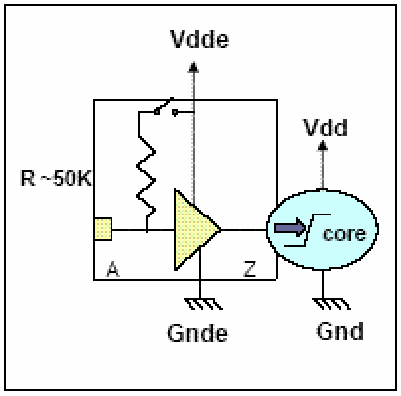


A	EN	IO (Input mode)	IO (output mode)	ZI
-	1	0	0	0
-	1	1	1	1
-	1	Z	Z	X
0	0	Z	0	0
1	0	Z	1	1
0	0	0	0	0
0	0	1	X	X
1	0	0	X	X
1	0	1	1	1

Fig 2.14 Block representation of bidirectional block

2.8.1.2 Specific functions of IO.

A). *Pull-up and pull-down:* The optional pull-up and pull-down resistors are intended to establish High and Low levels, respectively, at unused I/Os. The pull-up resistor optionally connects each IOB pad to VCCO. A pull-down resistor optionally connects each pad to GND. These resistors are placed in a design using the PULLUP and PULLDOWN symbols in a schematic, respectively. They can also be instantiated as components, set as constraints or passed as attributes in HDL code.
If there is no signal on the bus, a resistive load pulls up or pulls-down the bus. In case of pull-up when there is no signal present at input, 'A' gets pulled up to Vdde so output becomes high. Same is the case with pull-down load; it pulls down the signal to ground if there is no signal present.



A	Z
0	0
1	1
Z	1

A	Z
0	0
1	1
Z	0

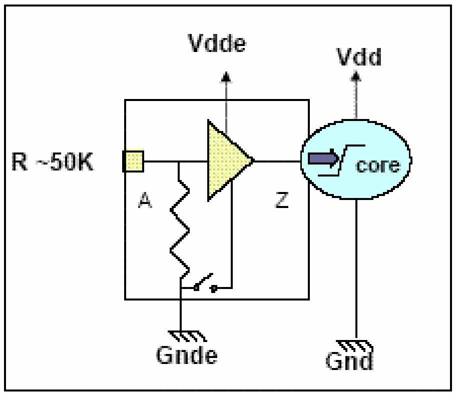
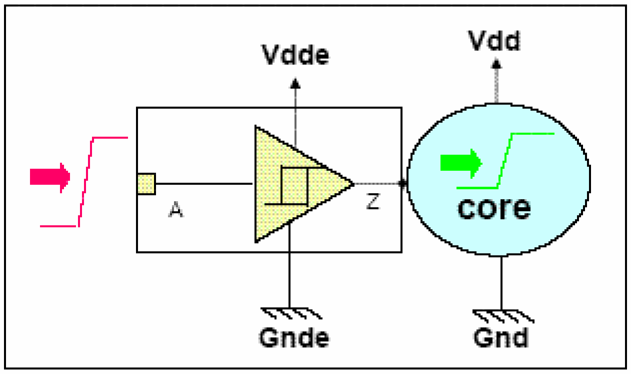


Fig 2.15 Block diagram representing pull-up and pull-down

B). *Hysteresis*: This feature is included, make core more immune to noise. There are two level of voltage known as trim points. Let us assume output voltage is low at the moment, slowly if we increase the input voltage. Without hysteresis, the output would have gone high as input would have crossed the standard trim point. But with hysteresis, output goes high when input crosses 0 to 1 trim point. Now if some noise interferes with input signal and input voltage fluctuates as long as input voltage does not go below 1 to 0 trim point output will remain high. This phenomenon is graphed in figure 3.8. The loop formed is known as hysteresis loop.



A	Z
0	0
1	1
Z	X

Fig 2.16 Block representation of hysteresis block

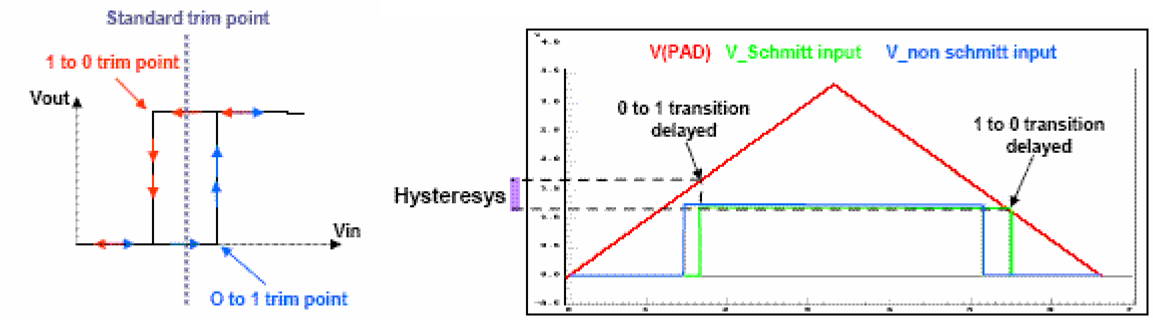


Fig 2.17 Hysteresis loop

C). *<u>Analog IO's</u>*: These IO's are designed to carry analog signal e.g. oscillators, ADC. They have different current and voltage requirements thus have different design.

D). *<u>Power Supply IO's</u>*: These IO's are designed to carry supply for core and IO. They have inbuilt ESD protection circuitry.

2.8.2 Requirements of IO

All IC or programmable ASIC/FPGA contain some type of input/output cell (I/O cell). These I/O cells handle driving logic signals off-chip, receiving and conditioning external inputs, as well as handling such things as electrostatic protection.

The following are different types of I/O requirements.

- <u>DC output</u>. Driving a resistive load at DC or low frequency (less than 1 MHz). Example loads are light-emitting diodes (LEDs), relays, small motors, and such. Can we supply an output signal with enough voltage, current, power, or energy?
- <u>AC output</u>. Driving a capacitive load with a high-speed (greater than 1 MHz) logic signals off-chip. Example loads are other logic chips, a data or address bus, ribbon cable. Can we supply a valid signal fast enough?
- <u>DC input</u>. Example sources are a switch, sensor, or another logic chip. Can we correctly interpret the digital value of the input?
- <u>AC input</u>. Example sources are high-speed logic signals (higher than 1 MHz) from another chip. Can we correctly interpret the input quickly enough?

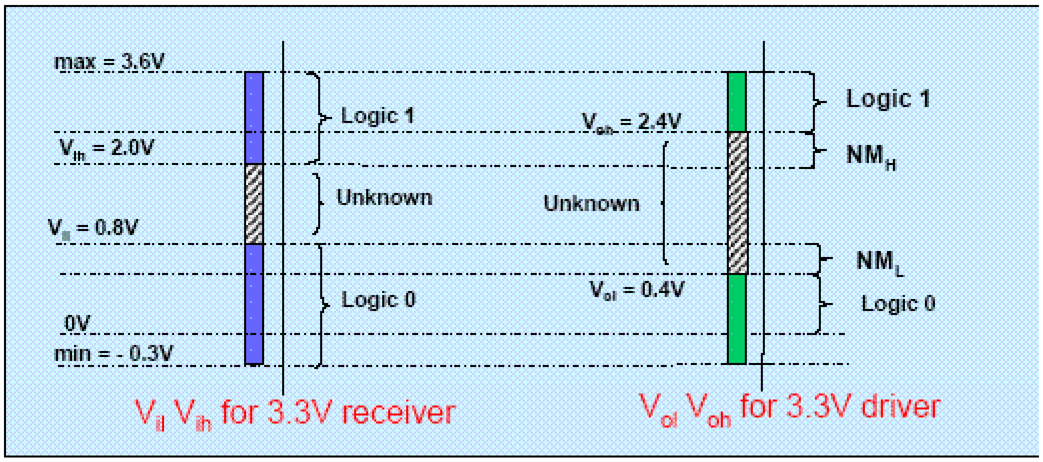
- Clock input. Examples are system clocks or signals on a synchronous bus. Can we transfer the timing information from the input to the appropriate places on the chip correctly and quickly enough?
- Power input. We need to supply power to the I/O cells and the logic in the core, without introducing voltage drops or noise. We may also need a separate power supply to program the chip.

2.8.3 Electrical Interface Characteristics of IOB.

The following parameters characterizes electrical interface of IOB:

2.8.3.1 Input and output voltage level.

VIL: It is the value of voltage which is taken as logic ‘0’ at the input.
VIH: It is the value of voltage which is taken as logic ‘1’ at the input.
VOL: It is the value of voltage which is taken as logic ‘0’ at the output.
VOH: It is the value of voltage which is taken as logic ‘1’ at the output.

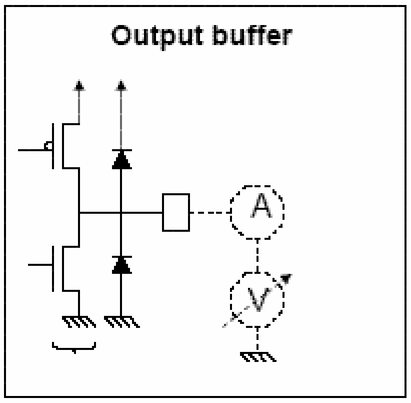


V_{ih}	minimum input voltage univocally recognized as logic 1
V_{il}	maximum input voltage univocally recognized as logic 0
V_{ol}	maximum output voltage when the expected value is logic 0
V_{oh}	minimum output voltage when the expected value is logic 1
NM_H	$V_{oh} - V_{ih}$, Noise margin to the logic 1
NM_L	$V_{oh} - V_{ih}$, Noise margin to the logic 0

Fig 2.18 Graphical representation of voltage levels

2.8.3.2 Input and output current level.

IIL: It is the value of injected current on logic state ‘0’ to reach VIL value.
IIH: It is the value of injected current on logic state ‘1’ to reach VIH value.
IOL: It is the value of injected current on logic state ‘0’ to reach VOL value.
IOH: It is the value of injected current on logic state ‘1’ to reach VOH value.
Iil, Iih, Iol and Ioh are not specification criteria in standard CMOS I/O.
These parameters represent the buffer drive capability used to reach Vil, Vih, Vol and Voh criteria.



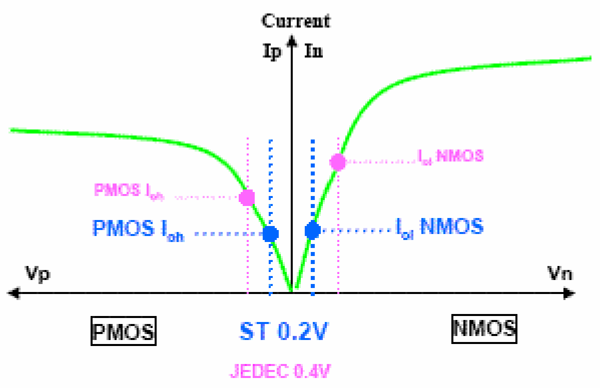


Fig 2.19 Current level

2.8.3.3 Rise, fall time and Propagation delay.

Rising Time (TR): Output switching time from the 10% to the 90% of the IO supply.
Falling Time (TF): Output switching time from the 90% to the 10% of the IO supply.
Rising Propagation Time (TPR): Delay time between the Input, calculated at the 50% of the core supply value, and the Output calculated at the 50% of the IO supply value.
Falling Propagation Time (TPF): Delay time between the Output, calculated at the 50% of the core supply value, and the Output, calculated at the 50% of the IO supply value.

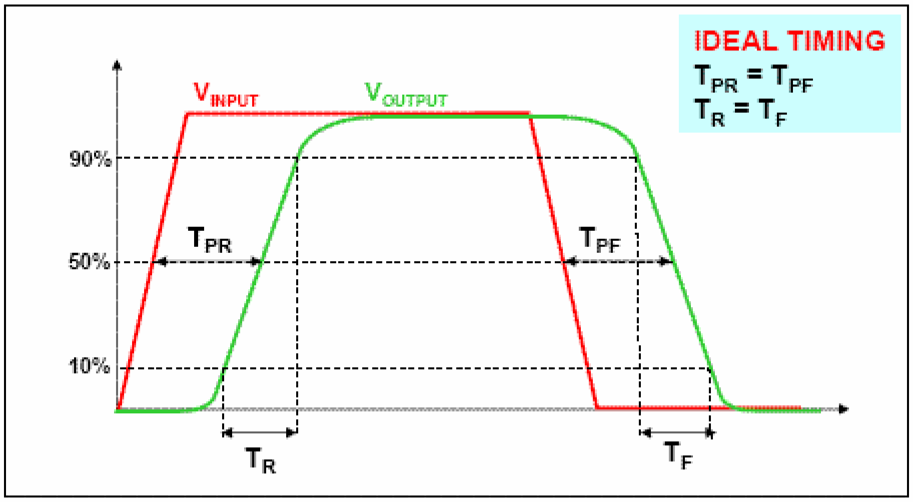


Fig 2.20 Waveform depicting rise, fall time and propagation delay

Ideally TPR should be equal to TPF and TR to TF. If they are not equal then output waveform distorts. For example if TPR< TPF, then duty cycle of output tend to increase with input frequency. And if TPR>TPF, then duty cycle of output decrease with increase in input frequency. A relationship depicting above mentioned phenomenon has been derived and included in the appendix.

2.8.3.4 Slew rate.

IMAX: Peak current on the IO supply

SRON: Switch on slew rate during capacitance charge, measured between 25% and 50% of the positive supply current peak. Positive

SROFF: Switch off slew rate during capacitance charge, measured between the 75% and 50% of the positive supply current peak. Negative

SFON: Switch on slew rate during capacitance charge, measured between 25% and 50% of the negative ground current peak. Negative

SFOFF: Switch off slew rate during capacitance charge, measured between 75% and 50% of the negative ground current peak. Positive

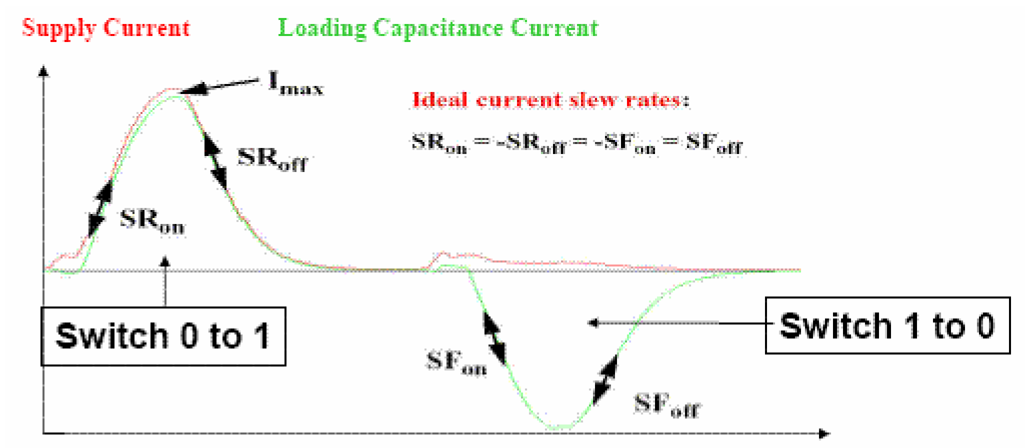


Fig 2.21 Definition of slew rate

2.8.4 Basic blocks of IOB

The Input/Output Block (IOB) provides a programmable, bidirectional interface between an I/O pin and the FPGA's internal logic. A simplified diagram of the IOB's internal structure appears in Figure 3.11. There are three main signal paths within the IOB: the output path, input path, and 3-state path and an ESD protection circuitry. Each path has its own pair of storage elements that can act as either registers or latches. The three main signal paths are as follows:

- *The input path* carries data from the pad, which is bonded to a package pin, through an optional programmable delay element directly to the I line. The IOB outputs lead to the FPGA's internal logic. The delay element can be set to ensure a hold time of zero.

- *The output path*, starting with the O1 and O2 lines, carries data from the FPGA's internal logic through a multiplexer and then a three-state driver to the IOB pad. In addition to this direct path, the multiplexer provides the option to insert a pair of storage elements.

- *Tri-state*: The 3-state path determines when the output driver is high impedance. The T1 and T2 lines carry data from the FPGA's internal logic through a multiplexer to the output driver. In addition to this direct path, the multiplexer provides the option to insert a pair of storage elements.

- *Keeper Circuit*: Each I/O has an optional keeper circuit that retains the last logic level on a line after all drivers have been turned off. This is useful to keep bus lines from floating when all connected drivers are in a high-impedance state. This function is placed in a design using the KEEPER symbol. Pull-up and pull-down resistors override the keeper circuit.

- *ESD Protection*: Clamp diodes protect all device pads against damage from Electro-Static Discharge (ESD) as well as excessive voltage transients. Each I/O has two clamp diodes: One diode extends P-to-N from the pad to VCCO and a second diode extends N-to-P from the pad to GND. During operation, these diodes are normally biased in the off state. These clamp diodes are always connected to the pad, regardless of the signal standard selected. The presence of diodes limits the ability of I/Os to tolerate high signal voltages.

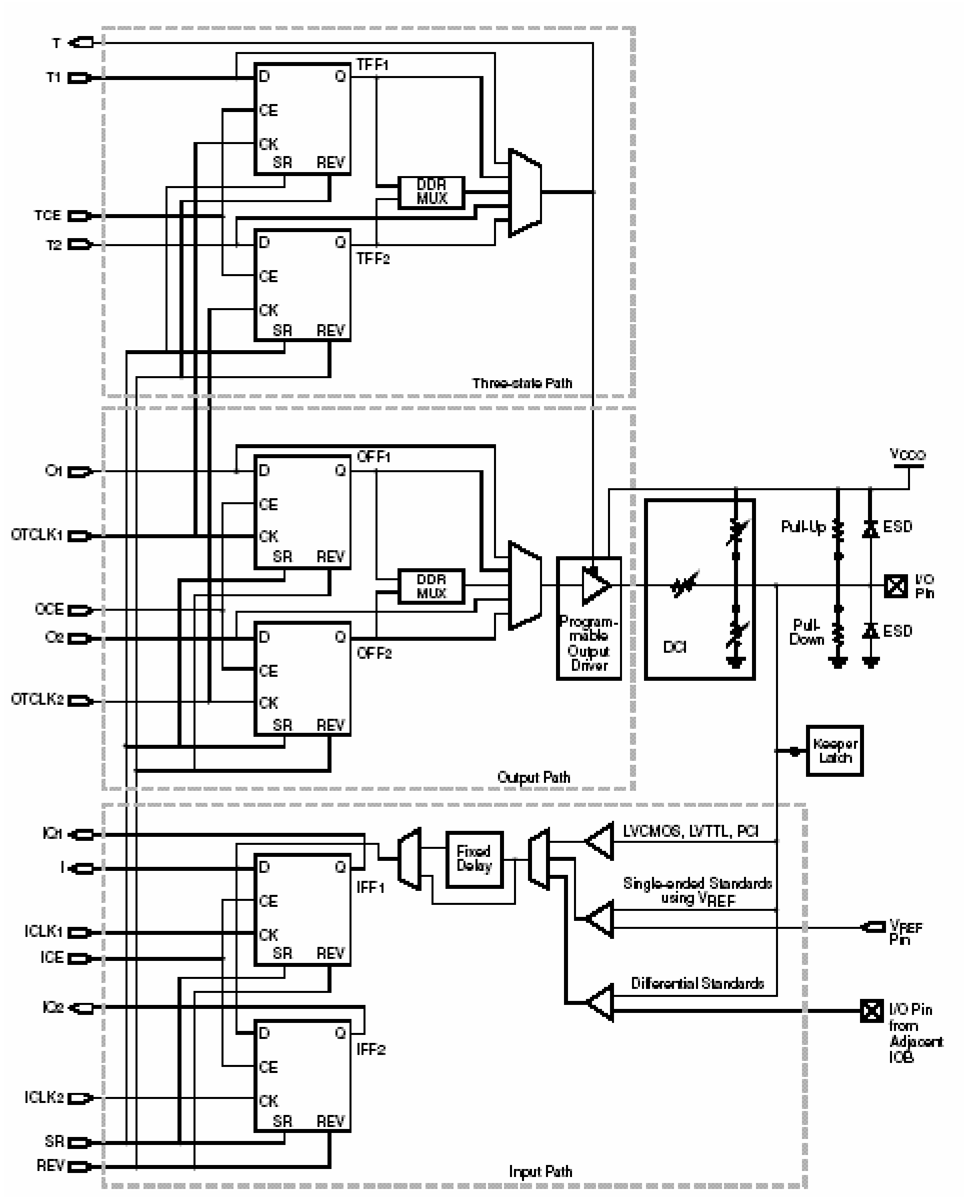


Fig 2.22 Simplified diagram of Xilinx Spartan FPGA IOB

- *Digitally Controlled Impedance (DCI):* When the round-trip delay of an output signal — i.e., from output to input and back again — exceeds rise and fall times, it is common practice to add termination resistors to the line carrying the signal. These resistors effectively match the impedance of a device's I/O to the characteristic impedance of the transmission line, thereby preventing reflections that adversely affect signal integrity. However, with the high I/O counts supported by modern devices, adding resistors requires significantly more components and board area. Furthermore, for some packages — e.g., ball grid arrays — it may not always be possible to place resistors close to pins. DCI answers these concerns by providing two kinds of on-chip terminations: Parallel terminations make use of an integrated resistor network. Series terminations result from controlling the impedance of output drivers. DCI actively adjusts both parallel and series terminations to accurately match the characteristic impedance of the transmission line. This adjustment process compensates for differences in I/O impedance that can result from normal variation in the ambient temperature, the supply voltage and the manufacturing process. When the output driver turns off, the series termination, by definition, approaches very high impedance; in contrast, parallel termination resistors remain at the targeted values. DCI is available only for certain I/O standards

2.8.4.1 Input Stage of IOB

The input stage of IOB is of different configuration. They are of three basic types of input stage namely:

A). Single ended: Signal w.r.t. ground is taken, level shifted and amplified and fed to next block. E.g. LVCMOS, LVTTL are single ended standards.

B). Pseudo-differential: In this type of stage signal is compared with respect to reference voltage (Vref). VIL and VIH are defined with respect to Vref voltage. E.g. for SSTL18,
VIL=Vref – 0.25 &
VIH=Vref + 0.25.

C). Differential I/O: There is two signal lines carrying data. As shown in figure one signal is Vpad while other is inverted value of Vpad. Differential amplifier is used to amplify two input signals. In differential standards, electrical interface is defined by common mode voltage (VICM) and differential voltage (VID) specification.

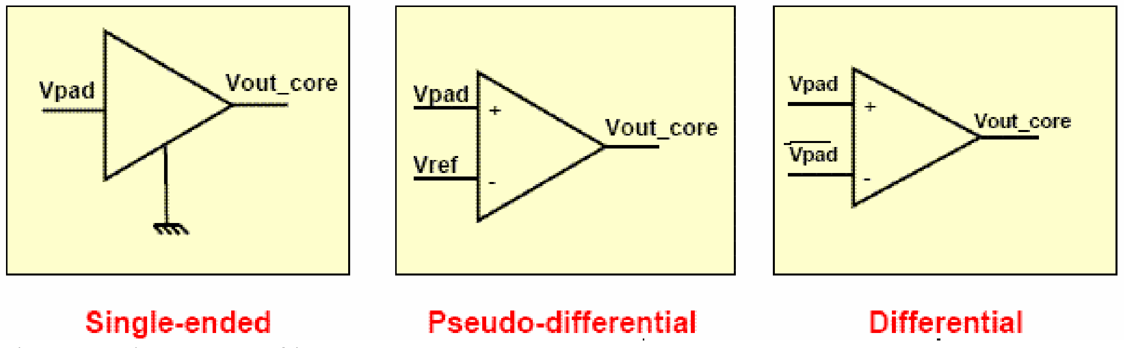


Fig 2.23 Three type of input stages.

2.8.4.2 Output stage of IOB

First and foremost, an output pad must have sufficient drive capability to achieve adequate rise and fall times into a given capacitive load. If the pad drives non-CMOS loads, then they should comply with different DC characteristic.

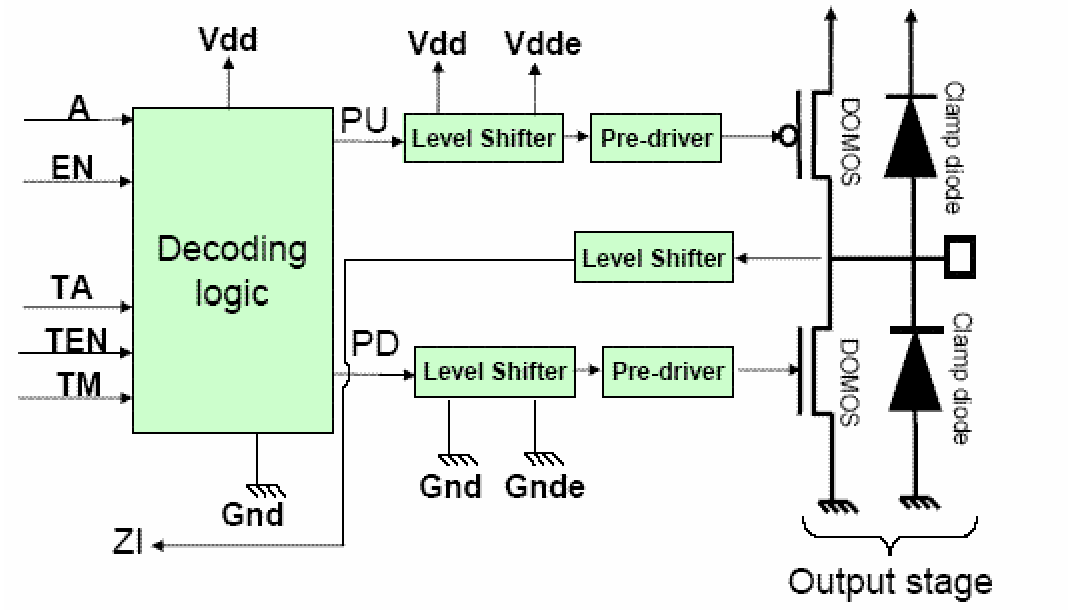


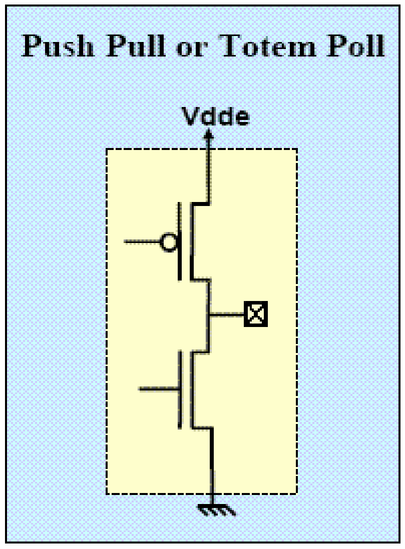
Fig 2.24 an output block

Basically output block contains:

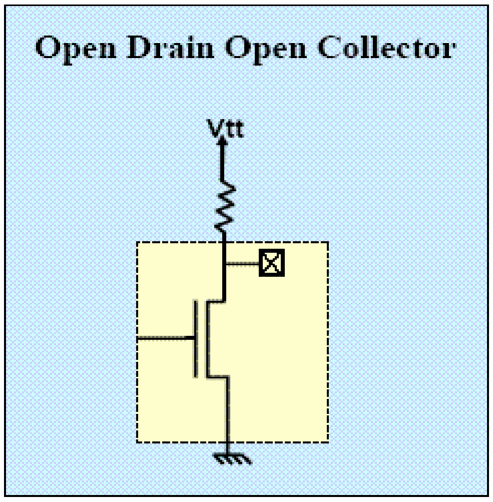
Decoding logic: It is needed to configure output pad in different mode. It may be used to set output pad to a specific standard.

Level Shifter: It converts the signal from core voltage level to the IO voltage level.

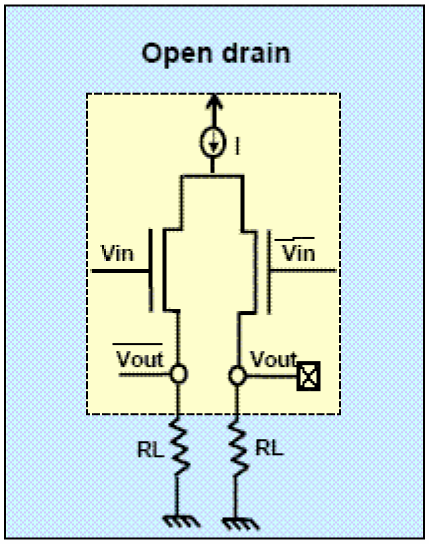
Output Stage: It can also be referred as driver stage. It drives the load it should comply to rise, fall time and current requirement of the load. Output stage can be of different types.

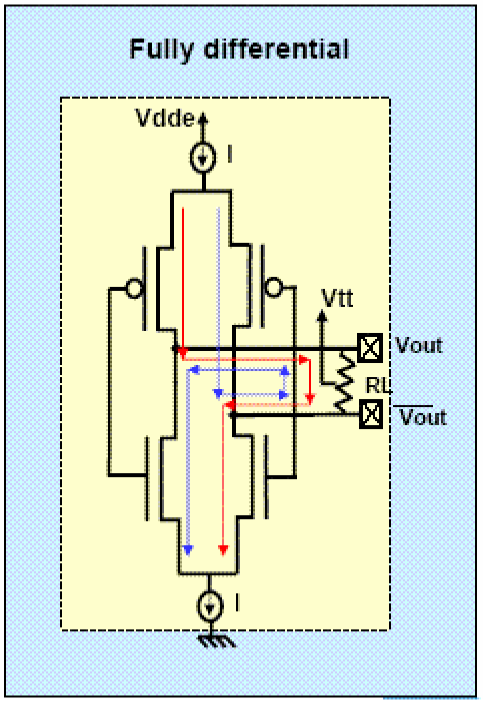


(a)



(b)





(c) **(d)**

Fig 2.25 Typical output stages

2.9 ESD Protection in I/O

An integrated circuit (IC) connected to external ports is susceptible to damaging electrostatic discharge (ESD) pulses from the operating environment and peripherals. The same ever-shrinking IC process technology that enables such high-port interconnects data rates can also suffer from higher ESD susceptibility because of its smaller fabrication geometry. Additional external protection devices can violate stringent signalling requirements, leaving design engineers with the need to balance performance and reliability.

2.9.1 What is ESD?

An ESD event is the transfer of energy between two bodies at different electrostatic potentials, either through contact or via an ionized ambient discharge (a spark). This transfer has been modelled in various standard circuit models for testing the compliance of device targets.

Static electricity is defined as an electrical charge caused by an imbalance of electrons on the surface of a material. This imbalance of electrons produces an electric field that can be measured and that can influence other objects at a distance. *Electrostatic discharge* is defined as the transfer of charge between bodies at different electrical potentials.

Creating electrostatic charge by contact and separation of materials is known as "triboelectric charging." It involves the transfer of electrons between materials. The atoms of a material with no static charge have an equal number of positive (+) protons in their nucleus and negative (-) electrons orbiting the nucleus.

In Figure 3.13, Material "A" consists of atoms with equal numbers of protons and electrons. Material B also consists of atoms with equal (though perhaps different) numbers of protons and electrons. Both materials are electrically neutral.

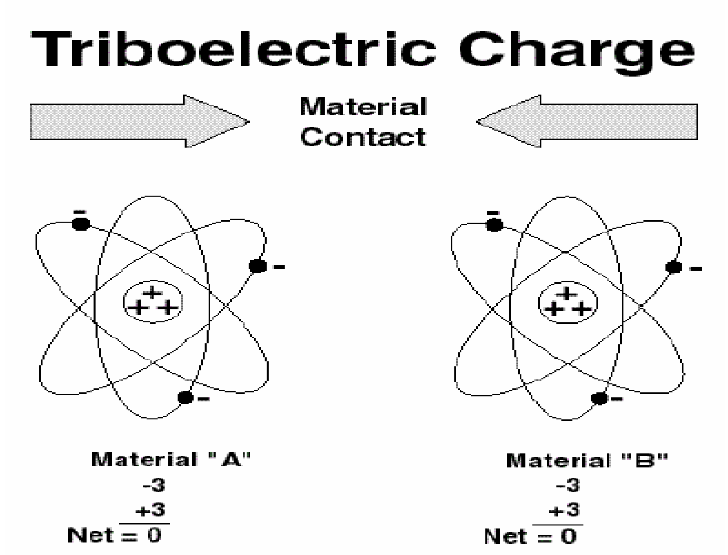


Fig 2.26 The Triboelectric Charge, Materials Make Intimate Contact

When the two materials are placed in contact and then separated, negatively charged electrons are transferred from the surface of one material to the surface of the other material. Which material loses electrons and which gains electrons will depend on the nature of the two materials. The material that loses electrons becomes positively charged, while the material that gains electrons is
negatively charged. This is shown in Figure3.14.

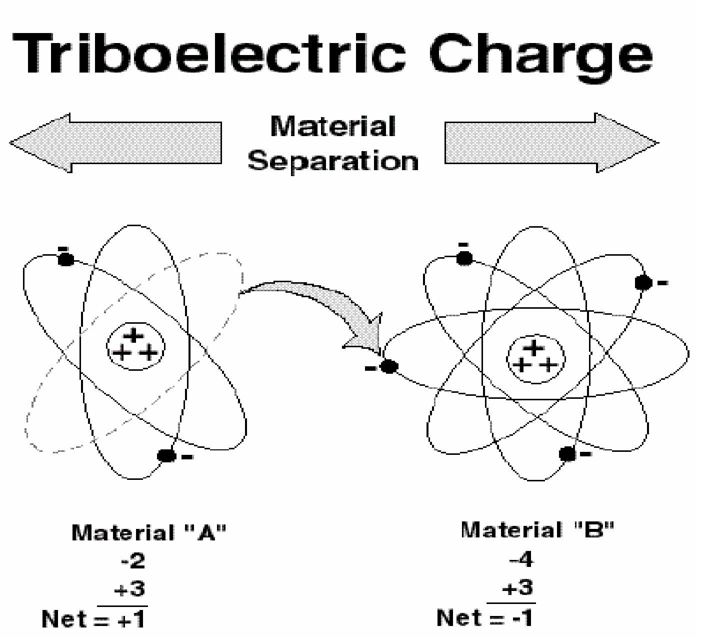
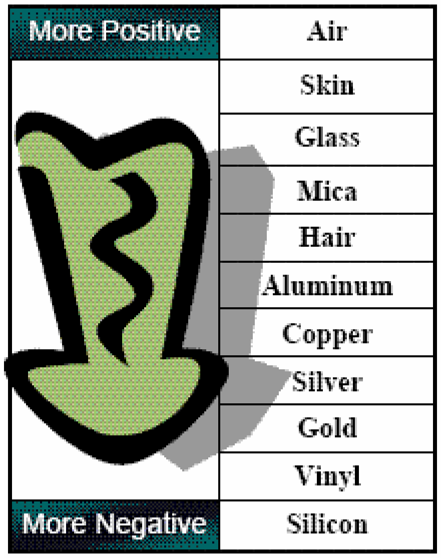


Fig 2.27 The Triboelectric Charge - Separation

The position inside the table points out charge polarity and amplitude.

- The first material takes positive charge and the other a negative one.
- Two materials far away from each other generate a stronger charge than the one generated when they are close.



More Positive	Air
	Skin
	Glass
	Mica
	Hair
	Aluminum
	Copper
	Silver
	Gold
	Vinyl
More Negative	Silicon

2.9.2 Effects of ESD

- Electrostatic discharge can change the electrical characteristics of a semiconductor device, degrading or destroying it.
- Electrostatic discharge also may upset the normal operation of an electronic system, causing equipment malfunction or failure.
- In clean rooms charged surfaces can attract and hold contaminants, making removal from the environment difficult. When attracted to the surface of a silicon wafer or a device's electrical circuitry, these particulates can cause random wafer defects and reduce product yields.
- Induces latch-up in device. It triggers the regenerative action of current amplification in parasitic transistor.

2.9.3 ESD standard models

Within the semiconductors world, there are three principle sources of electrostatic discharges. Three standards were developed to model these events.

- Human Body Model (HBD).
- Machine Model (MM).
- Charged Device Model (CDM).

<u>Human Body Model (HBD)</u>

- Represents the discharge of a standing people through a pointing finger.
- Ability to reproduce the field failures caused by human handling.
- Oldest model often considered as the ESD model.
- Discharge occurs in very short duration (100ns), current ranging from 1A to 4A.

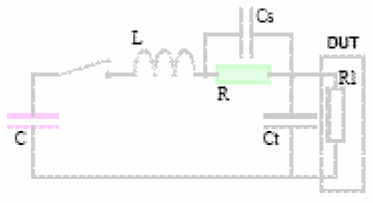


Fig 2.28 Human Body Model

International standards distinguish different classes depending on the last stress level passed.

Classes	Stress level (V)
0	<250
1A	250<X<500
1B	500<X<1000
1C	1000<X<2000
2	2000<X<4000
3A	4000<X<8000
3B	>8000

Table 2.2 Class level of HBD Model

<u>Machine Model (MM)</u>

- Represents the discharge of improper grounding equipment through the IC when it is picked up for placement in the socket.
- Ability to reproduce the field failures caused by machine handling
- Parasitic elements easily detected.
- Shorter duration, higher current than HBM.

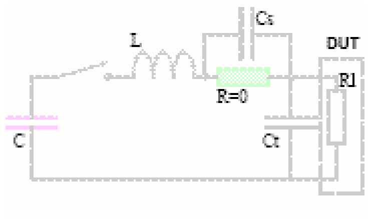


Fig 2.29 Equivalent circuit of MM

International standards distinguish different classes depending on the last stress level passed.

Classes	Stress level (V)
M1	<100
M2	100<X<200
M3	200<X<400
M4	400<X

Table 2.3 Classes for Machine Model

<u>Charged Device Model (CDM)</u>

- Represents the discharge of a charged device to ground through a single pin device.
- Ability to reproduce the real world ESD events.
- Parasitic components and environmental conditions play a very important role.
- Very short duration (10ns, very high current (10A and more).

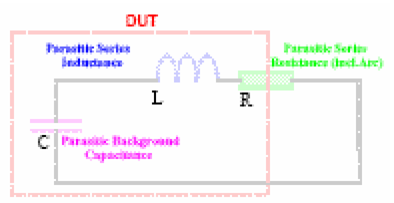
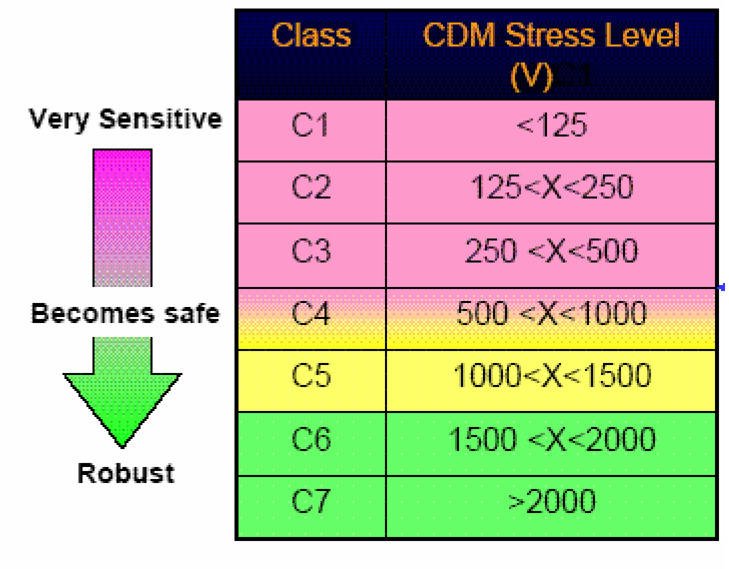

DUT
Parasitic Series Inductance
Parasitic Series Resistance (Incl.Arc)
L
R
C
Parasitic Background Capacitance

Fig 2.30 Equivalent of CDM

A classification is given to determine if devices are sensitive or not to ESD
and to help for defining a specification.


Very Sensitive
Becomes safe
Robust

Class	CDM Stress Level (V)
C1	<125
C2	125<X<250
C3	250 <X<500
C4	500 <X<1000
C5	1000<X<1500
C6	1500 <X<2000
C7	>2000

Table 2.4 Class level for Capacitor Device Model

2.9.4 What to protect in silicon?

- All the pins (inputs, outputs, powers) of a circuit are directly connected to the real world as a result they might be destroyed by a discharge.
- Some part of the circuit are more sensitive than others:
  - CMOS inputs: avoid gate oxide failure.
  - Gate outputs: Avoid the drain/substrate junction breakdown of the MOS.
  - Bipolar circuits: Avoid base-emitter junction breakdown.

2.9.5 High Speed ESD Protection

As IC manufacturers have achieved higher frequencies of input/output (I/O) interconnects, they have continued to decrease the minimum dimensions of the transistors, interconnections, and the silicon dioxide (SiO_2) insulation layers in their devices. This decrease results in smaller structures for higher-speed devices that are more susceptible to breakdown damage at lower energy levels. SiO_2 layers are more likely to rupture, and metal traces are more likely to open or bridge during an ESD event.

2.9.5.1 Protection strategy

- Protect the circuit without impacting its functionality.
- Protection devices are placed in parallel. They act like a switch: open during normal operation and closed during ESD event.

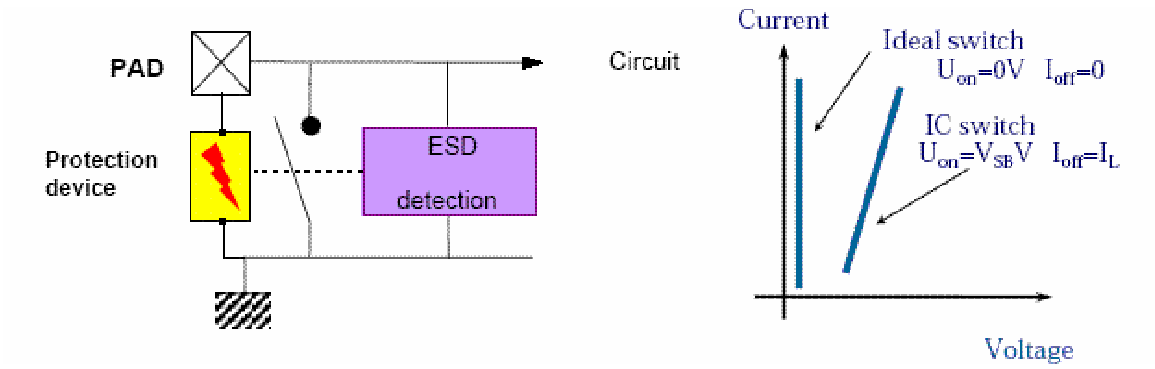


Fig 2.31 Basic ESD protection circuitry

2.9.5.2 ESD Protection Devices.

Basic protection strategy is to put the protective device in parallel to core as shown in figure 3.26

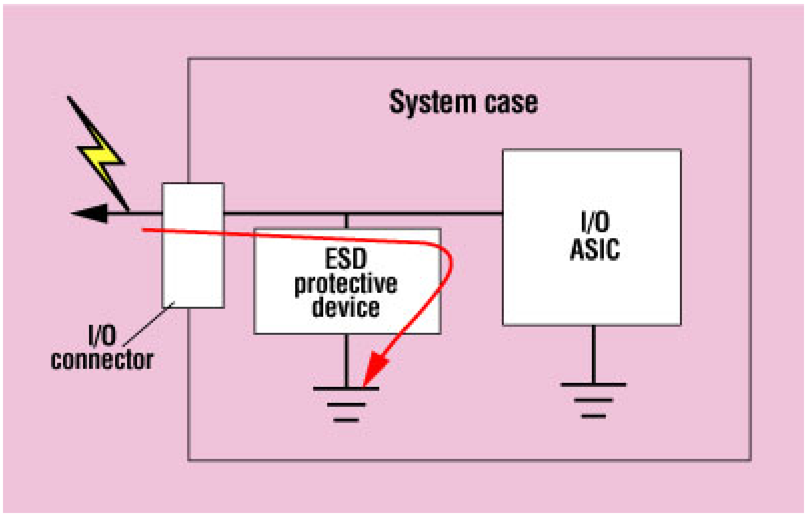


Figure 2.32. ESD protection devices attempt to divert a potentially damaging charge away from sensitive circuitry and protect the system from permanent damage.

A variety of technologies are used in devices for ESD protection. These are discussed below.

Zener Diodes

One traditional device, the zener diode, is generally poorly suited for very high-speed I/O interfaces because the lowest capacitance of existing devices is about 30 pF (shown as a parasitic capacitor in Figure 3.27). This capacitance is too high to pass high-frequency signals without significant distortion. This distortion results in unreliable detection of the signals and increased high-frequency roll-off. Zener diodes could be made with lower capacitances, but this would result in ESD voltages insufficient to meet the 6–8-kV protection levels necessary.

TVS Diodes

There are some TVS devices on the market that add a regular diode in series with the zener diode to effectively lower the net capacitance. To handle positive- and negative-polarity ESD pulses, a second zener and series diode pair (in the opposite polarity) must be placed in parallel with the first pair of diodes. Unfortunately, the resulting capacitance of 5–6 pF is still not low enough to avoid distortion of high-speed I/O signals.

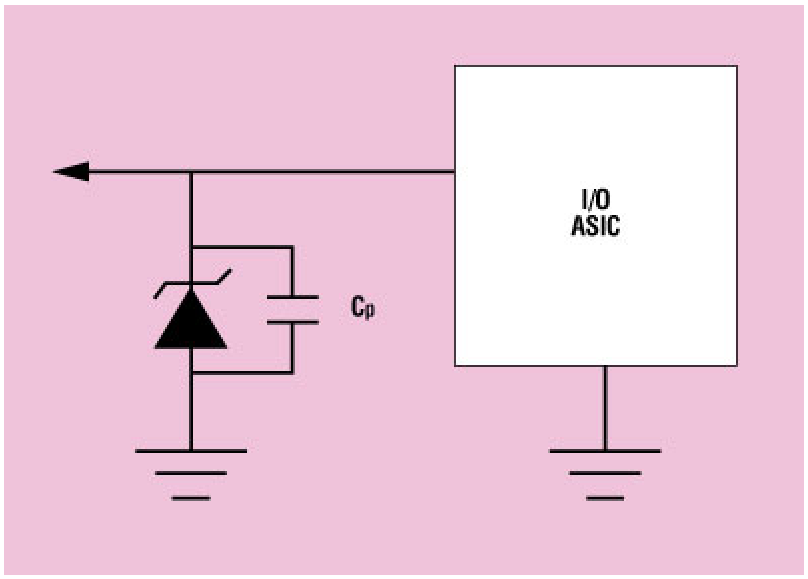


Figure 2.33. A parasitic capacitor here is too high to pass high-frequency signals without significant distortion.

MOVs

MOVs can achieve slightly lower capacitances than TVS devices, but currently the lowest-capacitance MOV device available has a capacitance of 3 pF, which can still exceed the allowable load on high-speed interconnects.

Dual-Rail Clamp Diodes

Zener diode capacitances are high because their structures must be sufficiently robust to tolerate reverse breakdown phenomena. To eliminate the need for the zener's breakdown, regular diodes can be used to clamp the ESD pulses to the power or ground rail. Using this solution, the current flow is always in the diode's forward direction, as shown in Figure 3.28. This setup allows the use of smaller, and therefore lower, capacitance diodes. Positive ESD pulses are clamped to the positive supply rail, and negative ESD pulses are clamped to ground. (The system-bypass capacitors and power supply are responsible for shunting this extra energy on the positive rail back to ground. This can sometimes be aided by also adding a local zener diode, which does not affect the signal load.).

However, this capacitance is relatively high, and an examination of the data sheets reveals that they were not designed for high-current ESD pulses. These diodes have no specifications that guarantee their use with the high current and voltages of ESD pulses or with repetitive high-current ESD pulses. Some will degrade and eventually fail at high ESD voltage and currents.

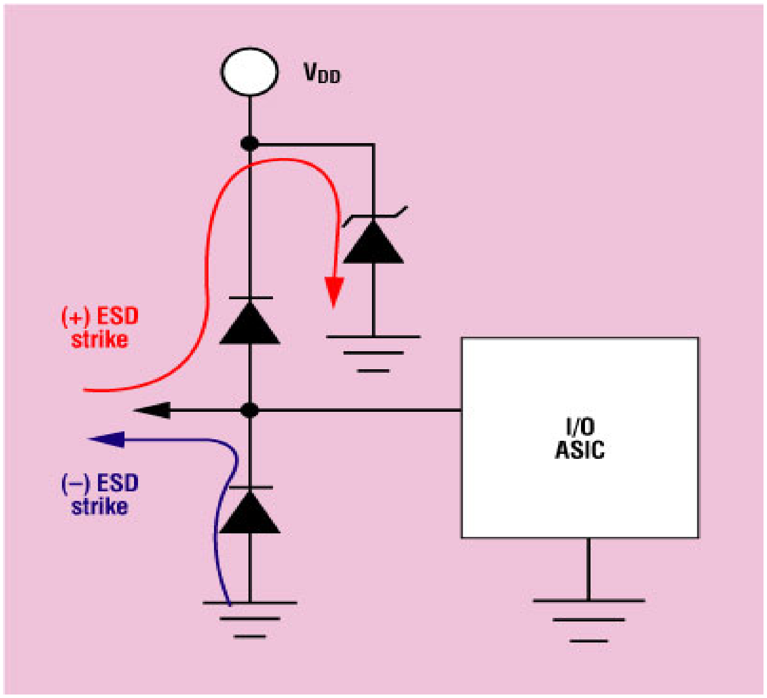


Figure 2.34 Regular diodes can be used to clamp the ESD pulses to the power or ground rail so the current flow is always in the diode's forward direction.

Polymer Devices

The polymer devices symbolized in Figure 3.28 have resistances that are voltage dependent. With a low voltage (e.g., 5 V), the impedance is in the Giga ohm realm. When a high voltage is applied across the polymer device, the resistance drops to a very low value, so that tens of amperes can be shunted to ground. What makes these polymers attractive for high-frequency applications is their sub-pico farad capacitance (0.05–1.0 pF). This low capacitance, however, comes with some not-so-attractive side effects. Unlike zener diodes that break down at the same voltage that they clamp to, a polymer device does not break down until it reaches a voltage that is much higher than the final clamping voltage. A typical polymeric ESD device does not break down until as much as 1000 V is reached. Then it snaps back to a clamping voltage of up to 150 V. After the charge is dissipated, the polymer returns to its high-impedance state.

Consequently, polymer devices can be used only in applications in which the ICs that are supposed to be protected must have their own built-in ESD protection that can tolerate the breakdown or trigger voltage of the polymer device (trigger voltages vary from 300 to 1000 V; clamping voltages vary from 60 to 150 V). These devices can be difficult to accurately characterize in manufacturing, so their data sheets often contain only typical specifications without guaranteed minimums and maximums.

Metal Oxide Silicon (MOS) Devices

A new technology uses a dual-rail clamp configuration as shown in Figure 3.28. The process technology to make the diodes, however, is fundamentally different. PicoGuard technology is derived from a MOS process that is optimized for minimum capacitance. Traditional diode structures are derived from simple bipolar technologies and tend to have higher capacitance levels. The new technology is the first to combine low capacitance with low-voltage clamping levels and high ESD tolerance.

These diodes provide ESD protection beyond IEC 61000-4-2 (±8-kV-and-above contact) with a capacitance of <1.3 pF maximum (~1.0 pF typical). They have a low insertion loss (virtually zero up to 3 GHz) and a clamping voltage below 15 V (V_{CC} +10 V, ground –10 V) with no higher trigger voltages. Other specifications include a sub-nanosecond response time, durability of more than 1000 ESD pulses, and a leakage current of 1.0 µA.

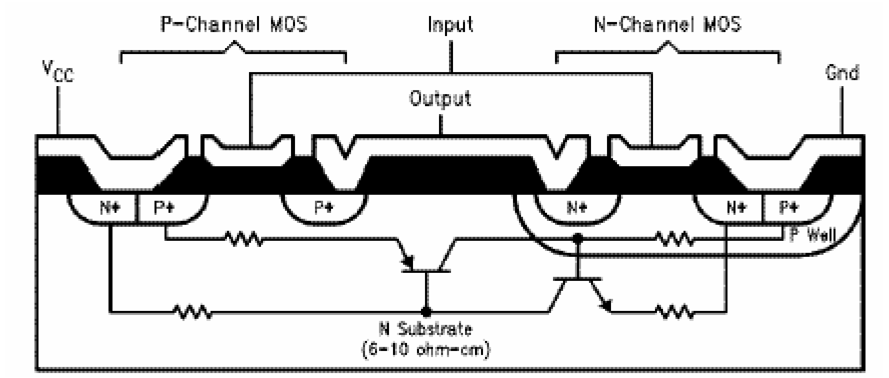
2.10 Latch up

Latch-up is a mechanism establishing a low resistance path between VDD and VSS. Very high current flows through the circuit, device doesn't work properly or it is destroyed. Latch-up activation causes must be characterized in order to determine some rules to prevent it.

Latch-up is a failure mechanism of CMOS integrated circuits characterized by excessive current drain coupled with functional failure, parametric failure and/or device destruction. It may be a temporary condition that terminates upon removal of the exciting stimulus, a catastrophic condition that requires the shutdown of the system to clear or a fatal condition that requires replacement of damaged parts. Regardless of the severity of the condition, latch-up is an undesirable but controllable phenomenon. In many cases, latch-up is avoidable.

2.10.1 Latch-up Model

The cause of the latch-up exists in all junction-isolated or bulk CMOS processes: parasitic PNPN paths. Figure 3.29, a basic N-substrate CMOS cross section, shows the parasitic NPN and PNP bipolar transistors which most frequently participate in latch-up. The P+ sources and drains of the P channel MOS devices act as the emitters (and sometimes collectors) of lateral PNP devices; the N-substrate is the base of this device and collector of a vertical NPN device. The P-well acts as the collector of the PNP and the base of the NPN. Finally, the N+ sources and drains of the N-channel MOS devices serve as the emitter of the NPN. The substrate is normally connected to VCC, the most positive circuit voltage, via an N+ diffusion tap while the P-well is terminated at GND, the most negative circuit voltage, through a P+ diffusion. These power supply connections involve bulk or spreading resistance to all points of the substrate and P-well.



.

Fig 2.35 Basic CMOS inverter cross-section with Latch-up circuit model

2.10.2 Latch-up Phenomenon

Normally, only a small leakage current flows between the substrate and P-well causing only a minute bias to be built up across the bulk due to the resistivity of the material. In this case the depletion layer formed around the reverse biased PN junction between P-well and the substrate supports the majority of the VCC-GND voltage drop. As long as the MOS source and drain junctions remain reverse biased, CMOS is well behaved.

In the presence of intense ionizing radiation, thermal or over-voltage stress, however, current can be injected into the PNP emitter-base junction, forward-biasing it and causing current to flow through the substrate and into the P-well. At this point, the NPN device turns on, increasing the base drive to the PNP. The circuit next enters a regenerative phase and begins to draw significant current from the external network thus causing most of the undesirable consequences of latch-up. Once established, a latch-up site, through the fields generated by the currents being conducted, may trigger similar action in both elements of the IC.

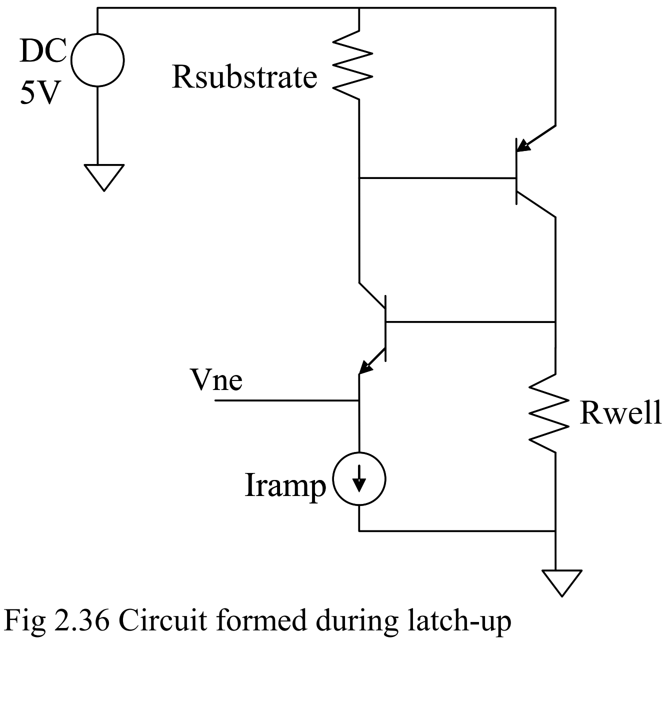


Fig 2.36 Circuit formed during latch-up

2.10.3 Preventing Latch-up

In the loop of positive current feedback formed by the parasitic PNP and NPN transistors of the latch-up structures, regenerative switching may result if sufficient loop gain is available. One must remember, though, that three conditions are necessary for latch-up to occur.

1. Both parasitic bipolars must be biased into the active state.
2. The product of the parasitic bipolar transistor current gains (Bnpn•Bpnp) must be sufficient to allow regeneration, i.e., greater than or equal to one;
3. The terminal network must be capable of supplying a current greater than the holding current required by the PNPN path. In processes utilizing epitaxial silicon, this current is usually in excess of 1A.

If any of these conditions is not met both during the initiation and in the steady state, then the latch-up condition is either non-sustaining or cannot be initiated. If the current to the latched structure is not limited, permanent damage may result.

2.10.3.1 Causes of latch-up

- Capacitive effect during the switching.
- Current injection on the output.
- Over-voltages on VDD or Input/Output.
- Avalanche of the Nwell/Substrate junction.
- Punch through between Nwell and N+.

2.10.3.2 Latch-up prevention

Basically by reducing the resistor values and reducing the gain of the parasitic transistors are the basis for eliminating latch up. Latch-up may be eliminated in two basic ways:

- Latch-up resistant CMOS processes.
- Layout Techniques.

Some ways to eliminate latch-up are listed below

- Reduce the current gain
  – NPN emitter and collector separated
  – Reduction of the minor carriers 'life time'
  – Reduction of the emitter efficiency
- Reduce Rnwell and Rpwell
  – Low resistive substrate (HCMOS8D) = latch-up free
  – Epi layer on substrate highly doped
  – Retrograde implants
- Remove the npnp structure
  – Deep trench isolation
  – SOI
  – Architecture with 3 wells

2.10.3.3 I/O Latch-up protection

Most likely place for latch-up to occur is in I/O structures where large currents flow, large parasitic may be present and abnormal circuit voltage may be encountered.
In these structures two options can be taken. The first is to use proven I/O structures designed by the experts who understand the process at detailed level. Second, rules may be applied to the design of these structures that minimize the possibility of latch-up. Typical rules (n-well process) include:

- Physically separate the n- and p- driver transistors (i.e. with the bonding pad).
- Include p+ guard rings connected to Vss around n-transistors. These guard ring acts as a dummy collectors and spoil the gain of the parasitic transistors by collecting minority carriers and preventing them from being injected into the respective bases (shown in fig 3.31).

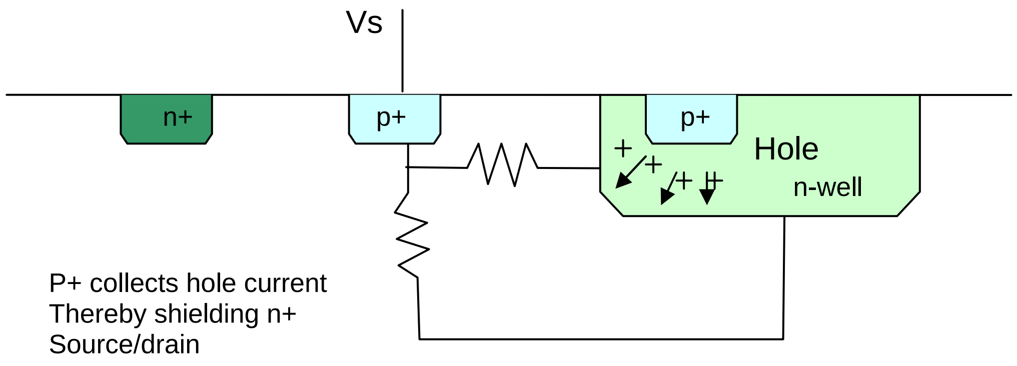


Fig 2.37 Use of dummy collector

- Include n+ guard rings connected to Vdd around p-transistors (figure 3.32).
- Source diffusion regions of the n-transistors should be placed so that they lie along equipotential lines when current flows between Vss and the p-wells: i.e. source fingers should be perpendicular to the dominant direction of current flow rather than parallel to it. This reduces the possibility of latch-up through the n-transistor source, due to an effect called 'field aiding'.

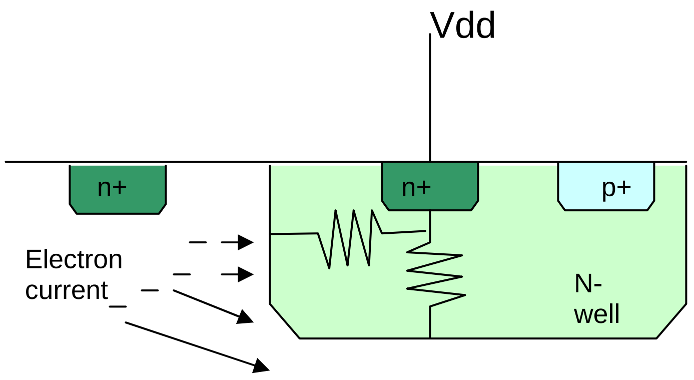


Fig 2.38 use of dummy collector

- Shorting n-transistors source regions to the substrate and the p-transistor regions to the n-well with metallization along their entire lengths will aid in preventing either of the diode from becoming forward biased, and hence reduces the contribution to latch-up from these components.
- The n-well should be hard wired (via n+) to power so that any injected charge is diverted to Vdd via a low resistance path. The n-well has a relatively high sheet-resistance and is susceptible to charge injection.
- The spacing between the n-well n+ and the p-transistor source contact should be kept to a minimum. This allows minority carriers near the parasitic npn-transistor emitter-base junction to be collected, and reduces Rwell. The rules for 1u process suggest one contact for every 10u-50u.
- The separation between the substrate p+ and the n-transistor source contact should be minimized. This result in reduced minority carrier concentration near the npn-emitter –base junction.

2.11 Jedec Standards

JEDEC is the leading developer of standards for the solid-state industry. Almost 2400 participants, appointed by some 270 companies work together in 50 JEDEC committees meet the needs of every segment of the industry, manufacturers and consumers alike. The publications and standards that they generate are accepted throughout the world.

The **JEDEC** Solid State Technology Association (Once known as the **J**oint **E**lectron **D**evice **E**ngineering **C**ouncil), is the semiconductor engineering standardization body of the Electronic Industries Alliance (EIA), a trade association that represents all areas of the electronics industry.

JEDEC was originally created in 1960 as a joint activity between EIA an NEMA, to cover the standardization of discrete semiconductor devices and later expanded in 1970 to include integrated circuits. JEDEC does its work through its 48 committees/ subcommittees that are overseen by the JEDEC Board of Directors. Presently there are about 300 member companies in JEDEC including both manufacturers and users of semiconductor components and others allied to the field

2.11.1 Jedec Activities

JEDEC committees hold frequent meetings throughout the year in domestic and international venues. All standardization work takes place at these meetings and companies must be a member to participate. Surveys are also often taken to find out what companies are doing in important areas that often leads to ballots.

JEDEC members ballot proposals and vote via the JEDEC voting machine prior to meetings. Voting is then reported at the meetings and modifications to the proposals are made in response to comments that are made during balloting.

After a ballot passes and comments are resolved and incorporated, the proposal goes to the JEDEC Board of Directors for final approval before publishing. After BoD approval, JEDEC publishes the standard or publication. Since 1998 JEDEC has provided its Standards and Publications on the JEDEC website at no charge.

2.11.2 Standards supported by IOB

IOB refers to the IOB of FPGA which are configurable. They follow a set of standards for communicating from peripheral devices. These standards are classified in three classes.

- Single ended
- Pseudo-differential
- Fully-differential

2.12 Summary

As the review has been covered of literature, next we discuss the procedure followed to verify the IOB. The details & syntax of the simulators used are provided in appendix.

Chapter 3

Procedure followed for Verification of IOB

3.1 Introduction

Even if a design is working perfectly under certain conditions e.g. voltage value, room temperature, etc, but that design may not behave properly if there is some variation in these conditions. To check whether a design will work circuit is verified by simulation by varying these conditions such as temperature, voltage, process variation parameter.

3.2 Verification

IOB verification is done according to the JEDEC standards. Specifications of the standard are given in previous chapter. Simulation is done for four cases i.e. best case, typical case 1, typical case 2 and worst case. Input path netlist taken is a post lay netlist. Thus according to these cases stimulus is given and parameters are measured on waveform.

3.3 Verification plan for IOB

1. We check the circuit for minimum input swing. Input voltage given to the block has Vil taken as Vil max and Vih taken as Vih min of the standard.
2. Fin is the input frequency at which the simulation is fired. For measurement of basic parameters, simulation is done at frequency at which circuit responds properly. E.g. for LVCMOS standards simulation is done at 50MHz.
3. Transient analysis is done for at least 5-6 cycles.
4. Measurement is done on the 4th cycle of the waveform.
5. First cycle of the waveform may not look ideal in waveform viewer. That cycle should be ignored. It could be because of convergence of the simulator.
6. Waveform should be checked for ringing.
7. Waveform should be consistent with respect to the input waveform.

3.4 Simulation Information

1. Simulation has been carried out for four cases, which are best, typical1, typical2 and worst case.

2. Process variation parameter range -2 to 2,
   Temperature is varied over range -40 to 100,
   Core voltage ranges from 0.9 to 1.1.
   IO Voltage ranges from 1.7 to 1.9.

3. Details of the four cases are tabulated below:

	Best Cases	Typical 1	Typical 2	Worst
Sigma	2	0	0	-2
Temperature (C)	-40	0	25	100
Vdd (V)	Maximum	Typical	Typical	Minimum

4. Value of Vdd1, VIL, and VIH has been taken according to the standard being verified.

5. Slew rate for all the standards have bee taken 1V/ns.

6. Results have been tabulated in four tables for each standard.
   a. Table1 shows the value of vdd1, VIL, VIH and Vref for that IO standard.
   b. Table 2 shows the values measured on the waveform during the simulation.
   c. Table 3 tabulates value of maximum frequency. We will call frequency Fcutoff. After this frequency circuit does not respond to input faithfully.
   d. Table 4 shows the frequency w.r.t. duty cycle.

7. There are three different maximum frequencies referred in the document.
   a. FIL: Frequency Input Limited. FIL denotes maximum frequency input generator can provide for a given slew rate. This is limitation from input generator i.e. not a limitation of the circuit.
   b. Fcutoff: This the frequency up to which the circuit responds to the input properly. As we increase frequency beyond Fcutoff, circuit starts missing some input cycles and does not give consistent output duty cycle. As we go on further increasing the input frequency circuit does not respond to any input.
   c. Fduty: This is the frequency limited by the output duty cycle.

8. For calculating Fduty a relation has been used. Derivation of this relation is provided in appendix.

9. Simulation for Table 1 has been carried out in HSPICE simulator whereas for Table 2, Table3 and Table 4 have been done on ELDO simulator.

10. For standards supported by input buffer Table 3 have not be tabulated. Due to difference in the rise delay and fall delay Fduty for 100% output duty cycle is reached well before Fcutoff. At Fduty for 100% there is consistent ‘1’ at output. Therefore in simulation Fcutoff could not be found out.

11. Transient analysis of 4-6 cycles has been done. Measurement has been done on 4th cycle.

3.5 *.cir* file

```
* INPUT PATH VERIFICATION*
* STANDARD NAME PSEUDO-DIFFERENTIAL*
*******************************************
* OPTION COMMANDS*

.OPTION POST

*******************************************
* INCLUDE FILES *
* library files from fab are included here*
.include 'netlist_file'
.lib '/skew.file' stats
.inc ‘fixed_corner'
.inc 'hspice.param'
```

```
.inc 'HSPICE/models/pfet1.inc'
.inc 'HSPICE/models/nfet1.inc'
.inc ' HSPICE/models/nfet2.inc'
.inc 'HSPICE/models/pfet2.inc'

*****************************************
*GLOBAL SIGNAL DECLARATION *

VGLOBALA GLOBALA GND CORE
VGLOBALB GLOBALB GND 0
VGLOBALC GLOBALC GND 0
VTEST_DATA TEST_DATA GND 0
VSELB_MXA SELB_MXA GND CORE
VEQ14 EQ14 GND 0

******************************************
* BLOCK SELECTION *

* BS2  BS1    BS0 BLOCK SELECTED *
*  0    0      0  INPUT BUFFER
*  0    1      0  DIFF_AMP1
*  1    1      0  DIFF_AMP2
*  1    0      0  DIFF_AMP_SIGNAL_STD

VBS2 BS2 GND 0
VBS1 BS1 GND CORE
VBS0 BS0 GND 0

******************************************
* CHARACTERISATION *

*.ALTER BEST CASE
*.PARAM SIGMA=+2
*.TEMP= -40
*.PARAM CORE=1.1
*.PARAM VIO=1.9
*.PARAM REFVOL=1.1

*.ALTER TYPICAL CASE1
.PARAM SIGMA=0
.TEMP= 0
.PARAM CORE=1.0
.PARAM VIO=1.8
.PARAM REFVOL=0.9

*.ALTER TYPICAL CASE2
*.PARAM SIGMA=0
*.TEMP= 25
*.PARAM CORE=1.0
*.PARAM VIO=1.8
*.PARAM REFVOL=0.9

* .ALTER WORST CASE *
*.PARAM SIGMA=-2
*.TEMP= 100
*.PARAM CORE=0.9
```

```
*.PARAM VIO=1.7
*.PARAM REFVOL=0.8

****************************************
* PARAMETER DEFINITION *

.PARAM PW='(PER-RISE-FALL)/2'      $ defining pulse width to be exactly half of pulse duration
.PARAM PER = 20n $f50               $ input frequency 50Mhz
.PARAM R=1N                         $ slew rate 1v/ns
.PARAM VILMAX='REFVOL-0.1'          $ setting VIL and VIH level
.PARAM VIHMIN='REFVOL+0.1'
.PARAM RISE='(VIHMIN-VILMAX)*R'   $ calculating rise time from slew rate.
.PARAM FALL='(VIHMIN-VILMAX)*R'

****************************************
* SUPPLY VOLTAGE *

Vvdd vdd GND CORE $parameter defined for core voltage
Vvdd1 vdd1 GND VIO $parameter defined for vdd1
Vref ref GND REFVOL

*******************************************
* INPUT PAD

VIOPAD2 IOPAD2 GND 0
*******************************************
* INPUT STIMULUS *

VIOPAD1 IOPAD1 GND PULSE(VILMAX VIHMIN 0n RISE FALL PW PER)

*******************************************
* SIMULATION COMMANDS *

.TRAN 0.1N 50N
.PROBE TRAN V(DATA_TO_CORE) V(IOPAD1) V(VDD) V(VDD1) V(IOPAD2)

.MEAS TRAN P2P PP V(DATA_TO_CORE)

.MEAS TRAN tpLH TRIG V(IOPAD1) VAL='VILMAX+(VIHMIN-VILMAX)/2' RISE=4 TARG
+V(DATA_TO_CORE) VAL='P2P/2' RISE=4

.MEAS TRAN tpHL TRIG V(IOPAD1) VAL='VILMAX+(VIHMIN-VILMAX)/2' FALL=4 TARG
+V(DATA_TO_CORE) VAL='P2P/2' FALL=4

.MEAS TRAN TON_OUT TRIG V(DATA_TO_CORE) VAL='P2P/2' RISE=3 TARG +V(DATA_TO_CORE)
VAL='P2P/2' FALL=3

.MEAS TRAN TOFF_OUT TRIG V(DATA_TO_CORE) VAL='P2P/2' FALL=3 TARG
+V(DATA_TO_CORE) VAL='P2P/2' RISE=4

.MEAS TRAN TPER_OUT TRIG V(DATA_TO_CORE) VAL='P2P/2' RISE=3 TARG
+V(DATA_TO_CORE) VAL='P2P/2' RISE=4

.MEAS TRAN DUTY_OUT PARAM='100*TON_OUT/(TON_OUT+TOFF_OUT)'
```

```
.MEAS TRAN TPER_IN TRIG V(IOPAD1) VAL='VILMAX+(VIHMIN-VILMAX)/2' RISE=3 TARG
+V(IOPAD1) VAL='VILMAX+(VIHMIN-VILMAX)/2' RISE=4

.MEAS TRAN DUTY_IN PARAM='100*TON_IN/(TON_IN+TOFF_IN)'

.MEAS TRAN VIL MIN V(IOPAD1)

.MEAS TRAN VIH MAX V(IOPAD1)

.MEAS TRAN TON_IN TRIG V(IOPAD1) VAL='VILMAX+(VIHMIN-VILMAX)/2' RISE=3 TARG
+V(IOPAD1) VAL='VILMAX+(VIHMIN-VILMAX)/2' FALL=3

.MEAS TRAN TOFF_IN TRIG V(IOPAD1) VAL='VILMAX+(VIHMIN-VILMAX)/2' FALL=3 TARG
+V(IOPAD1) VAL='VILMAX+(VIHMIN-VILMAX)/2' RISE=4

.END
```

3.5.1 Description of .cir file

The following points give the description of .cir file.

- Library of MOS model are attached in the .cir file. These file contain model definition, parameter definition and their values for MOS transistors used. These files are given from the FAB.
- All control signals are set according to the information given in the design document. The pin having no connection should be connected to ground or supply depending if that signal is active high or active low.
- Block selection bits are set. As different blocks support separate standards they need to be selected to verify a standard.
- Supply voltages are given to node specified.
- Parameters are defined. These include pulse width, slew rate, VIL, etc.
- Input stimulus is a pulse of 50% duty cycle. This represents digital data bits of logic ‘1’ and ‘0’ alternately.
- Measure commands to measure various parameters.
- Voltage value, temperature and sigma value are listed for each case. These are simulation are carried out using .alter statements.

3.5.2 Parameters Measured

The following is the list of parameters measured.

- Input on time.
- Input off time.
- Input time period.
- Output on time.
- Output off time.
- Output time period.
- Rise to rise delay.
- Fall to fall delay.
- Output voltage low level (VOL).
- Output voltage high level (VOH).
- Peak to peak value of output.
- Output Duty cycle.

3.4.3 Maximum frequency

There are three different maximum frequencies referred in the document.

a. FIL: Frequency Input Limited. FIL denotes maximum frequency input generator can provide for a given slew rate. This is limitation from input generator i.e. not a limitation of the circuit.
b. Fcutoff: This the frequency up to which the circuit responds to the input properly. As we increase frequency beyond Fcutoff, circuit starts missing some input cycles and does not give consistent output duty cycle. As we go further circuit does not respond to any input.
c. Fduty: This is the frequency limited by the output duty cycle. This limit is generally depends upon how much deviation in duty cycle successor circuit can tolerate. Generally it is around 70-75%.

3.5 Methodology followed

First simulation is carried out at frequency at which we are sure the circuit responds to input signal properly. For example for single ended standards Fin was taken 50Mhz. this is carried out for four cases.

Then using values of rise to rise delay and fall to fall delay Fduty is found using the relation. Simulation is fired at Fduty and simulation measured duty cycle value is noted and compared with theoretical value. For cases when difference between the delays is very less then the value of Fduty is large. If at Fduty the circuit does not respond, then Fcutoff is found by trial and error method. These frequencies are tabulated, they are marked to differentiate between Fduty, Fcutoff & FIL.

3.6 Summary

This chapter describes how verification of IOB was done. The strategy used is discussed here. The next chapter discusses the results of the simulation on these circuits.

Chapter 4

Results

4.1 Introduction

The parameters measured were tabulated and submitted to the company. As this project was done in STMicrelectronics, the numerical results have not been produced in this chapter because they are confidential in nature. Electrical specifications standard JEDEC standards are tabulated in this chapter.

4.2 Standards supported by IOB

IOB refers to the IOB of FPGA which are configurable. They follow a set of standards for communicating from peripheral devices. These standards are classified in three classes.

- Single ended
- Pseudo-differential
- Fully-differential

The following table lists the name of the standards.

Single Ended	*Pseudo-Differential*	*Fully-Differential*
LVCMOS 12	HSTL15	BLVDS25
LVCMOS 15	HSTL18	LVDSEXT25
LVCMOS 18	SSTL18	ULVDS25
LVCMOS 25	SSTL25	LDT25
LVCMOS 33	GTL	LVDS25
LVTTL	GTL+	

Table 4.1 Standards supported by IOB

4.3 Specification of the Standard.

4.3.1 Single Ended Standards

A. LVCMOS12

Parameter	***Test Condition***	***Min***	***Typ***	***Max***
VCCO		1.1	1.2	1.3
VREF		NA	NA	NA
VTT		NA	NA	NA
VIH (dc)		0.65*VCCO	-	VCCO + 0.3
VIL (dc)		-0.3	-	0.35 * VCCO
VOH	*IOH = -100 μA*	VCCO-0.1	-	-
VOH	*IOH = -2 mA*	0.75 * VCCO	-	-
VOL	*IOL = 100μA*	-	-	0.1
VOL	*IOL = 2mA*	-	-	0.25 * VCCO

B. LVCMOS15

Parameter	*Test Condition*	*Min*	*Typ*	*Max*
VCCO		1.4	1.5	1.6
VREF		NA	NA	NA
VTT		NA	NA	NA
VIH (dc)		0.65*VCCO	-	VCCO + 0.3
VIL (dc)		-0.3	-	0.35 * VCCO
VOH	*IOH = -100 µA*	1.2	-	-
VOH	*IOH = -2 mA (VIH=0.91V, VIL=0.49V)*	0.75 * VCCO (VCCO=1.4)	-	-
VOL	*IOL = 100µA*	-	-	0.2
VOL	*IOL = 2mA (VIH=0.91V, VIL=0.49V)*	-	-	0.25 * VCCO (VCCO =1.4)

C. LVCMOS18

Parameter	*Test Condition*	*Min*	*Typ*	*Max*
VCCO		1.65	1.8	1.95
VREF		NA	NA	NA
VTT		NA	NA	NA
VIH (dc)		0.65*VCCO	-	VCCO + 0.3
VIL (dc)		-0.3	-	0.35 * VCCO
VOH	*IOH = -100 µA*	VCCO-0.2	-	-
VOH	*IOH = -2 mA*	VCCO-0.45	-	-
VOL	*IOL = 100µA*	-	-	0.2
VOL	*IOL = 2mA*	-	-	0.45

D. LVCMOS25

Parameter	***Test Condition***	***Min***	***Typ***	***Max***
VCCO		2.3	2.5	2.7
VREF		NA	NA	NA
VTT		NA	NA	NA
VIH (dc)		1.7	-	VCCO+0.3
VIL (dc)		-0.3	-	0.7
VOH	*IOH = -1mA (VIH=1.7V, VCCO=2.3V)*	2		
VOH	*IOH = -8 mA (VIH=1.7V, VCCO=2.3V)*	1.8		
VOL	*IOL = 1mA (VIL=0.7V, VCCO=2.3V)*			0.4
VOL	*IOL = 8mA (VIL=0.7V, VCCO=2.3V)*			0.6

E. LVTTL

Parameter	*Test Condition*	*Min*	*Typ*	*Max*
VCCO		3.0	3.3	3.6
VREF		NA	NA	NA
VTT		NA	NA	NA
VIH (dc)	*VOUT ≥VOH (min) or VOUT≤ VOL (max)*	2	-	VCCO+0.3
VIL (dc)	*VOUT ≥VOH (min) or VOUT≤ VOL (max)*	-0.3	-	0.8
VOH	*IOH = -2mA (VCCO=3.0V)*	2.4		
VOL	*IOL = 2mA (VCCO=3.0V)*			0.4

4.3.2 Pseudo-differential Standards

A. HSTL 1.5
This Standard supports three classes HSTL15 class I, HSTL15 class II and HSTL15 class III.

HSTL15 Class I

Parameter	*Min*	*Typ*	*Max*	*Unit*
VCCO	1.4	1.5	1.6	V
VTT				
Vref	0.68	0.75	0.9	V
VIH (dc)	Vref+0.1	-	VCCO+0.3	V
VIL (dc)	-0.3	-	Vref-0.1	V
VIH (ac)	Vref+0.2	-	-	V
VIL (ac)	-	-	Vref-0.2	V
VOH(dc)	VCC-0.4	-	-	V
VOL(dc)	-	-	0.4	V
VOH(ac)	VCCO-0.5	-	-	V
VOL(ac)	-	-	0.5	V

Driver output slew rate	

* The values of VOL & VOH are to be measured in the proper load condition as specified in the JEDEC standard

HSTL15 Class II

Parameter	*Min*	*Typ*	*Max*	*Unit*
VCCO	1.4	1.5	1.6	V
VTT				V
Vref	0.68	0.75	0.9	V
VIH (dc)	Vref+0.1	-	VCCO+0.3	V
VIL (dc)	-0.3	-	Vref-0.1	V
VIH (ac)	Vref+0.2	-	-	V
VIL (ac)	-	-	Vref-0.2	V
VOH(ac)	VCCO-0.5	-	-	V
VOL(ac)	-	-	0.5	V
VOH(dc)	VCC-0.4	-	-	V
VOL(dc)	-	-	0.4	V
Driver output slew rate				

* The values of VOL & VOH are to be measured in the proper load condition as specified in the JEDEC standard

HSTL15 Class III

Parameter	*Min*	*Typ*	*Max*	*Unit*
VCCO	1.4	1.5	1.6	V
VTT				
VREF*	0.68	0.75	0.9	V
VIH (dc)	Vref+0.1	-	VCCO+0.3	V
VIL (dc)	-0.3	-	Vref-0.1	V
VIH (ac)	Vref+0.2	-	-	V
VIL (ac)	-	-	Vref-0.2	V
VOH (dc)	VCCO-0.4	-	-	V
VOL (dc)	-	-	0.4	V
VOH (ac)	-	-	-	
VOL (ac)			0.5	V
Driver output slew rate				

***** The values of VOL & VOH are to be measured in the proper load condition as specified in the JEDEC standard

B. HSTL 1.8

This Standard supports three classes HSTL class I, HSTL class II and HSTL class III

HSTL18 class I

Parameter	*Min*	*Typ*	*Max*	*Unit*
VCCO	1.7	1.8	1.9	V
VREF	0.8	0.9	1.1	
VTT	-	VCCO*0.5	-	V
VIH (dc)	VREF + 0.1	-	-	V
VIL (dc)	-	-	VREF – 0.1	V
VIH (ac)	-	-	-	V
VIL (ac)	-	-	-	V
VOH (dc)	VCCO-0.4 (IOH ≥ 8mA)			V
VOL (dc)	0.4 (IOL ≥ 8mA)			V
VOH (ac)	VCCO-0.5			
VOL (ac)	0.5			V

* The values of VOL & VOH are to be measured in the proper load condition as specified in the JEDEC standard

HSTL 18 class II

Parameter	*Min*	*Typ*	*Max*	*Units*
VCCO	1.7	1.8	1.9	V
VREF	-	0.9	-	V
VTT	-	VCCO*0.5	-	V
VIH (dc)	VREF + 0.1	-	-	V
VIL (dc)	-	-	VREF – 0.1	V
VIH (ac)	-	-	-	-
VIL (ac)	-	-	-	-
VOH (dc)	VCCO-0.4 (IOH ≥ 16mA)			V
VOL (dc)	0.4 (IOL ≥ 8mA)			V
VOH (ac)	VCCO-0.5			
VOL (ac)	0.5			V

* The values of VOL & VOH are to be measured in the proper load condition as specified in the JEDEC standard

HSTL 18 class III

Parameter	***Min***	***Typ***	***Max***	***Units***
VCCO	1.7	1.8	1.9	V
VREF	-	1.1	-	V
VTT	-	VCCO	-	V
VIH (dc)	VREF + 0.1	-	-	V
VIL (dc)	-	-	VREF – 0.1	V
VIH (ac)	-	-	-	-
VIL (ac)	-	-	-	-
VOH (dc)	VCCO-0.4 (IOH ≥ 8mA)			V
VOL (dc)	0.4 (IOL ≥ 24mA)			V
VOH (ac)	NA			
VOL (ac)	0.5			V

* The values of VOL & VOH are to be measured in the proper load condition as specified in the JEDEC standard

C. SSTL 1.8

SSTL 18 class I

Parameter	***Min***	***Typ***	***Max***	***Units***
VCCO	1.7	1.8	1.9	V
VREF	0.833	0.9	0.969	V
VTT	VREF – 0.04	VREF	VREF+0.04	V
VIH (dc)	VREF + 0.125	-	VCCO + 0.3	V
VIL (dc)	-0.3	-	VREF – 0.125	V
VIH (ac)	VREF + 0.25	-	-	V
VIL (ac)	-	-	VREF – 0.25	V
VSWING (Max)	-	1.0	-	V
SLEW	-	1.0	-	V/ns
VOH (ac)	VTT + 0.603	-	-	V
VOL (ac)	-	-	VTT - 0.603	V
IOH (dc)	-13.4	-	-	mA
IOL (dc)	13.4	-	-	mA

* The values of VOL & VOH are to be measured in the proper load condition as specified in the JEDEC standard

D. SSTL 2.5

This standard supports two classes SSTL 25 class I and SSTL25 class II

SSTL 25 class I

Parameter	***Min***	***Typ***	***Max***	***Units***
VCCO	2.3	2.5	2.7	V
VREF	1.13	1.25	1.38	V
VTT	VREF – 0.04	VREF	VREF+0.04	V
VIH (dc)	VREF + 0.15	-	VCCO +0.3	V
VIL (dc)	-0.3	-	VREF – 0.15	V
VIH (ac)	VREF + 0.31	-	-	V
VIL (ac)	-	-	VREF – 0.31	V
VSWING (Max)		1.5		V
SLEW		1.0		V/ns
VOH (ac)	VTT + 0.608	-	-	V
VOL (ac)	-	-	VTT - 0.608	V
IOH (dc)	-8.1	-	-	mA
IOL (dc)	8.1	-	-	mA

* The values of VOL & VOH are to be measured in the proper load condition as specified in the JEDEC standard

SSTL 25 class II

Parameter	***Min***	***Typ***	***Max***	***Units***
VCCO	2.3	2.5	2.7	V
VREF	1.13	1.25	1.38	V
VTT	VREF – 0.04	VREF	VREF+0.04	V
VIH (dc)	VREF + 0.15	-	VCCO +0.3	V
VIL (dc)	-0.3	-	VREF – 0.15	V
VIH (ac)	VREF + 0.31	-	-	V
VIL (ac)	-	-	VREF – 0.31	V
VSWING (Max)		1.5		V
SLEW		1.0		V/ns
VOH (ac)	VTT + 0.81	-	-	V
VOL (ac)	-	-	VTT - 0.81	V
IOH (dc)	-16.2	-	-	mA
IOL (dc)	16.2	-	-	mA

* The values of VOL & VOH are to be measured in the proper load condition as specified in the JEDEC standard

E. SSTL 3.3

This standard supports two classes SSTL 33 class I and SSTL 33 class II

SSTL33 class I

Parameter	***Min***	***Typ***	***Max***	***Units***
VCCO	3.0	3.3	3.6	V
VREF	1.3	1.5	1.7	V
VTT	VREF – 0.05	VREF	VREF+0.05	V
VIH (dc)	VREF + 0.2	-	VCCO + 0.3	V
VIL (dc)	-0.3	-	VREF – 0.2	V
VIH (ac)	VREF + 0.4	-	-	V
VIL (ac)	-	-	VREF – 0.4	V
VREF (for ac test conditions)		0.45 * VCCO		V
VSWING (Max)	-	2.0	-	V
SLEW	-	1.0	-	V/ns
VOH (ac)	VTT + 0.6	-	-	V
VOL (ac)	-	-	VTT - 0.6	V
IOH (dc)	-8	-	-	mA
IOL (dc)	8	-	-	mA

Vref is calculated using the range of VCCO or termination vol. as specified in the JEDEC standard

SSTL33 class II

Parameter	***Min***	***Typ***	***Max***	***Units***
VCCO	3.0	3.3	3.6	V
VREF	1.3	1.5	1.7	V
VTT	VREF – 0.05	VREF	VREF+0.05	V
VIH (dc)	VREF + 0.2	-	VCCO + 0.3	V
VIL (dc)	-0.3	-	VREF – 0.2	V
VIH (ac)	VREF + 0.4	-	-	V
VIL (ac)	-	-	VREF – 0.4	V
VREF (for ac test conditions)		0.45 * VCCO		V
VSWING (Max)	-	2.0	-	V
SLEW	-	1.0	-	V/ns
VOH (ac)	VTT + 0.8	-	-	V
VOL (ac)	-	-	VTT - 0.8	V
IOH (dc)	-16	-	-	mA
IOL (dc)	16	-	-	mA

Vref is calculated using the range of VCCO or termination vol. as specified in the JEDEC standard

F. GTL (Gunning Transceiver Logic)

Parameter	*Min*	*Typ*	*Max*	*unit*
VCCO	-	-	-	V
VTT	1.14	1.2	1.26	V
VREF*	0.74	0.8	0.86	V
VIH (dc)	Vref+0.05	0.83		V
VIL (dc)		0.77	Vref –0.05	V
VIH (ac)	-	-	-	
VIL (ac)	-	-	-	
VOH	-	-	-	
VOL		0.2	0.4	V
IOH	-			
IOL	40			mA

Vref is calculated using the range of VCCO or termination vol. as specified in the JEDEC standard

G. GTLP

Parameter	*Min*	*Typ*	*Max*	*Unit*
VCCO	-	-	-	V
VTT	1.4	1.5	1.6	V
VREF*	0.9	1	1.1	V
VIH (dc)	Vref+0.1			V
VIL (dc)			Vref –0.1	V
VIH (ac)	-	-	-	
VIL (ac)	-	-	-	
VOH	-	-	-	
VOL	-	-	0.6	V
IOH	-			
IOL	36			mA

Vref is calculated using the range of VCCO or termination vol. as specified in the JEDEC standard

4.4.3 Fully Differential Standards

Requirements for the differential standards (Input path)

S.No.	*Standard*	*VCCO (V)*			*VID (mV)*			*VICM(V)*		
		Min.	Typ.	Max.	Min.	Typ.	Max.	Min.	Typ.	Max.
1	BLVDS_25	2.38	2.5	2.63	-	350	-	-	1.25	-
2	**LVDSEXT_25**	2.38	2.5	2.63	100	540	1000	0.3	1.20	2.20
3	**ULVDS_25**	2.38	2.5	2.63	100	200	-	0.44	0.60	0.78
4.1	**LDT_25 (dc)**	2.38	2.5	2.63	200	600	1000	0.44	0.60	0.78
4.2	**LDT_25 (ac)**	2.38	2.5	2.63	300	600	900	0.38	0.60	0.84
5	**LVDS_25**	2.38	2.5	2.63	100	350	600	0.3	1.25	2.20

4.5 Summary

Actual results have not been shown in this chapter as they confidential in nature. The electrical specification of JEDEC standards which were followed for verification is listed in tabular format.

Chapter 5
Conclusion

In this thesis Input Output block of FPGA is verified according to JEDEC standards. IOB's used in FPGA's are made configurable to support different applications. In this thesis the postlay netlist of configurable IOB of FPGA is verified according to JEDEC standard. And the rise to rise delay, fall to fall delay and output duty cycle were measured. It is seen that the IOB comply with electrical standard described by the JEDEC documents. Maximum frequency of operation of IOB was also verified and values tabulated. The simulations were carried out on EDA tools Eldo and Hspice. Measurements were done on the output waveforms. A relationship between rise delay, fall delay, input & output duty cycle was observed. An expression was derived which explains the behaviour observed, & this expression lets us calculate the maximum frequency when output is limited by duty cycle. The theoretical values found by the relation were confirmed with simulation results.

There are three different maximum frequencies observed in the result.

d. FIL: Frequency Input Limited. FIL denotes maximum frequency input generator can provide for a given slew rate. This is limitation from input generator i.e. not a limitation of the circuit.
e. Fcutoff: This the frequency up to which the circuit responds to the input properly. As we increase frequency beyond Fcutoff, circuit starts missing some input cycles and does not give consistent output duty cycle. As we go on further increasing the input frequency circuit does not respond to any input.
f. Fduty: This is the frequency limited by the output duty cycle. This value depends upon the rise & fall time delay and maximum duty cycle tolerable according to the circuit specifications.

The following observations were made:

- IOB's were seen to be following the JEDEC standards.
- Rise time delay and fall time delay should be equal ideally, but in some cases difference between these delays were seen. This difference limits the operating frequency in terms of output duty cycle.
- The value of Fduty was found theoretically by keeping the output duty cycle 70%.
- Difference in delay is seen more in single ended IOB.
- While the pseudo-differential IOB exhibits moderate difference in delay less than single ended IOB
- Difference in delay is not prominent in differential standard. This factor does not affect the maximum frequency of operation.
- The formula by which the theoretical limit of duty cycle limited operating frequency assumes that the rise & fall time delay remains constant with increasing frequency.

References

Books/Manuals

1. N. Weste and K. Eshraghian, *Principles of CMOS VLSI Design*, 2nd edition, Addison-Wesley.

2. J. Rose, Brown, Francis, Vranesic, *Field Programmable Gate Arrays,* Kluwer Academic Publishers.

3. J. Rose, E. Gamal, *Architecture of Field Programmable Gate Arrays,* Proceedings of IEEE, Vol-81, No-7, July 1993.

4. Wesley Morris, *Latch-up in CMOS,* 41st Annual International Reliability Symposium, Dallas, Texas 2003.

5. *Understanding Latch-Up in Advanced CMOS Logic,* Application note, Fairchild semiconductor.

6. Guy Rabbat, *Hardware & Software Concepts in VLSI*

7. C. Kittel, Introduction to Solid State Physics, Wiley & sons,

8. R.S. Muller and T. I. Kamins, Device Electronics for Integrated Circuits, Second edition,Wiley & Sons

9. S. Wang, "Fundamentals of Semiconductor Theory and Device Physics": Prentice Hall

10. *Eldo user manual,* Mentor Graphics

11. *Hspice Command Reference,* Synopsys.

12. *Hspice simulation commands,* Synopsys.

13. *HsimPlus User Manual*, Nassda Corporation.

14. *Hspice tutorial,* Stanford University Website.

15. *Spartan Datasheet,* Xilinx website.

URLs

1. www.fpga4fun.com
2. www.esda.com, ESD Association
3. www.xilinx.com, Device datashheet
4. www.fairchildsemi.com/an/AN/AN-600.pdf, Understanding Latch-Up in Advanced CMOS Logic
5. cnx.rice.edu/content/m1031/latest.htm, Electrostatic Discharge and Latch-Up
6. assets.zarlink.com/AN/msan107.pdf, Understanding and Eliminating Latch-Up in CMOS Applications
7. www.analog.com/UploadedFiles/Application_Notes/ AN-707_0.pdf, ESD and Latch-Up Considerations Application Note (AN-707)
8. focus.ti.com/general/docs/lit/getliterature.tsp?literatureNumber=slya014a&fileType=pdf "Latch-Up, ESD, And Other Phenomena"
9. www.esdjournal.com, ESD Journal - The ESD & Electrostatics Magazine
10. www.minicircuits.com/appnote/an40005.pdf The Prevention and Control of Electrostatic Discharge
11. www.semiconfareast.com/esd.htm
12. www.informit.com/articles/article.asp? Electrostatic Discharge Precautions
13. resource.intel.com/telecom/support/install/config/esd/esd.pdf
14. www.sascosemi.com/download/leadfree/jesd97_en.pdf JEDEC standards and publications
15. www.edacafe.com/books/ASIC/ASICs.php EDACafe ASICs-the Book
16. www-ee.eng.hawaii.edu/~msmith/ ASICs/HTML/Book2/CH06/CH06.7.htm 6.7 Xilinx I/O Block
17. www.ieee-explore.com

Appendix

A. ELDO SYTAX REFERENCE

ELDO Simulator Commands and syntax

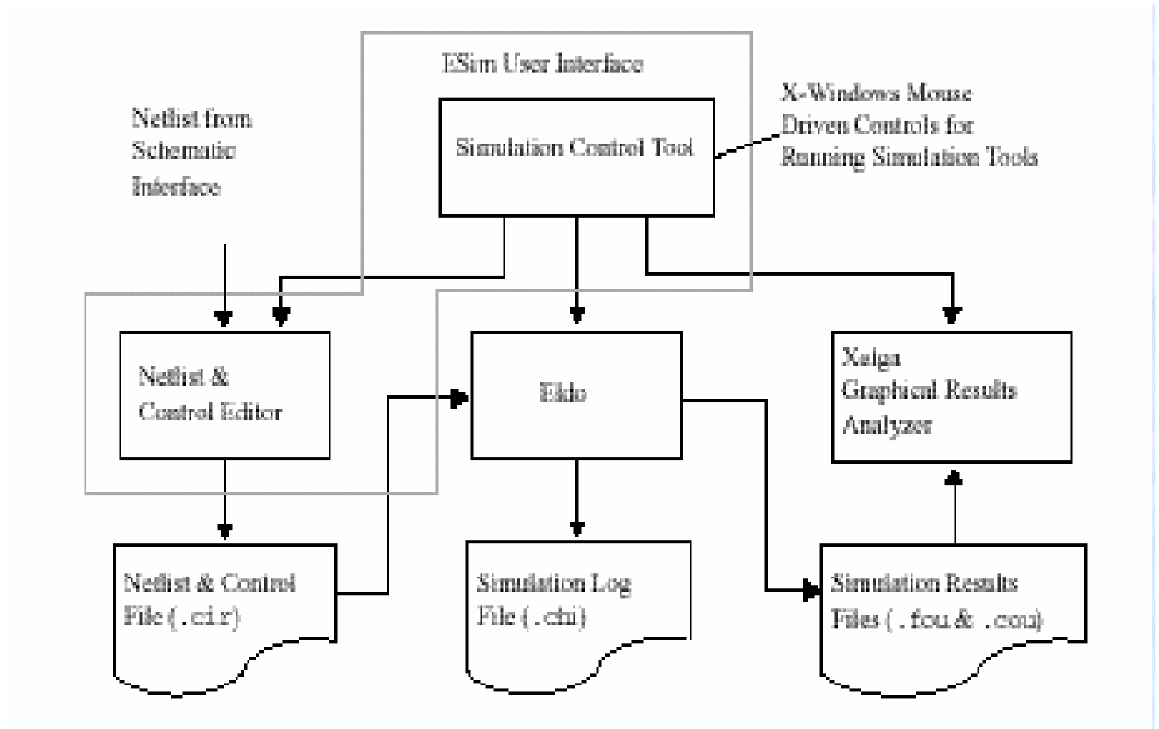

ESim User Interface
Netlist from Schematic Interface
Simulation Control Tool
X-Windows Mouse Driven Controls for Running Simulation Tools
Netlist & Control Editor
Eldo
Xelga Graphical Results Analyzer
Netlist & Control File (.cir)
Simulation Log File (.chi)
Simulation Results Files (.fou & .cou)

Figure 1.1 Eldo Input & Output Files

Eldo input & output files

<file>.cir
The main Eldo control file, containing circuit NETLIST, stimulus and simulation control commands.
<file>.chi
output log file containing ASCII data, including results and error messages.
<file>.cou
binary file containing Eldo analog simulation results data

Syntax for firing Eldo

eldo [-b] [-l log_file_name] [-o output_file_name] [-outpath output_dir_name] [-i] [-couf] cir_file_name [-noascii] [-noconf] [-spiout]

-b Runs the simulation in the background.
-l Log file name for a background simulation.
-o Output .chi file name.

-outpath Directory in which all output files are created.
-i Input .cir file name.
-**couf** Forces Eldo to create .cou files containing FLOAT rather than DOUBLE values, which results in saving 50% of the disk space.

Format of .cir file

First Line

The first line is format free and reserved for the circuit title. This line is mandatory and serves as the heading on graphical results output.

Continuation Lines

The length of one line is limited to 256 characters. A line may be continued by using the + character at the beginning of the new line.

Comments

This is comment '!' on same line
* Comment on new line
#Com
block
of
comment
#endcom

Scale Factors

F	1e-15
P	1e-12
N	1e-09
U	1e-06
M	1e-03
K	1e+03
MEG	1e+06
G	1e+09
T	1e+12
dB	for decibels

Arithmetic Functions & Operators

SQRT(VAL)	LOG(VAL)	LOG10(VAL)	DB(VAL)
EXP(VAL)	COS(VAL)	SIN(VAL)	TAN(VAL)
ACOS(VAL)	ASIN(VAL)	ATAN(VAL)	COSH(VAL)
SINH(VAL)	TANH(VAL)	SIGN(VAL)	ABS(VAL)
TRUNC(VAL)	DMIN(VAL1, VAL2)	DMAX(VAL1, VAL2)	
DERIV(VAL)			

Resistor

Rxx N1 N2 VAL

Rxx NP NN **VALUE**={EXPR}
xx : resistor name
N1, N2 : names of the resistor nodes
VAL value of resistor in OHM at nominal temperature
Example
r1 n3 n4 3.3k
r1 1 2 value={2k*v(3, 4)*i(v5)}

Capacitor
Cxx NP NN VAL [IC=VAL]
Cxx NP NN VALUE={EXPR}
IC=VAL: sets the initial guess for voltage across cap
UIC must be used in .TRAN statement Components
Example
c1 n3 n4 0.5pf
c1 1 2 value={2n*v(3, 4)*i(v5)}

Inductors
Lxx NP NN VAL [IC=VAL]
Lxx NP NN VALUE={EXPR}
Example
l1 n13 n8 5u
l1 1 2 value={2u*v(3, 4)*i(v5)}

Coupled Inductors
Kxx lyy lzz KVAL

Junction Diodes
Dxx NP NN MNAME
Example
*DIODE model definition
.model dio d level=3
...
*main circuit
d1 2 10 dio

BJT
Qxx NC NB NE [NS] MNAME
Example *BJT model definition

```
.model qmod npn bf=160 rb=100 cjs=2p
+ tf=0.3n tr=6n cje=3p cjc=2p vaf=100
...
*main circuit
q23 10 24 13 qmod
```

JFET- Junction Field Effect Transistor

Jxx ND NG NS MNAME
Example
*JFET model definition
.model je20 njf vto=-3.2 beta=0.98m
+ lambda=2.5m cgs=5p cgd=1.3p is=7p
...
*main circuit
j1 3 2 0 je20

MOSFET
Mxx ND NG NS NB MNAME][M=VAL] W=VAL L=VAL
M= device multiplier(for devices in parallel)
Example
M1 1 2 3 3 PMOS m=1 W=1u L=0.18u

Subcircuit Instance
Xxx NN {NN} NAME
Example
*SUBCKT definition
.subckt inv 1 2
r1 1 3 2k
r2 3 4 4k
r3 4 2 3k
.ends inv
...
*subcircuit instance
x1 1 48 inv

Types of sources
Independent Sources
Independent Voltage Source **V**
Independent Current Source **I**
Independent sources can be assigned a time-dependent value for transient analysis.
The time zero values of time dependent sources are used for DC analysis.

Time Dependent Sources
Exponential function ‘exp’
Pattern function ‘pattern’
Pulse function ‘pulse’
Piece wise linear function ‘pwl’
Sine function ‘sin’

Linear dependent sources
Linear Voltage Controlled Voltage
Linear Current Controlled Current F
Linear Voltage Controlled Current G

Linear Current Controlled Voltage H
E, F, G and H are constants representing voltage gain, current gain, transconductance and trans-resistance respectively.

Non-Linear Dependent Sources

Non-linear Voltage Controlled Voltage
Non-linear Current Controlled Current Source
Non-linear Voltage Controlled Current
Non-linear Current Controlled Voltage

Syntax of Sources

Independent Voltage Source

Vxx NP NN [DC] DCVAL] [TIME_DEPENDENT_FUNCTION]
vplus n12 n13 24
v7 n4 n9 ac 1.2 pwl (0 3 5n 0 10n 0)

Independent Current Source

Ixx NP NN [DC] [TIME_DEPENDENT_FUNCTION]
i23 n2 n3 1.0e-4

Exponential Source

Exp(v1 v2 [td1 [tau1[d2[tau2]]]])
V1=initial value
V2=target value of pulse
Td1=rise delay time
Tau1=rise time constant
Td2=fall delay time
Tau2=fall time constant

Independent Voltage Source

Vxx NP NN [DC] DCVAL] [TIME_DEPENDENT_FUNCTION]
vplus n12 n13 24
v7 n4 n9 ac 1.2 pwl (0 3 5n 0 10n 0)

Independent Current Source

Ixx NP NN [DC] [TIME_DEPENDENT_FUNCTION]
i23 n2 n3 1.0e-4

Exponential Source

Exp(v1 v2 [td1 [tau1[d2[tau2]]]])
V1=initial value
V2=target value of pulse
Td1=rise delay time
Tau1=rise time constant
Td2=fall delay time

Tau2=fall time constant

Pattern

Pattern vhi vlo delay trise tfall tbit bits [r]

VHI= voltage representing logic 1
Vlo=voltage representing logic 0
Delay=delay before pattern series start
Trise=rise time between pattern values
Tfall=fall time between pattern voltage representing logic
Tbit=time spent at 1 or 0
Bits=string of 1 and 0 representing pattern bit
R = periodic

Pulse function

Pulse (v0 v1 [td[tr[tf[pw[per]]]]])

V0: initial value of DC voltage or current
V1: pulse magnitude in volts or amperes
TD: delay time
TR: rise time(default value is TPRINT)
TF: fall time
PW : pulse width (default TSTOP)
PER: pulse period

Piece-wise linear

Pwl (t1 v1 t2 v2 t3 v3 tn vn [r])

Tn=time at vi is supplied
Vn=value of source at time ti
R: repetitive

Sine source

Sin (v0 va [fr[td[theta]]])

V0 : offset voltage
VA: amplitude of sine
FR : frequency in hz
TD: delay time in sec
THETA: damping factor in 1/sec
Vin 1 0 sin (0 10 1meg 1u 50e4)

Sin (v0 va [fr[td[theta]]])

V0 : offset voltage
VA: amplitude of sine
FR : frequency in hz
TD: delay time in sec
THETA: damping factor in 1/sec
Vin 1 0 sin (0 10 1meg 1u 50e4)

vsin n2 n3 sin (0 110 50 0 0)
vsin n4 n9 sin(0 50 50 .05 9)

Sub-circuit definition
Subckt name nn {nn}
<Circuit components>
.Ends [name]

SIMULATION COMMANDS
DC Analysis
OP ANALYSIS
TF FUNCTION
AC ANALYSIS
TRANSIENT ANALYSIS
PLOT and PRINT

DC Analysis (.DC)
Single analysis of the circuit's quiescent state or operating point
.DC CNAM [L|W] START STOP INCR
Variation of an element size or value, a parameter or a temperature CNAM: name of component on which geometrical or value variations are performed
r7 3 4 100k
.dc r7 10k 100k 10k
.DC PARAM P_NAME START STOP INCR
PARAM:keyword indicating that a temperature is to be varied P_NAME:name of globally declared parameter to be varied START, STOP: start and stop value of component CNAM, voltage, temp or current
r1 1 2 p1
.param p1=1k
.dc param p1 1k 10k 1k
.DC TEMP START STOP INCR
TEMP:keyword indicating that the temperature is to be varied. Voltage or current sweep of the specified source
.DC SNAM START STOP INCR [SNAM2 START2 STOP2 INCR2]
SNAM:Name of voltage or current source which performs the DC sweep INCR:increment of component, voltage, temperature or current sweep.

OP Analysis
.OP
This command forces eldo to determine the operating point of circuit with inductors short circuited and removed capacitors The operating point is saved in the .chi file which contains information such as power dissipation, node voltages and source currents
.OP [T1 {TN}]
T1, TN Simulation times at which operating point information will be recorded.
.OP TIME=VAL [STEP=VAL] [TEMP=VAL]
TIME=VAL Simulation time for which .OP results are written.
STEP=VAL Step value for which .OP results are written. This is the case for when the. STEP command is used.

TEMP=VAL Current temperature for which .OP results are written.

.OP DC=VAL [DC2=VAL] [STEP=VAL] [TEMP=VAL]

DC=VAL DC sweep value for which .OP results are written.
DC2=VAL Second DC value for which .OP results are written in cases where a double DCSWEEP analysis is performed.

.TF OV IN

IN: Input voltage source name. Must be an independent source
OV: Requests the output voltage of a specific node or current through a voltage source. The syntax is as follows:
Example
vin 1 0 5
.tf v(3) vin

.AC DEC|OCT ND FSTART FSTOP [UIC]

DEC Keyword to select logarithmic variation.
OCT Keyword to select octave variation.
ND Number of points per decade or octave.
FSTART Start frequency in Hertz.
FSTOP Stop frequency in Hertz.

.AC LIN NP FSTART FSTOP [UIC]

LIN: Keyword to select linear variation.
NP: Number of points over the range from FSTART to FSTOP.
UIC: If UIC is specified, no DC analysis is performed before the AC analysis. Instead the circuit may be initialized using .RESTART or .USE.

.TRAN Analysis

Transient output variables are calculated as a function of time over a user specified time interval. The initial conditions are automatically determined by a DC analysis (unless the UIC parameter is specified) with all sources that are not time dependent being set to their DC values.

.TRAN TPRINT TSTOP [TSTART [HMAX]] [UIC]

TPRINT Printing or plotting increment for the printer output (in seconds). There is no relation between the internal time-step which Eldo computes for the simulation and the TPRINT value.
TSTOP The transient analysis duration in seconds.
TSTART No outputs are stored from 0 to TSTART seconds.
HMAX Sets the maximal internal time-step. When HMAX is specified both in the .OPTION command and in the .TRAN command, the HMAX in .OPTION is considered by Eldo.
UIC Keyword which indicates that do not solve for the quiescent operating point before beginning the transient analysis.

PLOTTING OF SIMULATION RESULTS

.PLOT DC|AC|TRAN OVN [(LOW, HIGH)] {OVN [(LOW, HIGH)]}
[UNIT=NAME]
DC Specifies that the plots are required for a DC analysis.
AC Specifies that the plots are required for an AC analysis.
TRAN Specifies that the plots are required for a transient analysis.
OVN Requests plotting of the voltage at a specified nodes or current through selected circuit components. The maximum number of plots allowed in one .PLOT command is eight.
LOW, HIGH The optional plot limits LOW and HIGH may be specified for each of the output variables. All output variables of the same kind (voltage for instance) to the left of a pair of plot limits
(LOW, HIGH) will be plotted using the same lower and upper bounds. If plot limits are not specified, Eldo uses default values

.PROBE
.PROBE [[AC|DC|TRAN] [V|I|S] [PRINT]
.PROBE [[AC|DC|TRAN] [list_of_plot_specifications]]
DC Specifies that the prints are required for a DC analysis.
AC Specifies that the prints are required for an AC analysis.
TRAN Specifies that the prints are required for a transient analysis.
V Causes all node voltages to be saved— this is the default option.
I Causes all currents to be saved (node voltages are not saved).
When the circuit has more than 50 nodes use LIMPROBE parameter
.option limprobe=100.
r1 n3 n40 1k
c1 n49 n61 5p
.probe
.PROBE TRAN V(x1.*)

Library Calls
.LIB <FILENAME>
insert model definitions into an input NETLIST from a library file.
EX: NETLIST file
M2 D G S 0 **NE0** L=0.8U W=5.4U
.LIB mos.lib
File mos.lib
.MODEL NE0 NMOS
+LEVEL=3 UO=592 VTO=0.8 CJ=5.23e-4
+CJSW=1.83e-10 CGSO=270.6e-12
.MODEL NE NMOS

.LIB <file_name> [LIBTYPE]
LIBTYPE: specifies the library variant to be used
for ex: worst, best, typical
within library file_name you must have sections defined by
.lib <libtype>
…

.endl

Ex:
Library mos.lib
.lib best
.model MN NMOS level=3 vt0=.5
.endl best
.lib typ
.model MN NMOS level=3 vt0=.75
.endl typ
CIRCUIT NETLIST
.lib mos.lib typ
m1 vdd g 0 0 MN L=2U W=5U
vdd vdd 0 5
vg g 0 0.8

.ADDLIB N <dir_name>
search a directory for files with file extensions .mod and .ckt (or.sub) and include in the circuit description (.cir) file the models which are not there in .cir file
N: An integer between 1 to 6, allocating priority to the directory search command.
EX: mn1 a b c d **mna** w=3u l=1u
.addlib 2 /user1/examples
.addlib 1 /user1/models

.INCLUDE <file_name>
used to insert the contents of the file in cir file.
EX:
.include models

Save Simulation Run
• .SAVE [FNAME] DC|END|TIME=VAL1 [REPEAT] [ALT|SEQ] [TEMP=VAL2] [STEP=VAL3]
Writes information at specific times during simulation to a file
FNAME. If no filename is given, results are saved in <circuit_name>.iic.
DC :DC part of the simulation should be saved in FNAME.
END: Keyword indicating that the simulation results should be saved after transient analysis has been completed in FNAME.
TIME: The value of this keyword is the discrete instant in time after the start of simulation that the simulation results should be saved in FNAME.
VAL1 :Time, in seconds, at which the simulation results should be saved to FNAME.
FNAME: Filename and extension into which simulation information is written.
REPEAT Keyword, used in conjunction with the TIME, ALT and SEQ options. When selected, Eldo saves the status of the simulation at every time interval VAL1 to FNAME in accordance with the ALT and
SEQ parameters described below.

ALT Used in conjunction with repeat to create the files FNAME.a, FNAME.b, FNAME.a,... in an alternating order, thus overwriting the previous file of the same extension and resulting in only two files FNAME.a and FNAME.b being created.
SEQ Used in conjunction with repeat to create the files FNAME.1, FNAME.2, FNAME.3, ... FNAME.N, in a sequential manner, thus creating N files depending on the values of TIME and REPEAT.
TEMP Keyword indicating that data should be saved to FNAME depending on temperature.
VAL2 Temperature, in degrees, at which the circuit data should be saved to FNAME.

```
r1 1 2 p2
...
.step param p2 1k 5k 1k
.save status.ccf step=4k
```

Causes status of current simulation to be saved in status.ccf file
when parameter p2 reaches 4kW.

.RESTART FNAME [FILE=COUF]

This command restarts a simulation run with information previously saved using the .SAVE command.
FNAME Filename containing simulation information which should be used to restart a simulation run.
COUF Name of file containing the previous .cou file. When this is provided, Eldo concatenates the old and the new binary data files.

.USE FILE_NAME [NODESET|IC|GUESS]

FILE_NAME saved using the .SAVE <file_name> DC command
inserts these as .NODESET, .GUESS or .IC values
EX:

```
.use test1.exa nodeset
```

Specifies that DC values found in the file <test1>.exa should be read and used as .NODESET values.

```
.use circuit1.iic ic
```

Specifies that DC values found in the file <circuit1>.iic should be read and used as .IC values.

Eldo provides three different commands for using start conditions
.GUESS .NODESET .IC

.GUESS V(NN)=VAL {V(NN)=VAL}
Initial DC Analysis Conditions. Helps to calculate the DC operating point by setting voltage values at selected nodes for the first iteration of a DC operating point calculation. This command differs from the .NODESET command node voltages are only fixed for the first iteration of a DC operating point calculation, whereas when using .NODESET the node voltages are fixed for the duration of the first DC operating point calculation.
It is useful when the approximate whereabouts of the DC operating point is known, enabling the simulator to converge more quickly.
Ex
.guess v(n4)=6v v(n5)=2v v(n6)=-5v

.NODESET V(NN)=VAL {V(NN)=VAL}
EX
.nodeset v(n4)=6v v(n5)=2v v(n6)=-5v

.IC V(NN)=VAL {V(NN)=VAL}
It is used to fix node voltages for the duration of a DC analysis. If the UIC parameter is also present (in the .TRAN command) no DC analysis is performed and the voltages are initialized as defined in the .IC command.

Parameters
.PARAM PAR=VAL|PAR={EXPR}
.PARAM PAR="NAME"
It is used to assign values to parameter variables used in model and device instantiation statements.
Ex
r1 1 2 rval
c1 1 2 cval
l1 1 2 lval
.param rval=2k cval=3p lval=2u
.model mod1 nmos level=3 vto=vtodef
*main circuit
m1 1 2 3 4 mod1 w=wdef l=ldef
.param vtodef=1 wdef=20u ldef=3u
r1 1 2 p2
.param p1=1k p3=**{**2*p**1}**
.param p2=**{sqrt(**p**1**)+3*p**3}**
Ex
r1 1 2 p2
.param p1=1k p3=**{**2*p**1}**
.param p2=**{sqrt(**p**1**)+3*p**3}**

Miscellaneous Commands

.ALTER
[ELEMENT]
[subckt]
[command]
[Comment]
.END
used to run eldo with modified NETLIST
following commands, **.print, .plot, .conso, .extract, .global, .include, .op, .option**. are always added.

.CONNECT N1 N2
N1, N2 Names of the nodes to be connected.
Example
.connect n7 n5

.CONSO VN {VN}
The **.CONSO** command computes and displays the average current flowing through the specified voltage source(s) during the simulation period.
Ex
vdd 100 101 5v
...
.conso vdd
.tran 1ns 100ns

.DEFWAVE W_NAME=WAVE_EXPR
It is used to define a new waveform by relating previously defined waveforms and nodes.
.defwave pow=i(v1)*v(v1)
...
v1 in out ...
However, the following is allowed:
v1 in out ...
...
.defwave pow=i(v1)*v(v1)

.EXTRACT [AC|DC|DCSWEEP|TRAN] [LABEL=NAME]
.EXTRACT [AC|DC|DCSWEEP|TRAN] [LABEL=" string "]
DCSWEEP DCSWEEP extraction.
DCAC Extraction after the DC analysis performed prior to an AC analysis.
DCTRAN Extraction after the DC analysis performed prior to a TRAN analysis.
TRAN Extraction during TRAN analysis.
AC Extraction during AC analysis.

.GLOBAL NN {NN}

Declare global node(s), making them known throughout a circuit without having to declare them in each subcircuit.

Ex:

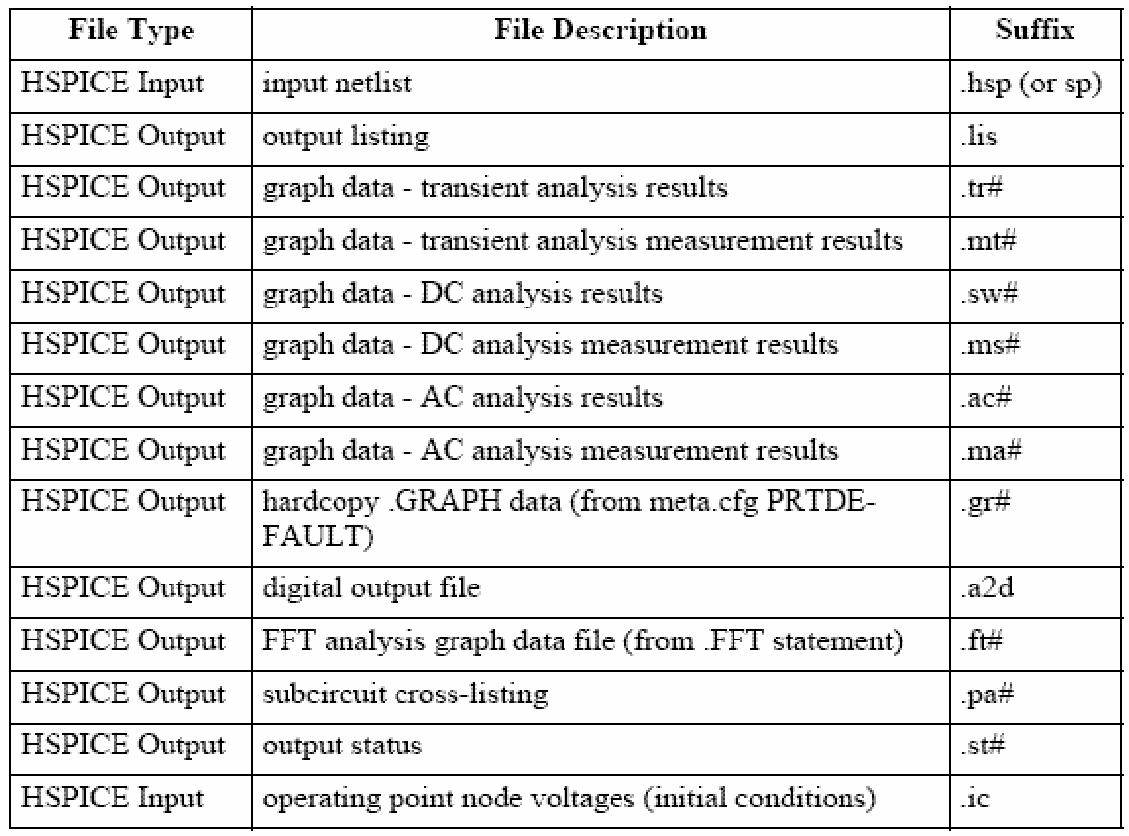
```
.global vdd vss
```

B. HSPICE SIMULATION & SYNTAX REFERENCE

HSPICE Basics

An input netlist file must be created to begin the design entry and simulation process. Once you have created the file (*filename.hsp/cir*), enter
Hspice *filename.hsp/cir>filename.lis*
to begin the analyses specified in the input file. HSPICE stores the simulation results requested in an output listing file and, if .option post is specified, a graph data file. When post is specified, the complete circuit solution (either steady state, time, or frequency domain) is stored. The results for any node voltage or branch current can then be viewed or plotted using Avanwaves. If you are converting a SPICE-3 input file to HSPICE format, it is only necessary that you add the line .option post somewhere in your file and put .end at the end (make sure you hit <cr> after the .end statement to form a complete line).

HSPICE also has specific file naming conventions to indicate the function of each file. All of the files associated with a particular design reside in one directory and are named by catenating the design name and a particular suffix (see Table 1). Both HSPICE and Avanwaves extract the design name from the input file and use it to form the output files.



File Type	File Description	Suffix
HSPICE Input	input netlist	.hsp (or sp)
HSPICE Output	output listing	.lis
HSPICE Output	graph data - transient analysis results	.tr#
HSPICE Output	graph data - transient analysis measurement results	.mt#
HSPICE Output	graph data - DC analysis results	.sw#
HSPICE Output	graph data - DC analysis measurement results	.ms#
HSPICE Output	graph data - AC analysis results	.ac#
HSPICE Output	graph data - AC analysis measurement results	.ma#
HSPICE Output	hardcopy .GRAPH data (from meta.cfg PRTDE-FAULT)	.gr#
HSPICE Output	digital output file	.a2d
HSPICE Output	FFT analysis graph data file (from .FFT statement)	.ft#
HSPICE Output	subcircuit cross-listing	.pa#
HSPICE Output	output status	.st#
HSPICE Input	operating point node voltages (initial conditions)	.ic

TableB.1 Filename suffixes. (*Note:* # is either a sweep number or a hardcopy file number.)

HSPICE Input Netlist File

Input netlist and library input files can be generated using any standard UNIX editor (vi, emacs, etc.). The order of the statements is arbitrary, except that continuation lines (those beginning with a plus (+) sign) must immediately follow the statement being continued, and the last .ALTER submodule must appear next to the end of the file before the .END statement. Comments may be added any place in the file. The input file name and equation length can be up to 256 characters. A summary of formatting rules is listed below: HSPICE uses a free-format input. Fields in a statement are separated by one or more blanks, tabs, a comma, an equal sign, or a left or right parenthesis. Upper and lower case is ignored except as filenames on UNIX systems. Statement length is limited to 256 characters. A statement may be continued by entering a plus (+) sign as the first non-numeric, nonblank character in the subsequent statement. ALL statements, including QUOTED strings such as paths and algebraics can be continued with a backslash (\) or a double backslash (\ \) at the end of the line to be continued. The single backslash preserves white space and the double squeezes out any white space between the continued lines. The double backslash guarantees pathnames are joined without interruption. Note: input lines can be 256 characters long, so folding and continuing a line is only necessary to improve readability.

HSPICE Input File Structure

The basic structure of an input netlist file consists of one main program and one or more optional submodules. The submodule (preceded by the .ALTER statement) can be used to easily alter and re-simulate an input netlist file with different options, netlist, analysis statements, and test vectors. Several high level call statements can be used to restructure the input netlist file modules. These are the
.INCLUDE, **.LIB** and **.DELLIB** statements. Using these statements, netlists, model parameters, test vectors, analysis, and option macros can be called into a file from either library files or other files. The input netlist file can also call an external data file. The external data file contains parameterized data for element sources and models. The basic elements of an input netlist file are:

TITLE	Implicit first line; becomes input netlist file title
** or $*	Comments to describe the circuit
.OPTIONS	Set conditions for simulation
ANALYSIS and TEMPERATURE	Statements to set sweep variables
PRINT / PLOT / GRAPH	Statements to set print, plot and graph variables
.IC	Sets input state, also can be put in subcircuits
SOURCES	Sets input stimulus
NETLIST	Circuit description
< .PROTECT>	Turns off output printback
.LIB *libraries*	Include .MODEL or .MACRO libraries
.INCLUDE *libraries*	Include .MODEL or .MACRO libraries
<.UNPROTECT>	Restores output printback
.ALTER	Sequence for worst case corners analysis
.DELETE LIB	Removes previous library selection
.LIB	Adds a new library case
.ALTER	Sequence for in-line case analysis
NETLIST	
PARAMETER redefinition	
ADDITIONAL COMMANDS	
.END	Terminates any ALTERs and the simulation

Table B.2 Basic elements of HSPICE netlist

Output Listing File

The results of each circuit simulation are saved in an output listing file with the same filename as the input but appended with a '.lis' suffix instead of '.hsp'. For example, the input file rcnet.hsp/cir would have an output listing file named rcnet.lis. The output listing file contains the simulation results specified by the .PLOT, .PRINT, and analysis statements in the input netlist files. If the input netlist file contains more than one simulation run (by use of the .ALTER, .INCLUDE, .DATA, or analysis statements), the output listing file also contains the results for each simulation run.

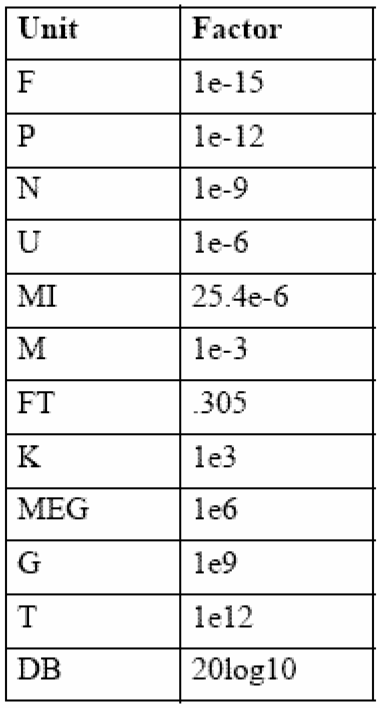
Graph Data File

Graph data files contain high resolution simulation results which can be viewed using a waveform viewer like Avanwaves. When .OPTION POST is included in the input netlist file, HSPICE produces a graph data file. By default it contains all the simulation's node voltages, branch currents, and internal state variables. One graph data file is created for each analysis specified in the input netlist file. Each file will be named by appending a suffix "XX#" to the design name, where "XX" denotes an analysis ("TR" – transient, "SW" – DC sweep, and "AC" – AC) and # is the simulation number for the given analysis. For example, rcnet.ac0 and rcnet.ac1 are generated if an input netlist file specifies an AC analysis for two different temperatures.

Scale Factor Notation and Units

Any letters that are not scale factors and immediately follow an entry number are ignored, with the exception of O or I. If an O or I follows a number, a fatal error results. The letters O and I are not allowed in alphanumeric numbers because they are easily confused with the numbers 0 and 1. The same unscaled number is represented by 10, 10amps, 10V,

10Volts, and 10Hz. The same scale factor is represented by M, MA, MSEC, and MMHOS. The same number is represented by 1000, 1000.0, 1000Hz, le3, 1.0e3, 1KHz.
Note: Scale factors are not accumulative as with other simulators (for example, 1KK does not equal 1MEG).



Unit	Factor
F	1e-15
P	1e-12
N	1e-9
U	1e-6
MI	25.4e-6
M	1e-3
FT	.305
K	1e3
MEG	1e6
G	1e9
T	1e12
DB	20log10

Fig B.3 Scale Factors

Algebraic Expressions

Any parameter defined in the netlist can be replaced by an algebraic expression with single quoted strings. These expressions can then be used as output variables in the .PLOT, .PRINT, and .GRAPH statements. The algebraic expressions greatly expand the user's options in creating an input netlist file. Important features of algebraic expressions are: Scaling or changing of element and model parameters

Parameterization

.PARAM x=5

Algebra

.PARAM x='y+3'

Functions

.PARAM rho(leff,weff)='x*leff*weff-2u'

Hierarchical subcircuit algebraic parameter passing.

```
subckt
inv in out wp=10u wn=5u qbar-ic=vdd
.ic
qbar=qbar-ic
…
.ends
```

Algebra in elements

```
R1 1 0 r='abs(v(1)/i(m1))+10'
```

Algebra in **.MEASURE** statements

.MEAS vmax MAX V(1)
.MEAS imax MAX I (q2)
.MEAS ivmax PARAM= ' vmax* imax'

Algebra in output statements

.print conductance=PAR(' i (ml)/v(22 ')

In addition to simple arithmetic operations (+, -, *,/), HSPICE also accepts the following quoted string functions:

sin(x)	pwr(x, y)	max(x,x)
cos(x)	sinh(x)	sqrt(x)
tan(x)	abs(x)	db(x)
atan(x)	cosh(x)	log10(x)
exp(x)	min(x,x)	
log(x)	tanh(x)	

Table B.4 HSPICE Functions

The Basic Components

Resistors

```
Rname N+ N- Value
```

- N+ represents the positive terminal, N− represents the negative terminal.
- Value is the resistance value
- The ultimate in simplicity.

Inductors and Capacitors

```
Cname N+ N- Value <IC=Initial Condition>
Lname N+ N- Value <IC=Initial Condition>
```

- IC is the initial condition (DC voltage for capacitors or DC current for inductors). It is optional
- The symbol < > means that the field is optional. If not specified, it is assumed to be zero. In case of an inductor, the current flows from N+ to N- .

Voltage and Current Sources:

Independent DC Sources

Voltage source: `Vname N+ N- <DC=> DCValue`

Current source: `Iname N+ N- <DC=> DCValue`

- `N+` is the positive terminal
- `N–` is the negative terminal
- `DCValue` gives the value of the source
- The name of a voltage and current source must start with V and I, respectively.

Independent AC Sources

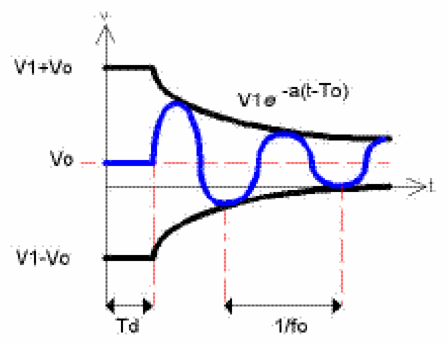
Voltage source: `Vname N+ N- AC ACValue, Phase`

Current source: `Iname N+ N- AC ACValue, Phase`

- `N+` is the positive terminal
- `N–` is the negative terminal
- `ACValue` gives the value of the source
- `ACPhase` gives the phase in degrees
- The name of a voltage and current source must start with V and I, respectively.

Transient Sources

Sinusoidal

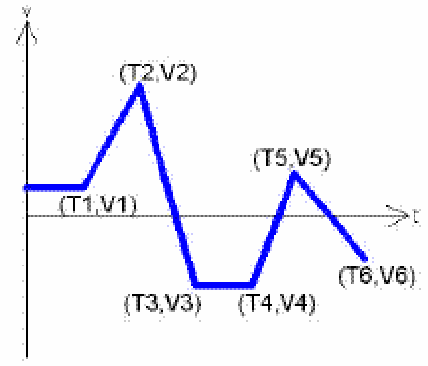


`Vname N+ N- SIN(VO VA fo <TD> <a> <PHASE>)`

- `Vname` = VO + VA exp[-a.(t - TD)] sin[2pi.f (t - TD) + (PHASE/360)]
- `VO` - offset voltage in volts.
- `VA` - amplitude in volts.
- `fo` -the frequency in hertz.
- `TD` - delay in seconds
- `a` - damping factor per second
- `Phase` - phase in degrees

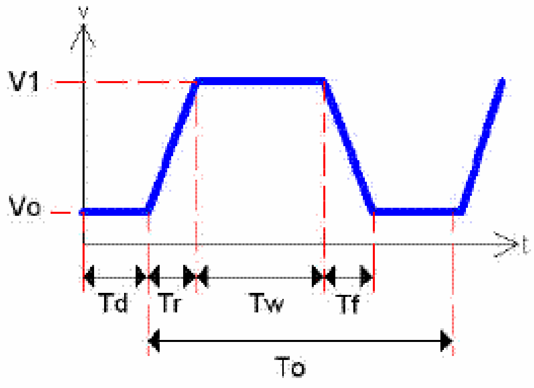
  (If TD, a and PHASE are not specified, it is assumed to be zero.)
- A cosine can be generated by shifting a sin by 90 degrees

Piece-Wise Linear



`Vname N+ N- PWL(T1 V1 T2 V2 T3 V3 ... Tn Vn)`
• (`Ti Vi`) specifies the value `Vi` of the source at time `Ti`

Pulse



`Vname N+ N- PULSE(Vo V1 Td Tr Tf Tw To)`
• `Vo` - initial voltage • `V1` - peak voltage • `Td` - initial delay time • `Tr` - rise time • `Tf` - fall time • `Tw` - pulse-width • `To` - period of wave

Dependent Sources

Voltage controlled voltage source: `Ename N+ N- NC1 NC2 Value`

Voltage controlled current source: `Gname N+ N- NC1 NC2 Value`

Current controlled voltage source: `Hname N+ N- Vcontrol Value`

Current controlled current source: `Fname N+ N- Vcontrol Value`

- `N+` and `N-` are the positive and negative terminals of the dependent source, respectively.
- `NC1` and `NC2` are the positive and negative terminals of the controlling voltage source, respectively.
- `Vcontrol` is the zero value voltage source used to measure the controlling current (the positive current flows into the positive terminal of the controlling voltage source!).
- `Value` is a multiplier value. In math terms: `Value*(NC1-NC2)=(N+ – N-)`

Diode

Element line (What appears in your circuit)

```
Dname N+ N- MODName
```

Model statement: (Appears in your deck to describe circuit element)

```
.MODEL MODName D (IS= N= Rs= CJO= Tt= BV= IBV=)
```

```
.model D1N4148 D (IS=0.1PA, RS=16 CJO=2PF TT=12N BV=100 IBV=0.1PA)
```

- The element name starts with D to indicate that the element is a diode,
- `N+` is the positive end and `N-` is the negative end (where the arrow points)
- `MODName` is the name of the model of the diode specified in the model line.
- `IS` - saturation current, (default=1E-14A),
- `N` - the emission coefficient, (=1),
- `RS` - the series resistance, (=0 ohm),
- `CJO` - junction capacitance, (=0F),
- `TT` - transit time, (=0sec),
- `BV` - reverse bias breakdown voltage, (=infinite) and
- `IBV` - the reverse bias breakdown current, (=1xE-10A).

If a parameter is not specified the default value (given in parenthesis) is assumed.

Bipolar Transistor

Element: (BASIC MODEL)

```
Qname C B E BJT_modelName
```

Model statement:

```
.MODEL BJT_modName NPN (BF=val IS=val VAF=val)
.MODEL BJT_modName PNP (BF=val IS=val VAF=val)
```

```
.model Q2N2222A NPN (IS=14.34F XTI=3 EG=1.11 VAF= 74.03 BF=255.9
NE=1.307 ISE=14.34F IKF=.2847 XTB=1.5 BR=6.092 NC=2 ISC=0 IKR=0 RC=1
CJC=7.306P MJC=.3416 VJC=.75 FC=.5 CJE=22.01P MJE=.377 VJE=.75
TR=46.91N TF=411.1P ITF=.6 VTF=1.7 XTF=3 RB=10)
```

- BF is the common emitter current gain ß,
- IS is the saturation current
- VAF is the Early voltage.
- If no values are specified, the default values are assumed (ß=100; IS=1E-16A, and VAF=[infinite]).
- As an example, the model parameters for the 2N2222A NPN transistor is given above. As you can see, there are many other parameters that can be specified. A more detailed list can be found in the SPICE manual

MOSFET

Element: (BASIC MODEL)

```
Mname ND NG NS <NB> ModName <L=VAL> <W=VAL>

Model:
.MODEL ModName NMOS (<LEVEL=val> <keyname=val> ... )
.MODEL ModName PMOS (<LEVEL=val> <keyname=val> ... )
```

- Like the BJT model, one can use a simplified circuit call, and simply specify width and length. The model being called will have additional parameters already specified. In this class, you will not have to worry about the modeling parameters.

Defining a subcircuit

A subcircuit is defined by a .SUBCKT control statement, followed by the circuit description as follows:

.SUBCKT SUBNAME N1 N2 N3 ...
Element statements
.
.
.
.ENDS SUBNAME

in which SUBNAME is the subcircuit name and N1, N2, N3 are the external nodes of the subcircuit. The external nodes cannot be 0. The node numbers used inside the subcircuit are strictly local, except for node 0 which is always global.

Using a subcircuit

The element statement for a subcircuit is similar to any other element. The format is as follows:

```
Xname N1 N2 N3 ... SUBNAME
```

in which Xname refers to the element (subcircuit) being used; N1, N2, N3 are the nodes to which the external nodes of the subcircuit are being connected, and SUBNAME is the name of the subcircuit being used. An example of an inverting opamp circuit using the subcircuit of the uA741 (see operational amplifiers above) is given below. The subcircuit is called x1.

```
vs 1 0 dc 5
r1 1 2 200
rf 2 3 1k
x1 0 2 3 opamp741
.dc vs 0 10 1
.plot dc v(3)
.end
```

SPECIFYING ANALYSIS: CONTROL STATEMENTS

By now you should have a basic understanding of the vocabulary SPICE uses to describe the physical circuit. Now we turn to analysis.

.OP Statement

DESCRIPTION:

When you include an .OP statement in an input file, HSPICE calculates the DC operating point of the circuit. You can also use the .OP statement to produce an operating point, during a transient analysis. You can include only one .OP statement in a simulation. If an analysis requires calculating an operating point, you do not need to specify the .OP statement; HSPICE calculates an operating point. If you use a .OP statement, and if you include the UIC keyword in a .TRAN analysis statement, then simulation omits the time = 0 operating point analysis, and issues a warning in the output listing.

SYNTAX:

.OP <*format*> <*time*> <*format*> <*time*>... <*interpolation*>

Command Argument	Definition
format	Any of the following keywords. Only the first letter is required. Default = ALL • ALL: Full operating point, including voltage, currents, conductances, and capacitances. This parameter outputs voltage/current for the specified time. • BRIEF: Produces a one-line summary of each element's voltage, current, and power. Current is stated in milliamperes, and power is in milliwatts. • CURRENT: Voltage table, with a brief summary of element currents and power. • DEBUG: Usually invoked only if a simulation does not converge. Debug prints back the non-convergent nodes, with the new voltage, old voltage, and the tolerance (degree of non-convergence). It also prints back the non-convergent elements, with their tolerance values. • NONE: Inhibits node and element print-outs, but performs additional analysis that you specify. • VOLTAGE: Voltage table only. The preceding keywords are mutually- exclusive; use only one at a time.
time	Place this parameter directly after ALL, VOLTAGE, CURRENT, or DEBUG. It specifies the time at which HSPICE prints the report.
interpolation	Selects an interpolation method for **.OP** time points during transient analysis, or no interpolation. Only the first character is required; that is, typing *i* has the same effect as typing *interpolation*. Default is not active. If you specify *interpolation*, all of the time points in the **.OP** statement (except time=0) use the interpolation method to calculate the OP value during the transient analysis. If you use this keyword, it must be at the end of the **.OP** statement. HSPICE ignores any word after this keyword.

EXAMPLE 1:

.OP .5NS CUR 10NS VOL 17.5NS 20NS 25NS

This example calculates:

• Operating point voltages and currents, for the DC solution.
• Currents at 10 ns, for the transient analysis.
• Voltages at 17.5 ns, 20 ns and 25 ns, for the transient analysis.

EXAMPLE 2:

.OP

This example calculates the complete DC operating point solution. The next section shows a printout of the solution.

.DC Statement

DESCRIPTION:

You can use the **.DC** statement in DC analysis, to:

• Sweep any parameter value.
• Sweep any source value.
• Sweep temperature range.
• Perform a DC Monte Carlo (random sweep) analysis.
• Perform a data-driven sweep.
• Perform a DC circuit optimization, for a data-driven sweep.
• Perform a DC circuit optimization, using start and stop.
• Perform a DC model characterization.

SYNTAX:

Sweep or Parameterized Sweep:

.DC *var1* START = *start1* STOP = *stop1* STEP = *incr1*
.DC *var1* START = *<param_expr1>*
+ STOP = *<param_expr2>* STEP = *<param_expr3>*
.DC *var1 start1 stop1 incr1*
+ <SWEEP *var2 type np start2 stop2*>
.DC *var1 start1 stop1 incr1 <var2 start2 stop2 incr2>*

Data-Driven Sweep:

.DC *var1 type np start1 stop1* <SWEEP DATA = *datanm*>
.DC DATA = *datanm*<SWEEP *var2 start2 stop2 incr2*>
.DC DATA = *datanm*

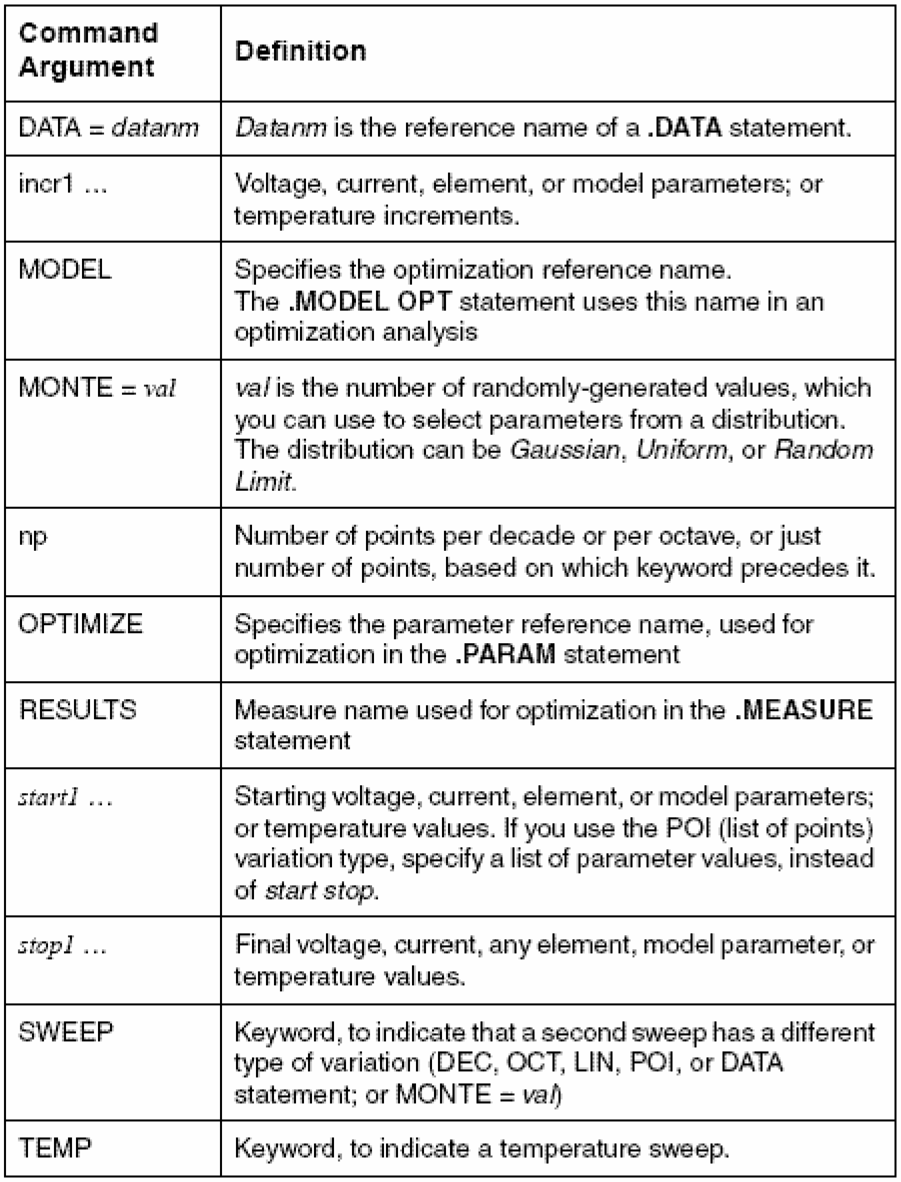
Monte Carlo:

.DC *var1 type np start1 stop1* <SWEEP MONTE = *val*>
.DC MONTE = *val*

Optimization:

.DC DATA = *datanm* OPTIMIZE = *opt_par_fun*
+ RESULTS = *measnames* MODEL = *optmod*
.DC *var1 start1 stop1* SWEEP OPTIMIZE = OPT*xxx*
+ RESULTS = *measname* MODEL = *optmod*

The format for the **.DC** statement depends on the application that uses it.



Command Argument	**Definition**
DATA = *datanm*	*Datanm* is the reference name of a **.DATA** statement.
incr1 ...	Voltage, current, element, or model parameters; or temperature increments.
MODEL	Specifies the optimization reference name. The **.MODEL OPT** statement uses this name in an optimization analysis
MONTE = *val*	*val* is the number of randomly-generated values, which you can use to select parameters from a distribution. The distribution can be *Gaussian*, *Uniform*, or *Random Limit*.
np	Number of points per decade or per octave, or just number of points, based on which keyword precedes it.
OPTIMIZE	Specifies the parameter reference name, used for optimization in the **.PARAM** statement
RESULTS	Measure name used for optimization in the **.MEASURE** statement
start1 ...	Starting voltage, current, element, or model parameters; or temperature values. If you use the POI (list of points) variation type, specify a list of parameter values, instead of *start stop*.
stop1 ...	Final voltage, current, any element, model parameter, or temperature values.
SWEEP	Keyword, to indicate that a second sweep has a different type of variation (DEC, OCT, LIN, POI, or DATA statement; or MONTE = *val*)
TEMP	Keyword, to indicate a temperature sweep.

EXAMPLE 1:
.DC VIN 0.25 5.0 0.25
This example sweeps the value of the VIN voltage source, from 0.25 volts to 5.0 volts, in increments of 0.25 volts.

EXAMPLE 2:
.DC VDS 0 10 0.5 VGS 0 5 1
2-32
Commands in HSPICE Netlists: .DC
This example sweeps the drain-to-source voltage, from 0 to 10 V, in 0.5 V increments, at VGS values of 0, 1, 2, 3, 4, and 5 V.

EXAMPLE 3:
.DC TEMP -55 125 10
This example starts a DC analysis of the circuit, from -55°C to 125°C, in 10°C increments.

EXAMPLE 4:
.DC TEMP POI 5 0 30 50 100 125
This script runs a DC analysis, at five temperatures: 0, 30, 50, 100, and 125°C.

.AC Statement

DESCRIPTION:
You can use the **.AC** statement in several different formats, depending on the application, as shown in the examples below. You can also use the **.AC** statement to perform data driven analysis in HSPICE. If the input file includes an **.AC** statement, HSPICE runs AC analysis for the circuit, over a selected frequency range, for each parameter in the second sweep. For AC analysis, the data file must include at least one independent AC source element statement (for example, VI INPUT GND AC 1V). HSPICE checks for this condition, and reports a fatal error if you did not specify such AC sources.

SYNTAX:
Single/Double Sweep
.AC *type np fstart fstop*
.AC *type np fstart fstop* <SWEEP *var* <START=>*start*
+ <STOP=>*stop* <STEP=>*incr*>
.AC *type np fstart fstop* <SWEEP *var type np start stop*>
.AC *type np fstart fstop*
+ <SWEEP *var* START="*param_expr1*"
+ STOP="*param_expr2*" STEP="*param_expr3*">
.AC *type np fstart fstop* <SWEEP *var start_expr*
+ *stop_expr step_expr*>

Sweep Using Parameters
.AC *type np fstart fstop* <SWEEP DATA = *datanm*>
.AC DATA = *datanm*
.AC DATA = *datanm* <SWEEP *var* <START=>start
<STOP=>stop
+ <STEP=>*incr*>
.AC DATA = *datanm* <SWEEP *var type np start stop*>
.AC DATA = *datanm* <SWEEP *var* START="*param_expr1*"
+ STOP="*param_expr2*" STEP="*param_expr3*">
.AC DATA = *datanm* <SWEEP *var start_expr stop_expr*
+ *step_expr*>

Command Argument	**Definition**
DATA = *datanm*	Data name, referenced in the **.AC** statement
incr	Increment value of the voltage, current, element, or model parameter. If you use *type* variation, specify the *np* (number of points) instead of *incr*.
fstart	Starting frequency. If you use POI (list of points) type variation, use a list of frequency values, not fstart fstop.
fstop	Final frequency.
MONTE = val	Produces a number (*val*) of randomly-generated values. HSPICE uses these values to select parameters from a distribution, either *Gaussian*, *Uniform*, or *Random Limit*.
np	Number of points, or points per decade or octave, depending on which keyword precedes it.
start	Starting voltage or current, or any parameter value for an element or model.
stop	Final voltage or current, or any parameter value for an element or a model.
SWEEP	This keyword indicates that the **.AC** statement specifies a second sweep.
TEMP	This keyword indicates a temperature sweep
type	Can be any of the following keywords: • DEC – decade variation. • OCT – octave variation. • LIN – linear variation. • POI – list of points.

EXAMPLE 1:
.AC DEC 10 1K 100MEG
This example performs a frequency sweep, by 10 points per decade, from 1 kHz to 100 MHz.

EXAMPLE 2:
.AC LIN 100 1 100HZ
This example runs a 100-point frequency sweep from 1 Hz to 100 Hz.

EXAMPLE 3:
.AC DEC 10 1 10K SWEEP cload LIN 20 1pf 10pf
This example performs an AC analysis, for each value of cload. This results from a linear sweep of cload between 1 pF and 10 pF (20 points), sweeping the frequency by 10 points per decade, from 1 Hz to 10 kHz.

.TRAN Statement

DESCRIPTION:

.TRAN starts a transient analysis, which simulates a circuit at a specific time.

Command Argument	**Definition**
DATA = datanm	Data name, referenced in the **.TRAN** statement.
MONTE = val	Produces a specified number (*val*) of randomly-generated values. HSPICE uses them to select parameters from a *Gaussian*, *Uniform*, or *Random Limit*.
np	Number of points, or number of points per decade or octave, depending on what keyword precedes it.
param_expr...	Expressions you specify: *param_expr1...param_exprN*.
pincr	Voltage, current, element, or model parameter; or any temperature increment value. If you set the *type* variation, use *np* (number of points), not *pincr*.
pstart	Starting voltage, current, or temperature; or any element or model parameter value. If you set the *type* variation to POI (list of points), use a list of parameter values, instead of *pstart pstop*.
pstop	Final voltage, current, or temperature; or element or model parameter value.
START	Time when printing or plotting begins. The START keyword is optional: you can specify a start time without the keyword. If you use **.TRAN** with **.MEASURE**, a non-zero START time can cause incorrect **.MEASURE** results. Do not use non-zero START times in **.TRAN** statements, when you also use **.MEASURE**.
SWEEP	Keyword. Indicates that **.TRAN** specifies a second sweep.
tincr1…	Specifies the printing or plotting increment for printer output, and the suggested computing increment for post-processing.

Command Argument	Definition
tstop1...	Time when a transient analysis stops incrementing by the first specified time increment (*tincr1*). If another tincr-tstop pair follows, analysis continues with a new increment.
UIC	Uses the nodal voltages specified in the .IC statement (or in the *IC =* parameters of the various element statements) to calculate initial transient conditions, rather than solving for the quiescent operating point.
type	Specifies any of the following keywords: • DEC – decade variation. • OCT – octave variation (the value of the designated variable is eight times its previous value). • LIN – linear variation. • POI – list of points.
var	Name of an independent voltage or current source, any element or model parameter, or the TEMP keyword (indicating a temperature sweep). You can use a source value sweep, referring to the source name (SPICE style). However, if you specify a parameter sweep, a .DATA statement, and a temperature sweep, you must choose a parameter name for the source value, and subsequently refer to it in the **.TRAN** statement. The parameter name must not start with V or I.
firstrun	The *val* value specifies the number of Monte Carlo iterations to perform. The *firstrun* value specifies the desired number of iterations. HSPICE runs from num1 to num1+val-1.
list	The iterations at which HSPICE performs a Monte Carlo analysis. You can write more than one number after *list.* The colon represents "from ... to ...". Specifying only one number makes HSPICE run at only the specified point.

SYNTAX:

Single-Point Analysis

.TRAN *tincr1 tstop1 <tincr2 tstop2 ...tincrN tstopN>*
+ <START = *val*> <UIC>

Double-Point Analysis

.TRAN *tincr1 tstop1 <tincr2 tstop2 ...tincrN tstopN>*
+ <START = *val*> <UIC>
+ <SWEEP *var type np pstart pstop*>
.TRAN *tincr1 tstop1 <tincr2 tstop2 ...tincrN tstopN>*
+ <START = *val*> <UIC>
+ <SWEEP *var* START="*param_expr1*"
+ STOP="*param_expr2*"
+ STEP="*param_expr3*">
.TRAN *tincr1 tstop1 <tincr2 tstop2 ... tincrN tstop*N>

+ <START=*val*> <UIC>
+ <SWEEP *var start_expr stop_expr step_expr*>

Data-Driven Sweep

.TRAN DATA = *datanm*
.TRAN *tincr1 tstop1* <*tincr2 tstop2* ...*tincrN tstopN*>
+ <START = *val*> <UIC> <SWEEP DATA = *datanm*>
.TRAN DATA = *datanm*<SWEEP *var type np pstart pstop*>
.TRAN DATA=*datanm* <SWEEP *var* START="*param_expr1*"
+STOP="*param_expr2*" STEP="*param_expr3*">
.TRAN DATA=*datanm*
+ <SWEEP *var start_expr stop_expr step_expr*>

EXAMPLE 1:
.TRAN 1NS 100NS
This example performs and prints the transient analysis, every 1 ns, for 100 ns.

EXAMPLE 2:
.TRAN .1NS 25NS 1NS 40NS START = 10NS
This example performs the calculation every 0.1 ns, for the first 25 ns; and then every 1 ns, until 40 ns. Printing and plotting begin at 10 ns.

EXAMPLE 3:
.TRAN 10NS 1US UIC
This example performs the calculation every 10 ns, for 1 µs. This example bypasses the initial DC operating point calculation. It uses the nodal voltages, specified in the .IC statement (or by IC parameters in element statements), to calculate the initial conditions.

.IC Statement

DESCRIPTION:
Use the **.IC** statement, or the **.DCVOLT** statement, to set transient initial conditions in HSPICE How it initializes depends on whether the **.TRAN** analysis statement includes the UIC parameter. If you specify the UIC parameter in the **.TRAN** statement, HSPICE does not calculate the initial DC operating point, but
directly enters transient analysis. Transient analysis uses the **.IC** initialization values as part of the solution, for timepoint zero (calculating the zero timepoint applies a fixed equivalent voltage source). The **.IC** statement is equivalent to specifying the IC parameter on each element statement, but is more convenient. You can still specify the IC parameter, but it does not have precedence over values set in the **.IC** statement. If you do *not* specify the UIC parameter in the **.TRAN** statement, HSPICE computes the DC operating point solution, before the transient analysis. The node voltages that you specify in the **.IC** statement are fixed, to determine the DC operating point. Transient analysis releases the initialized nodes, to calculate the second and later time points.

Command Argument	Definition
val1 ...	Specifies voltages. The significance of these voltages depends on whether you specify the UIC parameter in the **.TRAN** statement.
node1 ...	Node numbers or names can include full paths, or circuit numbers.

SYNTAX:
.IC V(*node1*) = *val1* V(*node2*) = *val2* ...

EXAMPLE:
.IC V(11) = 5 V(4) = -5 V(2) = 2.2

.TF Statement
The .TF statement instructs HSPICE to calculate the following small signal characteristics:
· the ratio of output variable to input variable (gain or transfer gain)
· the resistance with respect to the input source
· the resistance with respect to the output terminals

.TF OUTVAR INSRC
in which OUTVAR is the name of the output variable and INSRC is the input source.
Example: .TF V(3,0) VIN
The .TF statement can be used to find the Thevenin small signal equivalent resistance. (The Thevenin voltage is given by the node voltage at the open circuit terminal, as a result of the .OP statement).

LOOKING AT YOUR DATA: OUTPUT STATEMENTS

.PRINT and .PLOT

These statements will instruct HSPICE what output to generate. If you do not specify an output statement, HSpice will always calculate the DC operating points. The two types of outputs are the .PRINTs and .PLOTs . A print is a table of data points and a plot is a low-resolution graphical representation. The format is as follows:
.PRINT TYPE OV1 OV2 OV3...
.PLOT TYPE OV1 OV2 OV3...
in which TYPE specifies the type of analysis to be printed or plotted and can be DC, TRAN or AC. The output variables are OV1, OV2 and can be voltage between nodes, the voltage between a node and ground. With currents, you can also specify the currents between nodes and more importantly, the currents running through a particular voltage source, which is useful for power consumption. In addition, you can define the type of output by simply putting a suffix after V or I.

The suffixes are:

- M: Magnitude
- DB: Magnitude in dB (deciBels)
- P: Phase
- R: Real part
- I: Imaginary part

Examples:

```
* Plot the DC voltage between nodes 1 and 2, the voltage at node
* 3, and the current through the voltage source, Vmeas
.PLOT DC V(1,2) V(3) I(Vmeas)
* What does this ask for?
.PRINT TRAN V(3,1) I(Vmeas)
* How about this?
.PLOT AC VM(3,0) VDB(4,2) VM(2,1) VP(3,1) IR(V2)
```

Outputting to Mwaves/Awaves (.OPTION POST)

.PRINT and .PLOT output values to your screen (unless you pipe it into a *.lis or *.lst file). But to really produce the nice plots, you need to use MWaves.
MWaves requires that you place .option post somewhere in your spice deck.
HSPICE will then generate automatically the files that MWaves looks for when creating graphs and charts. If you don't, MWaves will only be able to plot a single point at best.

Searching for a Particular Value (.MEASURE)

.MEASURE is often used in circuit optimization. With it, you can find when a certain event occurs as you sweep various parameters.
You can use .MEASURE for finding:

- Rise, Fall and Time Delay
- Average, RMS, min, max, peak-to-peak and integral
- Find X when Y occurs
- Derivative and Integral Evaluations
- Equation Evaluations
- Relative Error (See Manual for Examples)

.MEASURE is a complex command. To explain all the nuances would make this quite a long and dull read. Instead, we are going to post some sample .MEASURE commands that you might find useful.

Rise and Fall

```
.MEAS TRAN rise TRIG V(1) VAL=.2 RISE=1
+ TARG V(1) VAL=.8 RISE=1
```

Gives the time it takes for node 1 to go from 20% to 80% of the maximum voltage (assuming a max voltage of 1V)

Time Delay

```
.MEAS TRAN tdelay TRIG V(1) VAL=2.5 TD=10n RISE=1
+ TARG V(2) VAL=2.5 FALL=1
```

This command takes a look between two points and calculates the the time delay as a signal pushes the voltage up at first node 1 and then node 2. The crossing does not count unless it lasts longer than 10 ns. It only counts the first time it rises and the first time it falls.

Average (and RMS, MIN, MAX and Peak to Peak)

.MEAS TRAN avgval AVG V(1) FROM=10ns TO=55ns
This takes the average value of node 1 from 10ns to 55ns and outputs it as avgval in the *.lis or *.lst file.

If you replace the keyword AVG with RMS, MIN, MAX and PP HSPICE will calculate that function for the time given.

Find and When

.MEAS TRAN DesiredCurr FIND I(Vmeas) WHEN V(1)=1V
Output to DesiredCurr the current through the votlage supply Vmeas when node 1 reaches 1V.

Derivatives and Integrals

.MEAS TRAN slewrate DERIV V(out) AT=25ns
This calculates the derivative of V(out) at 25ns. Derivative is always calculated with respect to the sweeping parameter (which is time in this case)

Using Equation Evaluations

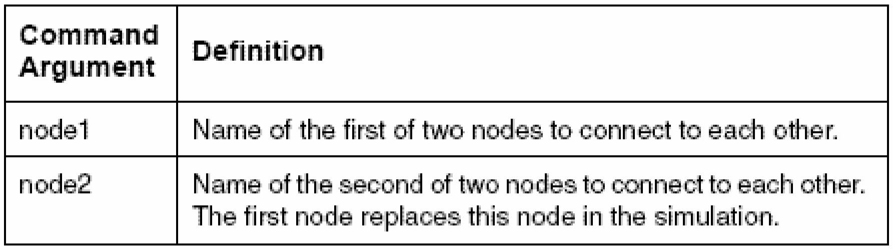
.MEAS TRAN slew DERIV v(1) WHEN v(1)='0.9*vdd'
In this case, we're calculating slew rate when v(1) is equal to 0.9*vdd.

.MEAS AC delay DERIV 'VP(output)/360.0' AT=10kHz
This calculates the delay which is equal to the derivative of the phase normalized by 360 degrees.

Node naming

.CONNECT

DESCRIPTION:
The **.CONNECT** statement connects two nodes in your HSPICE netlist, so that simulation evaluates two nodes as only one node. Both nodes must be at the same level in the circuit design that you are simulating: you cannot connect
nodes that belong to different subcircuits.



Command Argument	Definition
node1	Name of the first of two nodes to connect to each other.
node2	Name of the second of two nodes to connect to each other. The first node replaces this node in the simulation.

SYNTAX:

.CONNECT *node1 node2*

EXAMPLE 1:

```
...
.subckt eye_diagram node1 node2 ...
.connect node1 node2
...
.ends
```

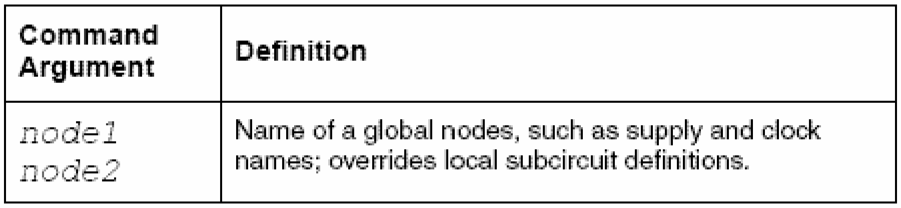
.GLOBAL

DESCRIPTION:

The **.GLOBAL** statement globally assigns a node name, in HSPICE. This means that all references to a global node name, used at any level of the hierarchy in the circuit, connect to the same node. The most common use of a **.GLOBAL** statement is if your netlist file includes subcircuits. This statement assigns a
common node name to subcircuit nodes. Another common use of **.GLOBAL** statements is to assign power supply connections of all subcircuits. For example, **.GLOBAL** VCC connects all subcircuits with the internal node name VCC.
Ordinarily, in a subcircuit, the node name consists of the circuit number, concatenated to the node name. When you use a **.GLOBAL** statement, HSPICE does not concatenate the node name with the circuit number, and assigns only the global name. You can then exclude the power node name in the subcircuit or macro call.

SYNTAX:

.GLOBAL *node1 node2 node3 ...*



Command Argument	Definition
node1 *node2*	Name of a global nodes, such as supply and clock names; overrides local subcircuit definitions.

EXAMPLE:

This example shows global definitions for VDD and input_sig nodes.

```
.GLOBAL VDD input_sig
```

Model Definition

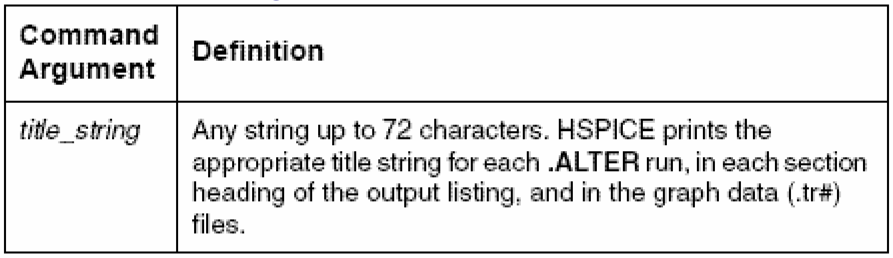
DESCRIPTION:
Use the **.MODEL** command to include an instance (element) of a pre-defined HSPICE model in your input netlist.
Commands in HSPICE Netlists: .MODEL
For each optimization within a data file, specify a **.MODEL** statement. HSPICE can then execute more than one optimization per simulation run. The *.MODEL* optimization statement defines:
• Convergence criteria.
• Number of iterations.
• Derivative methods.

SYNTAX:
.MODEL *mname type* <VERSION = *version_number*>
+ <*pname1* = *val1 pname2* = *val2* ...>
.MODEL *mname* OPT <*parameter*=*val* ...>

EXAMPLE 1:
.MODEL MOD1 NPN BF=50 IS=1E-13 VBF=50 AREA=2 PJ=3,
+ N=1.05

Alter Blocks
DESCRIPTION:
You can use the .ALTER statement to rerun an HSPICE simulation, using different parameters and data. Use parameter (variable) values for print and plot statements, before you alter them. The .ALTER block cannot include .PRINT, .PLOT, .GRAPH or any other input/ output statements. You can include analysis statements (.DC, .AC, .TRAN, .FOUR, .DISTO, .PZ, and so on) in a .ALTER block in an input netlist file. However, if you change only the analysis type, and you do not change the circuit itself, then simulation runs faster if you specify all analysis types in one block, instead of using separate .ALTER blocks for each analysis type. The .ALTER sequence or block can contain:
• Element statements (except source elements)
• .DATA statements
• .DEL LIB statements
• .INCLUDE statements
• .IC (initial condition) and .NODESET statements
• .LIB statements
• .MODEL statements
• .OP statements
• .OPTION statements
• .PARAM statements
• .TEMP statements
• .TF statements
• .TRAN, .DC, and .AC statements
SYNTAX:

.ALTER *<title_string>*



Command Argument	**Definition**
title_string	Any string up to 72 characters. HSPICE prints the appropriate title string for each **.ALTER** run, in each section heading of the output listing, and in the graph data (.tr#) files.

EXAMPLE:
.ALTER simulation_run2

C. HSIM SIMULATOR FEATURES & SYNTAX SUMMARY

HSIM

The HSIM simulator is the core of the HSIMplus platform. HSIM performs transient analysis, DC analysis, AC analysis and Monte Carlo analysis supporting the following circuit elements:

- MOSFET (Metal-Oxide Semiconductor Field-Effect Transistors)
- Bipolar transistors
- Diodes
- Junction field-effect transistors
- Resistors
- Capacitors
- Self and Mutual inductors
- Independent voltage and current sources
- Linear and nonlinear controlled voltage and current sources
- Lossless and lossy transmission lines

Interactive Circuit Analysis

HSIM's interactive circuit analysis provides a circuit debugging environment that interrupts simulation and performs interactive circuit diagnosis at selected points in time. HSIM analysis provides circuit information such as:

- Node voltage
- Node capacitance
- Element current
- Element conductance and capacitance
- Fan-in and fan-out elements to a node
- Element terminal nodes
- Active element drivers to a node
- Active loading elements to a node
- Excessive current checks
- DC path between two nodes

Using HSIM in a Nanometer VLSI Design Flow

A typical nanometer very large system integration (VLSI) design effort can be divided into:

- Pre-layout design flow
- Post-layout design flow

These design flows are discussed in the following sections.

Pre-Layout Design Flow

Pre-layout design flow is designed to create functionally correct designs for layout implementation. Estimated parasitics for the inter-block interconnects are often used for exploring the timing behavior at this early design stage. At the block level illustrated in Figure 2-2, Pre-Layout Design Flow, HSIM shows the use of three design types in a pre-layout flow. A full-chip functionality and timing simulation flow is an extension from the block-level flow. By assembling the netlists of all the individually verified blocks and providing the top-level input stimuli, the fullchip can be simulated without difficulties or complications.

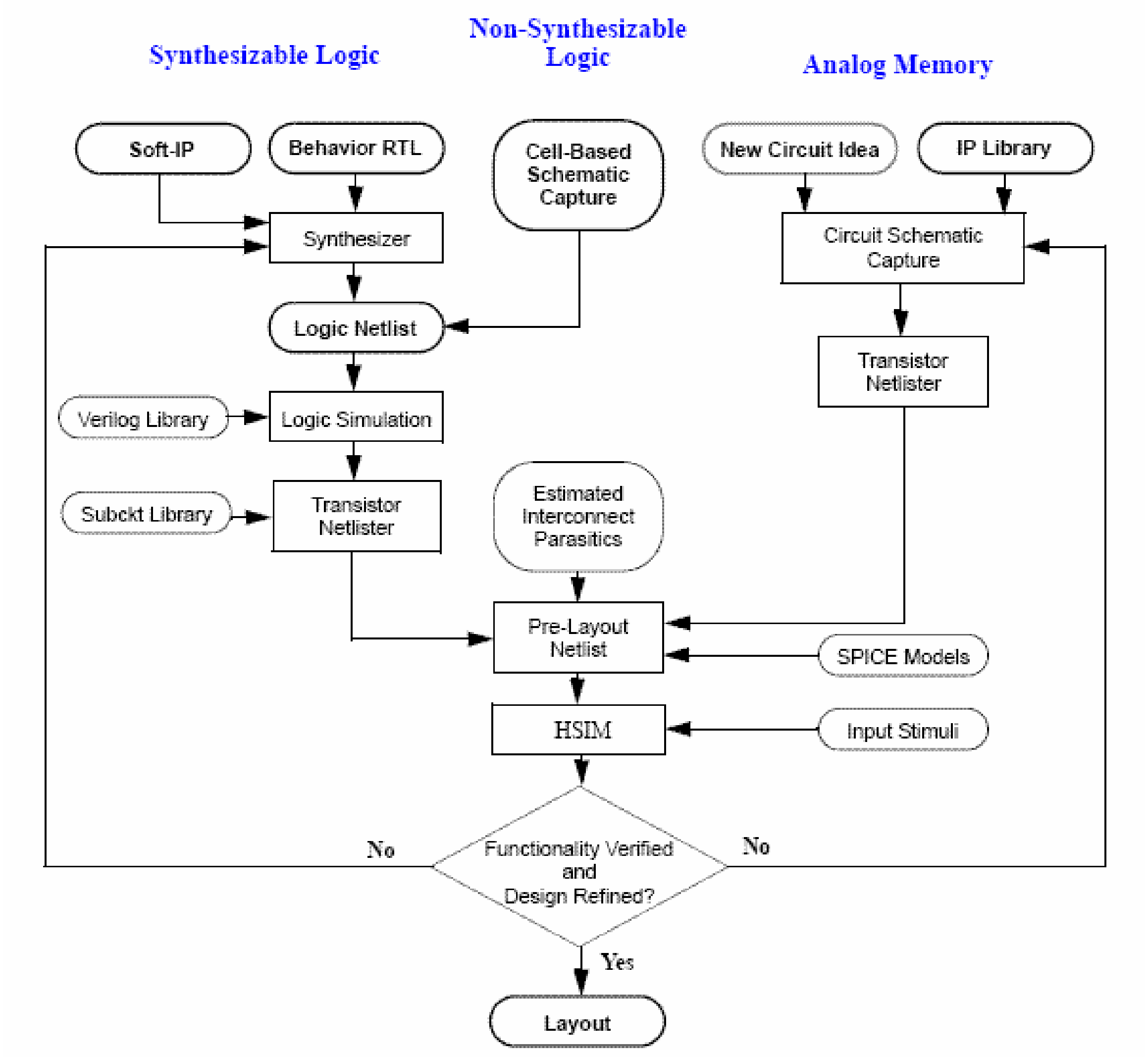


FIGURE C-1. Pre-Layout Design Flow

HSIM provides design flows for the following three design styles:

- Synthesizable Logic
- Non-Synthesizable Logic
- Analog/Memory

Synthesizable Logic Synthesizable logic uses high-speed standard cells for digital design. Simulations can be run to explore cross-talk issues along certain critical paths. Then, the information from this step is used to refine layout strategies that prevent cross-talk problems. Block current obtained through the simulation can be used for power bus sizing. To maximize HSIM's capabilities in the synthesizable logic flow, the sub-circuit library and the estimated interconnect parasitics must be provided. The estimated parasitics are
placed inside the circuit netlist with the other functional elements of the design, and then simulated by HSIM. The sub-circuit library transforms the design from a gate-level representation to a transistor-level representation. The estimated parasitics inject an early physical dimension into a pure logical design in order to study the potential physical effects

within the design. Estimated parasitics may include any of the following: • Long interconnects for a timing behavior assessment, particularly for clock net analysis (jitter and skew).
• Estimated cross-coupling capacitors for a cross-talk assessment
• All possible wire capacitors for a block current assessment

Non-Synthesizable Logic Non-synthesizable logic is often used for high performance application-specific integrated circuit (ASIC) and semi-custom designs. These types of designs typically utilize complex design techniques such as dynamic logic, and are subject to the circuit effects of noise and ground bounce. As logic simulators do *not* provide an accurate description of the behavior of these circuits, HSIM simulation is strongly recommended.
Analog/Memory For analog/memory design styles, HSIM provides speed and accuracy to successfully simulate these circuit types.

Post-Layout Design Flow
The main purpose of the post-layout design flow is to optimize the design by layout refinements and to verify circuit performance in the presence of layout parasitics. In particular, to determine the influence of IR drop and coupling capacitance on design characteristics. An overview of the post-layout design flow is shown in Figure C-2, Post-Layout Design Flow.

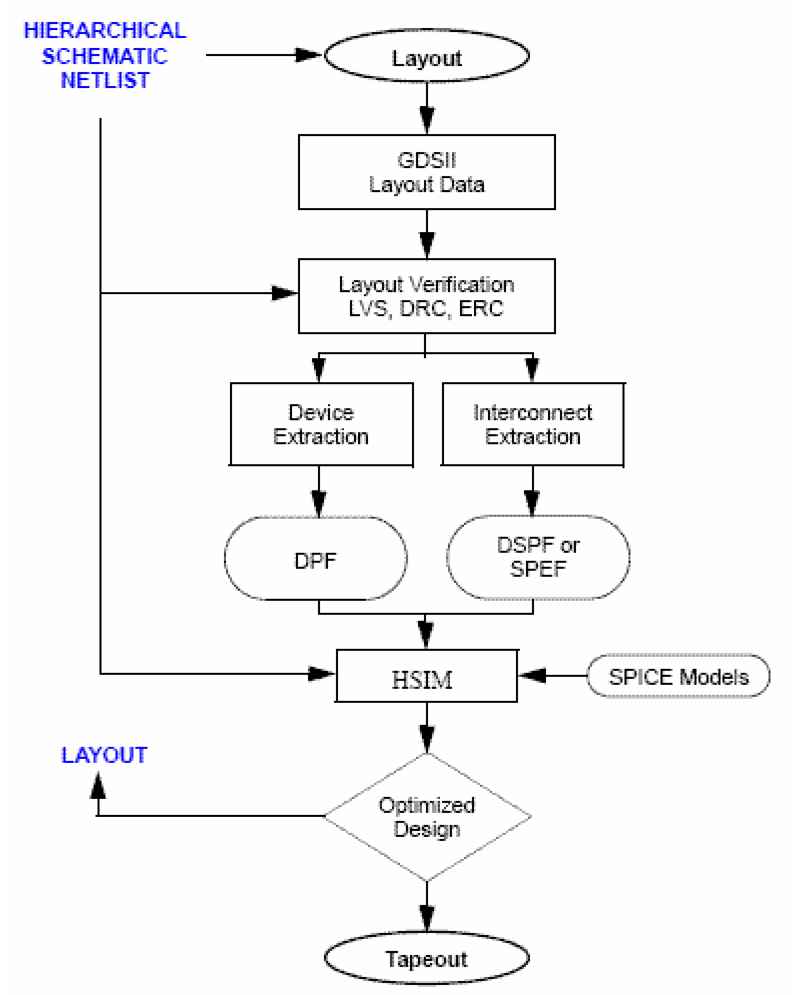


FIGURE C-2 Post-Layout Design Flow

GDSII LAYOUT DATABASE The post-layout flow begins with a GDSII1 layout database. The next stage is physical layout verification to ensure there are no major flaws in the layout, such as:

- DRC
- ERC
- LVS

LAYOUT FUNCTIONAL VERIFICATION

After physical layout verification, the basic functionality of the layout is verified. This is best accomplished by doing the following:

- Extracting the active components of the design.
- Back-annotating DPF to the pre-layout hierarchical netlist

Once this is completed, a top-level full-chip simulation can be performed. Top-level stimuli from the pre-layout flow can be used to ensure that the basic functionality of the extracted netlist has *not* changed.

DESIGN OPTIMIZATION

Once design functionality has been confirmed, layout parasitics can be incorporated into the simulator and the results of the simulation can be used to optimize the design. Optimization goals include:

• Enhancing circuit speed
• Maintaining power consumption within specification
• Increasing the design margin
• Improving circuit reliability and robustness
• Decreasing sensitivity to manufacturing variability

The circuit characteristics to address in this phase include:

• Speed
• Power
• Reliability
• Manufacturability

NOTE: GDSII has no official definition. Unofficially it stands for Graphics Design Station Two because it was the output format from the second generation of Calma design stations.

CIRCUIT EXTRACTION AND ANALYSIS

Enhanced design margins provide greater tolerance against manufacturing fluctuations. Greater circuit reliability ensures a long product life in the field. To refine a layout, a detailed circuit must be extracted and analyzed to determine if and where the circuit needs improvement.

A detailed extracted circuit netlist consists of the following:

• Active circuit devices
• The device's topological connectivity
• Interconnect parasitic for both signal and power nets

Many extraction tools are configured to extract both active devices and interconnect parasitic into a single, flat netlist file. This straightforward approach eliminates any potential problems with issues such as name matching between netlist and DSPF/SPEF files. The one proviso is that HSIM must have sufficient memory available to read-in the entire flat netlist. After the design netlist is parsed, HSIM partitions the design, looking for hierarchy with the available partitioned sub-circuits. This ensures that HSIM utilizes the hierarchy in the design for memory efficiency even if the input netlist is flat. Since HSIM's peak memory usage occurs while parsing the netlist and building the hierarchical simulation structures, if the circuit netlist can be parsed, simulation can usually proceed. This process works well for circuits of small to medium size but full advantage of the hierarchical simulation engine cannot be realized in the same way that it can with a hierarchical circuit of many levels, containing a large number of related small isomorphic instances. Most extraction tools also offer separate netlisting for active devices and interconnect parasitics after extraction is complete. HSIM is designed and implemented to take maximum advantage of this flow and data. Extracting the active devices to Nassda's Device Parameter Format (DPF) and interconnect parasitic to DSPF/SPEF provides for efficient back-annotation into the hierarchical simulation database created after parsing the hierarchical pre-layout netlist.

RESOLVING INTERCONNECT SEGMENTATION

Another issue is the resolution of the interconnect segmentation. When a fine interconnect segmentation resolution is requested, a flat extractor can create a very large number of extracted parasitic elements. Although HSIM uses efficient hierarchical data structures, it is *not* completely immune from the large data volume problems associated with parsing in these large flat netlists. The amount of extracted data must be controlled by selecting the appropriate segmentation resolution in the parasitic extraction process.
NOTE: For a large parasitic database, it is recommended to enable RC reduction features in HSIM.

This will substantially reduce the memory required to store the interconnect parasitics and ensures that the transient simulation proceeds with maximum efficiency. A memory circuit contains the same core cell that composes the majority of the layout. If extracted hierarchically, the netlist size can be reduced to one-fifth the size of the same netlist from a flat extraction. System-on-Chips (SoC) also have memory and other regular structures, although the storage saving might *not* be as great as that of memory circuits.
If the extracted interconnect parasitics can be generated in DSPF/SPEF, then the DSPF back-annotation capability allows a net-by-net RC reduction and instantiation. This feature removes the need to store all layout parasitics because each net is reduced and back-annotated one net at a time. This approach effectively eliminates the typical bottleneck associated with handling large flat parasitic netlists and provides greater circuit simulation capacity.
HSIM share signal net reduction technologies and the parameter settings operate identically in each simulator. For signal nets, HSIMplus offers hierarchical back annotation capability where the reduced signal net can be partitioned and distributed into the pre-layout netlist; preserving the benefits of hierarchical simulation as much as possible. Compared to HSIM's flattening back-annotation, hierarchical back-annotation offers the following benefits for signal nets:
• Small, but *not* insignificant, throughput improvement.
• Large increase in memory efficiency.
HSIMplus's power net capabilities offer a significant enhancement over HSIM. HSIM offers small power net reduction technologies such as RMIN elimination. However HSIM's flattening back-annotation of the globally coupled power nets defeats the hierarchical simulation engine and forces flat simulation. HSIMplus adds two key pieces of innovative technology that enable efficient full-chip post-layout simulation including all power nets:
• Power Net reduction
• Hierarchical power net back-annotation
Power Net reduction has two sub-flows that are applied in sequence:
1. Straight-forward timing-based sub-flow largely preserves the power net topology and applies resistor reduction based on a specified threshold, series merging, and parallel elimination to reduce the number of power net resistors by an order of magnitude.
2. The second sub-flow alters the power net topology and applies user controlled heuristics of varying aggression levels that transform the power net structures into significantly smaller networks.

These reduced power nets can be saved at intermediate steps in the reduction process facilitating quick turnaround times for repeated simulations. The final reduced power

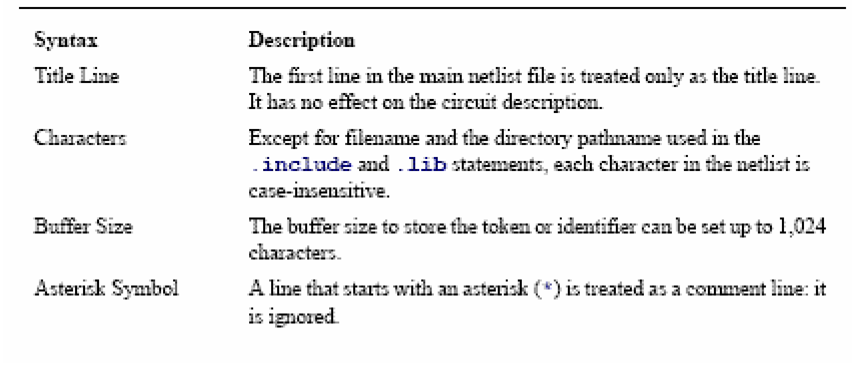
networks are then back-annotated to the hierarchical pre-layout database using HSIMplus's hierarchical back-annotation capability. Circuits that most lend themselves to a hierarchical simulator solution typically have, but are *not* exclusively limited to, small isomorphic instances that are greatly replicated, post-layout, with all signal and power net interconnect parasitics incorporated into the simulator. HSIM can precisely verify circuits of tens or even hundreds of millions of transistors at tape out providing the following benefits:
• Increased confidence in achieving first time simulation success.
• Increased probability of ramping the first design to high volume.
• High and consistent yields.

Netlist Syntax

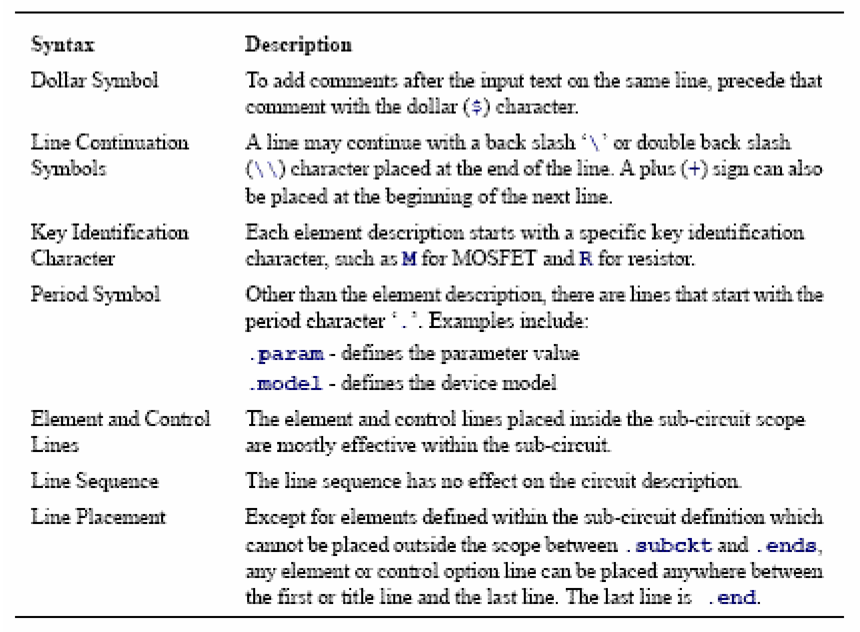
HSIM input netlists contain some or all of the following information:
• Circuit topology description consisting of circuit elements and their connectivity.
• Element models
• Simulation control parameters
• Stimulus input sources and output specifications
These data can be described in either single file or multiple files. When multiple files are involved, an **.include** statement must be used to include main netlist files. Refer to .include syntax.

Netlist Syntax Summary

HSIM netlist syntax can be summarized in Table C-1:



Syntax	Description
Title Line	The first line in the main netlist file is treated only as the title line. It has no effect on the circuit description.
Characters	Except for filename and the directory pathname used in the `.include` and `.lib` statements, each character in the netlist is case-insensitive.
Buffer Size	The buffer size to store the token or identifier can be set up to 1,024 characters.
Asterisk Symbol	A line that starts with an asterisk (*) is treated as a comment line: it is ignored.



Syntax	Description
Dollar Symbol	To add comments after the input text on the same line, precede that comment with the dollar (`$`) character.
Line Continuation Symbols	A line may continue with a back slash '`\`' or double back slash (`\\`) character placed at the end of the line. A plus (`+`) sign can also be placed at the beginning of the next line.
Key Identification Character	Each element description starts with a specific key identification character, such as `M` for MOSFET and `R` for resistor.
Period Symbol	Other than the element description, there are lines that start with the period character ' . '. Examples include: `.param` - defines the parameter value `.model` - defines the device model
Element and Control Lines	The element and control lines placed inside the sub-circuit scope are mostly effective within the sub-circuit.
Line Sequence	The line sequence has no effect on the circuit description.
Line Placement	Except for elements defined within the sub-circuit definition which cannot be placed outside the scope between `.subckt` and `.ends`, any element or control option line can be placed anywhere between the first or title line and the last line. The last line is `.end`.

TABLE C-1 Netlist Syntax Summary

Netlist Differences Between HSIM and SPICE

The input netlist format for HSIM is almost completely compatible with that of SPICE or HSPICE simulators. There are some differences which are described as follows:

• HSIM reads input netlist and performs the circuit simulation, even when the netlist contains elements that are supported by SPICE but are *not* supported by HSIM. Elements that are *not* supported are ignored and warning messages are displayed.

Simulation and Control Statements

.alter

The .alter statement repeats a simulation using alternative parameter values and modified netlist. In each simulation run, the output files, such as hsim.ic, hsim.fsdb, and so on, will have the suffix .a#, where # is the run number.

Example:

```
* first simulation run
.temp 25
......
.alter
.temp 100
.end
```

The circuit temperatures for the two runs are: Run 1; 25° C and Run 2; 100° C.

.data
The **.data** statement allows modifying parameter values in transient simulation. This command is for cell or block characterization and optimization. The group of parameter values is included in the input file.
.data *dlink*
+ param1 <param2 param3 ... paramN>
+ val1a <val2a val3a ... valNa>
+ val1b <val2b val3b ... valNb>
+ ...
.enddata
SYNTAX DEFINITIONS • *dlink dlink* is also used in the **.tran** command
• *param* Parameter names
• *val* Parameter values.
Example:
.tran 0.1n 100n **sweep data=**dlink1
.data dlink1
+ L W CLOAD
+ 0.18u 0.36u 10f
+ 0.13u 0.26u 5f
.enddata

HSIM accepts the following parameter values and performs the first transient simulation:
L=0.18u, W=0.36u, CLOAD=10f
Afterwards, the simulation is repeated for the following:
L=0.13u, W=0.26u, CLOAD=5f

.del param
The **.del param** statement removes selective parameter setting which is especially useful in the .alter section.
.del param *<param1> <param2>* ... <**subckt**=<*subckt_name*>>
For instance, a vector file can be removed from the simulation in the .alter on page 5-58.
Example:
. param HSIMVECTORFILE=vec1
......
......
.alter
.del param HSIMVECTORFILE
.param HSIMVECTORFILE=vec2
......
In this case, the vector file vec1 is removed in the **.alter** section and another file, vec2, is added to the simulation. If the **.del param** statement is *not* used, then both vec1 and vec2 files are included in the simulation in the .alter.

.end
The **.end** statement defines the end of an HSIM run.
.end <comments>

If **.end** is *not* specified in an input netlist, the default is end of file. An input netlist can contain multiple HSIM runs by having an **.end** at the end of each run. In each HSIM run, the output files, such as hsim.ic, hsim.fsdb, and so on, will have the suffix .e#, where # is the run number.

.endl
The **.endl** statement ends the library macro definition.
.endl <lib_entry_name>
Example:
.lib models.**lib** TT
The TT portion of models.**lib** is read in.
.lib TT
.model ck1 nmos **level**=49
... ...
.endl TT
.

.ends
The **.ends** statement specifies the end of a **subckt** definition.
.ends <*subckt_name*>
Example:
.subckt inverter in out
M1 2 1 0 0 nmos1 w=0.36u l=0.18u
M2 2 1 vdd vdd pmos1 w=0.36u l=0.18u
.ends

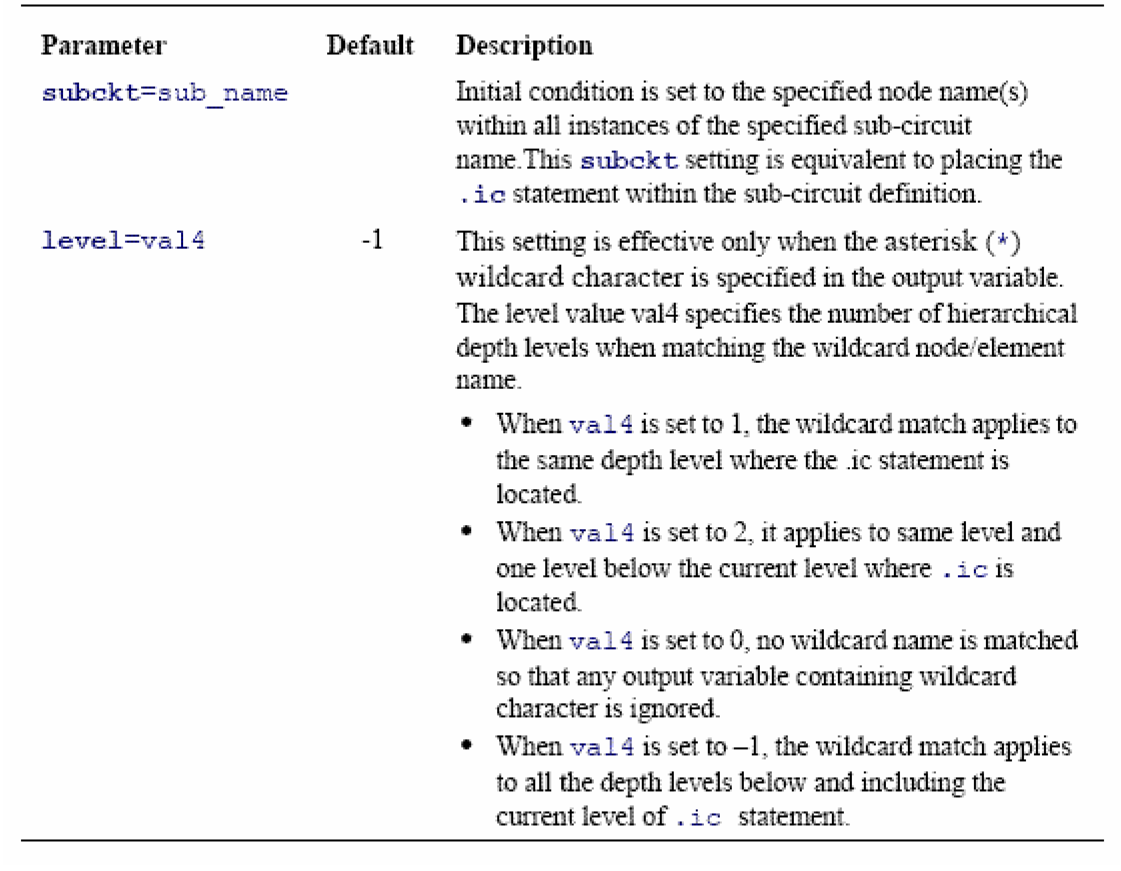
.force
.force node_name voltage_value <**subckt**=subckt_name> <**time**=force_time>
• node_name node_name can be a specific node name or a pattern.
• subckt_name subckt_name is the sub-circuit name. When the node_name sub-circuit parameter is used, the node is inside that subcircuit. Otherwise, node_name is assumed to be a hierarchical name.
• force_time force_time is the starting time that forces the specified node to stay at a specified constant voltage_value. Time unit is seconds.
.force forces node_name to stay at the same voltage_value as force_time until one of the following occurs:
• Simulation ends
• When the constant node voltage status is released by either of the following:
- .release
- rv.
.force does *not* work on voltage source or vector file input nodes. Refer to the following sections for additional information:
Example:
.force pump 6.5 time=100u

.global The **.global** statement defines global nodes.

.global *node1 <node2 ...>*
A global node can be directly referenced from any level of hierarchy. Any node name appearing at the top-level or any low level hierarchy is connected to the same node. Declaring a global node allows a direct reference to an internal sub-circuit node without defining the node in the sub-circuit port list. HSIM recognizes any of the following as the ground node:

- 0 (zero)
- gnd
- gnd!
- ground

.ic
The **.ic** statement defines the initial condition.
.ic v(*node1*)=*val2* <**v**(*node2*)=*val3* ...> <**subckt**=*sub_name*> <**level**=*val4*>
The specified node can be the node name of a single node or a pattern containing asterisk (*) wildcard character that represents a group of nodes matching the pattern. The optional setting of level is specified to control the scope of wildcard match.



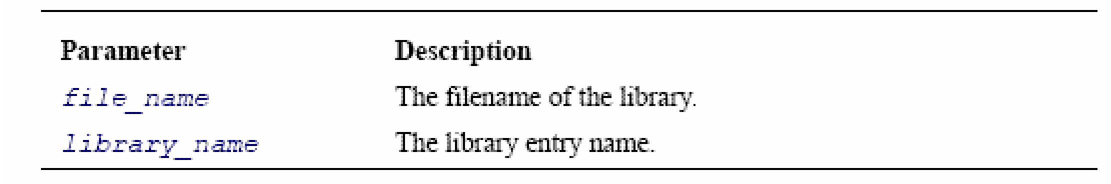
Parameter	Default	Description
`subckt=sub_name`		Initial condition is set to the specified node name(s) within all instances of the specified sub-circuit name.This `subckt` setting is equivalent to placing the `.ic` statement within the sub-circuit definition.
`level=val4`	-1	This setting is effective only when the asterisk (*) wildcard character is specified in the output variable. The level value val4 specifies the number of hierarchical depth levels when matching the wildcard node/element name. • When `val4` is set to 1, the wildcard match applies to the same depth level where the .ic statement is located. • When `val4` is set to 2, it applies to same level and one level below the current level where `.ic` is located. • When `val4` is set to 0, no wildcard name is matched so that any output variable containing wildcard character is ignored. • When `val4` is set to –1, the wildcard match applies to all the depth levels below and including the current level of `.ic` statement.

TABLE C-3 .IC Statement Parameters

Example:
.ic v(1)=1.8 v(4)=3.3 v(x1.x2.*)=2.2
.include The **.include** statement includes another data file.
.include *file_name*

or
.inc *file_name*
Example:
.inc netlist.net

.lib
The **.lib** statement includes the model library.
.lib *file_name library_name*
This is a library call statement which indicates the specified library entry in the library file should be read in.
or
.lib *library_name*
This is a library definition statement that begins the library entry in the library file. The **.endl** indicates the end of the library entry.



Parameter	Description
file_name	The filename of the library.
library_name	The library entry name.

TABLE C-4 .LIB Statement Parameters

Example:
.lib models**.lib** TT
The TT portion of models**.lib** is read in.
.lib TT
.model ck1 nmos **level**=49
... ...
.endl TT
A library is defined.

.malias
The **.malias** statement provides an alias for a model name.
.**malias** *model_name alias_name1 <alias_name2 ...>*
The words alias_name1 is aliased to model_name.

Example:
.**malias** tn013 ma mb
Both model names ma and mb are aliased to tn013.

.nodeset
This statement sets the starting voltage at DC initialization.
.nodeset v(*node1*)=*val2* <v(*node2*)=*val3* ...> <**subckt**=*sub_name*> <**level**=*val4*>
The specified node can be the node name of a single node or a pattern containing a asterisk (*) wildcard character that represents a group of nodes matching the pattern. The optional setting of **level** controls the scope of wildcard match. Refer to .ic.
The optional setting of **subckt** is for the nodes within all instances of the specified subcircuit name; this is equivalent to placing the **.nodeset** statement inside the sub-circuit definition.
Example:
.nodeset v(1)=1.8 v(4)=3.3 v(x1.x2.*)=2.2

.op
The **.op** statement dumps the operating condition at the specified time(s).
.op *<time1> <time2 ...>*
The DC operating point output is partially supported in HSIM. HSIM only creates the node voltages. The DC operating analysis should generate node voltages and element currents at the resulting DC operating point.

HSIMOPCOMPRESS
When **.op** statement is specified in the netlist, HSIM prints each node voltage at the specified time into a file. The format of the output file can be changed by setting HSIMOPCOMPRESS as follows:
- HSIMOPCOMPRESS=0 Regular text format (**.ic** extension) (default setting)
- HSIMOPCOMPRESS=1 **.gz** format
- HSIMOPCOMPRESS=2 **.Z** format

The prefix of the output file is **hsim** by default, but can also be specified on the command line by entering -o file_name. The default value of time is 0 ns.
NOTE: Element currents can be dumped at an operating point using the HSIMDUMPOPI=1 option. The default is 0.

.option
This command defines optional values.
.option *name1=val1 <name2=val2>* **. . .** *<nameN=valN>*
HSIM recognizes popular simulation options described in Table C-5, SPICE Simulation Options Recognized by HSIM.

Parameter	Default	Description
`aspec`		Makes the simulation compatible with `aspec` setting, such as when the `wl` option (see below) is activated. `scale` and `scalm` are set to scale geometry dimension to microns.
`badchr`		Produces a warning if n unprintable characters are detected in the input file.
`defad`	0	The default value for drain-bulk junction diode area in an MOSFET.
`defas`	0	The default value for source-bulk junction diode area in an MOSFET.
`defl`	0	The default value for channel length in an MOSFET.
`defnrd`	0	The default value for the number of squares for MOSFET drain resistor.

Parameter	Default	Description
`defnrs`	0	The default value for the number of squares for MOSFET source resistor.
`defpd`	0	The default value for drain bulk junction diode perimeter in an MOSFET.
`defps`	0	The default value for source bulk junction diode perimeter in an MOSFET.
`defw`	0	Default value for channel width of a MOSFET.
`genk`	1	When assigned the value of 1 (one), `genk` is the control parameter for the automatic calculation of second-order mutual inductance for multiple coupled inductors.
`klim`	0.01	Minimum mutual inductance threshold for calculation of second-order mutual inductance.
`nowarn`		Suppress warning messages.
`Parhier=global`		A parameter name specified at a higher hierarchical level which prevails over the same parameter name specified at a lower level.
`Parhier=local`	`global`	A parameter name specified inside a sub-circuit which prevails over the same parameter name specified at a higher hierarchical level.
`scale`	1.0	Element scale parameter. It scales the geometry of the transistor instance.
`scalm`	1.0	Model scaling factor. Relevant model parameters are scaled by this value.
`search=dir_path`		Sets the search path for model libraries and included files. The HSIM searches the specified directory for libraries used in the simulation.
`spice`		Makes the simulation compatible with SPICE setting, such as: • `tnom=27` • `defnrd=1` • `defnrs=1`
`tnom`	25 or 27	Simulation reference temperature. The default value is 25 degrees C unless `.option` SPICE is specified; then the default is 27° C.
`warnlimit=y`		Limits the number of warnings to appear in the `.log` file. Value `y` is the total number of warnings allowed for each warning type.
`wl`		MOSFET element geometry order is changed from (defaulted) length-width to width-length.

TABLE C-5 .OPTION Parameters

.param
The **.param** statement defines parameters.
.param *name1=val1 <name2=val2 ...>*
Examples:
1. .param a=2 b=4
2. .param c='b+3*a'

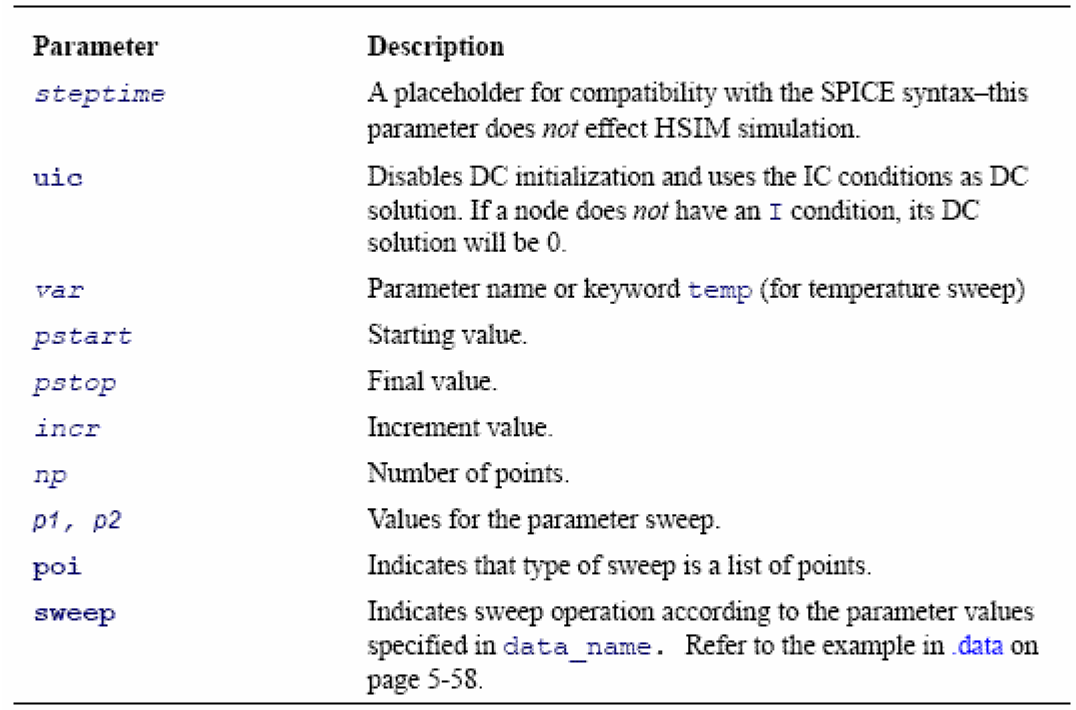
.release
.release node_name <**subckt**=subckt_name> <**time**=release_time>
• node_name node_name can be a specific node name or a pattern.
• subckt_name subckt_name is the sub-circuit name. When the node_name sub-circuit parameter is used, it is the node inside that subcircuit. Otherwise, node_name is assumed to be a hierarchical name.
• release_time release_time is the starting time that releases the specified node. Time unit is seconds. **.release** releases the node voltage from the value fixed by the **.force** command or by the interactive **rv** command. The node voltage are then determined by the regular simulation result.

.subckt
The **.subckt** statement defines a sub-circuit.
.subckt *sub_name term1 <term2 ...> <parameter1=val1> <parameter2=val2 ... >*
Example:
.subckt inverter in out
M1 2 1 0 0 nmos1 w=0.36u l=0.18u
M2 2 1 vdd vdd pmos1 w=0.36u l=0.18u
.ends

Sub-circuit Instance The sub-circuit instance is defined below:
Xaa term1 <term2 ...> sub_name <parameter1=val2> <parameter2=val3>
Example:
Xinv1 1 2 inverter

.temp
The **.temp** statement defines the temperature.
.temp *temp1 <temp2 ...>*
The temperature value is in degrees Celsius. If multiple temperatures are specified, HSIM will perform one simulation for each temperature with the output files (i.e. hsim.ic, hsim.fsdb, etc.) having a .t# suffix. The default temperature is 25 degrees Celsius.

.tran
The **.tran** statement defines transient analysis. Three versions of sweep syntax are supported.
1. .tran *steptime stoptime* <**uic**> <**sweep data**=*data_name*>
2. .tran *steptime stoptime* <**uic**> **sweep** *var pstart pstop incr*
3. .tran *steptime stoptime* <**uic**> **sweep** *var* **poi** np *p1 p2 ...*
Except for uic and sweep data, all optional parameters specified after stoptime are ignored. The **.tran** statement parameters are described in Table C-6.



Parameter	Description
steptime	A placeholder for compatibility with the SPICE syntax–this parameter does *not* effect HSIM simulation.
uic	Disables DC initialization and uses the IC conditions as DC solution. If a node does *not* have an I condition, its DC solution will be 0.
var	Parameter name or keyword temp (for temperature sweep)
pstart	Starting value.
pstop	Final value.
incr	Increment value.
np	Number of points.
p1, p2	Values for the parameter sweep.
poi	Indicates that type of sweep is a list of points.
sweep	Indicates sweep operation according to the parameter values specified in data_name. Refer to the example in .data on page 5-58.

TABLE C-6 .TRAN Parameters

AC Small-Signal Analysis

AC Small-Signal Analysis (AC) is a frequency domain analysis that calculates the small-signal response of a circuit to a combination of inputs. This is accomplished by creating a linear solution for the circuit at its DC operating point. AC analysis has the following features:

- Nonlinear devices are transformed to linear devices around their bias point value before running an AC analysis. Examples include:
  - Voltage-controlled sources
  - Current-controlled sources
- AC analysis only considers gain and phase responses of a circuit because it is a linear analysis
- AC analysis does *not* limit voltages or currents.

During AC analysis the simulator seeks a linear solution for the circuit in the frequency domain at the appropriate operating point. Hence, as a first step in conducting AC analysis, the operating point information is determined. Each voltage or current source can have any or all of the following components:

- AC component
- DC component
- Transient component

The following elements are supported in AC analysis:

- MOSFET
- BJT
- JFET
- Diode
- Resistor

• Capacitor
• Inductor
• Voltage and Current sources
- Voltage-Controlled Voltage Source (VCVS)
- Voltage-Controlled Current Source (VCCS)
- Current-Controlled Current Source (CCCS)
- Current-Controlled Voltage Source (CCVS)
• Mutual inductor

Invoking AC analysis After the operating point is determined, frequency domain analysis can begin. The
following statement invokes AC analysis and specifies the required frequency range.

.ac
.ac *sweep_type* nf *start stop*
AC analysis is conducted for the frequency range between start and stop.
sweep_type One of the following keywords:
dec - for decade increment
oct - for octave increment
lin - for linear increment
poi - for list of points
nf - number of frequencies for *lin* and *poi* types; or number of frequencies per decade for *dec* type, per octave for *oct* type.
start Starting frequency for *dec*, *oct*, *lin*
stop Final frequency for *dec*, *oct*, *lin*
Example:
.ac poi 4 1e8 5e8 8e8 1e9

NOTE: POI (Points of Interest)
AC analysis is conducted at four different frequencies:
• 100 MHz
• 500 MHz
• 800 MHz
• 1 GHz

AC frequency analysis can be performed when sweeping some external parameter(s) or value(s) of an independent source. An external sweep is specified by augmenting the **.ac** statement described above with the keyword sweep followed by one of the following specifications.
sweep *variable_name start stop incr*
sweep *variable_name sweep_type np start stop*
sweep *variable_name* poi *np p_1 p_2 ... p_n*
sweep data=*data1*
Here variable_name is one of the following:
• Independent voltage source name
• Independent current source name
• Parameter name
• Keyword: **temp** for temperature

DC Analysis

DC Analysis is used to determine the quiescent state or operating point information of the circuit. Different types of DC analysis are available in the HSIM simulator as described below:

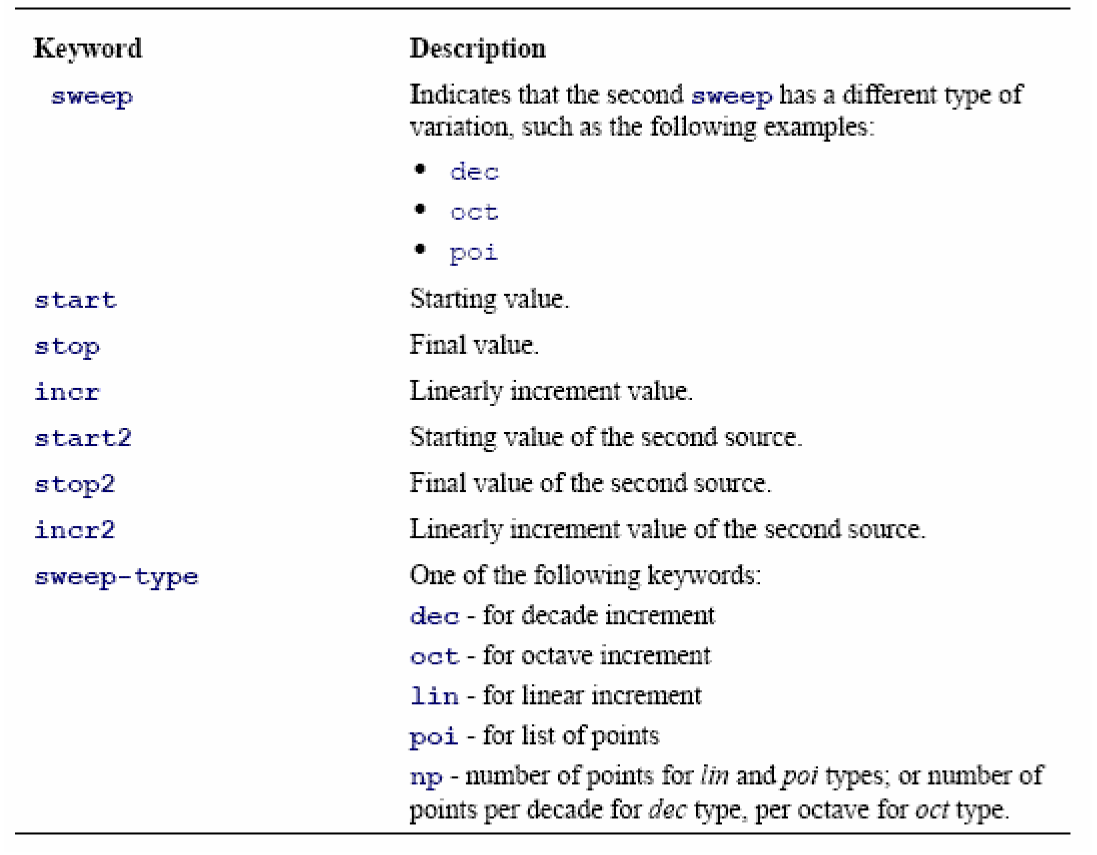
• Sweep a source value
• Sweep the values of two voltage or current sources
• Sweep a parameter value
• Sweep temperature value
• Any combination of the above Voltage or current sources must be specified as a netlist tem if they are included in the .dc statement.

Sweep One or Two Source Value(s)

Either of the following syntax statements can be used.

1. .dc *var start stop incr <var2 start2 stop2 incr2>*
2. .**dc** *var start stop incr* <**sweep** *var2* type np *start2 stop2*>

Sweeps the voltage or current source var. If the second source var2 is specified, then the first source is swept over its range for each value of the second source.



Keyword	Description
`sweep`	Indicates that the second `sweep` has a different type of variation, such as the following examples: • `dec` • `oct` • `poi`
`start`	Starting value.
`stop`	Final value.
`incr`	Linearly increment value.
`start2`	Starting value of the second source.
`stop2`	Final value of the second source.
`incr2`	Linearly increment value of the second source.
`sweep-type`	One of the following keywords: `dec` - for decade increment `oct` - for octave increment `lin` - for linear increment `poi` - for list of points `np` - number of points for *lin* and *poi* types; or number of points per decade for *dec* type, per octave for *oct* type.

TABLE C-7 Sweep One or Two Source Value Keyword Descriptions

Examples:

1. .dc i2 0 10m 0.5m

The current source i2 is swept from 0 A to 10 mA in increment of 0.5 mA.

2. .dc vds 0 2 0.1 vgs 0 2 1
Voltage source **vds** is swept from 0Vto 2V in increment of 0.1V at vds values of:
0.0V, 1.0V, 2.0V.
3. .dc v1 poi 4 0 0.3 0.5 1
The voltage source v1 is swept at values of: 0.0V, 0.3V, 0.5V, 1.0V.
4. .dc vds 1 3 0.2 **sweep** r1 dec 3 5k 500k
Voltage source vds is swept from 1V to 3V in increment of 0.2V with the resistor r1 value being swept from 5 Kohm to 500 Kohm by 3 values per decade (dec.).

Sweep Parameter Value

.dc *param_name start stop incr*
Sweeps *param_name*. **.dc** has the following parameters:
• *start* Starting value
• *stop* Final value
• *incr* Increment value
Example:
.dc param3 1 5 1
Parameter param3 is swept from 1 to 5 in increment of 1.
Sweep Simulation
Temperature
.dc temp *start stop incr*
Sweeps the simulation temperature. **.dc temp** has the following parameters:
• *start* Starting temperature
• *stop* Final temperature
• *incr* Increment value
Example:
.dc temp poi 4 0 25 50 75
HSIM is conducted at the following temperatures: 0 °C, 25 °C, 50 °C, 75 °C.

General HSIM Parameters

HSIMLIS
HSIM will concatenate all input files into a single file using the HSIMLIS command. When HSIMLIS=1, HSIM will generate hsim.lis or output_file.lis when invoking HSIM with -o output_file option.

HSIMWARNFILTER, HSIMMSGFILTER
HSIMWARNFILTER is used to filter out unwanted *Warnings* and HSIMMSGFILTER filters out Messages. By setting HSIMWARNFILTER or HSIMMSGFILTER to a given string, any *Warnings* or Messages that contain the given string will *not* be displayed. The syntax is as follows:
.param HSIMWARNFILTER=”removing unused subckt”
or
.param HSIMMSGFILTER=”removing unused subckt”

HSIMWARNSTOP

HSIMWARNSTOP is used to terminate simulation upon reaching a user-specified number of *Warnings*. When HSIMWARNSTOP=1, the maximum number of *Warnings* can be set by setting:
.option warnlimit=#

HSIMFLAT

In simulating sub-circuit cells, the efficiency gained by saving identical sub-circuit computations is significant enough to compensate for the overhead. If each sub-circuit cell has different behavior during the simulation, then the flat simulation might have an efficiency edge over hierarchical simulation. Hierachical simulation still has an advantage in saving the storage space. To allow a trade-off choice in memory usage and simulation speed, the control parameter HSIMFLAT enables selecting either a flat or a hierarchical simulation.
When HSIMFLAT=1, HSIM does *not* flatten the entire circuit. Instead, it only selectively flattens those sub-circuits that have no sufficient sub-circuit instances, or the sub-circuit that has been identified to have different behavior from other sub-circuit instances.
HSIMPREFLAT If HSIMPREFLAT=1, the circuit netlist is flattened before partitioning. This is equivalent to read in a flat netlist. HSIMPREFLAT is valuable in some cases of post-layout simulation where the parasitic Resistors and Capacitors (RC)s are defined within the sub-circuits.

HSIMDCINIT

DC initialization is activated after the netlist processing and hierarchical database building phase is completed and before the beginning of a transient simulation. However, under certain special situation, there is no need to perform DC initialization to achieve the desired results. The control parameter HSIMDCINIT is used to activate the DC initialization.

HSIMDCSTEP

HSIMDCSTEP determines the smallest time step used during DC initialization. Units are in picosecond and default value is 1000.

HSIMMULTIDC

HSIMMULTIDC invokes multi-rate algorithm which speed up DC convergence when set to 1. The default value is 1.

HSIMENHANCEDC

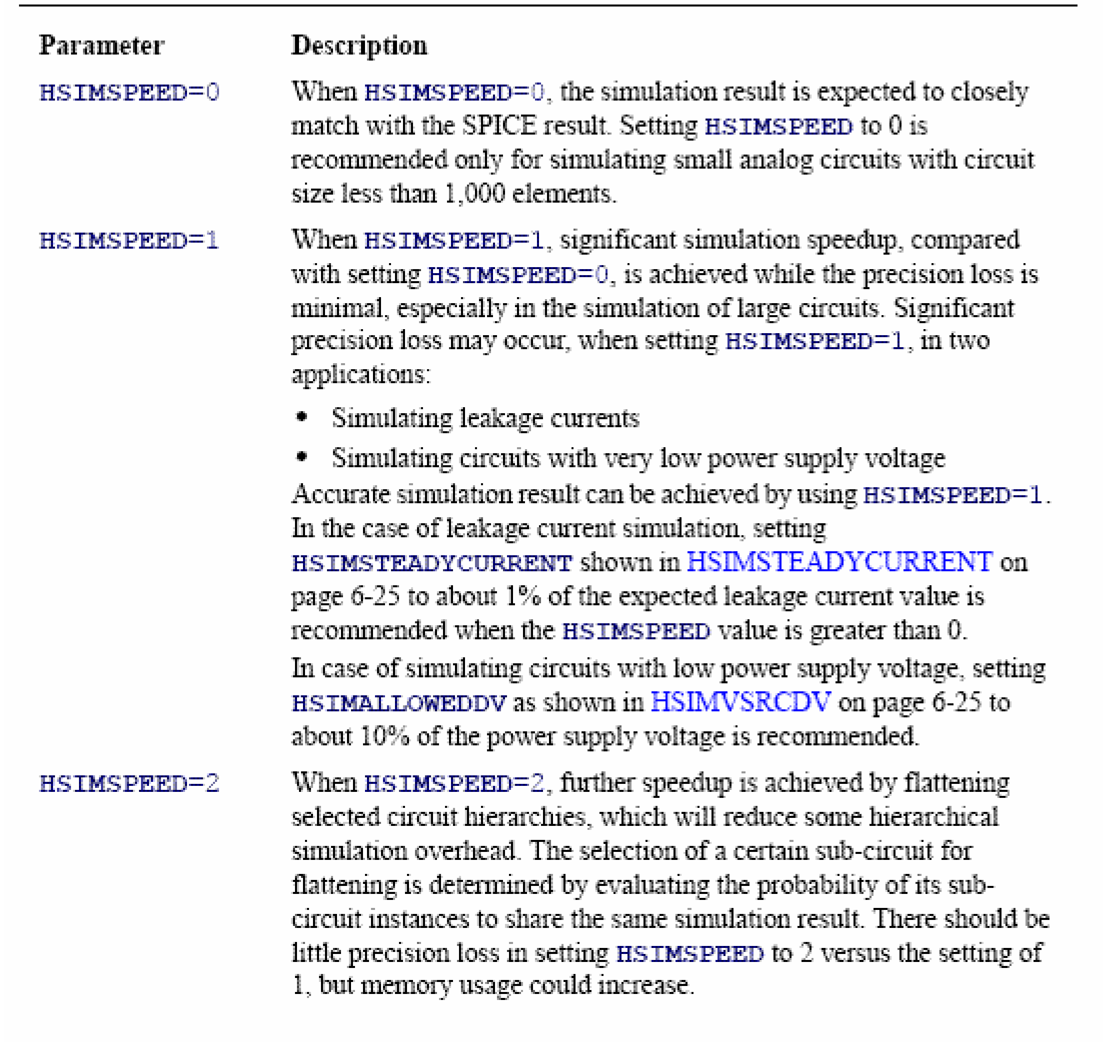
HSIMENHANCEDC invokes more conservative DC initialization algorithm and settings and increase the DC iteration limit to 1000 when set to 1. When set to 2, HSIM utilizes the voltage-dependent MOSFET capacitance model in dc initialization.
HSIMENHANCEDC=0|1|2

HSIMKEEPNODESET

HSIM automatically identifies cross coupled nodes in circuits and applies a logic 1 nodeset value to one of the nodes. This avoids a meta stable condition after DC initialization. Since the nodeset value is kept only at the first iteration, it can be overridden. When HSIMKEEPNODESET is set to 1, all nodeset values will be kept during the first 1/10 of the total iterations.

HSIMSPEED

To facilitate the selection on speed and precision performance, the control parameter HSIMSPEED automatically sets these parameter values. The settings of those parameters are defined in Table C-8, HSIMSPEED Parameters. The value of HSIMSPEED can be any integer from 0 to 8. Higher HSIMSPEED speed values cause faster simulation speed at reduced simulation precision.



Parameter	Description
`HSIMSPEED=0`	When `HSIMSPEED=0`, the simulation result is expected to closely match with the SPICE result. Setting `HSIMSPEED` to 0 is recommended only for simulating small analog circuits with circuit size less than 1,000 elements.
`HSIMSPEED=1`	When `HSIMSPEED=1`, significant simulation speedup, compared with setting `HSIMSPEED=0`, is achieved while the precision loss is minimal, especially in the simulation of large circuits. Significant precision loss may occur, when setting `HSIMSPEED=1`, in two applications: • Simulating leakage currents • Simulating circuits with very low power supply voltage Accurate simulation result can be achieved by using `HSIMSPEED=1`. In the case of leakage current simulation, setting `HSIMSTEADYCURRENT` shown in HSIMSTEADYCURRENT on page 6-25 to about 1% of the expected leakage current value is recommended when the `HSIMSPEED` value is greater than 0. In case of simulating circuits with low power supply voltage, setting `HSIMALLOWEDDV` as shown in HSIMVSRCDV on page 6-25 to about 10% of the power supply voltage is recommended.
`HSIMSPEED=2`	When `HSIMSPEED=2`, further speedup is achieved by flattening selected circuit hierarchies, which will reduce some hierarchical simulation overhead. The selection of a certain sub-circuit for flattening is determined by evaluating the probability of its sub-circuit instances to share the same simulation result. There should be little precision loss in setting `HSIMSPEED` to 2 versus the setting of 1, but memory usage could increase.

HSIMSPEED can be selectively set in local sub-circuits such that:
• Lower HSIMSPEED values are set for either of the following:
- Analog sub-circuits
- Timing critical sub-circuits
• Higher HSIMSPEED values can be set for digital sub-circuits.

Parameter	Description
HSIMSPEED=3	When HSIMSPEED=3 is the default HSIMSPEED value, additional speedup is provided through the multi-rate time stepsize selection. When HSIMSPEED value is set to less than 3, the time stepsize selection is uniform such that a common stepsize is selected for every sub-circuit. When HSIMSPEED value is greater or equal to 3, different time stepsizes are selected in different sub-circuits, such that smaller time step is selected for sub-circuits with faster transient activities while larger time step is selected for sub-circuits with slow transient activities. This speedup in multi-rate step selection is achieved by reducing the total number of time steps. The precision loss in setting HSIMSPEED=3, compared with lower values of HSIMSPEED is significant only in very few applications such as simulating high-sensitivity analog circuits. However, it is generally *not* necessary to lower the HSIMSPEED value: the control parameter HSIMANALOG is recommended which achieves analog simulation speed without sacrificing precision. Refer to High-Sensitivity Analog Circuit Simulation: HSIMANALOG on page 6-36
HSIMSPEED=4	When using HSIMSPEED=4, an increased speedup compared to HSIMSPEED=3 is achieved by relaxing the precision resolution. This selection is recommended for simulating digital circuits, memory circuits, or analog circuits with lower sensitivity. In such applications, the precision difference between HSIMSPEED values of 3 and 4 is minimal.
HSIMSPEED=5	HSIMSPEED=5 is recommended for functional simulation, including memory and mixed-signal circuits. In this case, simulation speedup is achieved by sacrificing timing delay precision. In general, 5-10% of timing delay error is expected when using this high-speed simulation mode.
HSIMSPEED=6	HSIMSPEED=6 is recommended for functional simulation of digital circuits or low sensitivity analog circuits.
HSIMSPEED=7	HSIMSPEED=7 is recommended for functional simulation of digital circuits or low sensitivity analog circuits. HSIMSPEED=7 is optimized for circuits such as ROM and CAM.
HSIMSPEED=8	HSIMSPEED=8 is recommended for functional simulation optimized for Flash memory and mixed-signal circuits. Two submodes, HSIMSPEED=8.4 and HSIMSPEED=8.8 take advantage of circuit latency in large circuit blocks to speed up simulation. These submodes are especially useful in circuits that contain voltage regulators or charge pumps.

TABLE 6-13. HSIMSPEED Parameters

HSIMITERMODE

HSIMITERMODE is the HSIM iteration control parameter and can be set integer from 0 to 2 as described as follows and in Table 6-16, Iteration Control Parameter.

HSIMITERMODE=0|1|2

SYNTAX DEFINITIONS

• HSIMITERMODE=0 Default.

• HSIMITERMODE=1 Uses Nassda's proprietary iteration scheme which balances accuracy and performance.

• HSIMITERMODE=2 A spice-like Newton Raphson iteration is used.

HSIMANALOG

To achieve optimal HSIM performance in analog and mixed-signal circuit simulation, HSIMANALOG controls the complexity of analog simulation algorithm. The higher the value specified by HSIMANALOG is, the more precise and time-consuming the analog simulation algorithm will be.

HSIMANALOG=-1 In simulating digital circuits where no analog circuit behavior is expected, set HSIMANALOG=-1. This assumes no sensitive coupling exists between neighboring subcircuits.

HSIMANALOG=0, HSIMANALOG=1

HSIMANALOG=0 or HSIMANALOG=1 settings are used to simulate full-custom circuits or memory circuits. Compared with HSIMANALOG=-1, these commands provide for increased complexity from more precise simulation in local feedback coupling between neighboring sub-circuits.

HSIMANALOG=2 HSIMANALOG=2 further increases the complexity of the analog simulation algorithm by extending the feedback couplings to a wider scope, such as including voltage control oscillators (VCO)s found in most PLL circuits. HSIMANALOG=2 is recommended for use with analog circuits containing highly sensitive topology such as the following:

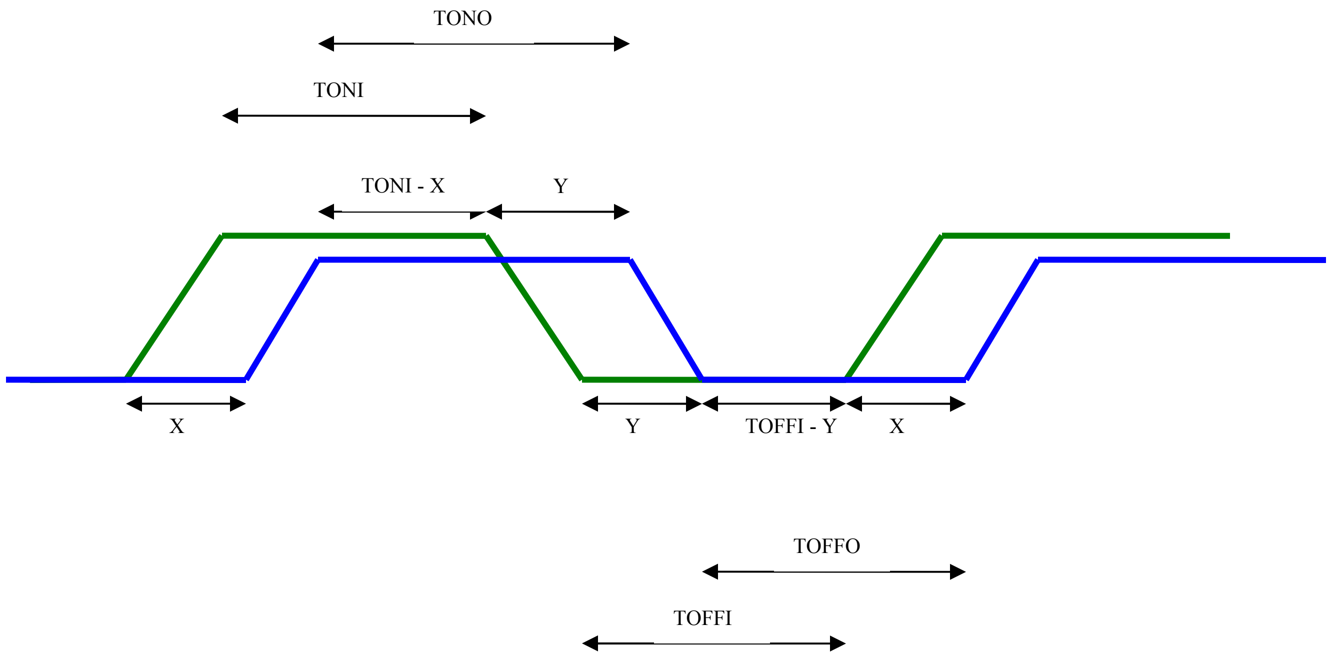
HSIMANALOG=3

HSIMANALOG=3 can further improve analog simulation precision. HSIMANALOG=3 automatically applies HSIMSPICE=3 to those MOSFETs involved in feedback coupling. This setting is recommended for simulating A/D and D/A converters.

D. EXPRESSION FOR DETERMINING MAXIMUM FREQUENCY

Convention Used:
X= Rise delay.
Y= Fall delay.
TONI= ON time of input wave.
TONO=ON time of the output wave.
TOFFI= OFF time of the input wave.
TOFFO= OFF time of the output wave.
DUTY O/P = Output duty cycle.
DUTY I/P = Input duty cycle.
TIN = Time period of input wave.
GREEN= Input wave.
BLUE= Output wave.

CASE 1: When TONI > X
Following waveform depicts the case.



From the figure

$$TONO = (TONI - X) + Y$$
$$TOFFO = (TOFFI - Y) + X$$

$$\text{DUTY O/P} = \frac{\text{TONO * 100….}}{\text{(TONO + TOFFO)}}$$

$$\text{DUTY O/P} = \frac{\{(\text{TONI} - X) + Y\} * 100\ldots}{\{(\text{TONI} - X) + Y\} + \{(\text{TOFFI} - Y) + X\}}$$

$$\text{DUTY O/P} = \frac{\text{TONI} + (Y - X\} \ * \ 100\ldots.}{\{\text{TONI} - X + Y + \text{TOFFI} - Y + X\}}$$

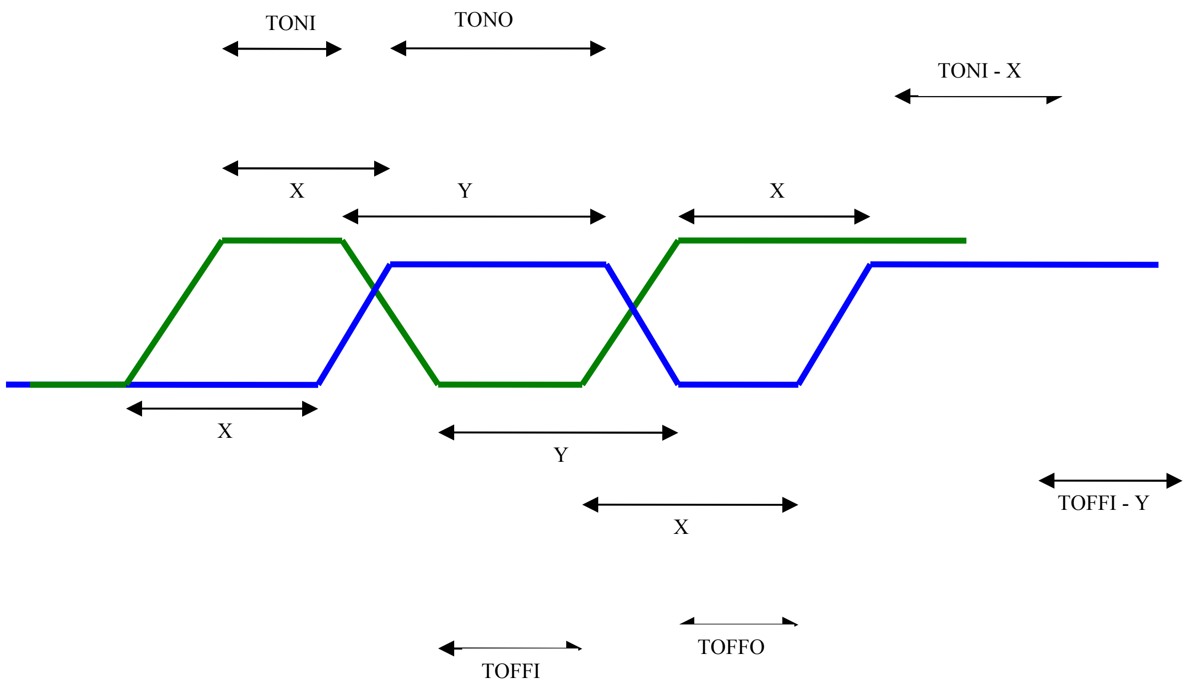
$$\text{DUTY O/P} = \frac{\text{TONI} + (Y - X\} \ * 100}{(\text{TONI} + \text{TOFFI})}$$

$$\text{DUTY O/P} = \frac{\text{TONI} + (Y - X\} * 100}{\text{TIN}}$$

$$\text{DUTY O/P} = \frac{\text{TONI} * 100}{\text{TIN}} + \frac{(Y - X)*100)}{\text{TIN}}$$

$$\textbf{DUTY O/P = DUTY I/P +} \frac{\mathbf{(Y - X)* 100}}{\textbf{TIN}}$$

CASE 2: When TONI < X



From the figure

$$TONO = TONI + Y - X$$
$$TOFFO = TOFFI + X - Y$$

Above two equations are the equations that we get in case 1.
So, using case1 result,

$$\textbf{DUTY O/P} = \textbf{DUTY I/P} + \frac{(Y - X)*100\ldots}{TIN}$$

Above relation can be modified to get:

$$\Delta\ \textbf{DUTY} = \frac{(Y - X)*100}{TIN}$$

Where ΔDUTY = DUTY O/P – DUTY I/P.

Using the result, maximum frequency may be found out
Example 1
For LVCMOS12 standard for input path,
DIN = 50%
DOUT = 60%
Y (TPHL) = 1.200e-09
X (TPLH) = 2.546e-09.

Using the result,

$$\Delta \text{ DUTY} = \frac{(Y - X)*100}{\text{TIN}}$$

$$10 = \frac{(2.546e\text{-}09\text{- }1.2000e\text{-}09)*\ 100}{\text{TIN}}$$

$$\text{TIN} = \frac{1.346e\text{-}09*100}{10}$$

TIN = 13.46e-09

FIN = 74.29 MHz

<u>Simulation result</u>
Run at Fin = 74.29MHz
DOUT = 59.970%
Thus the relation has been verified from simulation.

<u>Limitation of formula</u>

1. If difference between rise delay and fall delay is very small then if we find frequency for a particular output duty cycle, the frequency calculated will be high. It might happen, at that frequency circuit does not work.
2. The formula shows relation between delays, input frequency input and output duty cycle. It only shows how output duty cycle varies if there is mismatch in the rise and fall delay.