Overview

Design Automation is essential in dealing with the complexity of IC design and verification.

There are a wide range of tools available:

- *Analysis* and *verification* tools to determine if the electrical behavior of the design is within specification bounds.
  SPICE, IRSIM, Cadence Verilog simulation, Spectra
- *Implementation* and *synthesis* tools allow the designer to generate and optimize circuit schematics and layout.
  MAGIC, LAGER, Cadence Leapfrog, Composer, Virtuoso
- *Testability techniques*, *test generation* and *fault simulation* tools allow the designer to verify the functionality of the design in hardware.
  PODEM, FAN, LTEST and miscellaneous commercial tools.

We will spend a little time on analysis and verification tools but will focus primarily on the last set of device verification tools and methods.

Simulation and Verification

Let's first distinguish between these.

- Simulation

  The value of a design parameter such as noise margin, propagation delay or energy consumption is determined by applying a set of excitation vectors to a circuit model.

  The parameters are then extracted from the signal waveforms.

  The primary disadvantage is that the results depend strongly on the choice of excitations.

- Verification

  The parameters are extracted directly from the *circuit description*.

  For example, the critical path of a ripple carry adder can be determined directly from the schematic.

  The advantage is that this approach is **independent** of the excitation vectors.

  The disadvantage is that it requires an understanding of the circuit design style, e.g. dynamic or static logic.

Levels of Abstraction

The *complexity* of a circuit is related to the **level of abstraction** required to describe its operation in a meaningful way.

The **level of abstraction** can be characterized by the types of information processed by the circuit.

Control	Data	Level of Abstraction
Messages		System Level
Programs	Data Structures	Processor Level
Instructions	Words	Instruction Set Level
Logic values	Words	Register Level
Binary-valued Logic values		Logic Level
Multi-valued Logic Values		Switch Level
Analog Voltage Values		Circuit Level

One view of a system distinguishes a **data part** interacting with a **control part**.

Levels of Abstraction

- Circuit Level:

  Circuit is modeled as a complex network of transistors, resistors, capacitors and inductors.

  Control and the type of information processed by the data part of the design is represented as *continuous voltage values*.

- Switch Level:

  Transistors are modeled as switches, wires are modeled as resistors and capacitors.

  Control and data are represented in multi-valued logic (discrete signals with strengths associated with them).

- Logic Level:

  Primitive circuit elements are logic gates.

  Control and data are represented as *logic values* (combinational) or *sequences* of logic values (sequential).

Levels of Abstraction

- Register Level:

  Primitive circuit elements are gates, FSMs, shifters, boolean function units, adders, muliplexors, decoders, registers and data paths.

  Control is still represented as logic values.

  Information processed by the data part consists of **words** (groups or vectors of logic values).

  These data words are usually stored in registers.

- Instruction Level:

  Primitive circuit elements are PCs, IRs, ALUs, floating point units, register files, caches, etc. and data paths.

  Both control and data are organized as words.

  In this case, control information is referred to as *instructions* and the system is called an *instruction set processor*.

Levels of Design Abstraction

- Processor Level:

  Primitive circuit elements are devices and buses, e.g., processors, memory controllers, memory banks, caches, etc. and buses.

  At this level, the digital system can be regarded as processing a *sequence of instructions*, or programs, that operate on blocks of data called *data structures*.

- System Level:

  Primitive circuit elements are devices and buses or computer systems and networks.

  A different view of the system (not necessarily at a higher level of abstraction) is to consider it composed of independent subsystems which communicate via blocks of words called *messages*.

Circuit Level Simulation

SPICE is the most utilized computer simulation tool used in digital circuits.

However, it is basically an engine that solves a set of **non-linear differential equations** using iteration to converge on the solution.

It is accurate, given tight error bounds, but very time consuming.

It is nearly impossible to use for large commercial designs.

The resulting current and voltage waveforms are continuous waveforms.

Time, although it appears continuous, is actually a sequence of non-uniform time points at which the matrix representing the differential equations was solved.

Interpolation is performed over the missing time intervals.

Basis For Other Simulation Approaches

The non-linearity of semiconductor devices is one of the major impediments to higher simulation speeds.

If the designer is willing to give up some modeling accuracy, other tools are available that represent the circuit at *higher levels* of abstraction.

Abstraction is the primary means of reducing complexity.

This stands true for implementation and synthesis tools as well as simulation and verification tools.

For simulation, the most straightforward abstraction is to **discretize the data**.

For example:

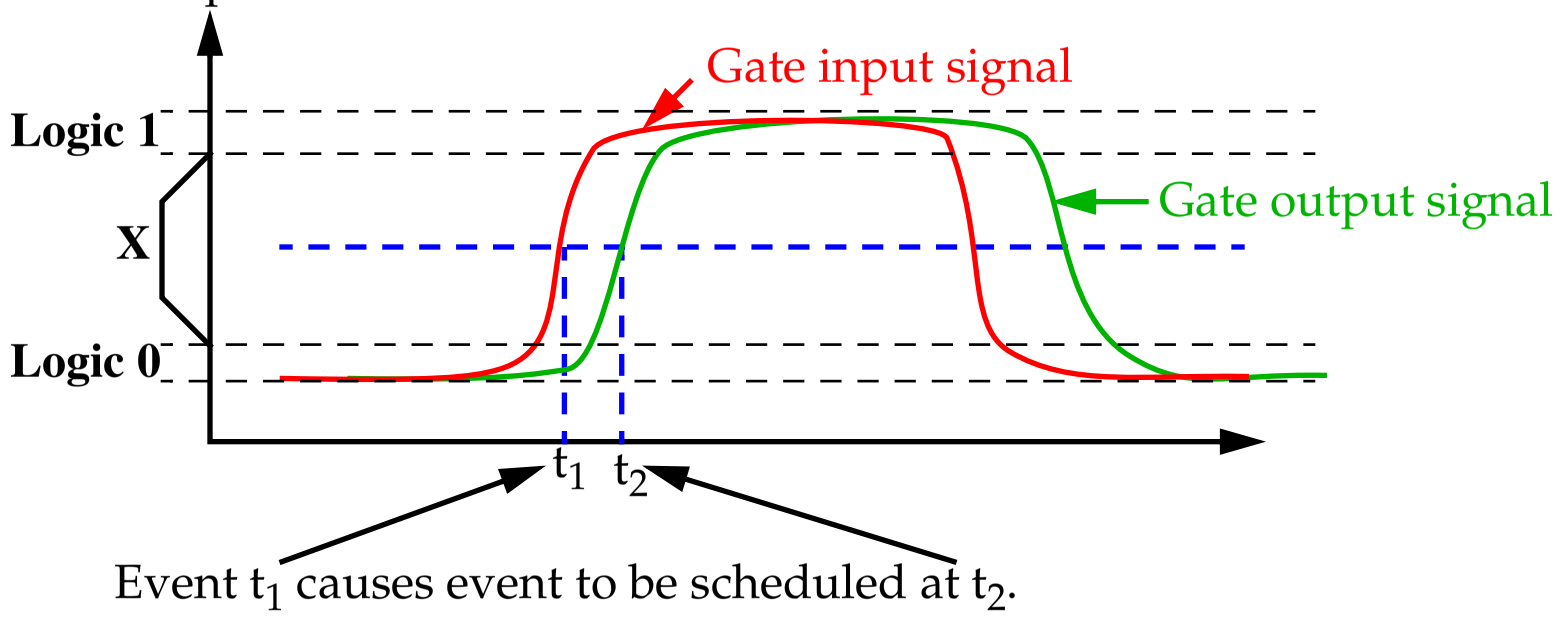
Analog voltage values are treated as digital data composed of 1s and 0s.

X is used for signals that do not fall within the threshold of a digital 0 or 1.

Commercial simulators (Cadence) offer a much wider range of digital values, such as Z (high impedance), and strength attributes for 1 and 0.

Basis For Other Simulation Approaches

It is also possible to **discretize time**.



Event-driven simulators only evaluate a gate at the time an event happens at one of its inputs.

The evaluation order is determined by putting projected events (t_2) on a queue and processes them in a time-ordered fashion.

Fan-out is accounted for by expressing delay through a gate as the sum of an *intrinsic delay* (t_{intr}) plus a *load dependent factor* (t_{load}).

Basis For Other Simulation Approaches

Further simplifications are possible.

- **Variable delay model**:

  The above model allows events to occur at any arbitrary point in time.

- **Unit delay model**:

  It is possible to discrete time even further by allowing events to occur only at integer multiples of a unit time variable.

- **Zero delay model**:

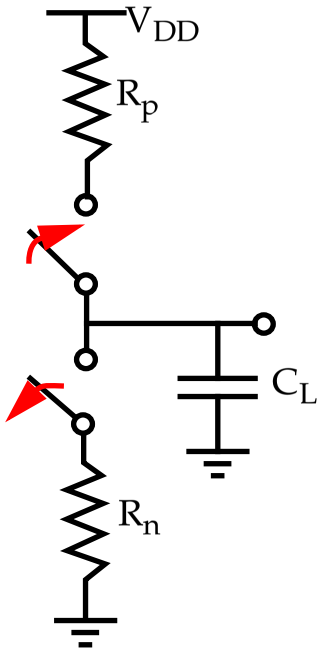
  The simplest model in which all events occur simultaneously with the arrival of a clocking event.

Switch Level Simulation:

In switch-level models, the transistor behavior is approximated using a *linear* resistance and the *load* is represented by C_L.

Therefore, the simulation amounts to analyzing an RC network.

Circuit is represented as a time-variant, linear network of resistors and capacitors.



Off-mode, the resistance is set to infinity.

On-mode, the resistance is set to the average on-resistance of the device.

Node values are determined by evaluating the steady-state values of the resistor network (0, 1 and X) and delay is given by R times C.

Logic (Gate) Level Simulation

Simulation at this level can use the same signal values as switch-level tools, but the simulation primitives are gates rather than transistors.

This makes it difficult to treat VLSI structures such as *tri-state* buffers and *pass* gates.

However, gate-level was the preferred design-entry level by designers for a long time.

Up until the introduction of logic synthesis tools.

Logic synthesis moved the focus to the **functional** and **behavioral** abstraction layer.

Functional Simulation

Functional simulation can be considered a simple extension to logic simulation.

However, the primitives can be of arbitrary complexity:

- A NAND gate
- A multiplier
- An SRAM memory

Register Level descriptions are simulated at this level.

The functionality of these units can be described by a:

- Modern programming language:
  C or C++
- Hardware description language:
  VHDL
  Verilog

Functional Simulation

VHDL and Verilog support both a *structural* model and a *behavioral* model.

- **Structural model:**

  Describes the connection of the functional units in a netlist format. It mirrors the intended hardware structure.

- **Behavioral model:**

  Describes the module as a set of input-output relations, and is independent of the implementation.

The signal levels of the functional simulator are similar to *switch* and *logic* levels.

The timing model can vary:

- Delays between input and output are specified as part of the behavioral description of the module.
- Zero delay model can be used.

Instruction, Processor and System Level Simulation

System level entities such as an embedded microcontroller are rarely modeled at the bit level.

Data moves over **buses** and instructions are bit patterns from an **enumerated set**, such as RD and WR.

Modeling at this level of abstraction allows the system to be understood easily and it is also very fast to simulate.

An action on a 64-bit bus, for example, only requires a single action.

However, *timing accuracy* is sacrificed since it is no longer possible to model individual wires.

Processor and System Level Simulation

Functional (structural) and behavioral descriptions exist at this level as well.

However, hardware delay loses its meaning and simulations are performed on a *per clock-cycle* or higher base.

For example, verification of a microprocessor may be performed on a per instruction basis.

User-defined data types such as 16-bit two's-complement words or enumerated instruction sets are used at this level.

VHDL and VERILOG are commonly used at this level of abstraction.