Chapter 2

Aspects of Hardware Description

2.1 Introduction

There are three fundamental aspects of any piece of hardware :

- Behavioural.
- Structural.
- Physical.

The behavioural aspect (functionality and timing) deals with the behaviour of hardware : *what* is its functionality and speed (without worrying about the constructional details).

The structural aspect (netlist) deals with the scheme of hardware construction : *which* parts have been selected for construction and *how* they have been interconnected.

The physical aspect deals with the physical organization of the hardware layout, placement and the routing of its components.

Of course, a complete information on the hardware requires a combination of both the behavioural and structural aspects. In some situations, we would additionally need information on the physical aspects (*e.g.* the layout of the IC or the PCB).

However, in many practical situations, we may need to focus only on a single aspect.

In fact the information on each of these aspects is itself represented at different levels of detail — using different abstraction levels. Each abstraction level is used to represent information on an aspect of the design at a certain convenient level of detail.

For example, the structural aspect (the net list) of a VLSI design can be described at several different abstraction levels *e.g.* the system structure level, the processor-buses level, the ALU-register level, the gate-flip-flop level, and the transistor level. Simultaneously, the behavioural and physical aspects of a VLSI chip/system can also be described at different levels of abstraction.

Very often the three aspects of a design are called the three design domains. And for each domain (aspect) there are several different levels of abstraction at which the design information in that domain is represented.

Therefore, a complete system of maintaining, providing, and exchanging information on a VLSI design must address all the three domains (aspects) individually or in combination.

However, a single design description language that permits design description in all the three design domains at each of the abstraction levels does not exist.

The Gajski's Y-Chart graphically represents the three design domains and the popular abstraction levels used in each of the domains and is shown in Figure 2.1.

Hardware description languages (HDLs) have been developed to provide a means of describing, validating, maintaining and exchanging design information on complex digital VLSI chips across multiple levels of design abstractions used during the design process in behavioural and structural domains.

Note that HDLs do not cover the physical domain. For design description in the physical domain, a domain-specific language such as `CIF` (Caltech Intermediate Format) or the `GDSII` format is used.

Other domain-specific languages / design description formats — both proprietary as well as in public-domain — have existed for a long time. Some of these have become either formal or defacto industry standards. `EDIF` standard for net lists is one such example of an industry standard in the structural domain.

2.2 Hardware Design Methodology

Systematic design methods (called *design methodologies*) are necessary for successfully designing complex digital hardware. Such design methods can be classified into following three broad categories :

- Top-down Design.
- Bottom-up Design.
- Mixed Top-down and Bottom-up Design.

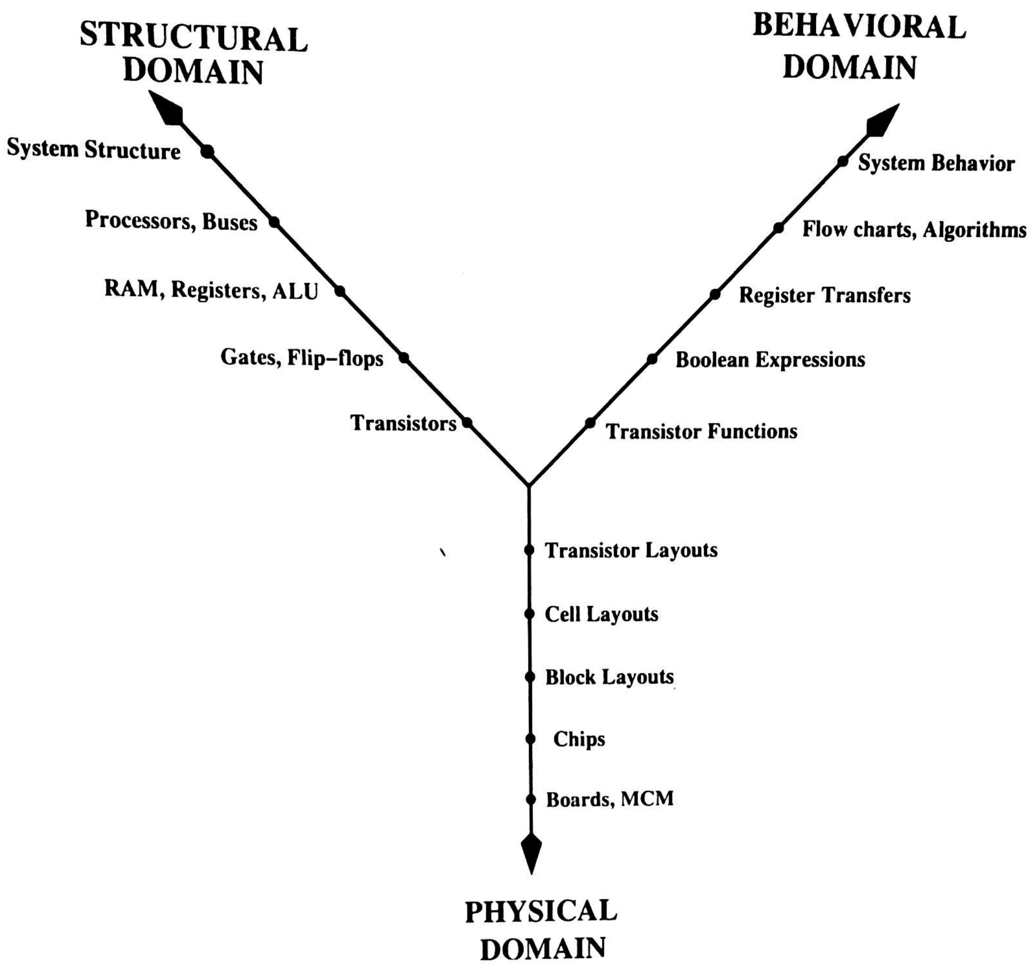

Figure 2.1: **Gajski's Y-Chart**

Top-down Design :

This method relies on realizing the desired complex behaviour of a hardware by *partitioning* it into interacting simpler sub-behaviours.

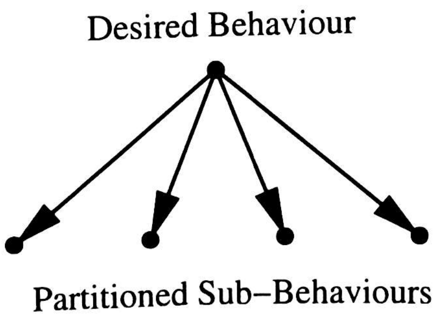

Figure 2.2: **Top-down Approach**

The result of behavioural partitioning can also be translated into an equivalent structural representation — as an interconnection of parts whose behaviours are the partitioned sub-behaviours. Since in the behavioural partitioning process the designer enjoys full freedom, the resulting sub-behaviours may or may not have pre-existing physical parts corresponding to them. Nevertheless, the behavioural partitioning and its corresponding structural representation do represent the first useful steps in the direction of realization of complete physical design — which is the final goal of the design process.

This methodology can be applied to the partitioned sub-behaviours themselves (recursively) to break down the original complex behaviour into an interacting set of simple-enough behaviours whose implementations (physical designs) are either known or can be constructed using known methods.

Bottom-up Design :

This method aims at realizing the desired complex behaviour by suitably selecting behaviours from a *given* set of behaviours and setting up interactions among them.

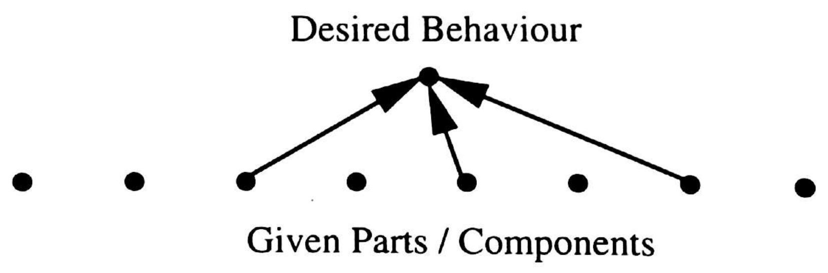

Figure 2.3: **Bottom-up Approach**

The corresponding structural description would be a netlist in which all the parts are selected (based on suitability of their behaviours) from an *available* library of parts and interconnected to realize the desired complex behaviour (building-block approach). Such an available set of parts/components is often a standard cell library or a library of TTL gates.

This method is adequate whenever the complexity of desired behaviour is not too high as compared to the complexity of behaviours of the available parts *i.e.* when the "semantic gap" is small. An example is the design of the logic function of a 4-bit ALU or a counter using the library of TTL gates.

However, this method does not work when the complexity of desired behaviour is much too high as compared to the complexities of behaviours of available parts *i.e.* when the "semantic gap" is large. An example is the design of the logic function of a Microprocessor using the library of TTL gates.

Mixed Top-down and Bottom-up Design :

This is the most commonly used and practical method of designing complex VLSI circuits.

It aims at realizing the desired complex behaviour by first following the top-down design method (recursively a few times, if necessary) and partitioning the desired complex behaviour into a set of intercommunicating simpler sub-behaviours, and then realizing the resulting simpler sub-behaviours by following the bottom-up design method *e.g.* by selection and interconnection of parts from an available library/set of parts with known behaviours.

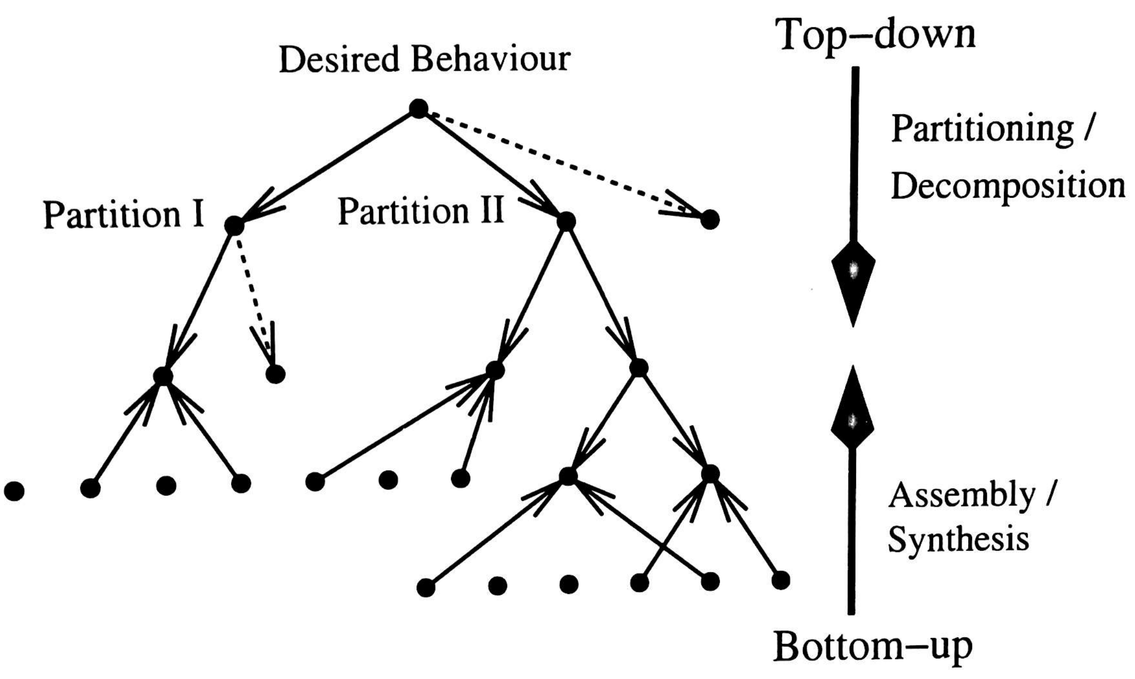

Figure 2.4: **Mixed Top-down and Bottom-up Approach**

Chapter 3

VLSI Design Overview

3.1 Introduction

VLSI design involves translating the given specifications into geometrical patterns that are used for fabricating chips.

This translation task is very complex and cannot be accomplished in one step. It is accomplished through a succession of translation steps of manageable complexity. Each translation step translates a more abstract (less detailed) design representation into less abstract (more detailed) design representation. Abstractions are the means of representing various views of the design with varying amounts of details.

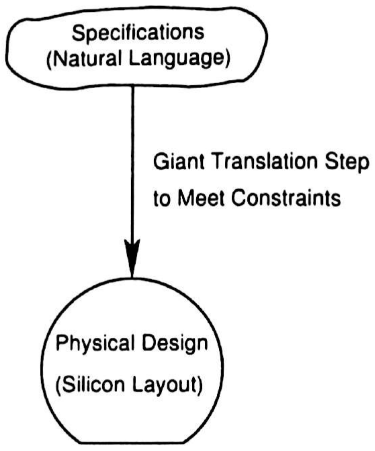

Figure 3.1: **Giant Design Translation Step**

Different design representation levels are called different abstraction levels. The different abstraction levels used for representing the behavioural, structural and physical aspects of a design are shown along the corresponding three axes in the Gajski's Y-Chart (see Figure 2.1).

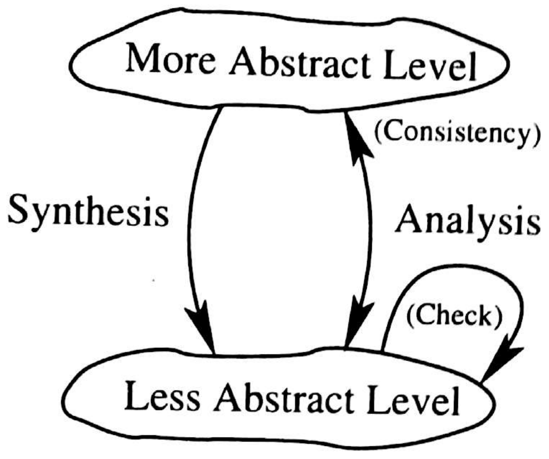

Figure 3.2: **Design Translation Definitions**

Note that each translation step is a one-to-many mapping. For a given design description at a certain level of abstraction there are many valid design translations at the next lower level of abstraction — each with a different cost-performance implication thus offering trade-offs to the designer.

There is also a combinatorial explosion of choices in the design space as one moves from higher to lower levels of abstraction.

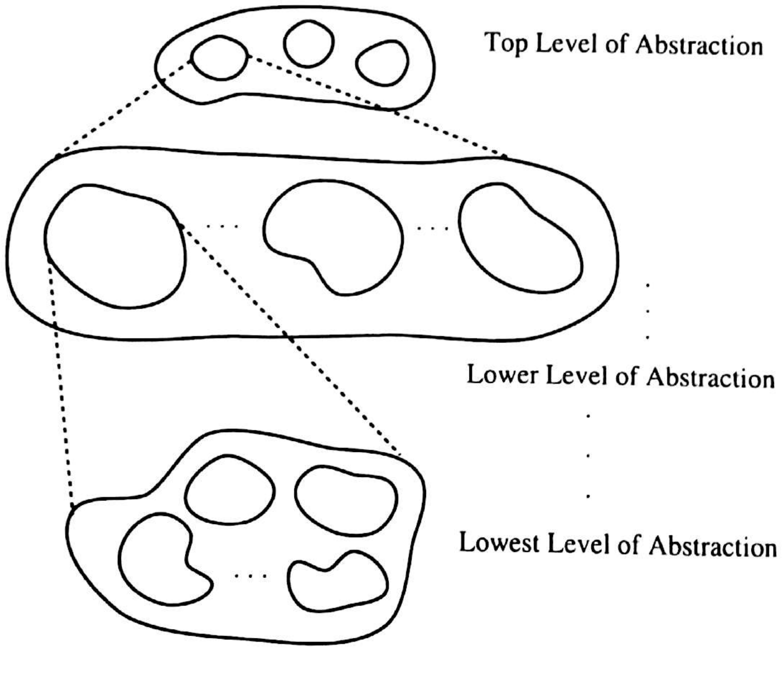

Figure 3.3: **Design Translation Mapping**

Therefore, the central issue in VLSI design is one of managing increasing complexity and making optimal choices as one goes through successive translation steps generating design descriptions / representations at successive lower levels of abstraction.

3.2 VLSI Design Flow

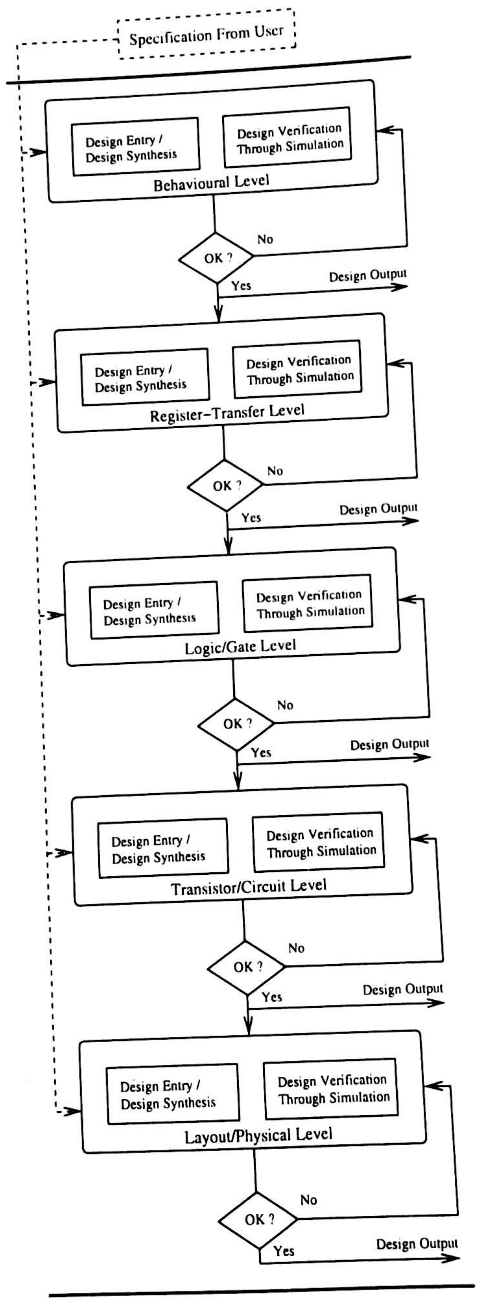

Figure 3.4: **VLSI Design Flow and Common Abstraction Levels**

3.3 Top-down Approach

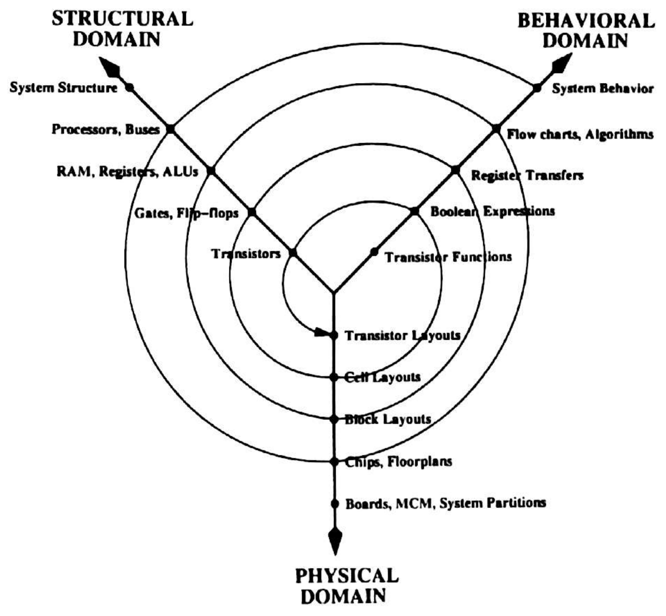

Figure 3.5: **Top-down Design Approach on Gajski's Y-Chart**

3.4 Mixed Approach

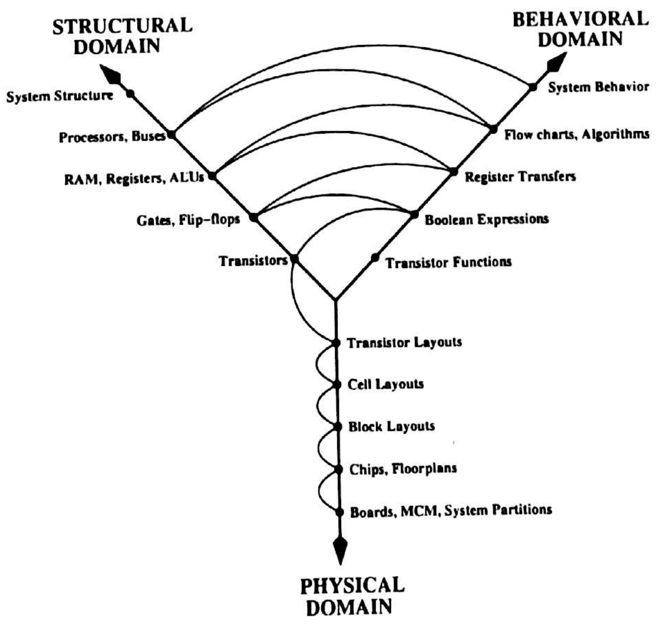

Figure 3.6: **Mixed Design Approach on Gajski's Y-Chart**

3.7 Steps in VLSI Design

3.7.1 Specification Capture

The Specification needs to be comprehended and captured (represented) through the use of a suitable level of abstraction. *e.g.* behavioural.

3.7.2 Architecture Design

Decomposition of the overall specifications (functional + timing) into the specifications (functional + timing) of the constituent blocks and their intercommunications.

e.g. How many blocks of *what* functionality and speed communicating in *what* manner ?

Exploration of the architectural alternatives (architectural space) including the possibilities of parallelism and pipelining.

Rough estimation of speed/throughput, area and power for each architectural alternative to decide on an architectural course to be taken *i.e.* selection of an optimal architecture best-suited for the application.

3.7.3 Register-Transfer Level Design

Further refinement of the design to arrive at a Register-Transfer Level (RTL) / dataflow level description.

For a complex processing specification, rarely (if ever) can one put enough circuitry on a chip that would realize the full processing in one step. Practically, one always faces limitations on the amount of processing circuitry that can be put on a chip.

Therefore, complex processing is invariably realized via a sequence of simpler processing steps repeatedly using this limited processing hardware and storing the intermediate results that are then used in subsequent processing steps *i.e.* all complex processing is realized through the use of sequential hardware.

The design issue, thus, boils down to the optimal choice of processing circuitry (one or more processing elements) to be incorporated on the chip, the storage elements to be incorporated on the chip, the intercommunication paths between the storage elements (both on-chip and off-chip) and the processing elements, and the associated sequencing circuitry to control the sequencing of processing steps and the actual processing done at each step.

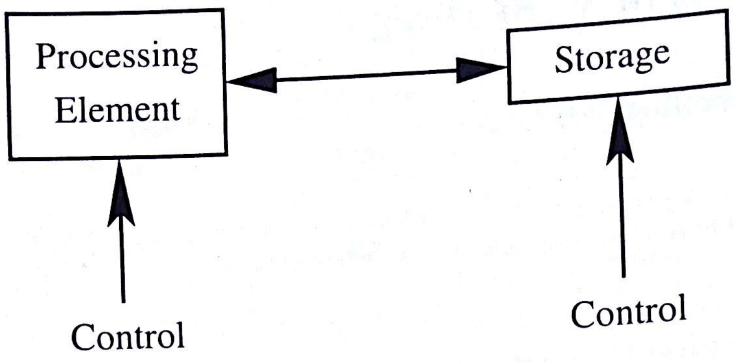

Figure 3.9: **Sequential Computation**

In general, the more complex the processing circuitry on the chip, the fewer the processing steps and storage required to achieve full processing. The vice-versa is also true in general.

Thus, the designer faces trade-offs here. By making optimal choices for his specific situation, the designer arrives at a specific architecture with associated elementary processing abilities through whose sequencing via a control part he can achieve the full processing desired.

A Finite State Machine (FSM) for control plus a Datapath for computation (FSM+D) is a popular abstraction model.

3.7.4 Logic Level Design

Each of the processing elements, the storage elements and the control part is translated into detailed logic gate level description or blocks of logic functions.

3.7.5 Transistor Level Design

A logic function can be realized with the help of transistor circuits of various kinds (with different topologies).

Each of the circuit kind (topology) represents a unique *logic style* (static logic, any one of the several dynamic logic styles *e.g.* domino, 2-phase dynamic, 4-phase dynamic, CVSL *etc.*).

Once again, the designer is faced with a multiplicity of choices — of logic design styles and associated trade-offs.

3.7.6 Stick Level Design

Transistor level diagrams are usually not directly translated into layouts — an intermediate translation step that converts transistor diagrams to stick diagrams is often used.

Stick diagrams capture the layer information and topology of the layers that are to be used in the layout. This metric-free notation provides the right abstraction that permits the designer to experiment with layout ideas and arrive at an optimal layout strategy.

3.7.7 Physical Design and Layout

Stick diagrams are converted to detailed geometrical design (layout) with appropriate widths, lengths, separations, overlaps between the layers *etc.* in view of the design rules and electrical considerations.

Design rules represent the constraints on geometrical patterns on different layers imposed by the considerations of reliable manufacturability of the circuit in a given manufacturing setup (foundry).

3.8 VLSI Design Methodologies

Different design methodologies differ in their choice of number and levels of design abstractions used during the design process and the manner of constraints on the translations between the abstraction levels.

These constraints are usually in the form of use of a particular structure type at the lower level of design abstraction while translating the design description that exists at a higher level of abstraction.

For example, while translating from the logic level abstraction to physical level, popular design methodologies are :

1. FPGA/CPLD.
2. Gate Array.
3. Standard Cell.
4. Full Custom.

The popular methodologies for implementing the control part are :

1. Hardwired Control.
2. Microcoded ROM-based Control.
3. PLA-based Control.

The choices for a sequential circuit design methodology are :

1. Synchronous design — clock distribution architecture to minimize clock skews.
2. Self-timed design.

3.9 Issues of Testing

VLSI designs require *two* kinds of testing :

- Functional testing (of the design) — for verifying that the design has correct functionality (as per the specifications).
- Production testing (of the chip) — to ensure that the produced chips have no manufacturing defects.

Functional testing of the design is done using functional test vectors which try out the design's functionality under different input conditions.

Production testing focuses on finding out if there are any manufacturing related defects. They typically aim to catch the *stuck-at* faults in a logic level circuit representation.

3.10 System Level Design Issues

For system level design, additional issues that need to be addressed are :

- Choice of the overall system architecture (partitioning into blocks).
- *What* block to realize *how* (*i.e.* map onto what components — off-the-shelf chips, custom ICs, microprocessor with associated software) ?

- Assessment of product cost, development cost and time for alternative architectures.

After deciding on an optimal architecture, the hardware design for hardware blocks and the software design for the targeted microprocessor(s) and/or DSP(s) for software blocks are carried out simultaneously while still maintaining an overall design representation to check against the current implementation (*hardware-software codesign*).

The issue of overall system test design / test architecture is again very important (*e.g.* self-test routines, JTAG conformance).

The implementation checks again need to be addressed *e.g.* thermal analysis, noise analysis, cross-talk analysis, transmission line effects — reflections, EMI/EMC issues.

At times, to design application specific hardware that implements certain algorithms, for reasons of speed the algorithms themselves need to be restructured so that they lead to a structured hardware (VLSI design) implementation.

Besides the use of abstractions and the hierarchy they imply, concepts of regularity, modularity and consistency in approach are put to effective use in tackling the complexity of a VLSI design and enhancing the chances of success while cutting down on the design time.

3.11 Summary : VLSI Design Methodology

Steps :

I. Decomposition / Partitioning / De-construction

II. Composition / Synthesis / Construction / Implementation

- Decomposition : Top-down $\downarrow$
- Synthesis/Implementation : Bottom-up $\uparrow$

Go down decomposing hierarchically until the blocks resulting from the decomposition process are simple enough to be synthesized (implemented) from available lower-level blocks or by a CAD tool or manually implementable.

Then construct the full implementation by traversing up the decomposition hierarchy.

3.12 Design Flow Using HDL

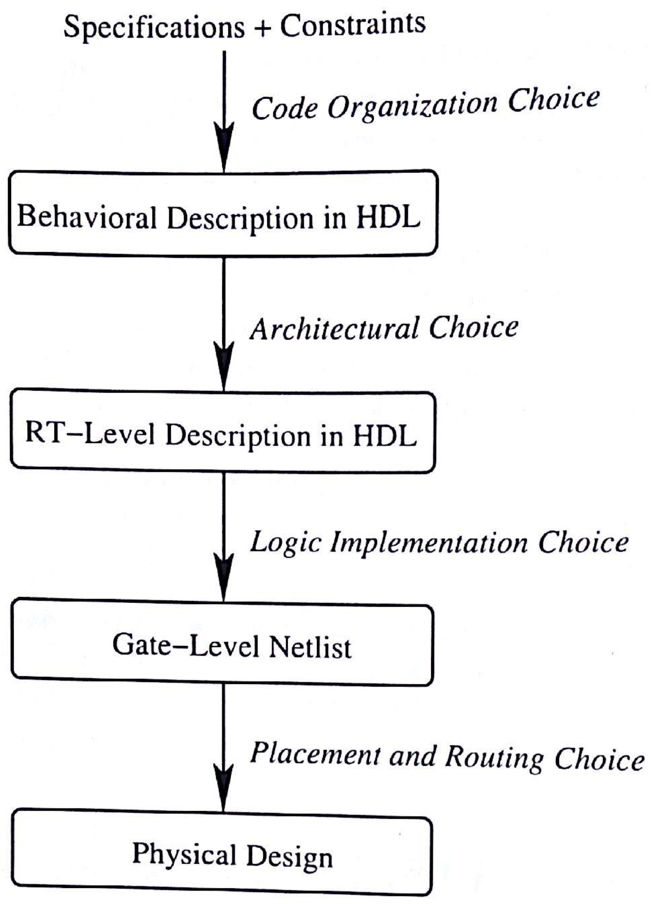
Specifications + Constraints
Code Organization Choice
Behavioral Description in HDL
Architectural Choice
RT–Level Description in HDL
Logic Implementation Choice
Gate–Level Netlist
Placement and Routing Choice
Physical Design

Figure 3.10: **HDL-Based Design Flow**

3.13 Front-End Design Steps Using HDL

3.13.1 Behavioural Design

To capture the behaviour of the chip in a formal form using a hardware description language (HDL).

- This step involves coding of the behaviour of the chip (usually specified in a natural language description or other informal means *e.g.* tables, diagrams *etc.*) in a formal hardware description language (HDL).
- The HDL description formally captures the functional behaviour (input to output transformations done by the chip) and timing behaviour at the pins of the chip.

- The description has no regard as to how the behaviour would be realized in a hardware form.

3.13.2 Architectural Exploration and RTL Description

Architectural Exploration involves choosing the number and type of processing elements and implementing/realizing the behaviour of the chip using these processing elements.

- All the transformations/operations on the operands contained in the behavioural description can be implemented by routing the operands through the processing elements available in the architecture — over a series of steps of the transformations / partial transformations and storage of intermediate results. An HDL description describing/capturing the above is called an RTL description.
- Architectural exploration and the associated design translation from behavioural to RTL level abstraction has traditionally been done manually. Designers have typically chosen 'optimal' architectures after necessary (or at least some) architectural experimentation.
- High-level synthesis tools (that translate behavioural description to RTL description while exploring different architectures) have begun to help the designers in selected application areas (such as Application Specific Digital Signal Processing elements).

  In a general sense, the high level synthesis tools, given a datapath architecture *i.e.* allocation of resources (processing elements and interconnection resources), can schedule the processing contained in the behavioural description onto the datapath architecture in a sequence of processing steps supported by the datapath.
- High level synthesis tools can also iterate on resource allocation to fit the constraint of implementing the behaviour in a certain number of steps.

However, architectural synthesis and optimization by-and-large (with the exception of restricted classes of problems) is still a human prerogative — with tools assisting the humans to manage the details.

3.13.3 Logic Synthesis

- Logic synthesis involves converting RTL description into optimal logic gate level netlist — subject to design constraints specified by the designer (*i.e.* area, speed or power).

- Given a library of logic gates/combinational elements and basic sequential circuits (*i.e.* latches, flip-flops), the logic synthesizer produces the optimal network of combinational elements and sequential circuits subject to the constraints specified by the designer.

3.13.4 Testability and Test Generation

- After synthesizing the logic level network, the designer can check the testability of different parts of his design and can enhance the testability of the design by guiding the tool to add testability hardware to the design based on one of the DFT techniques supported by the synthesis tool. This produces the final gate level netlist for the design.
- Automatic test pattern generation (ATPG) for the design is then done to generate the test data for testing the fabricated chips for manufacturing defects.

3.14 Back-End Design Steps

3.14.1 Floor Planning and Physical Partitioning

- This step partitions the chip-area into area segments and allocates logic gates/ logic blocks to be placed in each area segment.
- Floor planning provides a useful level of hierarchy in the optimization of the physical design (layout generation) by taking advantage of the designer's knowledge of functional and logical hierarchies of the design.

3.14.2 Placement

- Layouts corresponding to the logic gates/ logic blocks allocated to each of the area segments are then optimally placed within the area segment to minimize the estimated length of interconnect wiring among them.
- If the tool estimates the area segment to be small to hold the layouts of all the gates and their anticipated wiring in it, it communicates the same back to the designer in some manner and increases the area of the segment.
- The designer can also alter the aspect ratio of the area segment.
- Managing short interconnection lengths for the critical path(s) is one of the objectives of good placement.

- This is typically easier if all the logic elements in the critical path are assigned to the same area partition. Control of critical paths that span several area segments is typically more difficult.

3.14.3 Routing

- Once satisfactory placement has been done for all the logic gates / logic blocks in their corresponding area partitions, routing of interconnections within each area segments and among different area segments is then completed.
- This step might require some increase in the areas of some of the area segments and global routing space in-between the area segments if routing is not possible in the available area.

3.14.4 Parasitics Extraction and Back-Annotation

- Parasitic interconnection capacitances are extracted from the layout and their influence on circuit delays is imported (or back-annotated) into the circuit model.
- Static timing check and simulation (post-route timing check and post-route simulation) are then carried out to ensure that the layout circuit will meet the speed / timing specifications.

3.14.5 Design Audit

- Design audit reports are also generated to ensure there are no problems of clock skews, violation of transition time rules for clock nodes and other circuit nodes, excessive current densities through sections of supply and ground wiring.
- Consistency of physical and logical design databases is also checked (logic *vs.* layout check).

3.14.6 Design Rule Check

- Design rule check (DRC) ensures the manufacturability of the design on the target wafer fabrication process.
- It ensures that no geometrical features (widths, separations, overlays) have been used in the design that are inconsistent with the resolution of the process.

3.14.7 Foundry Services

- PG Tape Generation.
- Mask Making.
- Wafer Fabrication.
- Device Packaging.
- Packaged Device Testing.