Subject

Microelectronics

MASTER ON INFORMATION AND COMMUNICATION ELECTRONIC SYSTEMS

Integrated Circuit Design Flow. Overview

Student: Vladislav Rykov
e-mail: vrykov1@alumno.uned.es
Subject: Microelectronics

Abstract

With the increasing technology advances the integration density of integrated circuits (ICs) still grows proportionally with the Moore's law and expected to follow this trend in the near future. Albeit a subject of the IC design remains hidden from the most computer engineers and low-level programmers, it appears to be of high importance for thorough electronic product understanding, especially in cases of specific-purpose product design.

This work aims to uncover all important steps of the IC design and give detailed insights into its crucial phases. It is presented not as a manual, but rather as a theoretical guide which introduces basic as well as advanced concepts pertaining to this process. General design methodology, hierarchy abstractions, and special design considerations are introduced.

Keywords: Integrated Circuit (IC) design, microarchitecture, Register Transfer Level (RTL), Y-chart, synthesis, simulation

Index

1 Introduction **4**

2 Design Perspectives **6**

2.1 Hierarchy Abstractions 6

2.2 Y-chart: Behavioral, Structural and Physical Perspectives 8

3 Design Flow **13**

3.1 System Specification 14

3.2 Architecture Design 16

3.2.1 Architectural domain 19

3.2.2 Algorithmic domain 19

3.2.3 Data Dependency Graphs 20

3.2.4 Architectural Isomorphism 21

3.2.5 Iterative Decomposition 22

3.2.6 Time Sharing 22

3.2.7 Pipelining 23

3.2.8 Replication 23

3.3 Storage options 23

3.3.1 Area Implications 24

3.3.2 Memory Capacity 24

3.3.3 Latency Concerns 24

3.4 Alternatives Evaluation 25

3.5 Design Verification Procedures 26

3.5.1 Testbench 27

3.5.2 Design-Rule Checks 27

3.5.3 Circuit Extraction 28

3.5.4 Static Checks 28

3.5.5 Formal Verification 28

3.5.6 Other Techniques 29

3.6 Design Entry 29

3.7 Simulation 31

3.8 Prototyping with FPGA 32
3.9 Synthesis 32
3.10 Static Timing Analysis 34
3.11 Physical Layout Design 34
3.12 Post-layout Simulation 35

4 Special Considerations **35**
4.1 Regularity 35
4.2 Modularity 35
4.3 Locality 36
4.4 Design Challenges 36
4.4.1 Design for Testability 36
4.4.2 Design for Manufacturability 36
4.4.3 Design for Reliability 37

5 Conclusions **37**

1 Introduction

Marriam-Webster Dictionary gives a short and simple definition of an Integrated Circuit (IC), that is, a tiny complex of electronic components and their connections that is produced in or on a small slice of material (such as silicon). This simple definition outlines the basis for an industry which has kept a growth rate of 53% during last 50 years; the industry which annual revenue reached 481 billion of dollars US in 2018 ("Global and China Integrated Circuit Industries Markets", 2019); the industry which involves an engineering processes on a scale that a blood cell can be denoted as giant. Simple, yet incredibly complex.

When it turns to ICs it is impossible not to highlight their fundamental components, transistors. In 1947, Bell Laboratories which undoubtedly became historically important technological work environment, gathered three brilliant physicists. Walter Brattain and John Bardeen under the supervision of William Shockley invented the first transistor (Brinkman et al., 1997). Initially, it was called a point-contact transistor, 2 gold contacts closely placed on a piece of germanium crystal. Later, in 1956 they were awarded for the Nobel Prize in Physics. A replica of the first transistor can be seen on Figure 1.1 It was the beginning of a new era of information technologies.

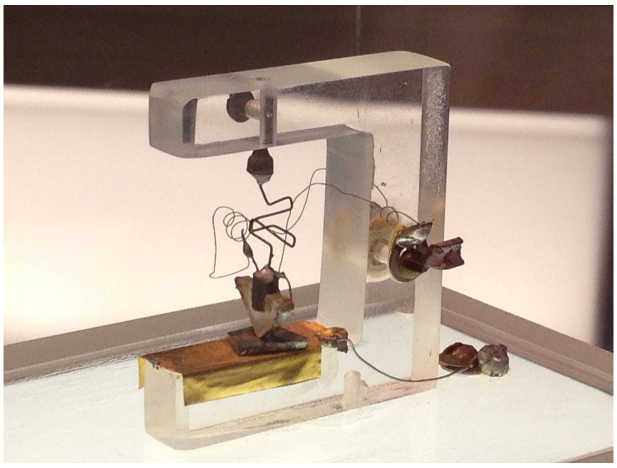

Figure 1.1: The replica of the first transistor

A decade later, an electrical engineer, namely Jack Kilby, came up with a concept of what we call today the IC (Kilby, 1976). He believed that all circuit components such as capacitors and resistors can be made of a semiconductor, and that all these components can be placed on a common semiconductor base which later was called wafer. The development path of computer technology was fruitful in terms of Nobel Prizes nominations and awards. Kilby was awarded for his invention 62 years later, in 2000. Figure 1.2 presents the prototype made by Kilby for the first IC.

The time was passing by, and, 50 years later Intel's processors were successfully accommodating more than 4 billion transistors. As was mentioned in the beginning, the growth rate during this period was 53%. This tendency was observed by Gordon Moore and became the basis for the well-known Moore's Law (Moore et al., 1965).

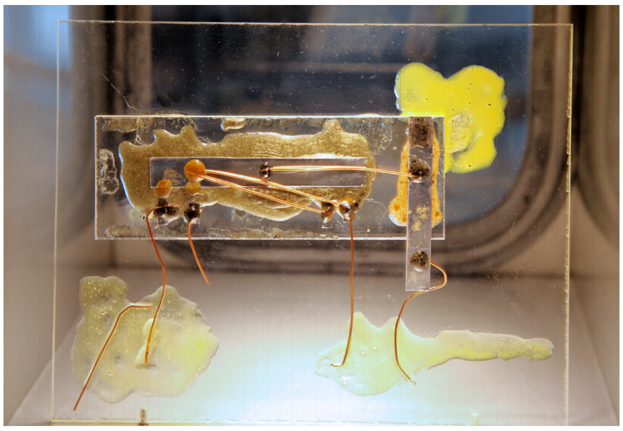
Figure 1.2: First IC prototype made by Jack Kilby

In a meanwhile, technologies related to engineering of fabrication, such as materials science, chemistry, physics and mechanical engineering were rapidly developing towards the miniaturization of components and improvements of fabrication processes. Gradually, transistors became smaller, faster, more power-efficient and cheaper to manufacture. These advancements incurred vast changes in society and humanity as a whole.

As the Moore's Law states that every 18 months the number of transistors doubles. With the fulfilling of this trend, there appeared a metric which scales the level of integration of ICs. The key parameter for measuring the scale of integration is the number of gates in an IC. The Table 1.1 shows the corresponding classification with metrics.

Level of Integration	Number of Gates
Small-scale	10
Medium-scale	1000
Large-scale	10000
Very large-scale	> 10000

Table 1.1: Levels of Integration and corresponding number of gates

The classification of integration level became meaningless since the number of gates was following the Moore's prophecy. Finally, all ICs up to 1980s stuck with the very large-scale integration term.

The other side of the described integration level development and fabrication processes was the increasing complexity of the design process and sophisticated procedures created to increase reliability of the IC production. Modern microcontrollers designs integrate processors, I/O interfaces, memories, and even Radio Frequency (RF) analog circuitry. The design process engineering and verification become incredibly important for successful designs.

The design process should be partitioned and shared among different engineering teams. Multiple levels of abstractions must be defined and the principles of structured design be applied.

Despite all the abstractions and complexity, the design of IC can be provided with general guidelines in a flow diagram form. The present work will consider the design steps in detail and provide consistent overview of the design process.

The document consists of several chapters. The first chapter introduces different IC design perspectives. The second chapter goes deep into IC design flow. The third chapter is about special considerations not covered in the design flow yet important to be mentioned. And finally, the document finishes with a conclusive meditation.

2 Design Perspectives

Integrated Circuits are created to serve a purpose. Be it as a general purpose MCU, or an application-specific purpose as ASIC, it is clear that a design should be guided according to certain requirements and criteria. There are multiple levels of requirements which will be discussed in this chapter.

In order to satisfy the requirements, modern ICs meticulously specify architectural details which later translated into dozens of functional blocks and communication interfaces. The levels of complexity formed in this manner are managed with abstractions, and the key term which is critically important in this context is the hierarchy.

2.1 Hierarchy Abstractions

There are many different forms to describe the hierarchy within ICs design. Most clear form is the one, described next in descending order of complexity. Big complex systems can be partitioned into a number of cores. Multiple units form one core. Each unit is built from various functional blocks. The blocks are composed by cells. And, finally, the cells are assembled from transistors which are the fundamental components of any IC design.

(Weste and Harris, 2015) propose partitioning of the overall digital design into five levels of abstraction: architecture, microarchitecture, logic, circuit, and physical designs. In the upper levels are placed functional requisites which later achieved by more technically defined bottom levels.

The architecture design stands as the highest level of abstraction. It describes how a system should function. A good example would be a microprocessor architecture. It defines a register set, Instruction Set Architecture (ISA), and memory model of the system. Microarchitecture design reveals a way in which the architecture design is arranged into functional units and registers. For example, Intel Pentium and AMD Athlon are kinds of microarchitectures which bring quite distinct trade-offs on power and performance, and differences at the lower levels as transistor counts.

Logic level of abstraction defines how functional units within a microachitecture are constructed and which features they have. An example would be an integer adder unit that can be based on carry select (Bedrij, 1962), ripple carry (Fang et al., 2002), or carry loockahead (“Carry-Lookahead Adder”, 2015) techniques. Circuit design level specifies the way the smallest components, that is, transistors, are used to satisfy the required logic design. Following with the example of adders, there are various circuit designs for each technique as pass transistors, CMOS or domino circuits can be used. Ultimately, physical design level defines a particular placement of components and the resulting IC layout.

It is worth to note that all abstract levels are cohesively dependent on each other and all contribute to reaching the desired objectives of a design. Dealing with these dependencies requires a parallel design approach as much as possible. IC design engineers need a good level of physical and circuit design background in order to sufficiently evaluate costs of modifications on microarchitectural level.

This being said, it is much easier to understand the system observing it from the highest level in the hierarchy, viewing each individual component as a black box with clearly established interfaces, compared to bare transistors view. Schematically, an IC design can be represented as a tree structure with the root being a complete chip design and nodes as subsequent abstract level units as presented in Figure 2.1.

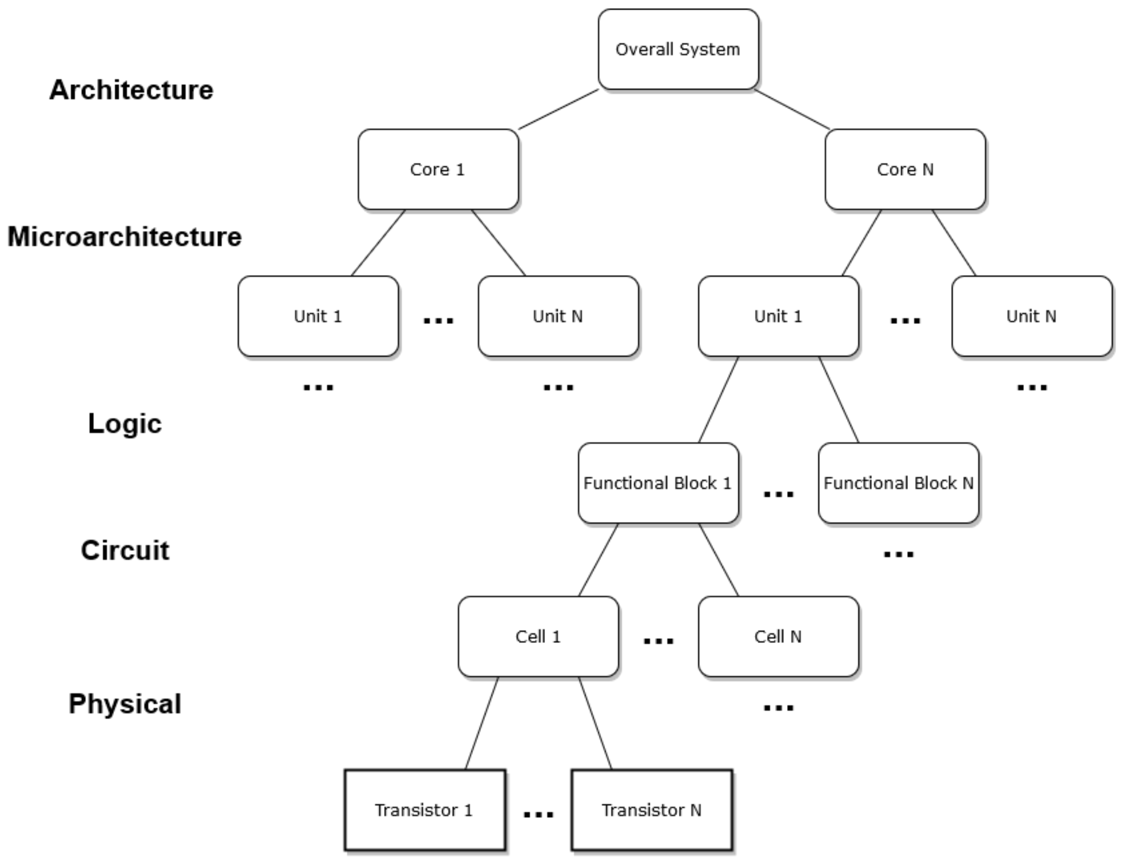

Figure 2.1: Integrated Circuit Design Abstractions Structure

2.2 Y-chart: Behavioral, Structural and Physical Perspectives

The IC design is a task that can be approached from different perspectives, and different tools can be applied to organize the given high level of complexity. Among other tools it is suitable to highlight the Y-chart.

The Y-chart is an alternative form proposed by (Gajski and Kuhn, 1983) to face design of a complex, normally electronic system in a systematic way with the aim to automate the process. It is formed by three radial lines or axes. Each axis stands for a particular perspective that a system can be seen from. Another part of the chart is circles with a common center which represent multiple levels of abstraction. The representation of the Y-chart is presented in Figure 2.2. Each perspective has various abstraction levels which start from higher levels - outer circles and descend to the most simple elements, transistors.

The Y-chart points out that an IC can be outlined with regard to three perspectives:

- Behavioral
- Structural
- Physical

The design cycle starts with consideration of behavioral perspective, transforming behavioral specifications into structural ones and finishing with details on the physical level.

Behavioral perspective indicates what a particular system or circuit does without any technical details. It can be seen as a front-end of the design view. The analogy of a black box is fitting well in this case. The design is observed as a black box with defined input and output interfaces. The most important thing on this perspective is a relation between corresponding inputs and outputs. However, such characteristic as clock signal and its frequency may also be important. For example, managing battery charging process can be the highest level of the behavioral perspective. More precise details can be further defined as voltage and current sensing, temperature control, timings and so on.

Structural perspective defines a structural description at each abstraction level. This perspective determines connectivity among individual modules required for desired behavior. Once a behavioral specification is defined it is always possible to implement it in multiple ways. The alternatives in this perspective usually vary in terms of performance, circuit complexity, and power efficiency on one side and testability, fabrication technology on the other. Sticking with the previous example of the battery charger IC, the highest structural levels will be composed by a charge control, timing circuitry interconnections, whereas the lowest level will represent transistors and other components interconnections.

The back-end of the design view is physical prospective. It specifies the arrangement and placement of the components within given space constraints. At high levels it may consist of a floorplan and at the lower level of geometrical forms of components.

In the course of the design process a series of decisions must be taken on each level of abstraction represented by circles in the Y-chart. As can be observed on Figure

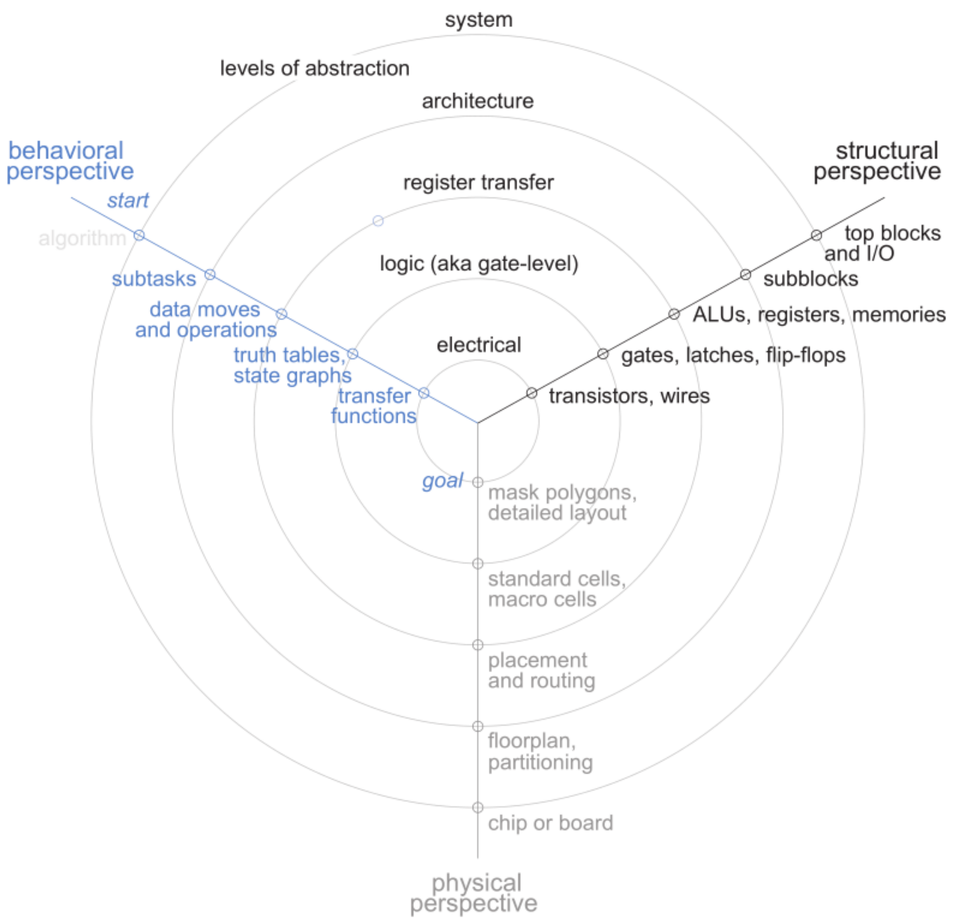

Figure 2.2: Y-chart hierarchy abstraction applied to an electronic system

2.2, there are five levels of abstraction: system, architecture, register transfer, logic and electrical. The next listing will generally characterize the decisions taken at each level.

System The decisions of crucial importance are taken on this level. They transcend the rest of the lower levels and affect as none as other level the final outcome of the design. Next issues are normally addressed and specified on this level:

- Operating conditions, general and specific functionality, characteristics of interest as costs, power, performance and so on
- Functionality of the system is split into subtasks
- Tradeoffs between software and hardware are explored and evaluated
- Decisions about main building blocks are taken.
- Data exchange protocols and interfaces are decided
- Operating modes, data formats, failure-handling routines are decided
- Defined subtasks are modeled and evaluated. After lower-levels feedback received refinements from behavioral perspective are carried out

The nature of a system that is being designed is the factor which has greater influence on design procedures, software tools, acceptance criteria, and design expertise.

Algorithm

Algorithm is not a direct part of the system level, however it is one of the major steps in the IC design, placed between the system and architecture levels.

Previously defined data processing requirements must be characterized with concrete computations that will be translated into hardware. Following list summarizes this step.

- Series of algorithms and respective formal representations (cellular automata, final state machine, or fuzzy logic) are proposed
- Optimizations on memory and computation complexity are estimated
- Arrangements for accommodation of accuracy and complexity tradeoffs are decided
- Analysis and contain-effects of finite word-length computations and exceptions are evaluated.
- Number representation schemes are decided
- All available alternatives are evaluated and the most suitable one is picked up
- Final resources optimizations are specified with more accuracy and details (memory, logic operations, and arithmetic)

The ultimate outcome of this stage is a bit-true software implementation which is used for verification of important parameters of an application, such as error rate, signal-to-noise ration (SNR), data compression factor, etc.

Architecture

IC architects determine application-required hardware resources and coordinate their interactions in order to come up with a desired algorithm and meet initial application constraints. The suggested arrangement of hardware must have structural features of future IC and yet maintain form of abstractions.

The interdependence between levels is more clearly appears here as the selection of target technology implies associated limitations and possibilities.

Eventually, the architecture design begins with quite abstract notions and step by step obtains a more detailed form. The process is described in the following list.

- The task of computation is projected over hardware organization.
- The interactions among subtasks are organized
- Hardware resources are allocated for every subtask
- Datapaths, that is, subcircuits which handle payload data, and their corresponding controllers are decided. The controllers govern datapaths according to the system state signals
- The choice is made among registers, off-chip or on-chip RAM, if needed

- Protocols and communication interfaces are decided
- A degree of parallelism in hardware is determined
- The use of pipelining and the number of stages are esteemed
- Manufacturing process, circuit style, and fabrication technology are decided.
- Which cell libraries in CAD software to use are decided if needed
- The first estimation of chip sizes and cost are are made

As a result of the work on this level of abstraction a high-level block diagram is generated. It contains controllers, datapaths, interfaces, memories, etc. An initial floorplan is estimated. The architecture is verified by simulations presented in further sections.

Register Transfer Register transfer level (RTL) is sometimes considered to be a part of the architecture level presenting more low-level architectural details. For the sake of clarity and following the hierarchy presented in Figure 2.2, it was taken and presented separately.

On this level the IC is exhibited as a series of storage functional blocks made by combinational subcircuits. The steps on this level presented below.

- Logic and arithmetic units implementation features decided
- Microcode or hardwired logic for controllers implementation is selected
- Decide between random logic or ROM usage
- Plan IC operations scheduling
- Bind computation operations to processing units
- Locations for shimming registers and pipelining are thought
- A degree of combinational depth and a way of its balancing are designed
- A clocking discipline is chosen
- Primary clock period time is determined
- Communication lines are considered for use of unidirectional and bidirectional buses
- Test schemes are determined to ensure verification process
- Way of IC initialization is devised.

The result of this stage is a number of more precise diagrams where every part of the IC starting from major blocks and finishing down with particular registers. It is important to note that the major blocks functionality is defined in behavioral terms and not in structural. Simulations is a crucial tool on this stage.

More detailed data allows more detailed refinement of the floorplan. After the refinement comparisons of die sizing and costs can be better verified.

On this stage better suited design level for each functional block is chosen. There are three options: schematic entry, synthesis, and hand layout.

Logic

On the logic abstraction level automation is prevalent part of the process. Major tasks have to do with specifications transformations into gate-level netlist and Boolean equations optimizations.

An IC architect opts to choose among semi-custom, full-custom or Field-programmable logic fabrication depth; which software libraries to use for cell-level design; dynamic or static circuit style; most appropriate fabrication technology to use; and, finally, the manufacturing process.

The implementation details are more clearly defined and thoroughly optimized. Timings and power-related characteristics for different IC modes are evaluated. Parts that narrow performance during pre-layout simulations are recognized and optimized. The outcome of this stage is the entire IC gate-level netlists with passed design rule checks and verification, power consumption estimation, and logic simulations.

Physical

The design on this level of abstraction is majorly in charge of organization of all subcircuits and functional blocks with their connection buses and signals for placing them on a wafer. The most important task here is floorplanning which consists in arrangement of major blocks into rectangular zones on the wafer taking into account signal propagation concerns. Clock lines allocation and power plans issues are addressed here.

In order to make possible reception of external signals and output of the internal ones a padframe has to be generated and necessary connections done. The connection of the IC wires with the padframe is the final step in the design and is called chip assembly.

The IC design shares with a PCB and FPGA designs one important step, place and route (PAR). During this step all components, down to the smallest ones, receive concrete locations on the chip die. Often, after the PAR, an additional optimization is required for the circuit logic.

Software used to perform the IC design is called Integrated Synthesis Environment (ISE). The output ISE files hold information that represent millions of polygons corresponding to their respective mask layers.

The overall IC layout must be thoroughly checked to prevent any possible failures. For this purpose exist verification routines normally offered by ISEs. Typical routines are listed below.

- Design Rule Check (DRC). Manufacturers can offer different processes with certain geometric limitations. These rules have to be introduced into the corresponding ISE configuration and examination run.
- Layout extraction process captures prepared netlist and a matching between the layout and schematic is done. The purpose of the matching is to ensure that the schematics are correctly transformed into the layout.
- Though design for manufacturability is normal practice in the IC design, manufacturability analysis is required. It evaluates complete IC suitableness for selected manufacturing process. Possibly hazardous spots are found and exposed to the designer.

- Post-layout simulation.

During the inter-perspective transformations there is an urgent need to ensure the correctness of the performed transformation. Thus, each transformation is tested with a comparison of the design before and after the transformation. For instance, if an IC response speed is specified, then a test is prepared and run from the structural perspective with the evaluation data from physical perspective. In this manner, fulfilling of the design goals can be ensured.

According to the Y-chart the design can be perceived as requirements transformation from one domain into another. The flow is oriented from behavioral perspective, to the structural and finishing with the physical design. Each domain should be rigorously described such as the final system can be clearly trailed back to the top behavioral perspective. The power and usefulness of this approach consists of a possibility to design really large systems defining sequentially corresponding perspective levels of abstraction.

A design is created to solve a particular problem. As a designer goes considering each perspective, there are a plenty of options to decide on. For example, from the behavioral perspective one can opt for wired communications and standards and protocols to use. The structural perspective offers possible clocking strategies, circuit styles, or logic family that can be used. Physical perspective suggests geometry details, packages, boards.

The Y-chart overlap with the scheme presented on Figure **??** in a special way. The perspectives were historically divided into a hierarchy of design abstraction levels generally introduced in Section 2.1. Focusing particularly on digital electronics there are three levels:

- Architectural Level
- Register Transfer Level (RTL)
- Circuit Level

Modern software systems for IC design perform the transformations automatically, however, a designer must be aware of this process running in background.

3 Design Flow

As it was mentioned, IC design can involve systems of different complexity. No matter what complexity degree is presented it can be traced down to individual transistors. For relatively simple ICs, the design can be performed by one person, sometimes with no need for rigorous following structured design. However, in case of more complex designs entire teams are involved in the process what makes indispensable a presence of formal systematic design approach.

The previous section covered most prominent approaches for structured IC design based on hierarchy and abstraction concepts. Steps which are usually taken on each stage of development were generally described. This chapter aims to give another, more practical and full view of the design process presented in more cohesive and sequential manner.

The full design flow, from system specification down to the IC production is presented in Figure 3.1. The chapter is organized according to this diagram where each section will describe in detail given phase.

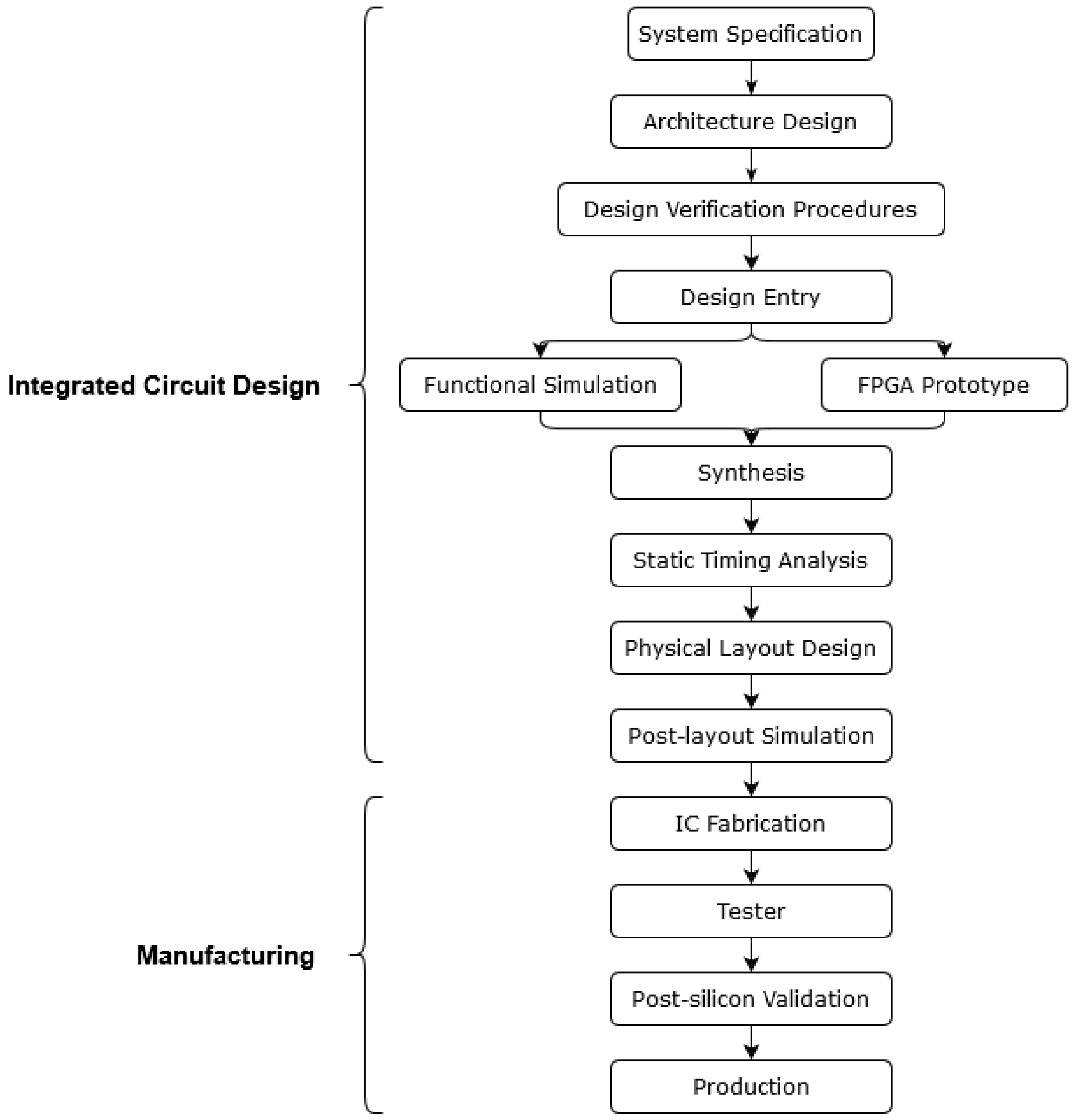

Figure 3.1: Integrated Circuit Design and Manufacturing Cycle Diagram

3.1 System Specification

The process of designing a new IC can be impulsed by customers who need to bring special functionality into their projects and found no suitable standard solution, or by IC manufacturers who want to cover a necessity in the market or simply bring new features and performance. No matter who triggers the process or of what scale a project is, it will always begin with a specification.

The specification is a document which clearly and explicitly defines the functionality that should provide the IC in form of a set of requirements, regardless how they might be achieved. The requirements can be presented concerning different aspects of an IC such as service, material, interfaces and so on. It can be presented in textual, graphical, or even using software representation.

Preparation of the specification has multiple steps which vary depending on who triggers the whole process. The first step is to prepare a preliminary specification. When an IC manufacturer is the one who triggers the process, then a market research has to be done beforehand identifying the demand for the potential product. Based on this research, the preliminary specification is prepared. At the first steps, it is highly important to have a contact with future users of the IC. It will allow gathering more information and help with the next step, preparation of a detailed specification.

When a customer is the one who triggers the process, then it becomes easier as no market research is required. Though, depending on a customer technical background the use case will require more or less deep study.

Next listing describes sections of a typical specification document (Yonehara).

List of Reviewers It is necessary to include names of engineers or titles of departments involved in the process. Customers also normally form part of the reviewers as they are who confirm the validity of the future design. Manufacturers or their representatives also have to be mentioned since they have to know if it is possible to manufacture the design under consideration with available technology.

Modification/Revision History This section includes all revision history with minor or major changes. This field is responsible for version control.

Table of Contents Some sort of index page where a list of all section with their corresponding names are exposed.

Glossary Depending on a project scale the Glossary section is present or not. Here all important abbreviations and proper project-pertaining terms are given a definition and explained.

Overview Overview of future functionality, capabilities and potential purposes and applications are unveiled in this section. On the other hand, functional requirements are included in the overview.

Performance Targets In terms of software development this section relates to non-functional requirements. That is, processing units frequencies, response times, power consumption, data transmission bandwidth, communication interfaces and so on.

Block Diagrams This section opens more visual perspective of how functional requirements will be reached. All chip subsystems on microarchitectural level are represented by blocks with inputs and outputs. The same section specifies I/O interfaces as well.

Figure 3.2 and Table 3.1 represent a snippet from concrete specification.

Graphs Tests is a vitally important part of the design process. Specification usually includes graphs with visualized information extracted from performed tests.

Tables Tables section includes reference data for interfaces, registers, and, generally, information of variable depth about IC and its subsystems.

Detailed Description In this section the most exhaustive descriptions of IC functionality take place. From the most top-level down to the bottom-most one. Interactions among subsystems, functional blocks, computational units are described, as well as methods and materials used for the manufacturing.

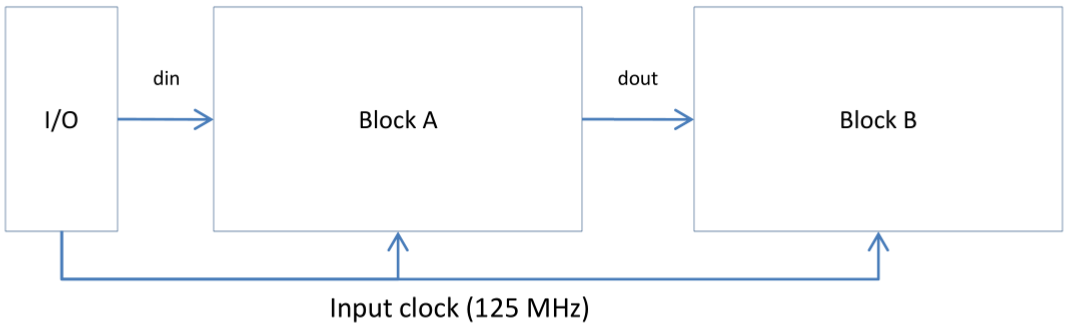

Figure 3.2: Block diagram snippet from a concrete IC specification

Port Name	Direction	Sources/Destination	Size	Description
CLK	Input	I/O	1 bit	Clock at 125 MHz
Din	Input	I/O	32 bits	Input data
Dout	Output	A and B	32 bits	Output data

Table 3.1: Specification for the block diagram from Figure 3.2

In summary, specification is a document, according to which the engineering team realizes the complete IC design, decides for which IC parts to use existent intellectual property or create a custom logic design, what functional-block level and microarchitectural-level communication interfaces will be like, the way of IC partitioning on all levels. Often, several teams are involved in the design, and specification is the point that helps to follow concurrent design flow. Finally, it is a place where all tests are documented and the results are matched to the initial requirements.

In order to follow certain standards there exist guidelines for IC characterization and specification for different industries like (*Guideline for characterization of integrated circuits*, 2013) and (*Performance Specification Integrated Circuits (Microcircuits) Manufacturing, General Specification for Department of Defence USA*, 2012).

3.2 Architecture Design

Architecture design is a process of deciding necessary hardware resources and their interactions for resolving IC-specific problems targeted by the specification (Kaeslin, 2008).

In the architecture design there is a list of priorities that guide the process. The said priority list is specified as follows.

1. Resulting architecture must yield the specified functionality
2. Meet specified performance, normally expressed in operation rate of throughput
3. Keep production costs optimized. Increase fabrication yield reducing circuit size for producing more functional parts per wafer
4. Energy efficiency must be concerned independently of an application if it will be supplied by batteries or mains.
5. Provide design agility, that is, the ability to rapidly move among possible modes of operation and to incorporate new features.
6. Time to market concern

The main reason for energy efficiency when a device supplied from mains is to reduce heat dissipation as much as possible.

Once the functional and non-functional requisites are defined in the specification document, engineering team can start to analyze it and extract architectural image of a future IC.

As it is known, the goal of any design is to carry out specific function or algorithm. This algorithm can be programmed with a programming language, i.e. C or C++, to be run on an instruction-driven hardware as MCU, CPU or DSP; or it can be performed by a hardwired electronics that fulfill necessary functionality. Thus, an IC designer has to take architecture-level decision - select a general-purpose architecture and program the algorithm, or to design a particular hardware architecture that carries out desired algorithm.

The aforementioned decision is the most important one before complex IC design process starts. It is worth to note that if a general purpose microarchitecture was chosen for the IC control than developers do not have to consider complex and extremely time-consuming issues as power and clock distributions, verification, tests, physical design, signal integrity and so on. They just focus on system-level architecture and functionality.

Normally, dedicated architectures are more energy-efficient and better on performance, though their designs consume much more time and human energy. On the other hand, there are certain algorithms that put limitations for the dedicated architectures. Memory-hungry, data-dependent and irregular algorithms are not appropriate for them. Accounting electronic data processing, event-driven systems that continuously interact with an environment, or user interfaces are examples of unsuitable applications for dedicated architectures. This unsuitability is given due to extremely complex mathematical models that need to be formed for IC formal description. Thus, these architectures are more appropriate for transformatorial applications where input data values must by transformed at output. Good examples are ciphering or filtering applications.

More precisely, there are certain criteria that can indicate when design of a dedicated architecture is certainly the best solution. Not all existent applications meet those criteria, however if some are met it is a positive indication towards dedicated architecture design.

1. Principal processing task are loosely coupled.

   It is intuitive to see that if an application can be easily split into separate tasks and the rules of interaction can be clearly defined, then it will be easy to put those tasks into functional blocks.

2. Simple and clear control flow.

   It means that how operations done does not depend on the data under processing, i.e. loop iterations number is always fixed; or an application does not require different modes of operations or varying configurations. The positive outcomes are that the required resources as memory can be easily estimated as well as resulting performance evaluated. The managing of such systems can be done by final state machines which are easily verifiable, simple, fast and efficient.

3. Finite precision arithmetics compatibility.

   The point is to avoid floating point arithmetics at all because it will increase minimal word width what leads to growing logic delays, parasitic capacitance, circuit sizes, and energy dissipation of arithmetic operations.

4. Regular and predictable data flow.

   Regularity is the basis for effective hardware resources shearing (time-sharing, iterative decomposition). Since functional blocks can be designed to transmit data between them over fixed local links communication overhead can be reduced, particularly interconnect and area delays.

5. Linear non-recursive time-invariant computation.

   These properties allow to reorganize the operations on data in different optimized ways so that, high throughput can be achieved.

6. Justified non-excessive storage requirements.

   Memory requirements must be upper-bounded. Excessive memory requirements reflect on IC area. If the memory occupies about a half of IC area, then it is impossible to integrate it an economical way and it must be placed outside.

7. Avoid functions that employ transcendental numbers.

   Ideally, the algorithm should not compute roots, trigonometric, exponential or logarithmic functions. And if it is the case, the actual domain must be defined and necessary values stored Look-Up Tables (LUT). The domain must be of limited size and humble accuracy approximations and interpolations accepted.

Considering a particular algorithm to match the mentioned criteria will undoubtedly help in taking the decision about dedicated architecture.

As was mentioned before, the markets often demand agility. The good thing, is that it is often possible to separate efficiency needs from agility. The reason consists in functional blocks which require efficiency. They are normally not prune to be changed in future. Interestingly enough, the same applies vice versa. For example, in a cellular phone often communication protocols, channel allocations, user interface, file systems are objects to changes, however frequency filtering, error correction coding and decoding, speech compression and decompression are usually kept the same.

With that said, the focus changes to how the actual architecture is formed, mainly transformation process. Two distinct domains architectural and algorithmic converge into the IC architecture. There are certain steps that have to be followed for each domain until the VLSI design appears.

3.2.1 Architectural domain

The focus here is put on the question of how to meet performance requirements for a particular algorithm using a minimum of hardware resources.

The answer lays in the organization of memories, controllers, datapaths, and the rest of necessary components, so that the computation is done with optimized IC parameters. The parameters are of different kinds as IC size, energy efficiency, agility, throughput, design effort and overall costs. Lastly, the output must be given according to previously established relationship. The latency, however can change.

The organization of functional blocks and units does not alter the actual implementation. Thus, making changes for the sake of improvement does not imply implementation losses at this level.

3.2.2 Algorithmic domain

The algorithmic domain focuses on achieving the minimum number of algorithmic computations given cost estimations for these operations. The designer tries to transform the original algorithm to one that is optimized for particular IC architecture or hardware. Not only operations, but number representation schemes and required data structures are to be optimized. From here comes that an IC alternatives normally are slightly different from each other give a particular set of optimizations.

In practice, the most evident representation or formal description of a particular algorithm is not a good point to start right away the designing process. There are always deviations from the original algorithm which allow optimizing energy efficiency and throughput when keeping as low as possible overall costs. Often, in order to suit a particular low cost model or keep the economical figures low, the efficiency or throughput are affected and the design suffers implementation losses.

However, in reality, adapting particular algorithm to the specific technical and economical features of hardware allows achieving huge improvements in IC size, agility, energy efficiency and throughput. The main task is to remove the computational weight, keep memory use as low as possible and accelerate computations avoiding unreasonable implementation losses.

Obviously, from one application to another, there will be a plenty of different trade-offs. Thus, the designer should decide which implementation losses are acceptable and to what extent the functionality may be affected.

On the scale of importance it would be not true to say that architectural domain is less important than the algorithmic. The algorithm design is of the most importance during the initial design phases.

The work of collaboration

IC designers are not the only staff that work in the development process. Actually, they are not the ones who come up with the algorithm design. System engineers have to tackle the right algorithm definition. They tend to think in abstract mathematical terms and despise important hardware architectural details. For them the algorithm is just a set of equations with corresponding operations. Often, forced by time constraints or any other pressure, they do not care about arithmetics used by the algorithm and can use floating-point calculations whereas a use of limited numeric domain can be sufficient for an application. This approach cannot be accepted for the IC design, thus, a bit-true implementation must be prepared first.

Normally, a lot of trade-offs and conflicting requisites must be reconciled until the ideal mathematical model of an algorithm can be considered as working and attractable at the market. Economical constraints always reduce design liberty and force desirable mathematical models to adapt.

While systems engineers design algorithms, hardware engineers are in charge of architecture design. The tight collaboration of these two groups is the key component for the successful solution. Actually, only through this collaboration reasonable trade-offs between ideal functional requirements and final IC functionality are found. Their intensive collaboration lays among specification, algorithm development and architecture design phases.

Concretely, systems engineers are responsible for evaluation of functional requirements indicated in the specification. Based on the performed analysis the design the algorithm. The hardware VLSI designers are in charge of architecture design and adapt it for implementation-specific technology. If the process is sequential and the interaction is poor, it is more likely that the final product will lack optimization due to previously mentioned reasons. In contrast, when the close interaction is the case, the best solution will be achieved.

3.2.3 Data Dependency Graphs

A useful tool used during the architecture development is a data dependency graph. It is used for algorithm representation. Operations are represented by vertices with weights which stand for time consumed for executing of this operation. Connections between operations are represented by arrows which also have weights. The arrow weights express the sum of computation periods that the next operation has to wait for the previous one to complete. Zero weight signifies that operations will share the same computation period.

By the way, the computation period is the amount of clock cycles between two successive computations. Another possible meaning for arrows is data transfer between operations and the weight, in this case, the amount of register involved in the data transmission. The data dependency graphs are very similar in appearance to typical state machines, though with few proper notations.

3.2.4 Architectural Isomorphism

In search of optimization, during last decades, VLSI designers came up with different techniques that allow to possibly find more optimal solutions not seen from the first glance. Since functional blocks or computational units are nearly independent units, their reorganization opens a room for architecture varieties they can be explored. The main condition is that the initial functional input-output relation will not change. If a certain architecture has a room for such alterations, then it is considered to be isomorphic.

This technique works together with the data dependency graphs. They help visually represent reorganized architectures. This way almost any architecture design can become a starting point for exploration for optimizations. Each possibility should be critically considered with advantages and disadvantages written down. Potentially, an advantageous complex architectural model will be elaborated this way.

Architectural alternatives need a formal way for their comparison. A mathematical model can be composed based on each data dependency graph. There a series of important variables that can be extracted if possible. Following listing reveals the most important ones and adds a brief explanation.

- Data throughput

  This measure is often considered to be the most important one as it expresses the IC performance. Normally, it is expressed as processed data items per time of unit, i.e. frames per second or samples per second.

- Latency

  It expresses how many computational cycles elapsed from the instance a data entry came into a system until the corresponding result appears at the IC output. When the output appears at the same clock cycle as the input, then the latency is zero.

- Time per data entry

  This measure indicates how much time passed between two sequential IC outputs. Generally, it is expressed in time unit/data entry, i.e. s/sample.

- Energy per data entry

  Quite important is the energy consumption. This characteristic signifies how much energy was consumed during a computational operation over a data entry, i.e. μW/sample. It can be expressed differently depending on a field it is used within. In MCU and CPU fields it is usually expressed in mega- or gigaoperations / energy unit, normally expressed in a fraction of watt.

- IC size

  The IC size can be understood as IC overall complexity or physical area occupied by it, as intuitively indicates the term.

- Longest path delay

  The amount of time it takes to transmit data along a combinational path with the longest length is denoted as the longest path delay. Physical constraints of electron propagation time incur architectural limitations in terms of operation speed.

- Cycles per data entry

  It characterizes computational period, that is, the amount of computational cycles between two data outputs.

- Time-size product

  It merges the IC size characteristic with its throughput and, this way, becomes a value for IC evaluation in terms of space and effectiveness.

Useful dependencies are expressed during the architecture design based on these variables and, taken their consistency, the unknowns can be figured out. The same applies vice versa; useful dependencies can be extracted which can mean architecture limitations.

3.2.5 Iterative Decomposition

Among other design techniques, the iterative decomposition has a prominence and is a must for iterative computations. The idea is simple, that is, to share available resources in a way that a computational operation that is done in an iterative manner reuse available hardware as much as possible. Substitute several hardware computational blocks with only one sounds good, though a control unit will be required for iterations supervision. The disadvantage of this technique is that it reduces possible degree of parallelism and decreasing the throughput. Next data entry will have to wait all computation cycles until it can enter. To solve this issue, the pipelining techniques borrowed from a typical CPU architecture will store input data entries, always when their amount is reasonable.

The extreme iterative decomposition will lead to a design of one arithmetic logic unit that performs all basic operations required for a particular algorithm. The RISC architecture was inspired by this idea. The rest of design will consist in tackling of how to efficiently provide datapaths and keep implementation losses low.

In spite of reduced IC size and used resources advantages, when it comes to comparison to a more dedicated IC architecture, iterative decomposition has lower energy efficiency and performance.

3.2.6 Time Sharing

Previous optimization technique was designed for single data stream operation, but what is a parallel processing of multiple streams is required. This is the case of image or video processing, where numerous computational units are placed in normally two dimensions. Applications employing this approach process data locally and have high performance outcome.

Efficiency is important factor, but it is costly in terms of physical space and economy, especially in case of parallel operation. These inconveniences push the time sharing idea further. It consists in having one or few operational units and use them in a round-robin manner, in other words, to cyclically send slices of incoming data streams to each unit creating the virtualization effect. It is also called multiplexing.

A direct implication of this technique is the multifunctional use of a datapath the leads to the operational unit. Since, it may perform different operations, data coming through this datapath may be of different kind.

3.2.7 Pipelining

The pipelining is a very well-known optimization technique that is a must in processing units of higher degrees. Its goal is to increase throughput by creating separate stages with similar latency. Without these registers the operational depth is too high and next operation must wait for the previous one to finish. The combinational depth is reduced and the throughput in increased inserting specifically designed pipeline registers.

3.2.8 Replication

When the performance optimization is strongly requited the replication approach can be applied. The idea is quite similar to the time sharing, however using single data stream and identical functional units. If performance is not good enough, add one more identical operational units and use them in a round-robin manner. The improvement factor will be proportional to the number of identical operational units.

3.3 Storage options

Almost all modern ICs require the use of memory at some moment. Exceptions are ICs designed for very basic tasks or that serve as components in a more complex system, i.e. operational amplifiers. The memory can be of two kinds depending on an application nature. If an application is sequential, then the memory called functional, however, if memory is required due to sophisticated design of a dedicated architectures, then it is called non-functional as it is an optimization byproduct.

The choice for memory storage often falls into one of these three categories:

- Flip-flop registers, placed inside the IC
- DRAM or SRAM memory, placed inside the IC
- DRAM or SRAM memory, placed outside the IC

The decision which memory to use greatly affects the whole architecture design. There are several outcomes from the decision of which memory will be used.

3.3.1 Area Implications

Comparison between flip-flop registers and DRAM or SRAM is required considering which temporal storage to incorporate. It is known that flip-flop memory offers exceptional read-write speeds, however, its area efficiency is considerably poorer. This disadvantage is given due to the register cell structure, whereas SRAM and DRAM bit cells are much smaller due to their simple structure.

However, DRAM has several inconveniences that make its choice questionable. First, DRAM content cells must be refreshed regularly what consumes power even in idle state. Second, more than 1/3 of DRAM chip area is occupied by control circuitry, namely, sense amplifiers, address decoders, buffers and switches. The list is not exhaustive, even though, it carries considerable overhead.

Generally, DRAM or SRAM overhead makes it undesirable when memory requirements are low, and flip-flop or latched memories based on 4T or 6T cells are usually preferred.

3.3.2 Memory Capacity

As it was said, for low memory requirements flip-flop or latched memories are preferred. When the requirements get higher always emerges a comparison between on-chip memory complexity and economical effects and extremely cheap off-chip DRAMs manufactured in huge quantities and adapted for customer needs.

Drawbacks of on-chip memory placement grow proportionally with the requirements. This tendency often force designers to opt for off-chip DRAM commodities when a dedicated architecture put high requirements.

MCUs and SoCs designers are frequently found on the boarder with medium memory requirements. They have to find a compromise, and sometimes architectures with extendable memory - small amount of on-chip memory which can be extended with off-chip DRAM - come of their hands (*STC12C5A60S2 series MCUSTC12LE5A60S2 series MCU Data Sheet*, 2011).

3.3.3 Latency Concerns

Depending on which memory type is used the latency will be different. There are memories that have zero latency. In this case the memory content appears at the output within the same clock cycle as the input address was introduced. This way work register files or flip-flop and latched memories. In any other case, the latency is greater than zero and it seriously affects the overall architecture design.

The latency can also vary due to DRAM modes implemented. It will be different for sequential accesses for standard Fast Page Mode (FPM), Extended Data Out (EDO), Burst EDO (BEDO), or Audio RAM (ARAM) (Matas and DeSubercausau, 1997). Independently of mode, paged RAMs affect the overall architecture.

Finally, the control circuitry that DRAM and SRAM carry inevitably increase the latency. This must be taken into account during the design process.

3.4 Alternatives Evaluation

Enormous efforts were brought for converting the architecture design process into an automated sequence of actions where the error is an exception. But the first steps in the IC design will always lack human criteria and will be an art-like procedure. The analytical methods provide impressive tools that allow in-depth analysis. Yet, it is not enough to reach optimum solution for a dedicated architecture.

Effective approach is to find out possible alternatives and explore them to a point where a reasonable comparison can be made. Extraction of advantages and disadvantages truly helps in this process. Though, it could seem to be an automated and mechanical process, it is not so in practice. Actually, it requires dedication, discipline, and creativity.

This section will provide an overview of the process of alternatives evaluation. Each step is separated, numbered, and provided with a brief insight.

1. Algorithm analysis.

   The process starts with the detailed, exhaustive analysis of an algorithm. Step by step the flow of data must be identified and character of computations evaluated. Hardware resources usage should be estimated and quantified to the maximum extent. Special attention must be paid

   - Data transmission rate between operational blocks
   - Memory requirements
   - Sufficient word width
   - Latency for all operational blocks

2. Determine necessary control circuitry.

   Computational flow must be controlled by direction points. Connections and interactions of such points with each other and with the outside world must be established. On the other hand, all data dependencies, necessary flexibility, and complexity should be analyzed.

   During this step there is a possibility to decide the degree of compromise between completely hard-wired architecture and instruction-controlled operation.

3. Start with a simple block diagrams.

   Instead of starting with a concrete data dependency graph, experience proves better to stare with a mere block diagram of major operational units. Data registers can be placed between operational blocks as needed. If RAM is preferred temporary storage, then latency issues must be further traced and verified during timing analysis phase.

4. Prepare a table where all key characteristics for each architecture variation can be easily, yet seriously, compared.

   Each row will represent hardware a particular resource. High-level operations also will take a row each. Then, the operations are decomposed into more primitive sub-operations that are matched later to sub-circuits. The decomposition follows until it becomes possible to evaluate quantitatively the whole operation. Some variables introduced in Section 3.2.4 such as area, energy, and path delays.

   This table can serve a starting point for hardware synthesis step. Since all operations are split into their primitives, which is basically the Register Transfer Level (RTL) operations, they can be described with Hardware Description Language (HDL) of choice.

   With the growing complexity the table will be rapidly populated. For comparing the alternative architectures, under each row must be saved a room for alternatives. This way, the comparison can be carried out at each level.

5. Important variables estimation.

   In the previous table the important variables were estimated for each most primitive hardware RTL unit. Here These values are summarized for the whole architecture variation. The longest path delay is the most laboriously calculated variable. It can vary due to possible low-level issues (White, 1989). In order to evaluate the whole architecture alternative it is quite important to have the longest path delay value. The issue solved by a premature simulation process. For its sake critical parts of the IC must be routed. Furthermore, the premature simulation also allows to estimate throughput, clock frequency and other variables, that could not have been estimated.

6. Critical analysis.

   The characteristic values obtained from the previous step are critically considered here. It will be possible to observe performance barriers and operations or sub-operations that over-demanding resources. These points must be examined, and, depending on which level they are found, the data dependency graph strategy for looking for alternatives can be reapplied.

7. Comparison against the requirements.

   Once the best candidate to follow with is apparently determined, the comparison with the initial requirements must be made. This will keep in focus the main objective and not let designers to deviate.

The described considerations and steps were given detailed insight due to their crucial importance withing the IC design process. They smoothly prepared the way for further more automated process of verification and synthesis which will be described in the next sections.

3.5 Design Verification Procedures

IC architecture is a quite complex undertaking and can have many levels of abstractions, full of details, implications, and operations. With such a level of complexity something can accidentally escape designer's attention and, finally, go wrong. And if exists this probability, it will probably happen. For this reason the design verification procedures were devised and became crucial for finding premature errors. In practice, during this process about a half of all IC errors are found.

Currently, the process of architecture design finishes with extremely detailed documents. The level of architecture details is very high. Depending on a design complexity, it becomes useless to cyclically employ circuit simulators as SPICE for the layout correctness checks. To cope with this situation, the design is split into the architecture level and the logic level. The first is checked using high-level programming languages as C/C++, and the logic level is simulated using HDL abstractions. The use of this set of tools allow a successful verification of the entire IC on mentioned levels. It is possible to check that the overall design is truly represented by its low-level circuits and sub-circuits. Also, power and latency requirements are checked during this process and matched with the specification values.

General design verification procedures will be summarized in this section.

3.5.1 Testbench

The testbech is a process that verifies the correctness of the circuit logic. It is a kind of script which reads a list of input and output values called test vector, and checks for mismatches. Input values represent IC input, that is, physical state of input pins. Output values represent physical state of the chip output pins when a given input is provided.

3.5.2 Design-Rule Checks

Any IC manufacturing process has physical limitations. This fact establishes geometrical constraints that IC designers must comply to. These constraints are called design rules. Mostly, it is presented in a form of a list where one entry has a constraint name and its value. Examples would be a minimum spacing between lines, minimum line width, or maximum layers overlap.

As a general rule, the design-rule checks are forthright, but there are possibilities where they become more complicated. This occurs when a designer wants to deviate from standard process, for instance, when certain cheap area must be taken advantage of in a special way. Then, the rule list is augmented to allow these operations (Baker and Terman, 1980).

The real design rules list is very long, but it is possible to come up with a simplified rules set which fits to the majority of current manufacturing processes. Examples of such lists are presented in (Mead and Conway, 1980) where special design-rule checker is employed.

3.5.3 Circuit Extraction

After the design-rule check is done, it is assured that the IC layout complies to a certain manufacturing process rules. Next step is generation of masks for IC manufacturing process. Though, more problems can be hidden in the mask descriptions generated by software used for the design. Circuit extraction is a process of extracting the electrical circuit from mask descriptions.

With this technique the right connectivity among nodes-transistors and design consistency are verified. Moreover, different errors can be found. As node numbers that were created but never merged and which appear in the list of transistors. These node can be the consequence of layout errors.

The connectivity is checked examining each transistor with its terminals and the relative position of its terminals to the connection lines and different layers. It is performed according to prepared algorithms, and different software designers use different algorithms. However, the idea is always simple, to raster the chip design and check individual connections.

Extensions can be also added to extract lengths, areas, width, and position of each layer and node. This information is very useful for further static analysis.

3.5.4 Static Checks

With the extracted previously information a number of static checks can be performed. The first of them is the matching of the extracted layout with the designed one. This is the simplest check.

Further, there are checks which make sure that every node in the chip can be pulled down and pulled up. Depletion mode transistors are checked to be used in an intended way. Threshold drops are also checked for errors. Using the geometrical characteristics of connection lines, all pull-down and pull-up ratios are computed and compared with the approved values.

3.5.5 Formal Verification

Theoretically, each requirement mentioned in the specification can be extracted and formalized. Formalized requirements become a foundation for formal mathematical analysis of the IC design correctness. There are many forms of formal analysis as Petri nets or signal flow graphs which, but not all of them are suitable for every system. Moreover, as IC is composed by many subcircuits and each of them is another system. This fact makes the situation even more difficult, as each formal analysis was developed for certain system type.

Nevertheless, mentioned limitations, formal verification is a powerful tool. According to the design algorithm, all operational steps can be transformed into logical Boolean expressions. Thus, these expressions are matched against actual corresponding IC logic.

One particular variation of the formal verification is extensively used called property checking. It is used to ensure that the designed architecture meet the design requirements written in the specification. The IC requirements are described using the property specification language as SVA or PSL. Based on this description a mathematical model of the IC is created. Finally, the IC specification requirements are compared with the created mathematical model to confirm the requirements match.

3.5.6 Other Techniques

In general, there are many techniques that designer can use for the design verification. Depending on the design phase, if logic is synthesized, post-synthesis transistor-level verification can be made, again, using the same test vector.

On later steps, simulations are actively involved in order to ensure the correct IC functionality. The simulations will be covered in further sections.

3.6 Design Entry

The IC architecture design finishes after its verification. However, regardless of already prepared and verified units, the IC is not designed completely, but rather partially. Design entry is a phase where complete system capturing occurs. Last decades it was moving towards substantial automation caused by several factors.

- Continuous increase in integration density.

  According to Moore's Law about every three years the circuit complexity increases by a factor of four. Therefore, customers strive to add more functionality to a single chip.

- Market pressure.

  Customers always aim to obtain advantage in their products for the competence. There is continuous pressure to shrink product development times.

- Time to market deflating.

These factors push design productivity towards automation and, generally, continuous improvements.

The main tool to aid in the design automation is the use of HDLs. Using HDL allows to describe a design entry in an abstract manner. The main benefit is the saved time for not working with low-level hardware details. On the other hand, designers can focus mainly on the IC functionality as the synthesis tools are in charge of automatic generation of physical and structural design aspects. Last but not least, it eases design reuse as all designs are stored in files that contain platform-independent parametric description of hardware.

Currently, the automation in process of transformation from structural design to physical details is highly developed and became industry de facto standard. However, the same for behavioral and structural aspects is not true. It is much more complex to precisely capture behavioral description and transform it to the structural form. The HDL is used for synthesis and translation of RTL hardware description into gate-level schemes which further transformed to cell-level designs using lower-level automation tools. Generally, HDL must express how IC subcircuits are interconnected and what functions they carry out indicating also time constraints. Table 3.2 presents current HDL with a corresponding brief description.

HDL	Description
Verilog	Language that can express behavioral and structural IC models, though with limitations in degree of abstraction. It facilitates testbench verification functionality. It has very similar syntax to C programming language, but does not force type checking.
VHDL	It is very similar to Verilog, but more powerful in abstractional aspects. Its syntax is similar to ADA.
SystemC	It is a C++ library extension used within a simulation kernel. Functions can accept arguments and clock information. The synthesis step is performed translating RTL statements to Verilog or VHDL.
SystemVerilog	It is a combination of Verilog and VHDL bringing the best of both. It allows object oriented programming paradigm, but unfortunately, currently does not support synthesis.

Table 3.2: Main currently used Hardware Description Languages (HDL)

Initially Verilog was designed by Advanced Integrated Design Systems and 1984 ("A brief Verilog history", 1996). Few years later it became de facto standard for IC design. VHDL was design by the USA Department of Defence for design of Very High Speed Integrated Circuits (VHSIC) (Pellerin, 1995). From there comes that VHDL is much more verbose and suitable for large team projects. Many Silicon Valley companies prefer to use Verilog, however telecommunication and defence companies use VHDL.

The design entry process is basically a programming experience, but it is crucial to remember that the final product is not software but hardware. The difference is that hardware always operates concurrently and software majorly sequentially.

Mainly, there are two HDL programming styles: behavioral and structural. The former defines what the IC does and the later defines the IC composition.

The code written in an HDL is interpreted by logic simulators which indicate whether the hardware meets specification requirements and generate waveforms alleviating the debugging. Using computer science terms, a logic synthesis software is similar to a compiler as it matches HDL statements to selected libraries of gates. If time requirements are introduced, it is able to optimize the design shrinking cells area. Produced by the synthesis circuits are not specially tuned for high speed, neither for extremely high level of integration; they are normal as would be designed by an experienced designer. With the current state of the art of the IC design, these processes are quite improved and suitable for the majority of dedicated architectures. It also makes possible the layout automatic generation.

The next logical step of digging into the HDL details would be exposing features of a particular language which is not the purpose of the present work.

3.7 Simulation

During manufacturing, the tests are of vital importance. Every sub-circuit and IC as a whole must be thoroughly tested. Special equipment is used for monitoring the IC behavior while known input is given and output is checked. Basically, the simulation resembles this process. It is a part of dynamic verification and aims principally to check behavioral side of the design.

As was noted en in the Section 3.5.5, statics analysis tools are limited. They are not so good for the behavioral analysis as the simulation.

The process of simulation is not a simple one. There are hundreds or thousands of operational units that need to be tested. Each of them must be accompanied by generated clock cycles. The focus of the simulation is observing the unit's output values. With such amount of operations, it becomes absolutely impossible and impractical to examine manually output waveforms, event occurred during the simulation, or actual output values rendered by units. On the other hand, a human agent can possibly miss one wrong output value, and, thus, the design time will be increased and the quality reduced. Therefore, visual inspection of the simulation results is distrustful. Special software must be adjusted to run simulations automatically and make checks against expected responses.

The expected responses are collected along with the corresponding input stimuli. This is called functional gauge. Functional correctness of sub-circuits or the whole IC is verified with the functional gauges. The test patterns are sometimes called testbench, though the original concept is different and explained in Section 3.5.1.

There are numerous software tool for the IC simulation. In spite of the difference, all of them follow the same sequence of steps. Typical simulation steps are listed below.

- Generate clock signal for the simulation.
- Read input vectors and supply values to units. This must be done in specified well-selected instants.
- Read response vectors from the units in a form of waveforms.
- Read expected output values and compare them with the corresponding rows of the response vector.
- Create a simulation summary where all found issues are listed.

Though, the simulation helps to find certain issues and correct them, it is not a perfect mechanism that reveals 100% of functional errors. Every simulation method is not exhaustive and has its defects and pitfalls. It is still a field of continuous improvement.

3.8 Prototyping with FPGA

Although the specification is written an as technically strict as possible language, comparing with the functional verification a serious gap is found in between. Often FPGA prototyping can be the best compromise between both.

Conceptually, a prototype constitutes a model that possess the same functionality as the final IC design, but without resembling it in electrical, timing, or architectural aspects. It can be developed as a program for MCU, DSP, or a general computer; can be written as Matlab or Simulink script or, as the most widely used, by setting up an FPGA.

The algorithm for FPGA prototyping is as follows

- Capture specification using formal mathematical methods.
- Formal description is adapted for the prototype.
- Actual prototype development. It should be implemented as general as possible for all the tests which can be performed.
- After the testing, depending on the outcome, the specification can be improved and corrected.

FPGA prototyping allows designers to discover mistakes at earlier stages of the IC development and polish the specification.

3.9 Synthesis

The process of synthesis, more precisely, can be defined as the process of selection an appropriate for a particular application architecture, and defining a mapping of the behavioral model, documented in and extracted from the specification, onto architectural design, and, finally, generating files with physical IC implementation defined (Gerstlauer et al., 2009). After the synthesis has finished, the IC model reaches a state where all important variables, such as throughput or IC area, are exactly defined. The synthesis on lower levels of the design is the same process, though, the result is a structural IC implementation. Figure 3.3 shows a place the synthesis phase is found during the design.

Synthesis is an iterative process over every system component up to the whole IC. During the design entry process the behavioral IC model, along with the pertaining constraints, is expressed using selected HDL. As was mentioned, the synthesis phase translates hardware code definition into corresponding implementation which is composed by the physical structural model and qualitative variables. At each iteration, design decision information from the previous synthesis iteration is added to the model definition. From this perspective, the model becomes a product of step-by-step IC units synthesis, and potential input for further synthesis steps. Some of the implementation details are netlist which describes components connectivity, pin-accurate models based on combination of netlists and bus-functional component models. Transaction-level modeling is used for construction of abstraction from pins and wires.

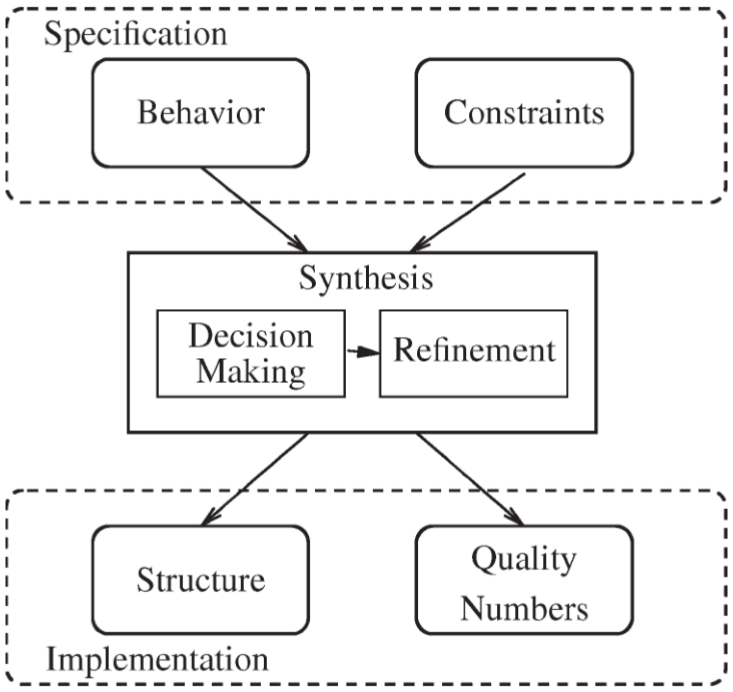

Figure 3.3: IC synthesis process

The quality numbers or architecture characteristic values are achieved making use of performance models. Performance models show overall contributions of separate units to the whole design quality. The variables are usually obtained using static analysis techniques and estimated for best/worse/average cases. According to (Gries, 2004) task-accurate performance models (TAMPs), instruction-set-accurate performance models (ISAPMs), and cycle-accurate performance models (CAPMs) are obtained after the synthesis based on the quality numbers.

As shown in Figure 3.3 a concrete implementation is a product of decision-making and refinement. Decision making refers to the procedure of resource allocation, physical spatial binding of coded behavioral IC units to the previously allocated resources, and calculation of temporal scheduling in case of possible resource contention cases.

Refinement refers to the process of merging the taken decisions into the behavioral model. The result of this process is the IC structural model. The qualitative evaluation of those decisions renders the quality numbers report.

Final step in the synthesis process is the optimization. In order to find optimal or near-optimal IC implementations a design-space exploration (DSE) must be carried out. It is a multiobjective optimization problem after which a set of optimal implementations based in the quality numbers is yielded (Gerstlauer et al., 2009).

Interestingly enough, the Y-chart proposed by (Gajski and Kuhn, 1983) and discussed in Section 2.2 is based on the synthesis process, which was conceived earlier in time, and of which appears to be the refined model.

3.10 Static Timing Analysis

During the static timing analysis the temporal system requirements must by checked. Behavioral perspective does not take into account clock cycles, but the structural one does, and, down to the lower implementation levels, a specific set of gates must meet particular cycle time. Temporal requirements are computed by a timing analyzer. It is a crucial tool for every IC designer.

Timing analyzers verify the requirements using actual gate timings. This process is undoubtedly useful, but often takes a huge amount of time.

Static timing analysis thoroughly computes all timing paths. The program takes inputs derived form the gate-level libraries used by the synthesis and which are estimated statically or derived from the actual floorplanning.

On this phase may appear false path issues related to the reset line that operate over many clock cycles. The analyzer reports that it cannot complete in one cycle. It is fixed manually by marking the line as a multicycle path.

The report accounts every path from one register's output to another register's input. Each path sums corresponding delay and output times. The report presents all paths sorted according to the time so that the slowest entries are the best candidates for improvement.

3.11 Physical Layout Design

Physical layout design starts with a floorplanning. During the floorplanning functional IC units are estimated in terms of occupied area and their associated placement is decided. It is quite important phase since it reveals whether shall an IC fit into the pre-calculated area according to the budget or not. At the same time, it evaluates wiring paths and their lengths with possible congestion issues.

Once the logic is defined, then initial floorplan might be prepared. Changes on the floorplan can affect the logical perspective as relative placement can reduce or increase communication latency.

One of the basic elements of the design on this level is a cell. A standard cell is a library that allows automatic generation of its physical layout. The synthesis tools use these libraries in resource mapping processes.

Area optimizations are often possible and achieved by including wires in cells that snap together. Of course, more design effort is required, but the reward is smaller area and faster operations. The aspect in designing such cells is pitch-matching. It means that connected cells must be equal in size along the edges of connection.

Another important tool used for the layout design is the slice plan. It is a diagram that depicts the ordering of wordslices and how the wiring tracks can be allocated within bitslices. It makes easier to compute wiring lengths and foresee places of possible wiring congestion. On the other hand, it greatly helps for datapath area estimation.

3.12 Post-layout Simulation

Post-layout simulation is a popular verification technique that is used often for final reassurance and designer's confidence. It is a subset of the pre-layout simulations sequence that aims mainly to check the equivalence with pre-layout IC model.

Finally, Design Role Check, Electrical Role Check and Layout vs Schematic checks are also repeated.

After the post-layout verification procedure the chip design is sent to the manufacturer where it passes through a series of the pre-fabrication tests. Fabrication then, takes place with further testing and validation procedures which are out of the scope of the IC design process, and, therefore, this work too.

4 Special Considerations

The IC design process involves many strategies, approaches taken from experience, good practices and other consideration that is more suitable to place in a separate section. This section reveals several important design guidelines worth to be mentioned.

4.1 Regularity

Section 2 pointed out that the hierarchy is the key term in the IC design process, and indeed, it is. It consists of dividing a whole system into a number of subsystems or functional units. But the issue of exceeding complexity is not solved with the hierarchy only. The main principle for coping with it is the regularity.

Employing the regularity, a designer divides the system hierarchy into a number of similar building units. It can be present at all levels of hierarchy: uniformly sized and placed transistors at the circuit level, or gates with identical geometry at the gate level.

It also helps during the verification checks. Since the same components were used fewer checks are required what makes the process faster.

4.2 Modularity

The modularity principle requires that all modules should have well-defined interfaces and functionality. It makes interaction among modules well characterized. In practical terms, it means a module has clearly defined structural, physical, and behavioral interface which exposes the functionality along with the name, timing and electrical constraints of the ports, and signal type.

Generally, it helps a designer document how a particular problem is approached and clarify the steps. On the other hand, software used for the design will check the attributes of a module more easily.

4.3 Locality

Employing modularity means as clear as possible defined interfaces. Internal details of a module are not important for other modules. Hiding this information aids in reducing visible complexity of the design. On HDL coding level it would also mean the reduction of global variables to the minimum (if possible, zero).

The other branch of locality refers to its temporal side which consists in adherence to a timing protocol or clock.

4.4 Design Challenges

The fabrication technology continuously increases in complexity and sometimes dramatically. More complex designs bring new challenges for the engineers. This section describes the most prominent of them.

4.4.1 Design for Testability

Design for testability is a design strategy that aims for increasing quality, security, and reliability before release (Zhang and van Roosmalen, 2010). It is a special challenge for multifunction products since methodology, strategy, and test equipment must be developed. On the other hand, the Automatic Test Equipment systems and tests programming costs increasingly grow as glow the volume, speed, and quality of products. Market pressure is present too, reducing time-to-market periods.

The principal steps in test development are

- Test specification or testbench conception and analysis.
- Test program implementation.
- Test program optimization and debugging.
- Test pattern generation.

4.4.2 Design for Manufacturability

This strategy aims to increase speed of rump-up process increasing the overall yield, reliability, and robustness. It is possible if early and systematic yield loss estimations are incorporated in the design process. Inclusion of the process-aware design flow will also enable to optimize the design flow and reduce expensive iterations.

4.4.3 Design for Reliability

Design for reliability aims to foretell and optimize processes and product reliability. The other name for this strategy is virtual prototyping, and FPGA prototyping discussed in Section 3.8 belongs to it.

Design for reliability extends much more than only behavioral perspective. It can involve activities as materials behavior research, its degradation cycle, and failure conditions. The advanced reliability tests include creation of overloading condition for IC functioning for failure analysis, and development of simulation models for evolution of failure predictions.

5 Conclusions

The present work provided a wide and deep overview of the general IC design flow cycle and gave deeper insight into more important phases as the architecture design and verification procedures.

Given the complexity degree of the IC design, there were presented the ways and detailed key methodologies for tackling with the issue. Architecture design techniques and important perspectives were meticulously described.

Finally, special considerations that were out of the IC design flow context yet very important were briefly discussed.

From my personal point of view, I obtained superfluous knowledge about this process, enough to understand the whole design process, yet not enough for designing by myself one. If it would be possible to apply this knowledge at some point of my career, I will be greatly satisfied.

As a computer engineer and embedded systems programmer, the knowledge of IC design gave me a thorough understanding of systems I am working with.

Bibliography

Global and china integrated circuit industries markets. (2019). https://www.prnewswire.com/news-releases/global-and-china-578-bn-integrated-circuit-industries-markets-2014-2018--2019-2023-300862541.html(31.05.2020)

Brinkman, W. F., Haggan, D. E., & Troutman, W. W. (1997). A history of the invention of the transistor and where it will lead us. *IEEE Journal of Solid-State Circuits*, *32*(12), 1858–1865.

Kilby, J. S. (1976). Invention of the integrated circuit. *IEEE Transactions on electron devices*, *23*(7), 648–654.

Moore, G. E. Et al. (1965). Cramming more components onto integrated circuits. McGraw-Hill New York, NY, USA:

Weste, N. H., & Harris, D. (2015). *Cmos vlsi design: A circuits and systems perspective*. Pearson Education India.

Bedrij, O. J. (1962). Carry-select adder. *IRE Transactions on Electronic Computers*, (3), 340–346.

Fang, C.-J., Huang, C.-H., Wang, J.-S., & Yeh, C.-W. (2002). Fast and compact dynamic ripple carry adder design, In *Proceedings. ieee asia-pacific conference on asic,* IEEE.

Carry-lookahead adder. (2015). https://www.electronicshub.org/carry-look-ahead-adder/(24.04.2020)

Gajski, D. D., & Kuhn, R. H. (1983). New vlsi tools. *Computer*, (12), 11–14.

Yonehara, F. P. Digital ic design methodology. http://lscad.facom.ufms.br/wiki/images/2/27/Asic-flow-methodology.pdf

Guideline for characterization of integrated circuits (AEC - Q003 Rev-A). (2013). Automotive Electronics Council. Component Technical Committee.

Performance specification integrated circuits (microcircuits) manufacturing, general specification for department of defence usa (MIL-PRF-38535J). (2012). Department of Defence USA.

Kaeslin, H. (2008). *Digital integrated circuit design: From vlsi architectures to cmos fabrication*. Cambridge University Press.

Stc12c5a60s2 series mcustc12le5a60s2 series mcu data sheet. (2011). ST CMCU Limited. https://www.buydisplay.com/download/ic/STC12LE5A60S2-ENG.pdf

Matas, B., & DeSubercausau, C. (1997). *Memory, 1997: Complete coverage of dram, sram, eprom, and flash memory ic's*. Integrated Circuit Engineering Corp.

White, S. A. (1989). Applications of distributed arithmetic to digital signal processing: A tutorial review. *IEEE Assp Magazine*, *6*(3), 4–19.

Baker, C. M., & Terman, C. J. (1980). *Tools for verifying integrated circuit designs*. Department of Electrical Engineering; Computer Science, Massachusetts …

Mead, C., & Conway, L. (1980). Introduction to vlsi systems. Addison-Wesley Reading, MA.

A brief verilog history. (1996). https://verilog.com/(29.05.2020)

Pellerin, D. (1995). Introduction to vhdl–for synthesis and simulation. *Accolade Design Automation Inc.*

Gerstlauer, A., Haubelt, C., Pimentel, A. D., Stefanov, T. P., Gajski, D. D., & Teich, J. (2009). Electronic system-level synthesis methodologies. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, *28*(10), 1517–1530.

Gries, M. (2004). Methods for evaluating and covering the design space during early design development. *Integration*, *38*(2), 131–183.

Zhang, G. Q., & van Roosmalen, A. (2010). *More than moore: Creating high value micro/nanoelectronics systems*. Springer Science & Business Media.