Lecture 2 Integrated Circuit Design Flow

Outline

Outline

2. IC Design Flow

- Integrated Circuit Design Flow
  - Front-End Phase
    - Focuses on the design and verification process in the hardware description stages.
  - Back-End Phase
    - Concentrates on the physical implementation of the IC
    - IC fabrication, testing, and packaging are briefly introduced in this section.

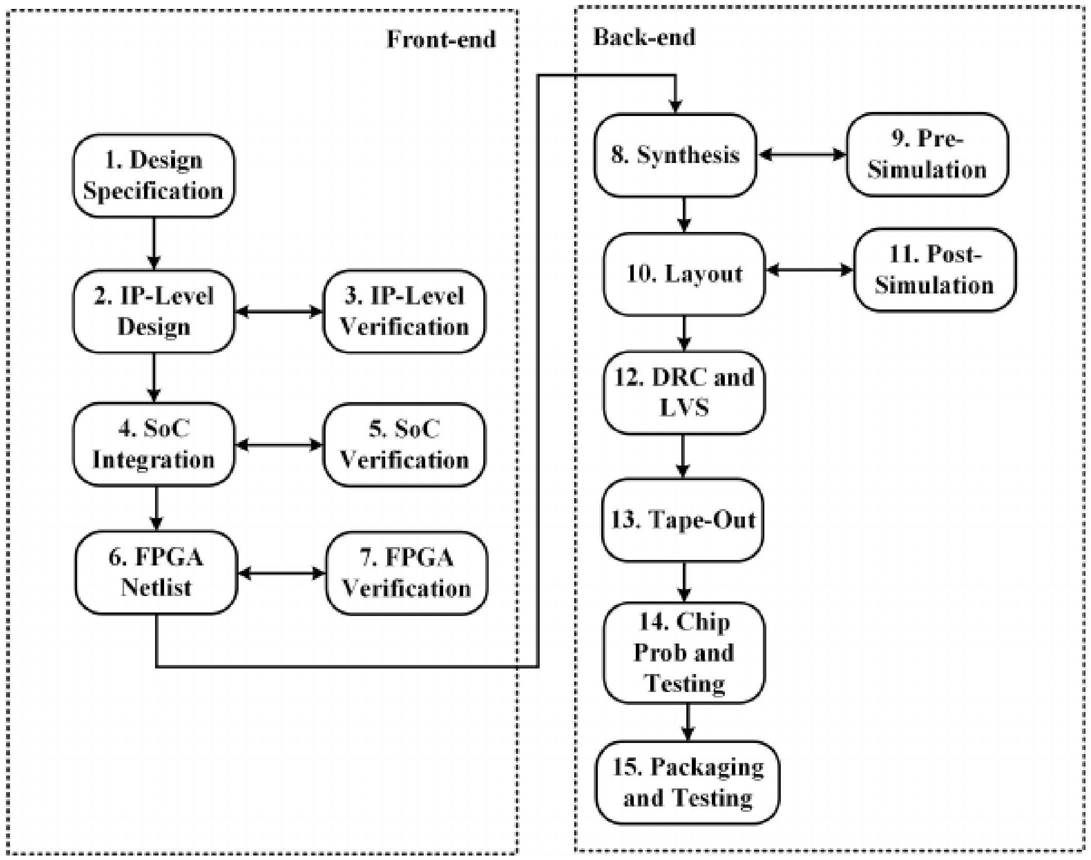
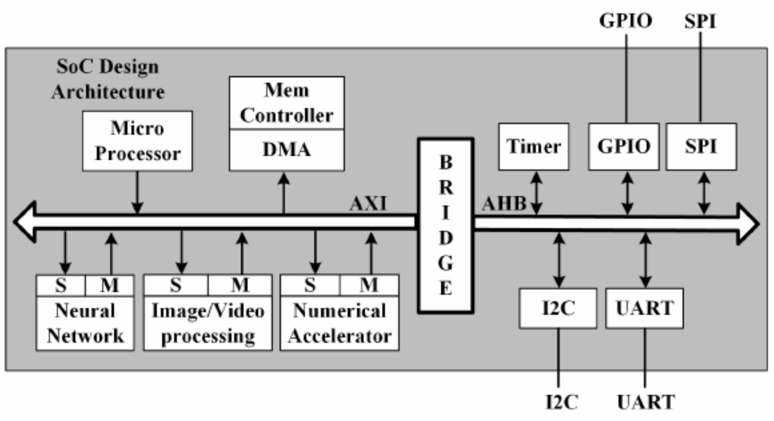


FIGURE 2.1
IC Design Flow.

2.1 Design Specification

- SoC (System-on-Chip) Architecture Specification
  - SoC topology and bus protocol
    - Enable seamless IP block integration: AMBA APB/AHB/AXI (ARM), Wishbone (Silicore), OCP (OCP-IP), etc.
    - Low power design, high-speed/throughput needs, integration of third-party IPs, chip size constraints, and more
- IP (Intellectual Property) Design Specification
  - Third-party IPs from IP vendors
    - IP vendors such as Synopsys, Cadence, ARM, MIPS, etc.
    - HDL IPs, gate-level netlist, hard macro (Aligns with the SoC design)
  - Self-design IPs
    - Block diagram
    - Timing
    - Bus interfaces
    - Functional registers



Outline

2.2 RTL Design in IP Level

- 2.2.1 IP-Level Design
  - IP Design Categories
    - SoC-related designs
      - Primarily concerned with constructing the SoC architectures
        - » Managing various modules located on the control bus
        - » Moving data between memories and design modules through the data bus
        - » Handling internal/external interfacing.
      - Examples:
        - » Bus bridges/wrappers, master/slave interfaces, CPUs, DMA controllers, memory controllers
        - » Industry-standardized interfaces and peripherals like UART, I2C, SPI, SDIO, GPIO, WatchDog, and Timer.
      - Software coding is not necessary
        - » Rely on industry standards, so software coding with identical functions is not typically required.
    - Algorithm Design Needs for Hardware Description

2.2 RTL Design in IP Level

- 2.2.1 IP Level Design
  - IP Design Categories
    - Algorithm Design Needs for Hardware Description
      - C/C++, Matlab, Scala, etc.
      - Provie a general guidance to hardware design: building blocks, precision, performance, etc.
      - Golden model for verification and evaluation
        - » To demonstrate the quality of the results and functions (such as structural evaluation, precision, error percentage, etc.) for future hardware
        - » To prepare golden results for RTL verification.
    - E.g. grayscale_image=rgb2grayscale(rgb_image).
      - FP/Integer Hardware? Design Structure? Memory?
    - Optimized E.g. Grayscale = 0.299R + 0.587G + 0.114B
      - Precision (double, single, etc.)?
      - Design Performance?

2.2 RTL Design in IP Level

- 2.2.2 Hardware Design Considerations
  - A. Algorithmic Design: Accuracy vs. Hardware Cost
    - Optimizations encompass techniques like:
      - Floating-point (FP) to fixed-point conversion
      - Approximate and/or imprecise design methods to conserve hardware resources
      - Striking a balance between quality and hardware to achieve the desired algorithmic accuracy.
    - FP matrix-vector multiplication:
      - Higher design accuracy typically leads to increased hardware costs.
      - Integer, FP8, FP16, FP32, FP64, FP128, etc.

$$\begin{pmatrix} z_0 \\ z_1 \end{pmatrix} = \begin{pmatrix} x_0 & x_1 \\ x_2 & x_3 \end{pmatrix} \cdot \begin{pmatrix} y_0 \\ y_1 \end{pmatrix}$$

2.2 RTL Design in IP Level

- 2.2.2 An IP Design Example: Matrix-Vector Multiplier
  - A. Algorithmic Design: Accuracy vs. Hardware Cost
  - B. Hardware Structural Design: Speed vs. Area-Power
    - Parallel computing, pipeline structures, design reuse, clock-gating, and many more
    - (a) A straightforward parallel implementation utilizing four FP multipliers and two FP adders
    - (b) Sequential structure saves chip area and power dissipation at the expense of reduced speed.

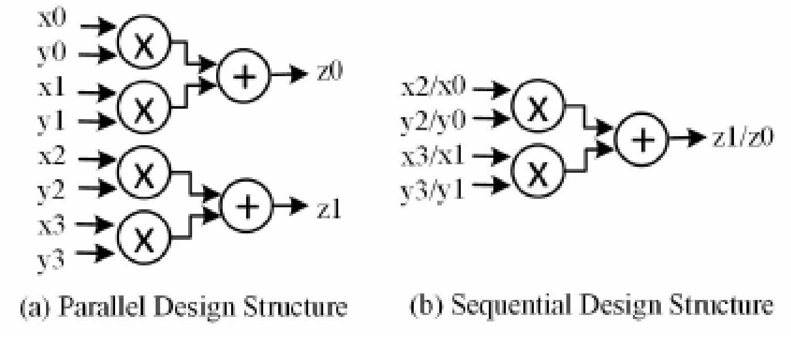

x0
y0
x1
y1
x2
y2
x3
y3
z0
z1
(a) Parallel Design Structure
x2/x0
y2/y0
x3/x1
y3/y1
z1/z0
(b) Sequential Design Structure

FIGURE 2.2
Parallel Design and Sequential Design Structures for Matrix-Vector Multiplication.

2.2 RTL Design in IP Level

- 2.2.3 Code Editors
  - VIM, Emacs, and WinEdit
    - VIM is recommended for several reasons.
      - Open-source text editor that has been available since 1976.
      - Fast and lightweight performance, making it suitable for editing large files or source code.
      - It is primarily used in the Unix/Linux environment but can also be used on Mac OS and Windows OS.
      - Supports a wide range of programming languages and file formats.
      - Fourthly, VIM is highly customizable using the ``.vimrc'' source file, allowing users to tailor their editor experience.

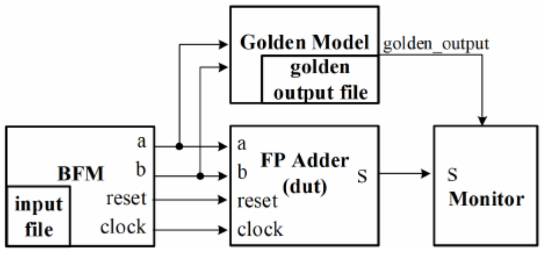
        ```
        set guifont=Consolas:h11
        ```

      - Its extensive collection of powerful commands that can greatly enhance programming efficiency
    - Tutorial: Appendix A

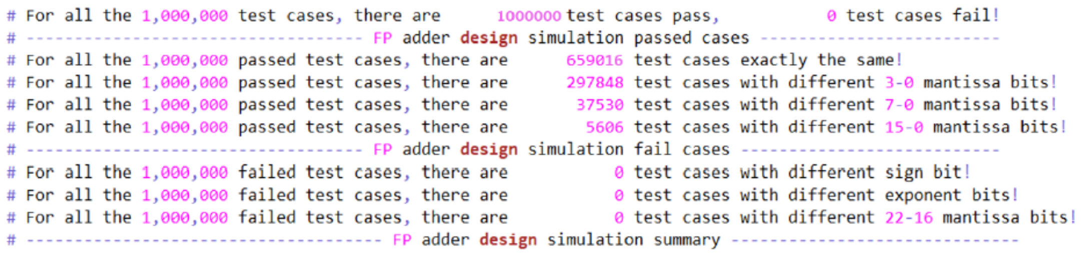
Outline

2.3 Functional Verification

- 2.3.1 Simulation Testbench with Verilog HDL
  - Automation
    - Functional verification involves iterative rounds of debugging, code modifications, and the execution of numerous test cases to enhance functional and code coverage.
    - Therefore, automation plays a vital role in simplifying this iterative simulation process.
    - Fig. 2.3: A basic testbench for a FP adder

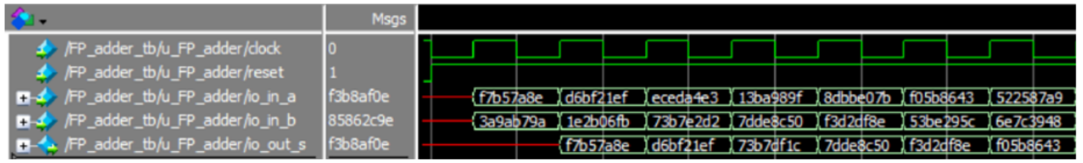


(a) General Simulation Testbench



```
# For all the 1,000,000 test cases, there are      1000000 test cases pass,          0 test cases fail!
# ---------------------------------- FP adder design simulation passed cases ---------------------------
# For all the 1,000,000 passed test cases, there are      659016 test cases exactly the same!
# For all the 1,000,000 passed test cases, there are      297848 test cases with different 3-0 mantissa bits!
# For all the 1,000,000 passed test cases, there are       37530 test cases with different 7-0 mantissa bits!
# For all the 1,000,000 passed test cases, there are        5606 test cases with different 15-0 mantissa bits!
# ---------------------------------- FP adder design simulation fail cases -----------------------------
# For all the 1,000,000 failed test cases, there are           0 test cases with different sign bit!
# For all the 1,000,000 failed test cases, there are           0 test cases with different exponent bits!
# For all the 1,000,000 failed test cases, there are           0 test cases with different 22-16 mantissa bits!
# ------------------------------------ FP adder design simulation summary ------------------------------
```

(b) Simulation Report/Log


Msgs
/FP_adder_tb/u_FP_adder/clock 0
/FP_adder_tb/u_FP_adder/reset 1
/FP_adder_tb/u_FP_adder/io_in_a f3b8af0e
/FP_adder_tb/u_FP_adder/io_in_b 85862c9e
/FP_adder_tb/u_FP_adder/io_out_s f3b8af0e
f7b57a8e d6bf21ef eceda4e3 13ba989f 8dbbe07b f05b8643 522587a9
3a9ab79a 1e2b06fb 73b7e2d2 7dde8c50 f3d2df8e 53be295c 6e7c3948
f7b57a8e d6bf21ef 73b7df1c 7dde8c50 f3d2df8e f05b8643

(c) ModelSim Simulation Waveform

FIGURE 2.3
Simulation Example of FP Adder

- 2.3.2 Verification Methods and Methodologies
  - A combination of software languages may be utilized alongside the Verilog HDL testbench
    - such as C/C++, Vera, and Matlab
  - Industry employs various verification methods and methodologies to ensure the quality and reliability of design ICs.
    - SystemVerilog
      - Extends the capabilities of Verilog HDL and includes features specifically developed for verification
      - *Assertions, coverage models, and constrained-random* stimulus generation.
    - UVM
      - Provides a set of methodologies, libraries, and guidelines that facilitate the creation of scalable and reusable verification components.

2.3 Functional Verification

- 2.3.2 Verification Methods and Methodologies
  - A. Coverage-Driven Verification
    - Functional coverage
      - Aims to verify that all features and functionalities described in the verification plan have been exercised by the RTL design.
    - Code coverage metrics
      - Offer quantitative measurements of how much of the RTL code has been exercised during the simulation.
      - Line coverage, statement coverage, branch coverage, and condition coverage.
  - B. Constrained-Random Test
  - C. Assertion-Based Verification
  - D. UVM

2.3 Functional Verification

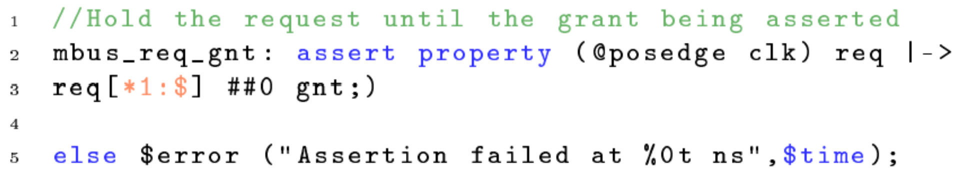
- 2.3.2 Verification Methods and Methodologies
  - A. Coverage-Driven Verification
  - B. Constrained-Random Test
    - Test Cases:
      - Direct tests: verify specific features and functionalities outlined in the design specification.
        - » Executing all direct tests ensures that the functional coverage reaches 100%, effectively covering all desired design behaviors.
      - Random tests: focus on exploring different combinations of features, particularly targeting abnormal or corner cases that cannot be directly tested.
      - Constrained-Random Verification (CRV): allows stimuli to be constrained and guided to effectively target the verification plan, functional coverage goals, and code coverage metrics.
  - C. Assertion-Based Verification
  - D. UVM

2.3 Functional Verification

- 2.3.2 Verification Methods and Methodologies
  - A. Coverage-Driven Verification
  - B. Constrained-Random Test
    - *RTL designers* typically handle the *direct tests*, while *verification engineers* take responsibility for conducting *random tests* that target abnormal cases and achieve line coverage goals.
    - *Direct test cases* can often drive more than *70%* of line coverage, and *constrained-random tests* can further increase line coverage to about *90%.*
    - The *remaining 5% beyond 90%* can be achieved by manually analyzing the coverage report and creating specific *corner cases* to target the remaining coverage gaps.
    - When a design bug is identified and fixed, a regression test must be performed. The regression test includes a collection of direct and random tests to ensure that the code changes do not impact other design features, functions, and timing.
  - C. Assertion-Based Verification
  - D. UVM

2.3 Functional Verification

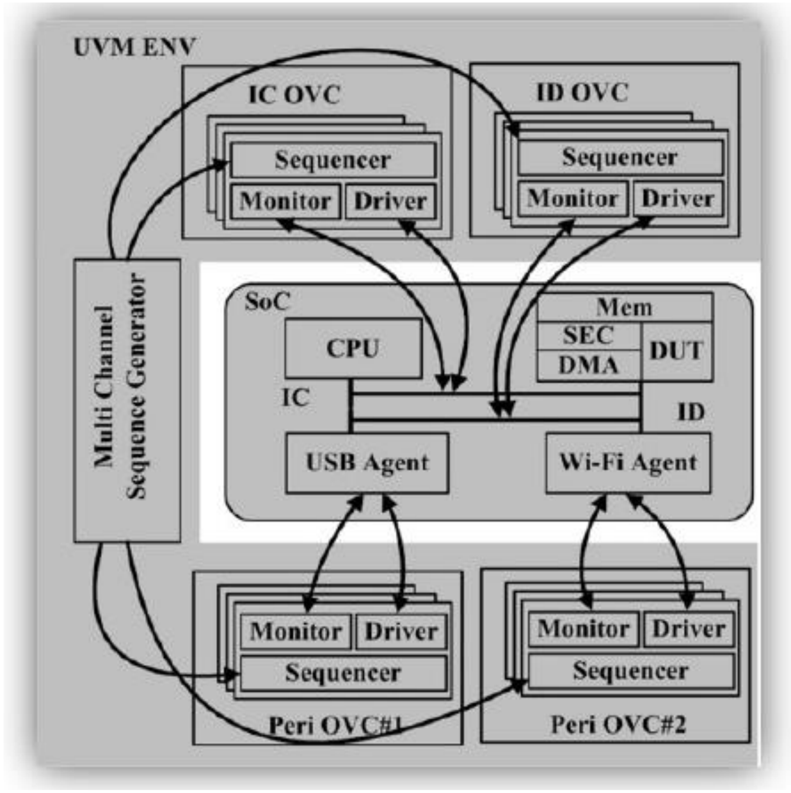
- 2.3.2 Verification Methods and Methodologies
  - A. Coverage-Driven Verification
  - B. Constrained-Random Test
  - C. Assertion-Based Verification
    - Primarily used for validating system design against specifications during simulation
    - Can be incorporated directly into the RTL code for validation, or integrated into the testbench as part of the broader verification process.
    - Example: Request-Grant Handshaking
      - In the event of a violation of this requirement, the simulator will pinpoint an error, detailing the specific simulation time where the discrepancy occurred.
  - D. UVM



```
//Hold the request until the grant being asserted
mbus_req_gnt: assert property (@posedge clk) req |->
req[*1:$] ##0 gnt;)

else $error ("Assertion failed at %0t ns",$time);
```

- 2.3.2 Verification Methods and Methodologies
  - A. Coverage-Driven Verification
  - B. Constrained-Random Test
  - C. Assertion-Based Verification
  - D. UVM
    - An extensively adopted de facto standard in IC design verification.
    - Reusable testbench!
      - UVM enables the creation of reusable testbench components by incorporating object-oriented programming (OOP).
      - Based on the SystemVerilog library and facilitates CDV, CRT, ABV, thereby facilitating the development of reusable verification components.



2.3 Functional Verification

- 2.3.3 EDA Tools for Simulation
  - Synoposys VCS
  - Cadence NC
  - Siemens EDA ModelSim
    - Formerly Mentor Graphics ModelSim after its acquisition by Siemens in 2017
    - GUI window or a TCL script; .vcd format
    - Tutorial: Appendix B
  - Debussy for waveform:
    - Small size dump waveform (.fsdb format)
    - Easy to trace signals
  - Why dump waveform?
    - Easily shared between verification and design teams for debugging purposes
    - Encompasses transition activities of all inputs, signals, and logic elements, making it valuable not only for functional simulation but also for evaluating dynamic power consumption.

Outline

2.4 SoC Integration

- SoC Architecture
  - AMBA AHB/AXI from ARM Holdings, Wishbone from Silicore Corporation, OCP from OCP IP, CoreConnect from IBM
  - Only IP components with compatible interfaces can be successfully integrated.
  - Example: SoC with AMBA AHB (low speed) and AXI (high-speed)

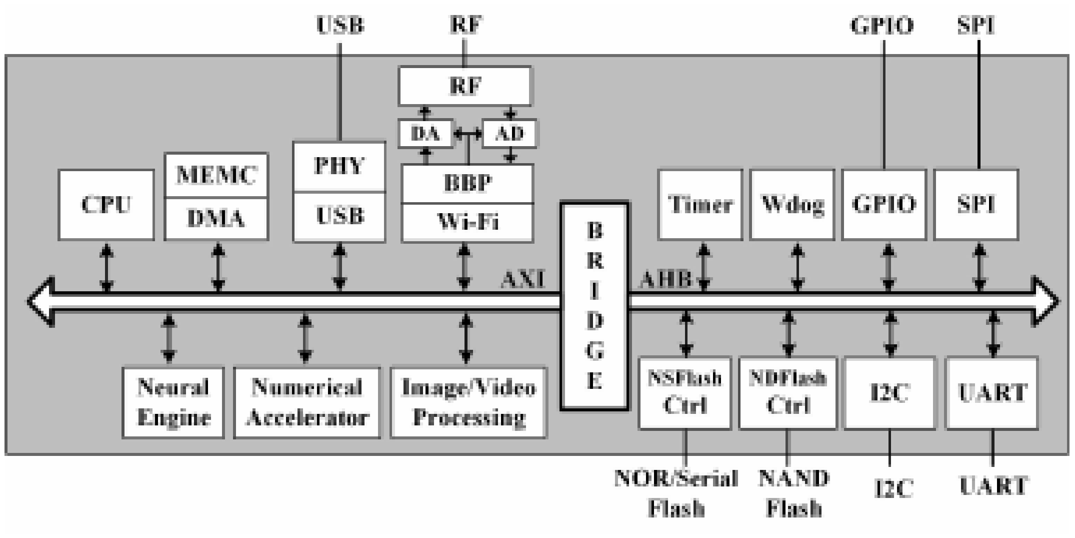


FIGURE 2.4
An Example of SoC Integration.

Outline

2.5 SoC Verification

- SoC Tree vs. IP Tree
  - Functional verification at SoC tree focus on testing the interconnection and integration of the entire system.
  - UVM components established in the IP tree can be reused in the SoC tree.
- SoC Verification Procedure
  - Run simulations by assigning a unique ID seed, observe the log file, and dump the corresponding waveform if any bugs are detected.
    - The unique ID seed allows for the reproduction of random stimuli, enabling the reexecution of simulations with the same test case after resolving design bugs.
  - Collaborate with designers to debug the waveform dump.
    - If the bug is originating from the testbench, the verification engineer should address and rectify it.
    - If the bug is stemming from the design code, the verification engineer should report it to the designers.

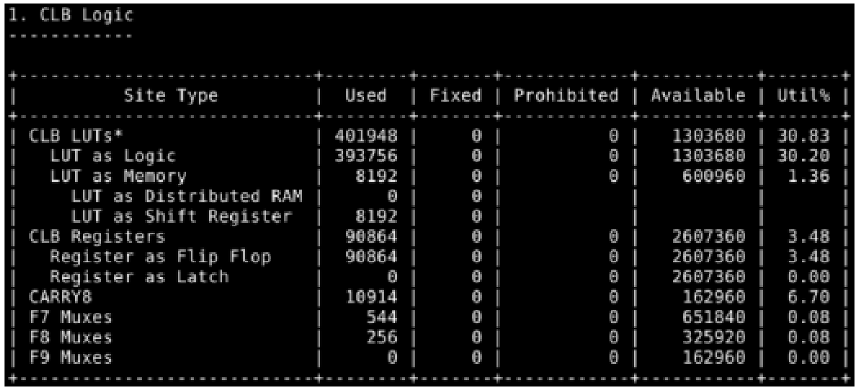
2.5 SoC Verification

- IP Tree and SoC Tree
- SoC Verification Procedure
  - Designers should resolve the design bugs and rerun the simulation with the unique ID seed.
    - After passing the specific test case, regression testing is necessary in both IP design and SoC integration to ensure that the bug fix does not impact other design components.
  - After resolving the bug, update the code with a new version using version control software.
    - Git, SVN (Subversion), or VSS (Visual SourceSafe).
  - Collect functional coverage and code coverage during system-level verification
    - The SoC coverage metrics will be lower than those achieved during IP-level verification.
    - IP design and verification primarily focus on design functions and features, whereas SoC verification is primarily concerned with system connectivity and integration.

Outline

2.6 FPGA Netlist

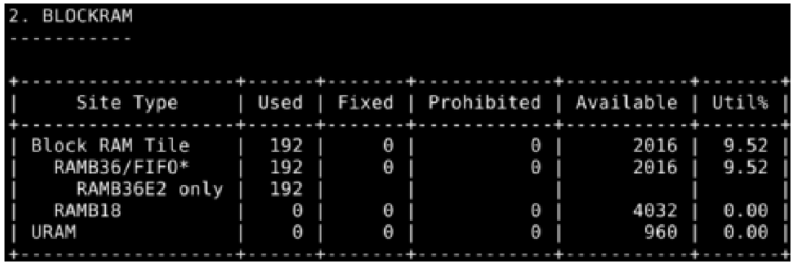
- 2.6.1 FPGA Development and Assessment
  - Synthesis: converts the RTL design into a netlist that can be mapped using available FPGA resources.
  - Implementation: involves placing and routing the netlist into a bitstream understood by the physical FPGA.
  - Synthesis and implementation report



```
1. CLB Logic
------------

+----------------------------+--------+-------+------------+-----------+-------+
|         Site Type          |  Used  | Fixed | Prohibited | Available | Util% |
+----------------------------+--------+-------+------------+-----------+-------+
| CLB LUTs*                  | 401948 |     0 |          0 |   1303680 | 30.83 |
|   LUT as Logic             | 393756 |     0 |          0 |   1303680 | 30.20 |
|   LUT as Memory            |   8192 |     0 |          0 |    600960 |  1.36 |
|     LUT as Distributed RAM |      0 |     0 |            |           |       |
|     LUT as Shift Register  |   8192 |     0 |            |           |       |
| CLB Registers              |  90864 |     0 |          0 |   2607360 |  3.48 |
|   Register as Flip Flop    |  90864 |     0 |          0 |   2607360 |  3.48 |
|   Register as Latch        |      0 |     0 |          0 |   2607360 |  0.00 |
| CARRY8                     |  10914 |     0 |          0 |    162960 |  6.70 |
| F7 Muxes                   |    544 |     0 |          0 |    651840 |  0.08 |
| F8 Muxes                   |    256 |     0 |          0 |    325920 |  0.08 |
| F9 Muxes                   |      0 |     0 |          0 |    162960 |  0.00 |
+----------------------------+--------+-------+------------+-----------+-------+
```

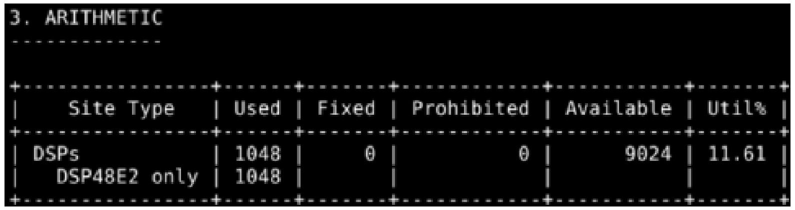
(a) Resource Utilization on LUT/Register Slices



```
2. BLOCKRAM
-----------

+-------------------+------+-------+------------+-----------+-------+
|     Site Type     | Used | Fixed | Prohibited | Available | Util% |
+-------------------+------+-------+------------+-----------+-------+
| Block RAM Tile    |  192 |     0 |          0 |      2016 |  9.52 |
|   RAMB36/FIFO*    |  192 |     0 |          0 |      2016 |  9.52 |
|     RAMB36E2 only |  192 |       |            |           |       |
|   RAMB18          |    0 |     0 |          0 |      4032 |  0.00 |
| URAM              |    0 |     0 |          0 |       960 |  0.00 |
+-------------------+------+-------+------------+-----------+-------+
```

(b) Resource Utilization on RAMs



```
3. ARITHMETIC
-------------

+----------------+------+-------+------------+-----------+-------+
|   Site Type    | Used | Fixed | Prohibited | Available | Util% |
+----------------+------+-------+------------+-----------+-------+
| DSPs           | 1048 |     0 |          0 |      9024 | 11.61 |
|   DSP48E2 only | 1048 |       |            |           |       |
+----------------+------+-------+------------+-----------+-------+
```

(c) Resource Utilization on DSPs

FIGURE 2.5
FPGA Synthesis and Implementation Report

- 2.6.2 Maximum Operational Frequency (MOF) and Timing Constraints
  - MOF can be calculated using the formula below, where T_R denotes the reference clock period and T_{WNS} represents the amount of slack in each clock cycle.

$$MOF = \frac{1}{T_R - T_{WNS}} \tag{2.2}$$

  - Assuming the design specification calls for a reference clock frequency of 250 MHz, which corresponds to a minimum clock period of 4 ns.

```
create_clock -add -name clk -period 4.0 -waveform {0 2}
[get_ports {clk}];
```

  - After creating the clock object, the timing report, including the value of T_{WNS}, can be generated by running the FPGA synthesis process with the desired clock period.
    - For instance, T_{WNS}=0.683 ns, the design circuit can achieve an MOF = 1/(4-0.683) = 301.5 MHz, meeting the design goals.

2.6 FPGA Netlist

- 2.6.3 EDA Tools for FPGA Demonstration
  - The FPGA tools: AMD Vivado and Intel Quartus.
  - FPGA tools serve multiple purposes, including
    - Verifying the equivalence between the design block diagram and RTL analysis results
    - Estimating the resource utilization and MOF of FPGA designs
    - Assisting users in establishing the link between Verilog descriptions and real hardware components.
  - Tutorial: Appendix C

Outline

2.7 FPGA Verification

- 2.7.1 Importance of FPGA Verification
  - Ensures the design 1) operates as intended, and 2) adheres to the physical constraints.
    - RTL verification
      - Focuses on verifying the logical behavior of the design <u>in an idealized environment</u>
    - FPGA verification
      - Examine the design's performance and physical <u>implementation on real hardware</u>
  - A practical method for establishing and validating the functionality of the hardware design within its intended execution environment.
    - E.g. Wireless communication channels emulation
      - RTL verification using SystemVerilog or other languages can be challenging.
      - FPGA technology allows the design bitstream to be run in an actual wireless setting.

2.7 FPGA Verification

- 2.7.2 Challenges of FPGA Verification
  - Challenges:
    - Limited verification methods
    - Analyzing simulation waveforms can be more intricate
      - The use of additional equipment such as logic analyzers and oscilloscopes becomes necessary.
    - May involve timing issues that may not consistently replicate
      - A design may pass testing correctly in some instances but fail in others due to corner cases and timing violations.
  - Solutions:
    - Ensure thorough verification of RTL code before progressing to FPGA implementation.
    - Extensive utilization of random testing and code coverage to detect functional bugs at the RTL itself, prior to the physical design and implementation on the FPGA.

Outline

2.8 ASIC Synthesis

- 2.8.1 ASIC Synthesis and Script File
  - Refers to the process in which the *RTL description is translated into a gate-level netlist*, forming the basis for chip layout.
  - Design Synthesis Example: 4-bit counter

```
//A 4-bit Counter Design with Verilog HDL
module counter_4b (input              rst,
                   input              clk,
                   input        [3:0] ini,
                   output reg   [3:0] cnt);
always @ (negedge rst, posedge clk) begin
  if (~rst) begin
    cnt <= ini        ;
  end else begin
    cnt <= cnt + 4'h1;
  end
end
endmodule
```

2.8 ASIC Synthesis

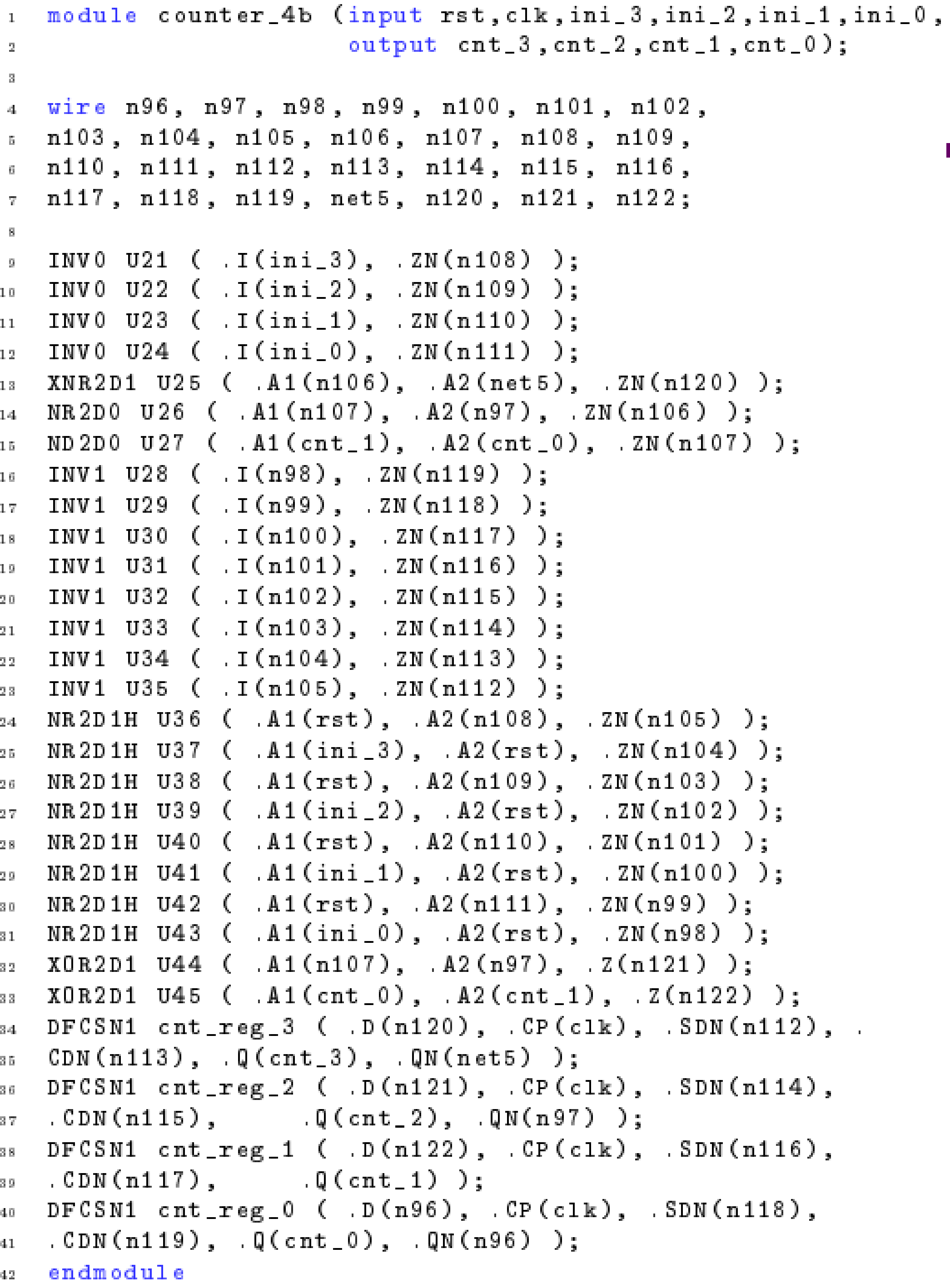
- 2.8.1 ASIC Synthesis and Script File
  - Script file to automate this process
    - Lines 1-4: Specify the directory path and as series libraries
    - Lines 6-7: The Verilog design and the current project
    - Lines 8-10: ``don't touch'' to prevent any optimization or alteration during synthesis, because the clock tree and reset network will not be established until the layout process.
    - Lines 11-12, write the synthesized design into a database file ``counter_4b.db'', and write out the gate-level description of the design in Verilog format to the ``counter_4b.v'' file.

```
1  search_path = {"." "./synthesis" }
2  link_library= {"*" "./synthesis/tcb773stc.db" }
3  target_library= {tcb773stc.db}
4  symbol_library= {tcb773s.sdb}
5
6  read -format verilog {"./counter_4b.v"}
7  current_design = "counter_4b"
8  create_clock -name "clk" -period 100 -waveform {"0" "50"}
9  set_dont_touch_network find( clock, "clk")
10 set_dont_touch_network find( port, "rst")
11 write -format db -hierarchy -output "./counter_4b.db"
12 write -format verilog -hierarchy -output "./counter_4b.v"
13 ......
```

2.8 ASIC Synthesis

2.8.2 Gate-Level Netlist and Synthesis Report

- By executing the script, the synthesis tool, such as Synopsys Design Compiler, can autonomously translate the RTL code into a gate-level netlist.
- For example, the gate-level description provided below outlines the cells, registers, and wires encompassing the entire design.



```
module counter_4b (input rst,clk,ini_3,ini_2,ini_1,ini_0,
                   output cnt_3,cnt_2,cnt_1,cnt_0);

wire n96, n97, n98, n99, n100, n101, n102,
n103, n104, n105, n106, n107, n108, n109,
n110, n111, n112, n113, n114, n115, n116,
n117, n118, n119, net5, n120, n121, n122;

INV0 U21 ( .I(ini_3), .ZN(n108) );
INV0 U22 ( .I(ini_2), .ZN(n109) );
INV0 U23 ( .I(ini_1), .ZN(n110) );
INV0 U24 ( .I(ini_0), .ZN(n111) );
XNR2D1 U25 ( .A1(n106), .A2(net5), .ZN(n120) );
NR2D0 U26 ( .A1(n107), .A2(n97), .ZN(n106) );
ND2D0 U27 ( .A1(cnt_1), .A2(cnt_0), .ZN(n107) );
INV1 U28 ( .I(n98), .ZN(n119) );
INV1 U29 ( .I(n99), .ZN(n118) );
INV1 U30 ( .I(n100), .ZN(n117) );
INV1 U31 ( .I(n101), .ZN(n116) );
INV1 U32 ( .I(n102), .ZN(n115) );
INV1 U33 ( .I(n103), .ZN(n114) );
INV1 U34 ( .I(n104), .ZN(n113) );
INV1 U35 ( .I(n105), .ZN(n112) );
NR2D1H U36 ( .A1(rst), .A2(n108), .ZN(n105) );
NR2D1H U37 ( .A1(ini_3), .A2(rst), .ZN(n104) );
NR2D1H U38 ( .A1(rst), .A2(n109), .ZN(n103) );
NR2D1H U39 ( .A1(ini_2), .A2(rst), .ZN(n102) );
NR2D1H U40 ( .A1(rst), .A2(n110), .ZN(n101) );
NR2D1H U41 ( .A1(ini_1), .A2(rst), .ZN(n100) );
NR2D1H U42 ( .A1(rst), .A2(n111), .ZN(n99) );
NR2D1H U43 ( .A1(ini_0), .A2(rst), .ZN(n98) );
XOR2D1 U44 ( .A1(n107), .A2(n97), .Z(n121) );
XOR2D1 U45 ( .A1(cnt_0), .A2(cnt_1), .Z(n122) );
DFCSN1 cnt_reg_3 ( .D(n120), .CP(clk), .SDN(n112), .
CDN(n113), .Q(cnt_3), .QN(net5) );
DFCSN1 cnt_reg_2 ( .D(n121), .CP(clk), .SDN(n114),
.CDN(n115),     .Q(cnt_2), .QN(n97) );
DFCSN1 cnt_reg_1 ( .D(n122), .CP(clk), .SDN(n116),
.CDN(n117),     .Q(cnt_1) );
DFCSN1 cnt_reg_0 ( .D(n96), .CP(clk), .SDN(n118),
.CDN(n119), .Q(cnt_0), .QN(n96) );
endmodule
```

2.8 ASIC Synthesis

- 2.8.2 Gate-Level Netlist and Synthesis Report
  - The schematic view of the 4-bit counter design
    - This illustrates the numerous logic gates deployed for combinational addition functions and the four flip-flops serving as sequential memory elements.
  - ASIC synthesis report
    - Summarizes the resource usage in terms of the number of ports and cells, along with the total cell area.

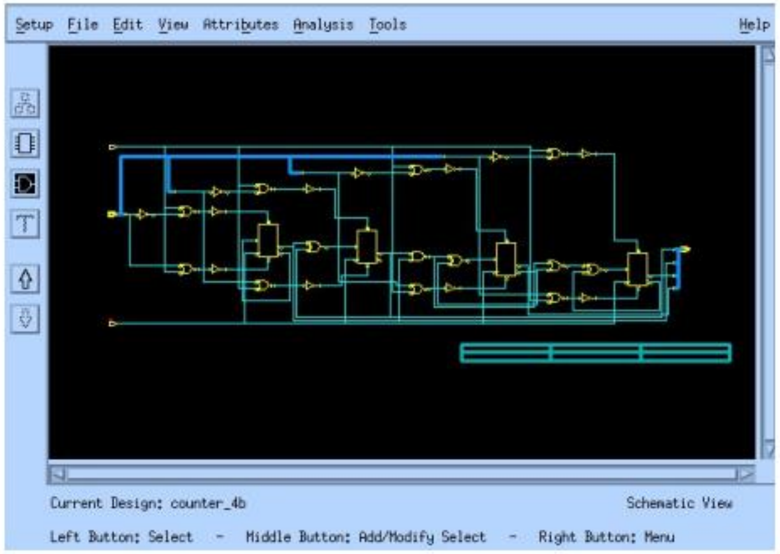

Setup File Edit View Attributes Analysis Tools Help
Current Design: counter_4b
Schematic View
Left Button: Select - Middle Button: Add/Modify Select - Right Button: Menu

FIGURE 2.6
The Schematic View of ASIC Synthesis.

```
//Synthesis Report
Library(s) Used:
tcb773stc (File: ./synthesis/tcb773stc.db)
Number of ports: 10
Number of nets: 36
Number of cells: 25
......
```

2.8 ASIC Synthesis

- 2.8.3 EDA Tools for ASIC Synthesis
  - Synopsys Design Compiler is the predominant synthesis tool employed for ASIC synthesis in the industry
  - OpenROAD represents an open-source toolchain that aspires to encompass all facets of chip design.
    - OpenROAD is part of the larger effort towards enabling open-source IC design
    - Publicly available to use and modify under their respective open-source licenses.
  - Concurrent process between FPGA verification and ASIC synthesis
    - In practice, ASIC synthesis and FPGA verification are often carried out simultaneously to overlap the design cycle.

Outline

2.9 Pre-Simulation and STA

- 2.9.1 Pre-Simulation vs. Static Timing Analysis
  - RTL Code vs. Gate-Level Netlist
    - RTL design primarily focuses on behavioral descriptions, without including timing delays associated with logic cells.
    - The gate-level netlist incorporates the timing information, including IO ports, logic cells, registers, and wire connections.
  - Pre-Layout Simulation (Pre-Simulation)
    - Goal: verify the functional equivalence between the RTL design and the gate-level netlist
    - Test cases: simple direct tests due to its slower nature compared to RTL design simulation.
  - Static Timing Analysis

2.9 Pre-Simulation and STA

- 2.9.1 Pre-Simulation and STA
  - RTL Code vs. Gate-Level Netlist
  - Pre-Layout Simulation (Pre-Simulation)
  - Static Timing Analysis
    - Critical timing checks of the synthesized netlist.
    - STA vs. Pre-Simulation
      - STA is much faster because it is not necessary to simulate the logical operation of the circuit.
      - STA is more thorough because it checks all timing paths, not just the logical conditions that are sensitized by a set of test vectors.
      - STA can only check the timing, not the functionality which can be done by the pre-simulation, of a circuit design.

2.9 Pre-Simulation

- 2.9.2 Pre-Simulation Example
  - A. Creating and Annotating SDF File for Pre-Simulation
    - Standard Delay Format (SDF) for Pre-Simulation
      - SDF is a file containing cell and register delays when executing the ASIC synthesis, which can be generated using the command *write_sdf*.

```
//Create SDF file during Design Compiler
write_sdf counter_4b.sdf
```

    - Annotation
      - The SDF file needs to be included in the testbench

```
// SDF Annotation in testbench
$sdf_annotate("./counter_4b.sdf",tb)
```

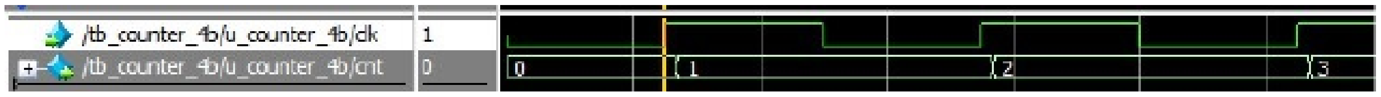
      - Once the annotation is complete, the netlist can be subjected to pre-simulation on simulators.

2.9 Pre-Simulation

- 2.9.2 Pre-Simulation Example
  - A. Creating and Annotating SDF File for Pre-Simulation
    - Considerations to build the verification environment
      - Third-Party IPs
        - » Simulation models can be utilized to replace the black boxes, allowing their inclusion in the simulation.
      - Clock and Reset
        - » The timing of the clock and reset signals can be disregarded during pre-simulation since the clock tree will be utilized in the layout phase, handling the timing aspects.
        - » In cases where the fanouts of the clock and reset paths are excessively high, it may be necessary to insert an empty module to replace these paths.
          - This is because a single path may be unable to drive a large number of fanouts, necessitating the use of additional buffering or dividing the fanout to ensure proper functionality.

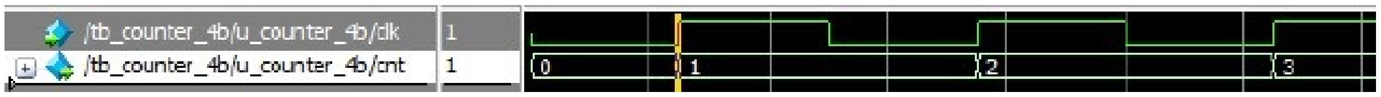
9. Pre-Simulation

- 2.9.2 Pre-Simulation Example
  - B. Netlist Pre-Simulation Waveform vs. RTL Simulation Waveform
    - (a) It is evident from the waveform that the output signal ``cnt'' from the register undergoes a bit delay following the rising clock edge.
    - (b) In contrast, the RTL simulation waveform does not exhibit such a delay.
    - This distinction arises because the RTL simulation stage lacks the delay information of clock-to-register output.



/tb_counter_4b/u_counter_4b/clk
1
/tb_counter_4b/u_counter_4b/cnt
0
0
1
2
3

(a) Netlist Pre-Simulation with 4-bit Counter

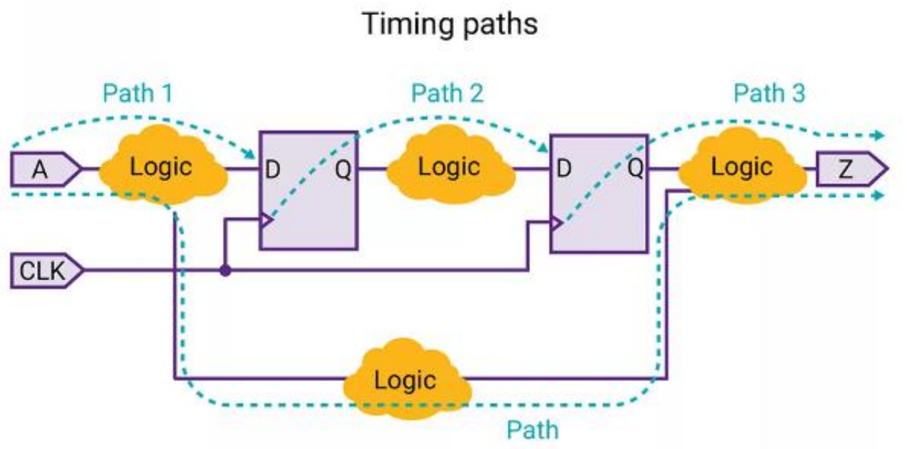


/tb_counter_4b/u_counter_4b/clk
1
/tb_counter_4b/u_counter_4b/cnt
1
0
1
2
3

(b) RTL Simulation with 4-bit Counter

FIGURE 2.7
Netlist Pre-Simulation vs. RTL Simulation

- 2.9.3 Static Timing Analysis and Timing Failures
  - A. Timing Check Failures and ECO
    - Setup Time Violations
      - The presence of a long data path where the signals pass through numerous logic elements between registers.
    - Hold Time Violations
      - These occur when the timing delay is insufficient for the signal to remain stable after the active clock edge.



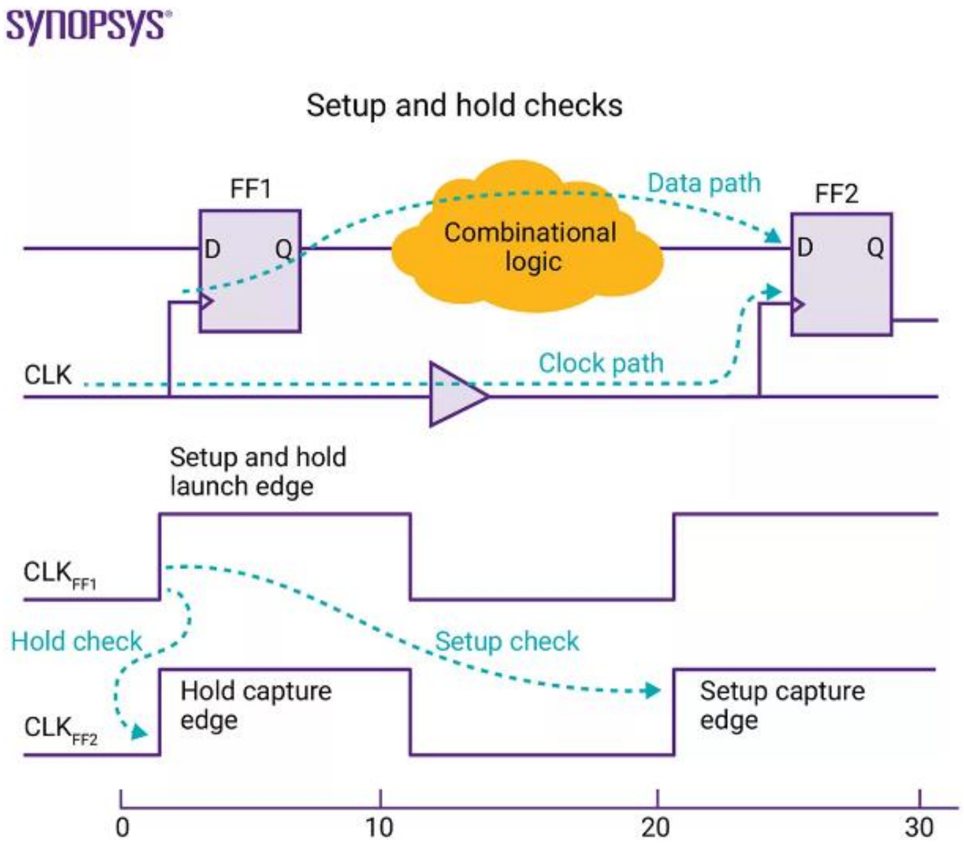
In the example, each logic cloud represents a combinational logic network. Each path starts at a data launch point, passes through some combinational logic, and ends at a data capture point.

Path	Startpoint	Endpoint
Path 1	Input port	Data input of a sequential element
Path 2	Clock pin of a sequential element	Data input of a sequential element
Path 3	Clock pin of a sequential element	Output port
Path 4	Input port	Output port

Ref: What is STA? Synopsys. https://www.synopsys.com/glossary/what-is-static-timing-analysis.html#a

9. Pre-simulation

- 2.9.3 Timing Check and STA
  - A. Timing Check Failures and ECO
    - ECO (Engineering Change Order)
      - Timing check failures can necessitate a return to RTL coding to rectify timing violations.
      - ECO means that designers can sometimes make modifications directly to the gate-level netlist to rectify design and timing issues, instead of making corrections back to the RTL code.



Ref: What is STA? Synopsys. https://www.synopsys.com/glossary/what-is-static-timing-analysis.html#a

- 2.9.3 Timing Check and STA
  - B. Static Timing Analysis and EDA Tools
    - Static timing analysis (STA)
      - Checking whether the netlist operates within specified timing constraints, such as setup and hold times, at the desired clock frequency
    - EDA Tools
      - Synopsys ***Primetime***
        - » Reading synthesized netlist along with timing constraint files;
        - » Building timing model with delay information of each gate, register, and net in the design;
        - » Analyzing setup time and hold time;
        - » Providing detailed reports indicating timing violations, critical paths, slack values (how much time a path can be delayed without causing a violation), and overall design performance.

Outline

2.10 Layout

- 2.10.1 What is Layout?
  - Involves converting the gate-level netlist into a comprehensive physical representation using automated placement and routing (APR) techniques.
  - Layout Process:
    - Placement: All VCC and GND paths, design blocks, IPs, IO pads around the chip layout, clock and reset trees are strategically positioned within the chip area.
      - In the synthesis process
        - » IPs within the netlist are indeed treated as black boxes or simulation models.
        - » The clock and reset trees are not explicitly represented in the netlist.
      - In the Layout phase
        - » Insert all the IPs into suitable locations on the chip
        - » Incorporating the clock and reset trees into the chip layout.

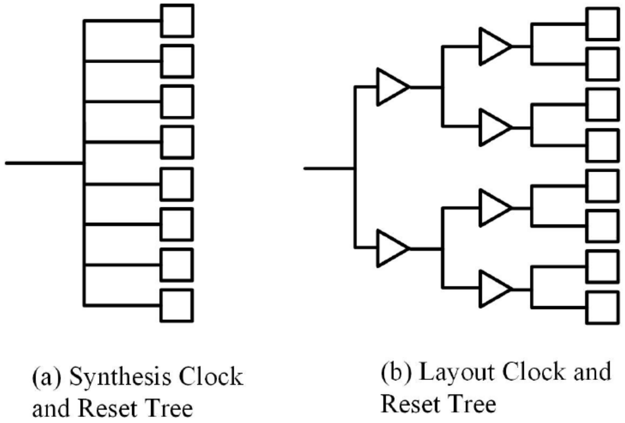

(a) Synthesis Clock and Reset Tree
(b) Layout Clock and Reset Tree

FIGURE 2.8
Clock and Reset Tree

2.10 Layout

- 2.10.1 What is Layout?
  - Layout Process:
    - Placement:
    - Routing: Establishes interconnections between the placed components, creating a complete chip-level network.
      - Involves determining the most efficient paths for signals to traverse while adhering to timing, power, and area constraints.
    - GDSII File Generation: The output of the placement and routing process is a file in the Graphic Data System II (GDSII) format.
      - This file contains a detailed description of the chip layout, including the positions and interconnections of all the components.
      - The GDSII file serves as the final representation of the chip's physical design.

2.10 Layout

- 2.10.2 Pad-Limited and Core-Limited Chip Size
  - Core-Limited sizing
    - The chip size is primarily determined by the gate count or the overall complexity of the core circuitry.
  - Pad-Limited sizing
    - The chip size is primarily influenced by the number of IO connections or the amount of peripheral circuitry.

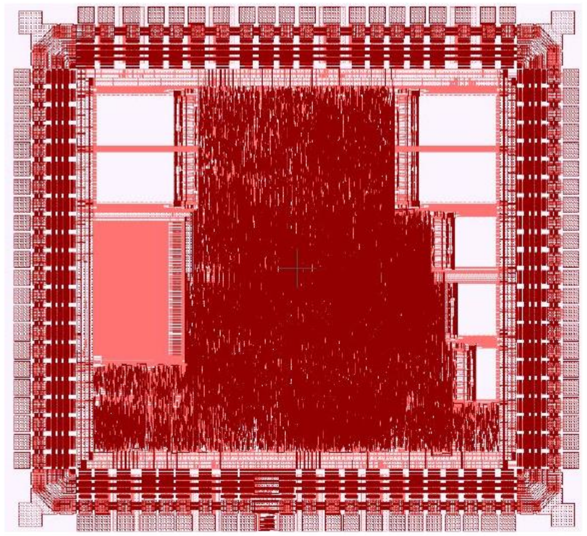


FIGURE 2.9
Layout Schematic

Outline

2.11 Post-Simulation and Post-STA

- Post-Simulation and Post-STA
  - Post-Simulation
    - Incorporates the wire delay, as well as the delays introduced by placed and routed cells and registers.
  - Post-Layout STA, or Post-STA
    - Not only the logical gate delays, but also the physical attributes of the layout, including wire lengths, parasitic capacitances, and resistances.
- 2.11.1 Post-Simulation
  - A. Creating and Annotating SPF File for Post-Simulation
    - Calculate the wire delay accurately, the parasitic capacitance and resistance resulting from the layout.
    - This information is typically provided in a file called the Standard Parasitic File (SPF).
  - B. Post-Simulation Timing

2.11 Post-simulation

- 2.11.1 Post-Simulation
  - A. Creating and Annotating SPF File for Post-Simulation
  - B. Post-Simulation Timing
    - Clock skew can be detected by analyzing the waveform.
      - Clock skew refers to the variation in arrival times of the clock signal at different components within the system design.
      - It is possible to encounter post-simulation failure due to timing violations caused by clock skew.
- 2.11.2 Post-STA
  - Post-STA
    - Conducted at the layout netlist level, where the physical arrangement of gates, wires, and other components on the chip is considered.
  - STA vs. Post-STA
    - STA is typically executed on the synthesized netlist for the initial timing assessment
    - Post-STA is carried out on the layout netlist to validate the timing once the design is physically implemented.

Outline

2.12 DRC and LVS

- Design Rule Check (DRC)
  - Ensure that the layout adheres to the specified design rules, which are necessary for flawless fabrication.
    - These rules include criteria related to manufacturing-related constraints such as spacing and metal layer usage.
  - EDA tools: Synopsys IC Validator and Siemens Calibre
- Layout Versus Schematic (LVS)
  - Ensure that the physical implementation of the design matches the intended functionality captured in the schematic.
    - It verifies that the same circuit topology, devices, connections, and electrical properties are present in both representations.
  - EDA tools: Synopsys IC Validator and Siemens Calibre

2.12 DRC and LVS

- Design Rule Check (DRC)
- Layout Versus Schematic (LVS)
  - Layout GDS Files
    - Play a guiding role in the lithography process
      - Ensuring that intricate patterns are precisely transferred onto the wafer, thereby crafting the tangible components of the circuit.
  - Schematic
    - Emerges from a higher-level schematic design.
      - This netlist functions as a benchmark against which the layout netlist is compared.
      - While schematics have an established connection with analog circuit design, it's noteworthy that they can also be pertinent to specific facets of digital design and/or mixed-signal circuits.

2.12 DRC and LVS

- Design Rule Check
  - LVS verification flow:
    - A netlist is extracted from the GDS layout file using a layout extraction step.
    - The IC Validator NetTran utility is employed to generate an ICV netlist, which serves as the basis for comparison between the layout and schematic.
    - The extracted netlist is then compared with the schematic netlist, verifying various aspects such as *the number of devices in the schematic versus the layout, the number of nets in the schematic versus the layout, and the types of devices in both the schematic and layout.*
    - Finally, result reports are generated, providing information on any discrepancies or inconsistencies identified during the LVS check.

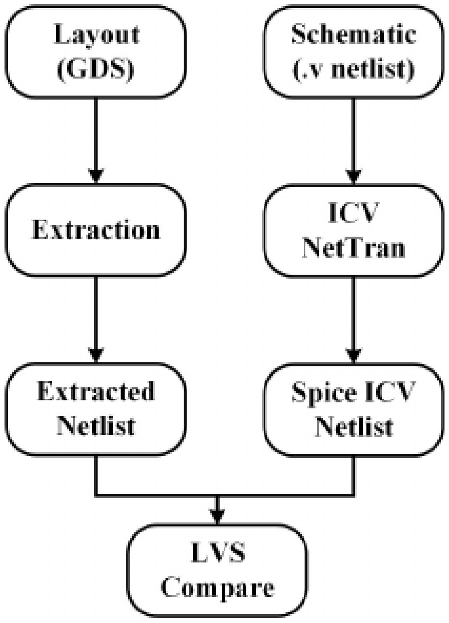


FIGURE 2.10
LVS Flow.

Outline

2.13 Tape-Out

- Tape-Out or Tapeout:
  - Originated in the 1970s when the design was recorded onto a magnetic tape and sent to the foundries for manufacturing.
- 2.13.1 Chip Tape-out Procedure
  - Performed in cleanroom environments and require precision and expertise.
  - Briefly includes:
    - Mask Creation: Masks are created based on the layout design. Each layer of the design has its own corresponding mask.

      Photolithography: The wafer is aligned with a mask using a photo aligner. Ultraviolet light is then passed through the mask, exposing the wafer to create patterns. This step is known as photolithography.
      - The term combines ``photo'' (light), ``litho'' (stone), and ``graphy'' (print), representing the process of ``printing with light''. In short, photolithography transfers geometric patterns from the mask to a photosensitive chemical photoresist on the substrate.

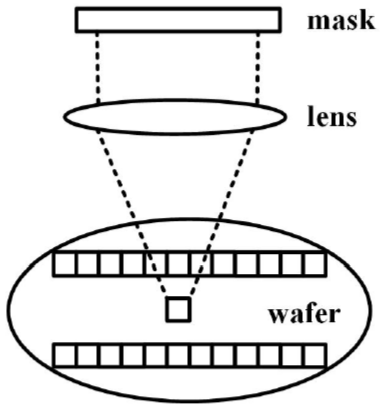
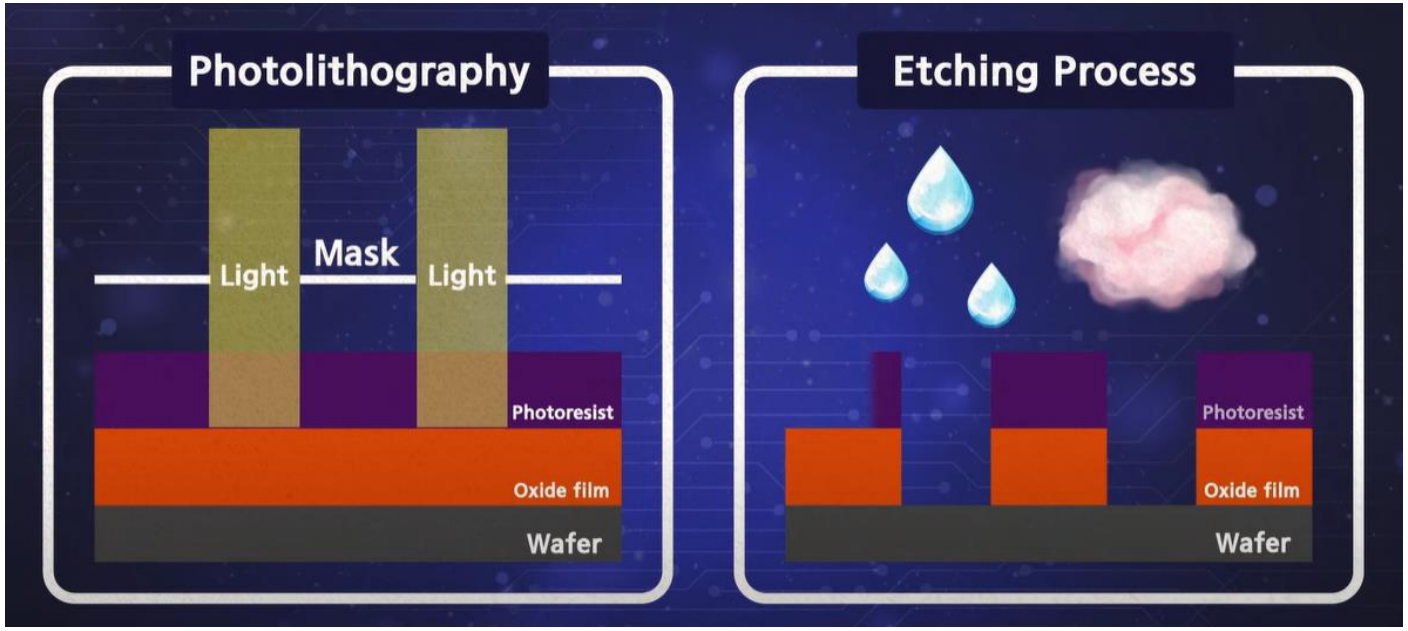


FIGURE 2.11
Photolithography

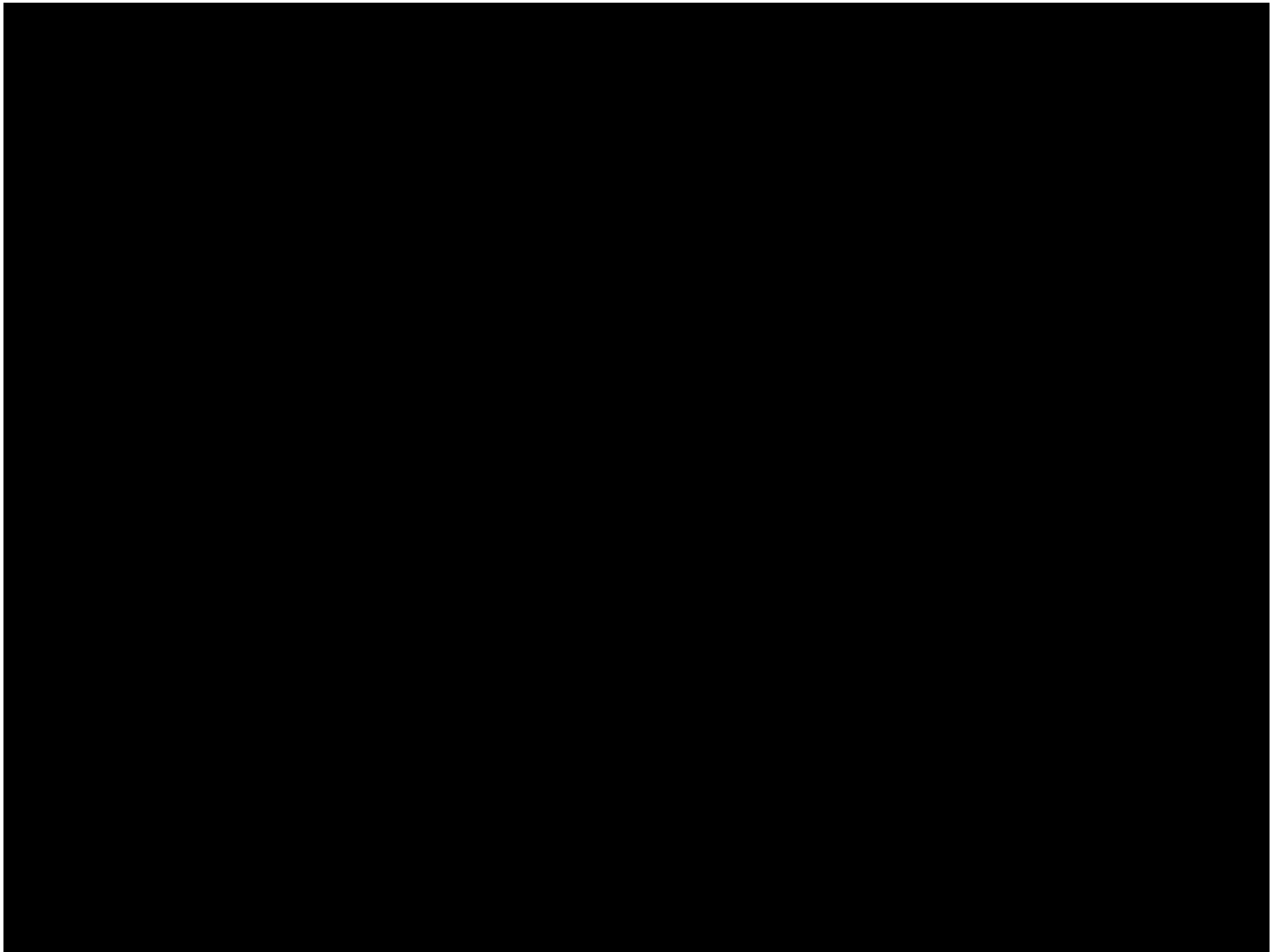
- 2.13.1 Chip Tape-out Procedure
  - Briefly includes:
    - Mask Creation:
    - Photolithography:
    - Etching: removes material from the wafer's surface that is not covered by the photoresist, creating the desired patterns.



2.13 Tape-Out

- 2.13.1 Chip Tape-out Procedure
  - Briefly includes:
    - Mask Creation:
    - Photolithography:
    - Etching:
    - Doping: involves altering the electrical characteristics of the silicon by introducing specific materials.
      - For example, doping can involve introducing atoms with one less electron than silicon (creating P-type regions) or atoms with one more electron than silicon (creating N-type regions). This process allows for the creation of P-type and N-type regions in the silicon.
    - Repetition: The above steps are repeated for each layer, sequentially building up the complete circuit structure.

2.13 Tape-Out

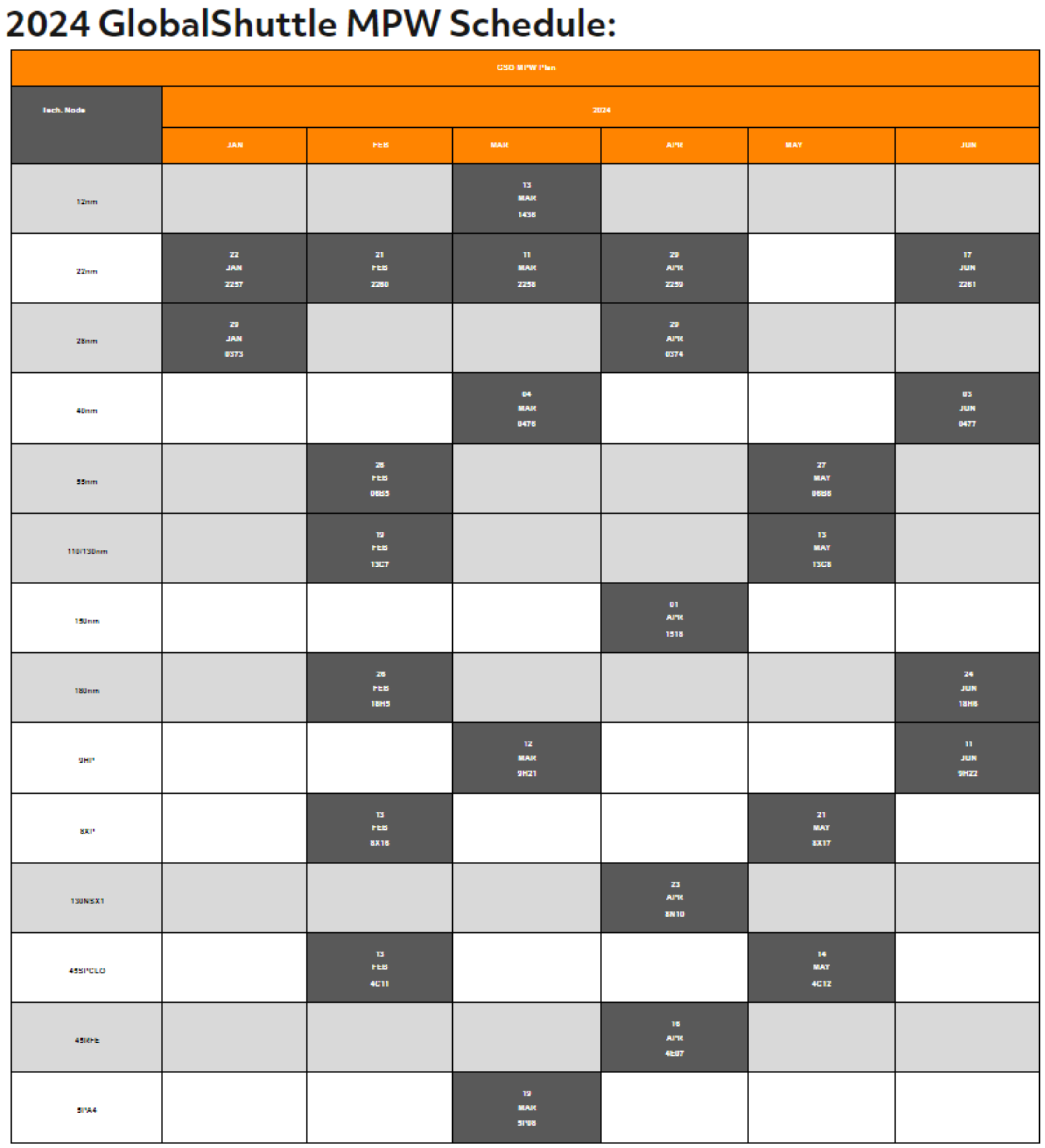


2.13 Tape-out

- 2.13.2 MPW/Shuttle and Full-Set
  - A. Chip Tape-Out Cycle and Cost
    - Time:
      - Non-recurring engineering (NRE): The phase from tape-out to chip readiness.
      - This crucial stage typically extends over several months.
    - Budget:
      - The tape-out cost of employing 16-24 nanometer Global Foundry technology for one square millimeter can range from tens of thousands of US dollars to millions of US dollars.
      - Not only financial implications but also the potential for time-to-market delays in delivering the final products.
      - Rigorous design verification, timing analysis, design rule checks, and manufacturability assessment
  - B. MPW/Shuttle
  - C. Full-Set

2.13 Tape-out

- 2.13.2 MPW/Shuttle and Full-Set
  - A. Chip Tape-out Risk
  - B. MPW/Shuttle
    - Multi-Project Wafer (MPW)/Shuttle Service:
      - Allows different chips or projects to share the same wafers, reducing individual tape-out risks.
      - Facilitates cost reduction by distributing the maskset price among multiple projects,
  - C. Full-Set



2024 GlobalShuttle MPW Schedule:

GSO MPW Plan						
Tech. Node	2024					
	JAN	FEB	MAR	APR	MAY	JUN
12nm			13 MAR 1436			
22nm	22 JAN 2257	21 FEB 2260	11 MAR 2258	29 APR 2259		17 JUN 2261
28nm	29 JAN 0373			29 APR 0374		
40nm			04 MAR 0476			03 JUN 0477
55nm		26 FEB 0885			27 MAY 0886	
110/130nm		19 FEB 13C7			13 MAY 13C8	
150nm				01 APR 1518		
180nm		26 FEB 18H5				24 JUN 18H6
9HP			12 MAR 9H21			11 JUN 9H22
8XP		13 FEB 8X16			21 MAY 8X17	
130NSX1				23 APR 8N10		
45SPCLO		13 FEB 4C11			14 MAY 4C12	
45RFE				16 APR 4E07		
SPA4			19 MAR SP08			

2.13 Tape-out

- 2.13.2 MPW/Shuttle and Full-Set
  - A. Chip Tape-out Risk
  - B. MPW/Shuttle
    - Multi-Project Wafer (MPW)/Shuttle Service:
      - The MPW process is also known as a Shuttle in some industrial groups.
  - C. Full-Set
    - On the other hand, the subsequent tape-out following the MPW/Shuttle run is referred to as a full-set, representing a complete maskset for the specific design.
    - The first tape-out is often done as an MPW/Shuttle run due to its lower cost compared to a formal NRE process.

Outline

2.14 Chip Prob and DFT

- 2.14.1 Chip Prob and Tester
  - Tester
    - Evaluate the functionality of the chip before packaging and mounting.
  - Tester performs various testing steps:
    - Powering on the Chip
    - Resetting the Chip and Sending Input
    - Observing and Comparing Output
    - Identifying Mismatches
  - Fig. 2.12
    - Insertion cards with multiple channels
    - Probe card:
      - Movable to be positioned and connected to different chips for testing purposes.
      - The pins on the probe card make contact with the pads on the chip, enabling the transmission and reception of input and output patterns.

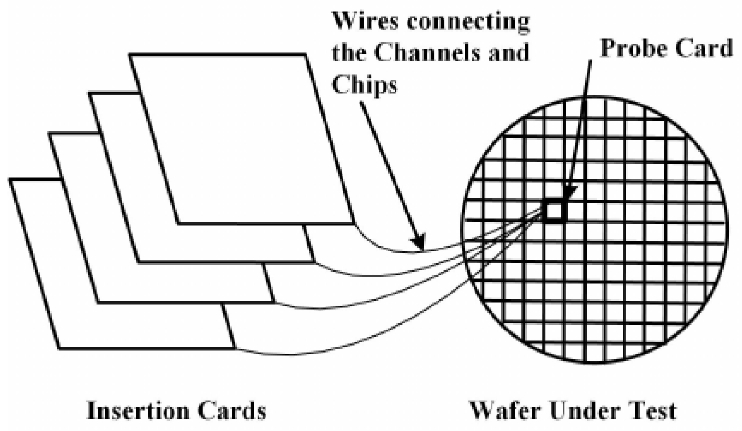


FIGURE 2.12
Chip Probe and Tester.

- 2.14.2 DFT and ATPG
  - A. Color-Coding for Testing ICs
    - Testing results are documented by the tester, associating them with a unique serial number on the wafer.
    - To distinguish chips with different test results, they are marked with different colors.
  - B. Testing Coverage and DFT
    - The toggle rate of nets, transitioning from binary zeros to ones and from ones to zeros, should reach around 70-90%.
    - Design for Test (DFT) and Auto Test Pattern Generation (ATPG)
      - To aid in generating test patterns, additional circuits can be incorporated into the chip. These circuits often include scan chains and scan flip-flops.
      - The process of generating test patterns is referred to as Auto Test Pattern Generation (ATPG), and the knowledge and techniques involved in test pattern generation are collectively known as Design for Test (DFT).

Outline

2.15. Packaging and Testing

- Packaging
  - Cut wafer along the scribe lines.
  - Defective chips will be discarded, while the functional ones will proceed to the packaging stage.
  - Only a small number of pads around the chip are led out as pins
    - Some pads reserved specifically for testing purposes.
- Final Testing

2.15. Packaging and Testing

- Packaging
- Final Testing
  - Many of the test patterns from the chip probing stage can be reused.
  - The same tester is used, but the probe card is replaced with a socket on the handler. The IC is placed onto the socket and tested.
  - Since not all the testing pads are accessible, some test patterns may not be utilized during this stage.
  - After the IC is mounted in the actual system, it may undergo further testing in its real-world environment.

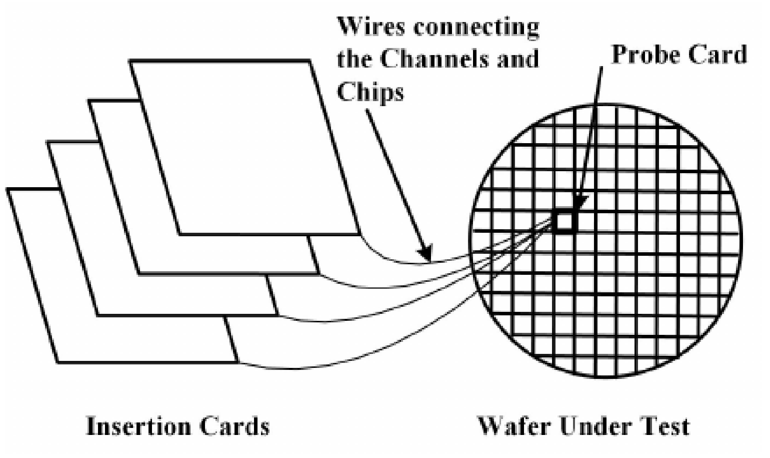


FIGURE 2.12
Chip Probe and Tester.

Outline

2.16 An Example to the IC Design Flow

- 2.16.1 Front-End Phase
  - A. Front-End Phases and Timeline
  - B. Front-End Design Team
    - The functional verification phase is conducted throughout the entire design phase and requires a team size of approximately 3-5 times the number of designers.
  - C. FPGA Verification

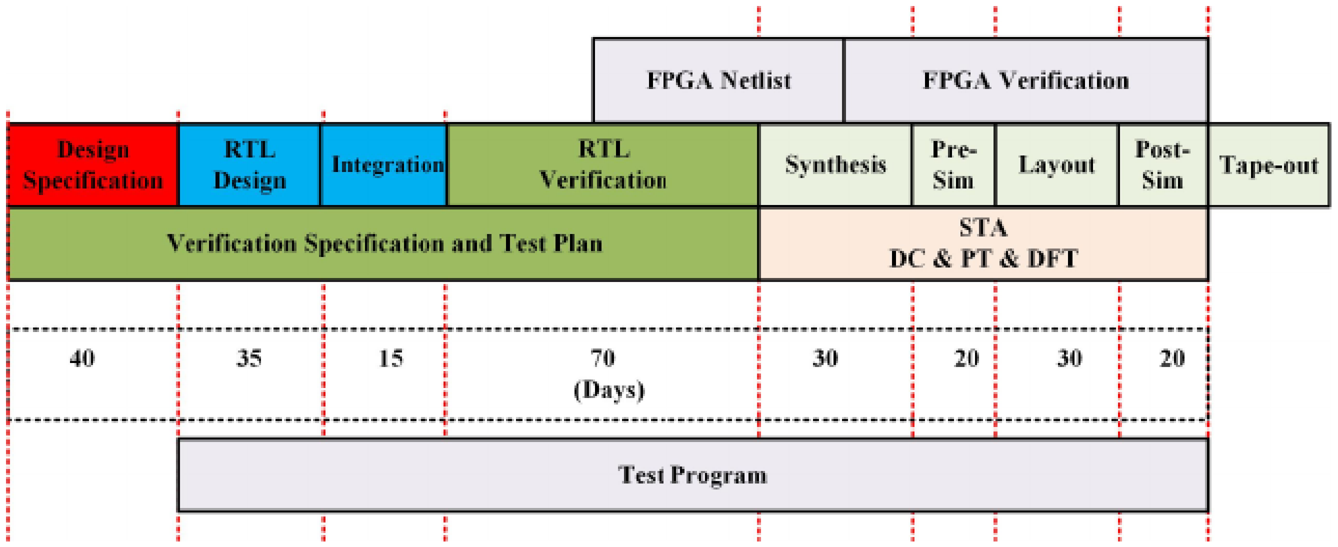


FIGURE 2.13
An Example of ASIC Design Cycle

2.16 An Example to the IC Design Flow

- 2.16.1 Front-End Phase
  - C. FPGA Verification
    - RTL for ASIC and FPGA:
      - The RTL code used for FPGA implementation should closely resemble the final ASIC synthesis and layout to ensure a reliable verification between the FPGA prototype and ASIC implementation.
    - IP Utilization on FPGA:
      - Due to resource limitations on the FPGA board, certain design modules may need to be adjusted or replaced by IP blocks to accommodate the system design within the FPGA platform.
    - Multiple FPGA Demonstration:
      - Large-scale projects may require the use of multiple FPGA platforms to establish the complete system.
    - FPGA Verification is not necessary for some projects/cases
      - FPGA implementation and verification in the design flow depends on project-specific requirements and constraints.
      - The utilization is based on projects and the level of confidence in the RTL verification results such as functional and code coverage.

2.16 An Example to the IC Design Flow

- 2.16.2 Back-End Phase
  - A. Back-End Phases and Timeline
  - B. Back-End Design Team

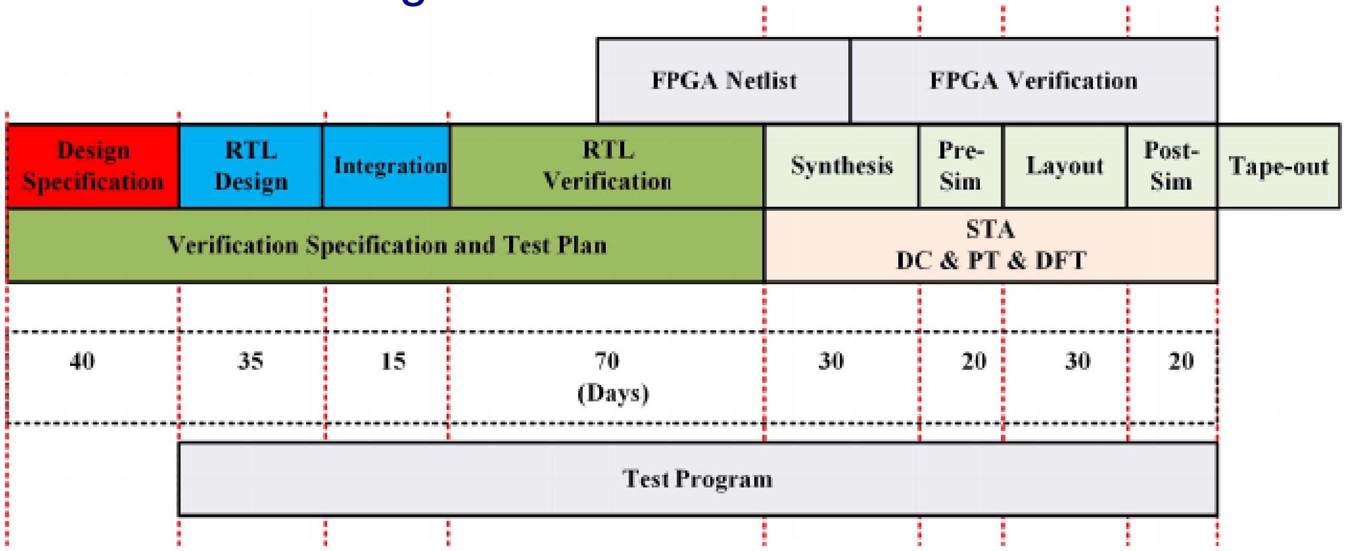


FIGURE 2.13
An Example of ASIC Design Cycle

  - Once the back-end phase concludes, the IC design flow is considered complete, and the focus shifts to tape-out and subsequent chip manufacturing processes.

Outline

2.17 Basic Qualifications Needed For IC Design

- key qualifications
  - Sought by the industry for entry-level engineer positions in the front-end IC design
    - RTL Design and Simulation
      - Tutorials on the Vim editor, TCL scripting for RTL simulation, and fundamental Linux/Unix commands are provided in Appendix A.
    - Functional Verification (Simulation) and FPGA Verification
    - Industry Design Rules and Design Standards

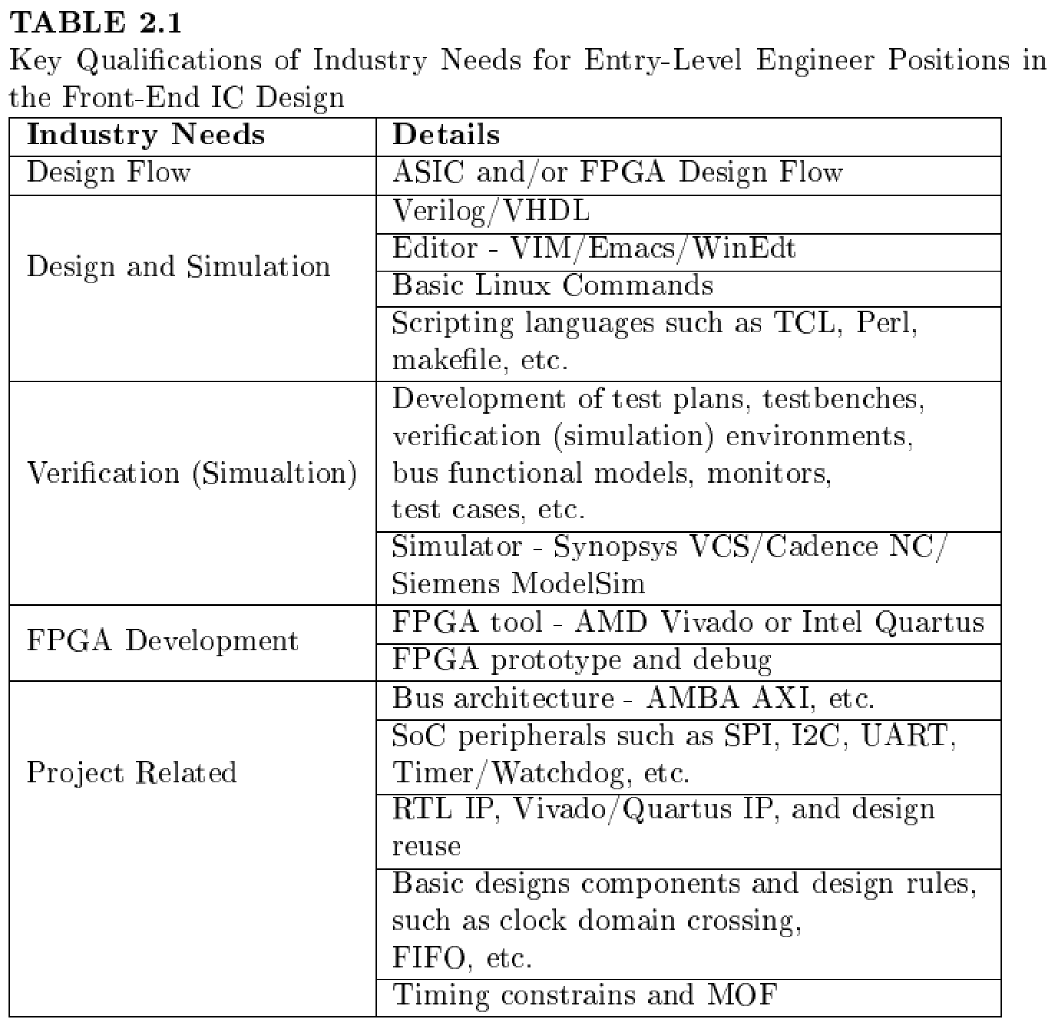


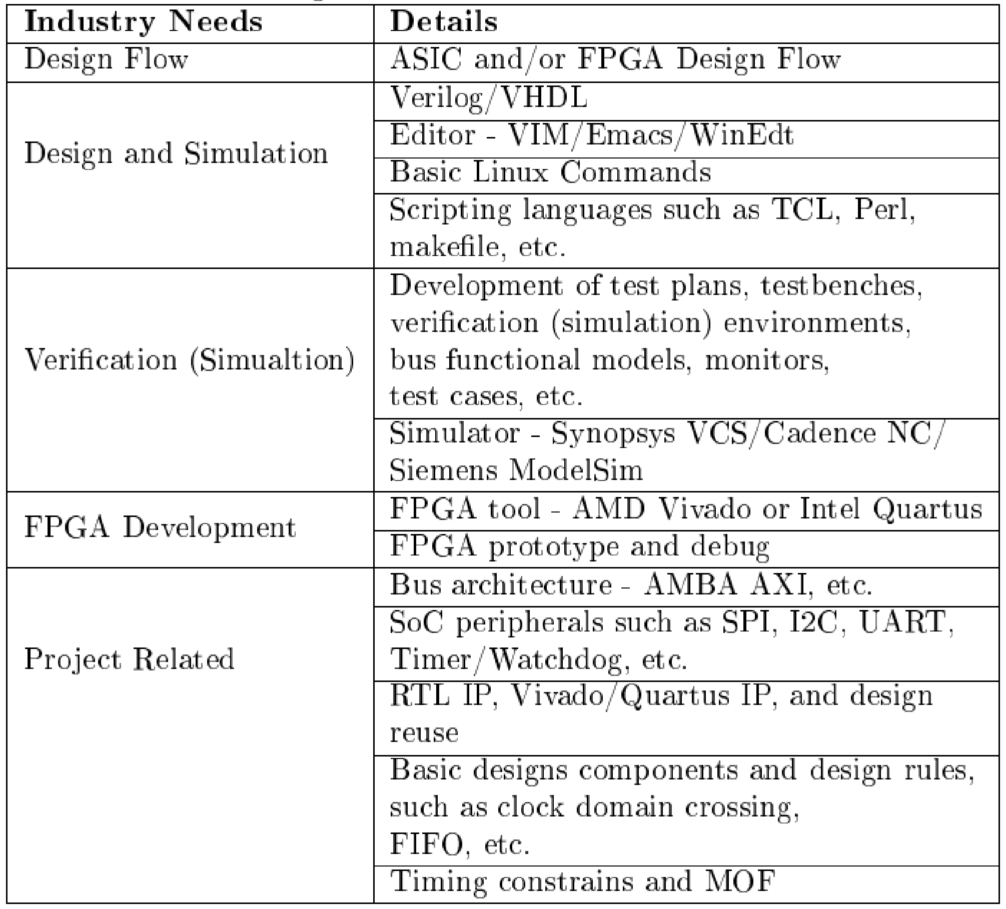
TABLE 2.1
Key Qualifications of Industry Needs for Entry-Level Engineer Positions in the Front-End IC Design

Industry Needs	**Details**
Design Flow	ASIC and/or FPGA Design Flow
Design and Simulation	Verilog/VHDL
	Editor - VIM/Emacs/WinEdt
	Basic Linux Commands
	Scripting languages such as TCL, Perl, makefile, etc.
Verification (Simualtion)	Development of test plans, testbenches, verification (simulation) environments, bus functional models, monitors, test cases, etc.
	Simulator - Synopsys VCS/Cadence NC/ Siemens ModelSim
FPGA Development	FPGA tool - AMD Vivado or Intel Quartus
	FPGA prototype and debug
Project Related	Bus architecture - AMBA AXI, etc.
	SoC peripherals such as SPI, I2C, UART, Timer/Watchdog, etc.
	RTL IP, Vivado/Quartus IP, and design reuse
	Basic designs components and design rules, such as clock domain crossing, FIFO, etc.
	Timing constrains and MOF

2.17 Basic Qualifications Needed For IC Design

- key qualifications
  - Sought by the industry for entry-level engineer positions in the front-end IC design
    - RTL Design and Simulation
    - Functional Verification (Simulation) and FPGA Verification
      - The tutorial of Siemens ModelSim is provided in Appendix B.
      - The AMD Vivado tutorial is provided in Appendix C.
    - Industry Design Rules and Design Standards

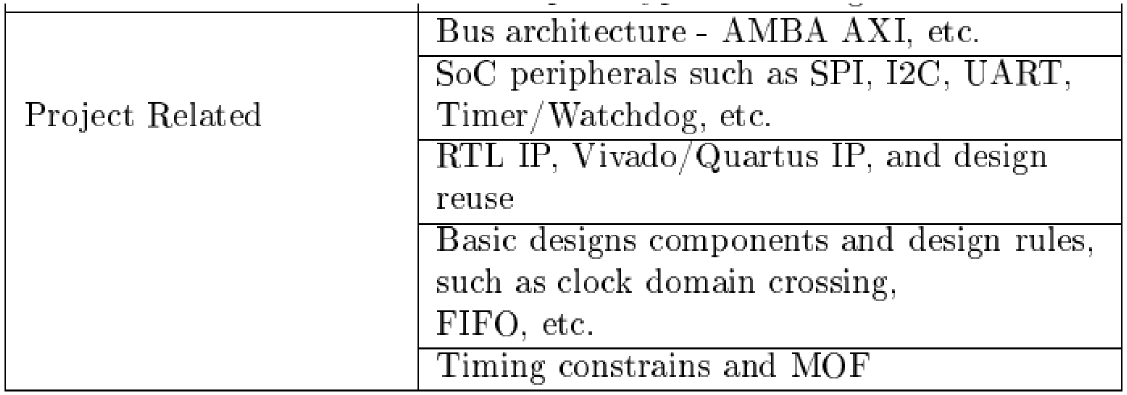
TABLE 2.1
Key Qualifications of Industry Needs for Entry-Level Engineer Positions in the Front-End IC Design



Industry Needs	**Details**
Design Flow	ASIC and/or FPGA Design Flow
Design and Simulation	Verilog/VHDL
	Editor - VIM/Emacs/WinEdt
	Basic Linux Commands
	Scripting languages such as TCL, Perl, makefile, etc.
Verification (Simualtion)	Development of test plans, testbenches, verification (simulation) environments, bus functional models, monitors, test cases, etc.
	Simulator - Synopsys VCS/Cadence NC/ Siemens ModelSim
FPGA Development	FPGA tool - AMD Vivado or Intel Quartus
	FPGA prototype and debug
Project Related	Bus architecture - AMBA AXI, etc.
	SoC peripherals such as SPI, I2C, UART, Timer/Watchdog, etc.
	RTL IP, Vivado/Quartus IP, and design reuse
	Basic designs components and design rules, such as clock domain crossing, FIFO, etc.
	Timing constrains and MOF

2.17 Basic Qualifications Needed For IC Design

- key qualifications
  - Sought by the industry for entry-level engineer positions in the front-end IC design
    - RTL Design and Simulation
    - Functional Verification (Simulation) and FPGA Verification
    - Industry Design Rules and Design Standards
      - Fundamental design blocks and design rules, like clock domain crossing (CDC), finite state machines (FSMs), synchronous/asynchronous FIFOs, design constraints pertaining to achievable clock frequency, timing considerations, and more.
      - Widely-used industrial protocols such as the AMBA AXI architecture, as well as bus peripherals like I2C, SPI, and UART.



Project Related	Bus architecture - AMBA AXI, etc.
	SoC peripherals such as SPI, I2C, UART, Timer/Watchdog, etc.
	RTL IP, Vivado/Quartus IP, and design reuse
	Basic designs components and design rules, such as clock domain crossing, FIFO, etc.
	Timing constrains and MOF