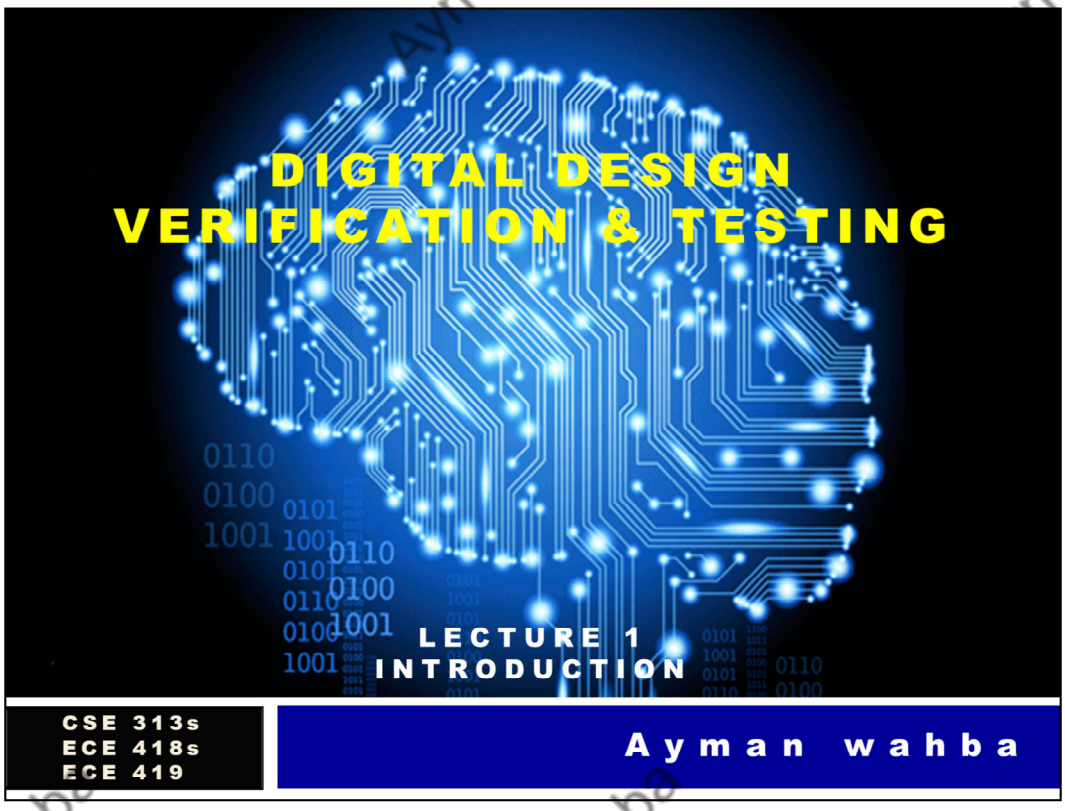
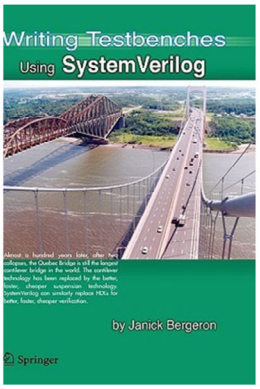

DIGITAL DESIGN
VERIFICATION & TESTING
LECTURE 1
INTRODUCTION
CSE 313s
ECE 418s
ECE 419
Ayman wahba

Course Information

- **Instructor**
  - Ayman M. Wahba, Ph.D. ayman.wahba@eng.asu.edu.eg
- **Lecture Time and Location**
  - Tuesday 10:00 – 12:00 Weekly
- **Office Hours**
  - Available through Microsoft Teams all days of the week
  - Available at my office on Saturday 9:00 – 13:00

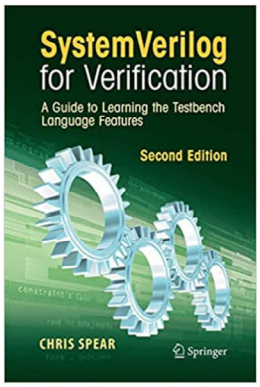
Textbook

- Janick Bergeron, Writing Testbenches using System Verilog, Springer, 2006.



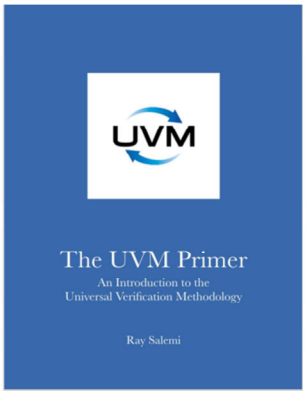
Textbook

- Chris Spear, System Verilog for Verification: A Guide to Learning the Testbench Language Features, *2nd* Edition, Springer, 2008.



Textbooks

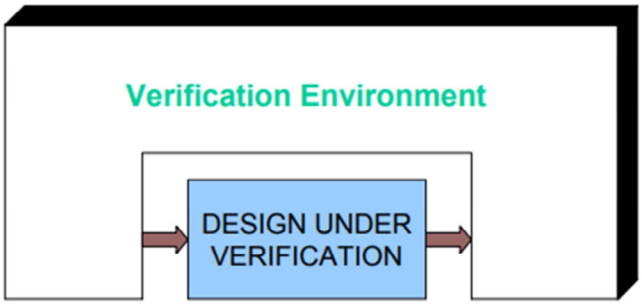
- Ray Salemi, The UVM Primer: An Introduction to the Universal Verification Methodology, Boston Light Press, 2013.



What is Verification?

7

- **Verification is the process used to demonstrate that the intent of design is preserved in its implementation**
- **70% of design effort goes behind verification**

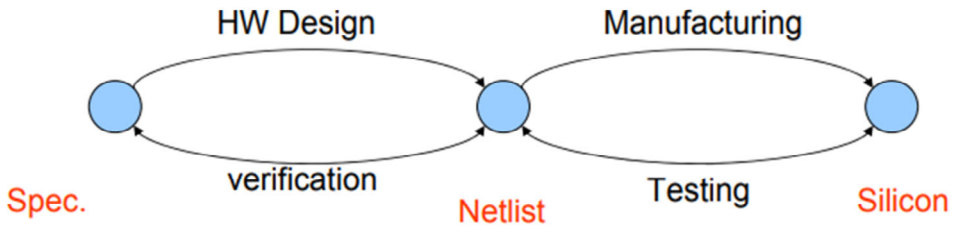


- Verification is a way in which you make sure an implemented design is functionally correct with respect to a design specification.
- Every design starts by capturing the design specifications
- These design specifications are then interpreted by a set of people who translate the specifications into a design entry typically using HDL like Verilog or System Verilog. Then we get a design “reference model”. This whole process is called RTL coding.

Testing versus Verification

8

- **Testing verifies that the design was manufactured properly.**
- **Verification ensures that a design meets its functional intent**

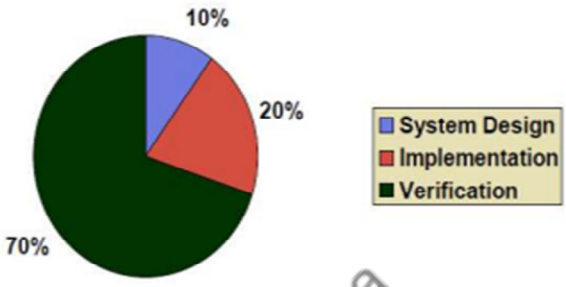


- Verification is a way in which you make sure an implemented design is functionally correct with respect to a design specification.
- Every design starts by capturing the design specifications
- These design specifications are then interpreted by a set of people who translate the specifications into a design entry typically using HDL like Verilog or System Verilog. Then we get a design "reference model". This whole process is called RTL coding.

Importance of Verification

9

- Bugs escaping to silicon can be costly including re-spin
- 70% of design cycles is spent on design verification.
- Ever increasing complexity of designs makes this harder.
- Hence Verification is always on the critical path for any product design

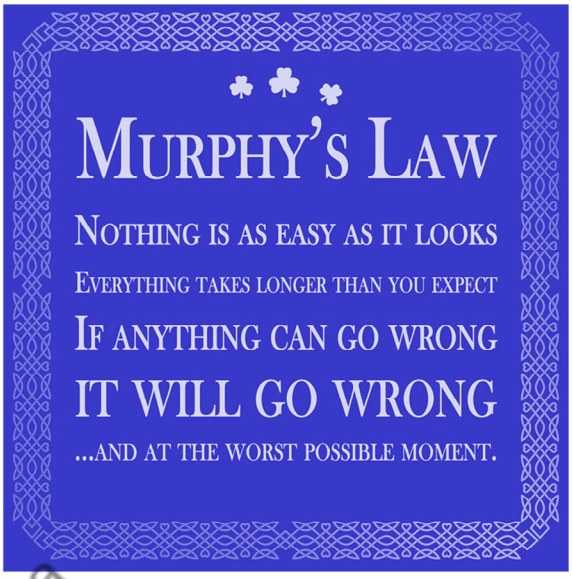


- It is important to understand that any bug that escapes the design process to the silicon can be very costly and can also result in a re-spin of the same chip. Unlike the software which can have a batch to solve any bugs even after production.
- There are several cases when after an ASIC is finalized into a chip it fails with real hardware. The bug has to be analyzed, fixed, and the ASIC has to go through a respin, meaning getting built again.
- Respins are expensive in terms of cost and deliveries, and that is why verification is very very important.
- The re-spin issue can be solved if people were prepared to spend more time and money on verification. Any bug that escapes into the silicon in a chip design can be very costly, that is why it is also important to understand the importance of verification overall the chip design process.
- Statistical data shows that around 70% of the project time is spent in verification.
- This is very important specially in the ever increasing complexity of different designs.
- In every design project Verification is on the critical path for the product design.
- Data collected from the industry shows that time spent in the system design represent only 10% of the project time. The implementation takes 20% of the project time, while 70% of the time is needed for the verification.

- Verification is not a testbench, nor is it a series of testbenches.
- Verification is a process used to demonstrate that the intent of a design is preserved in its implementation.
- We will see the differences between various verification approaches as well as the difference between testing and verification.
- If you survey hardware design groups, you will learn that between 60% and 80% of their effort is dedicated to verification.
- This may seem unusually large, but included in "verification" all debugging and correctness checking activities, not just writing and running testbenches.
- Every time a hardware designer pulls up a waveform viewer, he or she performs a verification task.
- With today's ASIC and FPGA sizes and geometries, the challenge is to get the right design, working as intended, at the right time.

Why to Verify?

11



Edward Murph:
(JAN-11-1918, JUL-17-1990)

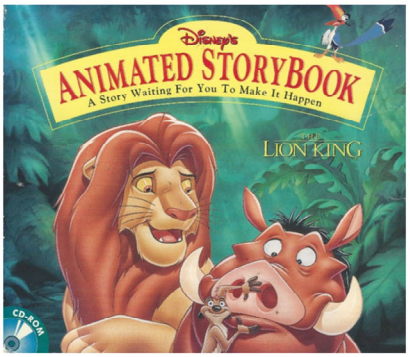
An American aerospace engineer who worked on safety-critical systems.

- Why to verify? Simply because nothing is perfect.
- Once you stat writing a code bugs arrive.
- The design verification step establishes the quality of the design and ensures the success of the project by uncovering potential errors in both the design and the architecture of the system.
- The next few slides show some projects which failed due to lack of verification, and the consequences were very serious.

Disney's Lion King Case (1994)

12

- The Disney company released its first multimedia CD-ROM game for children. The *Lion King Animated Storybook*.
- Sales were huge
- Disney failed to properly test the software on the many different PC models available on the market. The software worked on a few systems, but not on the most common systems that the general public had.



- W

Intel Pentium FDIV Bug, 1994

13

- Pentium failed to compute this operation correctly:
  (4195835 / 3145727) * 3145727 – 4195835
- The way Intel handled the situation:
  - Intel attempted to diminish its perceived severity through press releases and public statements.
  - Intel claimed the common user would experience it once every 27,000 years while IBM, manufacturer of a chip competing with Intel, claims that the common user would experience it once every 24 days
  - Intel had to replace the faulty chips.
  - This resulted in a loss of $495 Million

- Try to perform the following operation:

  (4195835 / 3145727) * 3145727 - 4195835 = ???

  The result should be 0, however the early versions of Pentium processors were generating another value.

  Try it yourself on different calculators, and on your phone. It will generate wrong results on many versions of iPhone.

- The way Intel handled the situation:
  - The problem was found before the chip was released.
  - Intel's management decided that it wasn't severe enough to warrant fixing it, or even publicizing it.
  - Once the bug was found, Intel attempted to diminish its perceived severity through press releases and public statements.
  - Intel claimed that the common user would experience it once every 27,000 years while IBM, manufacturer of a chip competing with Intel's Pentium, claims that the common user would experience it once every 24 days
  - When pressured, Intel offered to replace the faulty chips.
  - This resulted in a loss of $475 Million

More on Pentium FDIV bug

- October 19, 1994: The bug was discovered by Thomas R. Nicely (a professor of mathematics at Lynchburg College)
- October 24: The bug was reported to Intel
  - Intel admitted that they had been aware of the problem since May 1994, when the flaw was discovered by Tom Kraljevic, a Purdue University student working for Intel in Hillsboro during his internship.
- October 30: Thomas reported the bug to friends for independent confirmation
- November 7: EE Times wrote an article about the bug
- November 21: CNN published an article about the bug
  - Intel offered to replace parts for users who can show they were "affected"
  - Public outcry; IBM joins condemnation
- December 1994: Intel replaces for "all users who request it"
- January 1995: $495M charge against earnings ($750M today)
- Background: Intel had gone from a no-name PC-component supplier to starting "Intel Inside" in 1991

The original bug find was by Tom Kraljevic, a Purdue intern student working for Intel in Hillsboro! But Intel never made it public (or so says Wikipedia).

Yes, the half-billion dollar charge was from, according to Wikipedia, "only a small fraction of users" requesting replacement CPUs.

Did they mishandle the issue? Perhaps. They were learning how big, famous companies must behave!

The motto – if you're successful and you make money – then everyone wants a piece of you.

NASA Mars Polar Lander,1999

15

- On December 3, 1999, NASA's Mars Polar Lander disappeared during its landing attempt on the Martian surface.
- Reason for the malfunction was the unexpected setting of a single data bit.
- The lander was tested by multiple teams:
  - One team tested the leg fold-down procedure
  - Another team tested the landing process from that point on.
  - Both pieces worked perfectly individually, but not when put together.
- An investment of $165 Millions were lost

- The Mars Polar Lander, also known as the Mars Surveyor '98 Lander, was a 290-kilogram robotic spacecraft lander launched by NASA on January 3, 1999, to study the soil and climate of Planum Australe, a region near the south pole on Mars.
- On December 3, 1999, however, after the descent phase was expected to be complete, the lander failed to reestablish communication with Earth.
- A post-mortem analysis determined the most likely cause of the mishap was premature termination of the engine firing prior to the lander touching the surface, causing it to strike the planet at a high velocity.
- The total cost of the Mars Polar Lander was US$165 million. Spacecraft development cost US$110 million, launch was estimated at US$45 million, and mission operations at US$10 million.

Patriot Missile Defense System, 1991

16

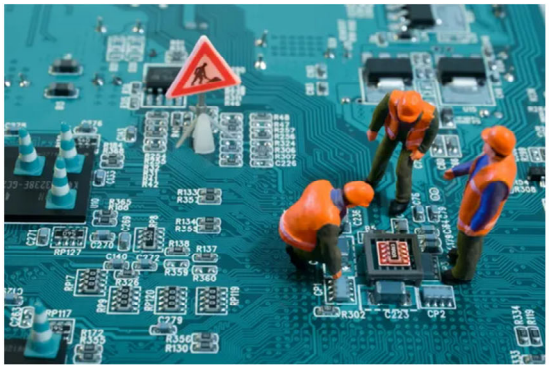
- Fail to defend against several missiles, including one that killed 28 U.S. soldiers in Dhahran, Saudi Arabia
- Timing error in the system's clock accumulated to the point that after 14 hours, the tracking system was no longer accurate.
- In the Dhahran attack, the system had been operating for more than 100 hours

- On February 25, 1991, during the Gulf War, an American Patriot Missile battery in Dharan, Saudi Arabia, failed to track and intercept an incoming Iraqi Scud missile. The Scud struck an American Army barracks, killing 28 soldiers and injuring around 100 other people.
- It turned out that the cause was an inaccurate calculation of the elapsed processor time since its boot, due to computer arithmetic errors.
- Specifically, the time in tenths of second as measured by the system's internal clock was multiplied by 1/10 to produce the time in seconds (e.g. 500 tenth of a second when multiplied by 1/10 gives 50 seconds).
- This calculation was performed using a 24 bit fixed point register. The value 1/10, which has a non-terminating binary expansion, was chopped at 24 bits after the radix point.
- Using 8 bits: 1/10 = (0.00011001) = 0.09765625. It is not 0.1
- Using 16 bits: 1/10 = (0.0001100110011001) = 0.099990844726562 It is not 0.1
- Using 24 bits: 1/10 = (0.000110011001100110011001) = 0.099999964237213 It is not 0.1
- The small chopping error, when multiplied by the large number giving the time in tenths of a second, led to a significant error.
- Indeed, the Patriot battery had been up around 100 hours i.e. 3,600,000 tenth of seconds. When this time is computed as described above it will be (3,600,000 * 0.099999964237213 = 359,999.8712539668 seconds). The difference is 0.1287460332 seconds
- A Scud travels at about 1,676 meters per second. In this 0.1287460332 seconds, the Scud travels around 216 meters (distance = speed * time). Consequently, Patriot would hit 216 meters away from the Scud.

Why lack of verification?

17

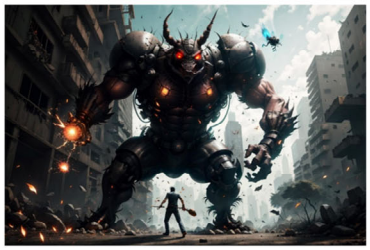
- It's clear that testing and verification are so important, so why do companies pay less attention to it?
  - To save money
  - To save time
  - To hide the inefficiencies
  - … etc.



- On

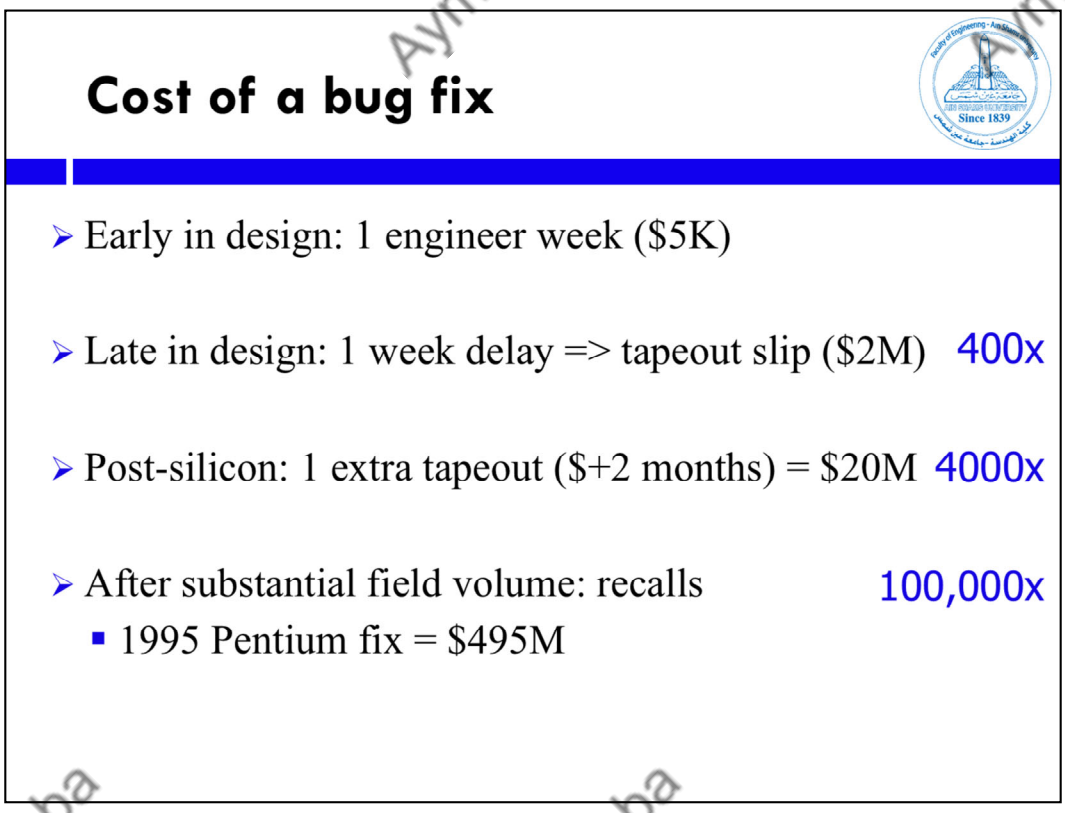
Why should we fix bugs ASAP

- Bugs in a design are like living bugs. They grow and evolve with time. It is better to find and kill them before they become so difficult to get rid of them.



As time passes the simple bugs evolve and need a huge effort to get rid of.

- If an error is discovered in the early phase of the design, the cost will be in the range of $5000.
- If the error is discovered in the late phases of the design, your time to market will be delayed by 1 week. (If you are selling one million units/year, at the rate of $100/unit. So the income is $100M/year or $2M/week). So if you are delayed by 1 week, you will lose $2M.
- It is hard to compute the cost of a tapeout. It depends on your volume, and how many layers it involves). If an error is discovered after tapeout, you will lose almost one month ($16M), plus the cost of the tape out. Assume $1M/layer, i.e. $4M for 4 layers. This leads to a total loss of $20M.

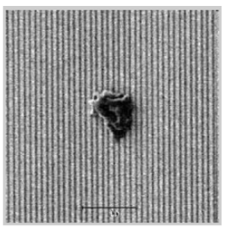


- If an error is discovered in the early phase of the design, the cost will be in the range of $5000.
- If the error is discovered in the late phases of the design, your time to market will be delayed by 1 week. (If you are selling one million units/year, at the rate of $100/unit. So the income is $100M/year or $2M/week). So if you are delayed by 1 week, you will lose $2M.
- It is hard to compute the cost of a tapeout. It depends on your volume, and how many layers it involves). If an error is discovered after tapeout, you will lose almost one month ($16M), plus the cost of the tape out. Assume $1M/layer, i.e. $4M for 4 layers. This leads to a total loss of $20M.

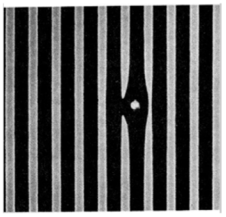
Main Sources of Bugs

20

- Design Errors:
  - This mainly comes due to misinterpretation of the specification.
  - Wrong component (e.g. AND instead of OR)
  - Incorrect wiring (e.g missing wire, extra wire)
- Fabrication Defects:
  - Due to problems in the fabrication process (e.g dust particle falling on the chip destroying thousands of transistors)
  - Stuck at faults
  - Bridging faults
- Physical failures:
  - Due to aging through the chip life cycle



Example of a contaminating particle



Example of a break in a meta line

- Defects: occur during manufacturing, and are detected by tests performed after the end of manufacturing (if covered by the tests). Defect densities are generally about 0.2/mm^2, with a 10 cm^2 wafer having about 200 defects on its surface.
- Failures: occur during service, after the chip has been sold and used.
- Stuck-at-faults: this is the most common fault model used in industry. It models manufacturing defects which occurs when a circuit node is shorted to VDD (stuck-at-1 fault) or GND (stuck-at-0 fault) permanently.
- Bridging faults: they correspond to shorts between signal lines. They may occur during manufacturing, between adjacent metal lines in the same chip layer, or between adjacent lines on different chip layers.

- Images: Maly, Wojciech. (1987). Realistic fault modeling for VLSI testing. 173- 180. 10.1109/DAC.1987.203239.

Evolution of verification

- The "good old days"
  - Designs were relatively simple
  - Everyone "verified" their own work
  - … and it usually worked fine!
- But then designs got more complex
  - Billions of transistors, numerous clock domains, embedded CPUs, …
  - Too many tests to write
  - Specialized infrastructure and skills are needed
  - And so… specialists are needed!
- Verifying your own design is a bad idea
  - We humans are really bad at finding our own flaws

Designs are becoming huge. For example, Nvidia Ampere chip has over 50 billion transistors

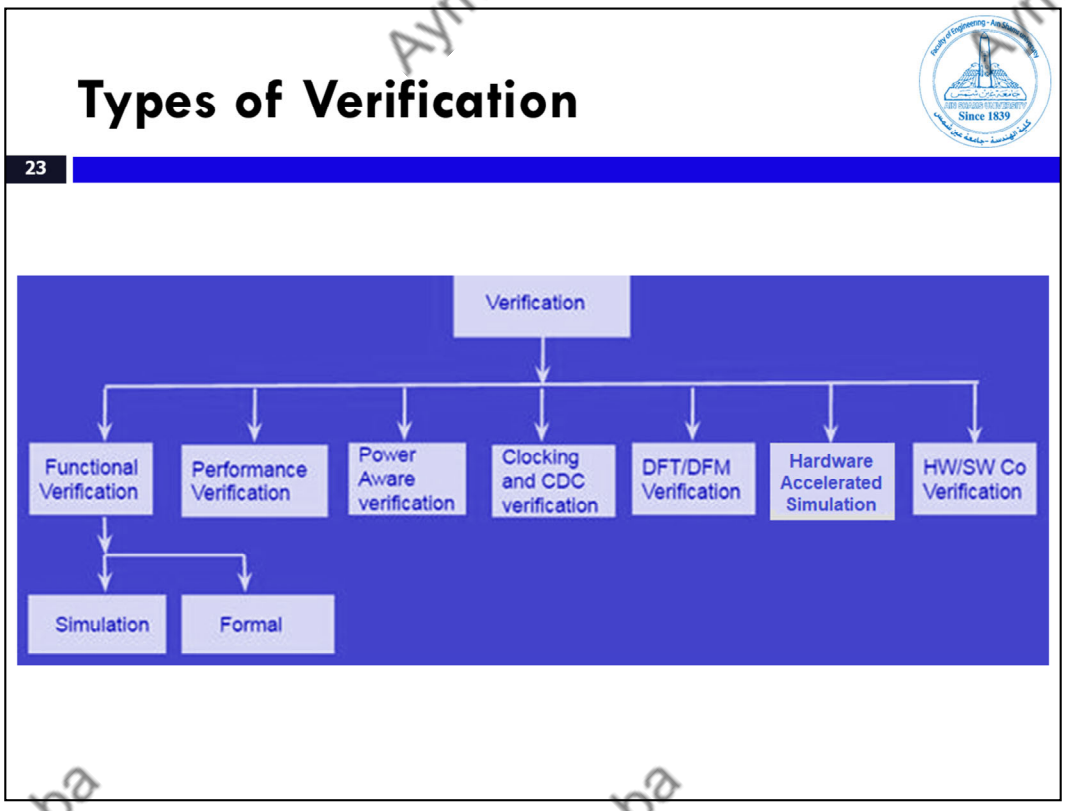
Why not trying all input scenarios?

22

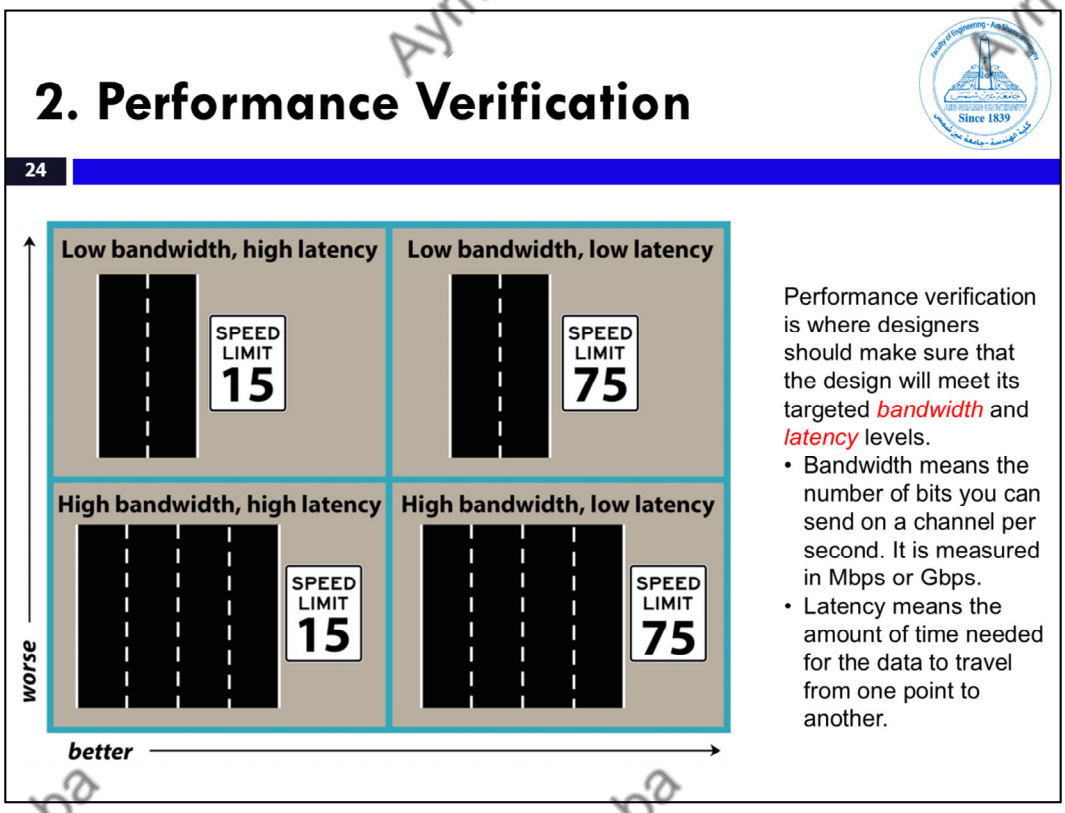
- Trying all possible combinations of circuit inputs to test it, is called exhaustive simulation.
- Although this approach is possible for circuits with a small number of inputs, but it is practically impossible for bigger circuits.
  - Consider a 64-bit full adder.
  - It has 129 inputs (two 64-bit operands and one carry in)
  - The number of input combinations is 2^{129}
  - Assume the tester can perform 1 Giga simulations per second)
  - So simulating all possible input combinations will take:

$$(2^{129} / 10^{9})/(60 \times 60 \times 24 \times 365) = 2.158 \times 10^{22} \text{ years}$$

- Defects: occur during manufacturing, and are detected by tests performed after the end of manufacturing (if covered by the tests). Defect densities are generally about 0.2/mm^2, with a 10 cm^2 wafer having about 200 defects on its surface.
- Failures: occur during service, after the chip has been sold and used.
- Stuck-at-faults: this is the most common fault model used in industry. It models manufacturing defects which occurs when a circuit node is shorted to VDD (stuck-at-1 fault) or GND (stuck-at-0 fault) permanently.
- Bridging faults: they correspond to shorts between signal lines. They may occur during manufacturing, between adjacent metal lines in the same chip layer, or between adjacent lines on different chip layers.



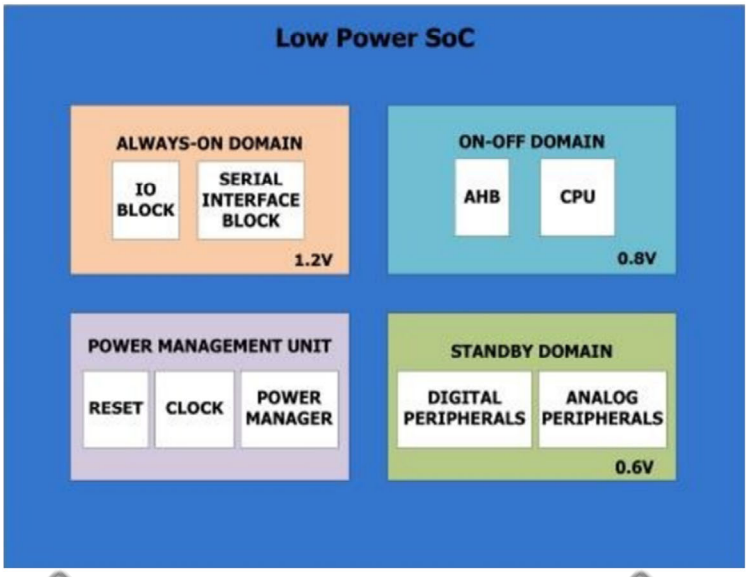
- There are many aspect of verification, as shown in the figure.
- We will give a hint about each one of them in the next few slides, and then we will focus more on the "*functional verification*"



- Performance verification is where designers should make sure that the design will meet its targeted *bandwidth* and *latency levels*.
- *Bandwidth* means the number of bits you can send on a channel per second. It is measured in Mbps or Gbps. (e.g. How many cars can enter Cairo – Alexandria road every second?)
- *Latency* means the amount of time needed for the data to travel from one point to another. (How long does it take a car to go from Cairo to Alexandria ? 2 hours for example)
- To get the required performance you must have enough resources to handle the traffic and load on the system.
- To bring the idea closer to your mind, we will give an example for clarification.
- Assume you have a restaurant, can it handle the traffic and demands?
  - Do you have enough ovens, burners, cooks, tables, waiters, etc.?
  - Also what would be the best and worst case latencies?
- Similar questions are asked if instead of a restaurant you are dealing with a CPU or a bus interface.
- A special language has been developed to help in performance verification. The Performance Description Language PDL provides a compact notation for the specification of non-functional attributes of VLSI systems.

3. Power Aware Verification

25



Power aware verification is essential to verify the operation of a design under active power management, including:

1. The power management architecture.
2. State retention and restoration of subsystems when powered down.
3. The interaction of subsystems in various power states.

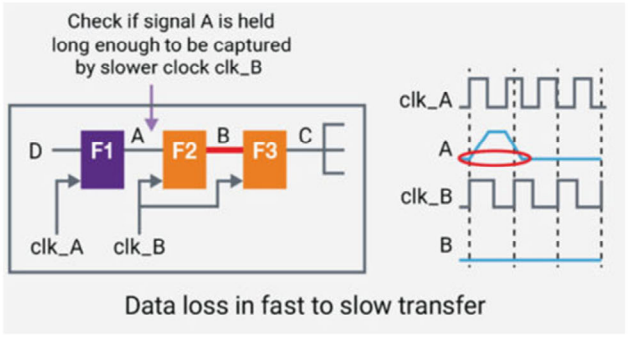
- Modern systems are extremely complex, and are typically used in portable systems that must support increasingly longer battery life, therefore the minimization of power consumption is not an option, it is a must.
- Even non-portable systems must avoid wasting energy, to minimize both power and cooling costs.
- In both cases, *active power management* is required to ensure energy efficiency.
- Active power management enables the design of low power chips and systems, and also creates many *new verification challenges*.
- In a System-on-Chip (SoC) design with active power management, various subsystems can be independently powered up or down, and/or powered at different voltage levels. It is important to verify that the SoC works correctly under active power management.
  - When a given subsystem is turned off, its state will be lost, unless some or all of the state is explicitly retained during power down. When that subsystem is powered up again, it must either be reset, or it must restore its previous state from the retained state.
  - When a subsystem is powered down, it must not interfere with the normal operation of the rest of the SoC.

Power aware verification is essential to verify the operation of a design under active power management, including the power management architecture, state retention and restoration of subsystems when powered down, and the interaction of subsystems in various power states.

- IEEE conceived a special standard format for the defining power management architecture, to facilitate its verification (IEEE 1801-2009 Unified Power Format (UPF)).

4. Clocking & CDC Verification

26



Data loss in fast to slow transfer

One of the problems that may happen due to Clock Domain Crossing

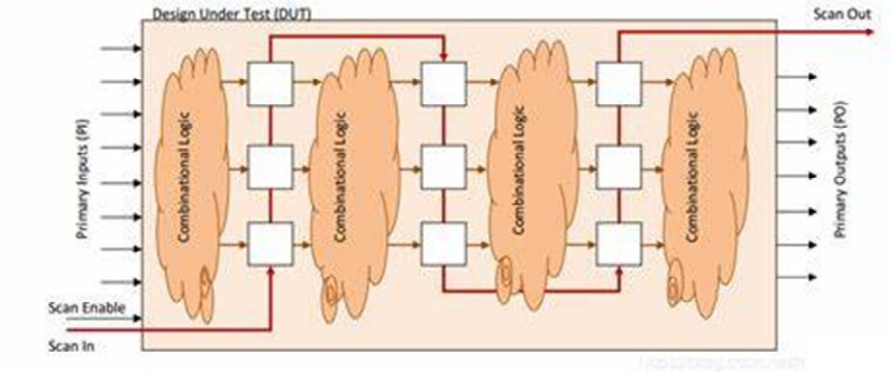
Dealing with multiple clock domains in a single SoC creates many types of problems such as:

1. Data loss in fast to slow clock domain transfer.
2. Improper data enabling, where the data is enabled while it is still changing.
3. Convergence of synched signals. When two signals are originating from different clock domains. If these two signals converge at an AND gate, the probability of glitches is high.
4. Reset synchronization: When a transmitting f/f in a clock domain is reset, and another receiving f/f in a different clock domain, has an uncorrelated reset or no reset, this results in glitches, and loss of functional correlation.

- With the increasing complexity of SoC (Systems on Chip), *multiple and independent clocks* are essential in the design.
- Clock Domain Crossing (CDC) verification becomes an integral part of any SoC design cycle.
- A typical SoC consists of several blocks (each with its own set of clocks) stitched together. This leads to several new paths at SoC which are not anticipated at the block level.
- There are several approaches for CDC verification:
  - **Flat SoC Verification:** This has been the conventional approach of running CDC on any design. In this scheme, we run verification on the whole design as one entity. This type of CDC verification however can be performed only when the whole design is available and well integrated. In this case we may find any critical bug very late in the design cycle as we are waiting for well integrated SoC to begin the verification. It is difficult to scale such kind of scheme for large SoCs because of huge number of violations and huge run time.
  - **Black boxing:** In this approach, certain blocks of the SoC may be black boxed for CDC analysis. By black boxing we save run time considerably as the block is not analyzed for CDC. This can be done selectively for bulky blocks.
  - **Gray boxing:** For a large SoC, this approach assumes that the blocks delivered to the SoC are CDC clean. We check only Inter block violations and not those that are totally within an block.

5. DFT Verification

27



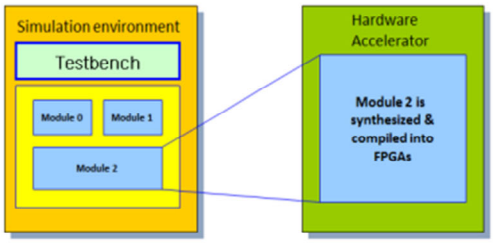
- Due to limitations in the number of chip pins, it is impossible to reach the internal parts of the chip to test them. One DFT technique is to insert what is called scan cells connected in a chain to be able to set internal nodes to some required testing values
- DFT verification ensures that the inserted cells are correctly implemented, do not interfere with the normal functionality of the design, and meet the performance requirements.

- DFT verification is the process of checking the design for testability (DFT) features in the design. DFT features are added to the design to enable testing of the chip after fabrication, such as scan chains, and built-in self-test (BIST) circuits. DFT verification ensures that these features are correctly implemented, do not interfere with the normal functionality of the design, and meet the test coverage and performance requirements.
- DFT verification is important for the following reasons:
  - First, it can save time and cost by detecting and fixing DFT errors early in the design cycle, before they propagate to the synthesis, layout, and physical verification stages.
  - Second, it can improve the test quality and reliability of the chip by ensuring that the DFT features work as intended and can detect and diagnose any faults in the chip.

6. Hardware-Accelerated Simulation

28

- Simulation performance is improved by moving the time-consuming part of the design to hardware.
- Usually, the software simulation communicates with FPGA-based hardware accelerator
- Challenges
  - Generally improves speed but degrades on HW-SW (Testbench) communication
  - Abstracting HW-SW communication at transaction level rather than cycle level desired for better speeds



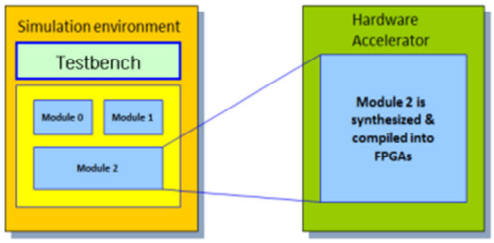
- HW Emulation
  - Full mapping of HW into an emulator (array of FPGAs)
  - More like a real target system. Speed up possible upto 1000X simulation
  - Debug is a challenge with limited visibility
  - Usually used for HW+SW co-verification

- Hardware accelerated simulation is an alternative method used especially at the higher levels of verification like a system or a full chip verification, primarily because the simulation-based verification tends to slow down as the size of design increases. That is when at full chip or SOC level since the design size is huge and the simulations run pretty slow and that is why we look for alternative methods.
- That is primarily because the simulator is a software program that runs on a host CPU that runs at every cycle or at every change in a signal.
- At a higher level as an alternative, we try to move the time consuming part of the design to the hardware. In the shown diagram on the left we have a set of modules representing your design implementation, and the testbench module which comprises all the behavioral components like stimulus generators, checkers, … etc., used in the simulation environment. The most time consuming part of this simulation environment is the design implementation in terms of modules, because those are the ones that need to get evaluated at every cycle, or at any event of signal change.
- In this approach, we take those time consuming modules, and synthesize and compile them into real hardware FPGAs, so they can run on a real hardware much faster.
- We still keep the testbench components running on the host CPU, and then we use some means of communication between the testbench components existing on the software simulator talking to the hardware implementation on the FPGA.
- So we can still gain speeds of 10x – 20x of your simulation.

6. Hardware-Accelerated Simulation

29

- Simulation performance is improved by moving the time-consuming part of the design to hardware.
- Usually, the software simulation communicates with FPGA-based hardware accelerator
- Challenges
  - Generally improves speed but degrades on HW-SW (Testbench) communication
  - Abstracting HW-SW communication at transaction level rather than cycle level desired for better speeds



- HW Emulation
  - Full mapping of HW into an emulator (array of FPGAs)
  - More like a real target system. Speed up possible upto 1000X simulation
  - Debug is a challenge with limited visibility
  - Usually used for HW+SW co-verification

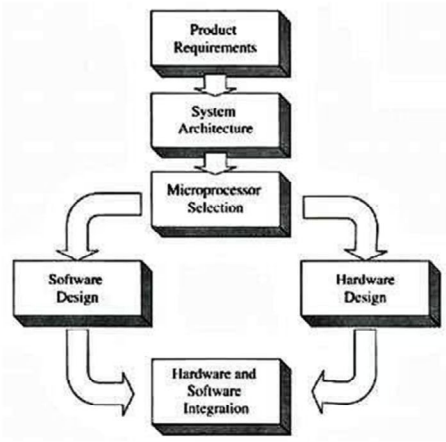
- This definitely comes with some challenges. As I mentioned the speed can be improved using this approach as we move some design components into a real hardware system like FPGAs. But then based on how the HW & SW communicate, this can still degrade your speed. The communication happens every clock cycle, or at every signal change event, so you lose a lot of benefit in terms of speedup.
- So, the approach that is normally used is to abstract this HW/SW communication to a transaction level rather than to a cycle level or signal change level. In that case you don't need to communicate between the HW and SW at every clock cycle. You will need only to communicate at the transaction boundary.
- This is accelerated HW simulation.

HW emulation:

- To further improve the speeds, we can do a full mapping of the hardware to the emulator (in this case we have an array of FPGAs not a single one).
- The hardware emulation will look more like a real target system. Speedups can reach like 1000x.
- There can be several approaches for doing HW Emulation. We can either synthesize your testbench if the testbench is either done in a synthesizable fashion on the emulator, or we can set up configurations which look like a real hardware system and then use another real stimulus generator that might be used for real silicon testing.
- In either cases debug is still a challenge on hardware emulations since we may not have a full visibility of the internals of the design.
- Hardware emulation is typically used for co-verifying the hardware and the software together.

7. HW/SW CoVerification

30

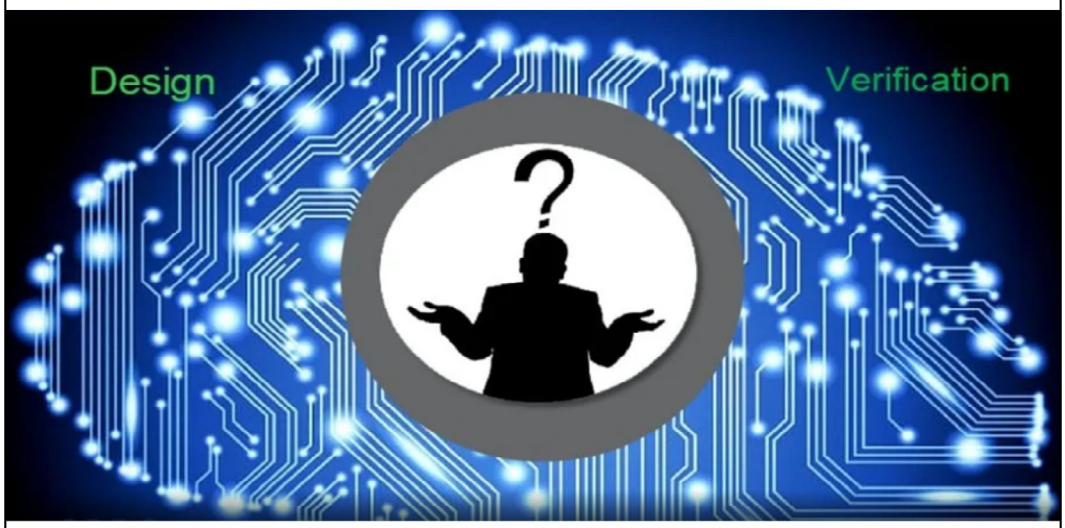


HW/SW co-verification means verifying embedded system software executes correctly on embedded system hardware. It means running the software on the hardware to make sure there are no hardware bugs before the design is committed to fabrication.

- Hardware/software co-verification is how to make sure that an embedded system software works correctly with the hardware, and that the hardware has been properly designed to run the software successfully, before large sums of money are spent on prototypes or manufacturing.
- Co-verification addresses one of the most critical steps in the embedded system design process, the integration of hardware and software. The alternative to co-verification has always been to simply build the hardware and software independently. Try them out in the lab, and see what happens, and hope for the best. If by chance it does not work, try to do what you can to modify the software and hardware to get the system to work.
- This practice is called testing and it is not as comprehensive as verification. Unfortunately, finding out what is not working while the system is running is not always easy. Controlling and observing the system while it is running may not even be possible.
- The first two commercial co-verification tools were Eaglei from Eagle Design Automation and Seamless CVE from Mentor Graphics. These tools appeared in the 1995-1996 time frame and both were created in Oregon. Eagle was later acquired by Synopsys, became part of Viewlogic, and was finally killed by Synopsys in 2001 due to lack of sales.

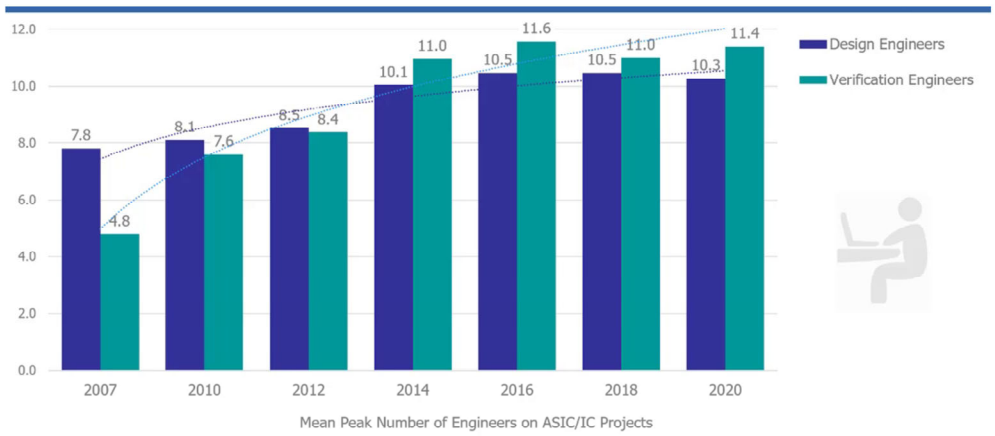
Design vs. Verification Career

31



- Ver

Design vs. Verification Engineers



- CAGR (Compound Annual Growth Rate) for verification is 6.88%
- CAGR (Compound Annual Growth Rate) for design is 2.16%

- CAGR: is the annualized average growth rate of a value between two given years.
- It is calculated as follows:
  - $CAGR_{\text{year X to year Z}} = [(\text{Value in year Z}/\text{Value in year X})^{1/(Z-X)} - 1]$

Verification Engineer Skills

33

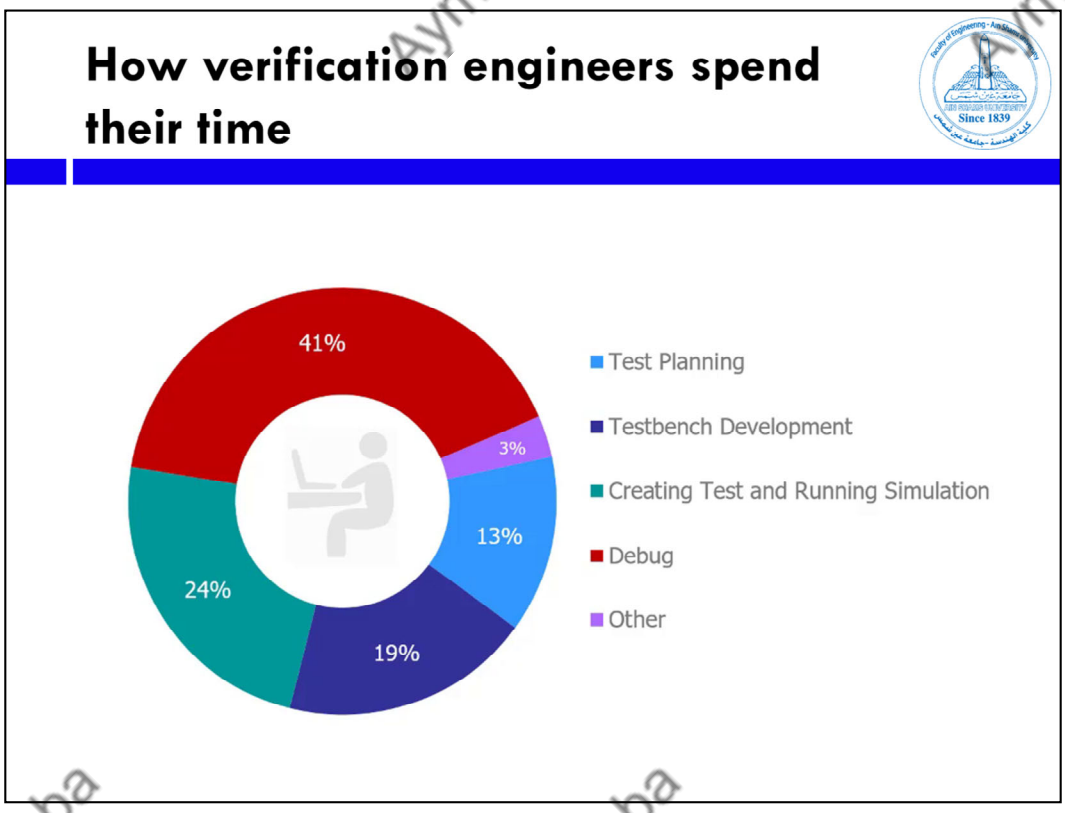
- Based on recent market demands, it is evident that there are lots of opportunities and a strong career path for verification engineers.
- Over the past few years, the complexity of designs have increased and continues to increase.
- Verifying a design is always crucial as any functional defect in the manufactured chip is going to cost huge amount of money in terms of a new tape out, also there will be a considerable risk of losing a design win opportunity in the market.
- In the life cycle of a design, there are always bugs to be found in less time in every project.

Role of verification engineers

34

- Read and understand the hardware specifications.
- Specifications are sometimes ambiguous, missing details, or having conflicting descriptions.
- Create the verification plan, and then follow it to build tests.
- The behavior of the device when used outside of its original purpose is not your responsibility, although you want to know where those boundaries lie.
- Make sure the design can accomplish its task successfully.

- The goal of hardware design is to create a device that performs a particular task, such as a DVD player, network router, or radar signal processor, based on a design specification.
- Your purpose as a verification engineer is to make sure the device can accomplish that task successfully – that is, the design is an accurate representation of the specification.
- Bugs are what you get when there is a discrepancy.
- The behavior of the device when used outside of its original purpose is not your responsibility, although you want to know where those boundaries lie.
- The process of verification parallels the design creation process.
- There is always ambiguity in interpreting the specifications, perhaps because of ambiguities, missing details, or conflicting descriptions.
- As a verification engineer, you must read the hardware specification, create the verification plan, and then follow it to build tests showing the RTL code correctly implements the features.


How verification engineers spend their time
Since 1839
41%
3%
13%
24%
19%
Test Planning
Testbench Development
Creating Test and Running Simulation
Debug
Other

Verification Engineer Skills

36

- Demanding both HW and SW skills
  - Digital logic, Analog, Computer Architecture, Memories
  - HDLs like SystemVerilog
  - SW Programming Concepts
    - Advanced methodologies like UVM heavily use SW concepts like OOP, factory patterns
  - Version control
    - Build, compile, managing branches
  - Scripting languages
    - Regressions, Automation

- A good verification engineer needs to have both hardware and software engineering skills. Along with strong foundation in Digital logic design, Computer architecture, Communication technologies and other domain knowledge. He should be a good programmer too.
- Most of current Verification infrastructure uses advanced software engineering concepts like Object oriented programming, factory patterns, continuous integration mechanisms as well as Hardware description languages like SystemVerilog and VHDL

Verification Engineer Skills

37

- Analytical thinking , Problem solving
  - Think how to break a design
  - Question design approach
  - Solve Debug challenges

- Communication , Collaboration
  - Most projects need team of design and verification engineers
- Innovation and continuous learning
  - Always need to find better ways

- Ver

Verification Career Questions (1)

38

Can I move from Verification to Design in my career?

- If you are good in verification and have System Verilog or Verilog handy, and have worked for few years in the same company, your company may allow you to be a designer. So in case of opening, managers try to find out who are good at executions internally. It is not uncommon to find people who worked in verification for few years, physical design for few year, and then logic designer.

Verification Career Questions (2)

39

Do design engineers get paid more than verification engineers?

- Right now a great verification engineer with experience in UVM, Formal, assertions, that can design a good strategy to achieve requirements/functional coverage is hard to find and can get paid really well, especially in domains like aerospace, defence, DO254 (safety standards for electronics that go into aircraft), nuclear, and automotive.
- It is not true, at all, that designers are paid more than verification engineers. On the contrary if you are a distinguished verification engineer you will be paid more.

Verification Career Questions (3)

40

Is it true that design engineers get to work on important stuff while verification engineers just look at test, coverage and bugs?

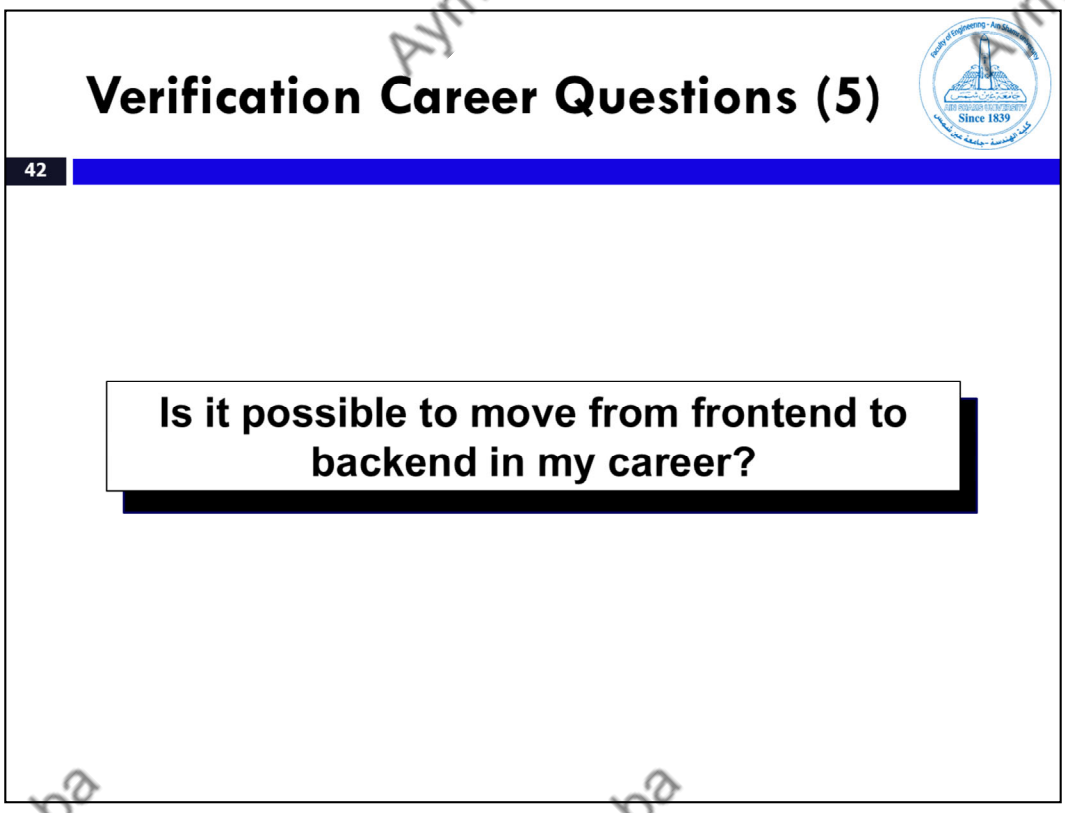
- Verification is the process to *prove that the RTL works as per the specification*.
- It is also the process to *find all the bugs present in the RTL design*.
- Moreover, it is the process to detect the defects and minimize the risks associated with the residual defects.
- Verification Engineer requires technical skills similar to RTL designers, in addition to other skills, as well.

Verification Career Questions (4)

41

Can I move from verification to software engineering?

- As a Verification engineer, you are supposed to be working on/creating testbenches which are OOP based.
- You probably use something like System Verilog in which you use classes and handles.
- What is required is getting familiar with a different stack, like Java or Python. Getting familiar with APIs, databases and in general will help.
- So, yes, you can make the switch. It will just be harder as your resumé wouldn't have the background recruiters expect to see. It just means that you'll have to be more resilient and persistent both with your preparations and with the recruiters.
- In fact, anyone can make the switch. A number of big and small companies alike don't even require you to have a computer science degree as long as you can code smartly.



- Although both frontend and backdown are two subdomains of VLSI but the work related to these domains are different.
- Front-end mostly deals with RTL coding and creating verification environment using Methodologies or HVL (Hardware Verification Language). It also deals with RTL integration, STA, CDC and writing testcases, and checking coverage. In short it mostly deals with coding.
- While in backend, the work is concerned with floor planning, routing, DFT, CMOS related concepts, analog layout, scripting and generally have less to no coding.

Verification Career Questions (6)

43

Can I become a good verification engineer if I don't like programming?

- To be a verification engineer requires technical knowledge in electronic design and also computer programming. So you have to excel in coding to be a good verification engineer.

Verification Career Facts

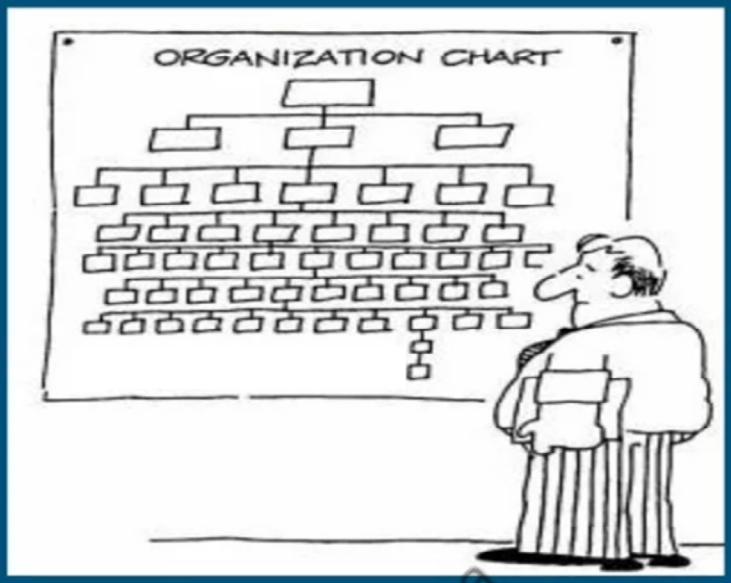
44

- Verification Engineer has a solid career path
  - Two decades back - Design engineer tested their designs
  - Today - Verification of a design needs dedicated skills
    - Scope for broader and deeper skills
  - Verification is increasingly complex and critical
    - Verification consumes majority of project time (Avg - 70%)
    - Increasing demand for Verification engineers (as per industry survey)
  - Verification engineers are involved in project from early stage of design
  - Considered at par with design engineers (including compensation)
  - Several well known Verification technologists across industry

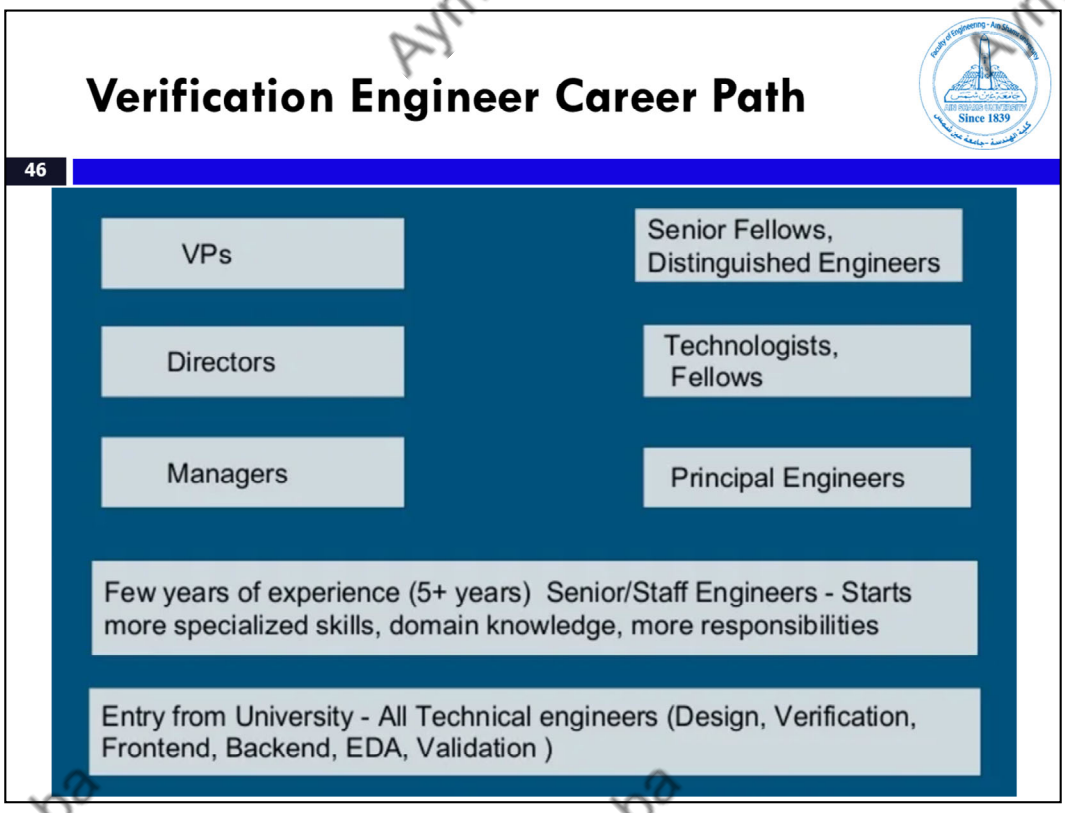
- Ver

Verification Engineer Career Path

45



- Ver



- Ver

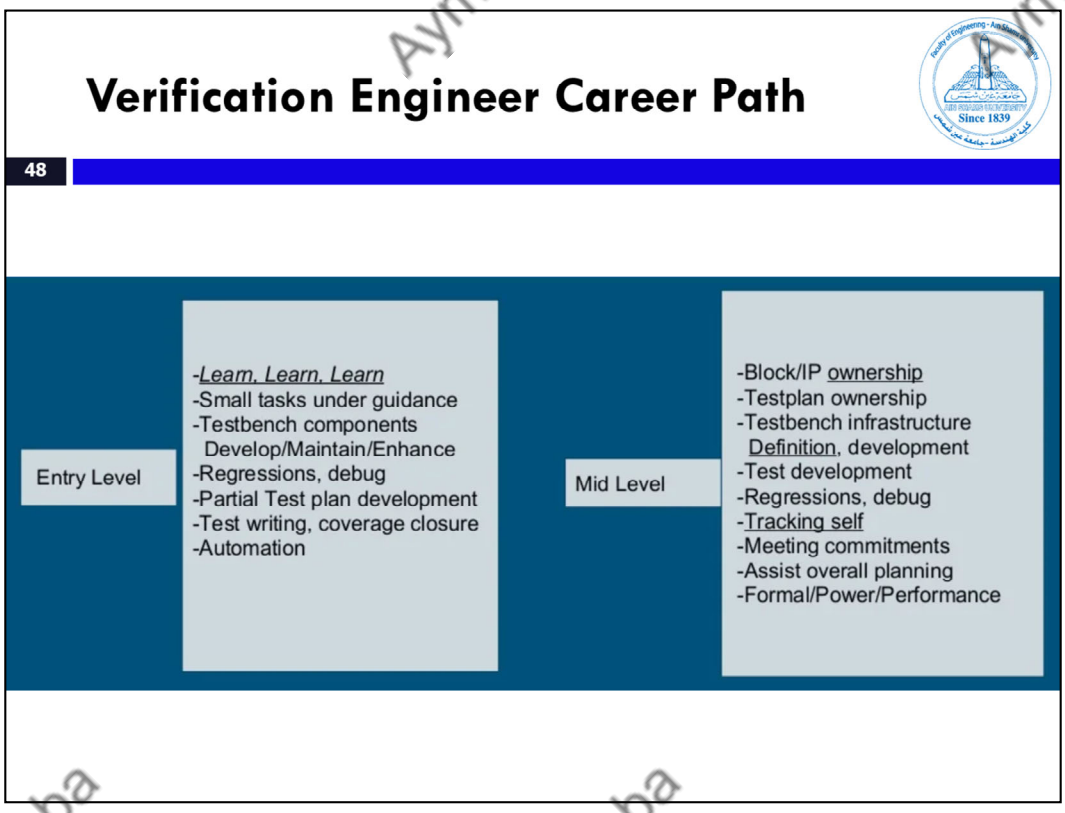
Verification Engineer Career Path

47

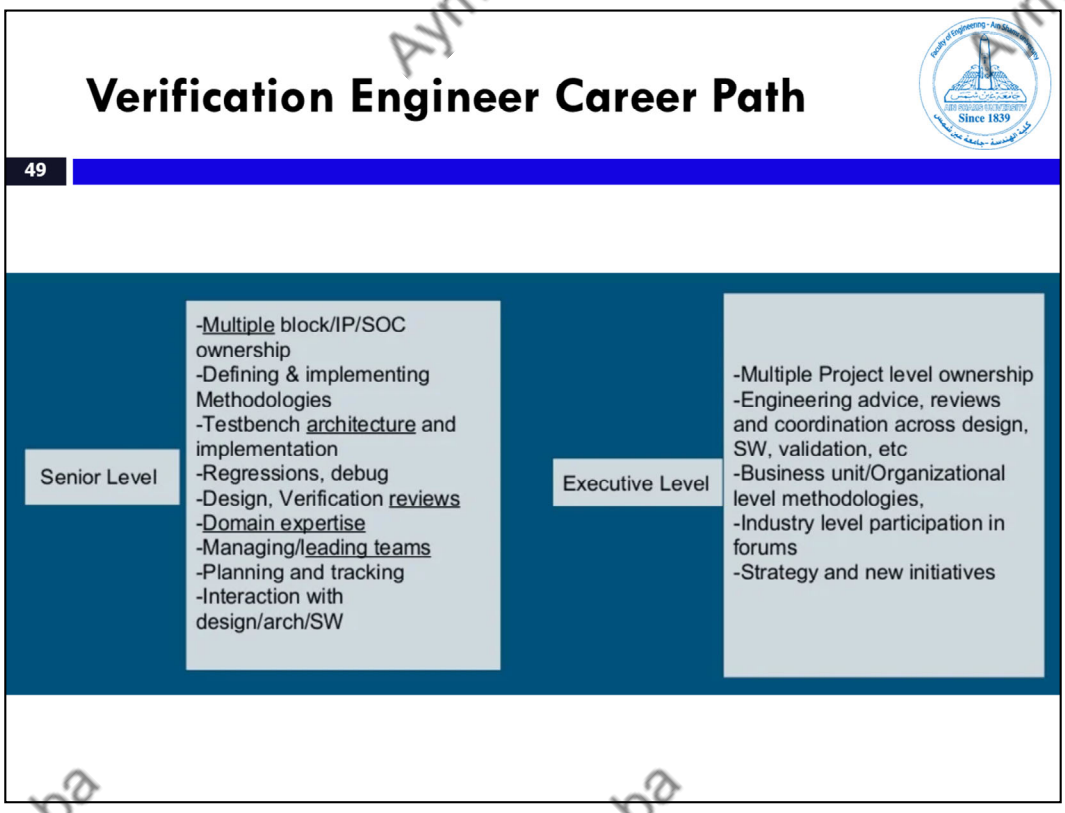
In 5 years

what will I be doing?

- Ver



- Ver



- Ver