


<https://www.youtube.com/watch?v=5tJPXYA0Nec>

# WHAT IS VERIFICATION ?

# Your Calendar

 11/09/2020

- Introduction to Verification
- SystemVerilog classes and random variables
- Lab: Using SystemVerilog

 18/09/2020

- Metrics Driven Verification Methodology
- UVM and Test Sequences
- Lab: Using UVM to develop tests

 25/09/2020:

- Functional Coverage
- Checking
- Lab: Checking and Coverage

# Introduction to Verification



# Introduction to Verification

## Session Objectives

- Know about what verification is

## Agenda


- Why do we verify ?
- What do we verify ?
- When do we verify ?
- Who verifies ?
- How do we verify ?

# About the author

 François Cerisier

<http://www.linkedin.com/in/fcerisier>

 Founder of AEDVICES Consulting, 2012

 Around “**some**” years experience in Design & Verification Consulting

 Involved in various verification methodologies of complex systems and IPs

- CDV: Coverage driven verification,
- FV: Formal Verification, Complete formal verification
- System modeling, Testbench qualification, graph based verification, hardware/software coverage verification, ...
- Requirements Based Design & Verification: DO254, Space Industry

 Verification Projects Types:

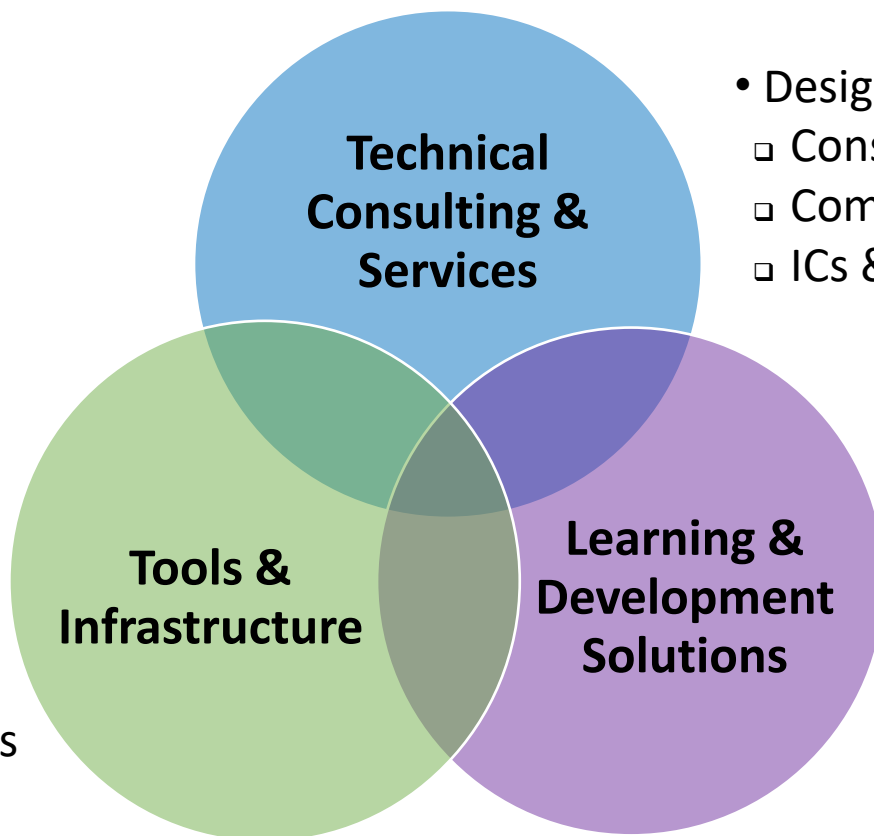
- Peripherals and IPs
- Interconnects
- Multimedia Sub-Systems
- System-On-Chips
- CPUs and DSPs
- Firmware / Hardware, Low Level RT Software
- EDA support for US start-ups

 Worked on projects:





- Verification IPs and EDA tools
- Application Engineering
- Methodology implementations:
  - Verification standards (UVM, PSS)
  - Non-regression, continuous integration, requirements driven verification, ++

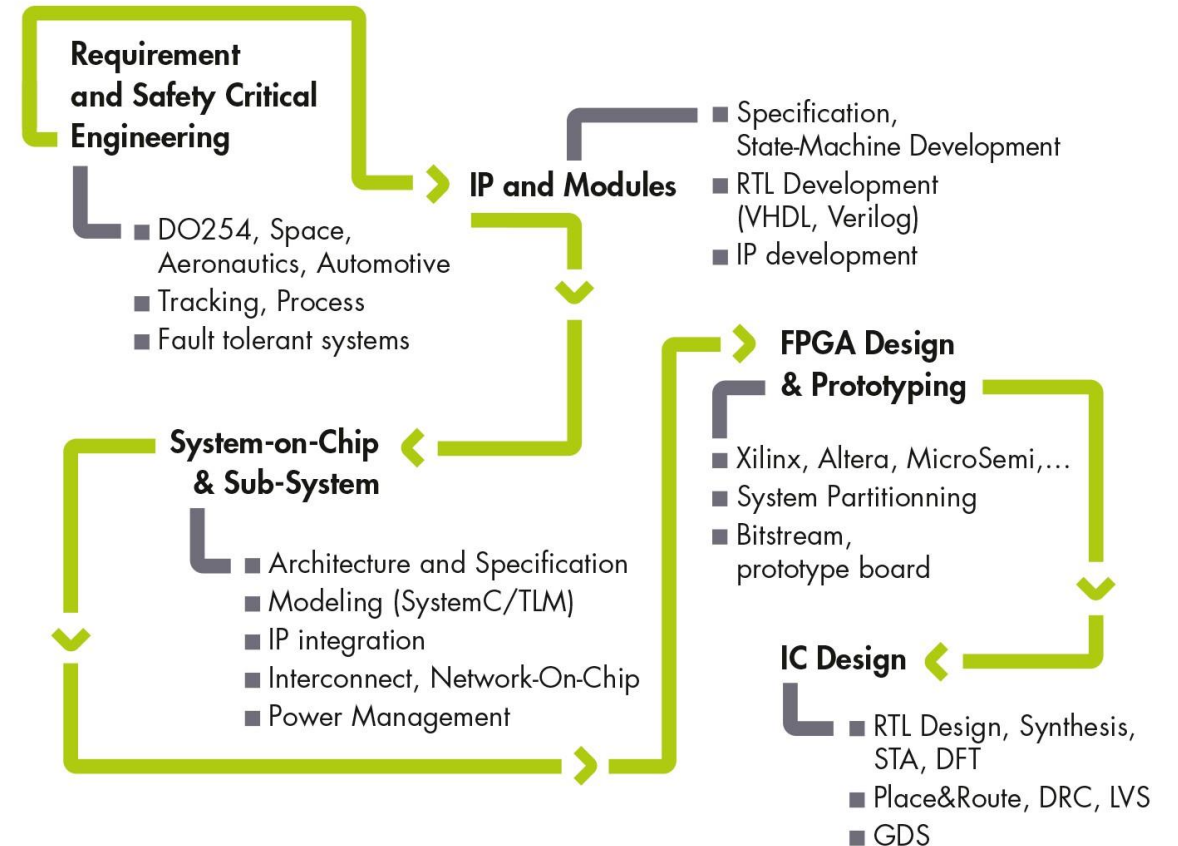
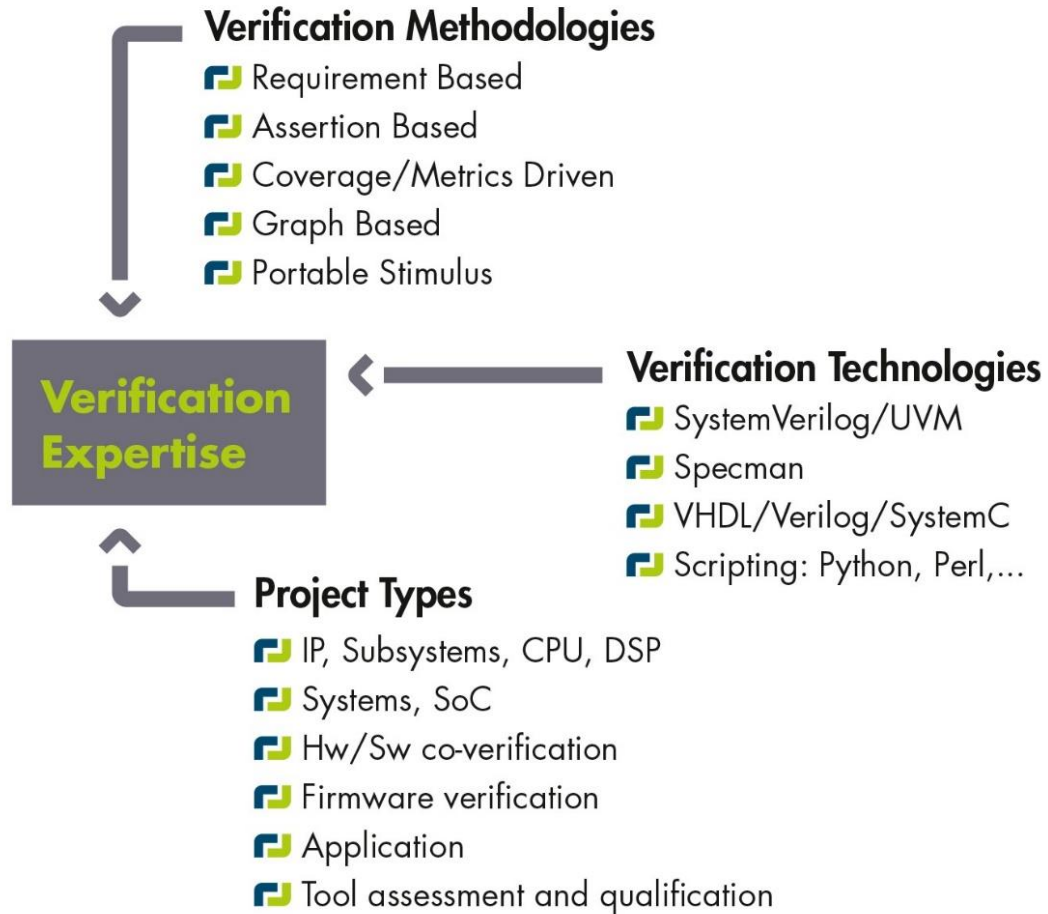


- Design & Verification Services
  - Consulting, Contracting
  - Competence Center Development
  - ICs & Embedded Systems

- Core and Crossover skills:
  - Classroom training
  - Independent Learning
  - Coaching/guidance

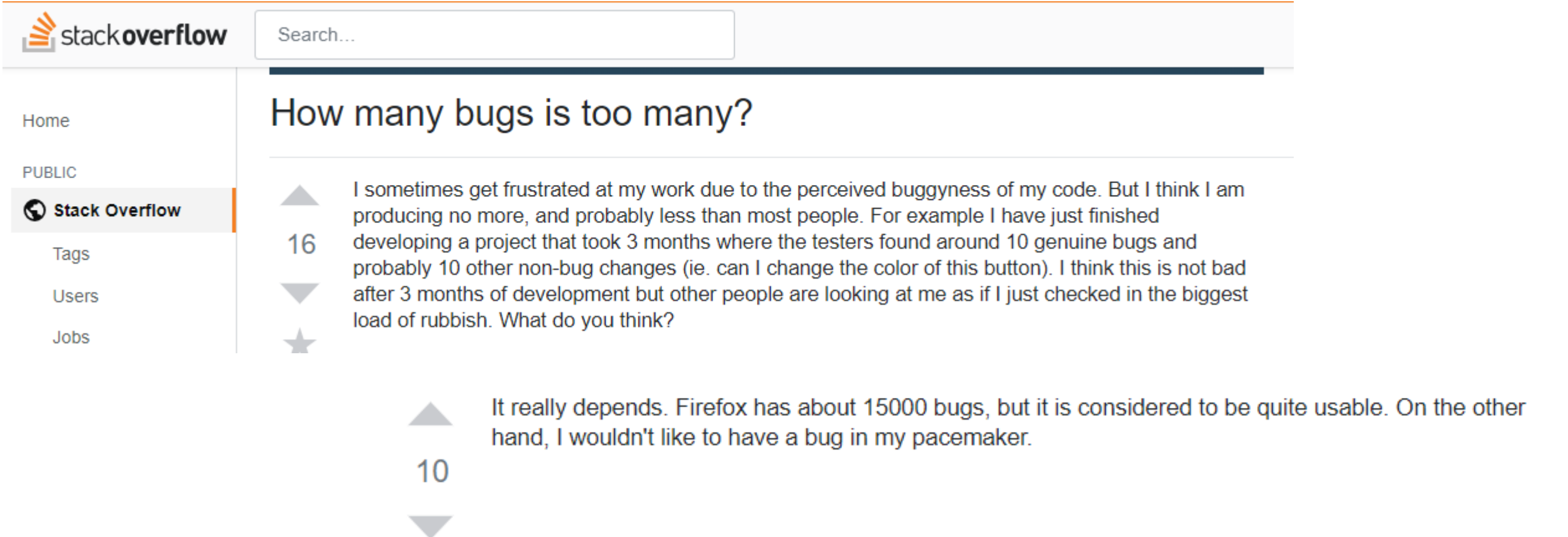


# Our Expertise





# How Many Bugs is too many ???



The screenshot shows the Stack Overflow interface. The left sidebar contains the 'Stack Overflow' logo and navigation links: Home, PUBLIC, Stack Overflow (selected), Tags, Users, and Jobs. The main content area displays a question titled 'How many bugs is too many?'. The question text is: 'I sometimes get frustrated at my work due to the perceived buggyness of my code. But I think I am producing no more, and probably less than most people. For example I have just finished developing a project that took 3 months where the testers found around 10 genuine bugs and probably 10 other non-bug changes (ie. can I change the color of this button). I think this is not bad after 3 months of development but other people are looking at me as if I just checked in the biggest load of rubbish. What do you think?'. The question has 16 votes, indicated by an upward arrow and the number '16'. Below the question are two answers. The first answer has 10 votes, indicated by an upward arrow and the number '10'. The second answer is partially visible.

stackoverflow

Search...

Home

PUBLIC

Stack Overflow

Tags

Users

Jobs

## How many bugs is too many?

▲ 16 ▲

I sometimes get frustrated at my work due to the perceived buggyness of my code. But I think I am producing no more, and probably less than most people. For example I have just finished developing a project that took 3 months where the testers found around 10 genuine bugs and probably 10 other non-bug changes (ie. can I change the color of this button). I think this is not bad after 3 months of development but other people are looking at me as if I just checked in the biggest load of rubbish. What do you think?

▼

▲ 10 ▲

It really depends. Firefox has about 15000 bugs, but it is considered to be quite usable. On the other hand, I wouldn't like to have a bug in my pacemaker.

Source: <https://stackoverflow.com/questions/625344/how-many-bugs-is-too-many>

# Why do we verify ?

Any Idea ?





# Why do we verify ?

 Any Idea ?



# Some Bugs

## Un bogue informatique avait contraint le Boeing 787 à être redémarré tous les 248 jours

Pour éviter une interruption totale du système électrique

Le 4 septembre 2018, par [Coriolan](#), Chroniqueur Actualités



De nos jours, toutes nos actions quotidiennes impliquent des programmeurs, du simple fait de passer un appel sur téléphone au pilotage des avions de ligne. C'est pourquoi on a vu pas mal de fois des gens tués dans des accidents de voiture à cause de bogues de logiciels alors que d'autres ont péri dans des crashes d'avions pour la même raison.

Parfois même, des projets de grande envergure sont condamnés à l'échec à cause d'un simple dysfonctionnement informatique lié au codage, à l'image du tristement célèbre vol 501 du lanceur européen Ariane 5 qui a eu lieu le 4 juin 1996 et qui s'est soldé par un échec. Ou plus récemment [la chute libre de l'atterrisseur Schiaparelli de l'ESA](#) (Agence spatiale européenne) sur le sol de Mars à cause d'un bogue informatique.

Vous l'aurez compris, même les grands projets coûtant parfois des milliards et des années de travail se trouvent voués à l'échec à cause de simples lignes de code. Mais qu'en est-il d'un avion de ligne moderne comme le Boeing

Actualité / Société

CONTRÔLE AÉRIEN

## Un avion disparaît des radars après un bug informatique

Par L'EXPRESS.fr avec AFP ,

publié le 13/07/2018 à 19:08



normal", a déclaré Stéphane Lesage. "Cela peut paraître beaucoup quatre kilomètres" mais "quand on vole à plus de 800 km/h, c'est quelques secondes" seulement, a-t-il expliqué, qualifiant cet événement de "vraiment grave". "Nous sommes passés proche d'une collision en vol", écrit le syndicat dans un communiqué.

# Cruise Control Bug !

Publié le 12/02/2013 à 14:32, Mis à jour le 12/02/2013 à 14:33

## Son régulateur de vitesse bloqué, il franchit 3 péages à 200 km/h

### Faits divers

Un automobiliste a eu une très grosse frayeur samedi soir sur l'A16, à hauteur d'Abbeville, dans la Somme. Un bug du régulateur de vitesse est survenu, obligeant le véhicule à rouler à 200 km/h pendant près de 170 km.

Le conducteur d'une Renault Laguna a été pris au piège de son propre véhicule. A cause d'une défaillance du régulateur de vitesse, plus il tente de freiner, plus la voiture accélère. Lorsqu'il s'aperçoit du problème, l'homme roule déjà à 160 km/h. Très vite, le véhicule atteint une vitesse de croisière de 200 km/h.

Pendant son long périple, l'homme parvint à alerter les secours. Les gendarmes viennent alors escorter la voiture folle sur l'autoroute et aident l'homme à franchir trois péages. Ce n'est qu'une fois la frontière belge passée que le conducteur a pu arrêter le véhicule, dans un fossé.

L'homme est sain et sauf mais a fait deux crises d'épilepsie avant d'être hospitalisé. Une enquête est ouverte pour déterminer l'origine du dysfonctionnement. Ce modèle de voiture, sans pédales d'accélérateur, ni freins, est adapté à la conduite des personnes handicapées.

# Definition of a Bug

A (software) **bug** is the common term used to describe an error, flaw, mistake, failure, or fault in a computer program or system that produces an incorrect or unexpected result, or causes it to behave in unintended ways.

Source: [http://en.wikipedia.org/wiki/Software\\_bug](http://en.wikipedia.org/wiki/Software_bug)

Error	: Wrong judgement	←	
Mistake	: Error caused by insufficient knowledge	←	Root Cause = Human
Flaw	: Imperfection	← - - - - -	
Failure	: Omission of expected action	← ······	
Fault	: Unsatisfactory feature	← ······	Root Cause = Physics





# Bugs Happen

designlines SoC

## Blog

### Bugs Happen

Dr. Lianfeng Yang, ProPlus Design Solutions

1/14/2016 03:00 PM EST

0 comments post a comment

Like 4

Tweet

in Share

11

G+



Small problems that could be ignored in the past are now becoming critical for advanced designs using small nodes as cost and risk increase significantly.

The world of chip design and manufacturing is no different than any other: Bugs happen.

[http://www.eetimes.com/author.asp?section\\_id=36&doc\\_id=1328686](http://www.eetimes.com/author.asp?section_id=36&doc_id=1328686)



# ◆ This is because of LOC

<https://www.mayerdan.com/ruby/2012/11/11/bugs-per-line-of-code-ratio>

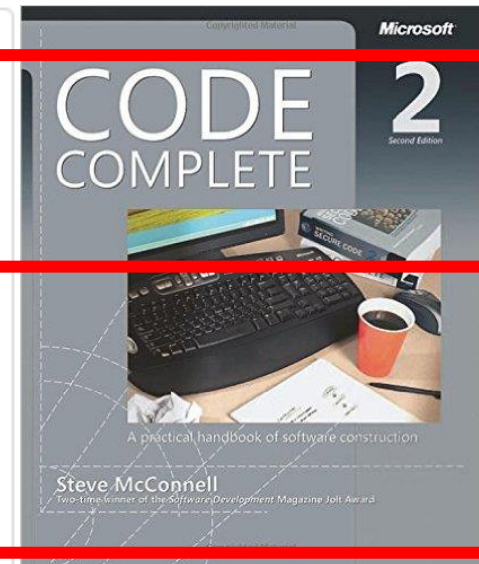
## ratio of bugs per line of code 11 November 2012

The more development I do the more I feel like increased Lines Of Code (LOC), nearly always results in increased bugs.

(a) **Industry Average:** "about 15 - 50 errors per 1000 lines of delivered code." He further says this is usually representative of code that has some level of structured programming behind it, but probably includes a mix of coding techniques.

(b) **Microsoft Applications:** "about 10 - 20 defects per 1000 lines of code during in-house testing, and 0.5 defect per KLOC (KLOC IS CALLED AS 1000 lines of code) in released product (Moore 1992)." He attributes this to a combination of code-reading techniques and independent testing (discussed further in another chapter of his book).

(c) "Harlan Mills pioneered 'cleanroom development', a technique that has been able to achieve rates as low as 3 defects per 1000 lines of code during in-house testing and 0.1 defect per 1000 lines of code in released product (Cobb and Mills 1990). A few projects - for example, the space-shuttle software - have achieved a level of 0 defects in 500,000 lines of code using a system of format development methods, peer reviews, and statistical testing."



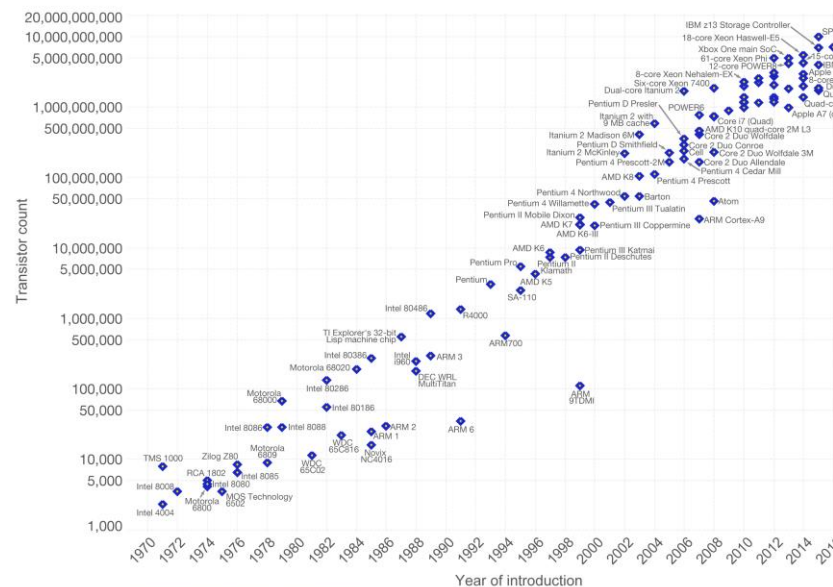
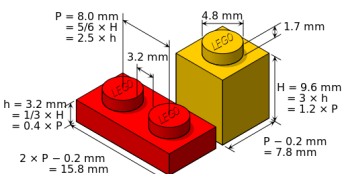


# Moore's Law

## Moore's Law – The number of transistors on integrated circuit chips (1971-2016)

Moore's law describes the empirical regularity that the number of transistors on integrated circuits doubles approximately every two years. This advancement is important as other aspects of technological progress – such as processing speed or the price of electronic products – are strongly linked to Moore's law.

Our World in Data

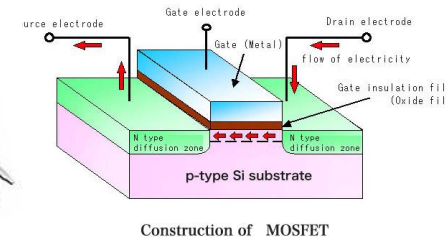


Data source: Wikipedia ([https://en.wikipedia.org/wiki/Transistor\\_count](https://en.wikipedia.org/wiki/Transistor_count))

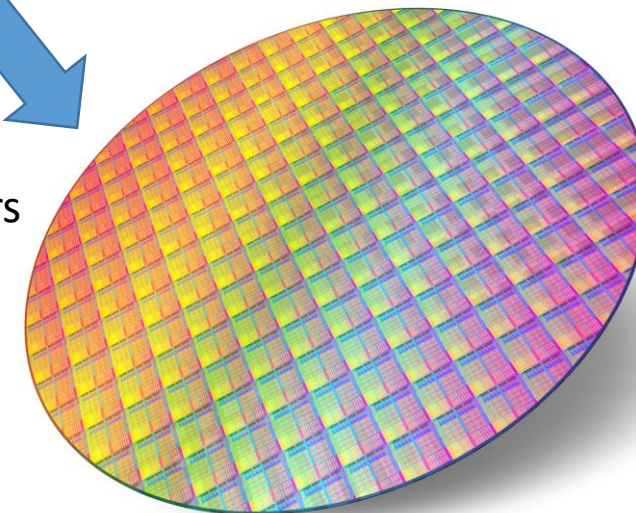
The data visualization is available at OurWorldinData.org. There you find more visualizations and research on this topic.

Licensed under CC-BY-SA by the author Max Roser.

Engineering Connected Intelligence: A Socio-Technical Perspective - Scientific Figure on ResearchGate. Available from: [https://www.researchgate.net/Moores-Law-Number-of-transistors-on-integrated-circuit-chips\\_fig1\\_323512822](https://www.researchgate.net/Moores-Law-Number-of-transistors-on-integrated-circuit-chips_fig1_323512822) [accessed 19 Nov, 2018]



$18 \times 10^9$  transistors



[https://en.wikipedia.org/wiki/Moore%27s\\_law](https://en.wikipedia.org/wiki/Moore%27s_law)

As of 2017, the commercially available processor possessing the highest number of transistors is the 48 core [Centriq](#) with over 18 billion transistors. <sup>[119]</sup>

2600 pieces



# The bugs' laws

## The 4<sup>th</sup> Law thermodynamics : Murphy's law:

- If something can go wrong, it will sooner or later
- Hopefully, cats always land on their feet

## Mother nature's laws:

- Bugs represent the most part of insects
- Erik J. van Nieuwerkerken has made a scientific estimate that there are 1,017,018 species of insects in the world.
- At any time, it is estimated that there are some 10 quintillion (10,000,000,000,000,000,000) individual insects alive.  
(<https://www.si.edu/spotlight/buginfo/bugnos> )

## The Law Of Simplicity / Occam's razor

- We can keep simple things simple
- but we can hardly keep complex things simple.

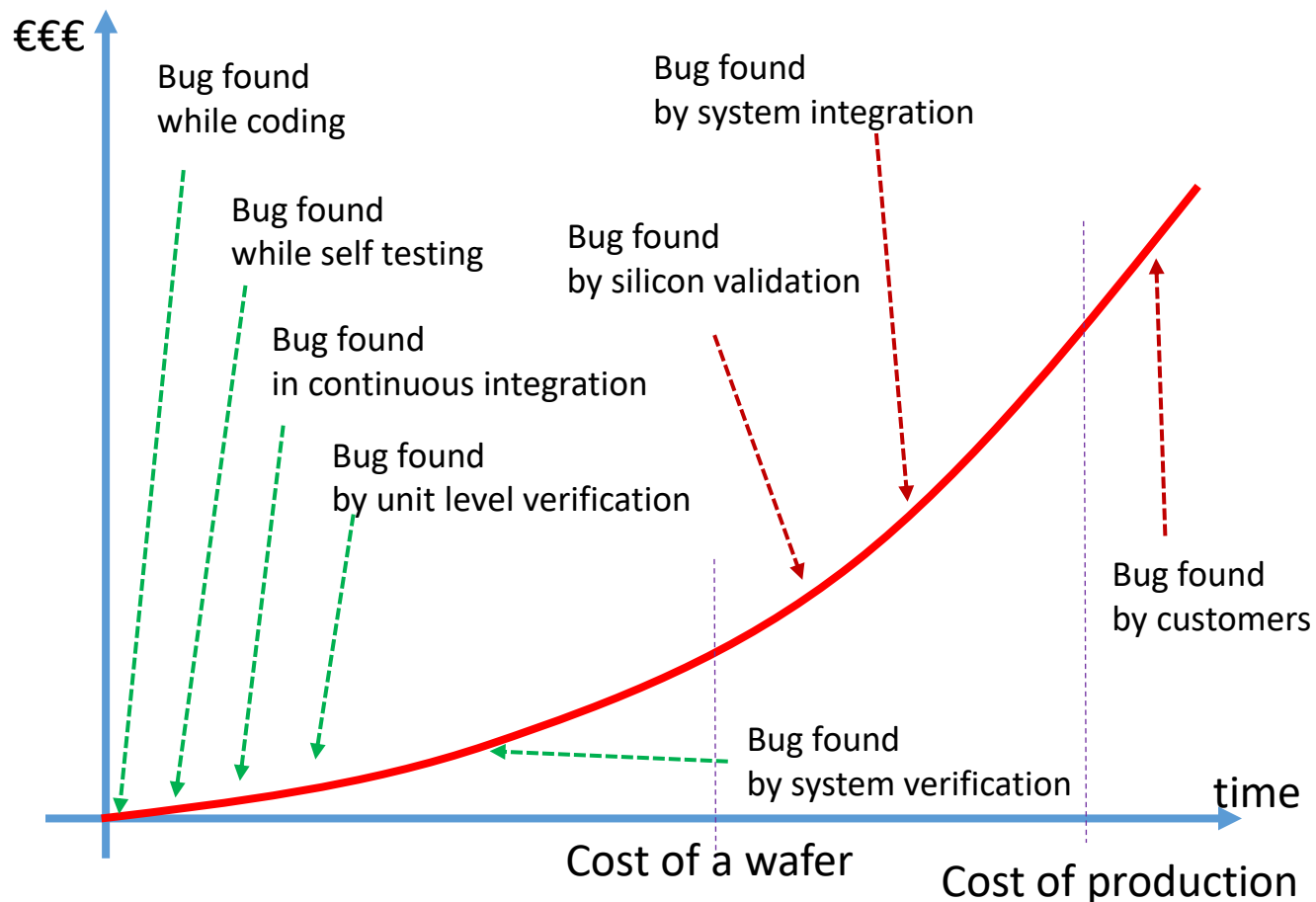
## Comparison

- Human brain has only 100 billions of neurons (mine has obviously less)
- As of 2015, the highest transistor count in a commercially available chip is over 10 billions transistors, in a 32-core PARC M7

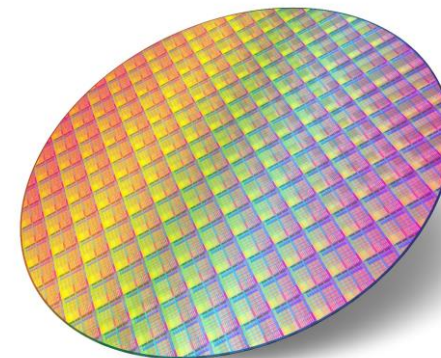


# Cost of a bug

The later a bug is found, the higher cost it has



Cost of a wafer



The cost to set up a [45 nm](http://en.wikipedia.org/wiki/Photomask) process [mask shop](http://en.wikipedia.org/wiki/Photomask) is \$200–500 million  
Source: <http://en.wikipedia.org/wiki/Photomask>

Average IC design cost for a 28nm device is about \$30 million + 60% when the mask cost is included.  
Source: <http://semiengineering.com/finfet-rollout-slower-than-expected/>

The minimum cost for a design at 10nm will be \$150 million...  
Source: <http://www.simmtester.com/page/news/shownews.asp?num=17243>

Adapted from: <http://www.agilemodeling.com/essays/costOfChange.htm>

# What is verification then ?





# Verification: definition

 Wikipedia:

- **Functional verification**, in [electronic design automation](#), is the task of verifying that the [logic design](#) conforms to specification. In everyday terms, functional verification attempts to answer the question "**Does this proposed design do what is intended?**"

 So, what is intended ?

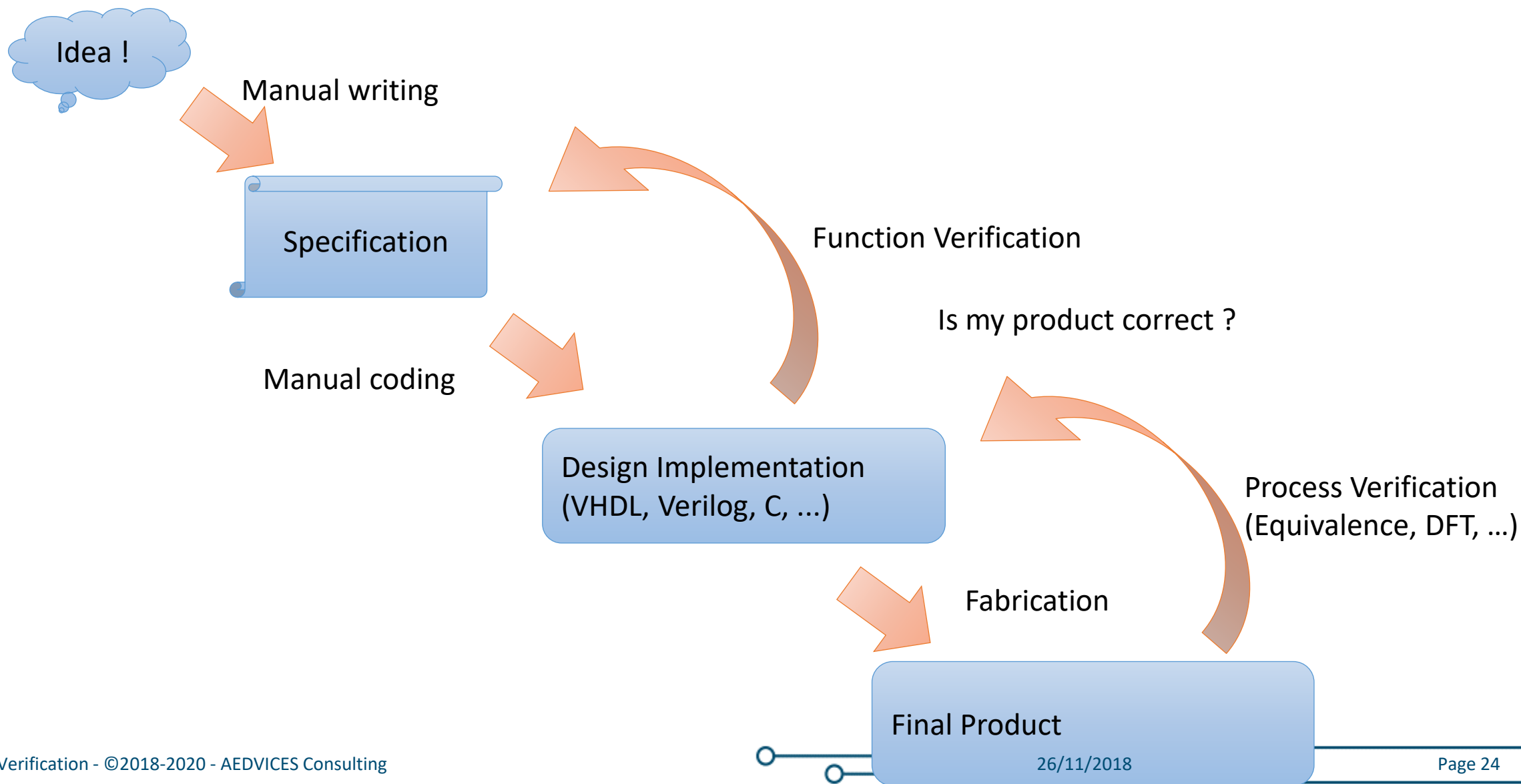
- **All bugs found and fixed**  
→ **No Bug, Zero Bug left**                      How do we prove the absence of bugs ?
- **Safe and/or Secure**  
→ **working so that life integrity and/or confidential data is not compromised**
- **Right first time**  
→ **working well enough so that there is no re-design (and save money)**
- **Silicon Success**  
→ **working well enough so that we can use it and sell it (and make money)**

# Verification vs Criticity

## Verification adapts to:

- Time to Market versus Quality
  - If a innovative product is late, there will be 0 revenue. Competition will make money, we will not.
  - If a innovative product is first on the market, it will win the market, **even if there are a few bugs.**  
→ Money
- Safety Critical Applications versus Entertainment Applications
  - “I don’t care if 1 pixel is slightly pink instead of red in this HD display, but I care that my video decoder does not restart in middle of a film”.
  - “A device that does not work is safe. The plane will not take off.”
  - “A device that does not work is unsafe. The fire alarm will not alert people.”  
→ Human Lives → Prejudice → Money
- What is the cost of having a major bug left in my design ?
  - Cost of a fix → Money
  - Reputation, loss of market shares → Money
  - Prejudice → Money
  - Human Lives → Prejudice → Money

# Verification of the Intent



# What do we verify ?





# What do we verify ?

Somehow, all kinds of designs

## Modules, IPs, Blocks

- Memory controller, UART, Cache, DMA
- FIFOs, Bus Bridges, Routers, Filters, ...
- Bus Interconnect, Network on Chip

...

## Sub-Systems

- Data Processing Hardware Accelerator (Audio, Video, ...)
- Power Management Unit

...

## Microprocessors (uP, uC, DSP, ...)

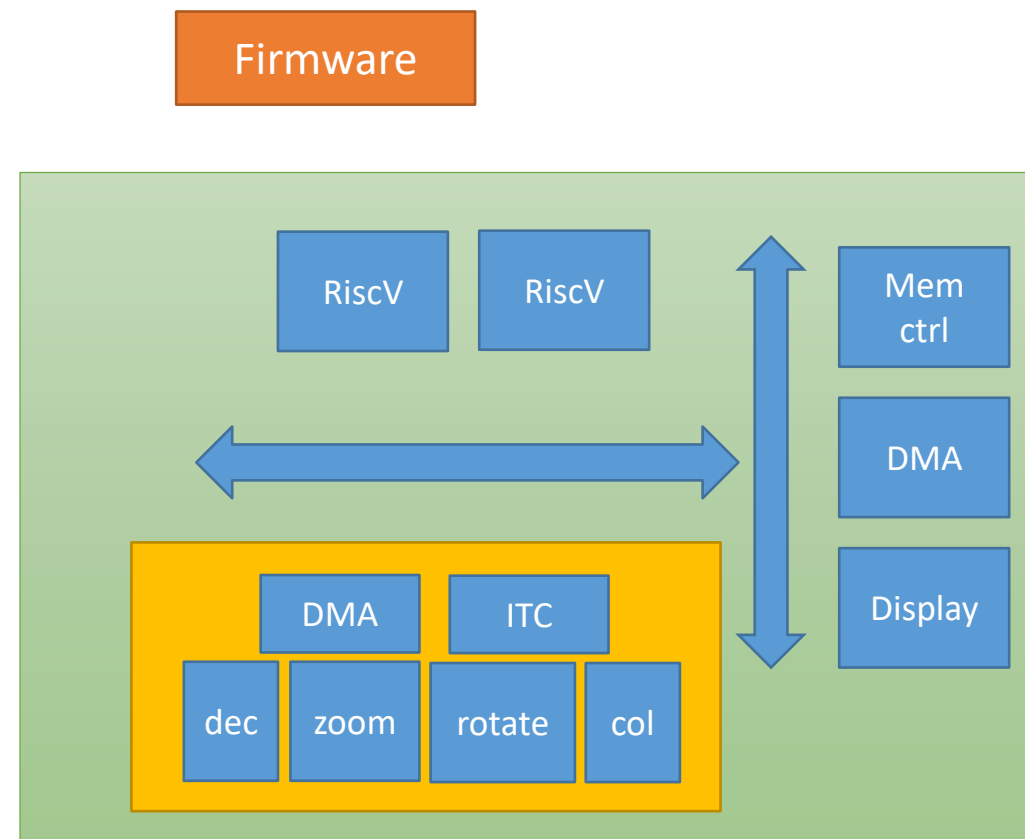
## System-On-a-Chip ( SoC )

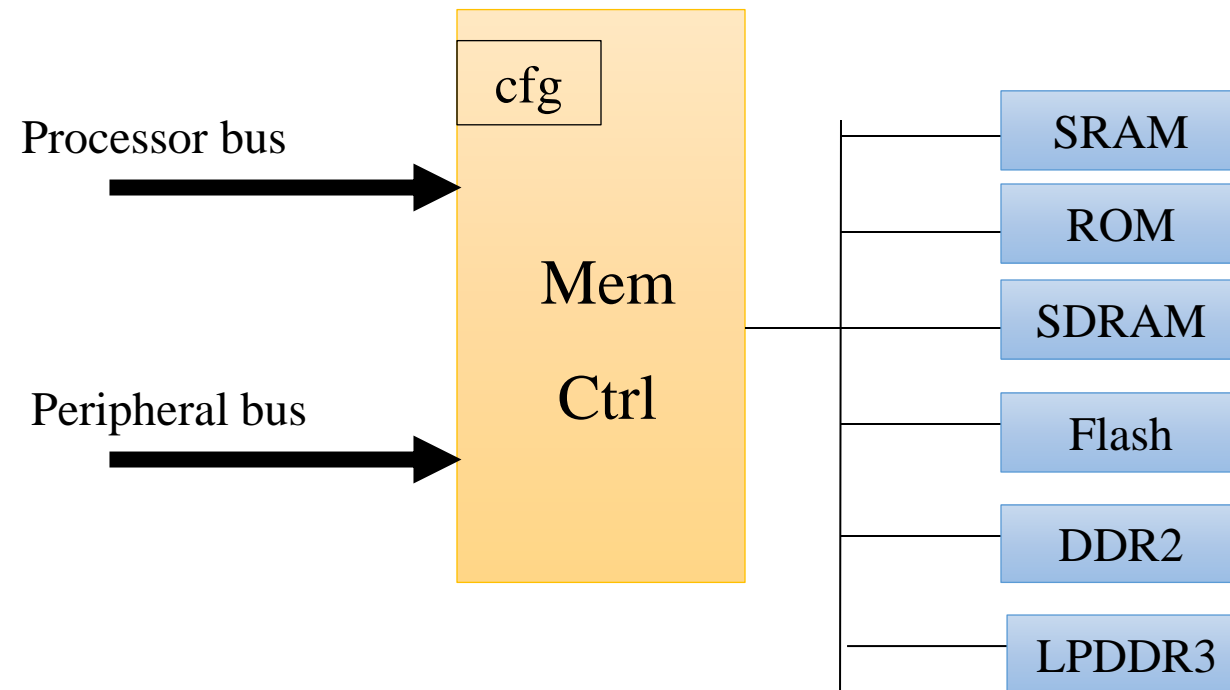
- Cell Phone
- Set-Top Boxes, TV Chips, ...
- Car on-board system

...

## Full systems

- Hardware with its embedded software

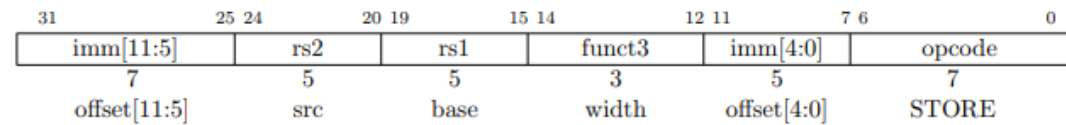
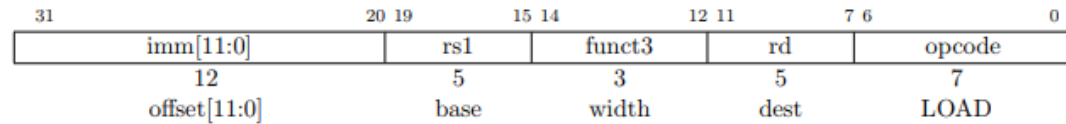




## Ex 2: a simple microprocessor

### 4.3 Load and Store Instructions

RV64I extends the address space to 64 bits. The execution environment will define what portions of the address space are legal to access.

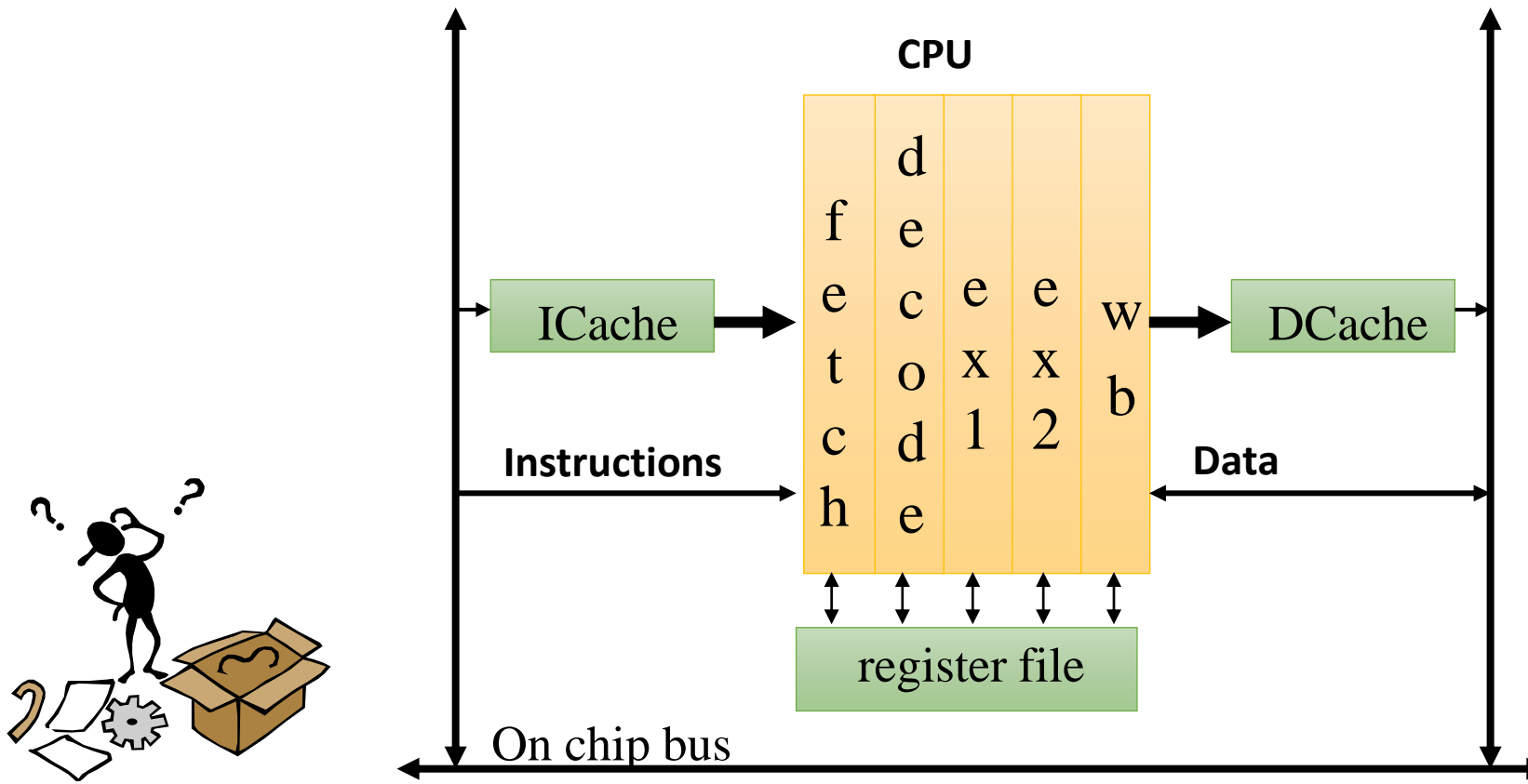


The LD instruction loads a 64-bit value from memory into register *rd* for RV64I.

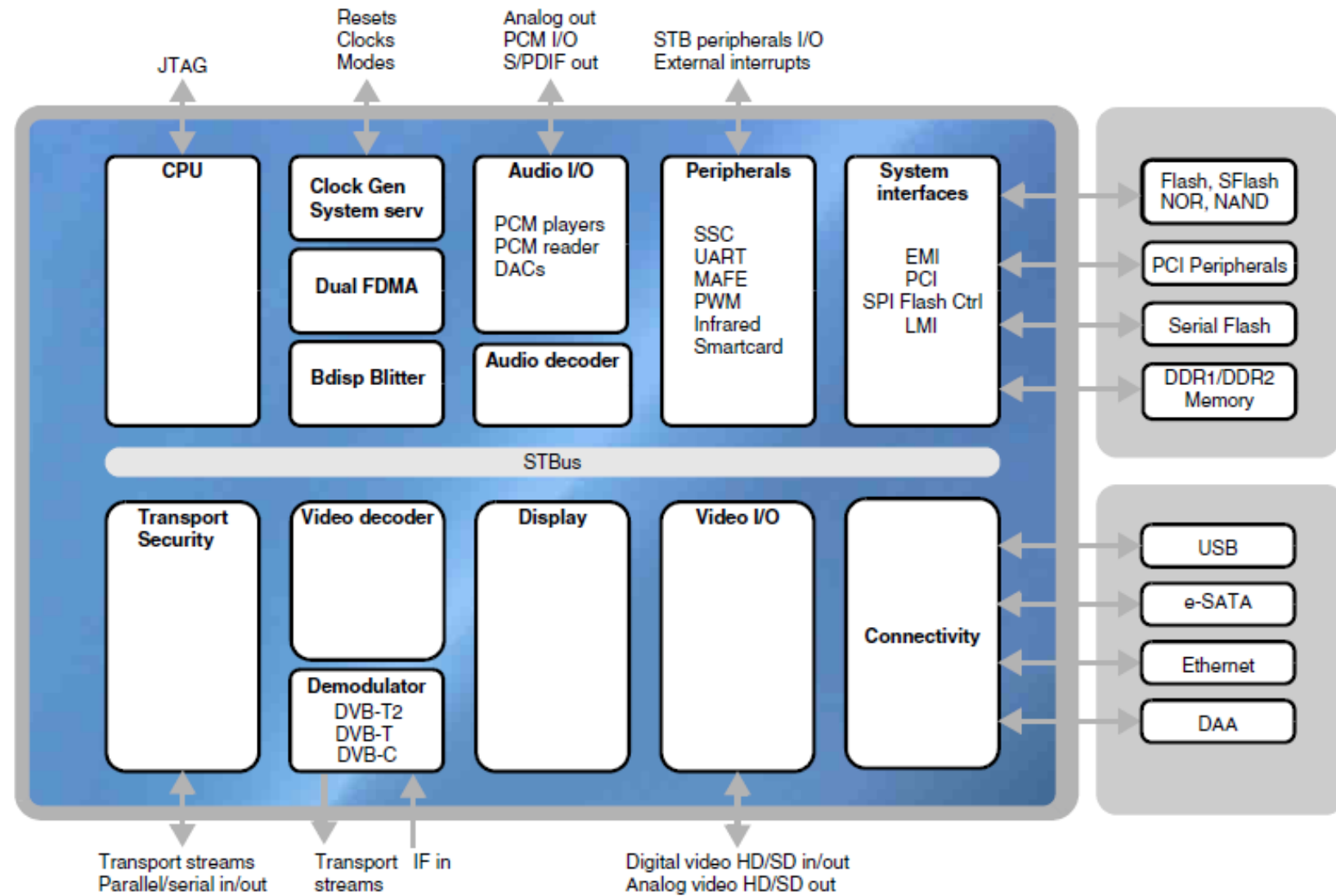
The LW instruction loads a 32-bit value from memory and sign-extends this to 64 bits before storing it in register *rd* for RV64I. The LWU instruction, on the other hand, zero-extends the 32-bit value



## Ex 2: a simple microprocessor micro-architecture



## Ex 3: Application Processor for Multimedia Systems



# Software versus Hardware

User

HMI / GUI

Applications

Applications

Applications

Services / Hypervisor

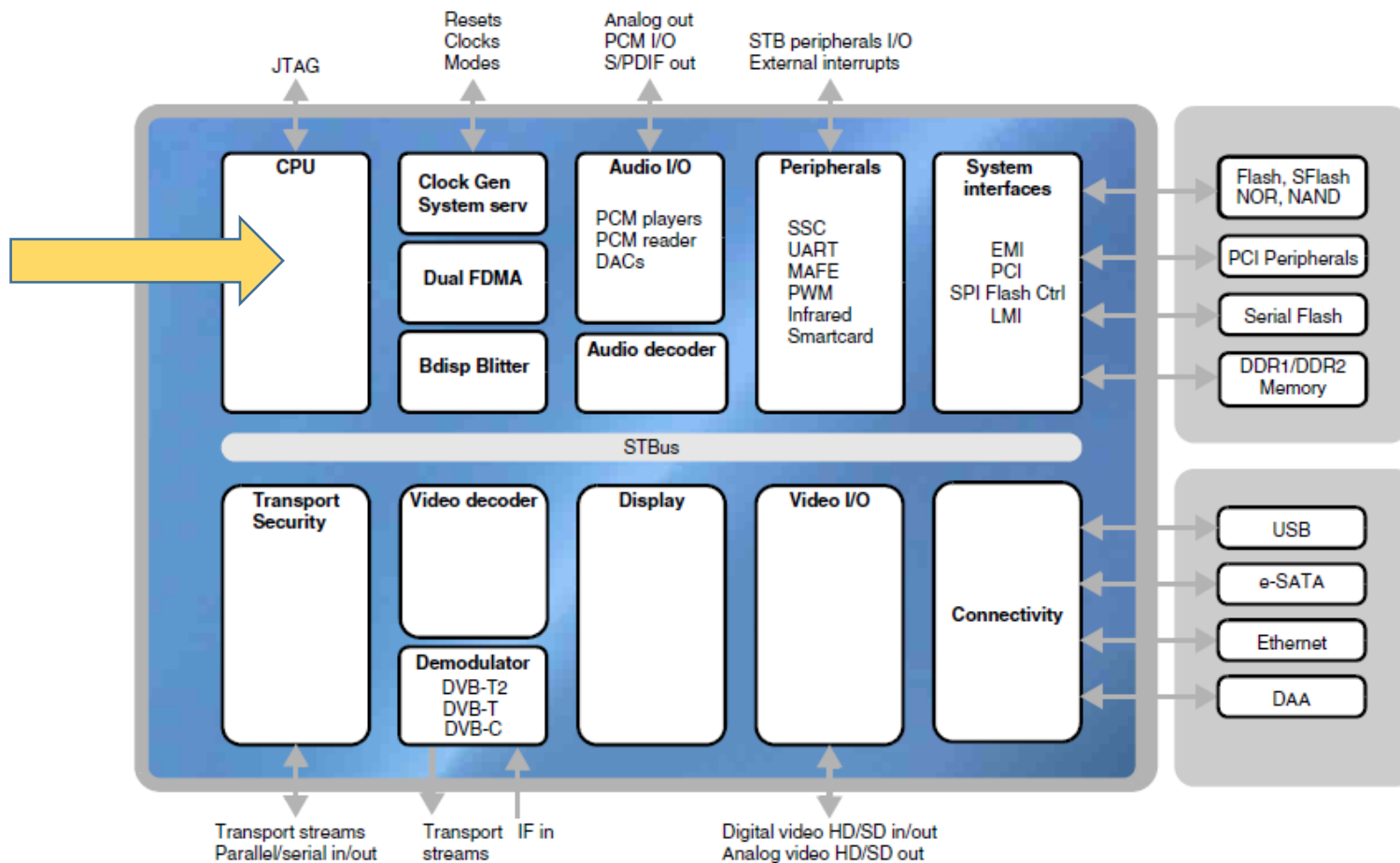
Drivers

BIOS

HAL

pure SW

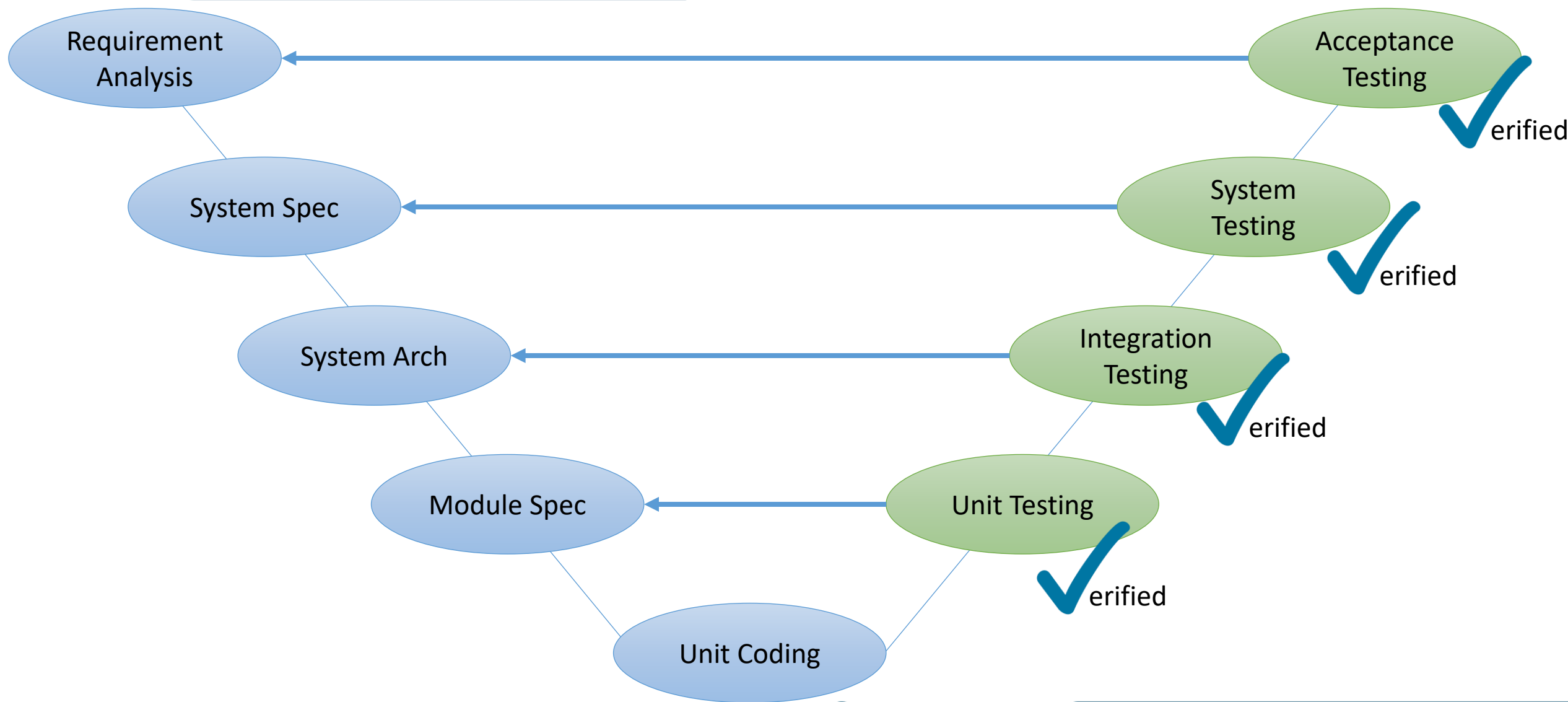
SW with HW knowledge



# When do we verify ?

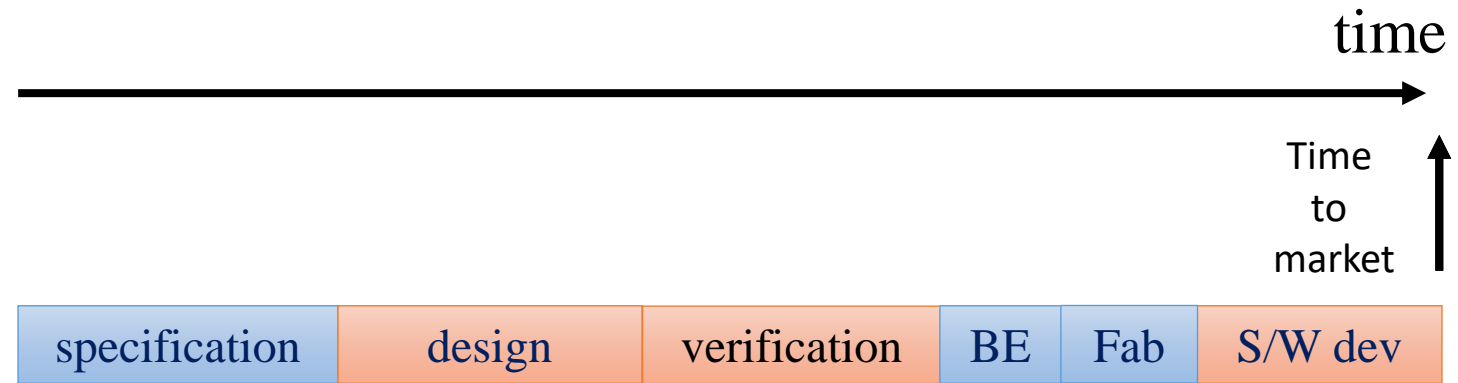


# The v-Model

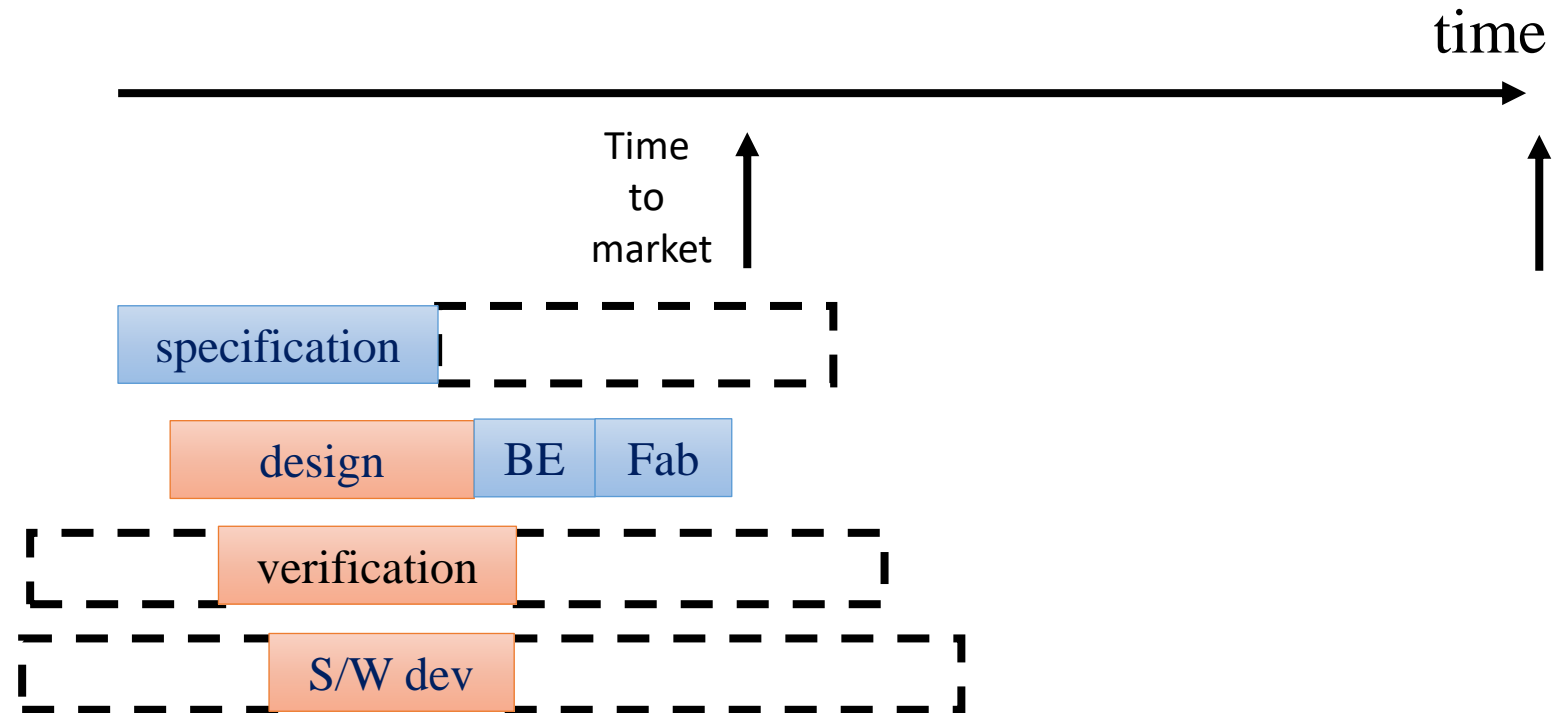




# Project Planning Resolved



# Project Planning Extended



# Time-To-Market = Planet Alignment



April 27, 2018

## NASA Sets Sights on May 5 Launch of InSight to Mars

NASA's next mission to Mars, InSight, is scheduled to launch Saturday, May 5, on a first-ever mission to study the heart of the Red Planet.



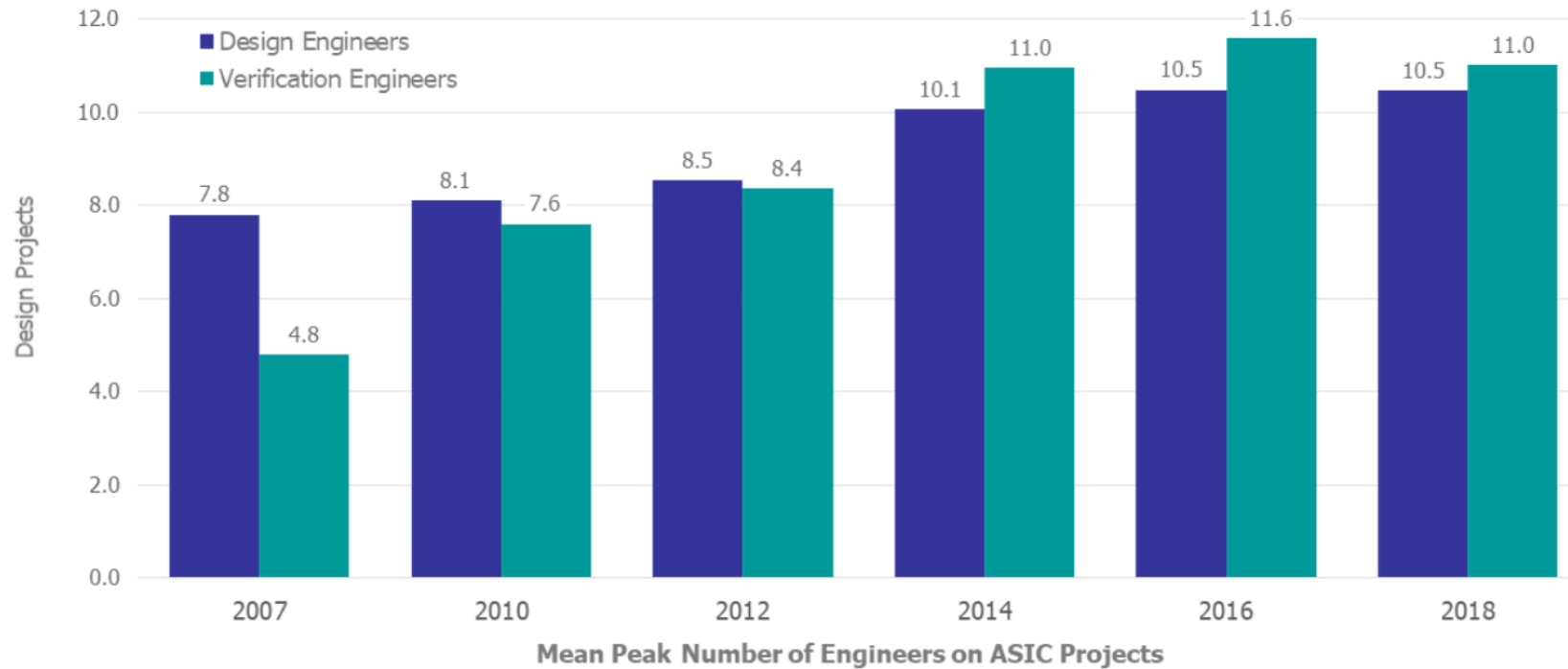
<https://www.solarsystemscope.com/>

# Who Verifies ?



# Verification Engineers vs Design Engineers

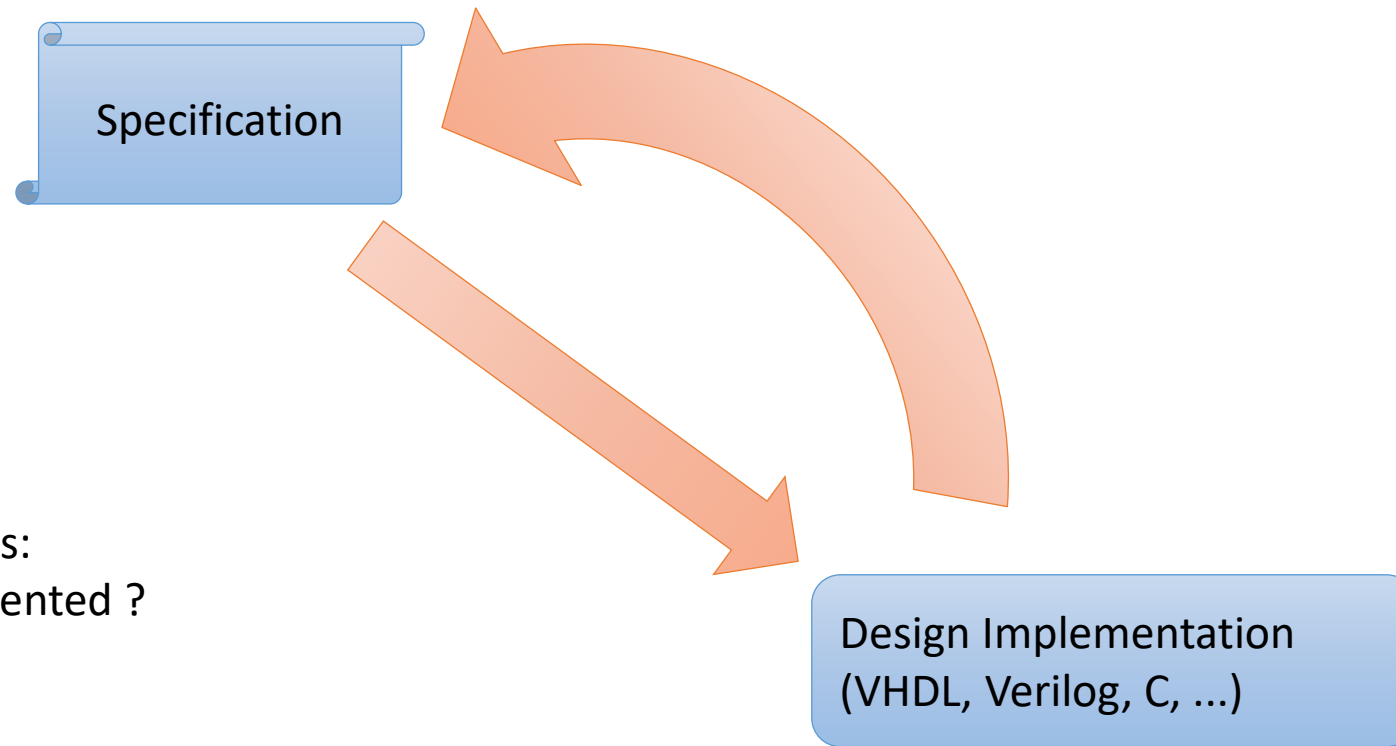
## ASIC: Mean Peak Number of Engineers on a Project



Source: Wilson Research Group and Mentor, A Siemens Business, 2018 Functional Verification Study

# ◆ Designers Verify their designs ?

Functional Verification

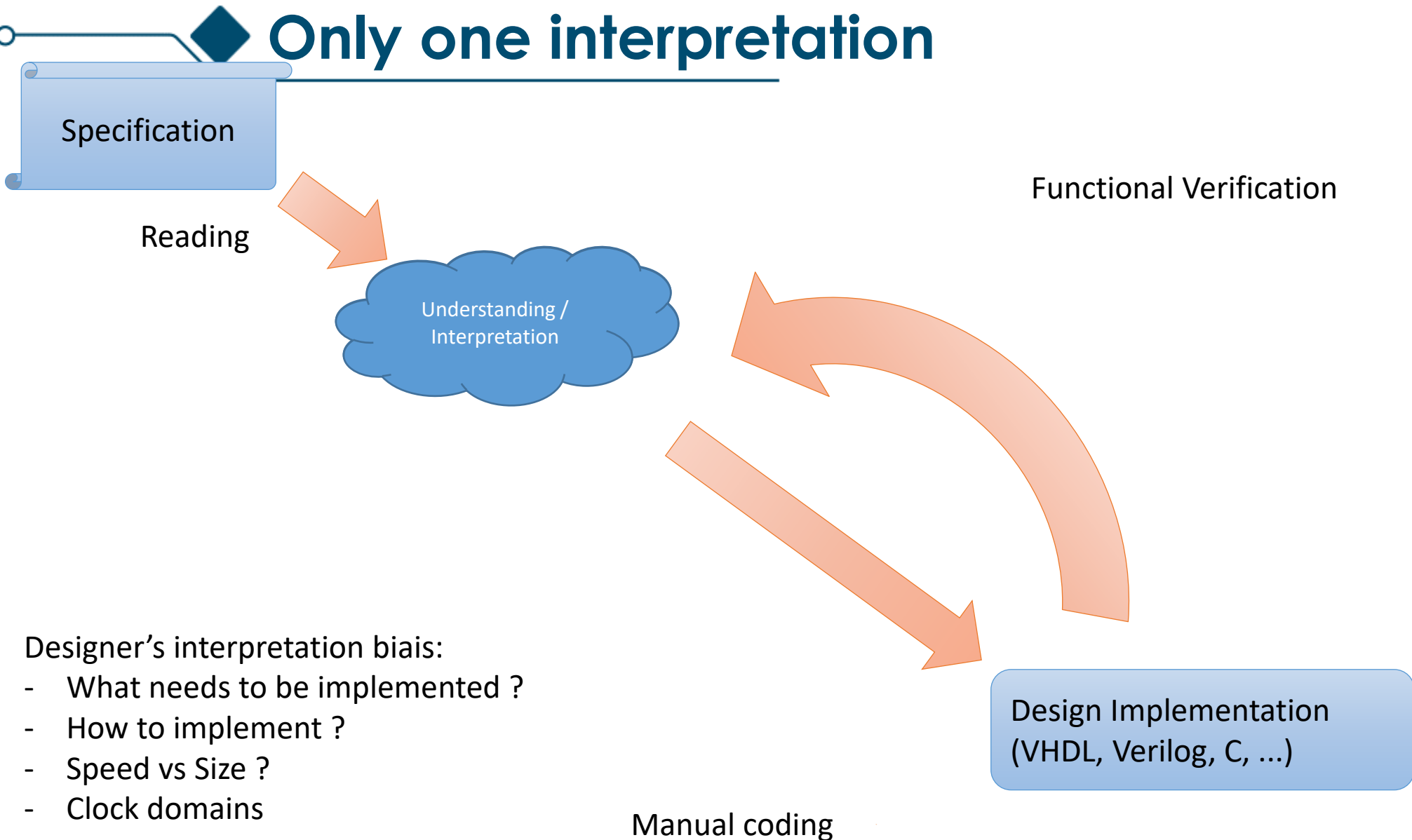


Designer's interpretation biais:

- What needs to be implemented ?
- How to implement ?
- Speed vs Size ?
- Clock domains

Manual coding

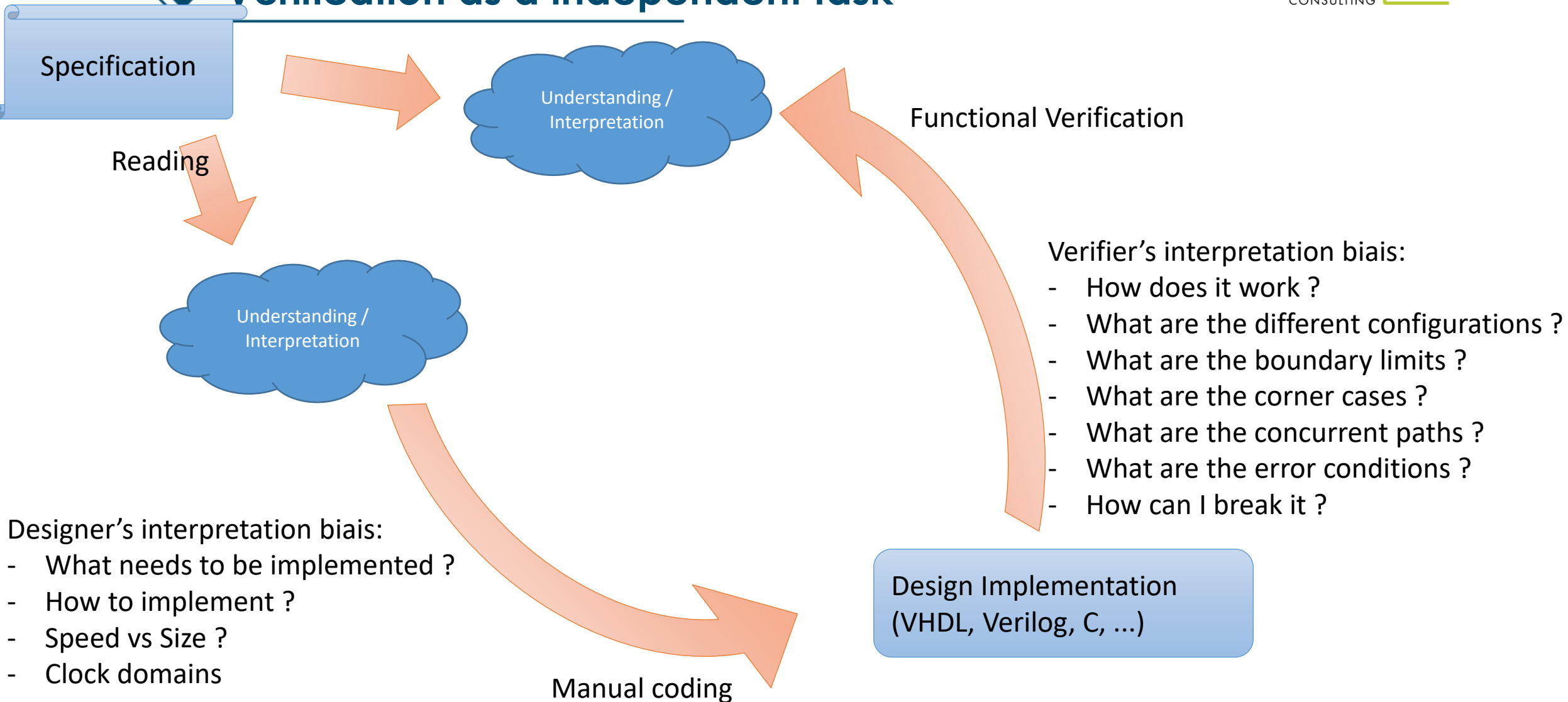
# Common mode: Only one interpretation



Designer's interpretation biases:

- What needs to be implemented ?
- How to implement ?
- Speed vs Size ?
- Clock domains

# Avoiding the common mode: Verification as a independent task





# The Verification Engineer Profile

Human Characteristics	Details
Curious	A severe need to understand more than what is said
Player	A severe willingness to find a way to break things
Perfectionist	Either all or nothing. A task is never complete, especially a test.
Rigorous	Never approximates and checks any point
Absurd Thinking	Reason by the absurdity, but demonstrate it is not absurd.
Reactant	What if I do the exactly opposite of what is said ?
Contradictory	Ability to knowingly contradict his/her own thinking with both arguments and counter arguments
Pragmatic	Cannot be perfect. So be pragmatic about what you do.
Distrustful	Never trust anyone, especially a designer
Human	Makes mistakes !

Technical Skills	Details
Hardware	Know what is a signal, a protocol
SoC	Know what is a SoC, a CPU Understand SoC micro-architectures
Firmware	Understand Low Level Software. What is a stack, what is a boot sequence
Software	Object Oriented Programming Software Architectures
Other Software paradigms	Constrained Programming Aspected Oriented Programming
Know-How	C, C++, Python, Perl, VHDL, Verilog, SystemVerilog, SystemC, TLM, UML, ... ... ...
Human	Makes mistakes !

# Verification Concerns

Avoiding the false positive as much as possible:

- Stimulus: Tests should be capable to exercise the functionalities
- Coverage: Tests should actually cover the functionalities
- Checkers: Tests should reports errors when things go wrong.

Think twice before writing:

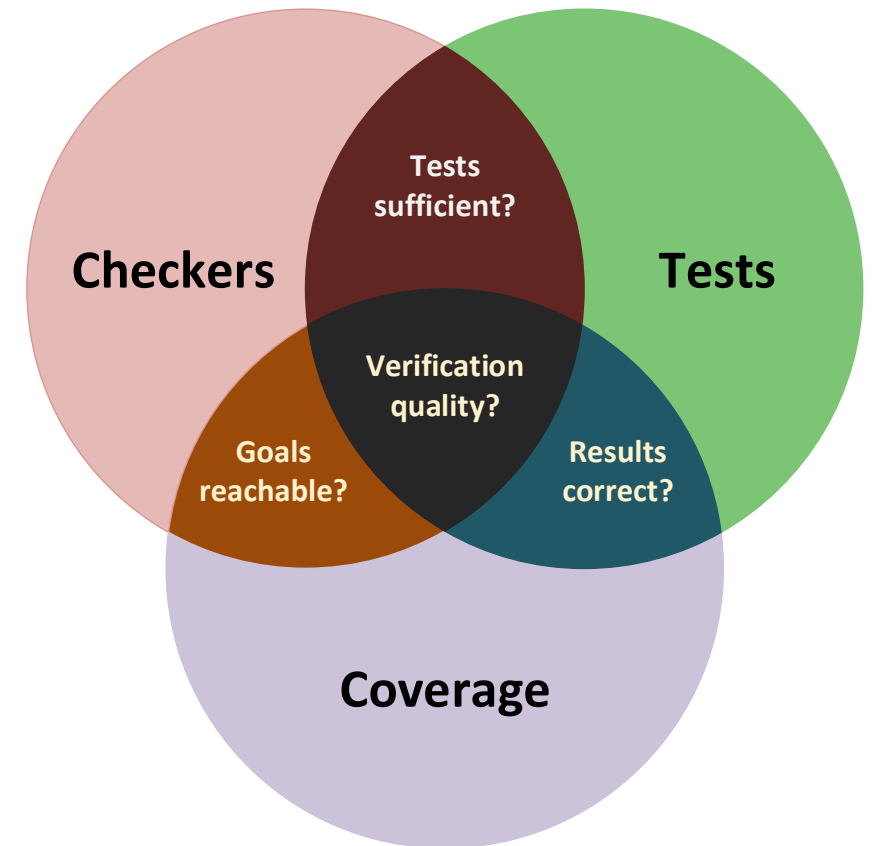
```
printf("PASSED")
```

# Verification Concerns

## How to avoid the false positive ?

- Did I test the design in all the interesting conditions ?  
→ Coverage: Measure the quality of the tests
- Does my design do what it is expected to do ?  
→ Checks: Verify actual vs expected behavior

Verification \ Design	<i>No-bug</i>	<i>Bug</i>
<i>Test fails</i>	False Negative (false alarm)	True Negative
<i>Test is okay</i>	True Positive	False positive (false promise)



# How do we Verify ?



# How do we Verify ?

## Simulation

- Requires **testbenches** to inject stimulus and check expected results
- High level of debug capability
- VHDL, Verilog, SystemVerilog, C, C++, ...

## Emulation

- Requires Synthesizable testbench
- Some debug capability
- VHDL, Verilog ( SCE-MI interfaces to simulated bench if needed )

## Prototyping

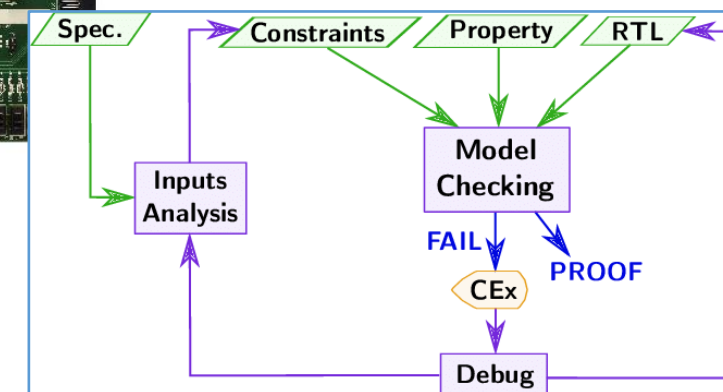
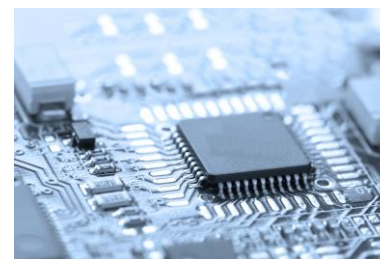
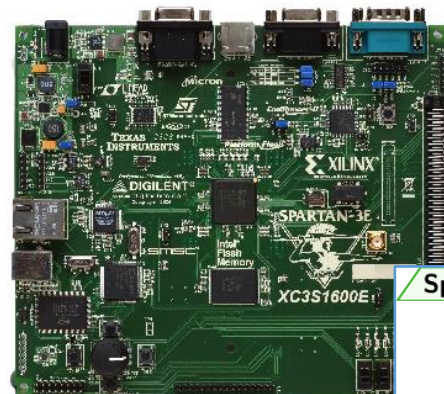
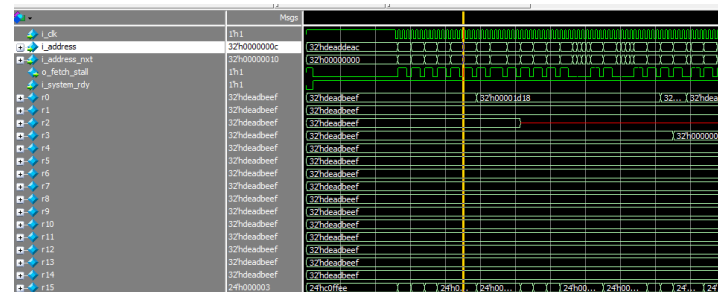
- Boards + FPGA : allow faster execution and runs of more realistic scenarios
- Limited debug capability

## Formal/Static verification – Property Checking

- Assertions/Properties are proven via formal/mathematical algorithms
- SVA, PSL, OVL, ...

## On Silicon Validation / Physical Testing

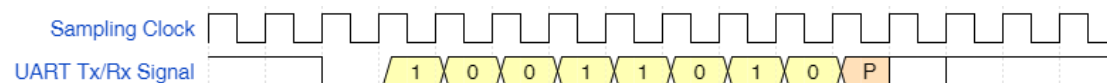
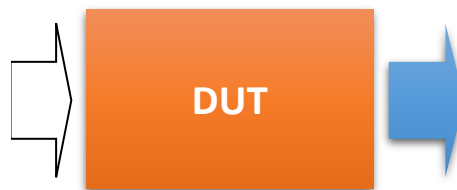
- *Product Test / Engineering is another topic*
- *Silicon validation concentrates on making sure the features are working ( and is not looking for bugs )*
- *Embedded bare-metal C, Free-RTOS, Linux, ...*
- *Labview*





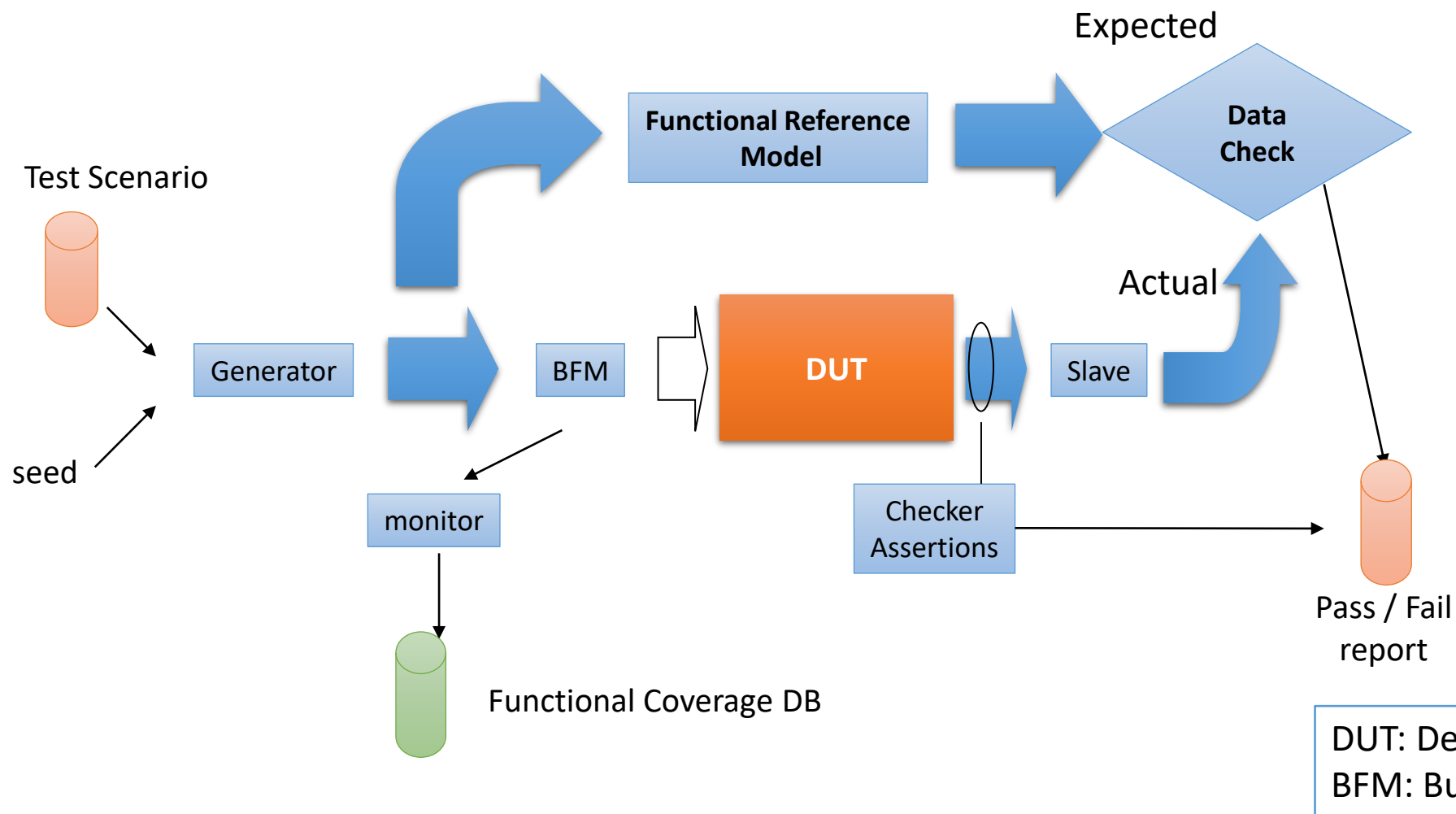
# Typical Verification Environment

```
74 // drive_transfer ( wishbone_transfer )
75 //-----
76 // Called by main sequencer
77 // Converts wishbone_transfer to actual signal values and events.
78 task wishbone_master_driver::drive_transfer(wishbone_transfer trans);
79
80   bit[`WISHBONE_DATA_MAX_WIDTH/8 -1:0] sel;
81   bit[`WISHBONE_DATA_MAX_WIDTH -1:0] shift;
82   bit[`WISHBONE_DATA_MAX_WIDTH -1:0] mask;
83   bit[`WISHBONE_DATA_MAX_WIDTH -1:0] le_data; // little endian data
84   bit[`WISHBONE_DATA_MAX_WIDTH -1:0] vif_data;
85
86
87   repeat ( trans.transmit_delay )
88   ..@(posedge vif.clk);
89
90   void'(begin_tr(trans, "wishbone_master_driver", "UVM Debug",
91   .."wishbone_master_driver driver transaction from get_and_drive"));
92
93   vif.cyc <= 1;
94   repeat (trans.nr_data)
95   begin
96   ...// Wait States
97   repeat ( trans.wait_states )
98   ..@(posedge vif.clk);
99   ...// Drive Address
100
101   vif.addr <= trans.addr & ~p_agent.bus_addr_mask;
102
103   ...// Shift and Mask related to the address alignemnt
104   shift = (trans.addr & p_agent.bus_addr_mask)*8;
105   mask = 0;
106   repeat (trans.width * 8)
107   mask = (mask << 1) + 1;
108
109   ...// Drive Write Data
110   if ( trans.direction == WB_WRITE ) begin
111   le_data = ((trans.data & mask) << shift) & ~p_agent.bus_data_mask;
112
113
114   if ( p_agent.endianness == WB_LITTLE_ENDIAN )
115   vif_data = le_data;
116   else
117   for (int ii=0;ii<p_agent.data_width/8;ii+=1)
118   for (int jj=0;jj<8;jj+=1)
119   vif_data[ii*8+jj] = le_data[p_agent.data_width-ii*8-8+jj];
120
121   vif.data_wr <= vif_data;
122   vif.we <= 1;
123
```



DUT: Design Under Test  
BFM: Bus Functional Model

# Typical Verification Environment



# Hw/Sw Co-Simulations

## Simulation

- VHDL/Verilog Hardware simulated with RTL simulator
- Compiled Software loaded in memory

## Pros

- Full hardware observability
- Actual RTL being simulated

## Cons

- Software debug limited (depending on tools)
- Slow



# Hw/Sw Co-Simulation Flow

software.c

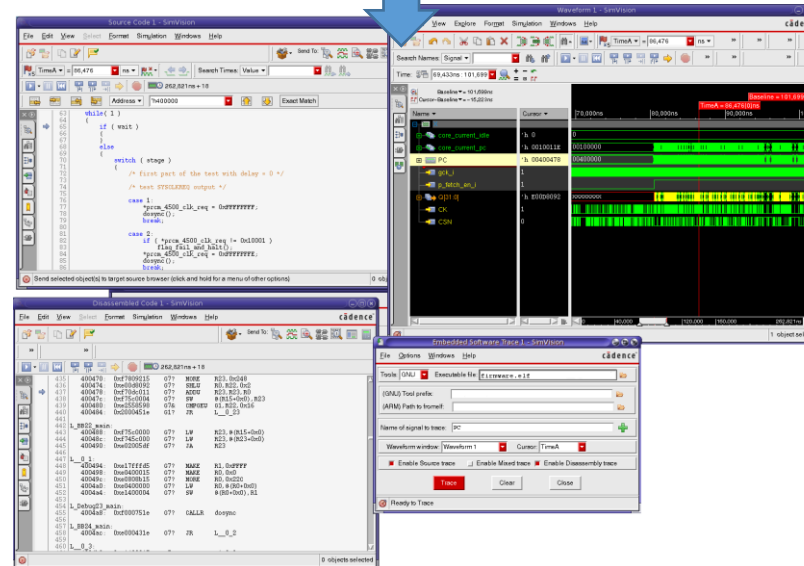
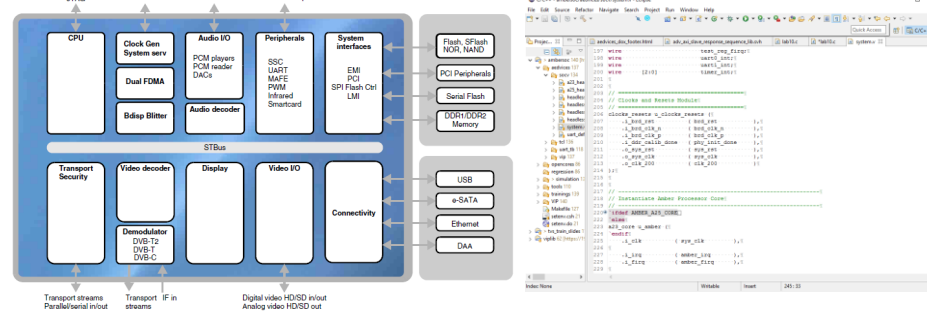
Vhdl / Verilog

Compile ( gcc / ld )

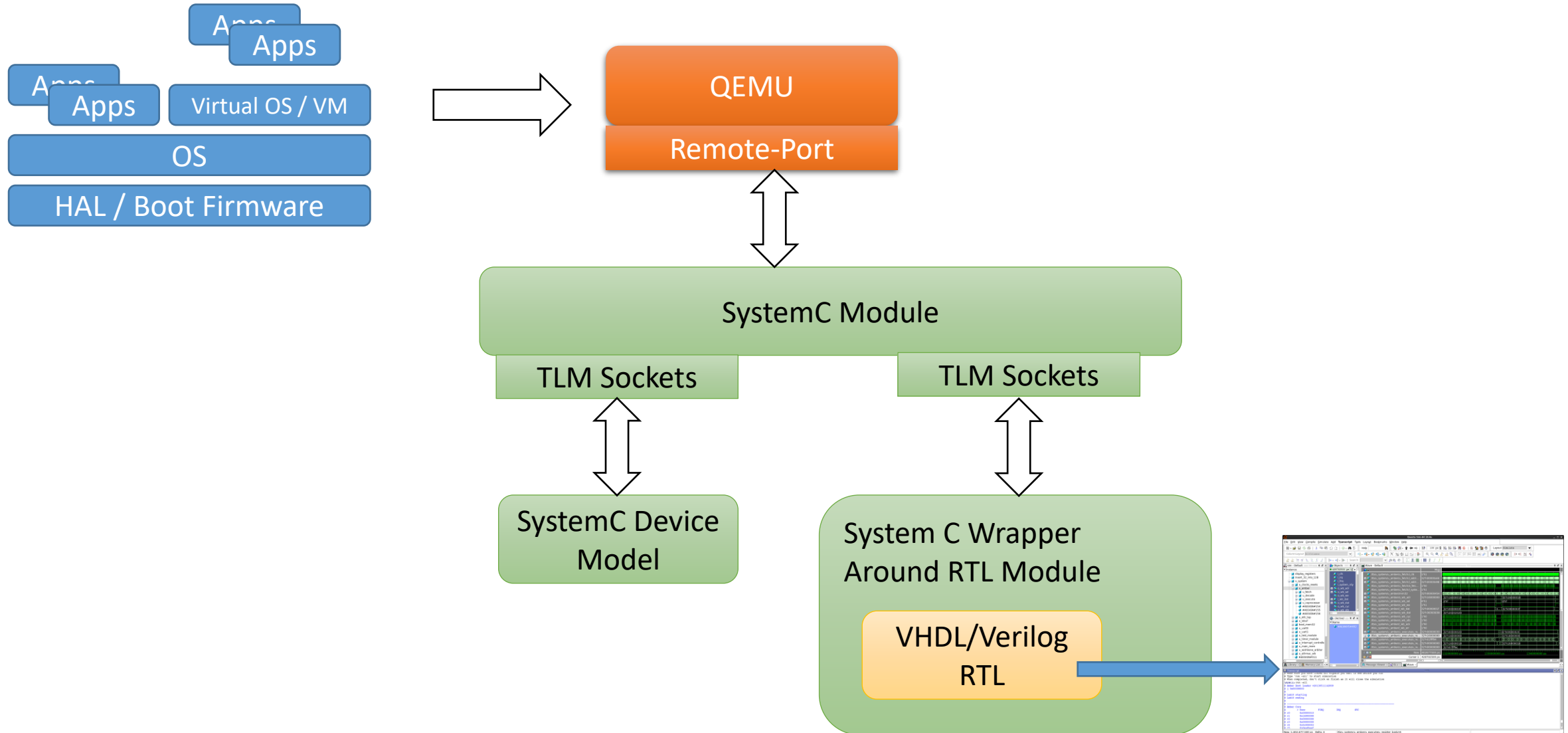
Compile / Elaborate / Load Sim

software.bin

Load Binary  
to VHDL Array  
/ Mem



# Virtualization with QEMU + RTL



# FPGA Prototype Flow

software.c

Vhdl / Verilog

Compile ( gcc / ld )

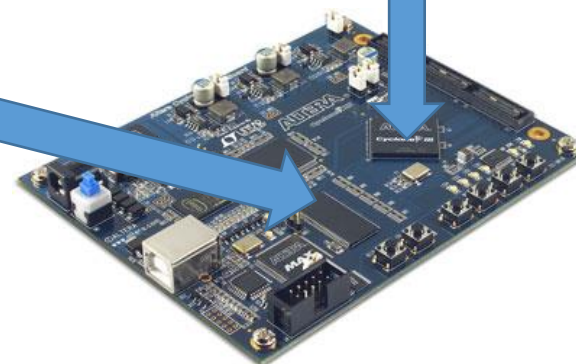
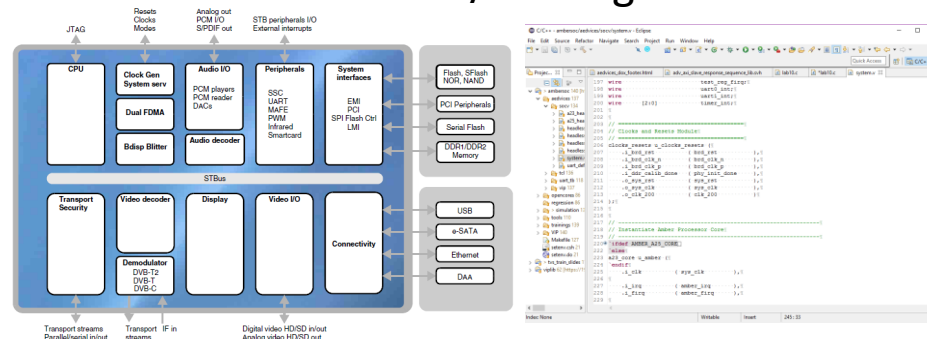
FPGA Synthesis

software.bin

Bit stream

Load Binary Thru

- JTAG
- Flash
- SPI
- USB
- Ethernet



# Verification Activity

Verification Plan

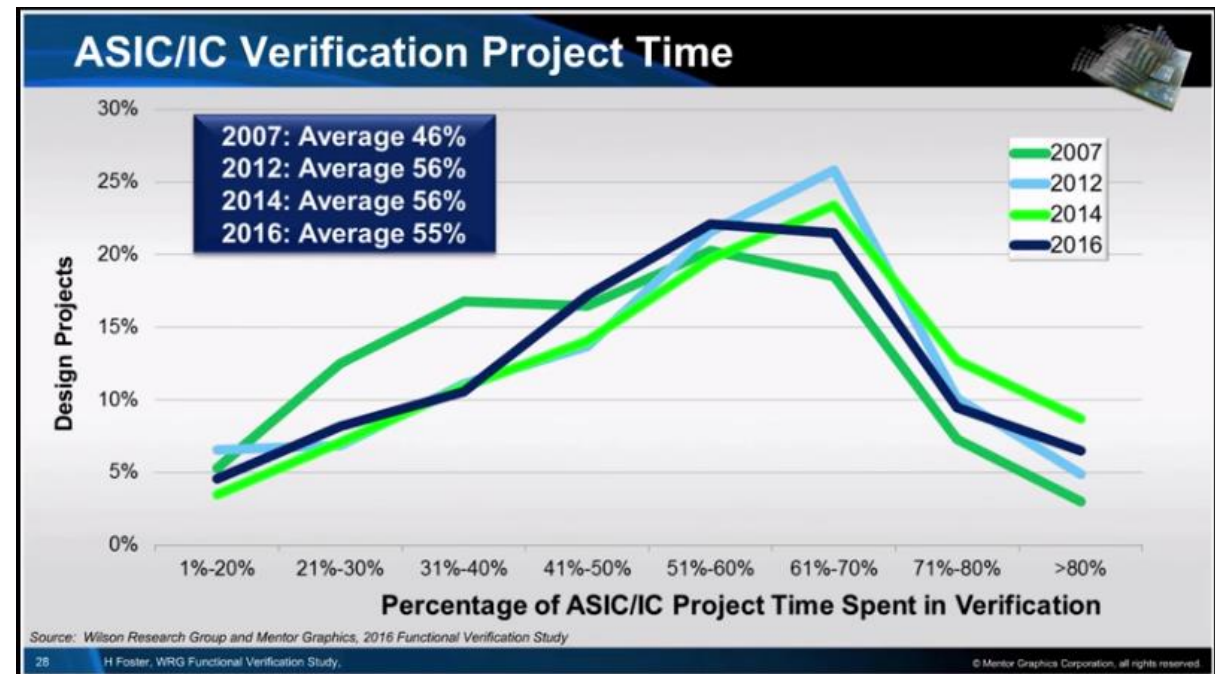
Verification Development

- Verification Environment / Test Bench Development
- Test Development

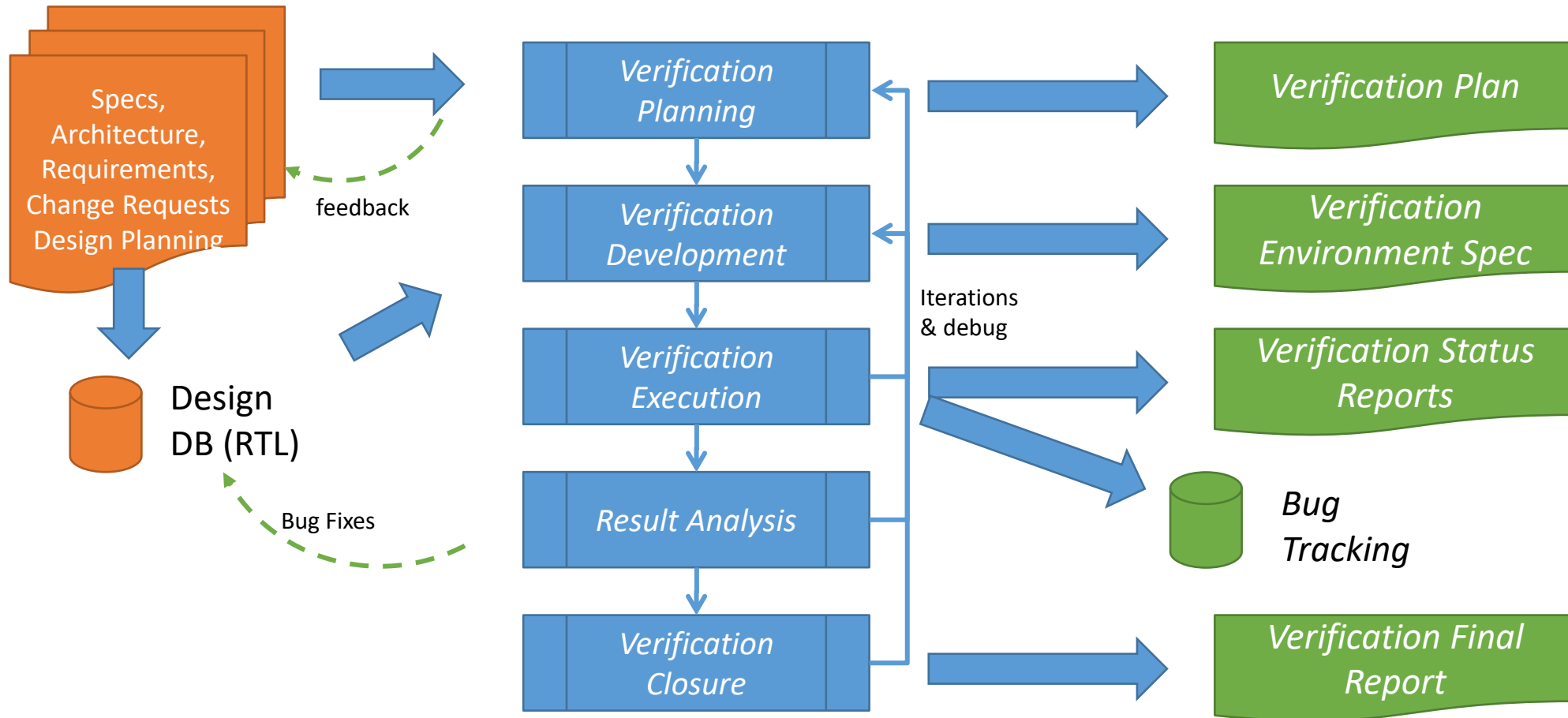
Test and Debug

Result Analysis

Verification Reports



# Global Verification Process



inputs

process

task

outputs

# How to implement verification ?

## C / C++

- Good and known programming languages
- Need to interface with simulators
- Lack of timing semantics and poor verification features

## VHDL or Verilog

- Not OOP
- Lack of verification specific features

## SystemC

- SystemC Verification Library
- Based on C++
- Non native verification features
- Lack of industry support ( verification IPs) but has good interoperability with Specman and SystemVerilog.

## E / Specman

- OOP: Object Oriented Programming
- AOP : Aspect Oriented Programming
- CSP: Constraint Satisfaction Programming
- Verification Features
- E-Reuse Methodology (eRM)

## System Verilog

- OOP
- CSP
- Verification Features
- Lack of clear guidelines or systematic approach

## UVM: Universal Verification Methodology

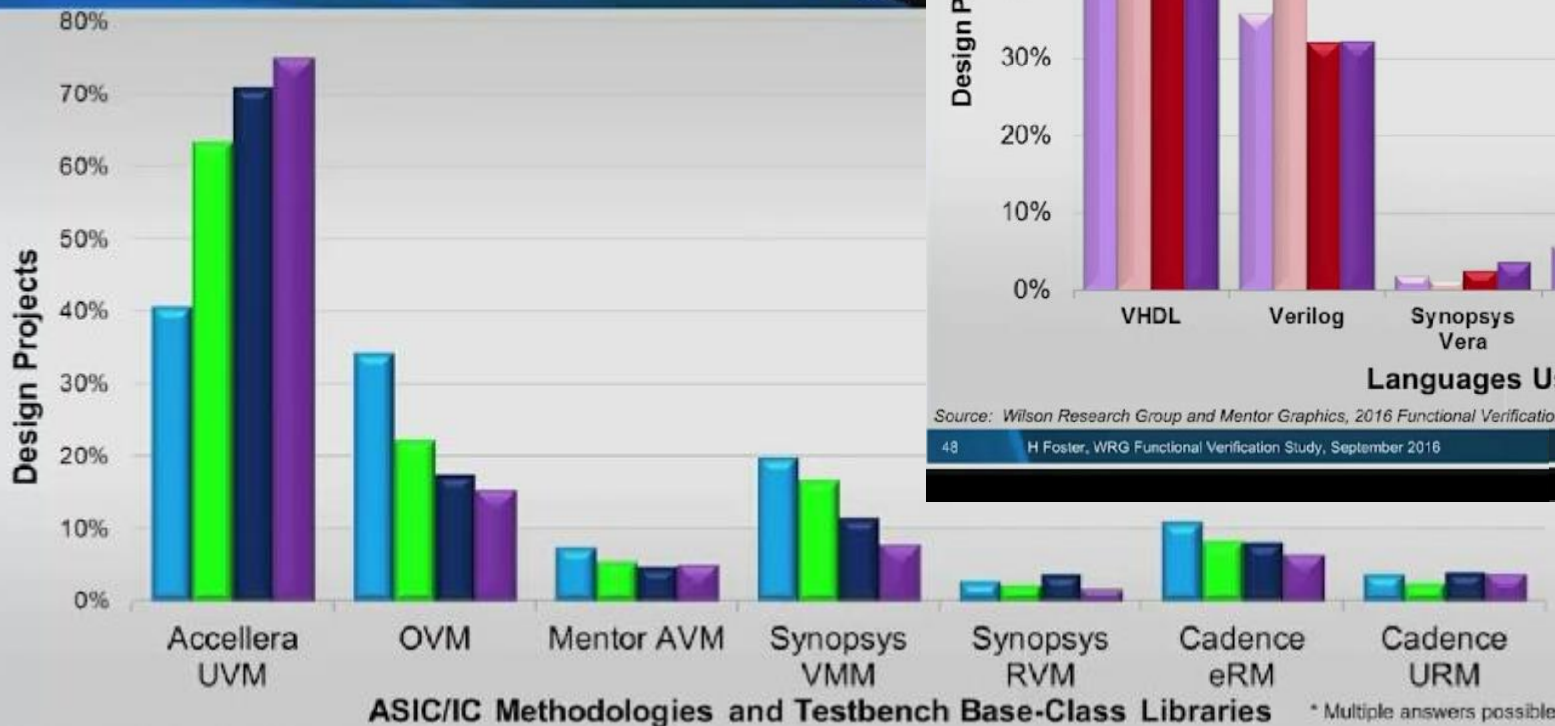
- Templates and Guidelines on top of SystemVerilog
- Common way of architecting testbenches

Today's Verification

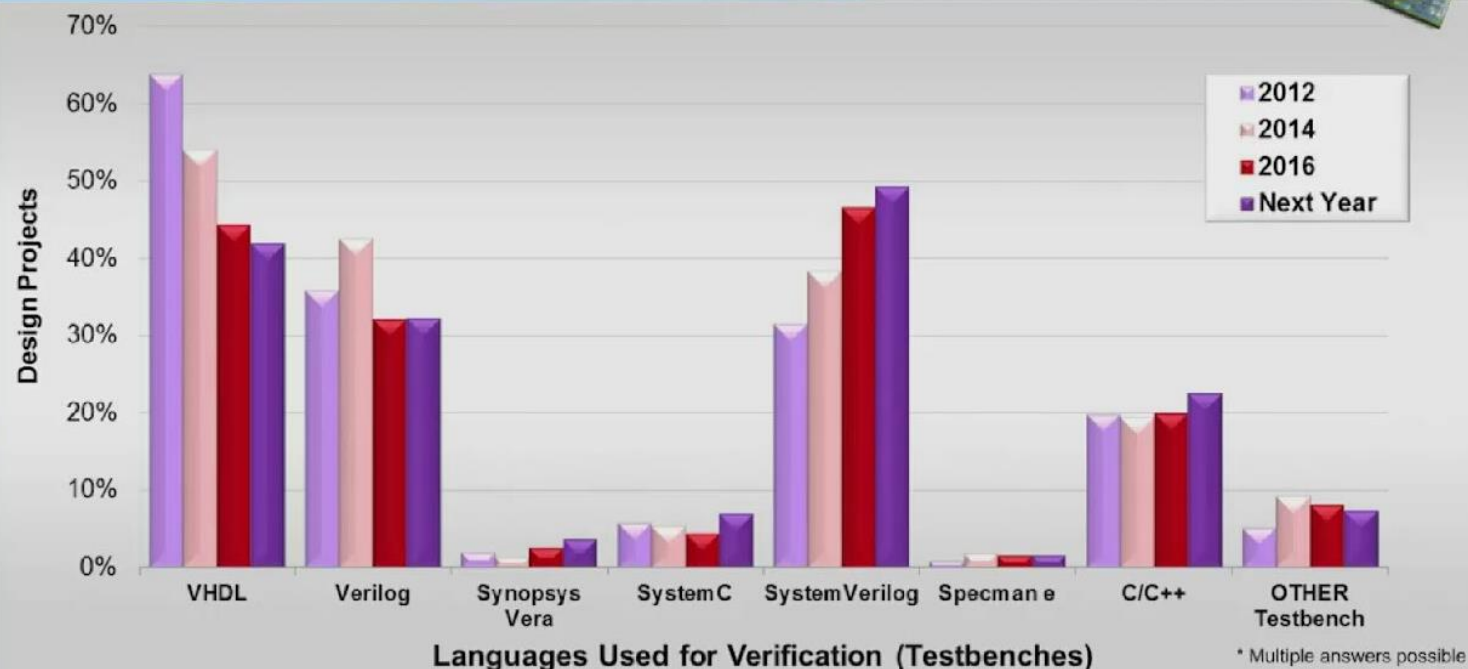


# Verification Languages

## ASIC/IC Testbench Methodology



## FPGA Verification Language Adoption Trends



Source: Wilson Research Group and Mentor Graphics, 2016 Functional Verification Study

48 H Foster, WRG Functional Verification Study, September 2016

© Mentor Graphics Corporation, all rights reserved.

## ✓ Why do we verify ?

- To avoid bugs
- To minimize cost of fixing bugs
- To save reputation, money, live
- To sell products and make money

## ✓ What do we verify ?

- Any design: CPU, SoC, peripherals, systems with embedded software

## ✓ When do we verify ?

- As early as possible

## ✓ Who verifies ?

- A verification engineer with verification mindset

## ✓ How do we verify ?

- The foundations of verification:
  - Verification Plan, Independent Analysis
  - Test Benches and Tests
  - Hierarchical verification
- SystemVerilog, UVM, C, others.
- Simulation, Prototyping, Emulation





# Quizz...



- Why is the verification performed by a different person than the developer ?
- What is a false negative ?
- Why do we verify ?

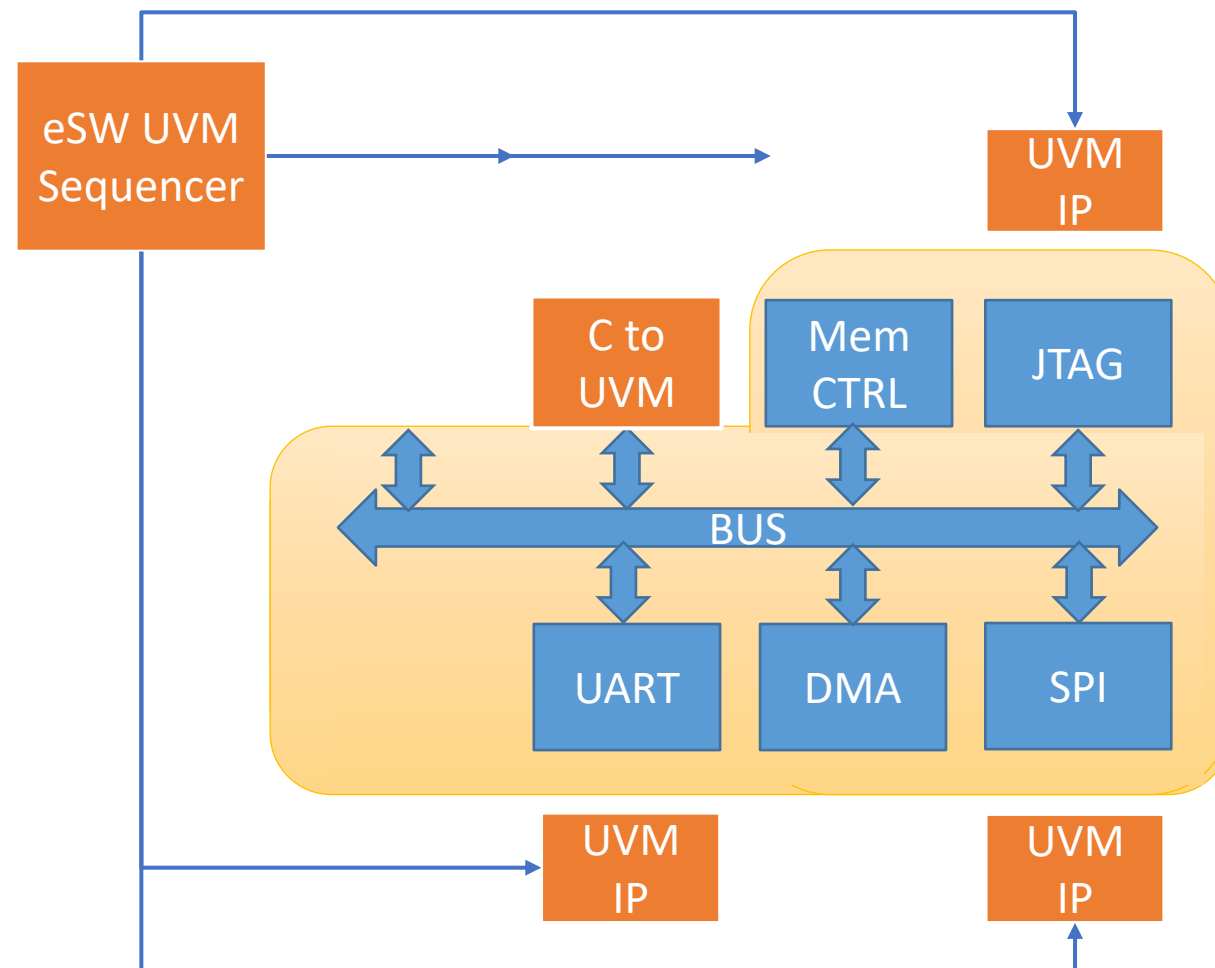
# Internship

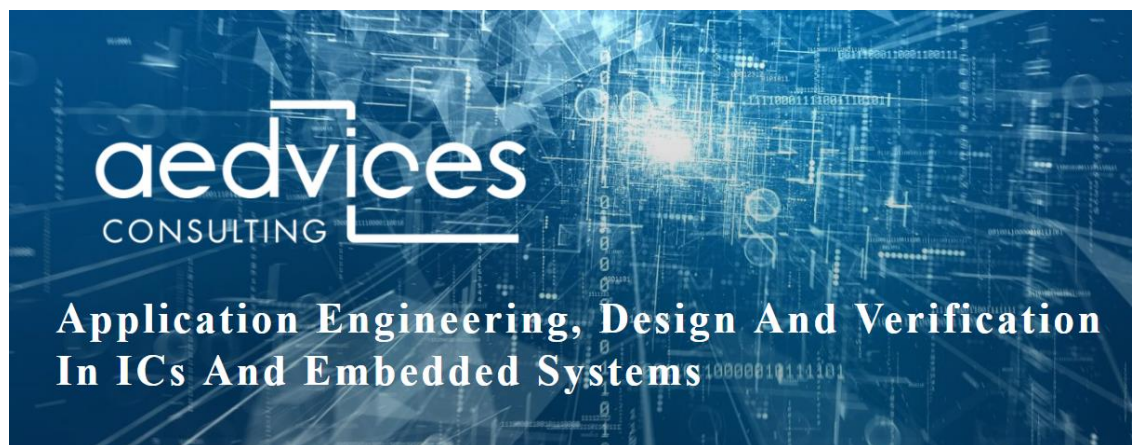
## Goals

- Improve a platform for verification methodologies demos
- Demonstrate Hardware/Software co-verification technics on a SoC
- Demonstrate Innovative and New Verification Approaches (Portable Stimulus)

## Tasks

- Learn Functional Verification Approach ( SystemVerilog / UVM )
- Understand the existing training and demo platform
- Replace existing CPU with either ARM Cortex-M or Risc-V
- Build a hardware / software co-verification environment demonstrating:
  - Headless ( CPU-less ) SoC verification with host code C execution
  - Virtualization of the CPU
  - UVM Based Software Driven Verification





[www.aedvices.com](http://www.aedvices.com)  
[contact@aedvices.com](mailto:contact@aedvices.com)