Verification at the Infineon Design Centre, Bristol

Dr. Tim Blackmore
7th December 2015

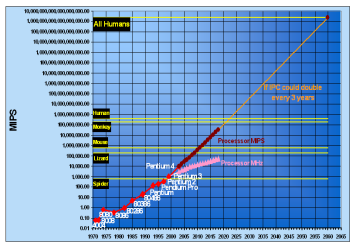
Overview

Semiconductor Industry

Semiconductor Industry

Moore's Law

- Number of transistors on a semiconductor doubles every two years
- Complexity and processing power doubles very two years
- Size and price of a semiconductor halves every two years



- Semiconductors have become ubiquitous
  - Computation, communication, power generation, trains, cars, domestic appliances, security, internet of things, …

Working in the Semiconductor Industry

Fast moving industry – have to cope with exponential growth in complexity

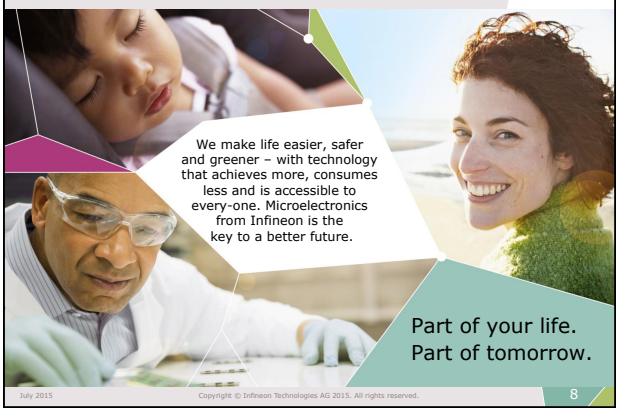
- 100x-200x increase in complexity in last 15 years
  - Generation-to-generation improvement in efficiency
  - Ability to think **innovatively**
  - Ability to learn from others - **continuous development**
- New problems with every iteration
  - Strong **problem solving skills**
- High degrees of automation
  - Use of EDA tools
  - Strong **programming, scripting and data analysing skills**
- Re-use as much as possible
  - Understanding of **modular development**

Semiconductor Industry in Bristol Area

"Silicon Gorge" - 5th largest conglomeration of high tech companies in Europe (Wikipedia)

- Infineon
  - World #1 in power semiconductors and #2 in automotive and in smart cards
- Huawei, Broadcom, Qualcomm
  - Global leaders in ICT (Information and Communication Technology) solutions based in China/US
- Imagination Technologies
  - A UK based world leader in supplying IP e.g. GPRs
- Start-ups and young companies such as XMOS, ...

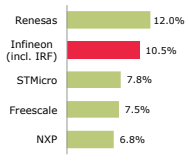
Infineon



Top positions in all major product categories

Automotive semiconductors

total market in 2014: $27.5bn

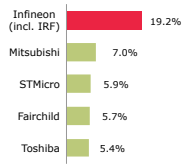


Automotive semiconductors incl. semiconductor sensors.

Source: Strategy Analytics, April 2015

Power semiconductors

total market in 2014: $16.2bn

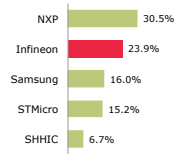


Discrete power semiconductors and power modules.

Source: IHS Inc., September 2015

Smart card Ics

total market in 2014: $2.63bn



Microcontroller-based smart card ICs.

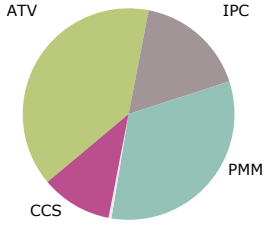
Source: IHS Inc., July 2015

Revenue split by segment

Q3 FY 2015 Revenue: € 1,586m

Automotive

Chip Card & Security



Industrial Power Control

Power Management & Multimarket

*Other Operating Segments; Corporate & Eliminations

Product range

Automotive (ATV)

- Microcontrollers 32-bit for Powertrain, Safety and ADAS
- Software development platform DAVE™
- Sensors: pressure, magnetic, radar
- Power: MOSFETS, IGBTs, smart power ICs: voltage regulators, bridges, driver Ics, CAN / LIN / Flex Ray™ transceiver*, DC-DC converters, power system Ics, system-on-chip, embedded power Ics
- Hybrid & Electric Vehicle: IGBT HybridPACK™, HybridPACK™ Double Sided Cooling (DSC) modules, gate driver ICs

Industrial Power Control (IPC)

- IGBT module solutions incl. IGBT stacks
- IGBT modules (high-power, medium-power and low-power)
- Discrete IGBTs
- Bare die business
- Driver ICs

Power Management & Multimarket (PMM)

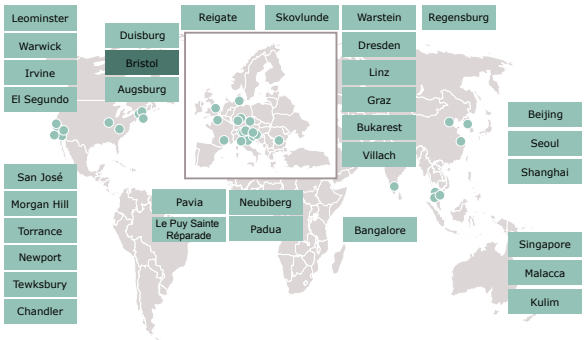
- Discrete high- and low-voltage power transistors
- Driver ICs
- Control ICs
- RF power transistors
- Small-signal components
- CMOS RF switches for antenna modules
- Antenna tuning ICs
- MEMS and ASICs for silicon microphones
- Customized chips (ASICs)

Chip Card & Security (CCS)

- Smart card (contact-less and contact-based) and embedded security ICs
- Extensive packaging and service portfolio
- Innovative solutions from basic security RFID and memories to high-end security controllers
- Leading technologies, e.g. SOLID FLASH™, Integrity Guard, VHBR, boosted NFC, Coil on Module

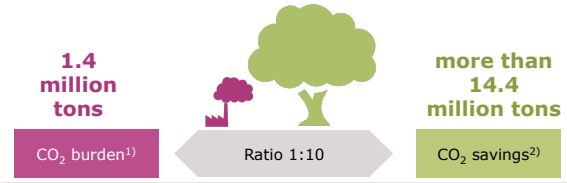
*FlexRay is a trademark licensed by FlexRay Consortium GbR

Our global R&D network



Infineon creates a net ecologic benefit

Emission Reduction enabled by our products and solutions



Net ecological benefit:
round 13 million tons of CO_2 emission reduction

1) This figure considers manufacturing, transportation, function cars, flights, materials, chemicals, waste/waste water, direct emissions, energy consumption, waste, etc. and is based on internally collected data and externally available conversion factors. All data relates to the 2014 fiscal year.
2) This figure is based on internally established criteria, which are explained in the explanatory notes. The figure relates to the calendar year 2013 and considers the following fields of application: automotive; lamp ballast control; PC power supply; renewable energy (wind, photovoltaic); and rives. CO_2 savings are calculated on the basis of potential savings of technologies in which semiconductors are used. The CO_2 savings are allocated on the basis of Infineon's market share, semiconductor content and lifetime of the technologies concerned, based on internal and external experts' estimations. Despite CO_2 footprint calculations are subject to imprecision due to the complex issues involved, the results are nevertheless clear.

The information and data given in this document apply to the Infineon Technologies group, except for International Rectifier companies.

Advanced Driver Assistance Systems (ADAS)

- ADAS applications include
  - Adaptive cruise control
  - Automatic parking
  - Collision Avoidance Systems
  - Driver Drowsiness Detection
  - …
- Typically ADAS applications require a car to have increased awareness of its surroundings
  - Uses RADAR, LIDAR, camera,
  - In future will use network connectivity
- Requires very fast processing of a lot of data at a low cost

Towards Autonomous Vehicles

- ADAS technology could be used for AVs
- Main barriers are legal, economic and cultural

https://www.youtube.com/watch?v=0tmCLZOM8kw

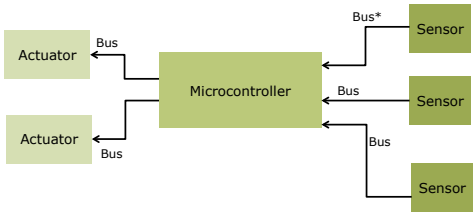
Infineon, Bristol

Infineon, Bristol

- Part of the Automotive *Microcontroller* Group of Infineon
- Working on developing microcontrollers used in application areas such as
  - Engine Management - increased fuel efficiency, reduced emissions
  - ADAS
  - Safety (Airbags, drive-by-wire, ABS, …)
- Around 40 permanent engineers working in concept, design and verification
  - 18 permanent verification engineers

What is an Automotive Microcontroller?

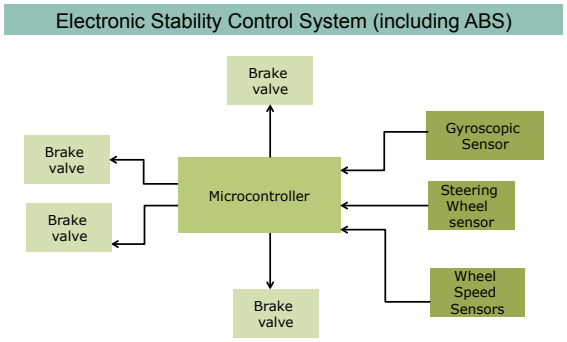
Typical Automotive System



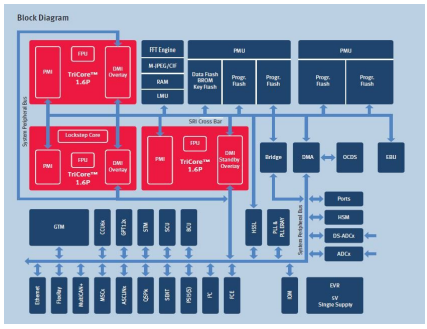
*E.g. Ethernet, CAN, Flexray, SPI

Infineon, Bristol

Example Automotive System

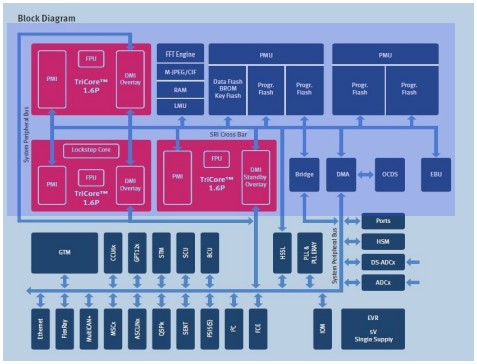


TC279TA Microcontroller

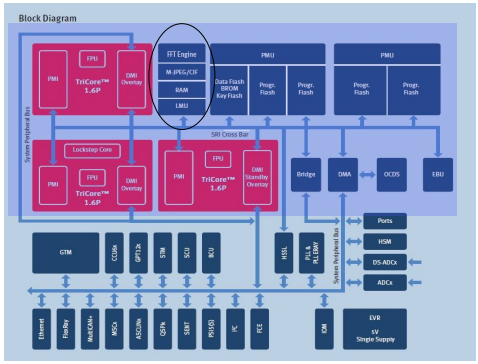


Source: http://www.infineon.com/cms/en/product/microcontroller/32-bit-tricore-tm-microcontroller/aurix-tm-family/aurix-tm-family---tc297ta/channel.html?channel=db3a304342c787030142dbf31c2b151c

TC279TA Microcontroller – Bristol

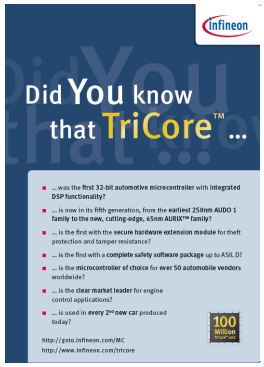


TC279TA Microcontroller - ADAS



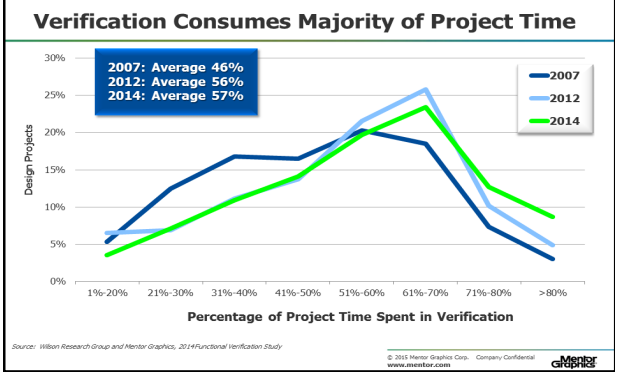
TriCore Processor

- Processor development interesting and challenging
- "The most widely distributed processor you've probably never heard of"
- 250 Million sold as of 19 November 2015
- On average every second vehicle contains one



Verification

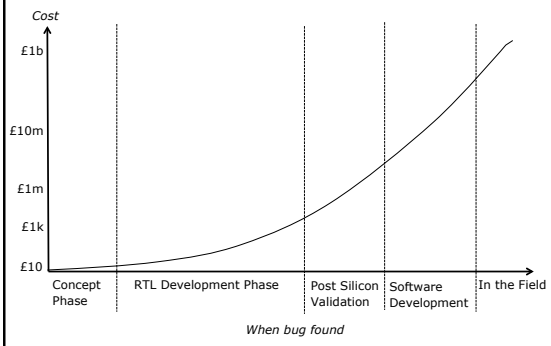
Amount of Verification - Industry



Cost of Not Doing Verification

- In 2014 General Motors spent $3.2 billion on recalls and victim compensation and set aside a further $900 million for future costs
  - Wiped out annual profits
  - $2.1 billion on recalls
  - Largest single problem was a faulty ignition switch (caused engine to shut off and prevented airbag from deploying)
    - Connected with 124 deaths
    - Recalled >29 million vehicles worldwide
    - Mechanical problem rather leading to an electronic failures – a bug in the electronics could have had a similar effect
    - Software problems can also have similar effects – but much cheaper to replace

Cost of Hardware Bugs – 'Shift Left'

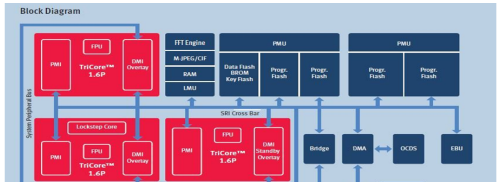


Summary

- Typically > 50% - and often > 70% - of project spent in verification
- Economic imperative
  - Fixing hardware bugs can be very expensive
  - Litigation can be very expensive
  - Loss of goodwill can be very expensive
- Safety
  - Incorrect function can lead to loss of life
  - Since 2011 automotive electronics has it's own safety standard – ISO26262

Module and System Level Verification

TC279TA Compute and ADAS Sub-system



- Bristol responsible for two sub-systems
  - The sub-system responsible for the 'compute'
  - The ADAS sub-system
- Includes the development of complex IP – such as TriCore

Challenges

- **Complexity** makes it infeasible for single intellect to fully understand verification requirements of a complete sub-system
  - Currently ~30 engineers working in verification
- **Controllability** – need complex software to control module inputs at sub-system level
  - Very difficult to stress interfaces sufficiently to reach all corner cases
- **Observability** – self-checking software not sufficient to observe all failures -> need models and assertions
- **Run times**
  - Simulation slows down as amount of logic increases
  - Formal tools can only prove simple properties on large designs

Addressing the challenges

Divide and Conquer

- Divide systems into sub-systems and sub-systems into modules
  - Split at standard and simple interfaces
    - Standard interfaces such as busses
    - Simple interfaces may just be handshakes
  - Minimises risks of missing bugs in module interaction
  - Can end up with large modules – such as processors
- Verification is then split
  - Module level - fully stresses module functionality
  - Sub-system and system level – checks interactions between modules and sub-systems

Module Level Verification

Goals of module level verification

- Rigorous verification of complete module
- Sufficient stimuli has been applied to achieve
  - Exacting code coverage goals (e.g. 100% line, branch and toggle coverage)
  - Functional coverage on
    - High level requirements
    - All features in Target Specification
    - All design corner cases
    - …
- And sufficient checkers are in place to detect any failure conditions

Module Level Verification

Goals achieved through

- Simulation with test-case automation
  - Stimuli generated by constrained-random generation (using Specman/e)
    - Many thousands of scenarios covering many millions of transactions generated every night
    - Most interesting scenarios saved and re-run across bug fixes
  - Checking done by
    - Transaction-level (!) reference models and scoreboards
    - Assertions to check signal and timing behaviour
- Targeted use of formal verification

System and Sub-system Level Verification

Target: Verify interactions between modules

- Typical Goals
  - Modules are connected together correctly
  - Functionality requiring interactions between modules
  - Main metric - toggle coverage on module I/O
- Goals typically achieved through
  - Self-checking C/C++ directed test cases, use-cases and benchmarks
  - Growing use of
    - Constrained-random (and graph-based) techniques
    - Reference models and assertions
    - Very targeted formal verification

Verification Technologies

Verification Technologies

Simulation	Formal
Run lots of example stimuli through DUV and check result	Exhaustive checking of properties
Complex Test Bench • Test case automation • Reference models • Object and aspect orientated languages	No test bench • Only constraints on inputs to avoid false fails • Constraints developed alongside properties
Can deal with any size of design • Run times will slow down on large designs	Capacity limitations mean only simple properties can be run on large designs
Measure effectiveness of stimuli and checkers using coverage	Measure completeness of properties using coverage

When we use formal

Early	Formal Friendly	High Risk Blocks	Simple checks
› No up-front test bench development › Take an Agile approach to design › Property-driven approach › Used by designers	› Low effort and high return › E.g. maths blocks such as adders and ECC encoders and decoders	› High effort but still good RFE › Candidate blocks: › Have many corner cases › Are mission critical › Highly configurable and highly re-used	› Properties generated by scripts or › Inbuilt push-button Formal Apps › Usable at (sub-) system level › E.g. Connectivity checks using xml

When we use simulation

Most of the time

- Work horse verification technology
- Expertise more generally available
- Scales better
  - In area (full module, sub-system, system)
  - In time (e.g. radar data streams)

Re-Use

Different types of re-use

Across Generations	Across Products	Across Test Benches	Across Technologies
› Complexity doubles every generation › Engineering effort remains approx. constant › Cannot start again with each generation	› Single generation will have high-end, low-end and application specific products › IP and VIP must be configurable › Integration will need fully verifying for each product	› Supported by re-use methodologies (e.g. UVM) › E.g. re-use of VIP for busses across all IP with bus interface › E.g. re-use of checkers across hierarchies	› Re-use between formal and simulation › E.g. Re-use of constraints and properties as assertions

Example Property

prev(vld) => !vld // Cannot get back-to-back valids

- Can be used as a
  1. Formal property – check behaviour of DUV
  2. Formal constraint – constrain how formal tool can 'drive' *vld* into the DUV
  3. Assertion in simulation – check design or test bench behaviour
- Supported by move from proprietary to standardised assertion languages (PSL, SVA) over the last ten years
  - Same property/constraint can be re-used in simulation without re-coding
- Why would you want to do this …?

Formal Constraints

Avoiding False Failures

- Formal properties can fail due to the tool driving illegal inputs
  - The DUV is behaving in a bad but unrealistic way
- Example
  - Protocol excludes possibility of back-to-back data
  - Property fails and the generated counter-example shows *vld* being driven in consecutive cycles
- Detection : debug of counter-example given by failing properties
- Solution : add constraint and re-run property

prev(vld) => !vld

Incorrectly coded constraints

False Passes

Example: Constraint

prev(vld) => !vld // Cannot get back-to-back valids

Incorrectly coded as

vld == !prev(vld) // Will make vld toggle!!

- Over constraint – formal tool is now not fully exercising DUV within protocol
  - E.g. bug in DUV related to sequence {!vld; !vld; vld} will not be found
  - A pass may be a false pass
- Over-constraints can be due to
  - Miscoded constraints
  - Misunderstanding of input behaviour

False passes are the most common reason for missing bugs when using formal property checking

Detecting False Passes

- Passing property gives nothing (or at least very little) to debug
- Over-constraints can be detected by
  - Analysing code coverage achieved by property set
    - Some formal tools now have code coverage capability built in
    - Any lines of code not verified by properties? Are these expected?
  - Checking constraints in simulation
    - Constraint can be run as an assertion in simulation

      Formal: *assume vld == !prev(vld) // Will make vld toggle*

      Simulation: *assume vld == !prev(vld) // Will check vld toggles*
    - If the assertion fails then the constraint is incorrect (or there is a problem with the test bench) and needs to be re-worked (and the property set re-run)
    - Will not detect all over-constraints but will certainly detect this one!

Checking Properties at Different Hierarchy Levels

- Checking the properties at different hierarchy levels (IP, Subsystem, SoC) can help with checking that the block has been properly integrated
  - Failing property indicates inputs are driven incorrectly e.g. an IP driving valid data too aggressively
- May not be possible to formally check properties at higher levels of hierarchy
  - Capacity issues for formal tools for large designs
  - New interfaces to constrain -> new false failures
- Can easily check properties as assertions in simulation at different hierarchy levels
  - Should be sufficient to just check the constraints since have formally proved that if these are satisfied the properties hold

Summary

Summary

- Semiconductor industry is fast moving
  - As a semiconductor company need to be able to handle drive for doubling of complexity every couple of years
- Automotive semiconductors among quickest growing segments – driven by ADAS, AVs and electrification
- Verification requirements for automotive applications are driven by conflicting requirements
  - Need to become more efficient
  - Large cost of getting it wrong – both economic and impact on safety

Summary

- Effectively and efficiently deal with complexity by
  - Divide and conquering
  - Careful planning
  - Achieving verification goals at appropriate level of hierarchy
  - Achieving verification goals with appropriate technology
  - Very good verification engineers working with leading edge tools and methodologies!
- Re-use is a major driver for efficiency
  - Example of re-use of constraints and properties
    - across simulation and formal technologies
    - Across different hierarchy levels

Addressing the challenges

Verification Planning

Example Goal Types	Examples	Example Techniques
Requirements	The DMA shall be able to move data from source to target at 3.2 GB per second	Benchmark @ sub-system level
Features	The DMA has two move engines	FC @ module level
Properties	If a move engine is not in use it must service any new move requests	Assertions or formal property checking @ module level
Corner Cases	Both move engines are requested at the same time	FC @ module level
Connectivity	All module interfaces have been connected properly	Toggle coverage or formal checks @ sub-system level
Metrics	All branches in the RTL have been executed	Combined* Code coverage @ module level