IAF0530/IAF9530

Dependability and fault tolerance

Lecture 9
Enemies of Dependability

Gert Jervan
gert.jervan@ati.ttu.ee

Downtime

- Planned downtime
  - Maintenance, repair, upgrade
- Unplanned downtime
- Dependability:
  - Turn unplanned downtime into planned downtime
  - Reduce downtime (magic nines)

Sources of Problems

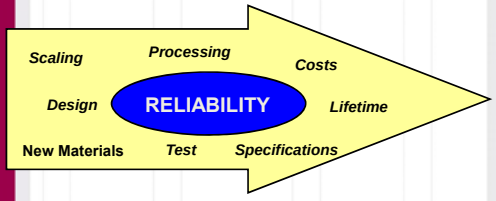
Category	Early 80s	Late 80s	90s	2000s
Hardware + environment	32%	29%	20%	Up
Software	26%	58%	40%	The same
Human Operators	42%	13%	40%	Down

Hardware

Hardware and Environment Failures

- Moving parts, high speed, low tolerance, high complexity: disks, tape drives/libraries
- Lowest MTBF found in fans and power supplies
- Often fans fail gradually → subtle, sporadic failures in CPU, memory, backplane
- Environment: power, cooling, dehumidifying, cables, fire, collapsing racks, ventilation, earthquakes, …

Hardware Reliability Challenges



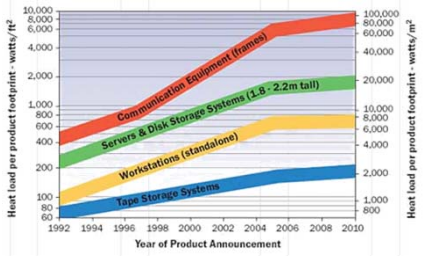
Reliability Dependencies and Impact to Cost

Hardware - Background

- Chip designers, device engineers and the high-reliability community recognize that reliability concerns ultimately limit the scalability of any generation of microelectronics technology
- Statistical methods and reliability physics provide the foundation for better understanding the next generation of scaled microelectronics
  - Microelectronics device physics
  - Reliability analysis and modeling
  - Experimentation
  - Accelerated testing
  - Failure analysis
- The design, fabrication and implementation of highly aggressive advanced microelectronics requires expert controls, modern reliability approaches and novel qualification strategies

7

Heat Density



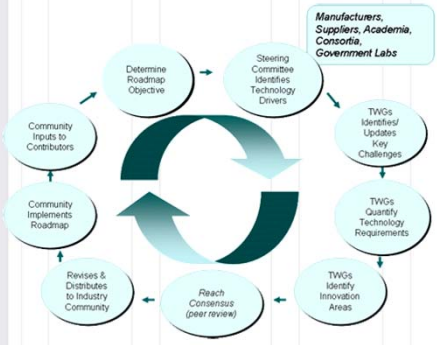
8

ITRS Roadmap

- ITRS predicts the main trends in the semiconductor industry spanning across 15 years into the future.
- The International Technology Roadmap for Semiconductors is sponsored by the five leading chip manufacturing regions in the world: Europe, Japan, Korea, Taiwan, and the United States.
- The objective of the ITRS is to ensure cost-effective advancements in the performance of the integrated circuit and the products that employ such devices, thereby continuing the health and success of this industry.

9

ITRS Roadmap



10

ITRS Roadmap

- www.itrs2.net
- ITRS 2.0: 2015
- ITRS 1.0 Editions:
  - 1994, 1997, 1999, 2001, 2003, 2005, 2007, 2009, 2012, 2013
  - Previously: SIA Roadmap

11

ITRS 2.0

Table MM01 - More Moore - Logic Core Device Technology Roadmap

YEAR OF PRODUCTION	2015	2017	2019	2021	2024	2027	2030
Logic device technology naming	P70M56	P48M36	P42M24	P32M20	P24M12G1	P24M12G2	P24M12G3
Logic industry "Node Range" Labeling (nm)	"16/14"	"11/10"	"8/7"	"6/5"	"4/3"	"3/2.5"	"2/1.5"
Logic device structure options	finFET FDSOI	finFET FDSOI	finFET LGAA	finFET LGAA VGAA	VGAA, M3D	VGAA, M3D	VGAA, M3D
L_g: Physical Gate Length for HP Logic (nm) [3]	24	18	14	10	10	10	10
Power Supply Voltage - Vdd (V)	0,80	0,75	0,70	0,65	0,55	0,45	0,40
Number of wiring layers	13	14	15	16	18	22	30
DRAM cell size (um2)	0,00259	0,00218	0,00130	0,00090	0,00058	0,00034	0,00024
DRAM cell size factor	6	6	4	4	4	4	4
DRAM retention time (ms)	64	64	64	64	64	64	64
DRAM soft error rate (fits)	1000	1000	1000	1000	1000	1000,0	1000,0
DRAM Gbits/chip target	8G	8G	16G	16G	32G	32G	32G

12

HiPEAC roadmap

- http://www.hipeac.net/roadmap
- The HiPEAC roadmap describes the HiPEAC vision on high-performance embedded architecture and compilation for the coming decade. It starts from societal challenges, application and industry trends, and technological constraints which lead to 7 technical challenges. This forms the basis for the HiPEAC vision "keep it simple for humans, and let the computer do the hard work" and its consequences. The roadmap ends with a SWOT analysis of the computing systems industry in Europe, and 6 research recommendations.

13

The problem to be solved:

How to design reliable system

out of

non-reliable hardware?

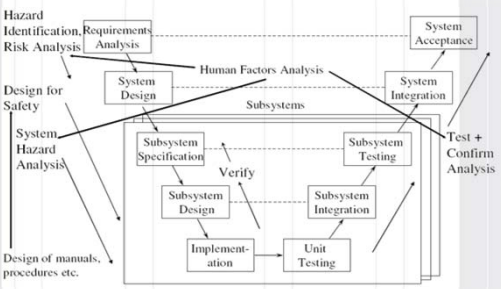
Human Factors

Human Factors

- The role of humans in safety-critical systems
- Human Reliability Analysis
  - task analysis
  - human error identification
    - human error model: Reason
  - human reliability quantification
  - mitigating human error
- Safe user interface design

16

Human Factors



17

Have we learnt since Therac-25

Software for Certain Medtronic Implanted Infusion Pumps Recalled

FDA Patient Safety News: Show #32, October 2004

- Medtronic is recalling certain software application cards.They're used in the company's Model 8840 N'Vision Clinician Programmers. These hand-held devices are used to program a number of implantable devices, including the SynchroMed and SychroMed EL implantable infusion pumps.

18

Have we learnt since Therac-25

- The recall is prompted by reports of data entry errors that have led to serious drug overdoses, including two patient deaths. The overdoses occurred when clinicians who were programming the pump entered the wrong time duration or the wrong interval --- for example, mistakenly putting the time interval between periodic drug boluses in the "minutes" field, instead of the "hours" field.

19

Have we learnt since Therac-25

- The recalled software may have contributed to these errors because one part of the screen did not have labels on the fields for hours, minutes, and seconds. Medtronic is now distributing replacement software that adds time labels to the screen to help reduce the risk of these kinds of programming errors.

20

Automation

- A driving force of automation is to compensate for human disadvantages
  - humans are unreliable components of systems requiring replacement by reliable computers
  - humans have limited capabilities in response time and capacity
- However, humans play an essential role in safety-critical decision making
  - computers are not flexible or adaptable, e.g., response in emergency situations
  - computers cannot make creative judgements or strategic decisions

21

Human Error and Risk

- Automation yields
  - Increased capacity and productivity
  - Reduction in manual workload and fatigue
  - Increased safety
- But
  - Need specialised training
  - Cost of maintenance
- Impact on human operators
  - Unclear if overall workload reduced
  - Increased complacency due to overconfidence?

22

Role of Humans

- **Monitor**: detecting errors
  - it may not be possible to determine if an error has occurred
  - the system may provide inadequate feedback
  - operators may become complacent
- **Backup**: in an emergency
  - operators may become de-skilled
  - information provided may be inadequate for intervention
  - automated systems are usually too complicated

23

Role of Humans

- **Partner**: responsible for part of a task
  - humans may be assigned “hard to automate” part
  - humans may be responsible for monitoring and maintaining
  - division of responsibility may make building a mental model harder

24

Do Humans Cause Most Accidents?

- 85% of work accidents are due to unsafe acts by humans rather than unsafe conditions
- Should we believe the statistics?
  - Data may be biased and incomplete: in 60-80% of accidents caused by operator's loss of control, 75% of those had system/safety malfunction that preceded the operator action
    - e.g. DC-10 crash deemed pilot error, involved autopilot headings alteration without telling the crew
  - Positive actions are not usually recorded
    - only 10% of recovery from emergency are pilot errors
  - Operators are expected to always recover from emergency
    - Error can be due to poor design

25

Do Humans Cause Most Accidents?

- Should we believe the statistics?
  - Operators have to intervene at limits, diagnose/respond quickly
    - E.g. consequences can be serious
  - Hindsight allows to identify a better decision
    - Operator's knowledge may be partial, or understanding erroneous
  - Separating operator error from design error is difficult
    - Examples from nuclear power plants:
      - Dials measuring the same quantities calibrated in different scales
      - Location of critical decimal points unclear
      - Critical displays located at back panels
      - Labels/colours inconsistent and misleading

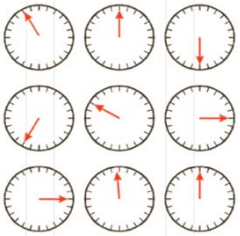
26

What are humans good at?

- Detecting correlations and exceptions
  - Patterns/clusters in graphical data
  - Breaks in lines
  - Visual/sound disturbances
- Detecting isolated movement
  - Waving
  - Flashing lights
- Detecting differences
  - Sounds, alarms, etc
  - Lights on/off
  - etc.

27

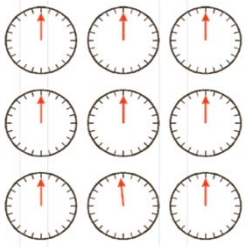
Example of Dial Controls



- **Bad interface**, cannot tell normal from abnormal.
- Advice is to fix normal at 12 o'clock position.

28

Example of Dial Controls



- **Good interface**: can spot abnormal position even for 5 deg change

29

Humans vs Machines

- Where machines have advantage…
  - Sensing/Actuating: broader range of sensors, able to perform in harsh environments
  - Cognition: no boredom, precision of calculations, repeatability, predictability
- Where humans have advantage…
  - Sensing/Actuating: image processing, edge & anomaly detection, flexibility
  - Cognition: ability to respond in unknown situations
- Should you trust humans or machines?
  - Boeing trusts people (pilot has ultimate authority).
  - Airbus trusts machines (flight control software has authority over pilot).

30

Human Machine Interaction (HMI)

- Hybrid discipline: psychology, engineering, ergonomics, medicine, sociology, mathematics
- Concerned with the impact of human operators and maintainers on system performance, safety and productivity
- Concerned with enhancing the efficiency, flexibility, comprehensibility and robustness of user interaction
- In the safety-critical context, the primary concern is to enhance robustness, possibly at the expense of efficiency and flexibility

31

Human Reliability Analysis (HRA)

- Identify potential operator errors that may lead to hazards and reduce error where risk is sufficiently high
- Four steps:
  - **task analysis**: characterise the actions performed to achieve particular goals
  - **human error identification**: identify possible erroneous actions in performing a task
  - **human reliability quantification**: estimate likelihood of error
  - **mitigation of human error**: identify control options

32

Task Analysis

- Tasks are activities to transform some given initial state into a goal state, i.e., goal-directed
- Structured from sub-tasks and elementary actions
- Each elementary action is concerned with a manipulation to be performed upon an object in the task domain
- Procedures for
  - normal operation of the system
  - maintenance of the system
  - emergency situations
- Logical sequence of actions that the operator engages in and the detailed physical executions that the operator

33

Human-Task Mismatch

- Human error is not a useful term
  - Implies possible to improve humans
- Human-Task Mismatch better term
  - Erroneous behaviour inextricably connected to the behaviour needed to complete a task
- Tasks
  - Involve problem solving, decision making
  - Need adaptation, experimentation, optimisation
- Levels of cognitive control [Rasmussen's]
  - Skills-based behaviour (smooth sensory based)
  - Rule-based behaviour (conscious problem solving)
  - Knowledge-based behaviour (goal known, planning by selection, trial and error, etc)

34

Experimentaton versus Error

- Designer relies mostly on knowledge-based behaviour
- Operator employs all three
  - In training, from knowledge- or rule-based to skills based
  - In unfamiliar situation, use knowledge-based to develop rules-based
  - Needs to maintain knowledge-based throughout
- Experimentation
  - Test a set of hypothesis through mental reasoning
  - May be unreliable
- Human error
  - unsuccessful experiments, in unkind environment
- Design for error tolerance

35

Human as Monitor

- Monitoring, rather than active control
  - Responsible for detecting/repairing problems
- Humans perform badly…
  - Task may be impossible
    - Cannot check in real-time if computer performs correctly
  - Operator dependent on information provided
    - Too much or too little is bad
  - Information is indirect
    - System handles most functionality
  - Failures may be silent or masked
    - E.g. autopilot disengages
  - Tasks are such that lower alertness results
    - Mechanical, lack of stimulation, can act without noticing

36

Human as Back-up

- Emergency only, rather than active control
  - Expected to take appropriate action
- Good design is essential
  - Can lower proficiency and increase reluctance to intervene
    - Infrequent usage
    - Cognitive and physical skills decline in absence of practice
    - High skills often needed!
      - E.g. emergency shutdown of nuclear plant
  - Fault-intolerant systems may lead to larger errors
    - May fail in ways difficult to anticipate
  - Harder to manage in crisis
    - Not fully aware of the internal state
    - Computer support for decision making

37

Human as Partner

- Both humans and automated system assigned control tasks
  - Number of human tasks reduced
  - Must be planned appropriately
- Modes
  - Partial automation
  - Shared control (primary responsibility with humans, but computer continuously performs checks)
- Potential problems
  - Good mental models are important
    - Must know the system state
  - Good communication is essential
    - Clarity, correctness

38

Accident Models

- Reduce description of accident to a set of events and conditions
  - Used in investigations, for prediction, etc
- Domino models
  - Social environment
  - Fault of a person
  - Unsafe act or mechanical/physical hazard
  - Accident
  - Injury
- Chain-of-events
  - Event trees, fault trees
- System theory
  - Accidents result from complex interactions

39

Human Tasks

- Simple tasks
  - Uncomplicated sequences
- Vigilance tasks
  - Detection of signals
- Emergency response tasks
  - May involve complex reactions
  - Performed under stress
- Complex tasks
  - Defined tasks, involve decision-making

40

Human Error Models

- Cognitive, e.g. Reason's model eight primary error groups
  - False sensation (lack of correspondence between subjective experience and reality)
  - Attentional failures (distraction, dividing attention)
  - Memory lapses (forgetting items)
  - Unintended words/actions
  - Recognition failures (wrongly observed signals)
  - Inaccurate and blocked recall (misremembering sequences)
  - Errors in judgement (misconceptions)
  - Reasoning errors (false deduction)
- Also Norman model of slips, mistakes in planning

41

Human-Task Mismatch again…

- Errors are an integral part of learning!
- Mechanisms of human malfunction
  - Skills-based level
    - Disorientation, motor skills failure
    - Stereotype take-over
  - Rule-based level
    - Incorrect recall of rules
    - Stereotype function
  - Knowledge-based level
    - Mental overload
    - Premature hypothesis (way of least resistance, point of no return)
- Also performance affecting factors (separately)
  - Work conditions, stress, social aspects

42

Human Factors Summary

- Understanding cognitive aspects essential
- Probability of failure difficult to predict
  - Human response affected by stress, fatigue, etc
- Must assume human error will happen sooner or later
  - Hardware support, failsafe operations
- Design for safety
  - Fault-tolerance
  - HCI (layout, communication, correctness etc)

43

Formal Methods, Verification, Validation

Verification vs. Validation

- Verification:
  "Are we building the system right"
  - The system should conform to its specification
- Validation:
  "Are we building the right system"
  - The system should do what the user really requires

45

Formal Methods

Introduction

- Formal methods – use of mathematical techniques in the specification, design and analysis of hardware and software
- Many of the problems associated with the development of safety-critical systems are related to deficiencies in specification

47

Specification

- Typically written in natural language
  - Suspectible to misunderstanding
  - Impossible to avoid misinterpretations
  - Question about completeness and consistency
- Assessment of correctness, completeness or consistency requires good understanding of specification and requirements

48

Semi-formal Requirements/Specification

- Requirements should be unambiguous, complete, consistent and correct.
- Natural language has the interpretation possibility. More accurate description needed.
- Using pure mathematic notation – not always suitable for communication with domain expert.
- Formalised Methods are used to tackle the requirement engineering. (Structured text, formalised English).

49

Specification

- Many techniques
- Formalized techniques:
  - CASE tools
  - Graphic/diagrammatic methods

50

Formal Methods

- Based on formal languages
  - Very precise rules
- System (formal) specification languages
  - Can only assist!
  - Main advantage: automated tests
    - Requirements → spec → design
    - Possibility to *prove*

51

Method Selection Criteria

- Good expressiveness
- Core of the language will seldom or never be modified after its initial development, it is important that the notation fulfils this criterion.
- Established/accepted to use with Safety Critical Systems
- Possibility of defining subset/coding rules to allow efficient automatic processing by tools.
- Support for modular specifications – basic support is expected to be needed.
- Temporal expressiveness
- Tool availability

52

Formal Specification Languages

- These languages involve the explicit specification of a state model - system's desired behaviour with abstract mathematical objects as sets, relations and functions.
  - VDM (Vienna Development Method ISO standardised).
  - Z-language
  - B-Method

53

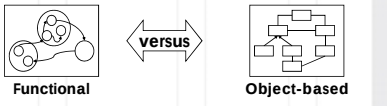
Modelling Requirements

- Models needed for communicating with domain experts (simulation)
- Automatic verification (model checker, theorem proving)

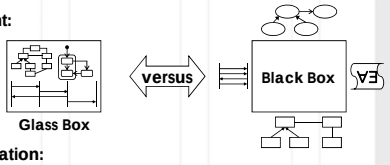
54

Some Modeling Styles

Decomposition:



View point:



Representation:

55

Formal Methods

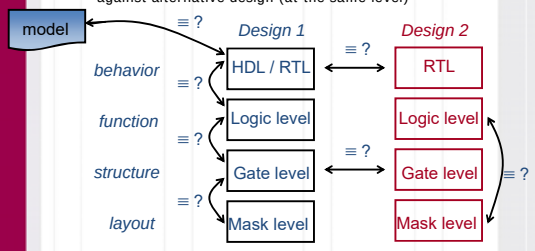
- Formal methods have been used for safety and security-critical purposes during last decades for e.g:
  - Certifying the Darlington Nuclear Generating Station plant shutdown system.
  - Designing the software to reduce train separation in the Paris Metro.
  - Developing a collision avoidance system for United States airspace.
  - Assuring safety in the development of programmable logic controllers.
  - Developing a water level monitoring system.
  - Developing an air traffic control system.

56

Verification

Verification

- Design verification = ensuring correctness of the design
  - against its implementation (at different levels)
  - against alternative design (at the same level)



58

Verification Methods

- Deductive verification
- Model checking
- Equivalence checking

Formal Verification

- Simulation - performed on the model
- Emulation, prototyping – product + environment
- Testing - performed on the actual product (manufacturing test)

59

Formal Verification

- Deductive reasoning (theorem proving)
  - uses axioms, rules to prove system correctness
  - no guarantee that it will terminate
  - difficult, time consuming: for critical applications only
- Model checking
  - automatic technique to prove correctness of concurrent systems: digital circuits, communication protocols, etc.
- Equivalence checking
  - check if two circuits are equivalent
  - OK for combinational circuits, unsolved for sequential

60

Why Formal Verification

- Need for reliable hardware validation
- Simulation, test cannot handle all possible cases
- Formal verification conducts exhaustive exploration of all possible behaviors
  - compare to simulation, which explores some of possible behaviors
  - if correct, all behaviors are verified
  - if incorrect, a counter-example (proof) is presented

61

Theorem Proving

- Formal methods
  - Formally, mathematically describe the system (hardware or software)
  - Formally, mathematically describe the properties you want to verify/validate (i.e. specifications)
    - Using available tools, mathematically PROVE the system will always exhibit the desired properties
- Do not have to use the same language to describe the system and the properties
  - calculus-based languages, logic based languages, temporal languages, etc.

62

Model Checking

- Algorithmic method of verifying correctness of (finite state) concurrent systems against temporal logic specifications
  - A practical approach to formal verification
- Basic idea
  - System is described in a formal model
    - derived from high level design (HDL, C), circuit structure, etc.
  - The desired behavior is expressed as a set of properties
    - expressed as temporal logic specification
  - The specification is checked against the model

63

Model Checking

- How does it work
  - System is modeled as a state transition structure (Kripke structure)
  - Specification is expressed in propositional temporal logic (CTL formula)
    - asserts how system behavior evolves over time
  - Efficient search procedure checks the transition system to see if it satisfies the specification

64

Model Checking

- Characteristics
  - searches the entire solution space
  - always terminates with YES or NO
  - relatively easy, can be done by experienced designers
  - widely used in industry
  - can be automated
- Challenges
  - state space explosion – use symbolic methods, BDDs
- History
  - Clark, Emerson [1981] USA
  - Quielle, Sifakis [1980's] France

65

Model Checking - Tasks

- Modeling
  - converts a design into a formalism: state transition system
- Specification
  - state the properties that the design must satisfy
  - use logical formalism: temporal logic
    - asserts how system behavior evolves over time
- Verification
  - automated procedure (algorithm)

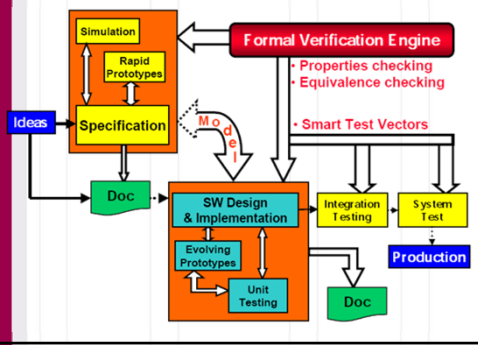
66

Model Checking - Issues

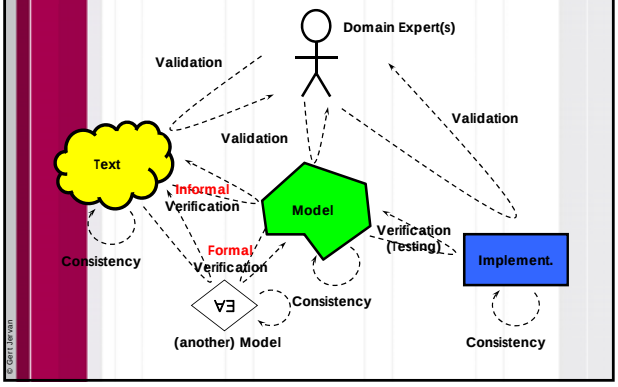
- Completeness
  - model checking is effective for a given property
  - impossible to guarantee that the specification covers all properties the system should satisfy
  - writing the specification - responsibility of the user
- Negative results
  - incorrect model
  - incorrect specification (false negative)
  - failure to complete the check (too large)

67

Verified software process



68

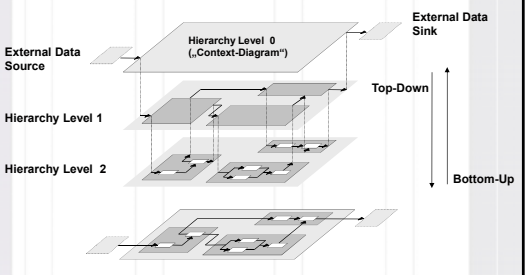


Functional Decomposition

- Functional decomposition breaks down complex systems into a hierarchical structure of simpler parts.
- Breaking a system into smaller parts enables users to understand, describe, and design complex systems.
- Functional decomposition consists of the following steps:
  - Define the system context.
    - This will help define the system boundaries.
  - Describe the system in terms of high-level functions and their interfaces.
  - Refine the high-level functions and partition them into smaller, more specific functions.

70

Functional Decomposition

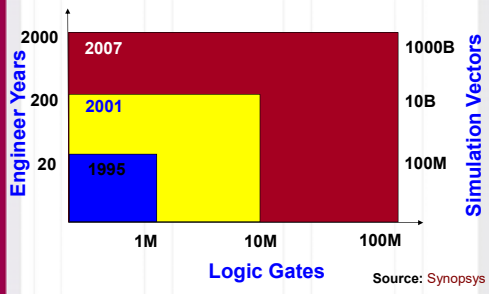


Hierarchical Structured Activity Chart

71

Validation

Functional Validation of SoC Designs



Source: Synopsys

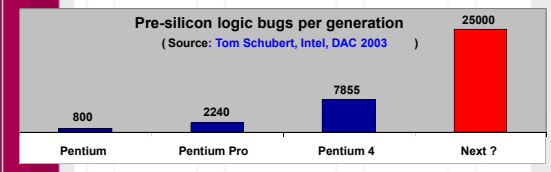
71% of SOC re-spins are due to logic bugs

Source: G. Spirakis, keynote address at DATE 2004

73

Functional Validation of Microprcessors

- Functional validation is a major bottleneck
  - Deeply pipelined complex microarchitectures



- Logic bugs increase at 3-4 times/generation
  - Bugs increase (exponential) is linear with design complexity growth.

74

The Validation Challenge

- Microprocessor validation continues to be driven by the economics of Moore's Law
  - Each new process generation doubles the number of transistors available to microprocessor architects and designers
  - Some of this increase is consumed by larger structures (caches, TLB, etc.), which have no significant impact to validation
  - The rest goes to increased complexity:
    - Out-of-order, speculative execution machines
    - Deeper pipelines
    - New technologies (Hyper-Threading, 64-bit extensions, virtualization, security, …
    - Multi-core designs
  - Increased complexity => increased validation effort and risk

High volumes magnify the cost of a validation escape

75

Microprocessor Design Scope

- Typical lead CPU design requires:
  - 500+ person design team:
    - logic and circuit design
    - physical design
    - validation and verification
    - design automation
  - 2-2½ years from start of RTL development to A0 tapeout
  - 9-12 months from A0 tapeout to production qual (may take longer for workstation/server products)

One design cycle = 2 process generations

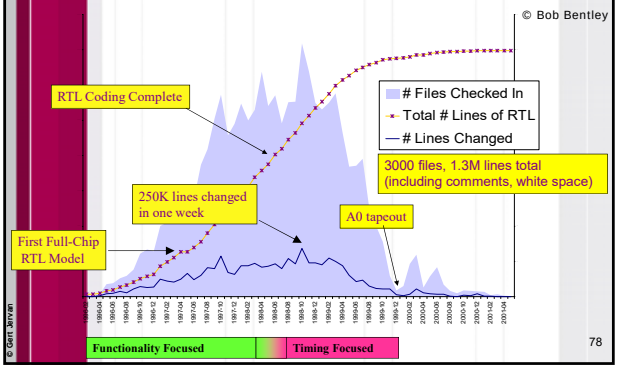
76

Pentium® 4 Processor

- RTL coding started: 2H'96
  - First cluster models released: late '96
  - First full-chip model released: Q1'97
- RTL coding complete: Q2'98
  - "All bugs coded for the first time!"
- RTL under full ECO control: Q2'99
- RTL frozen: Q3'99
- A-0 tapeout: December '99
- First packaged parts available: January 2000
- First samples shipped to customers: Q1'00
- Production ship qualification granted: October 2000

77

RTL – A Moving Target



78

RTL validation environment

- RTL model is MUCH slower than real silicon
  - A full-chip simulation with checkers runs at ~20 Hz on a Pentium® 4 class machine
  - A computer farm containing ~6K CPUs running 24/7 to get tens of billions of simulation cycles per week
  - The sum total of Pentium® 4 RTL simulation cycles run prior to A0 tapeout < 1 minute on a single 2 GHz system
- Pre-silicon validation has some advantages …
  - Fine-grained (cycle-by-cycle) checking
  - Complete visibility of internal state
  - APIs to allow event injection
- … but no amount of dynamic validation is enough
  - A single dyadic extended-precision (80-bit) FP instruction has O(10**50) possible combinations
  - Exhaustive testing is impossible, even on real silicon

79

How do you verify a design with…

- 42 million transistors
- 1 million lines of RTL code
- 600 – 1000 people working on it
- A 3-year design time
- Daily design changes

80

How do you verify a design which has bugs like this??

- The FMUL instruction, when the rounding mode is set to “round up”, incorrectly sets the sticky bit when the source operands are:
  src1[67:0] = X*2i+15 + 1*2i
  src2[67:0] = Y*2j+15 + 1*2j
  where i+j = 54 and {X,Y} are integers

81

And the answer is…

- Hire 70+ validation engineers
- Buy several thousand compute servers
- Write 12,000 validation tests
- Run up to 1 billion simulation cycles per day for 200 days
- Check 2,750,000 manually-defined properties
- Find, diagnose, track, and resolve 7,855 bugs
- Apply formal verification with 10,000 proofs to the instruction decoder and FP units
  - This found that obscure FMUL bug!

82

Pentium 4 Validation - Staffing

- 10 people in initial “nucleus” from previous project
- 40 new hires in 1997
- 20 new hires in 1998

83

P4 Validation Environment

- Hardware
  - IBM RS/6000 workstations (0.5-0.6Hz full processor model)
  - Pentium III Linux systems (3-5Hz full processor model)
  - Computing pool of “several thousand” systems
- Simulation statistics
  - About 1 million lines of code in SRTL model
  - 5-6 billion clock cycles simulated / week
  - 200 billion total clock cycles simulated overall

84

Cluster-Level Testing

- Divide overall design into 6 “clusters” + microcode
  - Develop “cluster testing environments” (CTEs) to validate each cluster separately (e.g. floating point, memory)
  - Then validate using full processor model
- Advantages of the approach
  - Controllability - control behavior at microarchitecture level
  - Early validation possible for each cluster
  - Decoupled validation possible for each cluster

85

Other Validation Features

- Extensive validation of power-reduction logic
- Code coverage and code inspections a major part of methodology
- Formal verification used for Floating Point & Instruction Decode Logic

86

Power Reduction Validation

- Power consumption was a big concern for Pentium 4
  - Need to stay within the cost-effective thermal envelope for desktop systems at 1.5+ GHz
- Extensive clock gating in every part of the design
- Mounted a focused effort to validate that:
  - Committed features were implemented as per plan
  - Functional correctness was maintained in the face of clock gating
  - Changes to the design did not impact power savings
- ~12 person years of effort, 5 heads at peak
- Fully functional on A-step silicon, measured savings of ~20W achieved for typical workloads

87

Formal Verification in P4 Validation

- Based on model checking
  - Given a finite-state concurrent system
  - Express specifications as temporal logic formulas
  - Use symbolic algorithms to check whether model holds
- Constructed database 10,000 “proofs”
- Over 100 bugs found
- 20 were “high quality” bugs not likely to be found by simulation
- Example errors: FADD, FMUL

88

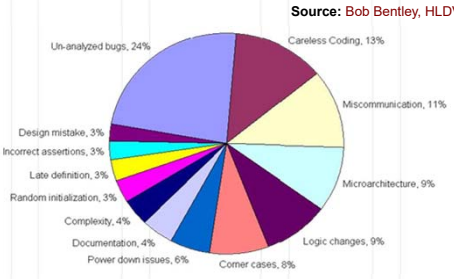
Validation Results

- 5809 bugs identified by simulation
  - 3411 bugs found by cluster-level testing
  - 2398 found using full-chip model
- 1554 bugs found by code inspection
- 492 bugs found by formal verification
- Largest sources of bugs: memory cluster (25%)

89

Pentium® 4 Bugs Breakdown

Source: Bob Bentley, HLDVT 2002



Micro-architectural complexity is a major contributor

90

Methodology drivers

- Regression
  - RTL is "live", and changes frequently until the very last stages of the project
  - Model checking automation at lower levels allows regression to be automated and provides robustness in the face of ECOs
- Debugging
  - Need to be able to demonstrate FV counter-examples to designers and architects
  - Designers want a dynamic test that they can simulate
  - Waveform viewers, schematic browsers, etc. can help to bridge the gap
- Verification in the large
  - Proof design: how do we approach the problem in a systematic fashion?
  - Proof engineering: how do we write maintainable and modifiable proofs?

91

Other Challenges

- Dealing with constantly-changing specifications
  - Specification changes are a reality in design
  - Properties and proofs should be readily adapted
  - How to engineer agile and robust regressions?
- Protocol Verification
  - This problem has always been hard
  - Getting harder (more MP) and more important (intra-die protocols make it more expensive to fix bugs)
- Verification of embedded software
  - S/W for large SoCs has impact beyond functional correctness (power, performance, …)
  - Not all S/W verification techniques apply because H/W abstraction is less feasible
  - One example is microcode verification

92

Verification

- There is a separate course: IAF0620 - Verification of Digital Systems (autumn semester)

93

Questions?