

INTERNATIONAL
TECHNOLOGY ROADMAP
FOR
SEMICONDUCTORS
2006 UPDATE

DESIGN

THE ITRS IS DEvised AND INTENDED FOR TECHNOLOGY ASSESSMENT ONLY AND IS WITHOUT REGARD TO ANY COMMERCIAL CONSIDERATIONS PERTAINING TO INDIVIDUAL PRODUCTS OR EQUIPMENT.

TABLE OF CONTENTS

Scope	1
Overall Challenges	2
Design Technology Challenges	3
Design Verification UPDATED	8
Design Verification –Challenges UPDATED	13
Design Verification—Solutions UPDATED	18
Additional Design Technology Requirements	24

LIST OF FIGURES

Figure 20	System Level Design Potential Solutions WAS	6
Figure 20	System Level Design Potential Solutions IS	6
Figure 22	Verification Technology Landscape UPDATED	13
Figure 23	Design Verification Potential Solutions WAS	18
Figure 23	Design Verification Potential Solutions IS	19

LIST OF TABLES

Table 12	Overall Design Technology Challenges	2
Table 13a	System Level Design Requirements—Near-term Years UPDATED	3
Table 13b	System Level Design Requirements—Long-term Years UPDATED	4
Table 14	Correspondence between Requirements and Solutions	7
Table 15a	Logic/Circuit/Physical Design Technology Requirements—Near-term Years	8
Table 15b	Logic/Circuit/Physical Design Technology Requirements—Long-term Years	8
Table 16a	Design Verification Requirements—Near Term NEW	11
Table 16b	Verification Requirements—Long Term NEW	12
Table NEW	Verification: Correspondence between Requirements and Solutions	20
Table 17a	Design for Test Technology Requirements—Near-term Years	21
Table 17b	Design for Test Technology Requirements—Long-term Years	22
Table 18a	Design-for-Manufacturability—Near-term Years UPDATED	23
Table 18b	Design-for-Manufacturability—Long-term Years UPDATED	23
Table 19	Near-term Breakthroughs in Design Technology for AMS	24
Table 20	Additional Design Technology Requirements	24
Table 21	Design Technology Improvements and Impact on Designer Productivity	25

DESIGN

SCOPE

After going through a major overhaul in the 2005 version, the 2006 design chapter update now features a full quantitative design technology roadmap. This year's update has focused primarily on providing meaningful updates of some of the figures, dates, and challenge items provided, including moderate revisions of the System-Level and Verification Sections and minor revisions of the rest of the sections. An increasing number of sections includes a table that relates challenges and solutions. Although a one-on-one relationship is usually not warranted, it is quite helpful in certain parts of the design flow. The 2007 version of this chapter will continue in this direction while increasingly accounting for alternative integration methods that add on to Moore's Law (heterogenous systems, system-in-packege (SIP), etc.).

OVERALL CHALLENGES

Table 12 Overall Design Technology Challenges

<i>Challenges ≥ 32 nm</i>	<i>Summary of Issues</i>
Design productivity	System level: high level of abstraction (HW/SW) functionality spec, platform based design, multi-processor programmability, system integration, AMS co-design and automation Verification: executable specification, ESL formal verification, intelligent testbench, coverage-based verification Logic/circuit/layout: analog circuit synthesis, multi-objective optimization
Power consumption	Logic/circuit/layout: dynamic and static (leakage), system and circuit, power optimization
Manufacturability	Performance/power variability, device parameter variability, lithography limitations impact on design, mask cost, quality of (process) models ATE interface test (multi-Gb/s), mixed-signal test, delay BIST, test-volume-reducing DFT
Reliability	Logic/circuit/layout: MTTF-aware design, BISR, soft-error correction
Interference	Logic/circuit/layout: signal integrity analysis, EMI analysis, thermal analysis
<i>Challenges < 32 nm</i>	<i>Summary of Issues</i>
Design productivity	Complete formal verification of designs, complete verification code reuse, complete deployment of functional coverage Tools specific for SOI and non-static-logic, and emerging devices Cost-driven design flow Heterogeneous component integration (optical, mechanical, chemical, bio, etc.)
Power consumption	SOI power management
Manufacturability	Uncontrollable threshold voltage variability Advanced analog/mixed signal DFT (digital, structural, radio), “statistical” and yield-improvement DFT Thermal BIST, system-level BIST
Reliability	Autonomic computing, robust design, SW reliability
Interference	Interactions between heterogeneous components (optical, mechanical, chemical, bio, etc.)

*ATE—automatic test equipment BISR—built-in self repair BIST—built-in self test DFT—design for test
EMI—electromagnetic interference ESL—Electronic System-level Design HW/SW—hardware/software
MTTF—mean time to failure SOI—silicon on insulator*

DESIGN TECHNOLOGY CHALLENGES

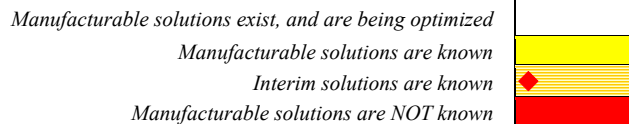
Table 13a System Level Design Requirements—Near-term Years *UPDATED*

Year of Production		2005	2006	2007	2008	2009	2010	2011	2012	2013
DRAM ½ Pitch (nm) (contacted)		80	70	65	57	50	45	40	36	32
<i>Design Reuse</i>										
WAS	Design block reuse [1] % to all logic size	32%	33%	35%	36%	38%	40%	41%	42%	44%
IS	Design block reuse [1] % to all logic size	32%	33%	35%	36%	38%	40%	41%	42%	44%
<i>Platform Based Design</i>										
WAS	Available platforms [2] Normalized to 100% in the start year [3]	96%	88%	83%	83%	75%	67%	60%	55%	50%
IS	Available platforms [2] Normalized to 100% in the start year [3]	100%	91%	87%	83%	75%	70%	60%	55%	52%
WAS	Platforms supported [4] % of platforms fully supported by tools [5]	3%	6%	10%	25%	35%	50%	57%	64%	75%
IS	Platforms supported [4] % of platforms fully supported by tools [5]	3%	6%	10%	25%	35%	50%	57%	64%	75%
<i>High Level Synthesis</i>										
	Accuracy of high level estimates (performance, area, power, costs) [6] % versus measurements	53%	56%	60%	63%	66%	70%	73%	76%	80%
<i>Reconfigurability</i>										
WAS	SOC reconfigurability [7] % of SOC functionality reconfigurable	23%	26%	28%	28%	30%	35%	38%	40%	42%
IS	SOC reconfigurability [7] % of SOC functionality reconfigurable	23%	26%	28%	28%	30%	35%	38%	40%	42%
<i>Analog/Mixed Signal</i>										
WAS	Analog automation [8] % versus digital automation [9]	12%	14%	17%	17%	24%	24%	27%	30%	32%
IS	Analog automation [8] % versus digital automation [9]	12%	14%	17%	17%	24%	24%	27%	30%	32%
	Modeling methodology, description languages, and simulation environments [10] % versus digital methodology [11] [12]	53%	55%	58%	60%	62%	65%	67%	70%	73%

4 Design

Table 13b System Level Design Requirements—Long-term Years *UPDATED*

Year of Production		2014	2015	2016	2017	2018	2019	2020
DRAM ½ Pitch (nm) (contacted)		28	25	22	20	18	16	14
Design Reuse								
WAS	Design block reuse [1] % to all logic size	46%	48%	49%	51%	52%	54%	55%
IS	Design block reuse [1] % to all logic size	46%	48%	49%	51%	52%	54%	55%
Platform Based Design								
WAS	Available platforms [2] Normalized to 100% in the start year [3]	46%	43%	42%	39%	36%	33%	32%
IS	Available platforms [2] Normalized to 100% in the start year [3]	48%	45%	43%	40%	37%	35%	32%
WAS	Platforms supported [4] % of platforms fully supported by tools [5]	80%	85%	90%	92%	94%	95%	97%
IS	Platforms supported [4] % of platforms fully supported by tools [5]	80%	85%	90%	92%	94%	95%	97%
High Level Synthesis								
	Accuracy of high level estimates (performance, area, power, costs) [6] % versus measurements	83%	86%	90%	92%	94%	95%	97%
Reconfigurability								
WAS	SOC reconfigurability [7] % of SOC functionality reconfigurable	45%	48%	50%	53%	56%	60%	62%
IS	SOC reconfigurability [7] % of SOC functionality reconfigurable	45%	48%	50%	53%	56%	60%	62%
Analog/Mixed Signal								
WAS	Analog automation [8] % versus digital automation [9]	35%	38%	40%	43%	46%	50%	52%
IS	Analog automation [8] % versus digital automation [9]	35%	38%	40%	43%	46%	50%	52%
	Modeling methodology, description languages, and simulation environments [10] % versus digital methodology [11] [12]	76%	78%	80%	83%	86%	90%	92%



Notes for Table 13a and b

[1] This requirement is not unique to system level design, but it is also a requirement to design, in general (See the SOC-PE Productivity Trends table in the System Drivers chapter).

DEFINITION

The portion of a design, which is not newly developed but which is composed of pre-existing components.

RATIONALE

Reuse is one of the main factors which drive design productivity, and it is one of the key concepts behind system level design.

The reuse in year n from a particular reference year can be calculated according to the following formula

$$reuse(n) = 1 - (1 - reuse) \cdot ((1 + pgrowth)^n / (1 + cgrowth)^n)$$

with

reuse: reuse in reference year

pgrowth: (expected) mean annual productivity growth rate, excluding the effect of reuse

cgrowth: (expected) mean annual growth rate of design complexity

and assuming, that the size of the design staff as well as the design cycle time stay constant during the considered period of time.

The rationale for the formula is that the gap between the productivity growth (without effect of reuse) and the complexity growth has to be filled by reuse, if the technological progress shall be fully exploited in SOC design.

[2] DEFINITION

A platform is a specific combination of system components that support specific application areas (e.g. wireless, automotive, consumer electronics/multimedia, Small Office Home Office (SOHO) networks, etc.). System components are one or more processors, (real time) operating system, communication infrastructure, memory, customizable analog and digital logic, and virtual sockets for new logic. Basic functionality for the application area is provided by a number of already integrated components. System differentiation is achieved by integration of few new components either in hardware or software.

RATIONALE

Platform based design is an important driver for design productivity, since it highly promotes reuse. In addition, system level specifications require platforms to which they can be mapped.

[3] Different platforms are expected to converge in the future, owing to advances in manufacturing technology and higher integration densities, wherefore the total number of platforms is expected to decrease.

[4] DEFINITION

(Full) Support for a particular platform means an integrated development environment that supports and automates architectural exploration, HW/SW partitioning, architectural/platform mapping, HW/SW co-verification, performance/area/power/costs trade-offs, HW and SW synthesis and HW/SW interface synthesis for that platform.

RATIONALE

A high degree of automation is a key to the success of system level design.

[5] Although there already exist some solutions today for some aspects of platform based modeling, a full integration has not yet been achieved.

[6] DEFINITION

The degree to which the estimated results match the measurements on the fabricated IC.

RATIONALE

For high-level synthesis techniques a high accuracy of estimations is essential in order to be able to deliver high quality synthesis results that meet user constraints, such as minimum performance, maximum area, et cetera.

Take into account the quality is much different for the different aspects, keep one row for now

[7] DEFINITION

The portion of a SOC, or a design, respectively, in terms of functionality implemented either in SW or HW that is reconfigurable.

RATIONALE

The growing system complexity will make it impossible to ship designs without errors in the future. Hence, it is essential to be able to fix errors after fabrication. In addition, reconfigurability increases reuse, since existing devices can be reprogrammed to fulfill new tasks.

[8] DEFINITION

Automation grade in analog design.

RATIONALE

Analog components are to be found in most electronic systems today and analog/mixed-signal design is an essential and important part of electronic design. Hence, as for digital design a high automation grade in analog design on as many levels as possible is required in the future, if the design productivity growth shall be maintained or increased.

[9] To which degree does the degree of automation in analog design match the degree of automation in digital design.

[10] DEFINITION

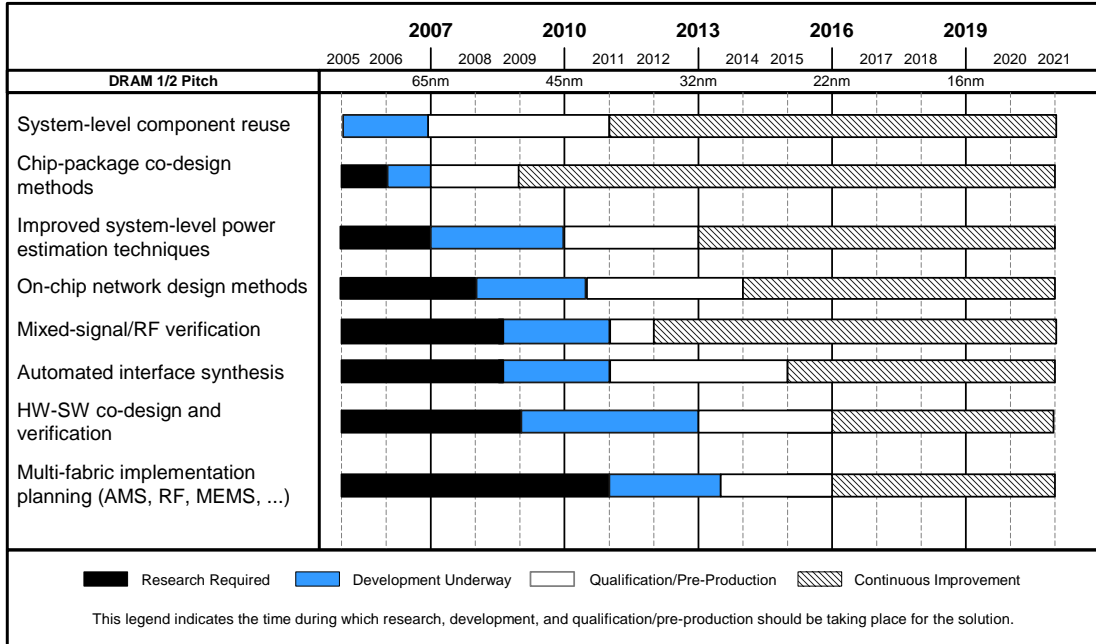
To which degree do analog methodology, description languages, and simulation environments match the maturity of their digital counterparts.

RATIONALE

As digital and analog design become nearly equally important on system level, analog design and modeling methodologies are required to reach a similar maturity as their digital counterparts in order to be able to keep up the productivity growth in system level design.

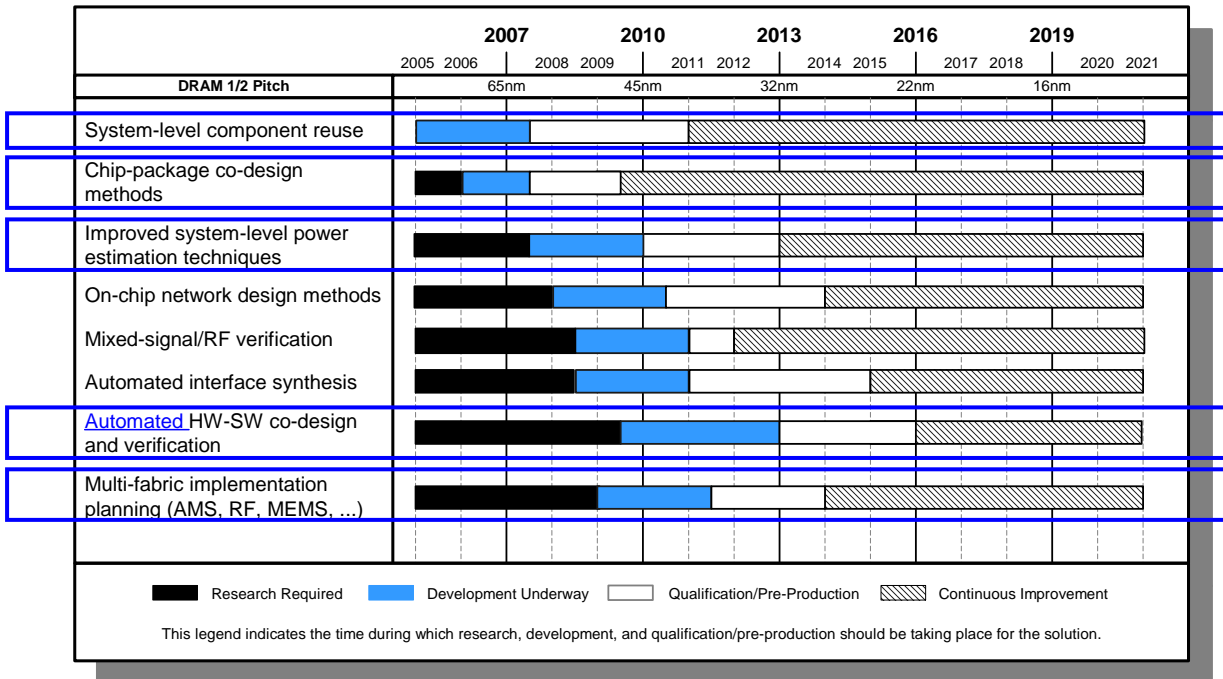
[11] To which degree do analog methodology, description languages, and simulation environments match the maturity of their digital counterparts?

[12] This number is based on a survey carried out in 2004.



MEMS—micro-electronmechanical systems

Figure 20 System Level Design Potential Solutions WAS



MEMS—micro-electronmechanical system

Figure 20 System Level Design Potential Solutions IS

Table 14 Correspondence between Requirements and Solutions

Design block reuse	System-level component reuse	The larger and more complex the components that can be reused, the larger the overall design reuse expected to be.
	On-chip network design methods	Standardized communication structures and interfaces support reuse, since IPs with standardized interfaces can be easily integrated and exchanged, and also the communication structure itself is reused.
Available platforms	Multi-fabric implementation planning (AMS, RF, MEMS, ...)	Enables to integrate different fabrics on the same die or in the same package (SIP), and, hence, should enable to reduce the absolute number of platforms, since what needed different platforms before could now be integrated on one.
Platforms supported	Automated interface synthesis	Automated interface synthesis is one building block to an integrated synthesis flow for whole platforms.
	Automated HW-SW co-design and verification	An obvious requirement for an integrated, platform-based system development approach
Accuracy of high level estimates	Improved system-level power estimation techniques	While area and performance estimation on higher levels has already been a topic for several years and made some progress, system-level power estimation is a rather innovative but important topic and has to catch up.
	Chip-package co-design methods	Allows taking effects of packaging, for instance impact on the timing, into account for estimations on higher levels.
SOC reconfigurability	On-chip network design methods	On-chip networks are flexible and reconfigurable communication structures.
Analog automation	Multi-fabric implementation planning (AMS, RF, MEMS, ...)	Multi-fabric implementation planning for AMS and RF components seems to be one building block to analog automation.
Modelling methodology, description languages, and simulation environments	Mixed-signal/RF verification	As for digital design, verification is of major importance and verification is increasingly the most important and most time-consuming activity in the design flow.

8 Design

Table 15a Logic/Circuit/Physical Design Technology Requirements—Near-term Years

<i>Year of Production</i>	2005	2006	2007	2008	2009	2010	2011	2012	2013
<i>DRAM ½ Pitch (nm) (contacted)</i>	80	70	65	57	50	45	40	36	32
Asynchronous global signaling % of a design driven by handshake clocking	5%	5%	7%	11%	15%	17%	19%	20%	22%
Parameter uncertainty %-effect (on sign-off delay)	5%	6%	6%	8%	10%	11%	11%	12%	14%
Simultaneous analysis objectives # of objectives during optimization	4	4	4	5	6	6	6	6	7
MTTF contribution reliability factor	1	1.1	1.2	1.3	1.4	1.6	1.7	1.8	1.9
Circuit families # of circuit families in a single design	2	2	3	3	4	4	4	4	4
Analog content synthesized % of a design	10%	13%	15%	16%	17%	18%	19%	20%	23%
Leakage # times per device	2	3	4	6	8	9.5	11	12	14

Table 15b Logic/Circuit/Physical Design Technology Requirements—Long-term Years

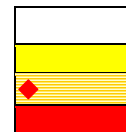
<i>Year of Production</i>	2014	2015	2016	2017	2018	2019	2020
<i>DRAM ½ Pitch (nm)(contacted)</i>	28	25	22	20	18	16	14
Asynchronous global signaling % of a design driven by handshake clocking	20%	25%	30%	30%	30%	35%	40%
Parameter uncertainty %-effect (on sign-off delay)	15%	18%	20%	20%	20%	22%	25%
Simultaneous analysis objectives # of objectives during optimization	8	8	8	8	8	8	8
MTTF contribution reliability factor	2	2.1	2.2	2.3	2.5	2.6	2.7
Circuit families # of circuit families in a single design	4	4	4	4	4	4	4
Analog content synthesized % of a design	25%	28%	30%	35%	40%	45%	50%
Leakage # times per device	16	24	32	32	32	32	32

Manufacturable solutions exist, and are being optimized

Manufacturable solutions are known

Interim solutions are known

Manufacturable solutions are NOT known



DESIGN VERIFICATION UPDATED

Design verification is the task of establishing that a given design accurately implements the intended behavior. Today, the verification of modern computing systems has grown to dominate the cost of system design, often with limited success, as designs continue to be released with latent bugs. In fact, in many application domains, the verification portion has become the predominant component of a project development, in terms of time, cost and human resources dedicated to it. In current projects verification engineers outnumber designers, with this ratio reaching two or three to one for the most complex designs. Design conception and implementation are becoming mere preludes to the main activity of verification.

This situation is the result of two processes. First, the functional complexity of modern designs is increasing at a breathtaking pace. Design size is growing exponentially with Moore's Law. In the worst case, functional complexity, as

measured by the number of distinct system configurations that must be verified, may be subjected to a doubly exponential blow-up.¹ Second, the historically greater emphasis on other aspects of the design process has produced enormous progress (automated tools for logic synthesis, place-and-route, and test pattern generation, etc.), leaving verification as the bottleneck. Without major breakthroughs, verification will be a non-scalable, show-stopping barrier to further progress in the semiconductor industry.

The overall trend from which these breakthroughs will emerge is the shift from ad-hoc verification methods to more structured, formal processes. The mainstream methodology used in industry today attempts to verify the functionality of a system design by repeatedly building models, simulating them on an ad hoc selection of vectors, and then patching any bugs that happen to be triggered. To this end, logic simulation techniques are predominant in the industry because they can generate simulation vectors at a very high rate. However, the coverage of the tests generated is usually very low, with the result that even months of simulation provide little confidence in the correctness of the design and, when design errors are found, the error analysis phase is daunted by very long and complex bug traces. Additionally, these traditional techniques require a lot of effort from the engineering team to direct the verification activity towards specific design areas of critical quality, or with low-coverage, etc. Nonetheless, these techniques have such high inertia in existing development processes that the cost to transition to alternative methodologies is very high. Because of this, the recent availability of formal and semi-formal techniques has seen only limited adoption in the industry at large.

More structured verification approaches organize the verification effort by generating a model of the system's intended behavior² (commonly called "golden model") and comparing the outcomes of the design's simulation with this model. In addition, coverage metrics are collected to attempt to evaluate the level of confidence in the correctness of the design-under-verification. It is common to organize the verification activities hierarchically, by addressing first individual components (generally developed by a single designer) within a chip, then the entire chip, and eventually the full system³, so that bugs that are contained within a single unit can be addressed and solved earlier and more easily. Today, hierarchical approaches have still coarse granularity so that the added complexity at each stage of verification presents often many difficult challenges. Moreover, the most complex aspect of this methodology is the verification of the communication interface between components, or units, and late or escaped bugs are often found in complex unverified interactions between units. Finally, there is a growing interest for formal and semi-formal verification techniques—today IC development teams are exploring new methodologies where mainstream validation is complemented by semi-formal verification, using a handful of commercially available tools. Formal methods span multiple aspects of verification, from a formalized verification process, to a formal notation for specification and verification, to formal solvers and proof techniques.

In tackling the complexity of very complex systems, and in particular, systems-on-a-chip, a strongly hierarchical verification methodology seems to be necessary, along with the development of specific methods and tools associated with each level of the hierarchy. In this context, the design and implementation pyramid can be summarized roughly as follows: 1) Module level (structural, dataflow, and behavioral module description and synthesis) where the verification tools must prove the correctness of systems in the range of 10K to 500K logic gates; 2) Sub-System level (for the most part, this is the result of structural composition of modules) producing systems with 100K to 10M logic gates; and 3) System/Full chip level (obtained by composing sub-systems) producing designs of 1M to more than 100M logic gates. System level verification entails also the verification of the interaction between the hardware and the software applications operating on it, adding an entire new dimension to the verification task. In a hierarchical verification approach, different verification techniques would be used at each design level. At the module level, the goal is to produce high-quality modules through the simulation of hand-crafted tests and the formal verification of properties and assertions. For sub-systems, verification would entail a mix of property verification for the interface and simulation with constrained random pattern generation. Property verification at the sub-system level could be achieved in the future, for instance,

¹ There are many ways of reasoning to arrive to this conclusion. For example, a new design that requires doubling the number of transistors on a chip is likely to also double the number of latches on the chip, which likely means roughly squaring the number of reachable states of the design. This analysis assumes that the correct behavior can be verified by examining the set of reachable states of the system, or by similar computation. If verifying correct behavior requires reasoning over sequences of states, the computational challenge will be even worse.

² Golden models are commonly described using high-level languages, such as SystemC or ANSI C.

³ Due to the complexity of full system simulation hardware emulation is sometimes used instead of simulation, particularly for large-market designs, where the additional costs can be easily absorbed. Hardware emulation buys several orders of magnitude of performance improvement in "simulation" speed, providing an invaluable aid to verification. It also enables an early start to system integration and software development. An emulation system, however, still simulates vectors one-at-a-time, so it can only deliver a constant-factor improvement, and cannot provide a scalable, long-term solution to the verification problem.

10 Design

through the definition of aggressive automatic design-abstraction techniques, and of automatic techniques for the refinement of such abstractions driven by counter-examples, which would allow proving sub-system properties while posing minimum demands on the corresponding engineering effort. At the system level, simulation-based assertion checking paired with coverage metrics seems the most appropriate among the techniques available. The verification of the embedded software running on the system is still performed separately, commonly starting long after the beginning of hardware verification, only after a first hardware prototype is available. However, due to the much faster growth trends for embedded software complexity compared to hardware complexity, new solutions need to be developed which allow the software effort to have access to a system's prototype at a much earlier stage, and that address specifically the verification of the hardware/software interface. Technological progress depends on the development of rigorous and efficient methods to achieve high-quality verification results through the development of appropriate coverage metrics, verification tools with a high ratio of coverage over resources required, and techniques to ease the burden of debugging a design, once a bug is found.

Post-silicon validation focuses on the validation of silicon devices after tape-out and before customer shipping. This is an area of verification that had grown rapidly in the past decade, mostly due to the large complexity of silicon-based systems, which in turn leads to many potential flaws that can only be exposed after system integration. The goals of post-silicon validation range from detecting logic and design bugs, which should manifest in each manufactured part, to expose electrical and process-related issues, which usually manifest only in a fraction of the components and, finally, to identify and exclude rare random manufacturing defects. The process involves building full prototype systems with the manufactured components, and run software test programs, which can range anywhere from embedded software to high level applications. Two key characteristics of post-silicon validations are as follows: 1) the high execution speed that enables the verification team to evaluate a system over several order-of-magnitudes more cycles than it is possible in pre-silicon simulation. Typical projects would execute a system for trillions of cycles during post-silicon validation. 2) the limited visibility inside the silicon parts. In fact, internal signals cannot be probed any longer as they could in a pre-silicon simulation environment, making test validation and debugging much more challenging operations. These inspection tasks are usually performed by accessing and probing the device's external signals and register interfaces using specialized tools, such as logic analyzers and oscilloscopes.

Table 16a and b reports a set of key quantitative requirements in design verification, necessary to support the trends of design complexity in the next technology generations. The requirements evaluate both the ability to guarantee the correctness of a design and the amount of effort spent in verification within a development project.

Table 16a Design Verification Requirements—Near Term *NEW*

<u>Year of Production</u>	<u>2005</u>	<u>2006</u>	<u>2007</u>	<u>2008</u>	<u>2009</u>	<u>2010</u>	<u>2011</u>	<u>2012</u>	<u>2013</u>
<u>DRAM 1/2 Pitch (nm) contacted</u>	<u>80</u>	<u>70</u>	<u>65</u>	<u>57</u>	<u>50</u>	<u>45</u>	<u>40</u>	<u>35</u>	<u>32</u>
<u>Productivity</u>									
Design size which can be verified by 1 engineer-year (in Millions of transistors - based on an SoC design and a 10 people engineering team)[1]		<u>6.1</u>	<u>8.0</u>	<u>10.5</u>	<u>13.7</u>	<u>18.0</u>	<u>23.6</u>	◆ <u>30.9</u>	◆ <u>40.6</u>
<u>Methodology</u>									
Design errors exposed using formal or semi-formal verification (versus simulation, expressed as a percentage)		<u>2.5</u>	<u>5.0</u>	<u>7.5</u>	<u>10.0</u>	<u>12.5</u>	<u>15.0</u>	<u>17.5</u>	<u>20.0</u>
Effort spent on system level verification: software, hardware and electrical effects (as a percentage)		<u>10.0</u>	<u>11.6</u>	<u>13.3</u>	<u>15.0</u>	<u>16.6</u>	<u>18.3</u>	<u>20.0</u>	<u>21.7</u>
Portion of the design specification formalized for verifiability (as a percentage)		<u>13.8</u>	<u>17.5</u>	<u>21.3</u>	<u>25.0</u>	<u>28.8</u>	◆ <u>32.5</u>	◆ <u>36.3</u>	◆ <u>40.0</u>
<u>Bugs</u>									
Escape rate: bugs found after first tape out. (per each 100,000 lines of design code)		<u>7</u>	<u>6</u>	<u>6</u>	<u>6</u>	<u>5</u>	<u>5</u>	<u>5</u>	<u>5</u>
Bugs found after system integration until tape-out (per each 100,000 lines of design code)		<u>56</u>	<u>63</u>	<u>69</u>	<u>75</u>	<u>81</u>	<u>88</u>	<u>94</u>	◆ <u>100</u>
<u>Reuse</u>									
Portion of the verification infrastructure (e.g. testbeds, coverage, checkers) which is newly developed (versus reused components and acquired IP) (as a percentage)[2]		<u>76.8</u>	<u>73.5</u>	<u>70.3</u>	<u>67.0</u>	<u>63.8</u>	<u>60.5</u>	◆ <u>57.3</u>	◆ <u>54.0</u>
Portion of the verification infrastructure which is acquired from third parties (i.e., verification IP) (as a percentage)[2]		<u>12.9</u>	<u>15.9</u>	<u>18.8</u>	<u>21.8</u>	<u>24.7</u>	<u>27.6</u>	◆ <u>30.6</u>	◆ <u>33.5</u>
<u>Functional coverage</u>									
Percentage of the design for which verification quality is evaluated through functional coverage		<u>43.4</u>	<u>46.9</u>	<u>50.3</u>	<u>53.8</u>	<u>57.2</u>	<u>60.6</u>	<u>64.1</u>	<u>67.5</u>
Coverage goal density (expressed as number of coverage goals for each million of transistors of the design) [3]		<u>1000</u>	<u>1333</u>	<u>1667</u>	<u>2000</u>	<u>2333</u>	<u>2667</u>	◆ <u>3000</u>	◆ <u>3333</u>

12 Design

Table 16b Verification Requirements—Long Term NEW

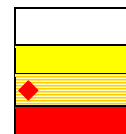
<u>Year of Production</u>	<u>2014</u>	<u>2015</u>	<u>2016</u>	<u>2017</u>	<u>2018</u>	<u>2019</u>	<u>2020</u>
<u>DRAM 1/2 Pitch (nm) contacted</u>	<u>28</u>	<u>25</u>	<u>22</u>	<u>20</u>	<u>18</u>	<u>16</u>	<u>14</u>
<u>Productivity</u>							
Design size which can be verified by 1 engineer-year (in Millions of transistors - based on an SoC design and a 10 people engineering team)[1]	◆53.4	◆70.2	92.4	121.7	160.4	211.5	279.1
<u>Methodology</u>							
Design errors exposed using formal or semi-formal verification (versus simulation, expressed as a percentage)	22.5	25.0	27.5	30.0	32.5	35.0	37.5
Effort spent on system level verification: software, hardware and electrical effects (as a percentage)	23.3	25.0	◆26.7	◆28.3	◆30.0	31.7	33.3
Portion of the design specification formalized for verifiability (as a percentage)	◆43.8	◆47.5	51.3	55.0	58.8	62.5	66.25
<u>Bugs</u>							
Escape rate: bugs found after first tape out. (per each 100,000 lines of design code)	◆4	◆4	◆4	3	3	3	2
Bugs found after system integration until tape-out (per each 100,000 lines of design code)	◆106	◆113	◆119	125	131	138	144
<u>Reuse</u>							
Portion of the verification infrastructure (e.g. testbeds, coverage, checkers) which is newly developed (versus reused components and acquired IP) (as a percentage)[2]	◆50.8	47.5	44.3	41.0	37.8	34.5	31.3
Portion of the verification infrastructure which is acquired from third parties (i.e., verification IP) (as a percentage)[2]	◆36.4	◆39.4	42.3	45.3	48.2	51.1	54.1
<u>Functional coverage</u>							
Percentage of the design for which verification quality is evaluated through functional coverage	◆70.9	◆74.4	◆77.8	81.3	84.7	88.1	91.6
Coverage goal density (expressed as number of coverage goals for each million of transistors of the design) [3]	◆3667	4000	4333	4667	5000	5333	5667

Manufacturable solutions exist, and are being optimized

Manufacturable solutions are known

Interim solutions are known

Manufacturable solutions are NOT known



Notes for Table 16a and b

All values, except for the first row (productivity) are linearly increasing or decreasing trends, where the initial values in 2006 are derived from industry survey data, and the final values are estimated.

[1] This requirement considers the SOC design productivity requirement specified in Table 20 – SOC logic Mtx per designer-year, and divides it by the percentage of design effort spent in verification, so that to obtain the amount of logic which can be verified in a year. The percentage of the effort spent in verification is a linear trend starting at 70% in 2006 and ending at 50% in 2020.

[2] The reuse data uses the equation: new verification infrastructure + reused + 3rd party verification IP = 100%. The table reports analytical values for the new verification infrastructures and acquired 3rd party IP, while the estimated size of reused components can be derived.

[3] That is, number of assertions or checkers inserted in a design or referring to an aspect of the design, for each corresponding million of transistors of logic generated when synthesizing that design.

The values in the table refer to a new development design project, in contrast with a proliferation within a family of designs. The productivity of verification must scale proportionally with the complexity growth of a design, in order to sustain similar time-to-markets. The classification of verification software between formal verification and simulation in time is going to morph into hybrid and semi-formal verification solutions, particularly in the long term horizon. We are

estimating, within those integrated hybrid solutions, which fraction will resemble a simulation-based approach versus a formal verification technique. To support this productivity growth, the methodology of verification must increasingly integrate more formal and semi-formal engines to complement simulation-based techniques. In addition, due to the increasing impact of software and mixed-signal components, and the importance of electrical effects due to the reduction in feature sizes, the verification of the interaction between heterogeneous components and the system as a whole will require an increasing fraction of the overall verification effort. Finally the shift towards a more structured approach to verification demands an effort towards the formalization of a design specification, which in turns leads to improved automation of the entire verification process.

We divide the verification development into three segments: 1) code newly developed specifically for the design being verified, 2) code acquired from third party (a.k.a. verification IP), and 3) code reused from previous designs within the same company. The table reports the first two components under “Reuse”, the third can be computed by difference from the other two rows. The last two columns in the table estimated the growing importance of functional coverage. While traditional coverage techniques focus on code and state coverage, functional coverage captures more directly the relevant aspects of a design functionality that need to be verified, but requires additional engineering effort because of its domain-specific character. The need to quantify the progress of verification in the near and long-term future demands improved deployment of this technique. Although the growing expertise among engineering teams in developing and reusing functional coverage is growing, the table indicates that a pervasive deployment is needed in the future.

DESIGN VERIFICATION—CHALLENGES **UPDATED**

Many of the most important challenges for verification are relevant to most or all system drivers. In the near term, the primary issues are centered on making formal and semi-formal verification techniques more reliable and controllable. In particular, major advances in the capacity and robustness of formal verification tools are needed, as well as meaningful metrics for the quality of verification. In the longer term, issues focus mainly on raising the level of abstraction and broadening the scope of formal verification. These longer-term issues are actually relevant now, although they have not reached the same level of crisis as other near-term challenges. In general, all of the verification challenges apply to SOC. MPUs present a distinct set of issues, both because of their leading-edge complexity, and because of the unique economics of an incredibly complex design family that is produced in incomparably high volumes. As a result, different domain-specific verification challenges and opportunities exist, both in the near- and long-term.

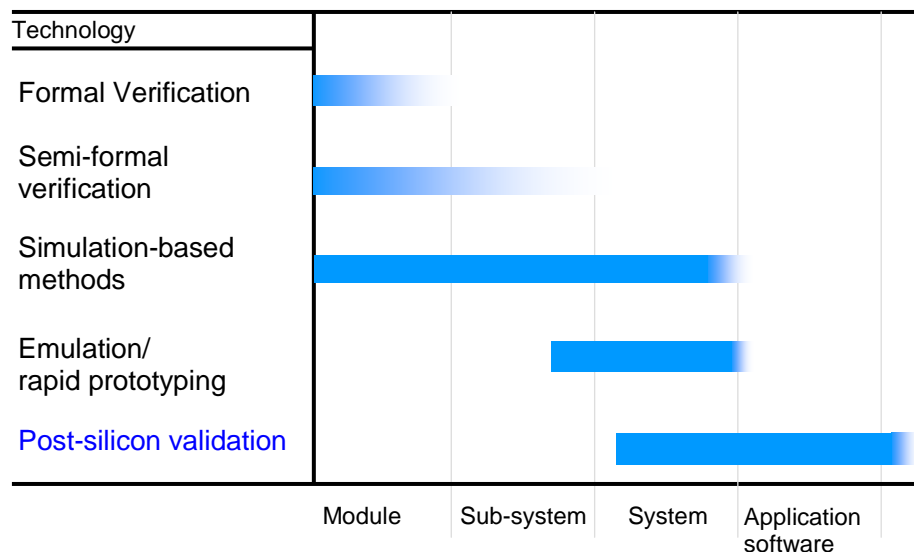


Figure 22 Verification Technology Landscape **UPDATED**

Capacity—Figure 22 relates the current landscape of available verification tools and design granularity. The horizontal axis presents increasing design complexity, from module components, to sub-systems, multiple modules connected together providing some high-level functionality, to systems, complete systems-on-a-chip integrating multiple functionalities on a single die, and to the application software running on the system. The vertical axis indicates the

14 Design

verification techniques available. Conventional simulation-based methods⁴ span a broad range; they are practically used for anything from module to full-system verification. However, the quality of the results varies. For small modules it is possible to run many simulation vectors in a short time and thus achieve a fairly good coverage of the design under verification. At the system level the performance of simulation is very low, since the complexity of the algorithms involved is linear on the design size and the number of simulation vectors. Thus, for very complex systems, only a few simulation vectors can be run within a reasonable amount of time, and the fraction of design coverage they provide is even much smaller if we consider the large complexity of the system that is being simulated. Formal verification can achieve very high coverage, granted that a broad set of properties are in place targeting the various aspects of the system. The downside of this technique is its limited scalability that usually cannot go beyond the module level. Semi-formal verification attempts to blend formal and simulation-based techniques. As the figure shows, it presents generally better capacity, sometimes at the cost of lower design coverage. For very complex systems, and when the project involves software development, software tools are insufficient to provide any confidence in the correctness of the system. Solutions used at this level include FPGA-based rapid prototyping and/or a silicon production of the hardware. The challenge is to create new solutions that together provide high coverage at all levels of complexity. The two orthogonal methodologies of formal verification and simulation in use today have both important downsides—while formal tools can only handle small to medium size designs, simulation-based tools can simulate designs of almost arbitrary complexity, but they provide a vanishingly small coverage even with extremely long simulation times. Emulation and rapid hardware prototyping perform several orders of magnitude faster than software logic simulators, thus providing the ability to achieve higher coverage. However, the improvement is only a constant factor and does not scale at the same rate as design complexity. Post-silicon validation is concerned with the overall correct functionality of the system composed of silicon hardware and application software, and as such it operates at the top level of design complexity integrating all components together.

The remainder of this section overviews and discusses the main challenges faced in design verification today. *Robustness*—A crucial aspect of current verification solutions that is not shown in Figure 22 is their reliability. On one hand, simulation-based methods are fairly predictable, because their execution time per simulation vector scales linearly with design complexity. Their performance, thus, will linearly decrease as we move towards the most complex design levels. Emulation techniques present a similar trend, while silicon prototypes are consistently reliable. On the other hand, formal verification techniques depend on highly temperamental heuristics, in order to cope with the complexity of the verification problem. For any given pair of design and verification algorithm, even an expert can be hard-pressed to determine whether the verification algorithm will complete. Common measures of problem size, such as transistor, gate, or latch counts, correlate only vaguely with formal verification complexity; it is fairly easy to find designs with less than one hundred latches that defy all known verification methods, as well as designs with thousands of latches that can be verified easily. Such unpredictability is not acceptable in a production environment. A crucial verification challenge is to make the verification process more robust. This can take the form of either improved heuristics for the verification algorithms, or improved characterization of the difficulty of verifying a given design, leading to methodologies for easy-to-verify designs.

Verification metrics—An important near-term verification challenge is the need to quantify the quality of the verification effort. In particular, a meaningful notion of coverage is needed. Multiple types of coverage metrics are available, each with their own benefits and limitations:

- Code coverage is concerned with measuring which fraction of the design source code has been stimulated during simulation (for example, lines of code, expression coverage). Variants of line coverage are concerned with covering all components of expressions in the code, or all cases in switch statements, etc. The limitation of line coverage is in its projecting a dynamic simulation execution to a static source code, thus missing many potential problems that could be still present because of all the possible paths of execution available during simulation.
- Structural coverage, such as state or pair arc coverage. State coverage focuses on the finite state machines of the design and measures which states have been covered by simulation. However, it is generally not possible to

⁴ “Conventional simulation” here refers to verification techniques based on simulating possible behaviors of the system one-at-a-time. “Formal verification” refers to any techniques, such as symbolic simulation, symbolic trajectory evaluation, model checking, and theorem proving, that provide the effect of an exhaustive analysis of all possible behaviors of the system. An occasionally useful distinction can be drawn between theorem-proving approaches, which tend to provide the greatest expressive and analytical power at the expense of requiring substantial human expertise, versus the other approaches, which tend to trade off theoretical power for greater automation. “Semi-formal” refers to a broad range of techniques that attempt to handle larger designs by sacrificing complete, formal coverage, usually by blending techniques from formal verification and conventional simulation.

consider the whole design as a single finite state machine, rather only a few machines are considered by coverage. The downside is that while the coverage provides good results for single machines, combinations of states resulting from products of multiple machines are not considered.

- Functional coverage targets each of the design's functionalities. Since these differ for each distinct design, the specific definition needs to be provided by the verification team based on the design's specification. The quality of the result is thus dependent also on the quality of the coverage definition.

Code and structural coverage metrics quantify aspects of the designs that do not necessarily imply its correctness. Functional coverage potentially could achieve this goal of measuring the fraction of a design's functionality that has been verified. However, the challenge is in the definition of a good functional coverage. In the current landscape, there is a lack and a need at the same time for a common foundation metric for functional coverage in a unified environment, so that achieving a certain level of coverage in a design could have meaning beyond the specific coverage used in that particular design. Moreover, there is no general methodology in designing functional coverage units; an abstract fundamental model to use as a guideline during the development is needed and could support the development methodology. Ultimately, there is a need for a defect model for functional bugs, much like there is a defect model for testing, to support the development of coverage and a unified metric for verification.

Software— The verification of complex systems, such as systems-on-a-chip entails the verification of the hardware components, of the hardware/software interface, and of the application software running on the system. The software components of an SOC can be divided into 1) application software, 2) hardware-independent layer, such as an operating system, which controls the execution of the application, and 3) lower hardware-dependent software, such as drivers, etc. Because in these systems the software layer can provide much of the functionality, a major challenge in SOC verification is how to verify the software and the hardware/software interface. Presently, software development is not as rigorous as hardware development in terms of design reviews, tools for analysis, and testing. Software is intrinsically harder to verify: it has more complex, dynamic data and a much larger state space. The most common software verification technique in use today is “on-chip-verification,” which entails running the software on a production version of the hardware components. While it allows very fast simulation, as it is required by the intrinsic complexity of software, its downside is that software verification can only start very late in the design cycle. Classical formal techniques for software verification are still too labor-intensive to be widely applicable for SOC, that is, systems with such large state spaces, and require very aggressively abstracted models of the software application. The verification of the hardware/software interface is a challenge on its own, since it requires verifying the two domains together. To make this task more manageable, there is a need for techniques that provide proper abstractions of the interface activity, solutions to check the correctness of the abstracted driver layers, and assertion checking tools for drivers' invariants at the non-abstract level. The near-term challenge will be to develop techniques that allow verification of even elementary and low-level pieces of software. The longer-term challenge will be to develop robust verification methods for software, and hardware/software interfaces, as well as an understanding of design-for-verifiability as applied to software.

Reuse—Pre-designed IP blocks promise to allow assembling SOCs of unprecedented complexity in a very short time. The major challenge is in developing the corresponding verification methodology to allow rapid verification of a system assembled from pre-designed (and pre-verified) blocks. Key issues are how to rigorously and completely describe the abstract behavior of an IP block; how to describe the environmental constraints assumed by the IP block; and how to exploit the hierarchy to simplify verification. Some IP components have started to ship with associated verification IPs, from standard protocols verification IPs, to abstract models for general IP blocks, protocol checkers to check environmental constraints surrounding the block, to transaction generators. However, these are still preliminary attempts, while the need is for consistent availability of verification IPs associated with any IP block in order to develop a reuse methodology. Besides verification components associated with IP blocks, there is some availability of independent verification IPs, such as environment generators for specific protocols. Near-term progress will most likely occur for standardized IP interconnects, such as on-chip buses, but the general problem for arbitrary IP block interfaces must be eventually solved.

Specialized verification methodology—The biggest issue in design verification is that all currently known algorithmic solutions are running out of capacity with respect to the designs being developed today. The only foreseeable way to overcome this issue in the short term is with an adequate verification methodology. Current trends in this direction include coverage-driven verification, both for simulation-based verification and semi-formal verification, more in use today than in the past; specification documents in formal notation, that make easily available the set of formal properties to be verified in the implementation; and coverage model templates. Many challenges are still to be solved to obtain a sufficiently robust and complete methodology. For instance, there is a need for ways to obtain consistent abstraction

16 Design

techniques of design components, interfaces, etc., that do not drop key aspects of the design in the abstraction, where, obviously, these aspects depend on the context where the abstraction is used. Formal specifications are starting to be developed; however, a major limitation is the completeness of such specifications—a challenge that becomes even more compelling in a world where different IP components in the same system are developed by completely unrelated design teams. The acceptance of new verification methodologies is progressing slowly; even the basic deployment of formal properties in a design is a challenge since it often requires a global understanding of some specific aspect of a design, which involves additional effort on the development team. Finally, the verification methodology is ultimately an evolving target that can only provide a risk reduction to the development, not a guarantee of correctness through a well-defined recipe.

Specialized design-for-verifiability—A few specialized activities in the area of design for verifiability are already foreseeable, enabling more effective verification. For instance, facilities for software and/or hardware debug can be put in silicon. In this direction there is preliminary work being done on self-checking processors, in which a small watchdog processor or a network of distributed hardware checkers verifies the correct execution of the main processor. For mixed-signal designs, the insertion of loop-back modes to bypass the analog portion of the design allows to verify the system as a fully digital design. The use of synchronizers in multi-threaded systems, and in particular at the software level, forces the tasks to not proceed independently past the synchronization points, creating checkpoints for verification and reducing the searchable state space. In MPU designs synchronizers would reduce the complexity of verifying speculative execution systems. The challenges in this area will be the development and the adoption of design-for-verifiability techniques for a few major domains. Efforts are also made in developing design methodologies by incremental refinement to produce systems that are correct-by-construction; however, it is not clear how automatic the refinement process can be made, while, on the other hand, manual intervention is a potential source of design errors.

New kinds of concurrency—As MPU designs become more complex, new kinds of concurrency become important. Already, many of the bugs that elude verification relate to cache coherence and other concurrency issues. New designs greatly complicate the verification process by increasing the level of concurrency via techniques such as chip-level multiprocessing and on-chip cache coherence protocols, and simultaneous multithreading. In the future, new kinds of concurrency both at the intra-processor level and in multi-processor systems, or in other hardware context will present difficult challenges to verification. There is a need for new models of failure to grasp this additional level of complexity. The solution will probably require a mix of hardware and software techniques to reduce the complexity of the interactions and make the concurrent protocols verifiable.

In the long term, the issues will be centered on the need of improved verification productivity, verification of extremely complex designs, and the need to address verification of heterogeneous systems. Below some of the most important challenges are presented.

Design for verifiability—As the solutions to the near-term challenges produce understandings of what is easy or hard to verify and how design errors occur, the longer-term challenge arises of how to codify that understanding into producing easy-to-verify designs. Without design-for-verifiability, it is unlikely that verification will be tractable for the designs envisioned beyond 2007. Major changes in methodology may be required, and some performance degradation is likely. A useful analogy is to sequential testability, where the computational intractability of sequential automatic test pattern generation (ATPG) has resulted in near-universal adoption of scan-based testing.

Higher levels of abstraction—As design moves to a level of abstraction above register transfer level (RTL), verification will have to keep up. The challenges will be to adapt and develop verification methods for the higher-levels of abstraction, to cope with the increased system complexity made possible by higher-level design, and to develop means to check the equivalence between the higher-level and lower-level models. This longer-term challenge will be made much more difficult if decisions about the higher-level of abstraction are made without regard for verification (e.g., languages with ill-defined or needlessly complex semantics, or a methodology relying on simulation-only models that have no formal relationship to the RTL model).

Specification for verifiability—How to specify the desired behavior of a design is a continuing challenge in design verification. Current available notations for specification are not powerful enough to approach this problem in a generalized way. A deeper understanding of what makes a specification clear or opaque, modifiable or intractable will be needed to guide development of languages that are used to specify ever more complex designs. For instance, there is a need for automatic ways to check the self-consistency of a specification document, so that different specifications do not state conflicting requirements. In addition, specific training is needed for designers to use these specification notations and to be able to develop formal specifications consistently.

Verification in presence of non-digital effects—To date, design verification has mainly focused on the discrete behavior of digital systems. The dual challenges of silicon complexity and system complexity will force future verification efforts to analyze a broader class of aspects. The complexity of silicon integrated circuit systems is making the clean, digital abstraction of a VLSI system increasingly precarious. Analog electrical effects will impact performance and, eventually, functionality. The existing simulation methodology (SPICE) for analyzing these effects is too slow, and may become unreliable as smaller devices become increasingly sensitive to process variations. In this direction there is preliminary work being done on architectural simulation of microprocessors, in which the high-level architectural simulator interacts with a low-level context-specific simulator to acquire metrics such as timing and voltage, which are then fed back for overall system evaluation with minimal performance impact. In the long term, formal techniques will be needed to verify these issues at the boundary of analog and digital, treating them as hybrid systems.⁵ Similarly, at the highest-levels of design, system complexity dictates that future verification tasks will likely require specifying and verifying analog and probabilistic behaviors (such as quality of service guarantees in a network processor). Thus, there will be the challenges of hybrid systems and probabilistic verification.

Heterogeneous systems—The development of new technologies that are placed side-by-side to a digital design in a silicon die presents a whole set of new challenges. Examples are microelectromechanical system (MEMS), electro-optical devices, and electro-biological devices. These new components will require modeling of both the interface between the digital portion and the non-digital components and a proper abstraction of the non-digital system behavior in order to still be able to verify the digital portion of the system.

Analog/Mixed-signal—Today, analog systems are mostly verified through classic systems analysis tools for continuous systems, through system modeling and analysis in the frequency domain. The bulk of verification happens in the post-design phase, by verifying test dies with analog lab equipment. Mixed-signal designs undergo separate verification activities for the digital and analog portions. In the future, mixed-signal systems will become a more relevant fraction of all silicon developments, bringing the development of proper verification methodologies in this arena to a critical level. The challenge in this area is in tying the verification of the analog portion of a system with its digital counterpart. One of the requirements in achieving this goal is in bridging the current performance gap between digital and analog simulation.

Soft failures—In the long term, there is a need to develop verification techniques that can provide information on a system's reliability in presence of soft errors. Research-level solutions are already available that can harden the storage elements of a design to be resilient to single event upset (SEU) faults. The common ground among these solutions is the use of specialized storage elements that can detect if the input to the storage cell changes shortly after a value has been latched, indicating the presence of an SEU in the cone of logic influencing the storage element.

Verification of redundancy—The quantifiable evaluation of the reliability of a system to transistor failures, manufacturing defects, etc., and the estimation of the level of redundancy needed to achieve a certain level of reliability is particularly important for safety-critical application. In the future, the increased complexity of electrical effects in a system performance and correctness will make these evaluations even more important, even for non safety-critical designs. Initial solutions have been proposed in research efforts to provide reliability to transistor failures and transient effect for general-purpose embedded microprocessors at extremely low area and performance costs.

⁵ Hybrid systems have both complex discrete behavior (e.g., finite-state machines) as well as complex continuous behavior (e.g., differential equation models). The discipline borrows techniques from both discrete formal verification as well as classical control theory.

DESIGN VERIFICATION—SOLUTIONS UPDATED

We present here some of the solutions that are available or are being developed to address the challenges described in the previous section. Figure 23 summarizes some of the key directions to attack the verification crisis along with their expected availability to development teams.

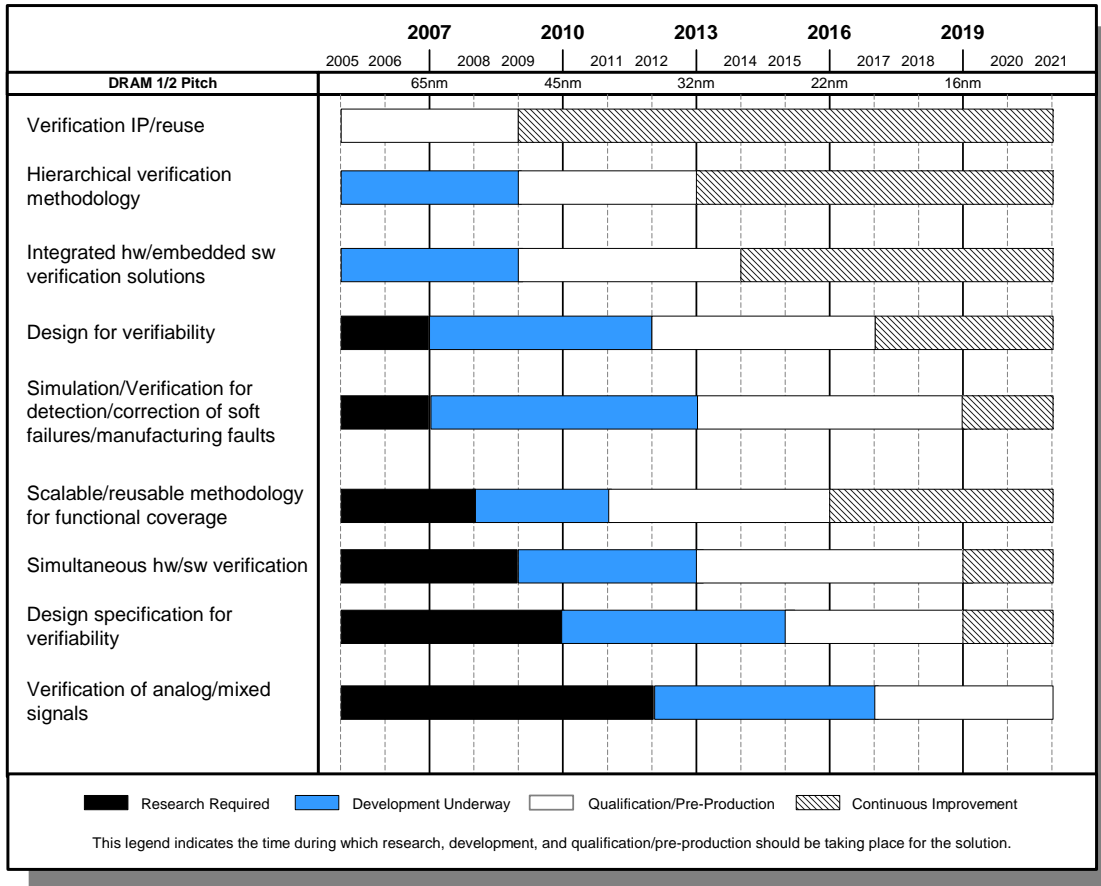


Figure 23 Design Verification Potential Solutions WAS

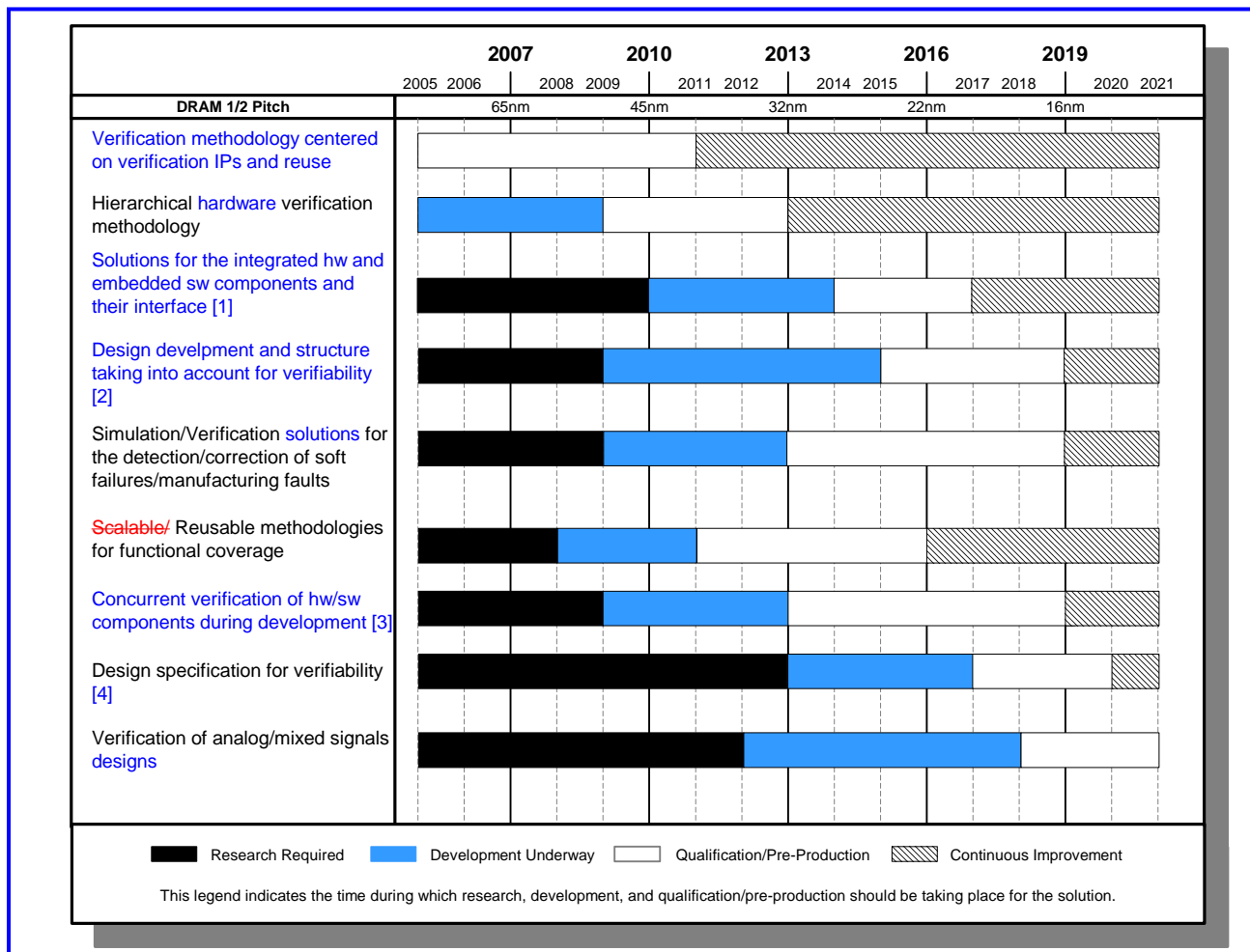


Figure 23 Design Verification Potential Solutions IS

ADDED—Notes for Figure 23:

[1] This entails verification techniques for high-level integrated hw/sw systems. Current design technologies are starting to automate the process of partitioning the functionalities of a system that should be implemented in hardware and those that should be implemented in software. Usually the result is a high level description of the system's components and whether they should be implemented in embedded software or in hardware. The complexity of such high level descriptions, however, is growing, and there is a need for solutions that provide validation of a system at this stage of development and, in particular, of the correctness of the high level interaction protocols.

[2] That is, techniques of developing a hardware design so that it is well-suited for verification.

[3] That is, verification solutions that address the correct interaction between the hardware layer and the embedded software layer. Current available solutions verify these two layers separately: The hardware layer is verified mostly through classic validation techniques. The embedded software is verified by running the software on a simulation model of the hardware (which is usually still under development). The lack of integrated solutions that can verify the two layers simultaneously creates the potential for escaped bugs that can only be exposed after tape-out, when software and hardware can be finally integrated. Because of the fast growing complexity trends of embedded software applications, this is becoming an increasingly pressing problem, requiring specialized solutions.

[4] That is, techniques to develop a specification (possibly in a formalized way) for a design, so that it is easy to verify that the specification itself does not contain contradictions or conflicting requirements.

Verification IPs are already available today and they are part of a growing methodology trends to boost the productivity of the process of verification. In the future this approach will be much more widespread, in parallel with the aggressive deployment of third party IPs in SOC designs. As discussed in the previous section, solutions structuring a hierarchical and integrated hardware/software methodology are being researched today and will start to become available in the next few years. The other solutions outlined in the figure are still at a conceptual stage today. Formal methodologies to develop easy-to-verify designs and to correct soft errors and defects have seen an initial exploration in the MPU domain,

20 Design

as mentioned in the previous section. Techniques for structural functional coverage and specification are even further from becoming mainstream solutions, however, they are crucial in solving the design-verification productivity gap.

Table NEW Verification: Correspondence between Requirements and Solutions

<u>Requirement</u>	<u>Solution</u>	
<u>Productivity of verification tasks</u>	<u>Verification methodology centered on verification IPs and reuse</u>	<u>Verification IPs and reuse reduce the amount of new verification development required in a project.</u>
	<u>Hierarchical hardware verification methodology</u>	<u>Structured methodologies improve the team productivity.</u>
	<u>Reusable methodologies for functional coverage development</u>	<u>Functional coverage is a time consuming task, which is specific for each distinct design. The development of reusability techniques is critical in increasing the verification productivity.</u>
	<u>Concurrent verification of hardware and software components during development</u>	<u>Advancing the verification of hardware in parallel with the one of software components can significantly shorten the time-to-market of a product. (in contrast with methodologies starting software verification only after first hardware prototype).</u>
<u>Formal and semi-formal verification centered methodology</u>	<u>Hierarchical hardware verification methodology</u>	<u>Enables the decomposition of the system into smaller blocks which are suitable to verification through formal techniques.</u>
-	<u>Design development and structure taking into account verifiability</u>	<u>Design for verifiability structures a design so that it is easier to prove relevant aspects. The addition of hardware structures can further simplifies design-time verification tasks.</u>
<u>Methodologies for system-level verification</u>	<u>Verification methodology centered on verification IPs and reuse</u>	<u>Verification IPs components enable an early start on system-level verification.</u>
	<u>Solutions for the integrated verification of hardware and embedded software components and their interface</u>	<u>Directly provide solutions for effective system-level verification.</u>
<u>Portion of design specification formalized for verifiability</u>	<u>Design specification formalized for verifiability</u>	<u>Formal languages and methodologies to support the formal specification of a design.</u>
<u>Escape rate after tape-out</u>	<u>Design structure taking into account verifiability</u>	<u>Development of hardware structures (checker-like) which can be used to detect and correct a system entering an escaped erroneous configuration after customer shipment.</u>
<u>System integration bug rate</u>	<u>Verification of analog and mixed-signal designs</u>	<u>Limit the bug rate due to analog effect.</u>
	<u>Simulation and verification solutions for the detection and correction of soft failures and manufacturing faults</u>	<u>Manufacturing faults occurring in post-silicon are detected in system-level integration. Techniques which can detect and correct electrical, transient and defects limit the effort dedicated to expose and correct these problems.</u>
	<u>Hierarchical hardware verification methodology</u>	<u>Supports managing system-level complexity through decomposition.</u>
<u>Functional coverage</u>	<u>Reusable methodologies for functional coverage development</u>	<u>Functional coverage is specific for each distinct design. Reusable solutions can boost the effort spent in coverage development and quality of the results.</u>

Table 17a Design for Test Technology Requirements—Near-term Years

Year of Production	2005	2006	2007	2008	2009	2010	2011	2012	2013
DRAM ½ Pitch (nm) (contacted)	80	70	65	57	50	45	40	36	32
<i>System Driver: Analog/Mixed-signal/RF</i>									
1. All-digital DFT for analog/mixed-signal/RF circuits and systems % digital circuits in DFT implementations	30	35	40	45	50	55	60	60	60
2. Correlation of DFT results with existing specification-based test methods. % results correlated	30	35	40	45	50	55	60	60	60
3. Availability of fault/defect models for DFT-oriented test methods. % AMS/RF blocks with accepted fault models	20	20	25	30	35	40	45	50	55
<i>System Drivers: MPU/PE/DSP/</i>									
1. DFT coverage of digital blocks or subsystems. % blocks with DFT	60	60	70	70	70	75	75	75	80
2. DFT for delay test of critical paths. % paths covered	50	50	55	55	60	60	60	60	70
3. DFT for fault tolerance in logic blocks. % blocks with fault tolerance	40	40	40	40	45	45	50	50	55
<i>System Drivers: Memories</i>									
1. DFT for yield improvement.	85	85	85	90	90	90	90	95	95
<i>General SOC/SIP requirements</i>									
1. DFT-support for logic and other circuit repair (except memory). % blocks with repair	50	50	50	60	60	60	70	70	70
2. DFT re-use for performance calibration, and measurement purposes. % DFT circuits re-used	30	30	35	35	40	40	40	45	45
3. DFT impact on system performance (noise, power, sensitivity, bandwidth, etc.). % performance impact (aggregate figure of merit)	15	15	15	15	10	10	10	10	10
4. DFT efficacy in test volume reduction. reduction magnitude	2x	2x	5x	5x	5x	10x	10x	10x	20x
5. DFT / ATE interface standard, including DFT control via standard test access protocols. % of test interface standardized	40	40	45	45	50	50	60	60	70

Manufacturable solutions exist, and are being optimized

Manufacturable solutions are known

Interim solutions are known

Manufacturable solutions are NOT known

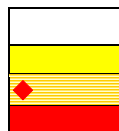


Table 17b Design for Test Technology Requirements—Long-term Years

Year of Production	2014	2015	2016	2017	2018	2019	2020
<i>DRAM ½ Pitch (nm)(contacted)</i>	28	25	22	20	18	16	14
<i>System Driver: Analog/Mixed-signal/RF</i>							
1. All-digital DFT for analog/mixed-signal/RF circuits and systems. % digital circuits in DFT implementations	60	80	85	90	90	100	100
2. Correlation of DFT results with existing specification-based test methods. % results correlated	60	80	85	90	90	100	100
3. Availability of fault/defect models for DFT-oriented test methods. % % AMS/RF blocks with accepted fault models	60	65	70	75	80	85	90
<i>System Drivers: MPU/PE/DSP/</i>							
1. DFT coverage of digital blocks or subsystems. % blocks with DFT	80	85	85	90	90	95	95
2. DFT for delay test of critical paths. % paths covered	70	70	80	80	90	90	100
3. DFT for fault tolerance in logic blocks. % blocks with fault tolerance	55	60	65	70	80	90	100
<i>System Drivers: Memory</i>							
1. DFT for yield improvement.	95	95	98	98	98	100	100
<i>General SOC/SIP requirements</i>							
1. DFT-support for logic and other circuit repair (except memory). % blocks with repair	80	80	80	90	90	100	100
2. DFT re-use for performance calibration, and measurement purposes. % DFT circuits re-used	50	50	60	60	70	70	70
3. DFT impact on system performance (noise, power, sensitivity, bandwidth, etc.). % performance impact (aggregate figure of merit)	10	10	5	5	5	5	5
4. DFT efficacy in test volume reduction. reduction magnitude	20×	20×	20×	50×	50×	50×	50×
5. DFT / ATE interface standard, including DFT control via standard test access protocols. % of test interface standardized	70	75	90	80	90	100	100

Manufacturable solutions exist, and are being optimized

Manufacturable solutions are known

Interim solutions are known

Manufacturable solutions are NOT known

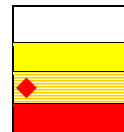


Table 18a Design-for-Manufacturability—Near-term Years *UPDATED*

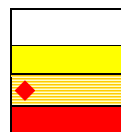
<i>Year of Production</i>	2005	2006	2007	2008	2009	2010	2011	2012	2013	<i>Driver</i>
<i>DRAM ½ Pitch (nm) (contacted)</i>	80	70	65	57	50	45	40	36	32	
Mask cost (\$m) from publicly available data	1.5	2.2	3.0	4.5	6.0	9.0	12.0	18.0	24.0	SOC
% V _{dd} Variability % variability seen at on-chip circuits	10%	10%	10%	10%	10%	10%	10%	10%	10%	SOC
% V _{th} variability Doping Variability impact on VTH	24%	29%	31%	35%	40%	40%	40%	58%	58%	SOC
% V _{th} variability Includes all sources	26%	29%	33%	37%	42%	42%	42%	58%	58%	SOC
IS % CD variability CD for now, might add doping later	12%	12%	12%	12%	12%	12%	12%	12%	12%	SOC
% circuit performance variability circuit comprising gates and wires	41%	42%	45%	46%	49%	50%	53%	54%	57%	SOC
% circuit power variability circuit comprising gates and wires	55%	55%	56%	57%	57%	58%	58%	59%	59%	SOC

Manufacturable solutions exist, and are being optimized

Manufacturable solutions are known

Interim solutions are known

Manufacturable solutions are NOT known

Table 18b Design-for-Manufacturability—Long-term Years *UPDATED*

<i>Year of Production</i>	2014	2015	2016	2017	2018	2019	2020	<i>Driver</i>
<i>DRAM ½ Pitch (nm)(contacted)</i>	28	25	22	20	18	16	14	
Mask cost (\$m) from publicly available data	36.0	48.0	72.0	96.0	144.0	192.0	288.0	SOC
% V _{dd} Variability % variability seen at on-chip circuits	10%	10%	10%	10%	10%	10%	10%	SOC
% V _{th} variability Doping Variability impact on VTH	81%	81%	81%	81%	112%	112%	112%	SOC
% V _{th} variability Includes all sources	81%	81%	81%	81%	112%	112%	112%	SOC
IS % CD variability CD for now, might add doping later	12%	12%	12%	12%	12%	12%	12%	SOC
% circuit performance variability circuit comprising gates and wires	58%	61%	62%	65%	66%	69%	69%	SOC
% circuit power variability circuit comprising gates and wires	59%	60%	60%	61%	61%	62%	62%	SOC

Manufacturable solutions exist, and are being optimized

Manufacturable solutions are known

Interim solutions are known

Manufacturable solutions are NOT known

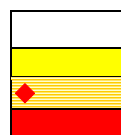


Table 19 Near-term Breakthroughs in Design Technology for AMS

Field of Breakthrough	2005 State-of-the-Art	2006/07	2008/09
Specification, validation, verification	Established AMS Hardware Description Languages	Multi-language support, AMS extension of HW/SW description languages for full system simulation	Complete specification-driven design flow; some specialized formal verification methods
Architectural design	Algorithm-oriented design (e.g., with Matlab/Simulink)	Language-based performance evaluation; closer coupling of architectural, block, and circuit level	Synthesizeable AMS description; power-aware HW/SW partitioning extended to AMS systems
Physical mixed A/D and RF design	Procedural layout generation, module generators for a few block types	Module generators for often re-used blocks, design centering, performance estimation	Synthesis: behavior to layout (at least for the most important building blocks)
Parasitics extraction, automated modeling, accelerated simulation	Electromagnetic immunity simulation works but is too complicated for broad usage	2D/3D model-based order reduction for interconnect systems and substrate effects on chip, thermal package modeling	New fault-tolerant circuit architectures, robustness against technology parameter variations; order reduction for all kinds of parasitics and antennas

ADDITIONAL DESIGN TECHNOLOGY REQUIREMENTS

Table 20 Additional Design Technology Requirements

Year of Production	2005	2006	2007	2008	2009	2012	2015	2018	Driver
DRAM ½ Pitch (nm)	80	70	65	57	50	36	25	18	
SOC new design cycle (months)	12	12	12	12	11	11	10	9	SOC
SOC logic Mtx per designer-year (10-person team)	3.3	4.3	5.4	7.4	10.6	24.6	73.4	113	SOC
SOC dynamic power reduction beyond scaling (X)	0.1	0.2	0.2	0.2	0.2	6	4.7	8.1	SOC
SOC standby power reduction beyond scaling (X)	2.4	3.4	5.1	6.4	8.73	18.8	44.4	232	SOC
% Test covered by BIST	25	30	35	40	45	60	75	90	MPU, SOC

Mtx—Million transistors

Manufacturable solutions exist, and are being optimized

Manufacturable solutions are known

Interim solutions are known

Manufacturable solutions are NOT known

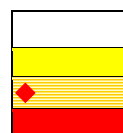


Table 21 Design Technology Improvements and Impact on Designer Productivity

<i>DT Improvement</i>	<i>Year</i>	<i>Productivity Delta</i>	<i>Productivity (Gates/Design-Year)</i>	<i>Cost of Component Affected</i>	<i>Description of Improvement</i>
None	1990		4K		
In-house place and route	1993	+38.9%	5.55K	PD Integration	Automated block placement and routing.
Engineer	1995	+63.6%	9.09K	Chip/circuit/PD Verification	Engineer than can pursue all required tasks to complete a design block, from RTL to GDSII.
Reuse—small blocks	1997	+340%	40K	Circuit/PD Verification	Blocks from 2,500–74,999 gates.
Reuse—large blocks	1999	+38.9%	56K	Chip/circuit/PD Integration Verification	Blocks from 75,000–1M gates.
IC implementation suite	2001	+63.6%	91K	Chip/circuit/PD Integration EDA support	Tightly integrated tool set that goes from RTL synthesis to GDSII through IC place and route.
RTL functional verification tool suite	2003	+37.5%	125K	SW development Verification	RTL verification tool (“cockpit”) that takes an ES-level description and partitions it into verifiable blocks, then executes verification tools on the blocks, while tracking and reporting code coverage.
Electronic system-level (ES-level) methodology	2005	+60%	200K	SW development Verification	Level above RTL, including both HW and SW design. It consists of a behavioral (where the system function has not been partitioned) and an architectural level (where HW and SW are identified and handed off to design teams).
Very large block reuse	2007	+200%	600K	Chip/circuit/PD Verification	Blocks >1M gates; intellectual-property cores
Homogeneous parallel processing	2009	+100–200%	1200K	Chip/circuit/PD Design and Verification	Many identical cores provide specialized processing around a main processor, which allows for performance, power efficiency, and high reuse
Intelligent test bench	2011	37.5%	2400K	Chip/circuit/PD Verification	Like RTL verification tool suite, but also with automation of the Verification Partitioning step.
Concurrent software compiler	2013	60%	3300K	Chip and Electronic System Design and Verification	Enables compilation and SW development in highly parallel processing SOCs
Heterogeneous massive parallel processing	2015	+100–200%	5278K	System Electronic Design and Verification	Each of the specialized cores around the main processor is not identical from the programming and implementation standpoint
System-level DA and executable specification	2017-19	+100–200%	10557K	System Electronic Design and Verification	Automates true electronic system design on- and off-chip for the first time, including heterogenous technologies.
Total		+264,000%			