Robust Intelligent Systems

Alfons Schuster
Editor

Robust Intelligent Systems

Springer

Editor
Alfons Schuster
School of Computing and Mathematics
University of Ulster at Jordanstown
Northern Ireland, UK
a.schuster@ulster.ac.uk

ISBN: 978-1-84800-260-9 e-ISBN: 978-1-84800-261-6
DOI: 10.1007/978-1-84800-261-6

British Library Cataloguing in Publication Data
A catalogue record for this book is available from the British Library

Library of Congress Control Number: 2008931892

Printed on acid-free paper

Springer Science+Business Media
springer.com

Preface

Our time recognizes robustness as an important, all-pervading feature in the world around us. Despite its omnipresence, robustness is not entirely understood, rather difficult to define, and, despite its obvious value in many situations, rather difficult to achieve.

One of the goals of this edited book is to report on the topic of robustness from a variety and diverse range of fields and perspectives. We are interested, for instance, in fundamental strategies nature applies to make systems robust—and arguably "intelligent"—and how these strategies may hold as general design principles in modern technology. A particular focus is on computer-based systems and applications. This in mind, the book has four main sections:

Part I has a look at robustness in terms of underlying technologies and infrastructures upon which many computer-based "intelligent" systems reside and investigates robustness on the hardware and software level, but also in larger environments such as the Internet and self-managing systems. The contributions in Part II target robustness in research areas that are inspired by biology, including brain-computer interfaces, biological networks, and biological immune systems, for example. Part III involves the exciting field of artificial intelligence. The chapters here discuss the value of robustness as a general design principle for artificial intelligence, stressing its potential in areas such as humanoid robotics and image processing. In a way, Part IV does the omnipresent nature of robustness justice by lifting it beyond earthly confines into the vastly inspiring and equally challenging space domain, scrutinizing its impact for space mission success, space system design, and bio-regenerative life support systems.

I would like to express my sincerest thankfulness to all authors contributing to this book for their dedicated efforts. The support, guidance, and help I received from Beverley Ford, Helen Desmond, and Frank Ganz at Springer, and from Springer itself, throughout this project was exceptional—thank you very much! I am also delighted to say thank you to the following individuals for their assistance in the review process of this edited book: Dr. Dewar Finlay, Dr. David Glass, Dr. Kieran Greer, Dr. Alexander Grigorash, Dr. Christian Hölscher, Dr. Neil Lester, Dr. Gaye Lightbody, Dr. George Moore, Professor Shahid Masud, Dr. Jürgen Vogel, Dr. Martin Stetter, and Dr. Philip Taylor.

Belfast, February 2008 Alfons Schuster

Contents

Contributors

Uwe Aickelin
School of Computer Science, University of Nottingham, NG8 1BB,
United Kingdom, uxa@cs.nott.ac.uk

Brendan Allison
Institute of Automation, University of Bremen, D-28359 Bremen, Germany,
allison@iat.uni-bremen.de

Cecilio Angulo
Automatic Control Department, Technical University of Catalonia, 08034
Barcelona, Spain, cecilio.angulo@upc.edu

David Bustard
University of Ulster, School of Computing and Information Engineering,
Coleraine, Co. Londonderry BT52 1SA, Northern Ireland, dw.bustard@ulster.ac.uk

Steve Cayzer
Hewlett Packard Labs, Bristol, BS34 8QZ, United Kingdom, steve.cayzer@hp.com

Mathäus Dejori
Siemens Corporate Research, Intelligent Vision and Reasoning Princeton, NJ, USA,
mathaeus.dejori@siemens.com

Jordi Duatis
NTE, 08186 Barcelona, Spain, jordid@nte.es

Martin Fuchs
University of Vienna, Faculty of Mathematics, 1090 Vienna, Austria,
martin.fuchs@univie.ac.at

Daniela Girimonte
European Space Agency, Advanced Concepts Team, 2201 Noordwijk,
The Netherlands, daniela.girimonte@esa.int

Bernhard Graimann
Institute of Automation, University of Bremen, D-28359 Bremen, Germany,
graimann@iat.uni-bremen.de

Axel Gräser
Institute of Automation, University of Bremen, D-28359 Bremen, Germany, ag@iat.uni-bremen.de

Julie Greensmith
School of Computer Science, University of Nottingham, NG8 1BB, United Kingdom, jqg@cs.nott.ac.uk

Rosa Laura Zavala Gutierrez
University of South Carolina, Department of Computer Science and Engineering Columbia, SC 29208, USA, zavalagu@engr.sc.edu

Michael Huhns
University of South Carolina, Department of Computer Science and Engineering Columbia, SC 29208, USA, huhns@engr.sc.edu

Dario Izzo
European Space Agency, Advanced Concepts Team, 2201 Noordwijk, The Netherlands, dario.izzo@esa.in

Gaye Lightbody
University of Ulster, School of Computing & Mathematics, Newtownabbey, Co. Antrim BT37 0QB, Northern Ireland, g.lightbody@ulster.ac.uk

Thorsten Lüth
Institute of Automation, University of Bremen, D-28359 Bremen, Germany, lueth@iat.uni-bremen.de

Olaf Maibaum
German Aerospace Center, Department of Simulation and Software Technology, D-38108 Braunschweig, Germany, olaf.maibaum@dlr.de

Christian Mandel
Institute of Computer Science, University of Bremen, D-28359 Bremen, Germany, cmandel@uni-bremen.de

Giorgio Metta
University of Genoa, Department of Communication, Computer and System Sciences 16145 Genoa, Italy, giorgio.metta@iit.it

Sergio Montenegro
German Aerospace Center, Institute of Space Systems, D-28359 Bremen, Germany, sergio.montenegro@dlr.de

Andreas Nägele
Siemens Corporate Technology, Department of Information and Communications, D-81730 Munich, Germany, andreas.naegele.ext@siemens.com

Arnold Neumaier
University of Vienna, Faculty of Mathematics, 1090 Vienna, Austria arnold.neumaier@univie.ac.at

Francesco Nori
Italian Institute of Technology, 16163 Genoa, Italy, francesco.nori@iit.it

Pere Ponsa
Automatic Control Department, Technical University of Catalonia, 08034 Barcelona, Spain, pedro.ponsa@upc.edu

Vicenç Puig
Automatic Control Department, Technical University of Catalonia 08034 Barcelona, Spain, vicenc.puig@upc.edu

Danijela Ristić
Institute of Automation, University of Bremen, D-28359 Bremen, Germany, ristic@iat.uni-bremen.de

Giulio Sandini
Italian Institute of Technology, 16163 Genoa, Italy, giulio.sandini@iit.it

Alfons Schuster
University of Ulster, School of Computing & Mathematics, Newtownabbey, Co. Antrim BT37 0QB, Northern Ireland, a.schuster@ulster.ac.uk

Roy Sterritt
University of Ulster, School of Computing & Mathematics, Newtownabbey, Co. Antrim BT37 0QB, Northern Ireland, r.sterritt@ulster.ac.uk

Martin Stetter
Siemens Corporate Technology, Department of Information and Communications, D-81730 Munich, Germany, stetter@siemens.com

Thomas Terzibaschian
German Aerospace Center, Department of Optical, Information Systems, D-12489 Berlin, Germany, thomas.terzibaschian@dlr.de

Diana Valbuena
Institute of Automation, University of Bremen, D-28359 Bremen, Germany, valbuena@iat.uni-bremen.de

Jürgen Vogel
European Media Laboratory, D-69118 Heidelberg, Germany, juergen.vogel@eml.org

Jörg Widmer
DoCoMo Communications, Laboratories Europe, D-80687 Munich, Germany widmer@docomolab-euro.com

Roger Woods
Queen's University Belfast, School of Electronics, Electrical Engineering & Computer Science, Belfast BT3 9DT, Northern Ireland, r.woods@qub.ac.uk

Part I
Robustness in Computer Hardware, Software, Networks, and Protocols

Chapter 1
Robustness in Digital Hardware

Roger Woods and Gaye Lightbody

Abstract The growth in electronics has probably been the equivalent of the Industrial Revolution in the past century in terms of how much it has transformed our daily lives. There is a great dependency on technology whether it is in the devices that control travel (e.g., in aircraft or cars), our entertainment and communication systems, or our interaction with money, which has been empowered by the onset of Internet shopping and banking. Despite this reliance, there is still a danger that at some stage devices will fail within the equipment's lifetime. The purpose of this chapter is to look at the factors causing failure and address possible measures to improve robustness in digital hardware technology and specifically chip technology, giving a long-term forecast that will not reassure the reader!

1.1 Introduction

The electronics market has been driven by the incredible growth in silicon density whereby the number of transistors on a single device doubles every 18–24 months, relating to an annual growth of 58%. This is commonly referred to as Moore's Law and is a relationship that is still holding some 30 years from the initial observation. By 2010, the silicon technology roadmap [Allan et al., 2002] estimates that chip sizes will be in the region of 4 billion transistors while reaching clock rates of 10 GHz; this will be expected to exceed 50 GHz by 2017 [IRTS, 2003]. Naturally, this increase in silicon density opens up a wealth of capabilities, permitting extremely complex functions to be implemented as a complete system-on-a-chip (SoC) instead of as a collection of individual components. In real terms, this has driven a number of consumer markets and the computer industry. High-tech products and devices such as digital TV, DVDs, stereos, PCs, PDAs, notebooks, and mobile telephones are all the result of these advances. In addition to area and speed

R. Woods
School of Electronics, Electrical Engineering and Computer Science, ECIT, Queen's University Belfast, Queen's Island, Queen's Road, Belfast, BT3 9DT, Northern Ireland
email: r.woods@qub.ac.uk

A. Schuster (ed.), *Robust Intelligent Systems,* DOI: 10.1007/978-1-84800-261-6_1,
© Springer-Verlag London Limited 2008

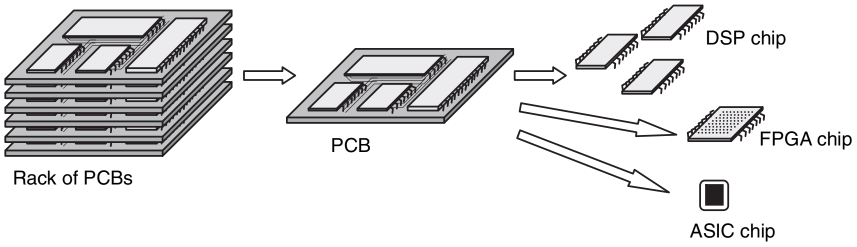


Fig. 1.1 Evolving hardware technologies

gains, reductions in power consumption have also been achieved. This is illustrated in Fig. 1.1, which shows a rack of printed circuit boards (PCBs) being replaced by a single PCB, or in some cases, a single chip such as a Digital Signal Processor microprocessor (DSPμ), a Field Programmable Gate Array (FPGA), or an Application Specific Integrated Circuit (ASIC). A more detailed description of these will be given within this chapter.

Whereas the shrinking of the dimensions of silicon technology provides more transistors as well as making them go faster and consume less power, it also causes problems with regard to robustness. As stated within this book, a robust system is widely viewed to be *a system that tolerates faults*. We are now dealing with robustness on several fronts. Firstly, with shrinking technology we are now coping with progressively sensitive transistors whose effects have to increasingly be dealt in the design process [Phillips, 2007]. Secondly, the methodologies needed to design advanced silicon chips have not matured at the same rate as the increase in silicon density. This has resulted in a gap between the rate of growth in silicon density (58%) to the rate of increase in transistors implemented per staff-month (21%), commonly referred to as the *design productivity gap* [Rowen, 2002], as illustrated in Fig. 1.2.

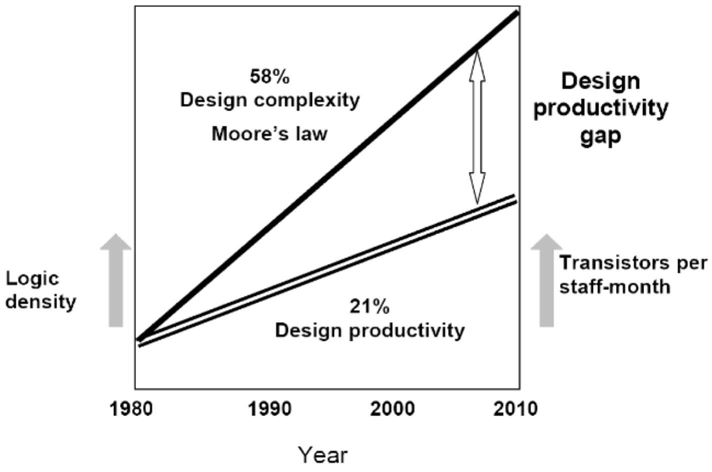


Fig. 1.2 Design productivity gap [Rowen, 2002]

Thirdly, the design problem has changed. More than a decade ago, a single chip represented a component of a system but now the chip can represent the full system itself, hence *system-on-chip*, involving a number of technologies ranging from sensors to technology for receiving data signals. These technologies typically involve different design approaches, and additional care is needed as they can interfere with each other. Lastly, there is also an expectation that electronic devices will not fail, which was a strong selling point over the less reliable mechanical and analog technology. This is increasingly becoming more difficult to achieve as both the technology becomes more unpredictable and the tools lag behind the methodology.

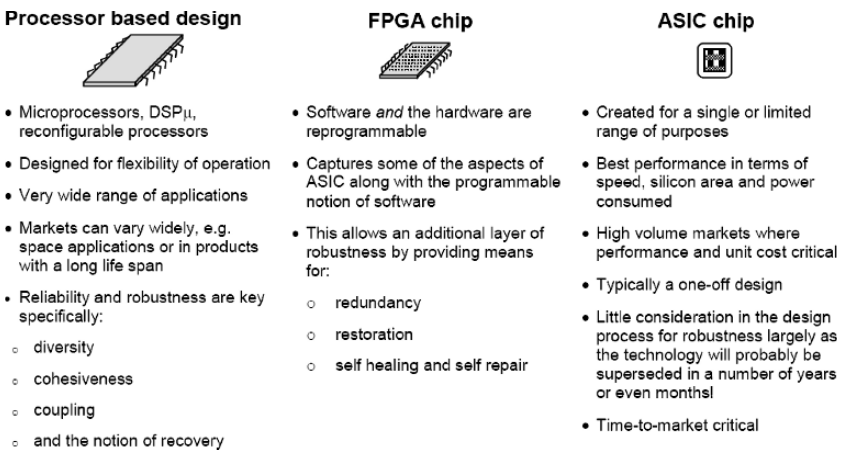
The purpose of the chapter is to address some of the issues regarding robustness of hardware, specifically digital hardware. The aim is to present an insight and discussion on the elements of digital hardware system design that provide a level of robustness. It starts with a brief discussion of issues of robustness in hardware highlighting increasing device complexity. The chapter then focuses on hardware technologies and describes the characteristics of various types of hardware platforms, namely ASIC, DSPμ, microprocessors, and FPGA. The description also gives details of their structure and programming model with a particular focus on robustness. The next section covers the types of faults that can occur in silicon hardware and the design and operation techniques available to avoid these faults. The latter sections will then give an overview of the approaches currently available for proving hardware robustness ranging from the highly practical to some of the more exotic solutions. Finally, the chapter will conclude by considering the future problems caused by process variations and the challenges this will create.

1.2 Digital Hardware Technologies

In the evolution of silicon technology over the past 40 years, a number of different types of hardware technologies have emerged and can be generally classified as being either ASIC, programmable based designs (DSPμ, microprocessors), or as FPGAs. A brief description of each is given in Fig. 1.3.

The first offerings were the 74 series logic chips in the 1960s, which were the building block for many "budding techies" up to recently. This was followed by the first microprocessor, the Intel 4004 processor in 1971, which along with microcontrollers have now formed the core for many applications with lower bandwidth requirements. The concept of developing custom large-scale integration (LSI) components emerged in the 1970s, but the development of the programmable logic array (PLA) in 1978 and the first FPGA in 1984 was the start of an increasing interest in programmable hardware structures. Remarkably, this was followed by a renaissance of developing dedicated hardware implementations or ASICs in the late 1980s.

The first aspect is to highlight that silicon fabrication works on the principle that each chip will meet the specification for which it was designed. Therefore, a margin of error is built into each design such that it is guaranteed to work at the required specification. To this end, two specifications are typically catered for, the *industrial* one and the more rigorous *military* specification; these represent the



Processor based design

- Microprocessors, DSPμ, reconfigurable processors
- Designed for flexibility of operation
- Very wide range of applications
- Markets can vary widely, e.g. space applications or in products with a long life span
- Reliability and robustness are key specifically:
  - diversity
  - cohesiveness
  - coupling
  - and the notion of recovery

FPGA chip

- Software *and* the hardware are reprogrammable
- Captures some of the aspects of ASIC along with the programmable notion of software
- This allows an additional layer of robustness by providing means for:
  - redundancy
  - restoration
  - self healing and self repair

ASIC chip

- Created for a single or limited range of purposes
- Best performance in terms of speed, silicon area and power consumed
- High volume markets where performance and unit cost critical
- Typically a one-off design
- Little consideration in the design process for robustness largely as the technology will probably be superseded in a number of years or even months!
- Time-to-market critical

Fig. 1.3 Basic technology comparison

worst-case conditions at which the device will operate. Heat is a key issue as it decreases device speed and can accelerate possible chip malfunctions but it can be taken into account in the design and simulation stages. The major focus on *right first time* design is driven by the spiraling costs of *masks*, which are the result of the design process and are used to fabricate the chip. Typically, they cost $1M to create [LaPedus, 2007]; a price that is incurred whether they are needed for a small number of prototypes or for a full deployment of a million devices. Even so, yet as many as 55% of designs are failing to pass first silicon [Robertson, 2004] therefore incurring a great loss of money and time!

In addition to the highly pessimistic design process, another interesting feature impacting robustness was first observed by Tsugio Makimoto of Sony. It was coined *Makimoto's wave* and is illustrated in Fig. 1.4. Makimoto observed that system construction seemed to be swinging between *standardization* where silicon chips were created as standard parts (e.g., TTL chips, microprocessors, and then FPGAs) and *customization* in the form of LSI and ASICs developed for single or a restricted

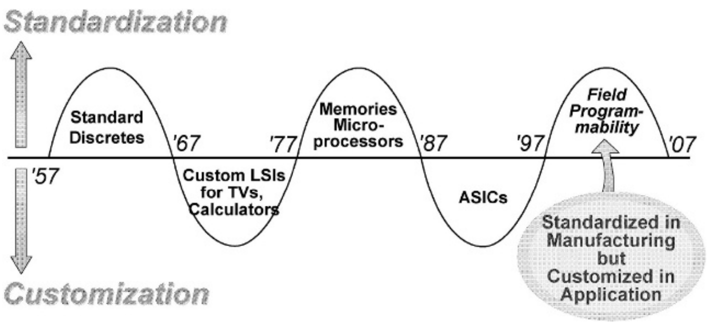

Standardization
Standard Discretes
'57
'67
Custom LSIs for TVs, Calculators
'77
Memories Micro-processors
'87
ASICs
'97
Field Programmability
'07
Standardized in Manufacturing but Customized in Application
Customization

Fig. 1.4 Makimoto's wave (Source: Electronics Weekly, January, 1991)

range of applications. For the *standardized* part, the key attribute was usability across a range of applications implying *granularity* in the form of a building block that could be used to build a larger function for a range of applications, therefore also supporting *diversity*. The customized solution implies a limited range of applications, therefore robustness is not a key criteria either in terms of the application range needed or, indeed, the lifetime of the product.

1.2.1 More Detailed Description of Technology

In the previous section, the various types of technologies were briefly outlined. This section provides a more detailed overview and their current status.

Microprocessors currently have clock rates of 4 GHz, and recent predictions under the International Technological Roadmap for Semiconductors [IRTS, 2003] suggest clock rates will exceed 50 GHz by 2017. Whereas these performance figures are staggering, it must be remembered that the microprocessor has a von Neumann architecture where computations are performed in a serial fashion, not ideal for highly parallel algorithms. It could be argued that the inefficiency is an acceptable price to pay for the high levels of programmability that the microprocessor offers, but energy considerations are becoming a critical design factor in many applications and this level of inefficiency cannot be tolerated. The programming route is via high-level software platforms and, as the user will no doubt be well aware, this platform has a wide application base. Evolutions over the past decade include *hyper-threading* [Koufaty and Marr, 2003] supported on the Intel Pentium 4 device and the evolution of multiple processors on a single die.

DSPμs are dedicated hardware platforms that are based on a different underlying architecture from the microprocessor, namely the Harvard architecture, which uses separate buses for instructions and data and has physically separate storage memories for program and data. The device is programmed in much the same fashion as the microprocessor and is applied largely in the DSP market as a commodity part. An overview of DSP architectures can be found in [Glossner et al., 2000] and [Tan and Heinzelman, 2003]. The major innovations in DSPμs include very long instruction words (VLIW) and single input multiple data (SIMD) architectural changes [Rui et al., 2003]. These innovations are possible due to the computational needs of many DSP computations.

FPGAs have emerged from being glue logic components in the early 1980s to offering complete processing platforms for complex systems. They offer programmability, but unlike the microprocessor/DSPμ, this involves changing the hardware architecture of the device. This offers a number of advantages from a robustness perspective. If the FPGA is used as part of a system, it is possible to reconfigure the hardware and operation to overcome faults caused either in the design stage or as a result of malfunction. Given that the device comprises a lot of similar cells that can be migrated to different parts of the device, it does introduce the possibility of *self-healing* [Andraka and Brady, 2002, Samudrala et al., 2004, Gokhale et al., 2006]. However, a key aspect of digital design implementation is the standardization and verification of designs, which the concept of *self-healing* acts against.

ASICs and toward system-on-chip (SoC) involve a detailed and costly design process plus the costs to produce the mask for fabrication. A number of variations include *standard cell*, which comprises cells of pre-designed logic blocks and *structured ASIC*, which is a more modern version of gate array technology where the structure is mostly developed and the only custom fabrication step is the definition of the interconnection. The one-off costs called non-recurrence engineering (NRE) costs can be amortized into the component cost when volumes are large. Thus the nature of the designs are such that they are heavily restricted, as the name suggests, and little thought will have gone into making the designs robust unless the application directly requires it. In this case, particular choices may be made to use, for example, radiation hard fabrication technologies if the applications require this level of protection, or even to duplicate circuitry [Hollander et al., 1995, Lacoe et al., 2000, Ruano et al., 2007].

ASSP (application specific standard product) (includes emerging platforms such as, for example, reconfigurable DSPs) have also emerged. It is clear from the descriptions above that there is no single suitable technology; ASSPs combine the *fixed architecture* structure of microprocessors or DSPμs but provide some level of hardware programmability to allow the processor to be customized to the specific application. This represents the new types of technologies hinted at in the post-2007 part of the Makimoto wave, which are largely standardized but which can be customized for specific applications.

In truth, a lot of variations of these classifications are beginning to appear. For example, FPGAs now have embedded microprocessors such as the Xilinx MicroBlaze and the Altera NIOS processor, which can be customized for specific applications. DSPμs are also becoming complex SoC platforms with dedicated hardware units for high-performance wireless communications and other functions. IBM has been developing platforms with embedded FPGAs and Intel has been embedding DSP engines within their RISC (reduced instruction set computer) technology. It is clear that hybrids are beginning to emerge allowing users to select architectures based on system requirements and making it difficult to categorize the technologies in the classical way that has been done here. These technologies are best compared against time-to-market, performance, price, ease of development, energy efficiency, as well as a number of other emerging factors, such as radiation hardware, ability to incorporate legacy designs, and upgradability and, of course, robustness.

1.3 Issues with Testing and Verifying Digital Hardware

Digital design is subject to faults in PCB or chip design formats due to errors in the manufacturing process. It is therefore essential that the circuit is tested both prior to delivery to the customer and also prior to operation in normal day-to-day use. From a manufacturing point-of-view, it is important to carry out *fault analysis* to detect where the fault has occurred in order to isolate any production problems. However, it is typical that faults will occur and a certain level of faults will be tolerated, thus it falls on the designer to create a series of tests that will allow *fault detection* thereby establishing whether a fault has occurred in order that the producer can eliminate faulty chips/PCBs and thus remove them at production.

In electronic design generally but IC design specifically, there are a number of causes of faults. These are classified as:

Open Circuit where the circuit connection is broken thus causing an "open" circuit. This can occur due to *electric migration* where the electric fields are so strong that the metal atoms migrate thereby causing a break in the connection and also due to *current overstress* where large currents make the wire lift from the silicon thereby breaking the connection and once again causing an open circuit.

Short Circuit which can be caused by defects in the fabrication material used (e.g., silicon oxide) or in the fabrication process itself where the mask used to create the transistors is not aligned properly. The result is that correct isolation is not provided and a short circuit can occur thereby causing malfunction.

Latch Up which is a special condition that can happen in CMOS (complementary metal oxide semiconductor) circuits where transient currents force CMOS gates to be stuck at a value, thereby preventing it from changing as required. It can be a temporary condition that terminates upon removal of the inputs or a condition that requires a full system reset or even replacement of damaged parts.

Single Event Upsets (SEU), which are errors induced by radiation effects when charged particles, typically from cosmic rays, lose energy by ionizing the silicon material through which they pass, leaving behind a wake of electron-hole pairs. Two types of radiation have been identified as the primary cause of SEUs in semiconductor devices: alpha particle radiation (which has now been largely avoided by higher-purity package materials) and atmospheric neutrons, originating from the effects of cosmic rays hitting the Earth's atmosphere, which remain the primary cause for SEU effects today. Typically, they appear as transient logic pulses or manifest themselves as bit changes in memory or flip-flop storage. They tend to be transient soft errors and are not destructive to the technology.

1.3.1 Static Faults

The static faults, which broadly speaking tend to be physical defects and electrical faults, are dealt with at the design stage and can be tested at manufacture and in some cases during normal operation by applying the *built-in self test* (BIST) sequence prior to operation. A physical fault will manifest as a logical fault that can be either static or dynamic. The dynamic faults appear at certain times and are usually caused by timing faults. This should be catered for within the design cycle and should be avoided. Static faults are logical faults and possibly due to one of the effects listed above and thus must be checked. The designer can incorporate the impact of these faults using a number of models.

One of the most common mechanisms that is also used commercially is the *stuck-at model*, which assumes that the logic gates will be *stuck-at-0* or *stuck-at-1* as a result of the physical defects and electrical faults. Therefore, the aim is to generate a test sequence that will reveal the error by producing an incorrect sequence at the output of the chip or circuit due to the presence of the fault. From a design perspective, we do this by stipulating the *stuck-at* condition and then determining which input sequences will produce the errored output sequence. We test every *stuck-at* condition in the circuit and generate a test file, which will then detect a percentage of the errors. This is known as *fault coverage*.

The relationship between faults and yield is given in Fig. 1.5. If the production yield of fault-free circuits on a wafer is Y, then $Y = 0$ means that all circuits are faulty and $Y = 1$ means that all circuits are fault free. If FC is the fault coverage, then $FC = 0$ means that tests do not detect any faults and $FC = 1$ means that tests detect all possible faults. Thus, the defect level (DL) is given by the graph outline. It is easy to determine that a higher fault coverage is desirable with typical production lines $(Y = 0.7 - -0.8)$.

The major difficulty with generating this test file to provide the fault coverage is that the designer has only a very limited access to the circuit in terms of inputs and outputs. In a chip, this will only be provided through the input and output pins and with multi-layered boards, and it is now becoming increasingly difficult to even monitor individual chips without adding test connections; this is costly in terms of area and increases the risk of design failure due to their very existence.

Two terms that are an important measure in fault detection and design for test are *observability*, which is used to indicate how well the state of a signal on an internal node may be detected on the output pins, and *controllability*, which describes the

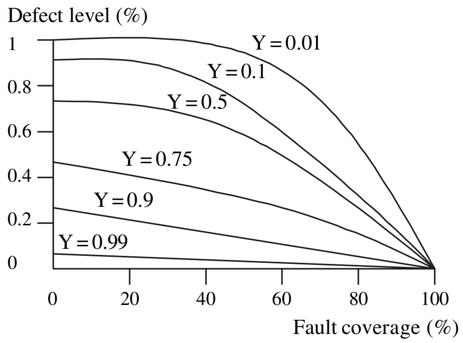


Fig. 1.5 Relationship between defect level and fault coverage [Hurst, 1988]

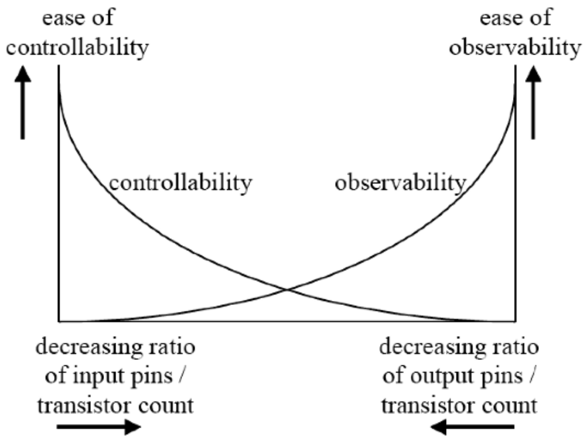


Fig. 1.6 Variation in observability and controllability in silicon chips

case with which a particular internal signal value can be set by applying signals to the input pins. Figure 1.6 shows how these values vary in a silicon chip.

The main problem is that as complexity grows, the number of logic levels between input and output grows, meaning that large proportions of the transistors can be neither tested nor observed. The designer's job is thus to add additional circuitry that makes the internal nodes more controllable and observable. This can be done by using a scan path to the circuit allowing the nodes to be linked into one big scan path in addition to their normal operation. This is now a standard within the IEEE, called the "IEEE boundary-scan standard 1149", and can be used at board level as well as chip level.

1.3.2 Dynamic Faults

There is a well-established mechanism for robustness in IC design with regard to static faults, as the chip manufacturer will normally insist that a chip design will be delivered in terms of the mask sets, etc., along with the necessary test data to provide 95% minimum fault coverage, i.e., 0.95 for Fig. 1.5. This value is used on the basis that their statistics tell them that 95% fault coverage is usually good enough to ensure that *no* chips sent from the production line will contain faults. However, dynamic faults are more problematic and represent a much greater problem with regard to robustness.

With regard to CMOS latch up, literature tends to suggest that a lot of problems can be avoided by using specific technology fixes and by giving careful attention to CMOS outputs, their loading, and the stresses applied to them in order to avoid the high transient currents that cause the problem [Naughton and Tyler, 2005, Boselli and Duvvury, 2005]. However, the SEU issue is becoming much more critical as the effect is greater as transistors are effectively becoming smaller compared with the particles as the technology scales. It is of particular concern for airborne or space industries because of the increased risk in higher atmospheres. A lot of vendors are now including additional test circuits to test for these errors as the circuit is operating [Hollander et al., 1995, Hentschke et al., 2002, Makihara et al., 2003, Samudrala et al., 2004, Golshan and Bozorgzadeh, 2007].

1.4 Robustness Approaches

Robustness was defined in this book and elsewhere and summarized as possessing some or all of the following features: *redundancy*, *compensation*, *diversity*, *mechanical robustness*, *granularity*, *restoration (cognitive)*, *cohesion and coupling*, *recovery, self-healing, self-repair (physical)*. From the authors' perspective, the following aspects feature strongly in the design and realization of digital circuitry namely *redundancy*, *diversity*, *granularity*, *recovery*, as well as *self-healing* and *self-repair* (physical).

1.4.1 Redundancy

Typically, redundancy is applied for two reasons in digital hardware implementation, namely for *improved design performance* and for *test purposes*.

1.4.1.1 Redundancy for Design Performance

Redundancy can be employed at many levels in hardware. At the system level, redundancy is directly related to cost and will be avoided unless there is a strong application need for it. For example, the equipment may be either remotely or inconveniently located making repair a costly exercise as the equipment is difficult to access. In this case, several modules may be co-located along with the necessary test circuitry to ensure correct detection of the errored state. The circuitry can be either switched on live, i.e., *hot insertion*, or turned on when the equipment is in reset mode, i.e., temporarily turned-off.

Given that redundancy capability for design can be exploited at the system level as suggested above, there is little requirement to deliberately introduce redundancy in the components described in Section 1.2.1 as this will involve additional silicon area and therefore increase component cost. There are a few exceptions to this. For example, the circuit in Fig. 1.7(a) implements the digital function f $= A.\overline{B}.D + B.C.D$.

However, when the input B changes, as shown, both AND gates can temporarily give a "0" thus giving an output of "0", which is not logically correct. This can be avoided by adding a *redundant* gate, shown in shading in Fig. 1.7(b), which always ensures a logical 1 to the OR when B changes ensuring a constant "1" and avoiding any glitching problems that may result. These types of redundancy techniques are employed for specific design quality reasons but should be avoided because the techniques used to perform static testing of the circuit in Fig. 1.7(a) are now complicated

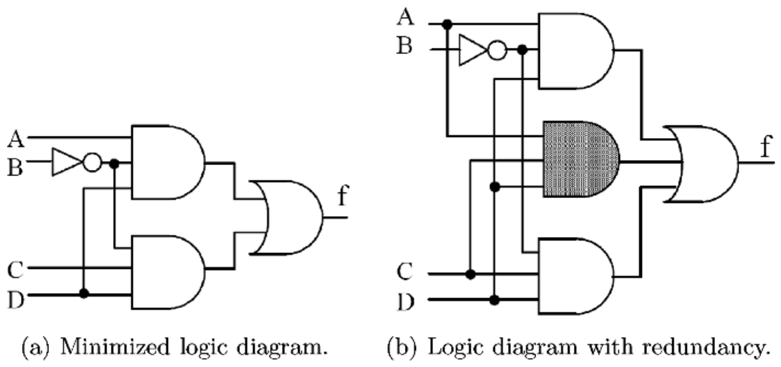

A
B
C
D
f
(a) Minimized logic diagram.
A
B
C
D
f
(b) Logic diagram with redundancy.

Fig. 1.7 Redundancy in logic design

by the addition of the gate in Fig. 1.7(b). In some cases, this level of application can make the circuits untestable.

FPGAs, on the other hand, offer an alternative to employing redundancy due to their internal structure. As highlighted earlier, the FPGA structure typically comprises a series of logic cells that can be interconnected in different ways by programming the interconnection. FPGAs are sold on the basis of discrete chip sizes, which simply scales the number of available resources. With fixed resources within a chosen device, some level of unused logic may exist, which the designer can exploit to provide redundancy.

1.4.1.2 Test Circuitry

As was highlighted in Section 1.3, testing is becoming an increasingly difficult task as transistor counts increase. For this reason, test is now part of the design cycle, and there is an increased "emphasis" on *design for test*, i.e., introducing redundant circuitry with the aim of making the circuit more testable. The key diagram in this discussion is Fig. 1.6, which shows that our ability to control and observe internal circuitry is decreasing as the ratio of input and output pins to transistor count decreases, making it ever more difficult to obtain adequate fault coverage as design complexity grows. A truer description of this graph would show a large proportion of the chip that we can neither control nor observe from a testing point of view. It is vital that we continue to demonstrate that electronics is a robust technology, thus there is a very strong need to increase these levels and achieve the necessary fault tolerance.

Design for testability is typically based on three concepts, employing ad-hoc techniques, scan path, and built-in self test (BIST). Ad hoc as the Latin suggests is employing design methods that are suitable for the design under consideration. This includes obvious things like the introduction of test points that allow the designer to get data into the heart of the circuit and also to observe output from the same or similar point; isolation of key circuits, e.g., the clock allowing the designer to enter a test clock as opposed to the normal clock; adding additional circuitry to allow bypassing of parts of the circuits, which is similar to the use of test points; strategies for allowing initialization of circuits in a different mode. All these techniques are reasonably obvious, but the introduction of test points and bypass circuitry infers additional circuitry. Typically, test circuitry will constitute an extra 10% silicon area, which is a unrequited cost as this area is only ever used for test purposes.

Scan path and BIST are two specific design techniques for testing. Scan path involves using a more complex register that allows all the registers in a circuit to be connected together in a *daisy chain*. This is a bit like providing *tunnels* into the circuitry allowing data to be fed right into and read from the heart of the circuit at a desired point. Effectively, this is a system means of providing improved test point access.

BIST, on the other hand, is a more crude method of testing. All the previous techniques involve creating the hardware and the necessary data streams to provide the testing of the circuitry. BIST employs a *random number generator* to produce a huge amount of data that is then fed into the circuit and a *signature analysis circuit* to capture the output data produced by the circuit under test, producing a final signature, namely a binary word. If any test problem has occurred, then the signature produced will be different, or strictly speaking, will have a very small statistical chance of being the same! All of the above methods involve introducing redundant circuitry with the aim of making the circuitry more testable and thus more robust.

Redundancy also features strongly in methods to protect against SEUs. Here, Triple Mode Redundancy (TMR) can be applied to allow a majority vote, that is, the logic is repeated three times and the outputs compared with a choice made for the two that are the same, providing that there is not a failure in the voting logic itself [Andraka and Brady, 2002, Samudrala et al., 2004]. Naturally, such extreme measures would be more applicable in safety critical scenarios and applications most susceptible to SEU effects.

1.4.2 Diversity

The notion of diversity can be best described in Makimoto's wave in Fig. 1.4, which shows how the various technology evolutions have swung between standardization and customization. As the name suggests, customization implies building hardware that has a very restricted application focus, whereas the standard components such as FPGAs and microprocessors can be applied to a diverse range of applications. The provision of suitable programming environments for both these technologies has meant a wide and varied range of applications. For example, most recently there has been a huge interest in employing FPGAs for scientific computing applications as acceleration to arrays of processors and companies such as Silicon Graphics and Cray who built their reputation of building processor chips are now offering processor/FPGA platforms. The attraction of being able to construct highly parallel architectures for some scientific applications gives huge potential.

The microprocessor itself has had a wide and varied application environment. For example, processors are now starting to be used quite widely in the automobile industry to provide the range of applications and comforts required by many drivers today. A whole genera of computer music has now been created where the composer is now using computers, typically Apple Macs, as instruments. This is only to name but a few applications. It should not be underestimated that this diversification is based on the inherent reliability of the technology, which relies heavily on the design principles described in the previous section.

1.4.3 Granularity

When we talk about granularity of a design, it can mean any one of a number of things. If the application scales, can the design scale with it? There are a number of levels of granularity. At the highest level, this could be a matter of using more PCBs to meet the system demands or adding additional chips. As we go down the level of granularity, it would mean enabling greater performance output from the chip designs. Here, performance enhancement could be met by retargeting the design to a more powerful technology, maybe the current FPGA devices or latest ASIC foundry technology. But true design scalability is a multidimensional challenge that has fueled a research area devoted to parameterizable design and *design for reuse* strategies. The following section will detail some of the key issues in need of consideration when undertaking a fully parameterizable design. The benefits of rising to these challenges is also discussed with emphasis given on how design practices such as these aim to add robustness to chip development.

1.4.3.1 Design Reuse—IP Cores

The increase in transistor count, leading to the incorporation of an entire system on a single device, has resulted in an exceptionally complex design route with component heterogeneity escalating problematic issues regarding chip design and in particular test and verification. As highlighted earlier, the design productivity gap of Fig. 1.2 means that we must think of dramatically different ways of constructing systems. Specifically, this means treating the design process in a more abstract fashion by representing designs as comprising components that, if possible, can be reused from existing designs.

The concept of creating a system in this modular fashion has effectively created an industry in third-party products referred to as silicon Intellectual Property (IP) cores, also known as virtual circuits (VCs). These range from actual silicon layout known as *hard cores* through to *soft cores*, which can be in the form of efficient code targeted to programmable DSP or RISC processors, or dedicated cores captured in a Hardware Description Language (HDL). Effectively, *design for reuse* methodologies provide flexibility allowing designs targeted to one project to be applied to another one with different specifications.

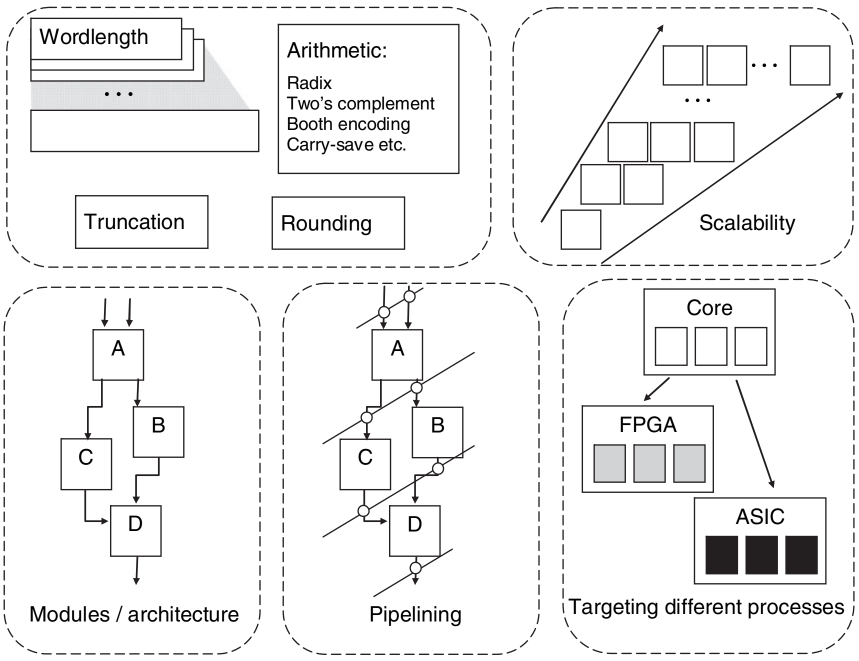


Fig. 1.8 IP parameters

The key aspect in developing IP cores is the concept of identifying the parameters needed for the general usage of the cores, as demonstrated in Fig. 1.8. Parameterization leads to a library of cores that can be targeted to a range of specifications, without the need to alter the internal workings of the code. The system should effectively allow a number of parameters to be fed into the top level of the code and then passed down through the different levels of abstraction of the code to the lowest levels [Shannon, 2002, Lightbody et al., 2003]. Obviously, considerable effort is needed at the architecture level to develop this parameterizable circuit architecture. This initial expense in terms of time and effort undoubtedly hinders the expanded use of *design for reuse* principles but could result in savings in the long run.

In Fig. 1.8, design aspects such as arithmetic effects, i.e., type of arithmetic used namely fixed or floating point representations, need to be defined as a parameter. *Pipelining*, which is a design technique to speed up performance, can be applied but it has timing implications. The cores must also be able to be applied across a range of platforms, which, as the earlier description showed, can vary quite dramatically in terms of their internal architecture. Identifying the key parameters and then designing the IP core for them requires a detailed understanding of the range of implementations in which the core may be used. The aim is to balance flexibility with the additional workload and resulting benefits. This is important as over-parameterization of a design not only affects the development time but also affects the verification and testing so to ensure that all permutations of the core have been considered [Gajski et al., 2000].

1.4.4 Recovery, Self-healing, Self-repair

Generally speaking, electronic systems are designed for usually a short life span when compared with mechanical products because of trends in product aging, i.e., yesterday's design. In addition, newer technology is emerging at such a pace that the performance gain of the new technology is such that the need to reconfigure or self-heal older technology is not worth the cost of adding the extra functionality to achieve this. However, this assumes that in the construction of such systems, the specification is fully defined and does not need to be altered at a later stage of the design flow. Given the nature of recent systems, the design process is now viewed as one that is constantly evolving.

Thus, the programmable nature of certainly microprocessor and, more recently, FPGAs present a highly attractive, low-risk proposition for system designers. A heterogeneous platform comprising processors and FPGAs will have a worse performance profile in terms of power-area-speed performance, but this drop-off in performance can be put against the added advantage of reconfigurability allowing change during the design cycle and, indeed, afterward. From an engineering perspective, this represents a good engineering safety net, but as the title of this section indicates, users have taken this to a further extreme effectively, with circuits that will self-repair or self-heal [Gokhale et al., 2006, Andraka and Brady, 2002, Samudrala et al., 2004].

1.4.4.1 FPGA Reconfiguration

The opportunity offered by the reconfigurability property of FPGAs is very appealing. It is first worthwhile explaining the operation of FPGAs to see what level of self-healing the device offers. Most FPGAs are based on static random access memory (SRAM). The basic principle is that the FPGA is programmed exactly like a memory, but rather than storing the information as in a conventional memory, the FPGA uses the data to program circuitry that performs the required function. Thus, reprogramming the memory is effectively changing the function. The programming information is created by the design tools that correctly mimic the function that the user programmed into the tools.

One of the main issues with processors is that they are very inefficient. Typically, [Hennessey and Patterson, 1996] in their famous text indicated that most computationally complex applications spend 90% of their execution time in only 10% of their code, and the instructions in this 10% of the code will vary from application to application. The attraction of being able to dramatically increase this level of performance by programming the architecture to match the performance requirements holds a great attraction from a power-area-speed perspective. The concept was captured by the notion of FPGA Custom Computing Machines (FCCMs) for which a dedicated conference has been run every year in Napa, California.

1.4.4.2 The Self-repair Rationale

FCCMs typically comprise FPGAs connected to a microprocessor that stores the reconfiguration data necessary to convert the FPGA into the required functional engine. The system can either be pre-designed to cope with a number of modes of operation, thus there is either a required sequence of reconfiguration of the device or the reconfiguration is triggered by the hardware itself to load the best configuration data to match the needs of the device. The reconfiguration can either be performed by completely reprogramming the device or partially reconfiguring it [Sezer et al., 1998]. The problem with full reconfiguration is that the device can be unoperational for up to several milliseconds. In a real application where the data needs to be processed in the region of millions of cycles per second, this means the temporary storage of a considerable amount of data. In addition, the rate at which reconfiguration takes place can then be counter-productive as the graph in Fig. 1.9 indicates. This graph was produced for a key image processing function in image compression and shows that even though the FPGA implementation is much faster than the processor implementation, the reconfiguration time required means that it will only be efficient after a certain time.

Researchers have gotten around this limitation in a number of ways, firstly by performing *partial reconfiguration*, which involves changing part of the programming information of the FPGA while it is still operating and actually designing the circuits to allow this to happen. This involves splitting the application into a number of sequential stages. Each of these stages could then be implemented on the FPGA in turn, with the device being reconfigured between stages to support the next operation. Alternatively or in addition to partial reconfiguration, the amount of reconfiguration information could be compressed therefore speeding up reconfiguration [Compton and Hauck, 2002].

The real panacea would be to develop a system where the designer does not have to worry about creating the programming and that the device would reconfigure itself. These systems are known as evolvable systems [Stoica et al., 2001] and are an active area of research. Researchers have been building truly evolvable systems for a range of applications by exploiting the reconfigurability property of FPGAs. Whereas this is attractive from an implementation perspective, it is laden with trou-

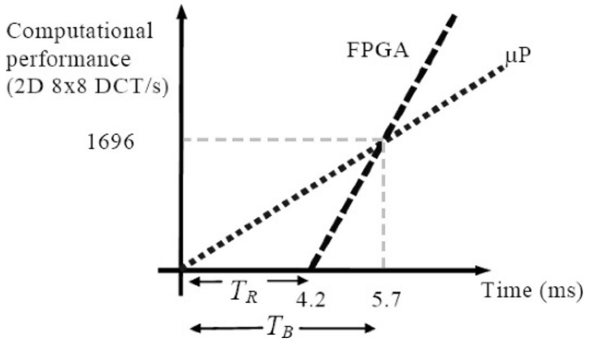


Fig. 1.9 Impact of reconfiguration time for a “8X8 2D discrete cosine transform (DCT)” implementation

ble for an engineer. For example, how can you be sure that the system has correctly reconfigured? This questions the concept of whether there is an ideal specification system that a product will work to or whether the user is prepared to accept a system that will vary in quality of operation. We think that this has to be sold to the public, but some of the discussion in the summary about where technology is evolving to may influence this argument.

1.5 Summary

The chapter has considered the robustness of hardware systems covering a range of topics including *redundancy*, *diversity*, *granularity*, *recovery*, as well as *self-healing* and *self-repair* (physical). The discussion has been given from an engineering perspective, arguably with a word of caution about the opportunities of reconfigurable FPGAs to employ self-healing or self-repair properties. The reality is that up to now, this additional level of functionality has been a luxury, and it should also be realized that the designer is responsible to ensure the correct operation of any circuitry, even circuitry added with the good intention to make it robust! The long-term future presents some interesting problems though. Reliability has been key to Moore's Law, but future indications suggest that upcoming technologies will start to be very unreliable due to the variation in performance of the transistors as a result of shrinking dimensions [Constantinescu, 2003]. We will then just have to work with the reality of dramatic transistor speed variation or transistors that may not work at all. Self-healing and self-repair may then be required steps in the design process. From an engineering perspective, this is a horrendous nightmare that might be best avoided, thus signaling an end to Moore's Law?

References

Allan, A., Edenfeld, D., Joyner, W.H., J., Kahng, A., Rodgers, M., and Zorian, Y. (2002). 2001 Technology Roadmap for Semiconductors. *Computer*, 35(1):42–53.

Andraka, R. and Brady, J. (2002). Low complexity method for detecting configuration upsets in SRAM-based FPGAs. In *Proceedings 5th International Conference on Military and Aerospace Programmable Logic Devices*, volume B4, Maryland, USA.

Boselli, G. and Duvvury, C. (2005). Trends and challenges to ESD and Latch-up designs for nanometer CMOS technologies. *Microelectronics and Reliability*, 45(9–11):1406–1414.

Compton, K. and Hauck, S. (2002). Reconfigurable computing: a survey of systems and software. *ACM Computer Survey*, 34(2):171–210.

Constantinescu, C. (2003). Trends and challenges in VLSI circuit reliability. *IEEE Micro*, 23(4):14–19.

Gajski, D., Wu, A.-H., Chaiyakul, V., Mori, S., Nukiyama, T., and Bricaud, P. (2000). Essential issues for IP reuse. In *Proceedings of the ASP-DAC 2000 Design Automation Conference, Asia and South Pacific*, pages 37–42, January 25–28, Yokohama, Japan. IEEE Standards Office.

Glossner, J., Moreno, J., Moudgill, M., Derby, J., Hokenek, E., Meltzer, D., Shvadron, U., and Ware, M. (2000). Trends in compilable DSP architecture. In *Workshop on Signal Processing Systems (SIPS'2000)*, pages 181–1999, October 11–13, Lafayette, LA, USA. IEEE Press.

Gokhale, M., Graham, P., Wirthlin, M., and Johnson, D. (2006). Dynamic reconfiguration for management of radiation-induced faults in FPGAs. *International Journal of Embedded Systems*, 2(1):28–38.

Golshan, S. and Bozorgzadeh, E. (2007). Single-event-upset (SEU) awareness in FPGA routing. In *Proceedings 44th Confenrence on Design Automation*, pages 330–333. ACM Press, New York.

Hennessey, J. and Patterson, D. (1996). *Computer Architecture: A Quantitative Approach*. Morgan Kaufmann Publishers Inc., New York.

Hentschke, R., Marques, F., Lima, F., Carro, L., Susin, A., and Reis, R. (2002). Analyzing area and performance penalty of protecting different digital modules with Hamming code and triple modular redundancy. In *Proceedings 15th Symposium on Integrated Circuits and Systems Design*, pages 95–100, September 9–14, Porto Alegre, RS, Brazil. IEEE Computer Society.

Hollander, H., Carlson, B., and Bennett, T. (1995). Synthesis of SEU-tolerant ASICs using concurrent error correction. In *Proceedings 5th Great Lakes Symposium on VLSI*, pages 90–93, March 16–18, Washington, USA. IEEE Computer Society.

Hurst, S. (1988). *VLSI Testing: digital and mixed analogue/digital techniques*. IEE, Savoy Place, London.

IRTS (2003). International roadmap for semiconductors. `http://www.itrs.net/`.

Koufaty, D. and Marr, D. (2003). Hyperthreading technology in the netburst microarchitecture. *IEEE Micro*, 23(2):56–65.

Lacoe, R., Osborn, J., Koga, R., Brown, S., and Mayer, D. (2000). Application of hardness-by-design methodology to radiation-tolerant ASIC technologies. *IEEE Transactions on Nuclear Science*, 47(6):2334–2341.

LaPedus, M. (2007). Open-silicon to drive down mask costs. *EE Times Online*. `http://www.eetimes.com/showArticle.jhtml?articleID$=$198900081`.

Lightbody, G., Woods, R., and Walke, R. (2003). Design of a parameterizable silicon intellectual property core for QR-based RLS filtering. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 11(4):659–678.

Makihara, A., Sakaide, Y., Tsuchiya, Y., Arimitsu, T., Asai, H., Iide, Y., Shindou, H., Kuboyama, S., and Matsuda, S. (2003). Single-event effects in 0.18/spl mu/m CMOS commercial processes. *IEEE Transactions on Nuclear Science*, 50(6):2135–2138.

Naughton, J. and Tyler, M. (2005). Best methods to minimize latch-up sensitivities in semiconductor circuits. In *IEEE Workshop on Microelectronics and Electron Devices*, pages 95–98, April 15, Boise, Idaho, US. IEEE Standards Office.

Phillips, I. (2007). When less means more; and more, the-same? In *IEEE International Symposium on Industrial Embedded Systems*, July 4–6, Lisbon, Portugal. IEEE Industrial Electronics Society.

Robertson, C. (2004). Silicon modeling of nanometer systems-on-chip. In *Proceedings 4th IEEE International Workshop on System-on-Chip for Real-Time Applications*, pages 19–22, July 19–21, Banff, Canada. IEEE Computer Society.

Rowen, C. (2002). Reducing SoC simulation and development time. *Computer*, 35(12):29–34.

Ruano, O., Reyes, P., Maestro, J., Sterpone, L., and Reviriego, P. (2007). An experimental analysis of SEU sensitiveness on system knowledge-based hardening techniques. In *IEEE Conference on Design and Diagnostics of Electronic Circuits and Systems (DDECS'07)*, pages 1–6, April 11–13, Krakow, Poland. IEEE Press.

Rui, S., Ying, H., Dong-Hui, W., Tie-Jun, Z., Qian, Y., and Chao-Huan, H. (2003). A 32-bit hybrid microprocessor design for multimedia applications. In *Proceedings 5th International Conference on ASIC*, page (2.3), Beijing, China, October 21–24. IEEE Press.

Samudrala, P., Ramos, J., and Katkoori, S. (2004). Selective triple Modular redundancy (STMR) based single-event upset (SEU) tolerant synthesis for FPGAs. *IEEE Transactions on Nuclear Science*, 51(5, Part 4):2957–69.

Sezer, S., Woods, R., Heron, J., and Marshall, A. (1998). Fast partial reconfiguration for FCCMs. In *Proceedings of the IEEE Symposium on FPGAs for Custom Computing Machines (FCCM'98)*, page 318, Washington, DC. IEEE Computer Society.

Shannon, L. (2002). *Impact of Intellectual Property Cores on Field Programmable Gate Array Designs*. National Library of Canada.

Stoica, A., Zebulum, R., Keymeulen, D., Tawel, R., Daud, T., and Thakoor, A. (2001). Reconfigurable VLSI architectures for evolvable hardware: from experimental field programmable transistor arrays to evolution-orientedchips. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 9(1):227–232.

Tan, E. J. and Heinzelman, W. B. (2003). DSP architectures: past, present and futures. *SIGARCH Computer Architecture News*, 31(3):6–19.

Chapter 2
Multiagent-Based Fault Tolerance Management for Robustness

Rosa Laura Zavala Gutierrez and Michael Huhns

Abstract Despite the use of software engineering best practices and tools, it would be very risky to assume that the software that is developed today is fault-free. Moreover, we have to consider the fact that the software could face unexpected situations not considered during its design. Robustness is a highly desirable and sometimes indispensable software requirement, especially for critical systems, where the consequences of a system failure can be catastrophic. This chapter outlines existing fault tolerance techniques, followed by a discussion of the potential that multiagent systems have to enhance the design of robust, fault-tolerant systems, thereby improving large-scale, critical, and complex system reliability.

2.1 Introduction

Making software robust—i.e., enabling it to be performed without failure under a wide range of conditions—has always been a desirable outcome, particularly in critical applications (e.g., control of aircraft, chemical plants, nuclear plants, financial transactions, or medical assistance) where the consequences of software failing go beyond simply annoying the users, and can cause huge financial losses, serious injuries, or the loss of life. The importance of software as an important contributor to catastrophic events has been well documented (e.g., [Leveson, 1995]).

There is a large compendium of work in software engineering addressing the problem of producing reliable and robust software systems. Methods, processes, technologies, and tools have been proposed for good software design and development. These approaches have been very beneficial in improving our ability to produce better software. Nevertheless, despite significant contributions, the ever increasing complexity and pervasiveness of software systems creates the need for continuous improvement. Additionally, the more dependable software systems

R.L.Z. Gutierrez
Department of Computer Science and Engineering, University of South Carolina, 301 Main Street, Columbia, SC 29208, USA
email: zavalagu@engr.sc.edu

A. Schuster (ed.), *Robust Intelligent Systems,* DOI: 10.1007/978-1-84800-261-6_2,
© Springer-Verlag London Limited 2008