Bit Impact Factor: Towards making fair vulnerability comparison

Serdar Zafer Can a, Gulay Yalcin b,⇑, Oguz Ergin a, Osman Sabri Unsal b, Adrian Cristal b,c

a TOBB University of Economics and Technology, Söğütözü Avenue 43, Söğütözü, Ankara 06560, Turkey
b Barcelona Supercomputing Center, c/ Jordi Girona, 29 08034 Barcelona, Spain
c I3A – CSIC – Spain National Research Council, Campus de la UAB, E-08193 Bellaterra, Catalonia, Spain

A R T I C L E   I N F O

Article history:
Received 30 October 2013
Revised 26 April 2014
Accepted 27 April 2014
Available online 10 May 2014

Keywords:
Architectural vulnerability factor
Soft errors
Vulnerability
Fault injection

A B S T R A C T

Reliability is becoming a major design concern in contemporary microprocessors since soft error rate is increasing due to technology scaling. Therefore, design time system vulnerability estimation is of paramount importance. Architectural Vulnerability Factor (AVF) is an early vulnerability estimation methodology. However, AVF considers that the value of a bit in a clock cycle is either required for Architecturally Correct Execution (i.e. ACE-bit) or not (i.e. unACE-bit); therefore, AVF cannot distinguish the vulnerability impact level of a bit. We introduce Bit Impact Factor metric which, we believe, will be helpful for extending AVF evaluation to provide a more accurate vulnerability analysis.

© 2014 Elsevier B.V. All rights reserved.

1. Introduction

Transient faults such as bit flips mainly caused by particle strikes, are important problems in a digital system design [1,2]. These particle strikes do not result in permanent faults but may lead to system crashes, and hence, are termed as “soft errors”. It is predicted that the soft error problem will increase in the future systems since, in every new generation of manufacturing technology, feature sizes decrease; consequently error susceptibility of digital circuits increases [3]. This increasing soft error rate makes reliability a major design concern in contemporary microprocessors.

The vulnerability of the system to soft errors should be quantified as early as possible at design time, so that required precautions can be taken. Also, it is important not to overestimate/underestimate the vulnerability of the system due to the associated performance/power overheads.

Mukherjee et al. define the Architectural Vulnerability Factor (AVF) of processor components to provide early reliability estimation [4]. AVF analysis is implemented based on the fact that systems are known to mask some of the faults either at the circuit level or architectural level and these faults do not propagate to the final outcome of a program. Quantifying this masking effect allows adjusting the level of error protection in the design of a digital system.

AVF is defined as the average ratio of the bits in the system that are required for Architecturally Correct Execution (i.e. ACE-bits) at a given clock cycle. AVF analysis does not care about the process of a bit flip creating the error, but it rather qualifies the outcome: if the flip on a stored bit results in a system level visible error, the flipped bit is defined as ACE-bit. In reality, a bit flip may lead to an unwanted outcome through different paths.

Mukherjee et al. advocate that the AVF of the whole processor can be calculated by summing the AVFs of all structures multiplied by their area normalized with respect to total chip area. However, this assumption introduces discrepancies in the vulnerability especially for systems with many components or for long-running workloads [5]. This is mainly due to two reasons: (1) AVF always assumes that all of the ACE-bits in a processor component or in an instruction have the same impact on vulnerability. (2) AVF assumes that ACE-bits in different components are equally important for the vulnerability. Due to these limitations, it is not possible to make an apple-to-apple reliability comparison between hardware components (e.g. register file, issue queue) or different parts of a hardware component (e.g. data or tag area of the cache).

In this study, our goal is investigating a new dimension that provides a comparison between different bit types and a comparison between different hardware components. To this end, we start from the fact that not all of the bits belong to the same field type, and we show that the impact of faults occurring on different fields are different.

For instance, a single bit fault occurring on the immediate value of an instruction may cause the result of the instruction to be faulty in one bit position. However, if a failure occurs in the source register...
identifier, it causes reading completely different source value from the wrong register which may cause multiple bit failures in the result of the instruction.

We first classify bit types in several hardware components (i.e., Register File, Reorder Buffer, and Issue Queue) in terms of vulnerability level, and we examine impact of a single bit flip in each class. We define Bit Impact Factor (BIF), which shows the vulnerability level of a bit, and indirectly allows the quantification of the relative vulnerability across processor components and component fields.

The contributions of this study are:

- We classify bit types in several hardware components (i.e., Register File, Reorder Buffer, and Issue Queue) in terms of vulnerability level.
- We define Bit Impact Factor (BIF) which shows the average number of bits affected in the next dependent component when the defined bit fails.
- We extend AVF with BIF dimension in order to provide more accurate vulnerability analysis and to allow connecting the vulnerability of different hardware components.

2. Quantifying soft error impact factor

In this section, we first explain AVF principles. Then, we classify bits within microarchitecture classified according to the information they store and on the vulnerability impact of bits.

2.1. Architectural Vulnerability Factor (AVF)

Even though a bit is flipped as a result of a particle strike, the program running on the processor may not be affected. This is because of the fact that the values of many bits inside the processor components are not required for the correct execution. The bits that are needed for the Architecturally Correct Execution (ACE) of the running program were previously termed as the “ACE-bits” by Mukherjee et al. [4]. Each structure inside a processor contains a variable number of ACE bits over program execution time which determines the level of its vulnerability to soft errors. If the number of ACE bits in the structure is high, it is more probable that a bit flip will result in an error. The level of a component’s vulnerability to soft errors is termed as Architectural Vulnerability Factor (AVF). AVF is a useful metric in the design time of a processor in order to verify that the processor meets the reliability targets and is calculated with the following equation:

\[
AVF = \frac{\text{Average Number of ACE Bits in a Hardware Structure}}{\text{Total Number of Bits in the Hardware Structure}}
\]  

(1)

2.2. Bit classification

In all of the vulnerability factor definitions, the objective is to decide whether or not a bit flip results in a visible fault by a predefined time. This approach assumes that there is no level of vulnerability; a bit can either be vulnerable or not. In reality, the impact of a fault in different fields of a processor component differs. Some faults may result in more bit flips during the fault’s propagation and lead to the ultimate system crash faster, while others may be localized into a single bit.

In Fig. 1, we present a case in an ACE instruction. In the instruction, any flips in some bits are more likely to cause an error than other bits. In the figure, the code multiplies a number by 3 and if the result is lower than 8, the number is increased by 2. At the end of the correct execution, R1 holds 4. After the comparison operation (i.e., I3), R0 becomes un-ACE since, later, it is overwritten by 0.

In the figure, we present two faulty cases for the multiplication instruction (i.e., I2). In the first case, the fault affects the immediate value and the number is multiplied by 2 instead of 3. The change in the immediate value does not change the result of the branch instruction and the final result in R1 still becomes 4. In the second case, the fault affects the destination register identifier and the result is written to R1 instead of R0. This fault does not affect the result of comparison operation either and branch is still not taken. However, the value written to R1 is quite different than the correct one. Thus, in the example, a possible fault in the destination register identifier is more vital than a fault in the immediate value.

The number of faulty bits that a bit flip causes can be used as a metric to show the impact of an error as it quantifies the maskability of the fault. As in the previous example, the probability of a maskable fault in the immediate value is higher than a maskable fault in the destination register identifier. Thus, we define Bit Impact Factor (BIF) which shows the average number of bits affected in the next dependent component when the defined bit fails.

A faulty value does not affect the system unless it is read from its location and used for a calculation. All of the read data will eventually arrive at the functional unit and an operation will be performed on it. Therefore the functional unit is the actual location where the degree of masking is determined by the number of faults arriving to the inputs of the unit and by the operation performed on the inputs (shown in Fig. 2). We classify information types stored in processor components according to the effects of them on the input parameters of the functional units. The classification is as following:

1. Regular Data (registers, immediate values, etc.)
2. Opcodes (ALU op, Function bits)
3. Source Identifiers (source physical register tags)
4. Destination Identifiers (destination tags, structure entry ids such as reorder buffer id and LSQ id)
5. Control Information (These bits hold the status of the stored information like ready and valid)
6. System Level Bits

In this section, we explain these groups with a Single Event Upset (SEU) model.

2.2.1. Regular data

Each bit, other than the control bits, of the register file as well as immediate values belong to the regular data group. When a bit flip occurs on any of the bits, a new value that is Hamming distance 1 apart from the original value is created. This new value may create a single bit difference at one of the source inputs of the functional unit as shown in Fig. 3.

Many operations can mask a single bit fault and the fault may not propagate to the result value depending on the operation. If any fault is observed at the output of the functional unit, it propagates to the next dependent instruction as the result value is stored inside the register file and used as a source operand.

2.2.2. Opcode

Opcode is passed to the processor by the compiler and is decoded to generate the instruction payload by the processor. Depending on the design of the instruction set, many opcodes may be similar to each other in terms of the opcode value performing similar operations. For example, in Alpha instruction set, opcodes between 14.02A and 14.7 EB perform variants of the SQRT operation. Therefore changing the opcode in a single bit location sometimes leads to no error and is totally masked.
2.2.3. Source identifiers

Some structures inside the processors such as the physical register identifiers hold pointer values pointing to the location of the actual data instead of holding the data themselves. For instance, Issue Queue (IQ) stores physical register identifiers so that the instruction can grab its operands before it proceeds to the execution phase. Also Rename Table uses these identifiers to track the dependencies among instructions.

Fig. 4 shows the effect of a fault occurring on a 7-bit physical register identifier (identifying 128 registers) inside the IQ. Depending on the location of the fault in the source tag 1, there are 7 erroneous locations that the tag field can point to. Since the source value is read from an erroneous register, the difference between the correct source value and the erroneous value will be high (i.e. different in the multiple bit position). Also, even if the register file is protected with ECC, once the identifier points to a wrong location, where the value is perfectly correct in its own context, ECC would not help at all correcting the fault. Obviously, the impact of a source pointer (i.e. source register pointer) on vulnerability is higher than the impact of regular data (i.e. register, immediate). This is similar to the fact that, in terms of reliability, tag area of the cache is more critical than the data area since a faulty address may lead to faulty match or mismatch. For register identifiers, the situation is even more drastic than the cache tags since any fault in a register identifier definitely leads to an access to a wrong register.

2.2.4. Destination identifiers

Each and every result producing instruction needs to write its result to a location pointed by the destination register identifier. Also index values for the entries of all of the structures that store data are of destination identifier type. For example, the reorder buffer id that is assigned to each instruction at decode stage is used to locate the entry when the instruction needs to update its condition flags after execution. A fault on a destination identifier corrupts two data locations at the same time: (1) The original location that needs to be written is not updated and becomes corrupted and (2) a wrong location is overwritten with the newly generated data that is not supposed to be written to that location. Fig. 5 shows an example where a destination register identifier inside the issue queue gets struck by a particle. The result value created by the instruction is not written to its correct destination. Therefore, it causes the value to be corrupted at its least significant bits that were supposed to be modified, effectively creating a 3-bit fault. At the same time the instruction updates a wrong location modifying 7 bits locations that were not supposed to be changed, effectively creating a 7-bit fault. Consequently, a single bit flip in the destination register identifier transforms into a 10-bit fault in the register file. Because of the fact that a destination identifier corrupts two locations at once, bit flips on these fields have more immediate impact than a regular data field or a source identifier.

Fig. 1. The figure shows a case that the correction of the destination register identifier is more essential than the correction of the immediate value. In the example, grey areas are un-ACE at the time.

Fig. 2. Inputs and outputs of a function unit.

Fig. 3. Effect of a fault occurring on a regular data bit.

Fig. 4. Effect of a fault occurring on a source pointer field.
2.2.5. Control information

Apart from the actual data and the pointer values, there are some control bits that indicate the status of some other bits. One example is the valid bits that indicate if the information stored inside the entry is valid or not. A particle strike on this single bit results in the loss of the entire row, effectively losing information. Depending on the number of bits that the control bit is assigned to, a particle strike on these bits transforms into a large number of faults. For example, the valid bit of a 64-bit register will corrupt all of the 64 bits at once and hence has more immediate impact in case it is flipped erroneously. Since the impact of a control bit is totally dependent on the microarchitecture and the purpose of the bit in the specific component, we do not provide any statistics in this paper.

2.2.6. System level bits

Processors hold architectural state that allows the communication with Operating System (OS). Some bits inside a processor are reserved to hold the required data to indicate an exceptional state that mandates the transfer of control to the OS. A particle strike on these bits immediately transforms to a system level visible error and the OS may try to recover from an exception that, in fact, does not exist. Since these bits affect the whole system rather than a localized group of stored data, they are more vulnerable than any aforementioned data types. The quantitative analysis of these system bits is beyond the scope of this work.

2.3. Bit Impact Factor (BIF)

We define Bit Impact Factor (BIF) which shows the level of vulnerability impact of a bit in the result of the functional unit. In order to calculate BIF, we measure the hamming distance between the faulty result and the correct result of the first dependent instruction, then, we divide this distance by the total number of bits of the result. BIF presents the maskability probability of a possible fault in the defined bit (i.e. the lower the BIF, the higher the maskability probability). Thus, it can be used for tuning the AVF calculation.

In order to measure the BIF of regular data, we execute each instruction with a faulty data which has a fault in one bit position (i.e. 64 different values for a 64-bit machine). Then we measure the average of the hamming distances between the correct result and the faulty results. Similarly, we calculate the BIF of the source register identifiers as the average of hamming distances between the correct result and the faulty results when the source register is read from one of the faulty locations (i.e. 7 possible faulty locations for a system with 128 registers). For instance, if the instruction reads the source register from R1, we repeat the execution of the instruction by using R0, R3, R5, R9, R17, R33, R65. We compare the results of these 7 executions with the correct execution and calculate the average values of the hamming distances. For destination register identifiers, we need to measure two hamming distances: (1) the hamming distance between the result of the instruction and the old value of the correct destination and (2) the average of hamming distances between the correct result and the possible faulty destinations (i.e. 7 faulty destinations for a register file with 128 entries). Finally, we calculate the BIF of destination register identifiers by summing these two distances.

2.4. Connection between AVF and BIF

In order to extend the AVF metric into a new dimension which demonstrates the vulnerability level of bits (instead of expressing that a bit is vulnerable or not), we define AVF $\text{weighted}$. In this section, we will show how we define and calculate AVF $\text{weighted}$. First, AVF expresses the probability that the given hardware structure fails due to a single bit failure. Let’s define Architectural Reliability Factor (ARF) which presents that the hardware structure can operate reliably although there was a single failure. ARF can be calculated as:

$$\text{ARF} = 1 - \text{AVF}$$  \hspace{1cm} (2)

BIF express the number of bits changes after executing the related instruction and is calculated as:

$$\text{BIF} = \frac{\text{Hamming Dist. Between the Correct Result and Faulty Results}}{\text{Total Number of Bits in the Result of Functional Unit}}$$  \hspace{1cm} (3)

Thus, BIF expresses the probability that a fault would produce a failure in the output of the execution. Thus, $\text{ARF}_{\text{weighted}}$, the reliability factor with the BIF metric, can be calculated as:

$$\text{ARF}_{\text{weighted}} = \text{ARF} \times (1 - \text{BIF})$$  \hspace{1cm} (4)

Finally,

$$\text{AVF}_{\text{weighted}} = 1 - \text{ARF}_{\text{weighted}}$$  \hspace{1cm} (5)

Thus;

$$\text{AVF}_{\text{weighted}} = 1 - (1 - \text{AVF}) \times (1 - \text{BIF})$$  \hspace{1cm} (6)

In this formula, our insight is that if a single bit fails, there is a possibility that further instructions can mask this single bit fail. We argue that fast propagating faults are less likely to be masked by the following instructions in the execution. Thus, we include BIF in the AVF calculation (i.e. $\text{AVF}_{\text{weighted}}$) which provides a tuning effect and fair comparison between different bit values. Note that, $\text{AVF}_{\text{weighted}}$ Cannot be used for calculating Failures in Time (FIT) or Mean Time To Failure (MTTF) yet.

3. Evaluation

We ran SPEC 2006 benchmark applications [6] on the MSIM microarchitectural simulator [7] with the simulation parameters given in Table 1 to observe the Bit Impact Factor (BIF) of register values, source and destination register identifiers and opcodes. We also measured AVF of applications and we calculate the $\text{AVF}_{\text{weighted}}$ by including BIF into AVF calculation. We compare AVF and $\text{AVF}_{\text{weighted}}$ with the vulnerability measured by the fault injection methodology. We inject 100 faults to each structure per application at the execution level. We use MSIM for fault injection as well. Cho et al. [8] proved that application level fault injections...
do not show as accurate results as the Register Transfer Level (RTL) fault injections such as IVM [9]. In this study, we inject faults in the simulator level as in IVM, however, we used a micro-architectural simulator instead of RTL. This is because AVF is calculated by using micro-architectural simulators and for a fair comparison we use the same tool.

In Fig. 6, we present the hamming distances between the correct result and the faulty results for (1) regular register data, (2) opcodes, (3) source register identifiers and (4) destination register identifiers. In the experiment, we apply all possible fault combinations produced after a single fault in the related structure. Then we measure the average number of bit-faults observed at the output of the function unit. This shows the impact level of a Single Event Upset (SEU) in the related bit type on the vulnerability of the system. Fig. 6(a) shows that single bit error in the input register data of a function unit would lead failing of 2 bits in the output, on average. This result is similar to the one presented by Li et al. [10] in which single bit failure in an integer function unit mostly (i.e. around 90% depending on the failure type) results in single bit failure in the output and in rare cases it presents multi-bit failures. Fig. 6(b) reveals that on average, a single bit flip on the opcode field results in a 6-bit error at the output of the function unit. Note that this result is very dependent on the design of the microarchitecture and the instruction set of the processor. Fig. 6(c) demonstrates that the BIF of source register identifiers is, on average, 16/64 (for a 64-bit ALU) which indicates that register identifiers has 8× higher impact on vulnerability than the regular register data. For destination register identifiers, we measure two distances: (1) The distance between the old result in the correct destination and produced result. (2) The average of the distances between the produced result and the values in the possible wrong destinations. The total of these two distances form the BIF of the destination identifiers as we showed in Fig. 6(d). The figure presents that destination identifiers have 14× higher impact than regular registers.

In Fig. 7, we compare the average AVF and AVF-weighted values of source identifiers, destination identifiers and register values. In the figure, we also present the vulnerability level of these structures according to the fault injection experiment which measures the vulnerability level of hardware structures more accurately with the cost of high simulation time. We measure the AVF of the executed instructions while we inject faults in the execution stage of the pipeline. The figure presents that register identifiers (source or destination) are more vulnerable than the data stored in the register. Moreover, it also presents that including BIF into AVF calculation (i.e. AVF-weighted) presents closer results to the vulnerability calculation of fault injection.

4. Related work

In this section we explain the previous studies on calculating the vulnerability of the system components.

Mukherjee et al. [4] introduce Architectural Vulnerability Factor (AVF) and Architecturally Correct Execution (ACE) analysis to measure a processor’s failure rate caused by soft errors early in the

| Table 1 | Simulation parameters. |
|-----------------------------------------------|
| Parameter | Configuration |
|-----------------------------------------------|
| Machine width | 4-Wide fetch, 4-wide issue, 4-wide commit |
| Window size | 80 Entry load/store queue, 128 entry ROB 128 entry register file |
| Function units | 2 Arithmetic logic, 2 floating point 1 INT Mult, 1 FP Mult |
| L1 Cache/ | 128 KB, 4 way set associative, 64 bytes line, 1 cycles hit time |
| L2 Cache unified | 256 KB, 16-way set associative, 64 bytes line, 6 cycles hit |

Fig. 6. The figure presents the average of hamming distances between the correct result and the possible faulty results for regular register data (a), opcode (b), source identifier (c) and destination identifier (d).
design process. They classify the unACE bits due to microarchitectural (i.e. idle, invalid or mis-speculated states, predictor structures) or architectural (i.e. NOPs, performance enhancing instructions, dynamically dead instructions or logical masking) reasons.

Li et al. [5] show that for the combination of a large system and a long-running workload on it, the AVF calculation may show a discrepancy from the accurate vulnerability. In order to converge to the accurate vulnerability of the processor, Wang et al. [11] extend the list of unACE bits. They show that Y-bits, control instructions and unused fields should also be classified as unACE. Biswas et al. [12] present even further detailed AVF analysis which shows how to calculate AVF accurately.

Several studies present how to calculate AVF for different hardware components. For instance, Biswas et al. [13] compute AVF for address-based structures such as caches, TLBs and store buffers for both tag and data areas in these structures. Also, each address tag is checked in the store area with the hamming distance of 1 in order to predict if a single bit fail in the tag causes a faulty match or mismatch in the structure. This methodology is used in [14] to show the reason of super-linear increase of vulnerability of L2 caches when the area of L2 cache is increased linearly.

AVF generally quantifies the vulnerability of bits at microarchitectural level. Sridharan and Kaeli extend this definition to architectural state by defining the Program Vulnerability Factor (PVF) [15]. PVF only observes the vulnerability at ISA level. Similarly Lee et al. [16] propose statically calculating the AVF of the register file. These schemes allow the quantification of the vulnerability of program segments since the vulnerability calculation is now independent of the microarchitecture itself. Quantifying more vulnerable parts of a program allows using redundancy only on targeted vulnerable segments of a program.

In order to establish the connection between the AVF and PVF, Sridharan and Kaeli extend the PVF study and define the Hardware Vulnerability Factor (HVF) [17]. HVF quantifies the vulnerability of only hardware by dividing the AVF into architectural and microarchitectural components. HVF gives more insight to hardware designers in pre-design stages, as HVF is not sensitive to operations that are masked at a program level, which can obscure changes in hardware behaviour. HVF also allows fast computation of AVF from PVF.

As well as architecture and workload, vulnerability of processor also depends on the device technology used. Fu et al. [18] include device parameters such as threshold voltage and transistor length in vulnerability calculation in order to characterize the impact of the process variation on reliability.

AVF assumes that all computations are equally important for applications. However, this assumption is not true especially for graphic applications. Sheaffer et al. [19] introduce Visual Vulnerability Spectrum (VVS) in order to analyze the impact of soft errors on graphic processors.

Besides simulator based AVF calculations, some other studies [20–23] focus on on-line AVF estimations after design time while the processor is actually in use in order to provide a dynamic adaptation of reliability solutions.

Since the vulnerability of the processor depends on the application, [24] focuses on meeting the need for a stressmark that can measure the observable worst-case soft error rate (SER). The code generator formed takes the parameters as input and generates a candidate stressmark as an output.

Non of these previous studies considers the fact that two different ACE bit may impact the vulnerability of the system differently. In this study, we extend AVF in this dimension.

5. Conclusion

In this study, we argue that every bit in the hardware does not have the equal vulnerability impact. We introduce a new metric; the Bit Impact Factor (BIF) which shows the impact of a single bit error in a hardware component by measuring the average number of errors in the result of the execution unit. We also include the BIF metric in the AVF and introduced AVFweighted to provide an accurate vulnerability estimation. We believe that AVFweighted will help comparing the vulnerability of different hardware structures.

Acknowledgment

This work was partially supported by the Scientific and Technological Research Council of Turkey (TUBITAK) under research Grants 112E004. The work is performed in the framework of COST ICT Action 1103 “Manufacturable and Dependable Multicore Architectures at Nanoscale”.

References


Serdar Zafer Can received the BS degree in electrical and electronics engineering from TOBB University of Economics and Technology, Ankara, Turkey, in 2012 and he is now MS student in computer engineering in TOBB University of Economics and Technology, Ankara. His current research interests include computer architecture, reliability, VLSI design and reconfigurable architectures.

Gulay Yalcin is a PhD student at Universitat Politecnica de Catalunya and a researcher student in Barcelona Supercomputing Center. She holds BS degree in Computer Engineering from Hacettepe University and MS degree in Computer Engineering from TOBB University of Economics and Technology. Her research interests are reliability and energy minimization in computer architecture.

Osman Sabri Unsal is co-leader of the Architectural Support for Programming Models group at the Barcelona Supercomputing Center. Dr. Unsal is also a researcher at the BSC-Microsoft Research Centre. He holds BS, MS, and PhD degrees in electrical and computer engineering from Istanbul Technical University, Brown University, and University of Massachusetts, Amherst, respectively.

Adrián Cristal received the “licenciatura” in Computer Science from Universidad de Buenos Aires (FCEN) in 1995 and the PhD. degree in Computer Science in 2006, from the Universitat Politécnica de Catalunya (UPC), Spain. From 1992 to 1995 he has been lecturing in Neural Network and Compiler Design. In UPC, from 2003 to 2006 he has been lecturing on computer organization. Currently, and since 2006, he is researcher in Computer Architecture group at BSC. He is currently co-manager of the “Computer Architecture for parallel paradigms”. His research interests cover the areas of microarchitecture, multicore architectures, and programming models for multicore architectures. He has published around 60 publications in these topics and participated in several research projects with other universities and industries, in framework of the European Union programmes or in direct collaboration with technology leading companies.