Project Number: FP7-611404

D3.1 – Report on major classes of hardware components

Authors

M. Kaliorakis, S. Tselonis, N. Foutris, D. Gizopoulos

Version 1.7 – 5/8/2014

Lead contractor: University of Athens
Contact person: Dimitris Gizopoulos University of Athens Department of Informatics & Telecommunications Panepistimiopolis, Ilissia 15784, Athens (Greece) Tel. +30 210 7275145 Fax. +30 210 7275214 E-mail: dgizop@di.uoa.gr
Work package: WP3
Affected tasks: T3.1

Nature of deliverable[1]	R	P	D	O
Dissemination level[2]	PU	PP	RE	CO

[1] *R: Report, P: Prototype, D: Demonstrator, O: Other*

[2] ***PU**: public, **PP:** Restricted to other programme participants (including the commission services), **RE** Restricted to a group specified by the consortium (including the Commission services), **CO** Confidential, only for members of the consortium (Including the Commission services)*

COPYRIGHT

INDEX

COPYRIGHT **2**

INDEX **3**

Scope of the document **4**

1. Hardware components reliability characterization **5**

1.1. Faults in hardware components **5**

1.2. The role of hardware components in the overall computing system reliability **8**

2. Lists and characteristics of hardware components **10**

2.1. Major hardware sub-systems: coarse granularity **10**

2.2. Breakdown of major hardware components – finer granularity **13**

2.2.1. Microprocessors sub-components 13

2.2.2. Accelerators sub-components 15

2.2.3. Memories sub-components 17

2.2.4. Peripherals sub-components 19

2.2.5. Interconnects sub-components 20

3. Hardware reliability and faults behavior classification **21**

4. Reliability-related characteristics of hardware components **21**

5. Public studies of hardware components reliability **21**

6. Candidate components and tools for CLERECO **22**

7. Conclusions **25**

8. Acronyms **26**

8.1. List of acronyms **26**

9. References **27**

Scope of the document

This document is an outcome of Task T3.1, "**Major classes of hardware components in future computing systems**", described in the description of work (DoW) of CLERECO project under Work Package (WP) 3 (WP3 – "Hardware components reliability characterization").

As described in the DoW, in Task T3.1, the five major classes of hardware components (processors, accelerators, memories, peripherals, interconnects) will be analyzed and decided including all possible variants employed in the computing continuum. T3.1 is an important initial task of the CLERECO project because the usefulness of the system reliability model depends on the reliability characterization of all existing hardware components. The task also identifies potential representative hardware components that are either publicly available or can be acquired during the course of the project, and can be used for the validation of the system methodology and the demonstration of the project.

This document aims to list typical components employed in the design of the hardware layer of both embedded computing (EC) systems and high performance computing (HPC) systems.

1. Hardware components reliability characterization

Forthcoming computing systems are expected to be highly unreliable, due to the combination of the extreme scaling process (moving towards 12nm manufacturing process nodes and beyond), the high design complexity (expanding design validation space), and a constant time-to-market demand (the interval between design phase and massive production remains roughly the same). Furthermore, single-event upsets (SEUs) from particle strikes, wear-out and aging behavior throughout the operational period of a system are expected to further reduce circuit reliability.

For instance, the failure probability of six-transistor SRAM bit is 7.3e-09 for 32nm and 1.5e-06 for 22nm, while the failure probability of a latch is 1.8e-44 for 32 nm and 5.5e-18 for 22nm [1]. These single-bit failure probabilities are expected to increase significantly in forthcoming manufacturing nodes; [1] roadmap reports an expectation for an SRAM failure probability of 5.5e-05 in 16nm and 2.6e-04 in 12nm, while the corresponding probabilities for latches are 5.4e-10 in 16nm and 3.6e-07 in 12nm.

Reliability characterization estimates the vulnerability of design components on transient (soft), intermittent, and permanent (hard) faults and produces valuable output to the design teams to make critical decisions on the protection mechanisms. CLERECO project will build a framework for early and fast estimation of the system reliability to facilitate such important design decisions across the entire computing continuum.

1.1. Faults in hardware components

The reliability characterization of the hardware components in the CLERECO framework will consider all different types of failure mechanisms and corresponding fault models that can affect their operation. WP3 will characterize the impact of various types of faults on the hardware components of a system. These fault types are briefly presented in Table 1 below, and a detailed discussion can be found in [2], [3], and [4].

Table 1: Fault types in hardware components[3].

Fault type	Occurrence/endurance	Sources
Transient (soft)	instant bit flip; disappears on next write	radiation, thermal cycling, transistor variability, erratic fluctuation of voltage
Intermittent	stuck at logic 0 or 1 for an amount of cycles; repeated at arbitrary periods	timing violations (e.g. changes in frequency), manufacturing residues, ultra-thin oxide breakdown, wear-out
Permanent (hard)	permanently stuck at state 0 or 1	manufacturing defects, materials wear-out and device aging (degradation)

[3] *This is a top-level classification of faulty hardware behavior in terms of the duration and persistence of faults; a very large number of formal "fault models" have been defined in the literature and is used in the industry [2]. As technologies evolve, new fault models appear while older ones become obsolete and are neglected. However, the top-level classification that we focus on remains valid for several decades and is expected to be so for the foreseeable future.*

Several types of permanent faults are experienced in CMOS transistors [2], [3], [4]. In particular, permanent faults are classified as either extrinsic or intrinsic. The sources of extrinsic faults are various process-related manufacturing defects within silicon devices and their rate usually decreases over time. Intrinsic faults arise from wear-out and aging of materials and their rate typically increases with time. In general, any faulty behavior that does not change within time is called a permanent failure and the model used to describe it is called a permanent fault. A permanent fault is often modeled by assigning a fixed value to a signal or memory element (this is the classic stuck-at-0 or stuck-at-1 fault model).

Intermittent faults occur due to manufacturing residues, timing violations and ultra-thin oxide breakdown. The corresponding fault model at logic level is intermittent stuck-at, intermittent delay and intermittent indetermination [5], [6]. Intermittent faults are often manifested in bursts and are highly dependent on temperature, voltage and frequency variations[4] [6]. A storage element or a logic gate that suffers from an intermittent fault is modeled by setting it to one (stuck-at-1) or to zero (stuck-at-0) for (random) selected timing intervals.

Transistor variability, thermal cycling, cosmic rays, α-particles, humidity, vibrations, power supply fluctuations, etc. are common causes of transient (soft) faults on memory elements (called as Single-Event-Upsets, SEUs). Furthermore, as technology scales the sensitivity of logic elements to transient phenomena due to environmental and operating modes effects (called Single-Event-Transients, SETs) has also increased. Modeling transient faults can be done through flipping the content of a memory cell in a (randomly selected) clock cycle.

Figure 1 visualizes the timing behavior of a permanent (a), an intermittent (b) and a transient (c and d) fault on a memory element. In particular, whenever a permanent faulty bit (a) is read from a design, its output will always be faulty. Figure 1-(b) shows that an intermittent fault is likely to be propagated erroneously. However, when a write operation is executed within sequential activations of an intermittent fault, then any subsequent read operation will read a correct bit value. Finally, Figure 1-(c) and Figure 1-(d) visualize the effect of a transient error. In Figure 1-(c) a single event upset (a transient fault) occurs between two consecutive write operations. The latter write masks the fault from the design because it updates the stored data. Thus, the following read operation delivers a correct outcome. On the contrary, for the case depicted in Figure 1-(d) the faulty value is propagated to the output.

[4] *Deliverable D2.3 ("Definition of operating modes for future systems") elaborated in the description of work (DoW) of CLERECO project under work package 2 (WP2) lists the typical operational modes of the Embedded Systems (ES), as well as General Purpose (GP) and the High Performance Computing (HPC) systems.*

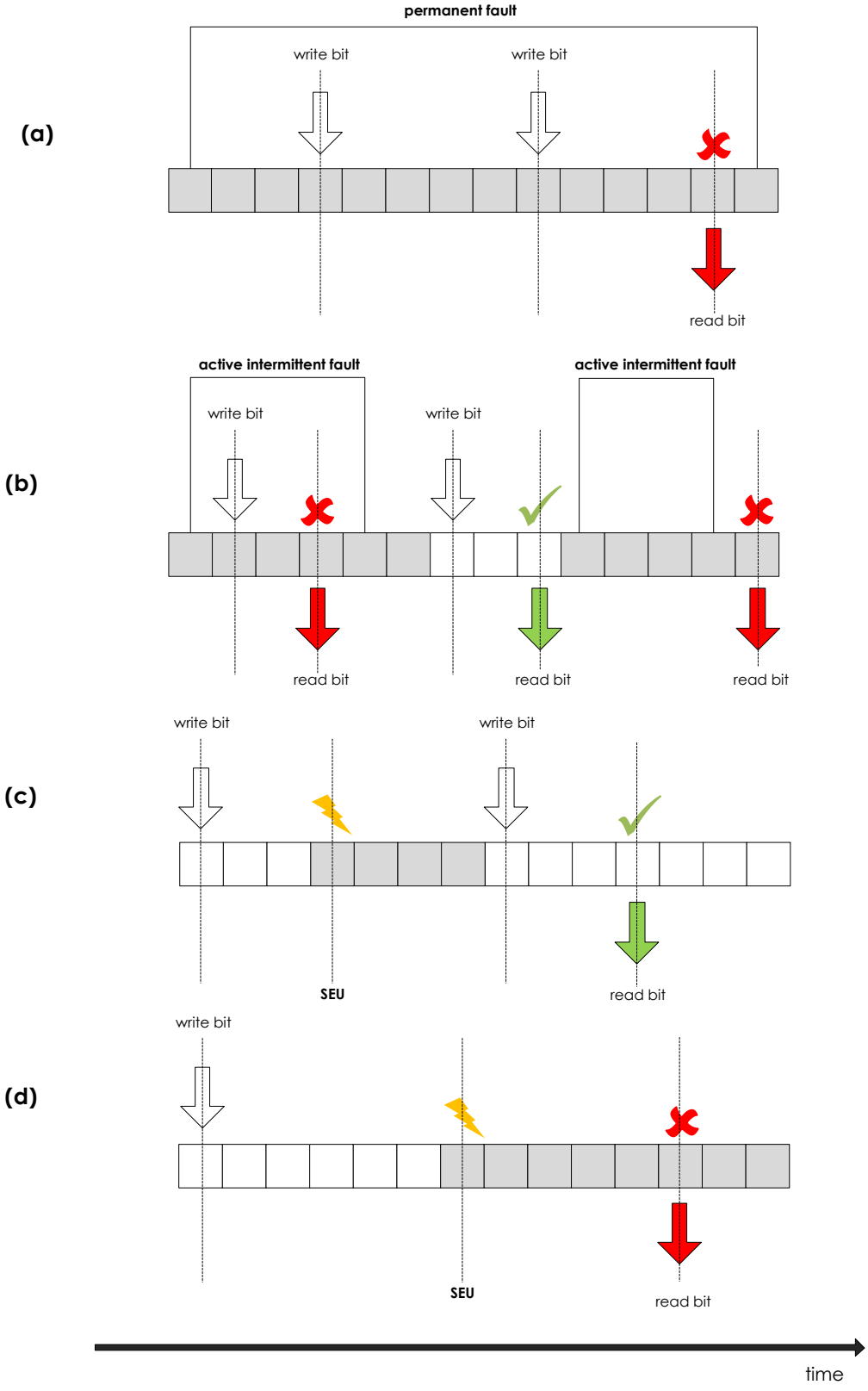

Figure 1: Permanent, intermittent, and transient fault behavior in time.

1.2. The role of hardware components in the overall computing system reliability

The reliability characterization of the hardware components of a computing system, both in the embedded and in the high-performance computing domains, is a fundamental step in the overall reliability evaluation process of a system. CLERECO aims to evaluate the combined effect of the raw technology layer reliability and the masking that the hardware architecture and software layers add on top of the technology. Figure 2 shows this basic idea of CLERECO. The low-level raw error rates of the physical devices are masked in several ways as their effects propagate to the hardware and software layers of the system stack. It is the CLERECO's goal to contribute with a full system reliability estimation methodology, which takes into consideration all these factors (technology, hardware and software) to provide an accurate estimation of the expected reliability of the system as early as possible during the design phase.

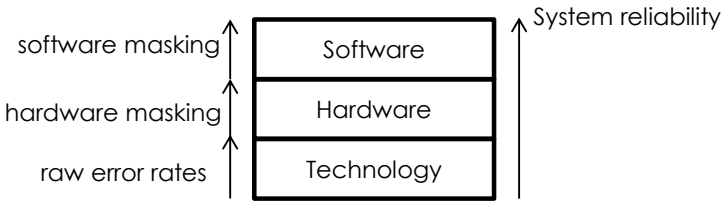

Figure 2: CLERECO combines raw error rates with masking effects of the hardware and software layers to estimate system reliability.

The System Vulnerability (or Reliability) Stack consists of three independent layers of abstraction that interact through well-defined interfaces (e.g., the ISA). The lower layer is associated with the primary elements of a circuit (e.g., transistors), and their vulnerability parameters are determined by the level of circuit's integration. The vulnerability of the hardware layer depends on the specific microarchitectural (e.g., cache levels, predictors, renaming logic, etc.) or architectural (e.g., the implementation of each instruction of the ISA) attributes of a circuit. The top level consists of software programs and the operating system, while its vulnerability depends on their characteristics. The vulnerability stack quantifies fault masking effects within an individual layer by focusing on its interfaces: a fault that does not propagate to an upper layer's interface will be masked. Thus, all the layers vulnerability measurements have to be combined to measure the full system vulnerability.

There are three kinds of masking observed in logic blocks [4], [7], [8], [9]:

- *Logical Masking*: A fault will be masked if it affects a portion of the circuit that does not logically affect the outcome of the circuit.
- *Electrical masking*: A fault can be electrically masked if the pulse created by the physical source of the fault (for example, a particle strike) attenuates before it reaches the forward latch.
- *Latch-window masking*: A fault can be latch-window masked if the pulse created by the physical source of the fault does not reach the forward latch at the clock transition.

As technology moves deeper into nanometer-scale technologies, the relative contribution of soft errors will continue to increase due to three factors [4]. First, as technology scaling rapidly decreases, the critical charge of gates becomes smaller. Second, as the clock frequency continues to increase, fewer error pulses will attenuate, decreasing the effect of electrical masking. Third, as the number of pipeline stages in modern processors continues to increase, the latch-window masking phenomenon will decrease.

Although a significant fraction of these low circuit-level faults are masked due to the aforementioned physical phenomena, still a noticeable number of faults will propagate to the next layer of the system reliability stack, i.e., the hardware (or micro-architectural) layer. The effect of these faults can be catastrophic for the proper functionality of the system.

There are several cases that a fault can be masked either in the microarchitectural or in the architectural sub-layer. Figure 3 summarizes all the masking effects on the system stack [4]. A fault can be masked in the microarchitecture sub-layer if it:

- Affects data or status bits that are either idle or invalid.
- Occurs in a bit associated with a mis-speculative execution such as branch prediction or speculative memory disambiguation.
- Occurs in dead bits. Dead bits are those bits that remain in valid state but they are not used by the processor. For instance, when an instruction is issued for the last time in the instruction queue, it may remain in valid state, waiting until the processor knows that no further reissue will be needed.

A fault can be masked in the architectural sub-layer for the following reasons:

- NOP instruction: The fault will not be masked only if it is placed in the field of opcode or destination register.
- Performance-enhancing instructions: Operations that influence the performance but not the correct execution (e.g., data prefetch instructions, branch prediction instructions).
- Dynamically dead instruction: First-level dynamically dead instructions (FDD) are those whose results are simply not read by any other instructions. Transitively dynamically dead instructions (TDD) are those whose results are used only by FDD instructions or other TDD instructions. An instruction with multiple destination registers is dynamically dead only if all its destination registers are unused.
- Logic masking: It arises when a logic or arithmetic operation masks a faulty bit. For instance, when one input of a logic-OR operation equals to one, then a fault in the other input is logically masked.

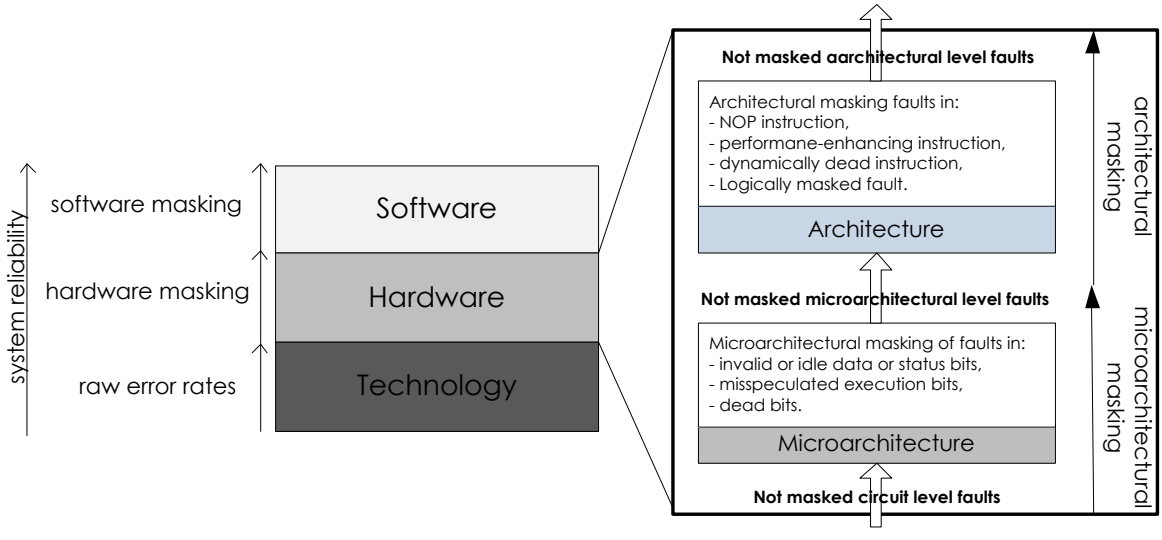

Figure 3: Masking effects on System Vulnerability Stack.

2. Lists and characteristics of hardware components

2.1. Major hardware sub-systems: coarse granularity

The five major hardware sub-systems that will be evaluated in WP3 are shown in Table 2 (as described in the CLERECO DoW). Subsequently, each of these major hardware sub-systems is decomposed in its sub-components, which will be separately evaluated from the reliability point of view.

Table 2: Major hardware sub-systems of computing systems.

Hardware sub-system	Comments	CLERECO Tasks
Microprocessors	Different RISC and CISC architectures are employed in Embedded and High-Performance computing systems[5] as well as DSP-based architectures and microcontrollers. Single-core as well as multicore and multithreaded versions of the microprocessors are being employed depending on the types of execution parallelism that the system exploits.	T3.2 and T3.7
Accelerators	Certain tasks of a computing system require special hardware components for their acceleration. Common hardware accelerators are: (a) SIMD (single-instruction multiple-data stream) accelerators; (b) Visual/Graphics Processing Units (VPUs/GPUs) either for graphics or for general-purpose processing of data-parallel tasks; (c) cryptographic cores in security-sensitive applications. Accelerators can be either discrete components in a board or system, or can be integrated with the main CPU(s) of the system in "fused" organizations, such as the Accelerated (or Advanced) Processing Units, APUs.	T3.3 and T3.7
Memories	Large parts of the hardware of computing systems are occupied by storage modules. The classic volatile storage components Static and Dynamic RAMs (SRAMs, DRAMs) are currently combined in embedded and high-performance computing systems with non-volatile counterparts (Flash memories, resistive memories of different technology such as PCMs, memristors, etc. as well as magnetic inter-component communication is realized through interconnection media that are classified as on-chip and off-chip. High-speed on-chip interconnects (bus-based architectures or Network-on-Chip (NoC) architectures are widely employed. System-level interconnects deal with larger scale off-chip memories). SRAMs and DRAMs are at the higher layers of the memory hierarchy (register files, L1/L2/L3 caches, main memory) of a computing system while the use of non-volatile memories is being exploited at different layers. Magnetic disk storage as well as optical storage is the lower level of a computing system hierarchy (these two technologies	T3.4 and T3.7

[5] *More than 20 different microprocessor vendors have contributed throughout the history of computing with unique microprocessor designs [42].*

	are gradually becoming of less importance in some computing segments like embedded systems).	
Peripherals	A diverse set of peripherals components complement the functionality of a computing system at all domains. Peripheral components consist of a digital electronics part that communicates with the system (the controller) and analog or mechanical or other parts for its main function.	T3.5 and T3.7
Interconnects	Inter-component communication is realized through interconnection media that are classified as on-chip and off-chip. High-speed on-chip interconnects (bus-based architectures or Network-on-Chip (NoC) architectures) are widely employed. System-level interconnects deal with larger scale off-chip communication.	T3.6 and T3.7

Figure 4 summarizes the hardware components of a computing system that will be considered from the reliability point of view in the CLERECO project.

All these different components have a different contribution to the overall system reliability depending on their several reliability-related parameters discussed in subsequent sections of this document. However, certain top-level components among the five categories shown in Figure 4 are to some extent more important and critical for the overall system reliability; for these components (and their sub-components) CLERECO devotes more effort on their reliability characterization. These prioritized top-level components are the Microprocessors, the Accelerators, and the Memories. All three types of components are:

- Heavily utilized during workload execution and thus faults in them are more likely to affect program execution (correctness, performance and power).
- To a large extent similarly vulnerable to the different faults types (permanent, intermittent and transient) because they are manufactured using the same underlying technologies: logic, SRAM, DRAM (and emerging technologies are investigated for all these three types of components).
- More "stable" architectures that have been employed in the HPC and EC domains, while for peripherals and interconnects, the technological landscape changes more frequently.

However this prioritization for Microprocessors, Accelerators and Memories does not mean that Peripherals and Interconnects will be neglected in CLERECO. There are certain system cases where they severely affect the system reliability and this impact should be taken into account in WP3 as well as the overall methodology in WP5.

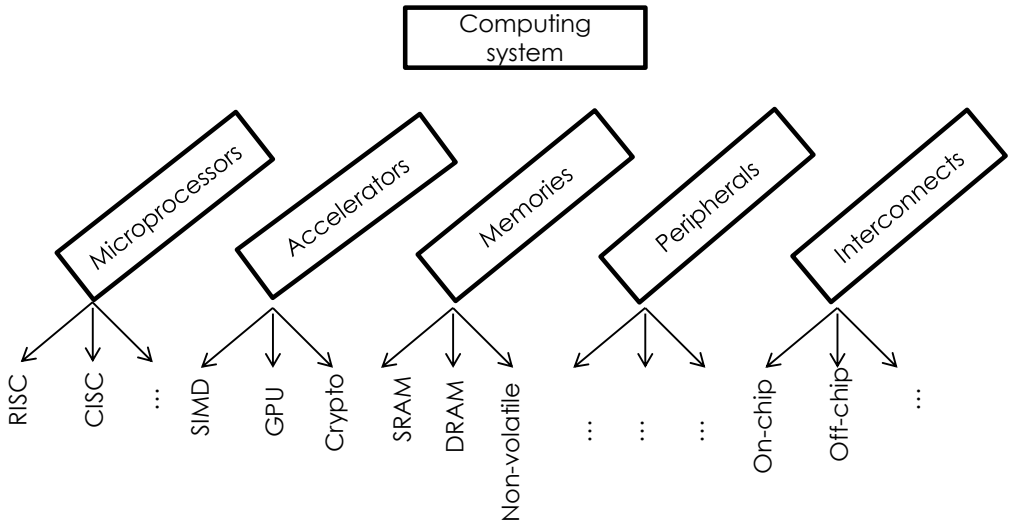

Figure 4: Hardware components in a computing system.

Table 3 lists indicative commercial products and types for the five major hardware classes (microprocessors, accelerators, memories, peripherals, interconnects) of Table 2 along with indicative manufacturers.

Table 3: The major hardware components of the computing continuum.

Hardware component	Architectural variants or product types	Manufacturer
Microprocessors	CISC, RISC, DSP, microcontrollers, multi-core, multi-threaded	Intel, AMD, IBM, ARM
Accelerators	GPUs (NVIDIA's Fermi/Kepler, AMD's ATI Radeon), Coprocessors (Intel Xeon Phi, Special Purpose Processors i.e. crypto cores), FPGAs, Embedded accelerators	NVIDIA, AMD, ARM, Imagination Technologies
Memories	SRAM, DRAM, Flash memories, resistive memories of different technology such as PCMs, memristors, magnetic memories	Kingston, Transcend, Micron, HP
Peripherals	DMA Controllers, Interrupt Controllers, VGA/LCD Controllers, General Purpose IOs (GPIO) & UARTs, PCI Express, USB, Bluetooth Controller, I2C, WNIC, SPI, 1-wire, Sensors, Actuators	HP, Epson, Freescale, ST, Bosch, TI, Panasonic, Denso, Canon, Analog devices, Infineon, Honeywell, GE, Sony, Mentor, Multicomp, Avago Technologies, Bidgeflux, Cree, Sharp, Osram, Optek Technology
Interconnects	Infiniband, Gigabit Ethernet, Cray Interconnect, Myrinet, Fat Tree, AMBA, Wishbone, SouthBridge & NorthBridge, NoC	Artemis, Sonics

2.2. Breakdown of major hardware components – finer granularity

The major hardware components listed in Table 2 consist of a large variety of internal sub-components. Each sub-component contributes to the major component's functionality as well as to its overall reliability. WP3 will separately evaluate the reliability of different sub-components of the major computing system hardware components.

2.2.1. Microprocessors sub-components

In this subsection we discuss the different microprocessor sub-components. On-chip memories (such as cache memories, TLBs, FIFOs, etc.) are not included here; they are listed on the memories sub-components section that follows. Furthermore, this subsection summarizes information regarding the purpose and usage of each sub-component as well as indicative range of their sizes.

Table 4 summarizes the microprocessors sub-components and briefly describes their functionality (some sizes information comes from the CPUs database in [42]).

Table 4: List of microprocessor sub-components.

Sub-components		Usage	Typical Size
Register file		The register file consists of registers. A register has its own id and data. Given the id of a register, its data may be accessed for read or write operation.	8 to 32 registers (8-bit, 16-bit, 32-bit, 64-bit)
SRAM	All microprocessor architectures		
Program counter (PC) or instruction pointer (IP)		The register that contains the address of the instruction being executed at the current time.	Depends on the available memory size
Latch	All microprocessor architectures		
Branch predictors		Speculates the branch direction, depending on prior branches history.	8K to 32K entries (1-bits, 2-bits per entry)
SRAM	All microprocessor architectures		
Branch Target Buffers		Speculates the branch address, depending on prior branches history	Set-associative or direct mapped array. Size can vary from 512 to 2048 entries (each entry size depends on the PC size)
SRAM	All microprocessor architectures		
Return Address Stack		Speculated the return address on function return	8, 16, 32 entries
Flip-flop	All microprocessor architectures		
Buffer		Queue temporally data (FIFO, or Circular buffer)	8, 128 entries
SRAM or Flip-flop	All microprocessor architectures		

Microcode storage /Firmware		Usually, microcode or firmware is a set of hardware-level instructions involved in the implementation of higher level machine code instructions in many processors.	Unknown; large range
ROM, SRAM, flash memory	All microprocessor architectures		
Fetch buffer		Queues the instruction fetched from the instruction cache.	16, 32 entries
SRAM	All microprocessor architectures		
Reorder buffer		Stores the program order of the committed instructions.	64, 128 entries
SRAM	All microprocessor architectures		
Load buffer		Stores data from memory.	32, 64 entries
SRAM	All microprocessor architectures		
Store buffer		Saves data send to memory.	32, 64 entries
SRAM	All microprocessor architectures		
Instruction decoder		Decode instructions and generates the control signals that fed to the pipeline.	Unknown
Logic	All microprocessor architectures		
Instruction scheduler		Schedules instructions for execution from the functional units.	Unknown
Logic	All microprocessor architectures		
Issue queue		An instruction queue holds decoded instructions until their operands and an appropriate functional unit is available.	8, 16 entries
SRAM	All microprocessor architectures		
Arithmetic-Logic Unit[6] (ALU)		The functional unit that executes integer arithmetic and logic operations.	Input/output width equals the word length of the processor
Logic	All microprocessor architectures		

[6] *Integer and real numbers functional components, such as the ALU and the FPU, are implemented by many different arithmetic algorithms. For instance, a carry-lookahead adder, or a carry-skip adder, or even a simple ripple-carry added can be integrated into an Arithmetic-Logic Unit. Reliability characterization of such functional hardware components will consider as many as possible design alternatives of the functional units' internal structure.*

Floating-point Unit (FPU)		The functional unit that executes floating-point operations.	Single and double IEEE 754 (or custom) precision
Logic	All microprocessor architectures		
Prefetcher		Identifies repeatable memory access patterns (either instruction, or data) in a workload and issues prefetch requests to the memory system.	Set-associative or direct-mapped array, 16 to 64 entries
SRAM	All microprocessor architectures		

2.2.2. Accelerators sub-components

Accelerators (also termed often co-processors) play a very important role in modern computing systems from the low-performance end (handheld and embedded devices) to the largest supercomputers. Their role is to off-load the main CPU (or CPUs) of the system from particular tasks for which they (the accelerators) have been designed specifically to make them more efficiently (faster usually). As current and forthcoming computing systems are expected to rely more heavily on the performance and power of accelerators, the overall system reliability will also depend on their reliability. Therefore, CLERECO will emphasize significantly on the reliability characterization of important hardware accelerators across the computing continuum.

Table 5 lists different accelerators of the computing continuum along with their sub-components and their purpose and usage; typical range of their size is also provided.

Table 5: List of accelerators sub-components.

Sub-component		Usage	Typical size
Various GPUs Architecture	Register file	The register file consists of registers. A register has its own id and data. Given the id of a register, its data may be accessed for read or write operation.	8K to 64K (32-bits per register)
	Instruction buffer	Queues the decoded instructions until they get issued.	Unknown
	Scoreboard	The Scoreboard logic checks for Write-After-Write (WAW) and Read-After-Write (RAW) data dependency hazards.	Unknown
	Handlers of branch divergence[7]	The basic mechanism that enables independent branching in each thread, while maintaining SIMD execution, is hardware generated active masks that indicate which threads are active and which threads are not. Inactive threads still execute the operation because of the SIMD control but the results of the computation are	Unknown

[7] *The branch divergence occurs when threads within a single warp take different paths*

<table>
<tr><td></td><td></td><td>masked and discarded. Moreover, the re-convergence Stack is another approach for handling branch divergence.</td><td></td></tr>
<tr><td></td><td>Processing Units</td><td>The processing units are included in SIMD Processors or Multiprocessors. A SIMD processor or Multiprocessor is roughly equivalent to what NVIDIA calls a Streaming Multiprocessor (SM) or what AMD calls a Compute Unit (CU). There are several processing units for execution of integer operations, floating point (single or double precision) operations and transcendental[8] operations. Furthermore, there is a Load/Store Unit (LD/ST Unit) for handling memory accesses. However, some architectures are able to execute operations either in vector or scalar fashion.</td><td>Geforce 8400 M GS
CUDA Cores per SM | CUDA Cores
8 | 16
Tesla K20c
CUDA Cores per SM | CUDA Cores
192 | 2496
Radeon HD 5870
Stream Processing Units | 1600
AMD Radeon™ R9 Series Graphics
Stream Processing Units | up to 5632</td></tr>
<tr><td></td><td>Operand Collector</td><td>The operand collector is allocated in order to buffer the source operands of the instruction for all threads or work items in the current warp or wave-front.</td><td>Unknown</td></tr>
<tr><td></td><td>Scheduler of blocks or work groups</td><td>A block or work groups scheduler works at chip level distributing thread blocks[9] or work groups to the available Streaming Multiprocessors or SIMD Units.</td><td>1 to 4</td></tr>
<tr><td></td><td>Scheduler of warps or wave-fronts</td><td>A warp or wave-front scheduler works at the Streaming Multiprocessor or SIMD unit level distributing warps of 32 threads or wave-fronts of 64 work</td><td>1</td></tr>
</table>

[8] *Operations such as reciprocal, square root, and log are characterized as transcendental operations.*

[9] *A cuda kernel consists of blocks of threads. The number of blocks and the number of threads that compose a kernel and a block respectively are user/programmer defined variables.*

		items to the execution units of Streaming Multiprocessor or SIMD unit respectively.	
Intel® MIC	Vector Processing Unit	The VPU provides support for integer and floating-point operations. Furthermore, the VPU employs an Extended Math Unit (EMU) for the execution of transcendental operations in a vector fashion with high bandwidth. The VPU supports various memory access patterns.	16 to 32 SP operations or 8 to 16 DP operations per cycle (per core)
	Register File	The register file consists of registers. A register has its own id and data. Given the id of a register, its data may be accessed for read or write operation.	32 vector registers
	Mask Register	Mask registers allow conditional execution over 16 elements in vector instructions and merge the results to the original destination.	8
Crypto-cores	Single Core Crypto-processor	A special purpose processor that executes cryptographic algorithms.	Unknown
	Task Scheduler	The Task Scheduler dispatches cryptographic tasks on all single core crypto-processors.	Unknown
	Key Scheduler	The key scheduler takes as input the key and it calculates the subkeys.	128 bits (key size)

2.2.3. Memories sub-components

On-chip and off-chip memory components are abundant in all types of computing systems and because of their fundamental characteristics and technology can severely affect the system reliability (positively or negatively depending on the employed protection mechanisms).

Table 6 summarizes the memory/storage components of the computing continuum and their sub-components, along with their usage and a typical range of their sizes.

Table 6: List of memories sub-components

Sub-component	Usage	Typical size
Main Memory	Internal memory storage for data and instructions.	0.5GB to 64GB
Cache Memories (L1, L2, L3, L4)	Temporal store of data and instructions. Cache memories are organized in hierarchical structures, where each cache level has a different access time (depends on the distance of the cache memory from the core).	8KB to 8GB
Translation Lookaside Buffer (instruction, data)	TLB is an on-chip SRAM memory structure that translates the virtual addresses (from the core), to physical addresses (to the memory system).	128 to 512 entries
Flash	Non-volatile storage device that can	1GB to 32GB

	be erased and reprogrammed	
Universal Memory Technologies	Non-volatile memory technologies. Data store either through change the electrical resistance (i.e. resistive memories), or through altering the state of the material (i.e. phase-change memories).	
Shared/Local Memory (NVIDIA's/AMD's GPUs)	The shared/local memory is a scratch-pad memory that can be shared by threads/work items within a thread block/work group. The shared/local memory is not cached and is explicitly managed by the programmer.	16KB to 64KB (per multiprocessor)
Local/Private Memory (NVIDIA's/AMD's GPUs)	Each thread/work item has its own local/private memory. If a SIMT core runs out of resources (i.e. registers), a thread will use local/private memory for spilling. The local/private memory resides in device memory, thus local/private memory accesses have high latency and low bandwidth like global memory accesses. Local/Private memory is a cached memory.	16KB to 512KB (per thread)
Texture Memory (NVIDIA's GPUs)	Texture memory is a read only memory that can improve performance and reduce memory traffic when read operations have certain access patterns (spatial locality). Texture memory is a cached memory.	6KB to 48KB (per multiprocessor)
Constant Memory (NVIDIA's/AMD's GPUs)	Constant memory holds data that will not change over a kernel's execution. In some cases, it is preferable to use constant memory rather than global memory, reducing the required memory bandwidth. Constant memory is a cached memory.	64KB
Global Memory (NVIDIA's/AMD's GPUs)	The global memory is either readable or writable. The programmer can transfer blocks of data from host to device and vice versa via global memory while all threads (even the threads of different kernels) have access to the same global memory. Global memory is a cached memory.	1GB to 12GB
Tag Directories	Caches are kept coherent with each other by the tag directory that is distributed in all cores.	
Directory	The shared data are stored in shared memory (i.e. the directory) to maintain the coherence between caches.	

2.2.4. Peripherals sub-components

Table 7 lists major peripheral components integrated on high-performance computing and embedded systems.

Table 7: List of peripherals sub-components

Sub-component name	Usage
DMA Controller	Direct Memory Access (DMA) provides to specific hardware components access to the memory sub-system, independently from the CPU.
PIC/APIC	A Programmable Interrupt Controller (PIC) functions as a manager in an interrupt driven system environment. In modern systems, Advanced Programmable Interrupt Controller (APIC) is a family of interrupt controllers that allows the implementation of multiprocessor systems.
VGA/LCD	The VGA/LCD controller handles the low-level details of communication with a monitor.
UART	A UART (Universal Asynchronous Receiver/Transmitter) controller is the interface to serial communications.
PCI Express	Programmable Communication Interface Express (PCIe) is a high-speed interconnection device between CPU and a co-processor, an accelerator, and a hard disk.
I2C	Inter-Integrated Circuit is a multimaster serial single-ended bus used for attaching low-speed peripherals to a motherboard, embedded system, cellphone, or other digital electronic devices.
USB (1.X, 2.0, 3.0, 3.1)	The USB controller handles the data transfers between a generic local bus and a Universal Serial Bus. The USB controller supports the connection between a host computer and a peripheral i.e. USB stick, mouse, keyboard, printer, etc.
Bluetooth	Bluetooth is a wireless technology standard for data transferring between short distances. A Bluetooth controller process the data exchanged through the wireless bus.
WNIC	The Wireless Network Interface Controller is a network interface controller that uses an antenna for communication through microwaves with a radio-based computer network.
SPI	The Serial Peripheral Interface is a synchronous serial data link, operating in full duplex mode for communication in short distance via the master/slave model. i.e. embedded systems, sensors, and SD cards
1-Wire	The 1-Wire is a device communication bus system that provides low-speed data, signaling, and power over a single signal. The 1-Wire provides lower data rates but in longer range than IIC and it is used to communicate with small inexpensive devices of embedded systems as digital thermometers and weather instruments.
Sensors of different types	Temperature, Acceleration, Gyrocopes, Magneto, light, digital compasses, inertial modules, pressure sensors, humidity sensors and microphones, EMC/noise detection, over/under voltage detection, vision
Actuators of different types	LEDs, fans, switches, breakers, MOSFETs

2.2.5. Interconnects sub-components

Table 8 lists major interconnection components (classified as on-chip and off-chip interconnection networks, depending on whether they connect remote computing systems or components within a computing system) integrated on high-performance computing and embedded systems

Table 8: List of interconnects sub-components

Sub-component name	Usage	Location
Infiniband	The InfiniBand is an industry standard, channel-based, switched fabric architecture, for server and storage connectivity. 10 to 40 Gbps.	Off-chip
10 Gigabit Ethernet	Ethernet is the major LAN technology as it is currently used for approximately 85 percent of the world's LAN-connected PCs and workstations. Gigabit Ethernet allows Ethernet to scale up to 10 Gbps.	Off-chip
Wi-Fi	A local area wireless technology that enables data transfers between remote devices (IEEE 802.11).	Off-chip
Gemini Interconnect	Gemini interconnect can handle tens of millions of MPI messages per second. It is designed to complement current and future many-core processors.	On-chip
Myrinet	Myrinet is a cost-effective, high-performance, packet-communication and switching technology that is widely used to interconnect clusters of workstations, PCs, servers, blade servers, or single-board computers.	Off-chip
Fat Tree	The Fat Tree adopts the three-layered hierarchical model. The link speeds get progressively higher as moving on higher levels of hierarchy in order to prevent overload.	Off-chip
Bi-directional Ring	Bi-directional Ring Network Topology has each node in a network connected to two other nodes in the network, while the first and last nodes are connected. The messages from one node to another travel from originator to destination via intermediate nodes while the message flow is bidirectional. The intermediate nodes serve as active repeaters for messages intended for other nodes.	On-chip
Wishbone	The Wishbone Bus is a specification for a portable interface between IP cores. It is recommended for interface between cores inside a chip (FPGA, ASIC, etc.) by OpenCores.org and it is public available.	On-chip
AMBA	The ARM® AMBA® protocol is an open standard, on-chip interconnect specifi-	On-chip

	cation for interfacing of functional blocks in a System-on-Chip (SoC). It facilitates the design process of implementations with large numbers of controllers and peripherals. Defining common interface standards for SoC modules makes design re-use feasible.	
Pair of Northbridge & Southbridge	The southbridge interconnection logic handles the lower – slow- priority communications of the motherboard. On contrary, the northbridge interconnection processes tasks with high performance needs.	On-chip

3. Hardware reliability and faults behavior classification

A critical task of CLERECO project is the definition of the appropriate reliability metrics to provide an accurate, early and fast estimation of the components reliability. WP 2 is assigned the task of defining the most relevant metrics to be used in WP3, WP4 and WP5. An analysis of the available reliability metrics will be presented in CLERECO deliverable 3.2.1 (D3.2.1).

4. Reliability-related characteristics of hardware components

Accurate reliability estimation is a critical task in chip's development cycle. However, reliability evaluation of hardware components is a multi-dimensional problem, since it is affected from various design-specific and technology-related parameters. A thorough analysis of all reliability-related attributes of the hardware components will be presented in CLERECO deliverable 3.2.1 (D3.2.1).

5. Public studies of hardware components reliability

A comprehensive review of the related efforts in the publicly available literature for the characterization of the hardware components of computing systems will be presented in CLERECO deliverable 3.2.1 (D3.2.1). For completeness of the current deliverable we include the references list here as well.

6. Candidate components and tools for CLERECO

In general, there are two different categories of tools that could be used to study reliability of hardware components: microarchitectural simulators and RTL- simulators. RTL simulators have many details of circuit-level characteristics facilitating the accurate hardware reliability estimation, but the low simulation throughput is a limiting factor. Hardware emulation, such as FPGA-based RTL emulation, can accelerate circuit-level simulation. However, adequately verified designs are usually uploaded on a hardware emulator, since it is a labor-intensive task. Therefore, design vulnerability measurements cannot be extracted early enough on the chip's development cycle. Furthermore, fault modeling on a hardware emulator, such as injecting faults on FPGA arrays, has inaccuracies, compared to the RTL simulator. On the other hand, microarchitectural simulators have the ability of executing faster simulations than RTL model simulators, but they suffer from some inaccuracies in their models that make them prone to overestimations of hardware components' vulnerability factors. In WP3, we aim to work on both categories of tools for as many components as possible, in an effort to correlate the behavior of the hardware component when faults are injected in different levels of abstraction. Table 9 illustrates the most suitable, mature and widely used simulators with their major characteristics that could be used by WP3 to study reliability at a more complete representation of the HPC and EC systems, while Table 10 presents corresponding RTL models that are publicly found and could be used to study the reliability of hardware components at lower levels, again both for the HPC and the EC systems.

Table 9: Microarchitectural and System Simulators

Simulator	Characteristics
Marssx86 (http://marss86.org/~marss86/index.php/Home)	• Multicore version of PTLsim[10] • x86 microprocessor model • Cycle accurate • Full system (QEMU emulator) • Interconnects (Split-phase on-chip bus, Switch) • Write-through, write-back multi-level caches • DRAM controller
GPGPUsim (http://www.gpgpu-sim.org/)	• NVIDIA GPUs simulator
Gem5 (http://www.gem5.org/Main_Page)	• Supports Alpha, ARM, SPARC, MIPS, Power, x86 • Full system (for Alpha, ARM, x86) • Multicore
Gem5+GPGPU-sim (http://cpu-gpu-sim.ece.wisc.edu/)	• v1 is based on Alpha ISA • v2 is based on ARM ISA, not stable
FusionSim (http://www.fusionsim.ca/)	• Combines PTLsim+GPGPUsim
Multi2Sim (https://www.multi2sim.org/)	• CPU/GPU simulator • Supports: x86, MIPS-32, ARM, AMD Ever-

[10] *PTLsim has been extensively used for uarch research (performance, reliability) and is not supported anymore; Marssx86 is its successor with many additional features.*

	green GPU, AMD Southern Islands GPU, NVIDIA Fermi GPU[11] (Full support for x86, AMD Evergreen and AMD Southern Islands GPUs; the rest are in progress and not fully validated)
SimNow (http://developer.amd.com/tools-and-sdks/cpu-development/simnow-simulator/)	• Functional simulator, not timing • Includes all x86 latest enhancements (no other includes all)
ESESC (http://masc.cse.ucsc.edu/esesc/)	• Enhancement of SESC; just released • ARM ISA • Model heterogeneous multicores • Detailed performance, power, and thermal models
Simics (http://www.windriver.com/products/simics/)	• Now a commercial product; university licenses available

Table 10: RTL models (unless otherwise stated, these components are available at www.opencores.org)[12]

Category	RTL model	Size
Synthesizable CPU Cores	OpenSPARC T1 (http://www.oracle.com/technetwork/systems/opensparc/opensparc-t1-page-1444609.html)	N/A
	CPU86 8088 FPGA IP Core (http://www.ht-lab.com/freecores/cpu8086/cpu86.html)	29,000 transistors
	8080 Compatible CPU	2082 LUTs
	AltOr32 - Alternative Lightweight OpenRisc CPU	N/A
	Amber ARM-compatible core	8732 LUTs
	AVR HP, Hyper Pipelined AVR Core	N/A
	HC11 Compatible - Gator uProcessor	N/A
	Leightweight 8051 compatible CPU	N/A
	Lightweight 8080 compatible core	204 LUTs
	Next 80186 processor	~6000 LUTs
	NextZ80	~2800 LUTs
	openMSP430	1650 LUTs
	OpenRISC 1000	N/A
	OpenRisc 1200 HP, Hyper Pipelined OR1200 Core	N/A
	Plasma – most MIPS I(TM) opcodes	8084 LUTs
	S1 Core	N/A
	Small x86 subset core	N/A
	Storm Core (ARM7 compatible)	N/A
	VHDL core of IC6821	N/A

[11] Developers claim that other GPU simulators are not cycle accurate because they simulate NVDIA's ptx ISA which is recompiled to native chip machine code. They also claim Multi2Sim is true cycle accurate for AMD GPUs. However, GPGPU-sim simulates also the native instruction set (so it is true cycle accurate) for the older architectures (before Fermi).

[12] Hardware components whose RTL models are not public available have not been listed

	Y80e - Z80/Z180 compatible processor extended by eZ80 instructions	2557 LUTs
	Zet - The x86 (IA-32) open implementation	N/A
	LEON based on SPARC V8 (www.gaisler.com)	
Arithmetic Cores	Cellular Automata PRNG	N/A
	Elliptic Curve Group	13,789 LUTs
	FPU	N/A
	FPU Double VHDL	N/A
	LZRW1 Compressor Core	~500 slices
Communica-tion Controllers	10_100_1000 Mbps tri-mode Ethernet MAC	1839 LUTs
	8b10b Encoder/Decoder	N/A
	a VHDL 16550 UART core	N/A
	VHDL CAN Protocol Controller	N/A
	DMX512 transceiver	N/A
	Ethernet 10GE Low Latency MAC	N/A
	Ethernet MAC 10/100 Mbps	28K gates or 2400 flip-flops
	I2C controller core	230 LUTs
	IEEE 802.15.4 Core (physical layer)	N/A
	sd card controller	N/A
	Serial ATA Host Bus Adapter Core for Virtex 6	N/A
	SPI Flash controller	N/A
	SPI Master/Slave Interface	41 slices
	UART to Bus	195 slices
	UART16750	378 slices
	USB 2.0 Function Core	N/A
	Wishbone SD Card Controller	N/A
Crypto-cores	AES	6503 LUTs
	Avalon AES ECB-Core (128, 192, 256 Bit)	N/A
	DESL Core	N/A
	DESLX Core	N/A
	DESX Core	N/A
	SHA3 (KECCAK)	9895 LUTs
DSP Cores	filtro_FIR	N/A
	Hilbert Transformer	N/A
	Low-Pass IIR Filter	N/A
	PID controller	N/A
	Pipelined FFT/IFFT 256 points processor	N/A
ECC Core	Reed Solomon Decoder (204,188)	3397 slices
	Viterbi Decoder (AXI4-Stream compliant)	N/A
	CF LDPC Decoder	N/A
	Turbo Decoder	N/A
	CF Interleaver	N/A
Memory	8/16/32 bit SDRAM Controller	N/A

Cores	BRSFmnCE	30 slices
	DDR3 SDRAM controller	N/A
	DPSFmnCE	32 slices
	High Performance Dynamic Memory Controller	N/A
NoC Interconnects	AHB to Wishbone Bridge	N/A
	Async-SDM-NoC	N/A
	System-on-Chip Wire (SoCWire)	N/A
System Controllers	Memory Controller IP Core	N/A
	PCIe SG DMA controller	N/A
	pci_mini	279 slices
Testing / Verification	PITbUtils	N/A
	PRBS Signal Generator and Checker	N/A
Video Controller	FastMemoryLink VGA framebuffer controller	N/A
	Image warping/Texture mapping core	2150 slices
	JPEG Encoder	6135 ALUTs
	VGA/LCD Controller	N/A

As mentioned in Section 2.1, as a starting point, WP3 studies the reliability of hardware components of CISC microprocessors (commonly used in HPC systems), RISC microprocessors (commonly used in EC systems) and accelerators (commonly used in HPC systems) architectures.

The experimental vehicles will be mainly three state-of-the-art tools, Marssx86, Gem5 and GPGPU-sim simulators, respectively. Many studies have already been based on these tools. For instance, Marssx86 is considered to accurately model the memory hierarchy system of an out-of-order x86_64 microprocessor [51]. In subsequent deliverables of WP3 the tools (simulators and models) that will be used for the characterization of all types of hardware components will be reported. The majority of tools and models will be derived from the lists of this deliverable.

7. Conclusions

Work package 3 (WP3) aims at analyzing the reliability of hardware components, such as microprocessor cores, accelerators, memory systems, peripherals and interconnection logic (and their major sub-components), throughout the whole computing spectrum. WP3 will consider several design parameters, such as component complexity, area, power budget, resource utilization, error masking probability, timing constraints, to estimate the potential impact of hardware faults (transient, intermittent and permanent) to the overall system reliability. The output of WP3 will be a library of reliability-characterized components that will be employed alongside the results of WP4 into the overall "CLERECO statistical reliability estimation tool" of WP5 as well as the demonstration activity of CLERECO project during the validation and proof-of-concept work package (WP6).

This report summarizes all major hardware component classes (processors, peripherals, accelerators, memories and interconnects) along with their most representative sub-components within the computing continuum spectrum (ranging from High Performance Computing to Embedded Systems). Furthermore, this deliverable lists the candidate hardware components, and tools which are either publicly available or can be acquired for the course of the project, and can be used for the validation of the system methodology and the demonstration of the project.

8. Acronyms

8.1. List of acronyms

Table 11: List of Acronyms

Acronym	Full text
ALU	Arithmetic & Logic Unit
AMBA	Advanced Microcontroller Bus Architecture
APIC	Advanced Programmable Interrupt Controller
CISC	Complex Instruction Set Computer
CPU	Central Processing Unit
CUDA	Compute Unified Device Architecture
DMA	Direct Memory Access
DSP	Digital Signal Processing
EC	Embedded Computing
ECC	Error-correcting Code
EMU	Extended Math Unit
FDD	First-level Dynamically Dead (instructions)
FPU	Floating-Point Unit
GPU	Graphic Processing Unit
HPC	High Performance Computing
I2C	Inter-Integrated Circuit
NoC	Network-on-Chip
PCM	Phase-change Memory
PCIe	Peripheral Component Interconnect Express
PIC	Programmable Interrupt Controller
RISC	Reduced Instruction Set Computer
ROM	Read-Only Memory
SIMD	Single Instruction Multiple Data
SIMT	Single Instruction Multiple Threads

SPI	Serial Peripheral Interface
SRAM	Static Random Access Memory
TDD	Transitively Dynamically Dead
TLB	Translation Look-aside Buffer
UART	Universal Asynchronous Receiver/Transmitter
USB	Universal Serial Bus
VGA	Video Graphic Array
VPU	Vector Processing Unit
Wi-Fi	Wireless Fidelity
WNIC	Wireless Network Interface Controller

9. References

[1] S.Nassif, N.Mehta, Y.Cao,"A Resilience Roadmap", International Conference on Design Automation and Test in Europe (DATE), Dresden, Germany, March 8-12, 2010.

[2] M.Abramovici, M.A.Breuer, A.D.Friedman, Digital Systems Testing and Testable Design, Wiley-IEEE Press, 1994.

[3] M.Bushnell, V.Agrawal, Essentials of Electronic Testing for Digital, Memory and Mixed-Signal VLSI Circuits, Kluwer academic publishers, 2002.

[4] S.Mukherjee, Architecture Design for Soft Errors, Morgan Kaufmann, 2008.

[5] J.Gracia,L.Saiz, J.C.Baraza,D.Gil,P.Gil, "Analysis of the influence of intermittent faults in a microcontroller",Design and Diagnostics of Electronic Circuits and Systems (DDECS-11), Bratislava,Czech Republic,April 16-18, 2008.

[6] C.Constantinescu,"Impact of Intermittent Faults on Nanocomputing Devices", International Conference on Dependable Systems and Networks (DSN-37), Edinburgh, UK, June 25-28, 2007.

[7] F.Wang, Y. Xie, "An accurate and efficient model of electrical masking effect for soft errors in combinational logic", Silicon Errors in Logic – System Effects (SELSE-2), 2006.

[8] Y.Lu, H.Zhou, "Retiming for Soft Error Minimization Under Error-Latching Window Constraints", Design Automation & Test in Europe (DATE), 2013.

[9] F.Wang, V.D.Agrawal, "Soft Error Rates with Inertial and Logical Masking", IEEE International Conference on VLSI Design (VLSID), New-Delhi, India, January 5-9, 2009.

[10] G.Reis, J.Chang, N.Vachharajani, R.Rangan, D.August, S.S.Mukherjee, "Design and Evaluation of Hybrid Fault-Detection Systems", ACM/IEEE International Symposium on Computer Architecture (ISCA-32), Madison, Winsonsin, USA, June 4-8, 2005.

[11] C.Weaver, J.Emer, S.S.Mukherjee, S.K.Reinhardt, "Techniques to Reduce the Soft Error Rate of High-Performane Microprocessor", ACM/IEEE International Symposium on Computer Architecture (ISCA-31), Munich, Germany, June 19-23, 2004.

[12] N.J.Ge□rge, C.R.Elks, B.W.Johnson, J.Lach, "Transient Fault Models and AVF Estimation Revisited", International Conference on Dependable Systems and Networks (DSN-40), Chicago, Illinois, USA, June 28 – July 1, 2010.

[13] S.S. Mukherjee, C.Weaver, J.Emer, S.K Reinhardt, T.Austin, "A Systematic Methodology to Compute the Architectural Vulnerability Factors for a High-Performance Microprocessor", International Symposium on Microarchitecture (MICRO-36), San Diego, CA, USA, December 3-5, 2003.
[14] V.Sridharan, D.R.Kaeli, "Using Hardware Vulnerability Factors to Enhance AVF Analysis", ACM/IEEE International Symposium on Computer Architecture (ISCA-37), Saint-Malo, France, June 21-23, 2010.
[15] V.Sridharan, D.R.Kaeli, "Eliminating microarchitectural dependency from architectural vulnerability", IEEE International Symposium on High Performance Computer Architecture (HPCA-15), 2009.
[16] V.Sridharan, D.R.Kaeli, "The Effect of Input Data on Program Vulnerability", Workshop on Silicon Errors in Logic – System Effects (SELSE-5), Stanford University, March 24-25, 2009.
[17] C.-K. Luk, R.Cohn, R.Muth, H.Patil, A.Klauser, G.Lowney, S.Wallace, V.J.Reddi, K.Hazelwood, "Pin: Building Customized Program Analysis Tools with Dynamic Instrumentation", Conference on Programming Language Design and Implementation (PLDI), Chicago, Illinois, USA, June 11-15, 2005.
[18] D.Hardy, I.Sideris, N.Ladas, Y.Sazeides, "The performance vulnerability of architectural and non-architectural arrays to permanent faults", International Symposium on Microarchitecture (MICRO-45), Vancouver, BC, Canada, December 1-5, 2012.
[19] N.Foutris, D.Gizopoulos, A.Chatzidimitriou, J.Kalamatianos, V.Sridharan, "Performance Assessment of Data Prefetchers in High Error Rate Technologies", 10th Workshop on Silicon Errors in Logic – System Effects (SELSE-10), Stanford University, April 2014.
[20] N.Foutris, D.Gizopoulos, J.Kalamatianos, V.Sridharan, "Assessing the Impact of Hard Faults in Performance Components of Modern Microprocessors", IEEE International Conference on Computer Design (ICCD 2013), Asheville, NC, USA, October 2013.
[21] K.R.Walcott, G.Humphreys, S.Gurumurthi, "Dynamic Prediction of Architectural Vulnerability from Microarchitectural State", International Symposium on Computer Architecture (ISCA-34), San Diego, CA, USA, June 9-13, 2007.
[22] A.Biswas, P.Racunas, R.Cheveresan, J.Emer, S.S. Mukherjee, R.Rangan, "Computing Architectural Vulnerability Factors for Address-Based Structures", International Symposium on Computer Architecture (ISCA-32), Madison, Wisconsin, USA, June 4-8, 2005.
[23] W.Zhang, X.Fu, T.Li, J.Fortes, "An Analysis of Microarchitecture Vulnerability to Soft Errors on Simultaneous Multithreaded Architectures", International Symposium on Performance Analysis of Systems and Software (ISPASS), San Jose, CA, USA, April 25-27, 2007.
[24] N.J.Wang, A.Mahesri, S.J.Patel, "Examining ACE Analysis Reliability Estimates Using Fault-Injection", International Symposium on Computer Architecture (ISCA-34), San Diego, CA, USA, June 9-13, 2007.
[25] N.Soundararajan, A.Parashar, A.Sivasubramaniam, "Mechanisms for Bounding Vulnerabilities of Processor Structures", International Symposium on Computer Architecture (ISCA-34), San Diego, CA, USA, June 9-13, 2007.
[26] P.Shivakumar, M.Kistler, S.W.Keckler, D.Burger, L.Alvisi, "Modeling the effect of technology trends on the soft error rate of combinational logic", International Conference on Dependable Systems and Networks (DSN), Washington, DC, USA, 23-26 June, 2002.
[27] S.Mirkhani, J.A.Abraham, "Fast Evaluation of Test Vector Sets Using a Simulation-based Statistical Metric", VLSI Test Symposium (VTS-32), Napa, California, USA, 13-17 April, 2014.
[28] J.Suh, M.Manoochehri, M.Annavaram, M.Dubois, "Soft Error Benchmarking of L2 Caches with PARMA", International Conference on Measurement and Modeling Computer Systems (SIGMETRICS), San Jose, CA, USA, June 7-11, 2011.
[29] H.Jeon, M.Wilkening, V.Sridharan, S.Gurumurthi, G.H.Loh, "Architectural Vulnerability Modeling and Analysis of Integrated Graphics Processors", Workshop on Silicon Errors in Logic – System Effects (SELSE-9), Stanford University, March 26-27, 2013.
[30] L.Duan, B.Li, L.Peng, "Versatile Prediction and Fast Estimation of Architectural Vulnerability Factor from Processor Performance Metrics", International Symposium on High-

Performance Computer Architecture (HPCA-15), Raleigh, North Carolina, February 14-18, 2009.
[31] A.Biswas, N.Soundararajan, S.S.Mukherjee, S.Gurumurthi, "Quantized AVF: A Means of Capturing Vulnerability Variations over Small Windows of Time", Workshop on Silicon Errors in Logic – System Effects (SELSE-5), Stanford University, March 24-25, 2009.
[32] R.Balasubramanian, K.Sankaralingam, "Understanding the Impact of Gate-Level Physical Reliability Effects on Whole Program Execution", International Symposium on High-Performance Computer Architecture (HPCA-20), Orlando, Florida, February 15-19, 2014.
[33] G.Yalcin, O.S.Unsal, A.Cristal, M.Valero, "FIMSIM: A Fault Injection Infrastructure for Micro-architectural Simulators", International Conference on Computer Design (ICCD-29), Amhest, MA, USA, October 9-12, 2011.
[34] M.-L. Li, P.Ramachandran, U.R.Karpuzcu, S. K. S. Hari, S.V.Adve, "Accurate Microarchitecture-Level Fault Modeling for Studying Hardware Faults", International Symposium on High-Performance Computer Architecture (HPCA-15), Raleigh, North Carolina, February 14-18, 2009.
[35] R.Shah, M.Choi, B.Jang, "Workload-Dependent Relative Fault Sensitivity and Error Contribution Factor of GPU Onchip Memory Structures", International Conference on Embedded Computer Systems: Architectures, Modeling, and Simulation (SAMOS-13), Samos, Greece, July 15-18, 2013.
[36] X.Li, S.V.Adve, P.Bose, J.A.Rivers, "Architecture-Level Soft Error Analysis: Examining the Limits of Common Assumptions", International Conference on Dependable Systems and Networks (DSN-37), Edinburgh, UK, June 25-28, 2007.
[37] M.Rebaudengo, M.S.Reorda, M.Violante, "An Accurate Analysis of the Effects of Soft Errors in the Instruction and Data Caches of a Pipelined Microprocessor", International Conference on Design Automation and Test in Europe (DATE), Munich, Germany, March 3-7, 2003.
[38] J.Suh, M.Annavaram, M.Dubois, "MACAU: A Markov Model for Reliability Evaluations of Caches Under Single-bit and Multi-bit Upsets", International Symposium on High-Performance Computer Architecture (HPCA-18), New Orleans, Louisiana, February 25-29, 2012.
[39] J.Carretero, E.Herrero, M.Monchiero, T.Ramirez, X.Vera, "Capturing Vulnerability Variations for Register Files", International Conference on Design Automation and Test in Europe (DATE), Grenoble, France, March 18-22, 2013.
[40] X.Li, S.V.Adve, P.Bose, J.A.Rivers, "SoftArch An Architecture-Level Tool for Modeling and Analyzing Soft-Errors", International Conference on Dependable Systems and Networks (DSN), Yokohama, Japan, June 28-July 1, 2005.
[41] L.Huang, Q.Xu, "AgeSim: A Simulation Framework for Evaluating the Lifetime Reliability of Processor-Based SoCs", International Conference on Design Automation and Test in Europe (DATE), Dresden, Germany, March 8-12, 2010.
[42] A.Danowitz, K.Kelley, J.Mao, J.P.Stevenson, M.Horowitz, "CPU DB: recording microprocessor history", Communications of the ACM, vol. 55, no. 4, April 2012. (DB is online available on: http://cpudb.stanford.edu/).
[43] S. Kim and K. Somani, "Soft Error Sensitivity Characterization for Microprocessor Dependability Enhancement Strategy," in Proceedings of International Conference on Dependable Systems and Networks (DSN), Washington, DC, USA, 23-26 June 2002.
[44] N.J. Wang, J. Quek, T.M. Rafacz, and S.J. Patel, "Characterizing the Effects of Transient Faults on a High-Performance Processor Pipeline," in Proceedings of International Conference on Dependable Systems and Networks(DSN), Florence, Italy, June 28 - July 1, 2004
[45] G. Saggese, N.J. Wang, Z. Kalbarczyk, S.J. Patel, and R. Iyer, "An Experimental Study of Soft Errors in Microprocessors," IEEE Micro, vol. 25, no. 6, pp. 30-39, Nov-Dec 2005.
[46] N.George, C. Elks, B. Johnson, J. Lach, "Transient Fault Models and AVF estimation revisited", in Proceedings of International Conference on Dependable Systems and Networks (DSN), Chicago, Illinois, USA, 28 June - July 1, 2010.

[47] M.Maniatakos, N.Karimi, C.Tirumurti, A.Jas, Y.Makris,"Instruction-Level Impact Analysis of Low-Level Faults in a Modern Microprocessor Controller".
[48] P.Ramachandrant, P. Kudvatt, J. Kellingtont, J. Schumannt, P. Sandat, "Statistical Fault Injection", International Conference on Dependable Systems & Networks(DSN): Anchorage, Alaska, June 24-27, 2008.
[49] B.Wibovo, A.Agrawal, J.Tuck, "Toward a Cross-Layer Approach for Dynamic Vulnerability Estimation", Silicon Errors in Logic – System Effects (SELSE-10), Stanford University, April 1-2, 2014.
[50] A.A.Nair, S.Eyerman, L.Eeckhout, L.K.John, "A first-Order Mechanistic Model for Architectural Vulnerability Factor", International Symposium on Computer Architecture (ISCA-39), Portland, OR, USA, June 19-23, 2012.
[51] J.Stevens, P.Tschirhart, M-T.Chang, I.Bhati, P.Enns, J.Greensky, Z.Cristi, S-L.Lu, B.Jacob, "An integrated simulation infrastructure for the entire memory hierarchy: cache, dram, non-volatile memory, and disk", Intel Technology Journal, vol.17, no 1, 2013.
[52] J.J.Cook, C.Zilles, "A Characterization of Instruction-level Error Derating and its Implications for Error Detection", International Conference on Dependable Systems & Networks (DSN): Anchorage, Alaska, June 24-27, 2008.
[53] D.Hardy, M.Kleanthous, I.Sidoros, A.G.Saidi, E.Ozer, Y.Sazeides, "An Analytical Framework for Estimating TCO and Exploring Data Center Design Space", International Symposium on Performance Analysis of Systems and Software (ISPASS), Austin, TX, USA, April 21-23, 2013.
[54] F.A.Bower, D.Hower, M.Yilmaz, D.J.Sorin, S.Ozev, "Applying Architectural Vulnerability Analysis to Hard Faults in the Microprocessor", International Conference on Measurement and Modeling Computer Systems (SIGMETRICS), Saint Malo, France, June 26-30, 2006.
[55] Y.Luo, S.Govindan, B.Sharma, M.Santaniello, J.Meza, A.Kansal, J.Liu, B.Khessib, K.Vaid, O.Mutlu, "Characterizing Application Memory Error Vulnerability to Optimize Datacenter Cost via Heterogeneous-Reliability Memory", International Conference on Dependable Systems & Networks (DSN): Atlanta, Georgia, USA, June 23-26, 2014.
[56] S.Mirkhani, S.Mitra, C-Y.Cher, J.A.Abraham, "Effective Statistical Estimation of Soft Error Vulnerability for Complex Designs", 10th Workshop on Silicon Errors in Logic – System Effects (SELSE-10), Stanford University, April 2014.
[57] S.K.S.Hari, R.Venkatagiri, S.V.Adve, H.Naeimi, "GangES Gang Error Simulation for Hardware Resiliency Evaluation", ACM/IEEE International Symposium on Computer Architecture (ISCA-41), Minneapolis, MN, USA, June 14-18, 2014.
[58] K.Parasyris, G.Tziantzoulis, C.Antonopoulos, N.Bellas, "GemFI A Fault Injection Tool for Studying the Behavior of Applications on Unreliable Substrates", International Conference on Dependable Systems & Networks (DSN): Atlanta, Georgia, USA, June 23-26, 2014.
[59] G.Gupta, S.Sridharan, G.S.Sohi, "Globally Precise-restartable Execution of Parallel Programs", Conference on Programming Language Design and Implementation (PLDI-35), Edinburgh, UK, June 9-11 2014.
[60] B.Fang, K.Pattabiraman, M.Ripeanu, S.Gurumurthi, "GPU-Qin A Methodology for Evaluating the Error Resilience of GPGPU Applications", International Symposium on Performance Analysis of Systems and Software (ISPASS), Monterey, CA, USA, March 23-25, 2014.
[61] S.Pan, Y.Hu, X.Li, "IVF: Characterizing the Vulnerability of Microprocessor Structures to Intermittent Faults", IEEE Transactions on Very Large Scale Integration (VLSI) Systems, vol. 20, no. 5, May 2012.
[62] A.Thomas, K.Pattabiraman, "LLFI: An Intermediate Code Level Fault Injector For Soft Computing Applications", Workshop on Silicon Errors in Logic – System Effects (SELSE-9), Stanford University, March 26-27, 2013.
[63] S.K.S.Hari, M-L.Pi, P.Ramachandran, B.Choi, S.V.Adve, "mSWAT: Low-Cost Hardware Fault Detection and Diagnosis for Multicore Systems", International Symposium on Microarchitecture (MICRO-42), New York, New York, USA, December 12-16, 2009.

[64] J.Wei, A. Thomas, G.Li, K.Pattabiraman, "Quantifying the Accuracy of High-Level Fault Injection Techniques for Hardware Faults", International Conference on Dependable Systems & Networks (DSN): Atlanta, Georgia, USA, June 23-26, 2014.
[65] H.Cho, S.Mirkhani, C-Y.Cher, J.A.Abraham, S.Mitra, "Quantitative Evaluation of Soft Error Injection Techniques for Robust System Design", Design & Automation Conference (DAC-50), Austin, TX, USA, May 29-June 7, 2013.
[66] S.Sudhakrishnan, R.Dicochea, J.Renau, "Releasing Efficient Beta Cores to Market Early", International Symposium on Computer Architecture (ISCA-38), San Jose, CA, USA, June 4-8, 2011.
[67] S.Nomura, M.D.Sinclair, C-H.Ho, V. Govindaraju, M.Kruijf, K. Sankaralingam, "Sampling + DMR Practical and Low-overhead Permanent Fault Detection", International Symposium on Computer Architecture (ISCA-38), San Jose, CA, USA, June 4-8, 2011.
[68] X.Fu, T.Li, J.A.B.Fortes, "Sim-SODA: A Unified Framework for Architectural Level Software Reliability Analysis", Workshop on Modeling, Benchmarking and Simulation (MoBS, Held in conjunction with International Symposium on Computer Architecture), June 18, 2006.
[69] J.Tan, N. Goswami, T.Li, X.Fu, "Analyzing Soft-Error Vulnerability on GPGPU Microarchitecture", IEEE International Symposium on Workload Characterization (IISWC), Austin, TX, USA, November 6-8, 2011.
[70] A.Gutierrez, J.Pusdesris, R.G. Dreslinski, T.Mudge, C. Sudanthi, C.D.Emmons, "Sources of Error in Full-System Simulation", International Symposium on Performance Analysis of Systems and Software (ISPASS), Monterey, CA, USA, March 23-25, 2014.