Project Number: FP7-611404

D5.3 - Design optimization heuristics (preliminary)

Authors[1]

A. Savino (POLITO), S. Di Carlo (POLITO), A. Vallero (POLITO), G. Politano (POLITO)

Version 1.0 – 19/03/2016

Lead contractor: Politecnico di Torino
Contact person: Alessandro Savino Control and Computer Engineering Dep. Politecnico di Torino, C.so Duca degli Abruzzi, 24 I-10129 Torino TO Italy E-mail: alessandro.savino@polito.it
Involved Partners[2]**:** POLITO
Work package: WP5
Affected tasks: T5.4

Nature of deliverable[3]	R	P	D	O
Dissemination level[4]	PU	PP	RE	CO

[1] *Authors listed here only identify persons that contributed to the writing of the document.*

[2] *List of partners that contributed to the activities described in this deliverable.*

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

[4] ***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

Scope of the document

This document is the main outcome of task T5.4 "**Design optimization heuristics**", elaborated in the Description of Work (DoW) of the CLERECO project under Work Package 5 (WP5).

Figure 1 depicts graphically the goal of this deliverable, its main results, the inputs it uses and the outputs it provides (including which WPs will use its outputs).

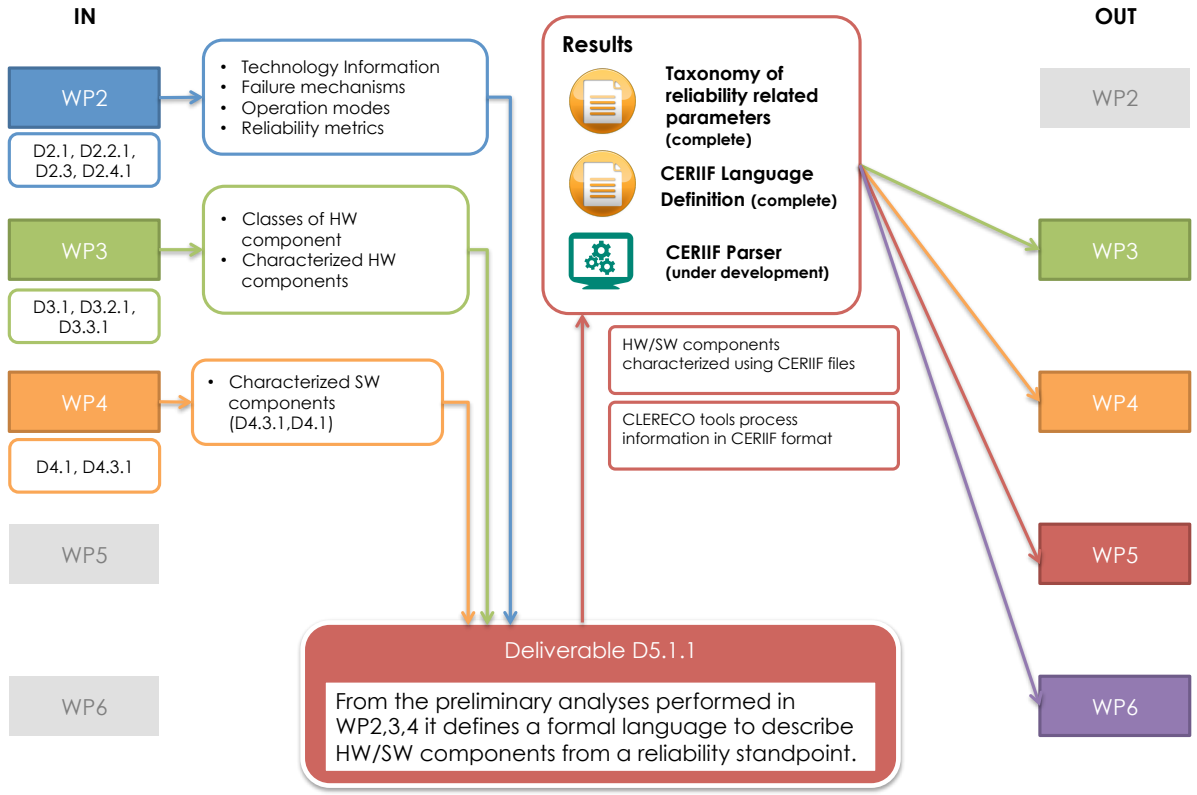

Figure 1: The Inputs and the Outputs of this Deliverable

The main goal and outcome of D5.4 is to define and to develop new automatic optimization strategies and heuristics enabling to optimize the design of a complex system maximizing its reliability but, at the same time, trading-off other design dimensions such as area, power and performance.

The optimization environment described in this deliverable resorts to information provided by all the characterization tools developed in the framework of the other CLERECO technical WPs, i.e., WP2, WP3, WP4 and WP5. In particular, the optimization process is strongly based on the system description and on the reliability estimation model described in deliverables D5.1.2 and D5.2.2. By resorting to the CLERECO Bayesian reliability estimation model developed in the project the algorithms and tools presented in this deliverable enable to explore different design alternatives, and, for each of them, to quickly evaluate the reliability of the designed system, thus looking for the best combination of components that enable to optimize the design.

In this context design alternatives correspond to different combinations of hardware and software components whose reliability has been characterized resorting to the available CLERECO tool chain.

This is a preliminary version of this document that focuses on the description of the optimization algorithm and its implementation into an EDA tool integrated in the CLERECO tool suite. In its final version extended experimental results will be included to show the optimization engine at work on a set of real cases.

The document is organized in the following sections:

- The **Introduction** summarizes motivations of this work and briefly reviews the literature focusing on reliability optimization of complex systems.
- The **Reliability Optimization Strategies** introduces the heuristics developed in the project to implement a system reliability optimization engine. In this section we first focus on the theoretical aspects of the defined algorithm, and then we provide technical details regarding its actual implementation.
- Eventually, the **Conclusions** summarize the main contributions of the document and outline the work that will be done in the remaining of the project.

1. Introduction

System optimization, trading-off several design dimensions is a crucial problem for system designers in several application domains ranging from very complex HPC systems to embedded systems and consumer electronics.

Optimizing the reliability of a digital system is a complex process that requires selecting the best combination of technologies, hardware components, software modules, and their related fault-tolerance mechanisms considering a cross-layer approach where protection against several classes of faults can be achieved at different layers of the system stack. This is usually a multi-objective optimization problem since the number of parameters that have to be considered and the number of design options is huge. Moreover, reliability must be traded off with other dimensions such as area, timing, power, etc. [7]. Nowadays, the number of parameters to consider is growing very close to a level that cannot be handled any more by system designers without the introduction of proper automatic optimization tools.

Currently, system optimization is demanded to decision makers and expert system designers. The common practice to achieve a given reliability constraint is to build a system with large margins in the technology and to add several levels of redundancy, either at the hardware layer, or at the software layer. Regardless the different implementations, at a high-level redundancy is achieved by introducing spare copies of the same components acting in parallel to perform the same operation, or by performing computations several times in consecutive time slots. In both cases, dedicated decision and voting algorithms able to take proper actions whenever discrepancies are detected in the multiple instances of the implemented functionalities are required to exploit the introduced redundancy. Unfortunately, with the increasing complexity of modern systems, this approach is becoming prohibitively expensive in terms of area, power and performance, especially for embedded computing systems [1][2][3][4].

One of the main limitations of the way systems are nowadays optimized, is that decisions are not driven by accurate system-level reliability estimates that take into account the peculiar interaction of the hardware architecture with the specific software workload. A given protection mechanism is usually selected based on the knowledge of its general efficiency or based on regulations introduced by safety standards defined for specific application domains. Given the lack of analytical results, protection strategies tend to be very conservative, in the sense that designers resort to them ever considering worst-case scenarios. This has the general drawback

of overdesigning the final system. Finally, the way different protection mechanisms interact together is in general not well understood, thus limiting the optimization to very simple design decisions.

In recent years, system-level reliability estimation methodologies avoiding complex and computational intensive RTL fault injection (FI) campaigns are emerging and the CLERECO project is taking a significant step forward in this direction. Most of these approaches rely on high-level models of the system; some of them focus on a single layer of the system stack (e.g., the hardware layer [5]), whereas others consider all layers starting from the technology up to the application software [6], [9]. In all cases, the accuracy of the evaluation depends on the accuracy of the model, but they all share the ability of providing reliability estimations faster than RTL fault injection.

The availability of fast reliability estimation models opens new chances for the realization of automatic system optimization methodologies. However, performing exhaustive design exploration to optimize a given system, especially in an early phase of the design cycle when the number of design choices and the number of degrees of freedom is high, is a complex problem that can easily turn into an exponential search problem. To deal with this complexity, proper heuristics able to find near-optimal solutions must be designed.

General-purpose optimization approaches based on stochastic procedures such as the ones proposed in [12][13][14], or meta-heuristics methods such as *simulated annealing* [16] and *genetic algorithms* [17] even if demonstrated effective in solving general optimization problems are difficult to adapt to the reliability design optimization problem in which peculiar constraints in the architecture of the system must be carefully considered when performing the optimization. Moreover, these approaches are in general slow and do not properly scale with the complexity required to analyze a real system.

In the reliability domain, very few works proposing automatic system optimization solutions have been published. Some of them are summarized in [7] and [8], where authors review a set of optimization techniques focusing on finding an optimal solution to the *Redundancy allocation problem* (RAP). Most of them are based on genetic algorithms, but they all start from the assumption that data redundancy is the selected fault-tolerance mechanism of the system. The intrinsic resiliency of the system to selected hardware faults is not analyzed and therefore not considered during the optimization process that only focuses on optimizing the amount of redundancy with respect to area constraints. Wattanapongsakorn and Levitan propose to perform system optimization by considering the interaction of both hardware and software components [9]. The technique resorts to a Simulated Annealing (SA) optimization approach [10]. The basic concept is evaluating random configurations of all available components, selecting at each iteration the best among configuration until a stop criterion is satisfied. The cost function is commonly the sum of the costs of all components, and the stopping criterion acts when there is no improvement in the best solution within a pre-defined number of iterations. Although the technique seems promising, it lacks several important features. First of all, the reliability model of the system is very limited, mainly based on a single probability of failure for each component. This does not completely takes into account the effect of a single component replacement with respect to the others. Second, the optimization does not guarantee to effectively explore the design space due to the randomization of the selection. The proposed cost function is very simple but the methodology itself seems open to more complex functions including multi-objective functions required to trade-off reliability with other design constraints. Finally, Shafique at al. [19] propose a reliability optimization framework where the sole software layer is modified during the optimization introducing step-by-step protection mechanisms to certain instructions until a desired level of protection is achieved or, at best, all unprotected instructions are protected by some mechanism. In this case, the hardware is only considered as a source of errors that are propagated to the software, thus no full system optimization is really proposed.

None of the considered publications systematically takes into account all design options that a designer can exploit when dealing with a complex electronic system project.

To overcome limitations of current approaches, in the next section, we focus on one specific general optimization technique that has proved to be very effective in problems that are similar to the CLERECO design optimization task. After describing its main characteristic we describe how it has been extended and improved in order to build a system optimization engine based on the CLERCO system reliability estimation model.

2. Design optimization strategies

2.1. Extremal Optimization

The Extremal Optimization (EO) theory [11] is a general-purpose optimization theory inspired by the self-organizing processes that can be observed in nature. Self-organized criticality (SOC) is a statistical physics concept to describe a class of dynamical systems that have a critical point as an attractor. Specifically, these are non-equilibrium systems that evolve through avalanches of changes and dissipations that reach up to the highest scales of the system. SOC is said to govern the dynamics behind some natural systems that have these burst-like phenomena including landscape formation, earthquakes, evolution, and the granular dynamics of rice and sand piles.

One of the most important characteristic of EO is its ability to deal with complex optimization problems (including NP-complete problems), where near-optimum solutions are widely dispersed and separated by barriers in the search space causing local search algorithms to get stuck or severely hampered [15][18][20][21][22]. This is a typical case in the optimization of a system where different near-optimal solutions can be located quite far in the solution space due to macro-changes in the hardware or software architecture.

To give a formal formulation of the EO algorithm let us introduce a set of basic notations and definitions.

- The solution space, i.e., the space of all possible admissible solutions to the optimization problem, is denoted with Ω.
- $S \in \Omega$ is a solution within the solution space. It can be defined as a tuple $S = (x_1, \cdots, x_n)$, where x_i is one out of n variables that must be optimized to find the optimal solution.
- $N(S) \subset \Omega$ is the *neighborhood* of the solution S. It includes all solutions that can be reached from S by making a single feasible modification of S. In terms of variables a solution $S' \in N(S) \subseteq \Omega$ is a solution differing from S for the value of a single variable.
- $C(S)$ is the cost function. It defines the metric to evaluate and compare different solutions in order to select the best one. If the problem is a minimization problem, it is commonly defined to correlate better solutions to lower function values. The cost function can be defined accounting for the contribution of the n variables composing the solution by defining a set of weights. Each weight λ_i models the contribution of variable x_i to $C(S)$. λ_i corresponds to the fitness value and, typically, depends the state of x_i in relation to other variables to which x_i is connected.

Figure 2 shows a very high-level pseudo-code describing how the EO algorithm performs its optimization process as described in [15].

1. Initialize a configuration S at will and set $S^{best} = S$. S^{best} will be the best solution so far.
2. For the current configuration S,
   a. Evaluate λ_i for each variable x_i;
   b. Find j with $\lambda_j \geq \lambda_i$ for all i; that is, x_j has the worst fitness;
   c. Choose a random $S' \in N(S)$ such as x_j must change;
   d. If $C(S') < C(S^{best})$, store $S^{best} = S'$;
   e. Accept S' as new S unconditionally.
3. Repeat at step 2 as long as desired.
4. Return S^{best} and $C(S^{best})$.

Figure 2 Extreme optimization algorithm

One of the most significant differences between EO and other optimization meta-heuristics is the need of defining a local cost contribution λ_i for each variable, instead of defining a single global cost function. This additional level of complexity is however extremely important during the optimization process. The local cost contributions are used to choose local movements in the design space while the global cost function is used to evaluate the fitness of the best-identified solution.

The EO strategy is particularly suited to work with the elements composing the CLERECO system reliability model as explained in the remaining of this section.

Each solution S represents a possible design alternative for the target system. Designs alternatives are defined as different implementations of the same system providing the same functionalities by exploiting different variants of technologies, hardware and software components, each one in turn characterized by different reliability features.

Since the main target is the optimization of the overall system reliability, each system and therefore each solution of the search space is modeled in the form of a Bayesian Network (BN) according to the CLERECO Bayesian Reliability model described in deliverable "**D5.2.2 - System Reliability Estimation Models**".

Each node of the Bayesian model and its related Conditional Probability Table (CPT) represents a variable x_i of the considered solution that may impact the reliability of the full system. For each node of the network, i.e., for each component of the system, a set of valid design alternatives is defined. These design alternatives are the ones that will be evaluated during the optimization process. All components that do not have design alternatives due to specific design constraints are not considered as variables in the solution and are not taken into account during the optimization process.

The solution space Ω is then defined upon all possible BNs that model the same kind of system (i.e., implementing the same final application) but employing different components and technologies as characterized in **WP2**, **WP3** and **WP4** (for further information refer to deliverables **D2.1**, **D3.3.2,** and **D4.3.2**).

According to the previous definitions, the function $N(S)$ maps all possible systems that can be defined by changing only a single variable (i.e., a single node in the Bayesian model) among the n variables defining the system. This function defines local changes in the solution space in which a single component of the system is modified to obtain a new design alternative.

If only reliability is taken into account during the optimization process, the λ_i functions represents, at first glance, the reliability impact of a single replaceable node on the reliability of the full system. This impact can be estimated through the CLERECO Bayesian Reliability Model based on the computed belief of the contribution of a failure in a given node to the estimated

failure of the system. The same can be said about $C(S)$ that, when considering only the reliability of the system, gives an estimation of the global reliability of the system computed as reported in **D5.2.2**.

As an example, let us consider the Bayesian model of the system reported in Figure 3. The figure represents a simple system whose basic implementation includes the following components:

- A technology layer (all <label>_22n nodes).
- A hardware layer including a microprocessor split into some of its internal components (x86_64, RF, L1 and L2 nodes) and connected to an external main memory (RAM).
- A software layer comprising the LINUX operating system (represented by the LINUX_OS_FN<number> nodes) and a full application (F<number> and Core_C_FN<numberl> nodes). The application is further split in two parts identifying a set of core functions (the Core_C_FN<numberl> nodes).

The proposed example is simple and not realistic but it is useful to explain a major limitation of the EO and motivating the need for the modified EO strategy introduced in CLERECO.

According to the EO optimization algorithm, starting from the initial architecture of the system, new design alternatives are generated by changing at each step a single node of the network. While this is a feasible strategy when simple components are taken into account, major problems arise with complex components.

Let us consider the L1 cache available the system of **Figure 2**. If two versions of this component are available in the design space (e.g., a parity unprotected and a parity protected L1 cache) one version can be easily replaced with another version and the new design option can be evaluated to understand this change on the reliability of the full system. However, this is not the case when a complex node such as a microprocessor that has different design alternatives is replaced. Replacing the node modeling the microprocessor requires to recursively replace all nodes modeling the components composing the microprocessor until a new stable configuration is reached.

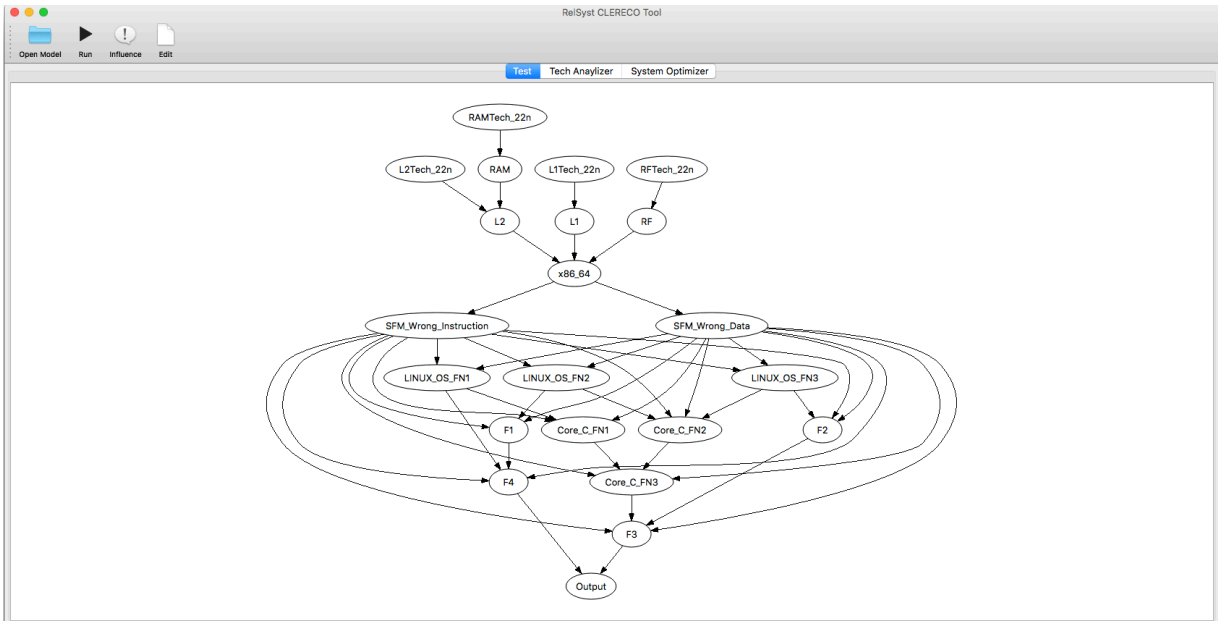

Figure 3: A BN modeled system view from the CLERECO tool.

In its basic formulation the EO algorithm is unable to deal with replacements based on clusters of variables. The optimization strategy always analyzes a single variable at a time. Given

this limitation, in the framework of the CLERECO project we have worked to define a new improved EO algorithm named Multi-Level Multi-Objective Extremal Design Optimization strategy that extends the classical EO strategy considering different levels of replacement including both single nodes or clusters of nodes. This extended strategy has been particularly optimized for its application to systems modeled using the CLERECO Bayesian model.

Finally, while our examples focus to reliability based cost functions, the proposed optimization approach can easily deal with multi-objective cost functions [23][24][25] that enable to drive the optimization not only based on the reliability level estimated for the system but also considering other design parameters such as area, power, performance, etc.

2.2. Multi-Level Multi-Objective EO

As introduced in the previous section, when applying the EO strategy to the optimization of the design of an electronic system the designer has to deal with a hierarchical set of design decisions each one influencing the available design space when moving to the lower hierarchical level.

In general terms, the designer needs to start from the selection of a set of macro-components defining the general architecture of the system. This includes selecting a given microprocessor architecture, the main memory architecture, the operating system and the functions the application software must provide. However, each macro-component is in general a complex element of system that can be hierarchically split into additional sub-components each one characterized by different available implementations. These subcomponents can be themselves complex components that can be further split into simpler components, and this process can be repeated until the desired level of detail for the system is reached.

All macro and micro modifications of the system might lead to changes both positive and negative on the reliability of the full system. For the sake of simplicity let us consider a system described with two levels of hierarchy differentiating macro-components from micro-components. The same approach can be then extended to any number of levels of hierarchy. With this hierarchical organization of the system, the optimization process can be split in two steps:

1. First, evaluate and select one "architectural" configuration of macro-components to maximize the reliability.
2. Second, for the actual configuration, evaluate possible micro-modifications to further improve reliability.

Anytime the second step reaches an end, thus no improvement in the reliability can be found, the designer should be able to go back to step one and try to start over with a different macro-architectural configuration. It is quite clear how this approach can be effective but also endless as soon as the set of macro- and micro- components grows. Moreover, since the different macro-components may have a different number of configuration/implementation alternatives, the number of design alternatives to explore is not constant and may be difficult to evaluate.

Based on these considerations we have defined an extended EO strategy named **Multi-Level Multi-Objective Extremal Optimization** (MLMOEO) that extends the general EO theory enabling to consider multiple levels of hierarchical optimizations defining different objective functions for each hierarchical level, each functions built considering multiple parameters and multiple objectives. The approach tries to combine the ability to improve the system towards large replacements of its macro-components and the possibility of further tuning each macro-component to reach a better optimization of the system design.

To define the MLMOEO let us introduce some additional definitions:

- The solution space $\Omega = \langle \omega_1, \cdots \omega_l \rangle$ is split in sub-solutions spaces (ω_k), each of them including only a portion of all feasible solutions.
- $S_k \in \omega_k$ is a solution within sub-space ω_k. Since all solutions still belong to the same global solution space Ω, we do not need to define a best solution S_k^{best} for each sub-space ω_k. A single global solution S^{best} is always considered according to the EO theory.
- $N_k(S) \subset \omega_k \subset \Omega$ is still the *neighborhood* solution space but it identifies only solutions in ω_k.
- $C_k(S)$ is the cost function associated to the solutions in the sub-solution space ω_k.

Resorting to this extension, each ω_k can address a different subset of solutions. This allows us to separately explore the solution space, assigning each subset to the concept of a different *level* of optimization. We define a *level* of optimization as a specific optimization goal, i.e., optimizing the system by selecting the best operating system available or looking for the best hardware platform. In such schema, the level owns a set of peculiar information and definitions:

- S_k expresses either a system where a macro-component replacement takes place or a system where a micro-component has been replaced in some of the macro-components already in place. Discriminating between these two scenarios can be easily implemented by defining a proper $N_k(S)$ function.
- $N_k(S)$ changes level by level, in order to properly select solutions within the ω_k sub-solution space associated with it.
- $C_k(S)$ can be different for each level. Thus, cost functions can cover different aspects of the optimization and be customized with respect to the purpose of the level.

When the concept of different levels in the design is introduced, an operator enabling the search process to move across levels is required. This operator called next-level function is defined as follows:

- $nL(k, S_k, S^{best}) \subset [1, l]$ denotes the next-level function. Given the current optimization level, the current solution and the best solution it computes the next optimization level to reach.

Based on these definitions **Figure 4** shows the MLMOEO algorithm.

1. Initialize the starting level k.
2. Initialize a configuration S_k at will and set $S^{best} = S_k$. S^{best} will be the best solution so far.
3. For the current configuration S_k,
   a. Evaluate λ_i^k for each variable x_i^k;
   b. Find j with $\lambda_j^k \geq \lambda_i^k$ for all i; that is, x_j^k has the worst fitness;
   c. Choose a random $S'_k \in N_k(S_k)$ such as x_j^k must change;
   d. If $C_k(S'_k) < C_k(S^{best})$, store $S^{best} = S'_k$;
   e. Accept S'_k as new S_k unconditionally.
   f. Evaluate the $k' = nL(k, S_k, S^{best})$ and let k' be the new k.
4. Evaluate the $eS_k(S^{best})$ function:
   a. If false, repeat at step 2 unless the desired repetitions reached.
   b. If true, proceed with step 5.

Return S^{best}.

Figure 4 Multi-Level Multi-Objective Extremal Optimization

When applied to the design optimization problem the MLMOEO enables to obtain some significant goals:

1. Partitioning the solution space not only allows us to deal with replacements of full hierarchies of components but also enables to split the optimization process in different steps each driven by a different cost function thus giving priority during the optimization to certain design decision (e.g., optimizing the system by working first on the hardware architecture and resorting to software optimization only as fine grained solutions).
2. Working with variable solution spaces that adapt themselves based on the decision taken at the higher optimization levels enables to optimize the search time.
3. Working with different variable lambda functions λ_j^k depending on the level allows the evaluation of each design alternative in a different way, depending on the optimization objective set for that level.

2.3. Reliability Design Optimizer (ReDO) implementation

The MLMOEO strategy has been implemented into an Electronic Design Automation (EDA) tool integrated with the whole CLERECO reliability design tool suite. As for all other CLERECO tools, the development follows the Object Oriented Programming paradigm and it is provided with a cross-platform GUI. The implemented optimization engine includes a set of advanced features:

- The different steps of the algorithm have been developed with a modular architecture, in order to be able to exploit a multi-thread implementation and to take advantage of modern multi-core microprocessors. This is a critical feature given the complexity of the optimization process.
- All functions (i.e., all $C_k(S)$, λ_j^k, etc.) are defined through a virtual C++ evaluation class. This enables to have a general implementation of the algorithm that is not limited to a specific optimization function. The cost function can be easily defined at run-time to allow run-time adaptation of the design objectives.
- The trajectory followed by the optimization process (i.e., all generated design versions generated during the optimization) is stored in order to allow backtrack analy-

sis. This positively impacts the amount of design alternatives the designer can further evaluate once the engine completes its elaboration by suggesting the best solution.

- An optional asynchronous timeout policy has been implemented. Since the computation time required to perform a full optimization campaign may be unknown, we decided to implement a time search constraint that defines the maximum amount of time the engine can spend to optimize the system. The characteristic of the implemented optimization procedure guarantees that at any time a local best solution is always available and therefore, whatever is the time limit set by the user, the tool guarantees to provide a locally optimized solution.
- Different from the timeout policy, the end of the optimization can be also defined based on predefined contracts on selected parameters, e.g., the optimization stops when a solution able to guarantee a given failure rate for the system is identified, regardless if other solutions with better characteristics are available,

Figure 5 shows the preliminary implementation of the GUI with all parameters that can be set before running the engine.

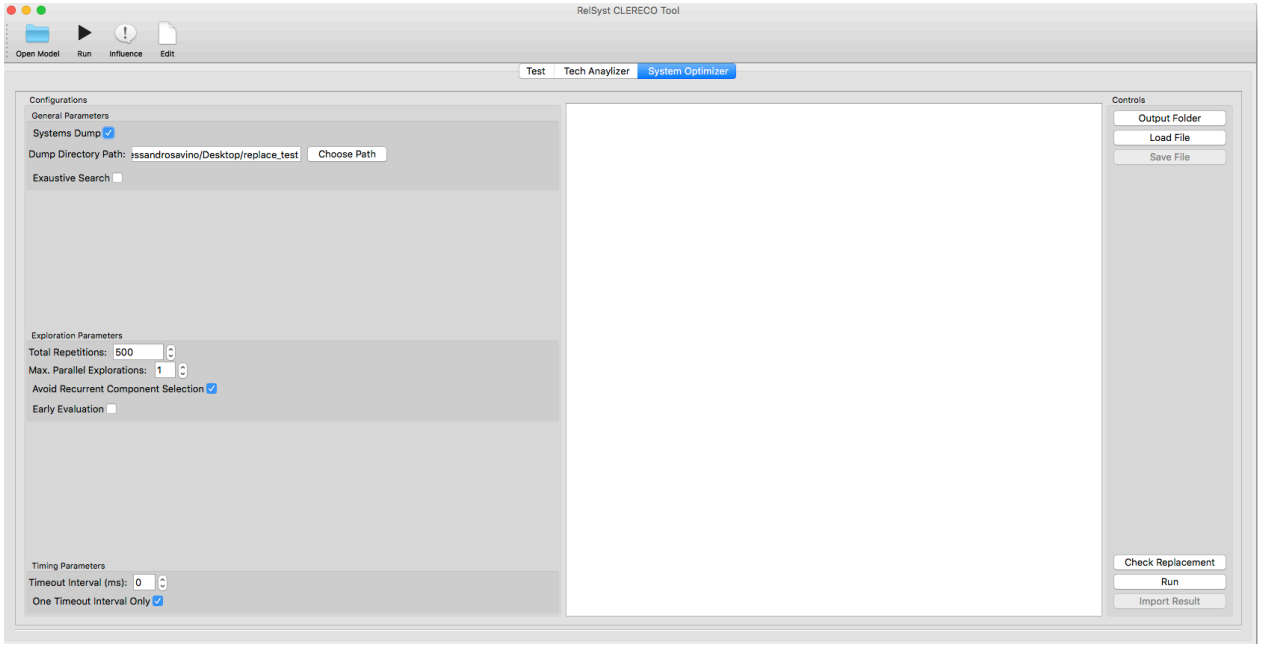

Figure 5: The EO engine configuration GUI

The optimization engine has been preliminarily validated with a set of experiments on simple benchmarks. In Table 1 we report information regarding the computation time based on some worst-case scenario test cases where:

- 4 levels of optimization have been defined:
  - Level 1 searches for the best technology layer between 2 available technologies.
  - Level 2 further searches for some reliability improvements based on different alternatives within the current technological layer.
  - Level 3 moves to hardware and software investigation by searching among:
    - 3 microprocessor architectures (x86, ARM11 and ARM15).
    - 2 different operating systems (Linux and Windows XP).
    - 2 sets of different implementations of the core function of the application (developed using C or Python languages).

They are all clusters of components since they include more then one component.

- Level 4 exploits some further reliability optimization by exploring alternative configurations/implementations of internal components of one of the clusters replaced at level 3.

All $C_k(S)$ and λ_j^k functions have been designed to take into account the reliability of the full system and the area required by the hardware layer. Data (i.e., areas, probabilities, etc.) have been randomly generated to stress the engine as much as possible. As an example, the next level policy applied for the data reported in Table 1 imposes to move back to level 1 after 50 evaluation steps and to level 3 after 10 evaluation steps. Remaining steps are equally distributed among level 2 and 4. This is a very drastic scenario where several movements among levels are performed. We believe that best policies should be more conservative avoiding a large number of drastic changes in the architecture of the system

.

Table 1: EO Engine preliminary computation times

Number of Evaluation Steps	500	1000	5000
Initialization Time (ms)	5	5	5
Evolution Time (ms)	4247	9860	53040

Figure 6 shows the system view tab of the ReDO tool, depicting the system selected by the optimization engine. Dark red nodes represent the updated nodes with respect to the original system that was reported in Figure 3. We can appreciate both the replacement of a full portion of the system (the technology layer and the microprocessor) and smaller improvements of the system by selecting different configurations/implementations of the components (i.e., the Core_C_FN2_DUP that represents the same function with duplication of variables and voting).

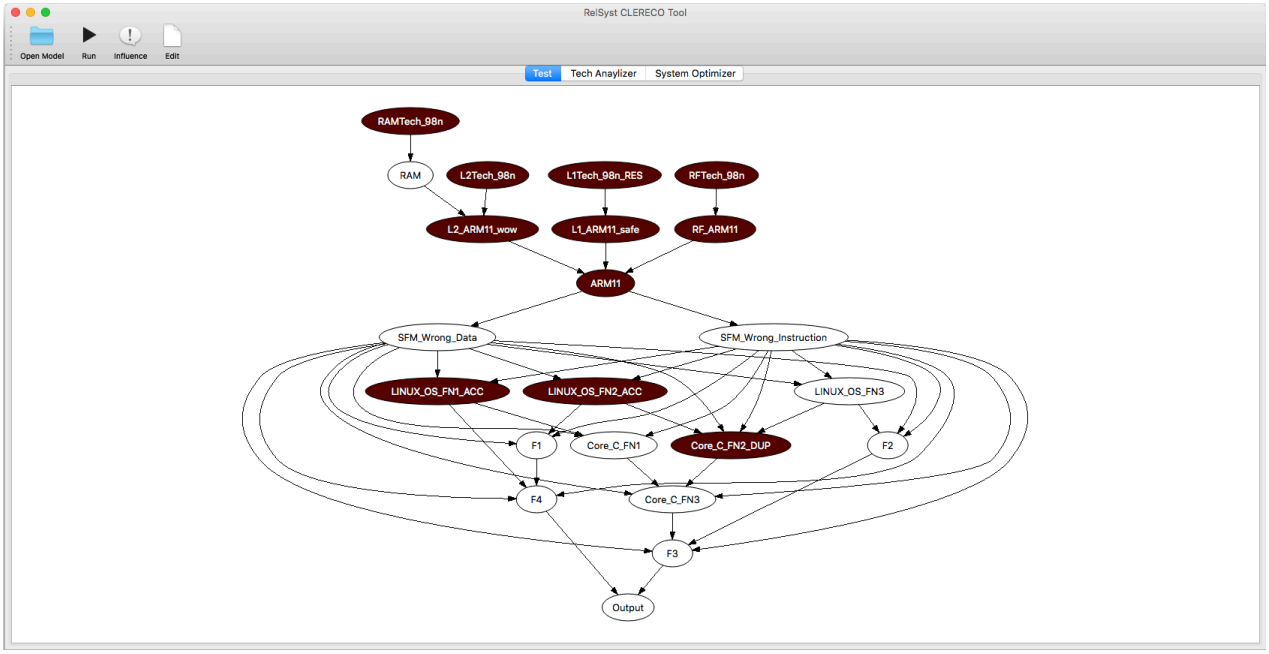

Figure 6: An optimized version of the system

3. Conclusion

This deliverable reported a general description of the reliability optimization heuristics developed on top of the CLERECO reliability estimation model. The majority of the work described in this deliverable was focused on the identification of the best optimization algorithm and on its implementation and integration in the overall CLERECO tool-suite.

Preliminary experiment performed on selected benchmarks provided interesting and promising results. In the remaining of the project an extensive campaign will be conducted both on benchmark applications and on the final CLERECO demonstrators to better assess the capability of this tools when dealing with complex real systems.

4. Bibliography

[1] R. K. Scott, J. W. Gault, and D. F. McAllister, "Fault-tolerant software reliability modeling," IEEE Transactions on Software Engineering, vol. SE-13, no. 5, pp. 582–592, May 1987.

[2] J. C. Laprie, J. Arlat, C. Beounes, and K. Kanoun, "Definition and analysis of hardware- and software-fault-tolerant architectures," Computer, vol. 23, no. 7, pp. 39–51, July 1990.

[3] K. Kanoun, M. Kaaniche, C. Beounes, J. C. Laprie, and J. Arlat, "Reliability growth of fault-tolerant software," IEEE Transactions on Reliability, vol. 42, no. 2, pp. 205–219, Jun 1993.

[4] A. Jackson, "Handbook-based high unit-value software reliability prediction method," in Reliability and Maintainability Symposium (RAMS), 2010 Proceedings - Annual, Jan 2010, pp. 1–4.

[5] A. Savino, S. D. Carlo, G. Politano, A. Benso, A. Bosio, and G. D. Natale, "Statistical reliability estimation of microprocessor-based systems," IEEE Transactions on Computers, vol. 61, no. 11, pp. 1521– 1534, Nov 2012.

[6] A. Vallero, A. Savino, S. Tselonis, N. Foutris, M. Kaliorakis, G. Politano, D. Gizopoulos, and S. Di Carlo, "A bayesian model for system level reliability estimation," in Test Symposium (ETS), 2015 20th IEEE European, May 2015, pp. 1–2.

[7] D. Coit, T. Jin, and H. Tekiner, "Review and comparison of system reliability optimization algorithms considering reliability estimation uncertainty," in Reliability, Maintainability and Safety, 2009. ICRMS 2009. 8th International Conference on, July 2009, pp. 49–53.

[8] Bo Xing, Wen-Jing Gao and T. Marwla, "The applications of computational intelligence in system reliability optimization," Computational Intelligence for Engineering Solutions (CIES), 2013 IEEE Symposium on, Singapore, 2013, pp. 7-14., doi: 10.1109/CIES.2013.6611722

[9] N. Wattanapongsakorn and S. Levitan, "Reliability optimization models for embedded systems with multiple applications," Reliability, IEEE Transactions on, vol. 53, no. 3, pp. 406–416, Sept 2004.

[10] H. Szu and R. Hartley, "Fast simulated annealing," Physics Letters A, vol. 122, no. 3, pp. 157 – 162, 1987.

[11] S. Boettcher and A. Percus, "Nature's way of optimizing," Artificial Intelligence, vol. 119, pp. 275 – 286, 2000.

[12] B. Selman, D. Mitchell, H. Levesqu, A new method for solving hard satisfiability problems, Proc. AAAI-92, San Jose, CA (1995), pp. 440–446

[13] K. Sneppen, P. Bak, H. Flyvbjerg, M.H. Jensen, Evolution as a self-organized critical phenomenon, Proc. Natl. Acad. Sci., Vol. 92 (1995), pp. 5209–5213

[14] I.H. Osman, J.P. Kelly (Eds.), Meta-Heuristics: Theory and Application, Kluwer, Boston, MA (1996)

[15] S. Boettcher, "Extremal optimization: heuristics via coevolutionary avalanches," Computing in Science Engineering, vol. 2, pp. 75–82, Nov 2000.

[16] S. Kirkpatrick, C.D. Gelatt, M.P. Vecchi, Optimization by simulated annealing, Science, Vol. 220 (1983), pp. 671–680

[17] D.G. Bounds, New optimization methods from physics and biology, Nature, Vol. 329 (1987), pp. 215–219

[18] A. Gharib, A. Harati, A.-T. Mohammad-Reza, and M. Jahan, "Toward application of extremal optimization algorithm in image segmentation," in Computer and Knowledge Engineering (ICCKE), 2012 2nd International eConference on, pp. 167–172, Oct 2012.

[19] M. Shafique, S. Rehman, P. V. Aceituno and J. Henkel, "Exploiting program-level masking and error propagation for constrained reliability optimization," Design Automation Conference (DAC), 2013 50th ACM/EDAC/IEEE, Austin, TX, 2013, pp. 1-9., doi: 10.1145/2463209.2488755

[20] A. Nakada, K. Tamura, H. Kitakami, and Y. Takahashi, "Population-based modified extremal optimization for contact map overlap maximization problem," in Advanced Applied Informatics (IIAIAAI), 2013 IIAI International Conference on, pp. 245–250, Aug 2013.

[21] P. Sanchez, H. Ayala, and C. Schaerer, "Mutual information extremal optimization for multimodal medical image registration," in Computing Conference (CLEI), 2014 XL Latin American, pp. 1–12, Sept 2014.

[22] J. Chen, G.-Q. Zeng, K.-D. Lu, W.-W. Peng, Z.-J. Zhang, and Y.-X. Dai, "Extremal optimization algorithm with adaptive constants dealing techniques for constrained optimization problems," in Industrial Electronics and Applications (ICIEA), 2014 IEEE 9th Conference on, pp. 1745–1750, June 2014.
[23] in-Rong Chen, Yong-Zai Lu, A novel elitist multiobjective optimization algorithm: Multiobjective extremal optimization, European Journal of Operational Research, Volume 188, Issue 3, 1 August 2008, Pages 637-651, ISSN 0377-2217, http://dx.doi.org/10.1016/j.ejor.2007.05.008.
[24] M. R. Chen, Jian Weng and Xia Li, "Multiobjective extremal optimization for portfolio optimization problem," Intelligent Computing and Intelligent Systems, 2009. ICIS 2009. IEEE International Conference on, Shanghai, 2009, pp. 552-556., doi: 10.1109/ICICISYS.2009.5357781
[25] Falco, Ivanoe and Cioppa, Antonio and Maisto, Domenico and Scafuri, Umberto and Tarantino, Ernesto, A Multiobjective Extremal Optimization Algorithm for Efficient Mapping in Grids (inbook), Applications of Soft Computing: From Theory to Praxis, Springer Berlin Heidelberg, 2009, doi: 10.1007/978-3-540-89619-7_36, pp. 367-377,