Tuzov, I.; Andrés, DD.; Ruiz, JC. (2018). Tuning synthesis flags to optimize implementation goals: Performance and robustness of the LEON3 processor as a case study. Journal of Parallel and Distributed Computing. 112:84-96. https://doi.org/10.1016/j.jpdc.2017.10.002

# Tuning synthesis flags to optimize implementation goals: performance and robustness of the LEON3 processor as a case study

Ilya Tuzov[a,*], David de Andrés[a], Juan-Carlos Ruiz[a]

[a] *ITACA, Universitat Politècnica de València, Campus de Vera s/n, 46022, Spain*

**Abstract**

The steady growth in complexity of FPGAs have led designers to rely more and more on manufacturers' and third parties' design tools to meet their implementation goals. However, as modern synthesis tools provide a myriad of different optimization flags, whose contribution towards each implementation goal is not clearly accounted for, designers just make use of a handful of those flags. This paper addresses the challenging problem of determining the best configuration of available synthesis flags to optimize the designers implementation goals. First, fractional factorial design is used to reduce the whole design space. Resulting configurations are implemented to estimate the actual impact, and statistical significance, of each considered synthesis flag. After that, multiple regression analysis techniques predict the expected outcome for each possible combination of these flags. Finally, multiple-criteria decision making techniques enable the selection of the best set of synthesis flags according to explicitly defined implementation goals.

*Keywords:* synthesis options, implementation goals optimization, design space exploration, multiple regression analysis, multiple-criteria decision making.

*Corresponding author

*Email addresses:* tuil@disca.upv.es (Ilya Tuzov), ddandres@disca.upv.es (David de Andrés), jcruizg@disca.upv.es (Juan-Carlos Ruiz)

## 1. Introduction

Design optimization is a common problem designers should face when dealing with complex digital designs. In addition to implementing the required logical function, designers must meet a number of different, and often conflicting, implementation goals, such as that the design should run at a given clock frequency (*performance*), take a given number of internal resources (*utilization*), run within a given power or energy budget (*consumption*), and provide a given grade of robustness [1].

Once the correctness of the functional specification is verified (usually through simulation), and due to the high complexity of modern FPGA devices, designers must rely on existing back-end tools to translate the HDL description of the design into a logical-level Field-Programmable Gate Array (FPGA) netlist (*synthesis*), map those logical elements to the actual physical device (*placement*), and interconnect those elements through the existing wiring (*routing*) [2]. Each of these back-end tools take as input the output from the previous stage, and applies a number of different optimizations to try to meet the desired implementation goals. Accordingly, as operating in a daisy chain fashion, the optimizations applied at the very first stage (the synthesis process) are critical towards meeting the required goals, as bad parametrization decisions will irremediably affect the rest of stages.

Due to its critical role in meeting the implementation goals, FPGA manufacturers and third party companies providing synthesis tools include a wide range of different optimization options suitable to different kind of devices, architectures, and scenarios. Nevertheless, far from alleviating the task of designers, the myriad of options available makes it very difficult to know the precise contribution of each one to a particular goal. Some options may have a greater impact in the implementation goals than others, some of them may have opposite effects, and most of them are never used because it is not clear enough what could rightly be expect from them [3].

Determining the best configuration of available synthesis flags for a given im-

plementation goals would require exploring the whole design space (all possible combinations of synthesis flags at different levels). FPGA manufacturers provide different tools for design space exploration, like Xilinx's SmartXplorer [4] or Altera's Design Space Explorer II [5], which perform several different implementations (changing the synthesis options) of the same design looking for the configuration that best meets the required implementation goals. Nevertheless, even when many-core machines or computer clusters can be used to perform the exploration in parallel, the time required to sweep the whole design space is prohibitive. For instance, considering that the 34 different synthesis options from Xilinx's XST synthesizer could be set at just two possible levels, and the implementation and simulation of the design takes just 1 minute (very optimistic), exploring the whole design space ($2^{34}$ configurations) will take roughly 32000 years running on a single-core machine.

Several works have dealt with the problem of design space exploration from different perspectives. For instance, a given architectural configuration for chip multiprocessor may required a couple of weeks to be simulated, so statistical simulation [6] or predictive modelling [7] are some of the proposed approaches to reduce that design space. Focusing on reconfigurable devices, evolutionary approaches were proposed in [8] to find a good solution in High-Level Synthesis with conflicting design objectives, [9] focused on the parametrization of soft-core processors through a greedy search method, and a calibration free algorithm for automatic optimization of design parameters was proposed in [10]. None of these works specifically focused on the parametrization of the synthesis, placement and routing processes, but on the architectural features of the designs to be implemented onto the reconfigurable device.

In this particular context, an approach based on machine-learning autotuning was presented in [11] to sample the parameter space and thus reduce the time devoted to the configuration search process. Nevertheless, although this approach, and those provided by FPGA manufacturers, may find a suitable configuration for the requested goals, the particular contribution of each selected option and their interactions are not accounted. Accordingly, designers are at

a loss when deciding how to configure the synthesis tool for each given design.

A very preliminary first step towards achieving this goal was taken at [12], which estimated the impact of different Xilinx's ISE optimization options on the power consumption of different security algorithms. However, that study considered just 4 different options (only one of them was related to the synthesis process), and focused on just one primary goal (power consumption). In addition, the contribution of each particular parameter to that goal was not determined, just the difference between configurations.

This paper focuses on the challenging issue of estimating the actual contribution of each synthesis flag towards a particular implementation goal, and determining the best possible configuration of the synthesis options to meet a given set of goals. As the whole design space cannot be explored within reasonable timing limits, we present a methodology based on operational research methods that can be used to i) greatly reduce the design space (*fractional factorial design*), ii) determine whether each synthesis flag statistically significantly impact the given implementation goals (*analysis of variance*), iii) predict the expected result for any combination of the synthesis flag across the whole design space (*linear regression and generalized linear regression*), and iv) select the best possible configuration according to specifically stated implementation goals (*multiple-criteria decision making*)). The feasibility and usefulness of this methodology is exemplified through a case study that considers how synthesis flags affect the performance and robustness properties of the LEON3 processor [13] implemented on a Virtex-6 FPGA, using Xilinx's ISE Design Suite and XST synthesis tool.

The rest of the paper is organized as follows. Section 2 establishes the theoretical framework for the proposed study. Section 3 defines the method for tuning the synthesis options to optimize performance and robustness, and describes its implementation for the Xilinx ISE toolkit. Section 4 describes the target LEON3 processor model and the workload selected as case study. Section 5 analyzes all the obtained results from an statistical perspective, determining the precise impact of each factor according to a multiple linear regression

model, and determining the best possible configuration according to different optimization goals. Finally, Section 6 presents the main conclusions and future work.

## 2. Background: Design of Experiments and its Statistical Analysis

The problem behind getting the precise contribution of each synthesis option towards a given implementation goal (*screening*), and thus being able to determine the best configuration to meet this goal (*response surface*), can be analysed following a statistical design of experiments procedure. This procedure allows researchers to plan experiments so that the data obtained can be analysed to yield valid and objective conclusions [14].

### *2.1. Full and fractional factorial design*

In particular, these problems usually require a *full factorial design*, in which every setting of every factor appears with every setting of every other factor. *Factors* ($X_i$) are those process inputs (in this case synthesis flags) that are deliberately changed to observe their effect on the *response variables* ($V_j$) (process outputs). Static properties of the design are directly estimated after its implementation, like resources utilization and minimum clock period, whereas dynamic properties, like dynamic power consumption, are estimated by simulating the design behaviour under a given workload. Likewise, robustness properties, like percentage of detected failures, are commonly assessed by the deliberately introduction of faults into design while running a given workload by means of simulation-based fault injection (SBFI) experiments. Consequently, estimating the response variables even for a single combination of factors settings is a very time-consuming problem, and the exploration of a full factorial design becomes unfeasible with increasing number of factors.

A selected subset of factors' settings form a *fractional factorial design*, which can considerably reduce the design space and, thus, the time required to estimate the effect of those factors. The fractional factorial design of experiments

exploits two main principles [15] [16] to reduce the design space without affecting the validity of drawn conclusions: i) *sparsity of effects*, which states that the number of relatively important effects and interactions in a factorial design is small, and ii) *hierarchical ordering*, which states that lower order effects are more likely to be important than higher order effects, main effects are more likely to be important than interactions, and effects of the same order are equally likely to be important. The combinations of factors settings should be carefully chosen so the design is both *balanced* (the combination of factors settings for any group of factors have the same number of observations) and *orthogonal* (the effects of any factor balance out (sum to zero) across the effects of the other factors) [14]. Furthermore, it is necessary to take into account the *resolution* of the design, which describes the degree to which estimated main effects are aliased (or confounded) with estimated low-level interactions [17]. The resolution of a design is one more than the smallest order interaction that some main effect is confounded with. Accordingly, to precisely determine the contribution of each factor towards the implementation goals, a fractional factorial design with resolution IV should be considered, so main effects are not confounded among one another nor with any two-factor interaction.

Fractional factorial designs are commonly denoted as $I_R^{K-P}$, where $I$ is the number of possible settings (*levels*) of each factor, $K$ is the number of factors, $R$ is the resolution of the design, and $1/I^P$ is the size the fraction with respect to a full factorial design. According to [18], 'two-level designs ($I = 2$) should be the cornerstone of industrial experimentation for product and process development, troubleshooting, and improvement.' In fact, most synthesis flags present a dual nature ('Yes/No', 'True/False', 'Enable/Disable'), whereas the rest can be modelled as a maximum and minimum threshold ('0%/100%'), or assume a compromise subset of settings ('Speed/Area'). Nevertheless, there exist some situations in which it could be necessary to consider factors at more than two levels ('Normal/High/Fast', for instance), at the cost of an even larger design space. If all factors are quantitative, then two-level designs with centre points should be employed [18]. Otherwise, existing algorithms, like the one proposed

in [19], can be used to generate the required mixed-level fractional factorial design.

### *2.2. Analysis of variance (ANOVA)*

The analysis of variance (ANOVA) is used to determine whether there are significant differences between the means of two or more independent (unrelated) groups [14]. The null hypothesis to be tested is that there is no difference in the population means of the different levels of a given factor. This procedure splits the total variation in the response variable into a part due to random errors (*sum of squares of error*) and a part due to changes in the levels of the selected factor (*sum of squares of treatment*). The degrees of freedom are split in the same way, and are used to compute the mean square for treatments and errors. If the null hypothesis is true, then both mean squares estimate the same quantity and its ratio (*F-test*) should be close to 1. If the probability (*p-value*) of an $F$ value being greater than or equal to the observed value is less than or equal to the significance level ($\alpha$, usually 0.05), then the null hypothesis is rejected (meaning that setting a given factor to one level or another has a statistically significant impact in the response variable observed).

### *2.3. Linear and generalised linear regression*

*Regression analysis* is a statistical technique that can be applied in this context to estimate the parameters of an equation relating a particular response variable to a set of factors. In such a way, it is possible to predict the value of the response variables for the whole design space even when only just a fraction has been actually explored. A linear regression model assumes that the relationship between the response variable and the factors is linear ($y = \beta_0 + \beta_1 X_1 + \ldots + \beta_n X_n$) [20]. The *coefficient of determination* ($R^2$) explains the proportion of the variance in the response variable that can be explained by the factors, i.e. how well the model fits the data (values for the response variable obtained through the fractional factorial design). In case the linear regression model does not adequately fit the data, then a more complex generalized linear regression

model can be used. The generalized linear regression is a generalization of the general linear model that can be used to predict responses for both variables with discrete distributions or which are not linearly related to the predictors.

### *2.4. Multiple-criteria decision making (MCDM)*

Multiple-criteria decision making (MCDM) methods provide a way for structuring complex problems and considering multiple, and usually conflicting, criteria to make more informed and better decisions [21]. As it is the case when implementing designs on reconfigurable devices, there is usually not a unique optimal solution for problems involving multiple criteria, so it is necessary to take into account the preferences of the decision maker (usually by weighting the relative importance of each criterion) to differentiate between available solutions. Different methods, each one with its own mathematical foundations, have been developed along the years, such as the Weight Sum Model (WSM) and the Weight Power Model (WPM) [22], the Analytic Hierarchy Process (AHP) [23], the VIseKriterijumska Optimizacija I Kompromisno Resenje (VIKOR) [24], or the Technique for Order of Preference by Similarity to Ideal Solution (TOPSIS) [25]. Any of these methods could be used to determine the best possible configuration of synthesis flags according to explicitly defined implementation goals.

## 3. Tuning synthesis options: proposed method and its actual implementation

All the previously described statistical techniques are the cornerstones of the proposed method, whose flowchart is depicted in Figure 1. It comprises two basic phases: implementation (phase 1) and analysis (phase 4). Simulation (phase 2) and fault injection experiments (phase 3) are executed only in case of considering implementation goals related to dynamic properties and robustness, respectively. Additionally, a preceding preparatory phase is defined to design the experiments to run.

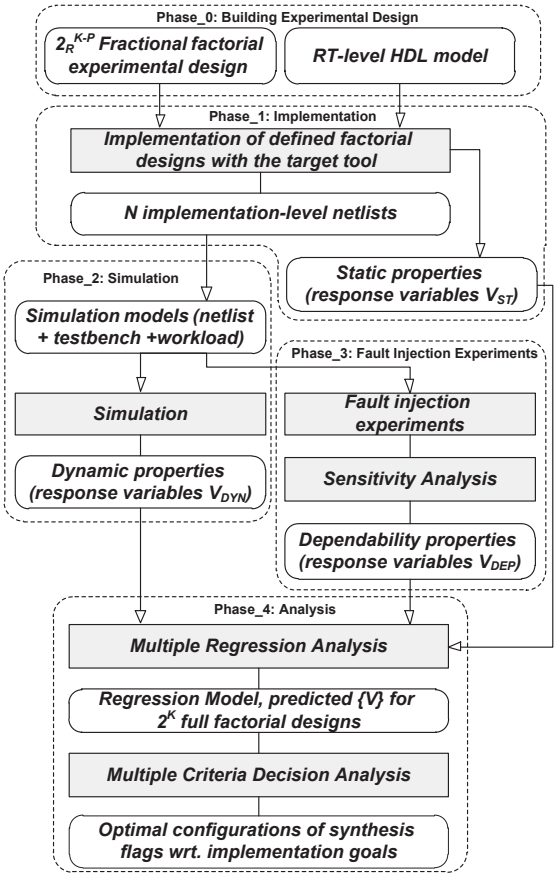


Figure 1: Experimental flow for identification of optimal configuration of synthesis flags

The proposed method has been implemented for the Xilinx ISE Design Suite [26], providing a detailed experimental procedure and a tool, which performs fractional factorial experiments in an automated way. Although most implementation tools are vendor-specific, they follow a quite similar design flow, thus the proposed implementation may be adapted to different toolkits. Implementation and simulation phases are executed in parallel in a multi-core machine, whereas simulation-based fault-injection experiments are executed in a grid-based computing system to achieve the maximum speed-up for the whole procedure.

### 3.1. Design of experiments

The main purpose of the preparatory phase is to define a fractional factorial design for the set of synthesis options (factors) selected among those available in the given synthesis tool (Xilinx's XST, in our case).

As previously explained, it is preferable to consider factors at just two levels due to existence of efficient analysis techniques and a greater reduction of the space to explore. Since our first goal is to estimate the precise contribution of each factor towards the implementation goals, the resolution of the experiment should be at least $IV$ (the effects of the main factors are not confounded with any two-factor interactions). Resulting fractional factorial design for $K$ two-level factors with resolution $IV$ is denoted as $2^{K-P}_{IV}$, and consists of a $K \times 2^{K-P}$ table where each cell ($L_{c,i}$) defines the setting of each factor ($X_i$) at the given treatment combination ($c$) (referred to as configuration from now on). Predefined designs for up to 11 factors can be found in [27] and [18], whereas MATLAB's Statistics and Machine Learning Toolbox [28] can be used to generate two-level fractional factorial designs with $2^{K-P}$ experiments and a given resolution using the Franklin-Bailey algorithm. For instance, a $2^{12-7}_{III}$ design can be obtained by running the MATLAB code from Listing 1.

```
1 generators=fracfactgen('a b c d e f g h i j k l',5,3);
2 [dff,confounding]=fracfact(generators);
```

Listing 1: Fractional factorial design generation with MATLAB

The resulting table is formatted and stored in a custom configuration file (*.xml) along with the set of options required for configuring and executing the rest of automated steps. This required information comprises the synthesis, map, place-and-route, and simulation tools command line options, the default clock period, the common source and destination directories, the fault models for the SBFI experiments, and maximum number of processes that could be executed in parallel, among other parameters.

### *3.2. Model implementation and static properties estimation*

The implementation phase parses the custom configuration file to build an internal list of all considered configurations (rows of factors combinations in the table). A particular synthesis file (*.xst*) is generated for each configuration ($c$) according to the specified factor values ($L_{c,i}$), so as to properly configure the synthesis process (XST tool) for each defined experiment. The *netgen* tool is used to create the required post-synthesis netlists.

After that, a constraints file (*.ucf*) is generated to specify a new goal for the maximum clock period (initially that defined in the configuration file) for that particular configuration. Then, in sequential order, the *translate*, *map*, and *par* tools are run to obtain the final implementation of the system on the target FPGA. Their particular reports are checked to detect any problem in the implementation process.

If no problem is detected, the *trce* (static timing) tool is executed to obtain a timing analysis report (*.twr*). This report is analysed to determine whether the clock constraints are met. If this is the case, the maximum clock period constraint is decreased by a parametrizable *delta_clk* factor. Otherwise, the maximum clock period constraint is increased by *delta_clk*. This implementation process is rerun until no gain in clock frequency is found, thus the maximum possible clock frequency for that particular configuration is reached. Finally, the post-map and post-par netlists are created using the *netgen* tool.

The estimations of those response variables related to static properties of the implementation of each configuration ($V_{\mathrm{ST}}$), like resources utilization and clock frequency, are retrieved from implementation (*.par*) and timing (*.twr*) reports respectively.

### *3.3. Model simulation for dynamic properties estimation*

The simulation phase is required to estimate those response variables related to dynamic properties of the system ($V_{\mathrm{DYN}}$), like power consumption. To do so, the resulting implementation-level netlist must be included into the test

environment and compiled to obtain the simulation model, which should be simulated under a representative workload.

First the workload settings are tuned according to the timing report for each particular configuration, and a simulation executable file is generated by means of the *fuse* tool. The commands required to monitor the switching activity of the system are included into a simulation script (*isim.cmd*), and the simulation time is scaled with respect to the finally attained clock frequency. The *testbench_isim_par.exe* executable file runs the simulation.

The resulting switching activity interchange format file (*.saif*) is fed to the *xpwr* tool to obtain an estimation of the dynamic power consumption in the resulting log file.

### *3.4. Fault injection for robustness properties estimation*

The estimation of those response variables related to the robustness, or dependability in general, of the system ($V_{\mathrm{DEP}}$) requires the simulation of the system both in absence and in presence of faults. Simulation-based fault injection (SBFI) [29] has been selected as a suitable technique, among the wide range of techniques existing nowadays, as it provides the best observability and repeatability of experiments, which is critical when deploying the very same experiments for different configurations. Accordingly, it is necessary to take into account the structural differences between implementations to ensure that the same faults are injected at same points at the same time during the simulation.

Each implementation-level netlist is parsed according to the naming convention of the selected synthesis tool (Xilinx's XST) to identify the common set of primitives (as defined in the vendor library) across all configurations (see Figure 2). As considering FPGAs as implementation targets, Flip-Flops (FFs) and Look-Up Tables (LUTs) are generally selected as suitable fault injection points, although other components like DSP or BlockRAM units could be also considered.

Resulting sets of common and complementary fault injection points are used to build an experiment specification file for each configuration. Each fault in-

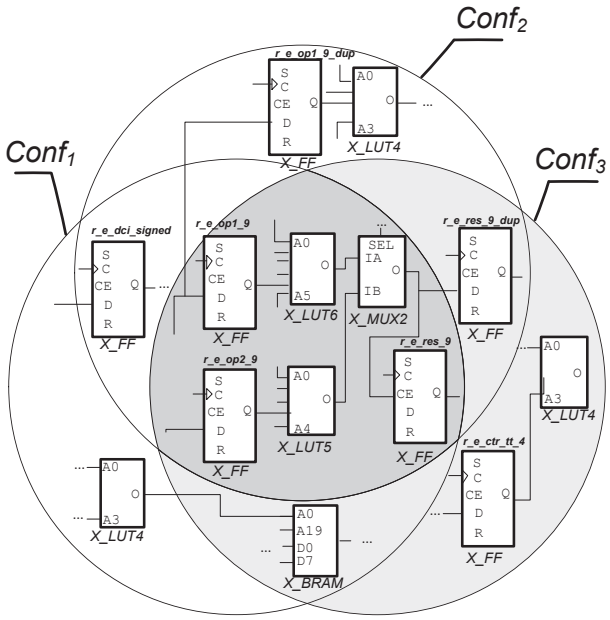


Figure 2: Matching primitives between multiple implementation-level netlists: common set (dark shaded) and complementary set for configuration 3 (light shaded)

jection points is assigned a fault model, forced value, injection instant, duration and observation time. The values assigned for elements within the common set are exactly the same for all configurations, ensuring the introduction of exactly the same faults. The clock period for the simulation is set up according to the maximum clock frequency reported for each configuration, thus the timing of all simulation processes is scaled according that value.

Prepared SBFI experiments are now executed in order to compare the behaviour of the system in presence and absence of faults. The execution of the experiments is controlled by means of a set of TCL scripts including commands for the selected simulator (Mentor Graphics' Model [30] in our case) to inject faults and monitor the system, like *force*, *change*, or *examine*. Observations collected for each experiment (state of the system according to internal signals, like FFs outputs) at given observation times are stored in observation dump files.

Once all observation dumps are collected they are compared with the reference one to identify four different failure modes that are generic enough to be applied to any system under study: i) *masked fault*, when outputs are correct and no errors (incorrect internal state) are observed at the end of observation interval, ii) *latent faults*, when outputs are correct but an erroneous internal state is observed (it may produce a failure later if extending the execution period), iii) *signalled failure*, when outputs are incorrect and the system raises an error flag (notifies the detection of the failure), and iv) *silent data corruption* (SDC), in case of failure not signalled by the system.

The result of this sensitivity analysis process is an estimation of the rates of these failure modes, the distribution of errors across the design hierarchy, and the fault to failure latency.

### *3.5. Analysis of results*

Once the values $M_{c,j}$ for all response variables ($V = V_{\mathrm{ST}} \cup V_{\mathrm{DYN}} \cup V_{\mathrm{DEP}}$) are available, the last phase that takes place is analysis, which includes the analysis of variance (ANOVA), building a regression model, and multi criteria decision making.

The ANOVA procedure is used to determine whether each considered factor (synthesis flag) statistically significantly contributes to each implementation goal. As previously stated, this is the case when the obtained p-value is below the significance level (usually 0.05). Those p-values can be easily computed by the MATLAB's Statistics and Machine Learning Toolbox [28], as depicted in Listing 2.

```
1 Tdata=readtable('Data.xlsx','Sheet',1);
2 T=Tdata(:,{'X1','X2','X3','MyResponseVariable'});
3 anovan(T.(3),{T.X1,T.X2,T.X3},'model','linear',
4         'varnames',{'X1','X2','X3'});
```

Listing 2: N-way analysis of variance for testing the effects of multiple factors on the mean of a response variable using MATLAB: (1) Reading data from an Excel file, (2) selecting data related to 3 factors and one response variable, and (3) computing the ANOVA

All obtained data can be also used to compute a regression model to predict the value of the response variables for any setting of synthesis options. The statistical significance of the obtained model should also should be assessed, checking the p-value to be less than the significance level (usually 0.05). Likewise the coefficient of determination $R^2$ denotes the proportion of the variance in the response variable that can be explained by the factors.

This proposal first makes use of a linear regression model and, in case the the quality of that model is considered insufficient to predict the responses (an $R^2$ threshold should be defined), a more complex generalized linear model is computed instead. This approach has been implemented using the MATLAB's Statistics and Machine Learning Toolbox [28], as shown in Listing 3. The procedure is iteratively applied, changing the starting model for the setpwise regression from 'linear' to 'interactions'.

```
1 Fit=fitlm(T,'linear','ResponseVar',MyResponseVariable');
2 if Fit.Rsquared.Ordinary < threshold
3     Fit=stepwiseglm(T,'linear','ResponseVar','MyResponseVar');
4     if Fit.Rsquared.Ordinary < threshold
5         Fit=stepwiseglm(T,'interactions',
6                         'ResponseVar','MyResponseVar');
7     end
8 end
```

Listing 3: Building an increasingly complex linear regression model until the coefficient of determination is above a given threshold using MATLAB

Finally, the expected values of each response variable for the whole set of $2^K$ possible synthesis configurations are computed, thus enabling to determine the best synthesis options configuration according to given implementation goals. In the particular case of implementing designs on FPGAs, those implementation goals are usually conflicting as increasing the clock frequency is likely to increase also the dynamic power consumption. In such cases, when there is no information about the particular preferences of the designer, it is sufficient to provide the Pareto optimal set (or Pareto frontier). It consists of all those

configurations in which it is impossible to improve a response variable without making worse another one. As no subjective information is provided, all of those configurations are considered a good solution towards optimizing the implementation goals.

However, if the designer provides her preferences about which implementation goal is more important than other (usually in the form of weights), then it is possible to estimate the best configuration of synthesis options according to different multi-criteria decision making (MCDM) methods. Although several different MCDM methods exists, this works focuses on the Weighted Sum Method (WSM) [22] to combine the different response variables into a single score accounting for the quality of the implementation according to the designer's goal. It must be noted that, in case of dealing with response variables expressed in different units, it is necessary to normalize (usually between 0 and 1) the predicted values prior to computing the WSM. In this case, Equation 1 can be used to normalize predicted response variables according to whether they should be interpreted as *the-higher-the-better* values, like clock frequency, or *the-lower-the-better* ones, like dynamic power consumption. Equation 2 shows how to compute the final score $S$ according to the *weights* ($\omega$) defined by the designer for each predicted response variable after normalization ($V^{*'}$).

$$V^{*'} = \begin{cases} V^* / V^*_{MAX}, & \text{if the-higher-the-better response variable} \\ V^*_{MIN} / V^*, & \text{otherwise} \end{cases} \tag{1}$$

where:

$V^*_{MAX}$**:** maximum $V^*$ across all configurations

$V^*_{MIN}$**:** minimum $V^*$ across all configurations

$$S = \sum_{i=1}^{M} \omega_i \times V_i^{*'} \tag{2}$$

where:

$M$**:** number of response variables

The configuration obtaining the highest score will be that optimizing the implementation goal according to the designer's preferences.

### *3.6. Summary*

The proposed method makes use of commonly used statistical techniques to determine the actual contribution of each synthesis flag towards the final implementation goals taking into account static and dynamic properties of the system, including robustness related ones. The defined procedure can also compute the best possible configuration of these flags according to the designer's preferences. This whole process has been automated, as depicted in Figure 3, to work under Xilinx's ISE Design Suite, Mentor Graphics' ModelSim, and MATLAB, although it can be adapted to use other manufacturers tools.

## 4. Case Study: implementing a LEON3 processor on a Virtex-6 FPGA

The case study selected to show the feasibility of the proposed approach consists in finding the actual contribution of a set of selected synthesis flags from Xilinx's XST tool, and which is their best configuration, towards the final clock frequency, dynamic power consumption, number of FFs and LUTS used, and robustness when implementing a LEON3 processor on a Virtex-6 FPGA (6vcx240tff784-2). This section details the different configuration parameters required for running the experimentation.

### *4.1. Factors*

The Xilinx's synthesis tool (XST) [31] provides 72 different options grouped into three categories: synthesis options, HDL options, and Xilinx specific options. Due to space limitations and to show the generality of the proposed approach, only those options commonly found in most state of the art synthesis tools have been selected as *factors* for experimentation. For instance, the effort the tool devotes to the synthesis process is defined by the *-opt_level*, *-name SYNTHESIS_EFFORT*, *-map_effort*, and *-effort* parameters in the Xilinx's XST, Intel's Quartus Prime, Synopsys' Design Compiler, and Cadence's

Encounter RTL Compiler, respectively. It must be noted that this does not limit the applicability of the proposed approach, which can take into account any available option indistinctly, when considering the implementation of a given design in a particular technology using specific tools.

The following list describes each of the selected factors, labelled from $X_1$ to $X_9$, its considered two levels (*low*/*high*), and its default value (in bold) when running a synthesis process with XST.

$X_1$ *Optimization Goal:* ***Speed*** */ Area.* This factor specifies the global optimization goal for the design. *Speed* reduces the levels of logic to achieve higher clock frequency, whereas *Area* reduces the total amount of logic used for design implementation. Faster circuits require more parallelism (increased utilization) but, as mentioned in [3], second-order effects of the FPGA implementation may produce unexpected effects. To prevent effects caused by FPGA utilization close to 100% a device with appropriate capacity should be selected for experimentation.

$X_2$ *Optimization Effort:* ***Normal*** */ High.* States the synthesis optimization level effort. *Normal* implies optimization by means of minimization and algebraic factoring algorithms, whereas *High* performs additional optimizations tuned to the selected device architecture. An existing third level (*Fast*) has not been considered as it just turns off some optimizations with the goal of minimising the synthesis runtime.

$X_3$ *Power Reduction:* ***No*** */ Yes.* Set to *Yes* optimizes the design to consume as little power as possible. As mentioned in [31], using that option may have a negative impact on the final overall utilization and performance.

$X_4$ *Keep Hierarchy:* ***No*** */ Yes.* Specifies whether the design units should be merged across the hierarchy. It simplifies post-synthesis simulation/verification, while potential negative effects in utilization and consumption could be expected. The third possible value (*Soft*) has not been considered, as it behaves as *Yes* but without enforcing this value to the rest of

implementation processes (place and route).

$X_5$ *Safe Implementation:* ***No*** */ Yes.* Specifies whether the implementation of Finite State Machines (FSM) will automatically recover from any illegal states. This option is associated with *FSM Encoding Algorithm*, which is set to the default *Auto* level. As recovery logic is introduced, an increase in device utilization and robustness could be expected.

$X_6$ *Resource Sharing: False /* ***True****.* Determines whether to share arithmetic operator resources. Although it usually involves creating additional multiplexing logic to select between factorized inputs, a reduction in device utilization could be expected.

$X_7$ *Register Duplication: False /* ***True****.* Specifies whether to replicate registers to control registers fanout, thus improving performance. A minor increase in device utilization is expected.

$X_8$ *Equivalent Register Removal: False /* ***True****.* Determines whether flip-flops that are equivalent or have constant inputs are removed. This increases the fitting success because of the logic simplification implied by the flip-flops elimination. Opposite effects than *Register Duplication* should be expected.

$X_9$ *LUT Combining: No / Area (default is* ***Auto****).* Merges LUT pairs with common inputs into single dual-output LUT6 to improve design utilization. It may decrease performance. *Area* performs the maximum possible combining, whereas *No* disables combining. The third possible value for this particular factor (*Auto*), which looks for a trade-off between performance and utilization, has not been considered in favour of the two more aggressive alternatives.

It must be noted that, as the goal is to precisely determine the impact of each factor on the response variables, the options for the rest of processes involved in the implementation process (*translate*, *map* and *place-and-route* in Xilinx's ISE design flow) have been set to their default value.

*4.2. Response Variables*

Implementation goals that can be directly estimated after the target design is placed and routed, and that could be dependent on the synthesis options selected, include *maximizing performance*, and *minimizing utilization* and *consumption*.

As FPGAs are inherently synchronous systems, the *performance* of any implemented design is usually estimated by means of the *maximum clock frequency* (MHz) (or minimum clock period (ns)) attainable. The timing analysis tools provided by the manufacturer (Xilinx's Timing Analyzer [32]) will be used to estimate this variable.

Estimating the device *utilization* through a single variable, after implementing the target design, is not so easily done when considering FPGAs as the underlying implementation technology. For instance, Xilinx's Map Report gives a precise account of the *number of flip-flops* (FFs) and *look-up tables* (LUTs) used to implement the sequential and combinational logic, respectively, of the design. The precise contribution of each variable to the final goal should be defined before running the experimentation.

There are two different variables that contribute to the total *power consumed* by the implemented design, the static and the dynamic power consumption. Synthesis options are very unlikely to have any impact on the static power consumption, as it mainly depends on the selected target device. So this study will just consider the effect of synthesis options on the *dynamic power consumption*, which makes reference to the additional power consumed due to the switching activity in the implemented design. Xilinx's XPower tool [33] provides a full report for both types of power consumption.

Implementation goals related to robustness are estimated by means of a sensitivity analysis of the results obtained after the execution of SBFI experiments. Resulting response variables include the previously defined rates of four failure modes (masked fault, latent fault, signalled failure and silent data corruption) for each type of fault targets and fault model. Accordingly, targeting only FFs (sequential logic) and LUTs (combinational logic), and selecting only two mod-

els of permanent faults (stuck-at-1, stuck-at-0) and a single model of transient fault (bit-flip applicable only to sequential logic), will provide a set of 20 response variables. Although these variables are estimated separately, they are all joined together into a single score for each considered failure mode.

### *4.3. Fractional Factorial Design*

Even though the number of considered factors has been reduced to just 9, it will take $2^9 = 512$ different configurations to run a full factorial design experiment. Accordingly the $2^{9-4}_{IV}$ fractional factorial design defined in [27], and listed in Table 1, has been selected to reduce the total number of configurations to just 32 while maintaining all the required properties. The low- and high-level for each factor has been coded as 0 and 1, respectively. In such a way, each configuration can be easily identified by a single vector with each bit set to the corresponding level of the 9 considered factors. For instance, configuration 0 is represented by $\{000001111\}$, stating that factors $X_1$ to $X_5$ should take a low level, whereas factors $X_6$ to $X_9$ should take a high level.

### *4.4. LEON3 Processor Model*

LEON is a family of 32-bit processors compliant with the SPARC V8 architecture, whose fault-tolerant version is used as the European Space Agency (ESA) standard microprocessor. The third generation of this widely used soft core processor, LEON3 [13], is fully customizable from a single-core without cache and MMU minimal configuration up to 16-core AMP/SMP high-performance assembly. Its full source VHDL model is distributed under the GNU GPL license.

The general-purpose LEON3 configuration selected for this case study is presented in Figure 4. The single-core assembly includes a 7-stage pipelined integer unit, data and instruction caches, and a register file consisting of 8 windows of 16 registers each. The boot image is loaded from main PROM, whereas a UART is used as debugging interface.

Table 1: Proposed $2^{9-4}$ fractional factorial design, with low and high levels coded as 0 and 1, respectively.

| | **Factors** | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| **Config.** | $X_1$ | $X_2$ | $X_3$ | $X_4$ | $X_5$ | $X_6$ | $X_7$ | $X_8$ | $X_9$ |
| **0** | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 |
| **1** | 1 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 |
| **2** | 0 | 1 | 0 | 0 | 0 | 0 | 1 | 0 | 0 |
| **3** | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 1 | 1 |
| **4** | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 1 | 0 |
| **5** | 1 | 0 | 1 | 0 | 0 | 0 | 1 | 0 | 1 |
| **6** | 0 | 1 | 1 | 0 | 0 | 1 | 0 | 0 | 1 |
| **7** | 1 | 1 | 1 | 0 | 0 | 1 | 1 | 1 | 0 |
| **8** | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 1 |
| **9** | 1 | 0 | 0 | 1 | 0 | 0 | 1 | 1 | 0 |
| **10** | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 |
| **11** | 1 | 1 | 0 | 1 | 0 | 1 | 1 | 0 | 1 |
| **12** | 0 | 0 | 1 | 1 | 0 | 1 | 1 | 0 | 0 |
| **13** | 1 | 0 | 1 | 1 | 0 | 1 | 0 | 1 | 1 |
| **14** | 0 | 1 | 1 | 1 | 0 | 0 | 1 | 1 | 1 |
| **15** | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 |
| **16** | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 |
| **17** | 1 | 0 | 0 | 0 | 1 | 0 | 1 | 1 | 1 |
| **18** | 0 | 1 | 0 | 0 | 1 | 1 | 0 | 1 | 1 |
| **19** | 1 | 1 | 0 | 0 | 1 | 1 | 1 | 0 | 0 |
| **20** | 0 | 0 | 1 | 0 | 1 | 1 | 1 | 0 | 1 |
| **21** | 1 | 0 | 1 | 0 | 1 | 1 | 0 | 1 | 0 |
| **22** | 0 | 1 | 1 | 0 | 1 | 0 | 1 | 1 | 0 |
| **23** | 1 | 1 | 1 | 0 | 1 | 0 | 0 | 0 | 1 |
| **24** | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 0 |
| **25** | 1 | 0 | 0 | 1 | 1 | 1 | 0 | 0 | 1 |
| **26** | 0 | 1 | 0 | 1 | 1 | 0 | 1 | 0 | 1 |
| **27** | 1 | 1 | 0 | 1 | 1 | 0 | 0 | 1 | 0 |
| **28** | 0 | 0 | 1 | 1 | 1 | 0 | 0 | 1 | 1 |
| **29** | 1 | 0 | 1 | 1 | 1 | 0 | 1 | 0 | 0 |
| **30** | 0 | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 0 |
| **31** | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| **Default** | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | Auto |

Just to notice the complexity of this processor, the flattened post place-and-route model for those configurations with *Keep Hierarchy = No* consists of 1 *.vhd* and 1 *.sdf* file (as expected), but those configurations with *Keep Hierarchy*

= *Yes* present a total of 173 *.vhd* and 88 *.sdf* files.

### *4.5. Workload*

In order to obtain an accurate estimation of the dynamic power consumption and robustness properties, a realistic workload has been defined for the target model. This workload, and adaptation of the *basicmath* automotive benchmark from the MiBench benchmark suite [34], performs several matrix multiplications. The program, developed in C, fills the matrices with pseudo-randomly generated 32-bits integers to produce outputs in the full 32-bits range. Results are stored in SRAM memory and written to the standard output (debugging UART interface), which is in turn forwarded to the simulator console.

### *4.6. Fault Injection and Observation Targets*

The *IU3* Integer Unit has been selected as the target for fault injection in this case study. As the HDL model is implemented on an FPGA device, all used FFs and LUTs have been selected for injecting faults into the sequential and combinational logic of the design, respectively. Table 2 lists the minimum/maximum number of elements targeted (configuration-dependent), the considered fault models, the number of experiments for each fault model and target, and the injection interval with respect to time required to run the workload. It must be noted that, in the case of bit-flips, 3 experiments are performed, and with a wider injection interval, as their impact is very dependent on the time they are injected.

The list of observation points, which defines the state vector captured during observation, has been defined to be able to determine whether a failure has occurred and the internal state of the target integer unit. In particular, it includes all the registers from the integer unit, the register file from the processor, and the memory area storing the output of the workload.

Table 2: Configuration of fault injection experiments

| Entity | Number of targets | Fault models | Experiments per target | Injection interval |
|---|---|---|---|---|
| FFs | 832/840 | stuck-at-1 | 1 | [0.1 : 0.5] |
| | | stuck-at-0 | 1 | [0.1 : 0.5] |
| | | bit-flip | 3 | [0.1 : 0.7] |
| LUTs | 1692/1820 | stuck-at-1 | 1 | [0.1 : 0.5] |
| | | stuck-at-0 | 1 | [0.1 : 0.5] |

## 5. Results

The previously described experimental procedure was applied on the selected target processor and workload. The values obtained for response variables $V_1 - V_4$ (performance, consumption, utilization of FFs and LUTs) for each configuration are listed in Table 3, whereas estimations of response variables $V_5 - V_{24}$ (robustness properties) are listed in Table 4. These tables also list the best and worst observed values as a reference for comparison between the default and selected configurations.

From Table 3, it can be easily seen that considered factors do not have a great impact on FFs utilization ($V_4$, with a 3.44% of improvement between best and worst observations), whereas the rest of considered response variables present a large room for improvement across selected configuration (ranging from 24.86% to 48.42%).

In the case of robustness properties (see Table 4), this improvement depends on the selected fault model and target. For FFs this improvement is higher when injecting bit-flips ($2.92\% - 13.72\%$, across all four failure modes) than when injecting stuck-at faults ($0.46\% - 5.60\%$ for stuck-at-0, $0.69\% - 1.94\%$ for stuck-at-1). For stuck-at faults injected into LUTs this improvement ranges between $8.61\% - 14.63\%$ for stuck-at-0 and $5.8\% - 19.8\%$ for stuck-at-1.

Table 5 joins together information related to the ANOVA and linear regression processes. On the one hand, each factor has been tested for its statistical

Table 3: Observed values for performance, consumption and utilization (response variables $V_1 - V_4$).

| | **Performance** | **Consumption** | **Utilization** | |
|---|---|---|---|---|
| | **clk. freq. (MHz)** | **dyn. power (mW)** | **Flip-Flops** | **LUTs** |
| **Config.** | $V_1$ | $V_2$ | $V_3$ | $V_4$ |
| 0 | 111.570 | 127.81 | 3570 | 6111 |
| 1 | 111.173 | 145.12 | 3622 | 6372 |
| 2 | 117.730 | 142.92 | 3634 | 6970 |
| 3 | 105.396 | 139.87 | 3559 | 5694 |
| 4 | 125.078 | 148.81 | 3598 | 6778 |
| 5 | 100.412 | 128.09 | 3622 | 5743 |
| 6 | 117.716 | 137.83 | 3661 | 6098 |
| 7 | 105.419 | 137.64 | 3559 | 6217 |
| 8 | 125.078 | 150.55 | 3661 | 6118 |
| 9 | 100.341 | 133.21 | 3559 | 6247 |
| 10 | 133.404 | 152.45 | 3598 | 6649 |
| 11 | 105.731 | 129.74 | 3622 | 5607 |
| 12 | 125.063 | 145.55 | 3633 | 6933 |
| 13 | 100.351 | 128.86 | 3559 | 5582 |
| 14 | 117.716 | 142.85 | 3571 | 5917 |
| 15 | 105.352 | 142.24 | 3622 | 6300 |
| 16 | 117.716 | 142.29 | 3688 | 6654 |
| 17 | 100.341 | 126.08 | 3658 | 5848 |
| 18 | 111.235 | 131.70 | 3665 | 5850 |
| 19 | 111.297 | 138.69 | 3681 | 6348 |
| 20 | 117.994 | 140.67 | 3694 | 5856 |
| 21 | 111.433 | 140.64 | 3658 | 6442 |
| 22 | 117.855 | 133.60 | 3673 | 6679 |
| 23 | 105.352 | 136.70 | 3681 | 5564 |
| 24 | 125.125 | 143.07 | 3677 | 6606 |
| 25 | 100.160 | 128.73 | 3681 | 5767 |
| 26 | 111.371 | 123.65 | 3701 | 5789 |
| 27 | 105.519 | 139.57 | 3657 | 6303 |
| 28 | 111.185 | 125.30 | 3664 | 5805 |
| 29 | 105.396 | 140.18 | 3681 | 6405 |
| 30 | 117.772 | 138.80 | 3688 | 6647 |
| 31 | 105.574 | 136.30 | 3657 | 5634 |
| Default | 125.282 | 145.10 | 3641 | 6369 |
| Best value | 133.404 | 123.65 | 3571 | 5582 |
| Worst value | 100.160 | 152.45 | 3694 | 6970 |

Table 4: Observed values (%) for robustness (response variables $V_5 - V_{24}$).

| | **LUTs** | | | | | | | | **Flip-Flops** | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | **Stuck-at-1** | | | | **Stuck-at-0** | | | | **Stuck-at-1** | | | | **Stuck-at-0** | | | | **Bit-Flip** | | | |
| | **M** | **L** | **S** | **SDC** | **M** | **L** | **S** | **SDC** | **M** | **L** | **S** | **SDC** | **M** | **L** | **S** | **SDC** | **M** | **L** | **S** | **SDC** |
| **Config.** | $V_5$ | $V_6$ | $V_7$ | $V_8$ | $V_9$ | $V_{10}$ | $V_{11}$ | $V_{12}$ | $V_{13}$ | $V_{14}$ | $V_{15}$ | $V_{16}$ | $V_{17}$ | $V_{18}$ | $V_{19}$ | $V_{20}$ | $V_{21}$ | $V_{22}$ | $V_{23}$ | $V_{24}$ |
| 0 | 21.66 | 5.79 | 18.91 | 53.64 | 40.19 | 4.69 | 7.28 | 47.85 | 10.20 | 27.49 | 20.77 | 41.54 | 37.70 | 13.09 | 9.48 | 39.74 | 74.63 | 7.52 | 4.04 | 13.81 |
| 1 | 19.05 | 6.78 | 20.11 | 54.06 | 38.69 | 4.38 | 6.72 | 50.20 | 10.26 | 27.80 | 21.00 | 40.93 | 38.07 | 13.13 | 9.19 | 39.62 | 74.86 | 7.48 | 3.90 | 13.76 |
| 2 | 20.86 | 6.11 | 19.32 | 53.71 | 39.68 | 4.29 | 6.88 | 49.15 | 10.25 | 27.89 | 20.74 | 41.12 | 38.02 | 13.11 | 9.30 | 39.57 | 75.96 | 7.39 | 3.81 | 12.83 |
| 3 | 19.31 | 6.30 | 20.01 | 54.38 | 37.96 | 4.53 | 6.59 | 50.91 | 10.22 | 27.40 | 21.03 | 41.35 | 37.74 | 13.10 | 9.25 | 39.90 | 76.52 | 7.37 | 3.57 | 12.54 |
| 4 | 21.67 | 5.64 | 19.18 | 53.51 | 40.08 | 4.59 | 7.46 | 47.87 | 10.20 | 27.49 | 20.77 | 41.54 | 37.70 | 13.09 | 9.36 | 39.86 | 76.35 | 7.40 | 3.52 | 12.73 |
| 5 | 19.13 | 6.75 | 20.07 | 54.05 | 38.62 | 4.46 | 6.69 | 50.23 | 10.26 | 27.80 | 20.88 | 41.05 | 38.07 | 13.13 | 9.07 | 39.74 | 76.17 | 7.32 | 3.70 | 12.81 |
| 6 | 20.99 | 6.32 | 19.23 | 53.46 | 40.00 | 4.34 | 6.98 | 48.68 | 10.25 | 27.89 | 20.86 | 41.00 | 38.02 | 13.11 | 9.42 | 39.45 | 75.96 | 7.39 | 3.69 | 12.95 |
| 7 | 18.90 | 6.56 | 20.44 | 54.11 | 37.92 | 4.49 | 6.85 | 50.74 | 10.22 | 27.40 | 20.91 | 41.47 | 37.74 | 13.10 | 9.25 | 39.90 | 76.08 | 7.37 | 3.61 | 12.94 |
| 8 | 22.28 | 6.30 | 18.68 | 52.74 | 41.46 | 4.64 | 7.13 | 46.77 | 10.25 | 27.89 | 20.74 | 41.12 | 38.02 | 13.11 | 9.30 | 39.57 | 76.36 | 7.31 | 3.54 | 12.79 |
| 9 | 19.10 | 6.72 | 19.68 | 54.50 | 38.26 | 4.85 | 6.95 | 49.94 | 10.22 | 27.40 | 20.79 | 41.59 | 37.74 | 13.10 | 9.13 | 40.02 | 75.96 | 7.41 | 3.69 | 12.94 |
| 10 | 20.84 | 6.20 | 19.58 | 53.39 | 40.59 | 4.88 | 7.00 | 47.53 | 10.20 | 27.49 | 20.89 | 41.42 | 37.70 | 13.09 | 9.48 | 39.74 | 76.83 | 7.36 | 3.60 | 12.20 |
| 11 | 18.62 | 6.91 | 19.73 | 54.74 | 38.41 | 4.57 | 7.03 | 50.00 | 10.26 | 27.80 | 21.00 | 40.93 | 38.07 | 13.13 | 9.19 | 39.62 | 76.49 | 7.36 | 3.70 | 12.45 |
| 12 | 22.25 | 5.96 | 18.55 | 53.23 | 41.25 | 4.80 | 7.34 | 46.60 | 10.25 | 27.89 | 20.86 | 41.00 | 38.02 | 13.11 | 9.42 | 39.45 | 76.60 | 7.27 | 3.69 | 12.44 |
| 13 | 18.92 | 6.42 | 19.98 | 54.68 | 37.89 | 4.66 | 7.07 | 50.38 | 10.22 | 27.40 | 20.91 | 41.47 | 37.74 | 13.10 | 9.01 | 40.14 | 75.96 | 7.41 | 3.53 | 13.10 |
| 14 | 20.91 | 6.03 | 19.82 | 53.25 | 40.84 | 4.60 | 7.12 | 47.44 | 10.20 | 27.49 | 20.89 | 41.42 | 37.70 | 13.09 | 9.48 | 39.74 | 75.79 | 7.56 | 3.84 | 12.81 |
| 15 | 19.07 | 6.36 | 19.84 | 54.74 | 38.20 | 4.47 | 6.89 | 50.44 | 10.26 | 27.80 | 20.88 | 41.05 | 38.07 | 13.13 | 9.19 | 39.62 | 76.73 | 7.36 | 3.54 | 12.37 |
| 16 | 21.49 | 5.72 | 19.99 | 52.80 | 40.53 | 4.44 | 7.00 | 48.03 | 10.25 | 27.89 | 20.74 | 41.12 | 38.02 | 13.11 | 9.42 | 39.45 | 76.00 | 7.43 | 3.81 | 12.75 |
| 17 | 20.27 | 5.92 | 20.22 | 53.59 | 38.95 | 4.56 | 7.23 | 49.26 | 10.22 | 27.40 | 20.79 | 41.59 | 37.74 | 13.10 | 9.13 | 40.02 | 76.00 | 7.37 | 3.65 | 12.98 |
| 18 | 21.76 | 6.10 | 19.97 | 52.18 | 39.99 | 4.59 | 6.94 | 48.49 | 10.20 | 27.49 | 21.01 | 41.30 | 37.70 | 13.09 | 9.48 | 39.74 | 74.59 | 7.52 | 4.08 | 13.81 |
| 19 | 20.08 | 6.29 | 20.14 | 53.49 | 38.73 | 4.46 | 7.72 | 49.08 | 10.26 | 27.80 | 21.00 | 40.93 | 38.07 | 13.13 | 8.95 | 39.86 | 74.94 | 7.44 | 3.90 | 13.72 |
| 20 | 21.55 | 5.78 | 19.75 | 52.92 | 40.35 | 4.49 | 7.01 | 48.15 | 10.25 | 27.89 | 20.86 | 41.00 | 38.02 | 13.11 | 9.42 | 39.45 | 76.04 | 7.43 | 3.69 | 12.83 |
| 21 | 20.10 | 6.18 | 19.76 | 53.96 | 38.88 | 4.51 | 6.82 | 49.80 | 10.22 | 27.40 | 21.15 | 41.23 | 37.74 | 13.10 | 9.13 | 40.02 | 74.79 | 7.45 | 3.97 | 13.79 |
| 22 | 21.89 | 5.71 | 19.99 | 52.41 | 40.03 | 4.48 | 7.05 | 48.43 | 10.20 | 27.49 | 20.89 | 41.42 | 37.70 | 13.09 | 9.36 | 39.86 | 75.83 | 7.52 | 3.88 | 12.77 |
| 23 | 20.10 | 6.40 | 19.99 | 53.51 | 39.01 | 4.45 | 7.25 | 49.29 | 10.26 | 27.80 | 21.00 | 40.93 | 38.07 | 13.13 | 9.19 | 39.62 | 76.49 | 7.32 | 3.62 | 12.57 |
| 24 | 21.88 | 5.54 | 19.62 | 52.97 | 40.59 | 4.41 | 7.01 | 47.99 | 10.19 | 27.46 | 20.86 | 41.49 | 37.77 | 13.07 | 9.47 | 39.69 | 76.18 | 7.31 | 3.64 | 12.87 |
| 25 | 19.90 | 6.33 | 19.90 | 53.88 | 39.17 | 4.62 | 7.24 | 48.97 | 10.26 | 27.80 | 20.76 | 41.17 | 38.07 | 13.13 | 9.07 | 39.74 | 76.01 | 7.40 | 3.70 | 12.89 |
| 26 | 21.92 | 6.13 | 20.35 | 51.60 | 40.58 | 4.38 | 7.31 | 47.72 | 10.24 | 27.86 | 20.95 | 40.95 | 38.10 | 13.10 | 9.29 | 39.52 | 74.72 | 7.46 | 4.05 | 13.77 |
| 27 | 19.76 | 5.92 | 20.22 | 54.11 | 38.66 | 4.48 | 7.58 | 49.28 | 10.22 | 27.40 | 21.15 | 41.23 | 37.74 | 13.10 | 9.25 | 39.90 | 76.16 | 7.45 | 3.69 | 12.70 |
| 28 | 22.04 | 5.69 | 19.65 | 52.62 | 41.00 | 4.38 | 6.95 | 47.67 | 10.20 | 27.49 | 21.01 | 41.30 | 37.70 | 13.09 | 9.48 | 39.74 | 74.87 | 7.52 | 4.04 | 13.57 |
| 29 | 19.75 | 6.05 | 20.03 | 54.17 | 38.81 | 4.51 | 7.19 | 49.49 | 10.26 | 27.80 | 21.00 | 40.93 | 38.07 | 13.13 | 9.19 | 39.62 | 76.17 | 7.36 | 3.62 | 12.85 |
| 30 | 22.00 | 6.22 | 20.21 | 51.57 | 40.09 | 4.59 | 7.56 | 47.76 | 10.25 | 27.89 | 20.74 | 41.12 | 38.02 | 13.11 | 9.42 | 39.45 | 76.08 | 7.51 | 3.89 | 12.51 |
| 31 | 19.81 | 6.15 | 20.20 | 53.84 | 39.04 | 4.84 | 7.17 | 48.95 | 10.22 | 27.40 | 21.03 | 41.35 | 37.74 | 13.10 | 9.25 | 39.90 | 76.12 | 7.41 | 3.65 | 12.82 |
| Default | 21.84 | 6.03 | 19.13 | 53.01 | 40.19 | 4.64 | 7.08 | 48.09 | 10.20 | 27.49 | 20.77 | 41.54 | 37.70 | 13.09 | 9.36 | 39.86 | 76.31 | 7.32 | 3.56 | 12.81 |
| Best value | 22.28 | 5.54 | 20.44 | 51.57 | 41.46 | 4.29 | 7.72 | 46.60 | 10.26 | 27.40 | 21.15 | 40.93 | 38.10 | 13.07 | 9.48 | 39.45 | 76.83 | 7.27 | 4.08 | 12.20 |
| Worst value | 16.62 | 6.91 | 18.55 | 54.74 | 37.89 | 4.88 | 6.59 | 50.91 | 10.19 | 27.89 | 20.74 | 41.59 | 37.70 | 13.13 | 8.95 | 40.14 | 74.59 | 7.56 | 3.52 | 13.81 |

*Failure modes: M - Masked fault, L - Latent fault, S - Signalled fault, SDC - Silent Data Corruption*

significance for each response variable. Those factors that statistically significantly impact each response variable (p-value $< 0.05$) are in bold typeface.

Surprisingly, some factors do not contribute significantly to any response variable. It must be noted that this does not mean they do not impact somehow the response variables, but that this impact is not enough to be considered statistically significant for this case study.

Two of the more surprising cases is that the *Optimization Effort* ($X_2$) option does not significantly contribute to improve performance goals (speed, power and utilization), and that *Power reduction* ($X_3$) does not really reduce the

Table 5: Estimators ($\beta_{i,j}$) accounting the impact on the response variable $V_j$ of a *high* level on factor $X_i$. Those with a statistically significant impact are in bold typeface.

| **Response variables** | | | **Factors** | | | | | | | | | **Intercept** | $R^2$ | $R^2_{GL}$ |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | $X_1$ | $X_2$ | $X_3$ | $X_4$ | $X_5$ | $X_6$ | $X_7$ | $X_8$ | $X_9$ | | | |
| Clk. freq. (MHz) | | $V_1$ | **-14.023** | 0.376 | -0.220 | 0.464 | -2.013 | 2.449 | -1.562 | -0.486 | **-5.531** | 122.236 | 0.799 | 0.910 |
| Dyn. power (mW) | | $V_2$ | -3.512 | 0.599 | 0.538 | 0.162 | -4.223 | 0.481 | -3.713 | -1.499 | **-8.128** | 147.133 | 0.516 | 0.840 |
| Flip-Flops | | $V_3$ | **-18.625** | 0.250 | -0.750 | 0.500 | **72.125** | -0.250 | -4.375 | **-43.125** | -0.125 | 3636.375 | 0.921 | - |
| LUTs | | $V_4$ | **-336.688** | -62.563 | -20.813 | -57.188 | -71.188 | -5.938 | 17.938 | -50.563 | **-722.938** | 6827.875 | 0.944 | - |
| | M | $V_5$ | **0.016** | 0.000 | 0.001 | -0.001 | -0.001 | 0.000 | -0.001 | **-0.045** | 0.000 | 10.248 | 0.983 | - |
| Flip-Flops | L | $V_6$ | **-0.086** | 0.000 | 0.004 | -0.004 | -0.004 | 0.000 | -0.004 | **-0.400** | 0.000 | 27.890 | 0.999 | - |
| Stuck-at-1 | S | $V_7$ | **0.106** | 0.068 | 0.026 | 0.004 | 0.064 | 0.023 | -0.026 | 0.053 | 0.008 | 20.740 | 0.466 | 0.994 |
| | SDC | $V_8$ | -0.041 | -0.068 | -0.031 | 0.001 | -0.059 | -0.023 | 0.031 | **0.398** | -0.008 | 41.120 | 0.861 | 0.961 |
| | M | $V_9$ | **0.036** | 0.001 | -0.009 | 0.009 | 0.009 | -0.001 | 0.009 | **-0.326** | 0.001 | 38.023 | 0.992 | - |
| Flip-Flops | L | $V_{10}$ | **0.017** | 0.001 | 0.002 | -0.002 | -0.002 | -0.001 | -0.002 | **-0.026** | 0.001 | 13.113 | 0.942 | - |
| Stuck-at-0 | S | $V_{11}$ | **-0.259** | 0.030 | 0.016 | 0.014 | -0.001 | 0.015 | -0.016 | 0.060 | 0.000 | 9.353 | 0.794 | 0.998 |
| | SDC | $V_{12}$ | **0.201** | -0.030 | -0.009 | -0.021 | -0.006 | -0.015 | 0.009 | **0.285** | 0.000 | 39.520 | 0.869 | 0.906 |
| | M | $V_{13}$ | 0.166 | 0.146 | 0.239 | 0.364 | -0.391 | -0.245 | -0.055 | -0.182 | -0.175 | 75.949 | 0.318 | 0.871 |
| Flip-Flops | L | $V_{14}$ | -0.039 | 0.025 | 0.001 | -0.016 | 0.039 | 0.005 | -0.011 | 0.045 | 0.010 | 7.382 | 0.300 | 0.997 |
| Bit-Flip | S | $V_{15}$ | -0.111 | 0.024 | -0.056 | -0.064 | 0.119 | 0.044 | 0.029 | 0.009 | 0.021 | 3.737 | 0.348 | 0.703 |
| | SDC | $V_{16}$ | -0.013 | -0.197 | -0.184 | -0.282 | 0.233 | 0.194 | 0.038 | 0.131 | 0.146 | 12.926 | 0.337 | 0.906 |
| | M | $V_{17}$ | **-2.133** | -0.264 | 0.019 | 0.015 | **0.671** | -0.078 | -0.044 | -0.014 | 0.030 | 21.456 | 0.921 | - |
| LUTs | L | $V_{18}$ | **0.425** | 0.134 | -0.053 | 0.036 | **-0.314** | 0.111 | -0.030 | **-0.221** | 0.085 | 6.078 | 0.781 | 0.911 |
| Stuck-at-1 | S | $V_{19}$ | **0.470** | **0.310** | 0.016 | -0.065 | **0.429** | -0.060 | 0.033 | 0.084 | -0.013 | 19.183 | 0.625 | 0.972 |
| | SDC | $V_{20}$ | **1.238** | -0.177 | 0.016 | 0.016 | **-0.786** | 0.027 | 0.039 | 0.154 | -0.103 | 53.282 | 0.827 | 0.992 |
| | M | $V_{21}$ | **-1.878** | **-0.312** | -0.027 | **0.327** | 0.273 | -0.056 | 0.003 | -0.169 | 0.154 | 40.356 | 0.894 | 0.970 |
| LUTs | L | $V_{22}$ | 0.016 | -0.034 | -0.007 | **0.121** | -0.066 | 0.076 | 0.021 | **0.103** | 0.011 | 4.425 | 0.438 | 0.839 |
| Stuck-at-0 | S | $V_{23}$ | -0.064 | 0.052 | -0.013 | 0.129 | 0.191 | 0.029 | 0.041 | -0.054 | -0.064 | 6.971 | 0.266 | 0.942 |
| | SDC | $V_{24}$ | **1.927** | 0.293 | 0.047 | **-0.577** | -0.398 | -0.047 | -0.066 | 0.123 | -0.098 | 48.244 | 0.821 | 0.982 |

*Failure modes: M - Masked fault, L - Latent fault, S - Signalled fault, SDC - Silent Data Corruption*

*GL - Generalized linear regression*

power consumed at all. The *Keep Hierarchy* ($X_4$) option was supposed to negatively impact the final implementation, but it seems this is not the case, and it even slightly improves the robustness properties for stuck-at-0 at LUTs (reduces the rate of silent data corruption $V_{24}$ and improves fault masking $V_{21}$). Both *Resource Sharing* ($X_6$) and *Register Duplication* ($X_7$) have no noticeable impact either, but this could be more easily attributed to the particular architecture of the selected target processor.

Likewise, it is interesting to note that only two factors affected the maximum clock frequency of the implementation: *Optimization Goal* ($X_1$) (which was expected) and *LUT combining* ($X_9$). The dynamic power consumption is surprisingly only affected by *LUT combining* ($X_9$). These two synthesis options ($X_1$ and $X_9$) also significantly impact the total number of LUTs used to imple-

ment the design, which seems quite logical in this case. *Safe Implementation* ($X_5$) with *Optimization Goal* ($X_1$) and *Equivalent Register Removal* ($X_8$), also impact the final number of FFs.

No factor seems to impact significantly the robustness when considering the occurrence of bit-flips. This could either mean that some other synthesis flags excluded from this case study may have an influence on this response variable, or that due to the particular architecture of the selected processor no synthesis flag can really affect it.

On the other hand, Table 5 lists the whole set of estimators and intercept computed for each response variable. Those estimators define a multiple linear regression model in which the relationship between the response variable and the factors is linear ($y = Intercept + \sum_{i=1}^{9} \beta_i X_i$). For some response variables the initially computed linear model provided a poor fit ($R^2 < 0.5$), whereas for some others it was insufficiently accurate ($R^2 < 0.9$) to precisely predict the response variable. In these case, the generalized linear model has been computed, taking into account interactions of factors, which fitted the data really well for all configurations ($R^2$ ranges between 0.839 and 0.998). Resulting regression models (linear or generalized linear) statistically significantly predict the response variables for considered factors, meeting the required p-value $<$ 0.05. It must be noted that the produced generalized linear models used between 2 and 34 terms, that is why the whole set of estimators and equations have not been included in this paper. Just as an example, the linear regression model for estimating the number of FFs is shown in Equation3 and the generalized linear regression model for estimating the clock frequency is shown in Equation 4.

$$\begin{aligned} \text{FFs} = 3636.375 &- 18.625X_1 + 0.25X_2 - 0.75X_3 + 0.5X_4 \\ &+ 72.125X_5 - 0.25X_6 - 4.375X_7 - 43.125X_8 - 0.125X_9 \end{aligned} \quad (3)$$

$$\text{Clock Frequency} = 123.82 + 2.4487X_6 - 5.5307X_9 - 6.5274X_1X_4 + 6.7499X_1X_5 + 6.7684X_5X_7 \quad (4)$$

Using the obtained regression models the expected values of the response variables has been computed for the whole set of $2^9$ possible synthesis configurations. This information is very useful to quickly determine the best synthesis options configuration to meet a given implementation goal. When considering several implementation goals, a multi-objective optimization may be conducted, identifying a Pareto optimal solutions (improving a certain goal is only possible at the expense of other goals). Figure 5 represents a Pareto frontier optimizing two response variables: dynamic power consumption and clock frequency. The dots represent the 512 configurations and the connected dots indicate the best trade off between minimizing the power consumption and maximizing the clock frequency. However, it does not explain which of those solutions should be selected, as there is no indication about the designer's preferences. Likewise, when optimizing response variables $V_1 - V_4$, Pareto optimality provides a total of 182 possible solutions, and since all of them are equally good in the absence of clearly defined preferences, the designer is at a loss when making a decision.

MCDM techniques in general, and the weighted sum method (WSM) used in this case study in particular, can be used to define how the estimated value for each response variable should be aggregated according to provided preferences. In such a way a single score could be obtained to determine the best possible synthesis flags configuration to optimize the given implementation goals.

Table 6 lists how weights have been distributed among response variables according to different implementation goals. These goals include maximizing one of the variables, combining all but giving a higher priority to one of them, or equally distributing the weights among all of them for a balanced design. This goals could be understood as sample profiles in absence of actual requirements provided by a designer. Each designer should adjust those weights according to her particular preferences. It must be noted that utilization has been con-

sidered to consist of both the number of FFs and LUTS, so they share the utilization weight. Response variables representing the same failure mode for different fault models and targets have been considered equally important (with no priority for any of those variables). Accordingly, in Table 6 each failure mode is listed as an aggregate value of five corresponding response variables. For instance, the weight for the rate of masked faults $\omega_5$ is divided between variables $V_5, V_9, V_{13}, V_{17}$ and $V_{21}$ in equal parts.

Table 6: WSM scores for the best, default, and worst configurations according to different implementation goals.

| | **Weights** | | | | | | | | | | | | | Quality of |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | **CF** | **DP** | **FFs** | **LUTs** | **M** | **L** | **S** | **SDC** | Best | Best | Worst | Worst | Default | default |
| **Goal** | $\omega_1$ | $\omega_2$ | $\omega_3$ | $\omega_4$ | $\omega_5$ | $\omega_6$ | $\omega_7$ | $\omega_8$ | configuration | score | configuration | score | score | score |
| Speed | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | {000101000} | 1.0 | {100100101} | 0.7416 | 0.9637 | 85% |
| Consumption | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | {000010101} | 1.0 | {000000000} | 0.7982 | 0.8596 | 30% |
| Utilization | 0 | 0 | 1/2 | 1/2 | 0 | 0 | 0 | 0 | {111101011} | 0.9929 | {000010100} | 0.8877 | 0.9219 | 32% |
| Performance | 1/3 | 1/3 | 1/6 | 1/6 | 0 | 0 | 0 | 0 | {010110111} | 0.9467 | {100100000} | 0.8614 | 0.9151 | 62% |
| Just M | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | {001111101} | 0.9953 | {110001110} | 0.9462 | 0.9854 | 79% |
| Just L | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | {000011110} | 0.9848 | {100101001} | 0.9360 | 0.9639 | 57% |
| Just S | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | {010011000} | 0.9755 | {101000001} | 0.9152 | 0.9285 | 22% |
| Just SDC | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | {010101000} | 0.9855 | {100001111} | 0.9368 | 0.9669 | 61% |
| Availability | 0 | 0 | 0 | 0 | 1/2 | 1/2 | 0 | 0 | {000111110} | 0.9868 | {110101011} | 0.9448 | 0.9747 | 71% |
| Safety | 0 | 0 | 0 | 0 | 1/2 | 0 | 1/2 | 0 | {010110000} | 0.9796 | {111101001} | 0.9354 | 0.9569 | 48% |
| Balanced Safe | 1/6 | 1/6 | 1/12 | 1/12 | 1/4 | 0 | 1/4 | 0 | {010110111} | 0.9592 | {100100000} | 0.9008 | 0.9360 | 60% |

*CF - Clock Frequency, DP - Dynamic Power*

*Failure modes: M - Masked fault, L - Latent fault, S - Signalled fault, SDC - Silent Data Corruption*

For each one of the defined goals, the best and worst configurations maximizing and minimizing, respectively, the score under the WSM are also listed. For instance, to maximize the clock frequency it is required to set a high level for *Keep Hierarchy* ($X_4$) and *Resource Sharing* ($X_6$), while keeping all other options at low level, and to minimize utilization all synthesis options should be set at high level, except for *Safe Implementation* ($X_5$) and *Register duplication* ($X_7$).

Likewise, the score computed for the *default* configuration of Xilinx's XST tool is also provided, including its quality with respect to the whole range of scores for the 512 possible configurations. It is worth commenting that this default configuration seems to be clearly oriented towards achieving the highest possible clock frequency, and it also provides relatively high results in fault

masking and availability. Nevertheless, this default configuration behaves very poorly in terms of utilization and power consumption, as its score is in the lowest 32% and 30%, respectively, for all possible configurations. It also provides poor result in terms of signalling failures (22%), which also negatively affects the safety (48%). Accordingly, designers focusing on improving the device utilization, power consumption and safety should not rely on the options provided by default.

## 6. Conclusions

In spite of the critical importance of synthesis options towards meeting the defined implementation goals, designers are usually at a loss when trying to determine which options, from the myriad available, have any impact on their target designs. Due to poor documentation, and the enormous amount of time required to explore the whole design space, only a small fraction of all these options are actually used.

This work addresses this problem by defining a method for determining the best possible configuration of synthesis options to meet a given set of goals. First of all, the design space is reduced by defining a fractional factorial design, making actually feasible the experimentation for a large set of synthesis flags. The execution of these experiments comprises three consecutive phases: i) implementation of designs according to defined configurations, providing the estimation of static properties such as clock frequency and utilization, ii) simulation of implemented designs to estimate the dynamic properties, like dynamic power consumption, and iii) simulation-based fault injection followed by sensitivity analysis to estimate the robustness properties. The analysis of variance processes the responses of those experiments to determine whether each synthesis flag statistically significantly impacts the given implementation goals. By means of multiple regression analysis the expected results are predicted for any combination of the synthesis flags across the whole design space. Finally, multiple-criteria decision making techniques are exploited to select the best pos-

sible configuration according to specifically stated implementation goals. The implementation of the proposed method targeting Xilinx ISE Design Suite, has been fully automated, taking advantage of parallel processing on PC and Grid-based computing system to speed-up the whole experimental process.

Main results obtained, when taking the LEON3 processor as a case study, show that only six out of nine considered options statistically significant impact the defined implementation goals: five options impacting the robustness, four options impacting utilization, and just a single option has a significant impact on clock frequency and power consumption. This could point to the existence of a large set of synthesis options that are only useful for very particular cases. Furthermore, the default options for Xilinx's XST tool are clearly defined to achieve the highest possible clock frequency, at the expense of getting a poor device utilization, power consumption and safety. The conducted study also supports the hypothesis that the robustness can be improved by just properly configuring the synthesis tool.

Although the LEON3 processor could be considered representative of embedded microprocessors for critical-systems, our future work focuses on extending this analysis to target a whole set of benchmark designs considered representative of different types of circuits. In such a way, drawn conclusion could be generalized instead of being dependent on the considered case study. Likewise, this analysis could be further extended to determine the actual impact of the whole set of options from both synthesis and the rest of implementation processes (map, place, and route). What is more, beyond determining the precise contributing of each option towards the implementation goals, this procedure could also be used to locate hidden or unknown interactions between options belonging to different processes (like a given synthesis option interacting with a routing option) and/or tools (particular interactions between tools from different vendors which may affect the final implementation).

**Acknowledgements**

This work has been partially funded by the Ministerio de Economía, Industria y Competitividad de España under grant agreement no TIN2016-81075-R, and the "Programa de Ayudas de Investigación y Desarrollo" (PAID) de la Universitat Politècnica de València.

**References**

[1] W. Wolf, FPGA-Based System Design, Prentice Hall, 2004.

[2] S. Hauck, A. DeHon, Reconfigurable Computing: The Theory and Practice of FPGA-Based Computation, Morgan Kaufmann Publishers, 2008.

[3] S. Kilts, Advanced FPGA Design: Architecture, Implementation, and Optimization, Wiley-IEEE Press, 2007.

[4] Xilinx Inc., Command Line Tools User Guide (2013).
URL `http://www.xilinx.com/support/documentation/sw_manuals/xilinx14_7/devref.pdf`

[5] Altera Corp., Quartus Prime Pro Edition Handbook Volume 2: Design Implementation and Optimization (2015).
URL `https://www.altera.com/en_US/pdfs/literature/hb/qts/qts-qpp-5v2.pdf`

[6] Q. Guo, T. Chen, Y. Chen, L. Li, W. Hua, Microarchitectural design space exploration made fast, Microprocessors and Microsystems 37 (2013) 41–51.

[7] D. Genbrugge, L. Eeckhout, Chip multiprocessor design space exploration through statistical simulation, IEEE Transactions on Computers 58 (12) (2009) 1668–1681.

[8] C. Pilato, D. Loiacono, A. T. andFabrizio Ferrandi, P. L. Lanzi, D. Sciuto, Computational Intelligence in Expensive Optimization Problems, Springer-Verlag Berlin Heidelberg, 2010, Ch. Speeding-Up Expensive Evaluations in

High-Level Synthesis Using Solution Modeling and Fitness Inheritance, pp. 701–723.

[9] D. Sheldon, R. Kumar, R. Lysecky, F. Vahid, D. Tullsen, Application-Specific Customization of Parameterized FPGA Soft-Core Processors, in: IEEE/ACM International Conference on Computer Aided Design, 2006, pp. 261–268.

[10] M. Kurek, T. Becker, T. C. Chau, W. Luk, Automating Optimization of Reconfigurable Designs, in: IEEE 22nd International Symposium on Field-Programmable Custom Computing Machines, 2014, pp. 210–213.

[11] A. Mametjanov, P. Balaprakash, C. Choudary, P. D. Hovland, S. M. Wild, G. Sabin, Autotuning FPGA Design Parameters for Performance and Power, in: 23rd IEEE International Symposium on Field-Programmable Custom Computing Machines, 2015, pp. 84–91.

[12] D. Meidanis, K. Georgopoulos, I. Papaefstathiou, FPGA Power Consumption Measurements and Estimations Under Different Implementation Parameters, in: International Conference on Field-Programmable Technology, 2011, pp. 1–6.

[13] Cobham Gaisler AB, Leon3 processor product sheet (2016).
URL `http://www.gaisler.com/doc/leon3_product_sheet.pdf`

[14] NIST/SEMATECH, NIST/SEMATECH e-Handbook of Statistical Methods (2013).
URL `http://www.itl.nist.gov/div898/handbook/`

[15] C. Wu, M. Hamada, Experiments: Planning, Analysis, and Optimization, Wiley Series in Probability and Statistics, Wiley, 2011.

[16] K. Fang, R. Li, A. Sudjianto, Design and Modeling for Computer Experiments, Chapman & Hall/CRC Computer Science & Data Analysis, CRC Press, 2005.

[17] J. M. Juran, J. A. D. Feo, Juran's Quality Handbook, McGraw-Hill Education, 2010.

[18] D. C. Montgomery, Design and Analysis of Experiments, 9th Edition, John Wiley & Sons, New York, 2017.

[19] R. Fontana, S. Sampò, Minimum-Size Mixed-Level Orthogonal Fractional Factorial Designs Generation: A SAS-Based Algorithm, Journal of Statistical Software 53 (10) (2013) 1–58.

[20] D. A. Freedman, Statistical Models: Theory and Practice, Cambridge University Press, 2009.

[21] A. Ishizaka, P. Nemery, Multi-criteria Decision Analysis: Methods and Software, Wiley, 2013.

[22] E. Triantaphyllou, Multi-Criteria Decision Making Methods, in: Multi-criteria Decision Making Methods: A Comparative Study, Vol. 44 of Applied Optimization, Springer US, 2000, pp. 5–21.

[23] T. Saaty, Decision making with the analytic hierarchy process, International Journal of Services Sciences 1 (1) (2008) 83–98.

[24] S. Opricovic, G.-H. Tzeng, Extended VIKOR method in comparison with outranking methods, European Journal of Operational Research 178 (2) (2007) 514–529.

[25] K. Yoon, A reconciliation among discrete compromise solutions, Journal of the Operational Research Society (1987) 277–286.

[26] Xilinx Inc., Synthesis and Simulation Design Guide (v 14.4) (2012).
URL `http://www.xilinx.com/support/documentation/sw_manuals/xilinx14_7/sim.pdf`

[27] G. E. P. Box, J. S. Hunter, W. G. Hunter, Statistics for experimenters: design, innovation, and discovery, Wiley series in probability and statistics, Wiley-Interscience, Hoboken (N.J.), 2005.

[28] The MathWorks, Inc., Statistics and Machine Learning Toolbox™User's Guide (2016).
URL https://es.mathworks.com/help/pdf_doc/stats/stats.pdf

[29] A. Benso, P. Prinetto, in: Fault Injection Techniques and Tools for VLSI reliability evaluation, Frontiers In Electronic Testing, Kluwer Academic Publishers, 2003, Ch. VHDL Simulation-Based Fault Injection Techniques, pp. 159–176.

[30] Mentor Graphics, ModelSim (2016).
URL https://www.mentor.com/products/fv/modelsim/

[31] Xilinx Inc., XST User Guide for Virtex-6, Spartan-6, and 7 Series Devices (v 14.5) (2013).
URL http://www.xilinx.com/support/documentation/sw_manuals/xilinx14_7/xst_v6s6.pdf

[32] Xilinx Inc., Timing Closure User Guide (v 14.3) (2013).
URL http://www.xilinx.com/support/documentation/sw_manuals/xilinx14_7/ug612.pdf

[33] Xilinx Inc., Power Methodology Guide (v 14.5) (2013).
URL http://www.xilinx.com/support/documentation/sw_manuals/xilinx14_7/ug786_PowerMethodology.pdf

[34] M. R. Guthaus, J. S. Ringenberg, D. Ernst, T. M. Austin, T. Mudge, R. B. Brown, MiBench: A free, commercially representative embedded benchmark suite, in: IEEE 4th Annual Workshop on Workload Characterization, 2001, pp. 3–14.

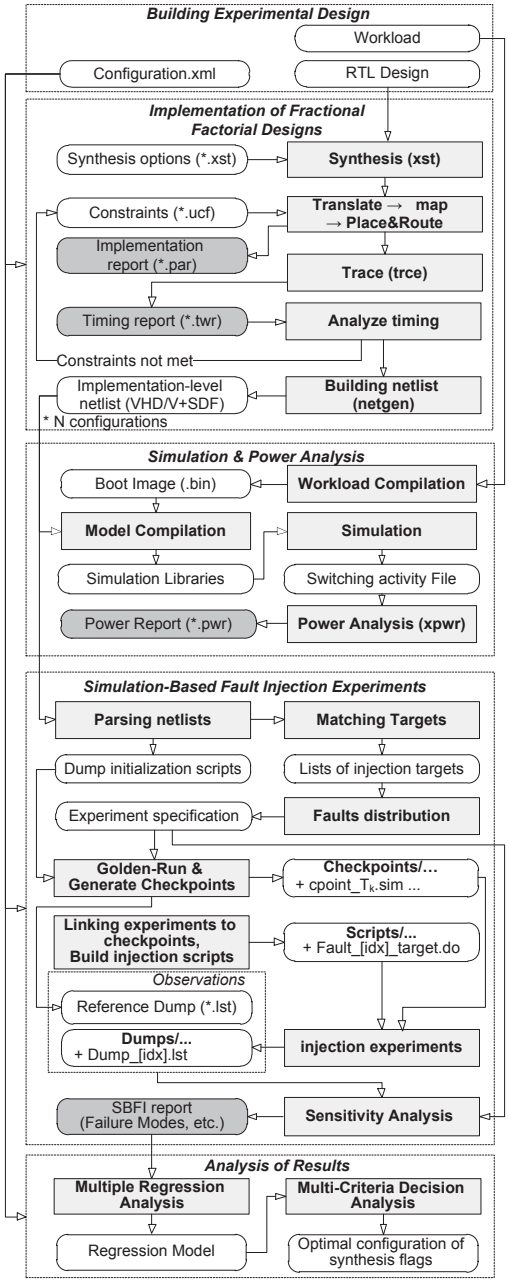


Figure 3: Flowchart of the proposed approach implemented for Xilinx ISE Design Suite

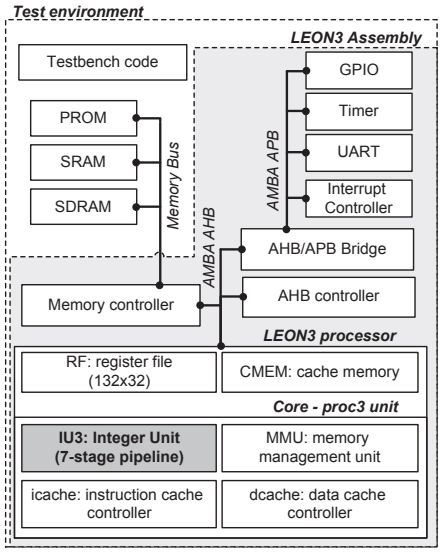


Figure 4: LEON3 target configuration and test environment.

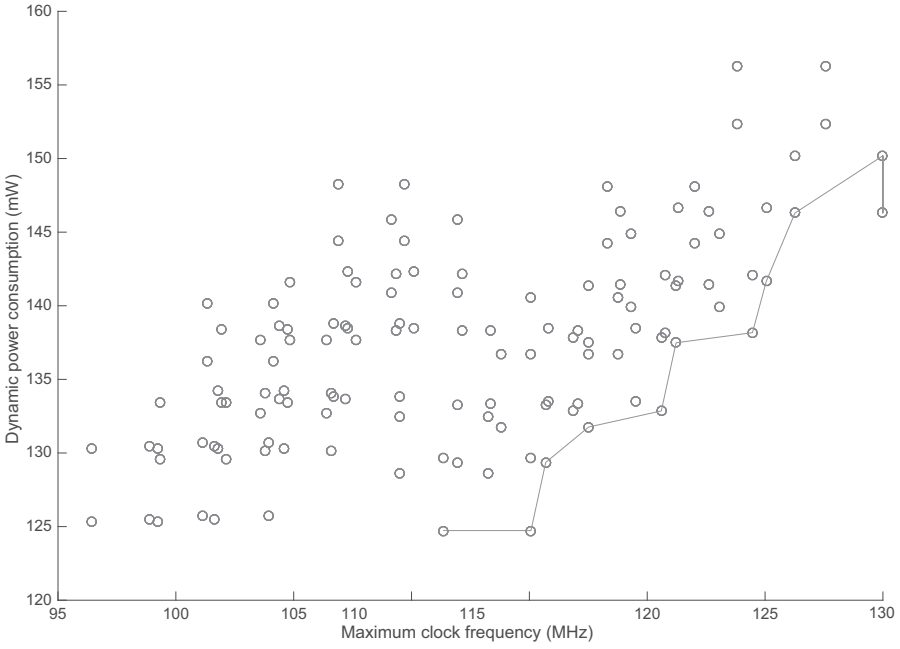


Figure 5: Pareto frontier indicating optimal solutions for minimizing dynamic power consumption and maximizing clock frequency