EUROPEAN SPACE AGENCY

MICROELECTRONICS SECTION

Final Report

YGT Final Report

Atmel FPGA

Filomena Decuzzi

YEAR 2009-2010

Summary

Field Programmable Gate Arrays (FPGAs) are becoming more and more interesting in space and avionic applications, where reconfigurability, high performance and low-power consumption can be fruitfully used to develop innovative systems. FPGAs have the capability of being reconfigured several times.
The high flexibility combined with high performance makes the FPGA a valid solution for several applications. However, missions take place in a harsh environment, rich in radiation, which can induce errors within electronic devices. The radiation may interact with silicon devices and induce Single Event Effects (SEEs), like Single Event Upset (SEUs) and Single Event Transients (SETs) in the configuration memory and user memory.
The effects of the SEEs depended on the technology used. Nowadays, re-programmable FPGAs can be SRAM-based FPGAs, when the information defining the configuration of the device (i.e. the configuration memory) is stored using SRAM memory cells, or Flash-based FPGAs, when the configuration memory is implemented resorting to floating-gate memory cells. The SRAM is the technology used for implementing volatile configuration memory. Such memories - in the commercial field - are sensitive to SEU (SEUs) induced by radiation. In SRAM-based FPGAs both the configuration and the user memory are sensitive to SEUs.
Since space applications have to survive to environments rich in radiations, silicon vendors proposed different solutions. Actel adopted floating gate cell based on a 130nm process that radiation testing has showed to be insensitive to heavy ions [1]. Xilinx adopt Commercial-Of -The-Shelf (COTS) devices with proper SEE mitigation techniques as redundancy and memory scrubbing [2]. Atmel developed Radiation Hardened By Design (RHBD) configuration memory cells and flip-flop that are less sensitive to ionizing radiation.
The Atmel FPGA have been studied, for the whole project, in different aspects. As far as RHBD Atmel devices are considered, although robust against SEEs, they still have a not null cross-section. In order to use them successfully in safety-critical application, designers must guarantee that the obtained system satisfies the dependability level requested for their application. As a matter of fact, designs make

seldom use of the whole configuration memory bits; only a portion of the configuration memory is used to program the device. As result, in order to predict the robustness of FPGA, a design-based analysis is desirable.

The results of this design depended analysis shows how the reliability of a design depend on :

- design description;
- design implementation.

The design description depends on the designer and it is not so easy to change or to drive with rules oriented to the reliability. Moreover, the description of the design is synthesized by commercial tool whose the behavior is heuristic and not optimizing for dependability.
On the other hand, the design implementation is done by the place and route tool. If the commercial place and route tool allows the use of constrains, a correct setting can lead to a better reliability level. So the study of techniques to improve the reliability of the design has led to focus on the place and route strategies. In the investigation, three aspects are taken into account:

- use of capability of the logic architecture;
- placement strategy;
- routing strategy.

Two studies are performed to address the above mentinated aspects. A first study shows how the appropriate use of the logic architecture leads to an improvement in terms of area and reliability. The second study shows the high importance of the placement strategy on the following routing step and on the global performance of the design.

For the first study, according to other users reports, an anomaly in the use of the capability of the logic architecture was detected. It was found that the capability of the logic resources were not completely exploited. The study shows how this leak has a grave impact in terms of area and reliability of the design. The results of this study induced the Atmel company to take measures on this point and now the problem results already fixed in the last release of the Precision RTL synthesis tool.

Finally, the placement strategy was investigated. Place and Route strategy is a mandatory to address later the problem of dependability. Investigating on this strategy, some other more fundamental problems are shown up. The use of the placement and routing Figaro has turned out that the aspects of timing can be improved from the automatic solution given by Figaro. Several users feedbacks [4] [**?**] [**?**] had led the priority of the study on the timing performance of the system.

Since the commercial place and route tool Figaro does not support any kind of external constrains for the logic placement, the task was oriented to develop a platform to use another placement algorithm in place that the one implemented in Figaro. The Platform to Upgrade and Redo Placement Leading Efficiency (PURPLE) was developed to be joined Figaro. This platform allows the use of several independent placement algorithms on the Atmel architecture. PURPLE has the capability to give the netlist description of the circuit under test, contains a placement engine and converts the placement solution given by the engine in the proper format readable by Figaro. In this way PURPLE hooks on Figaro, creating a parallel implementation flow.
The last step provides PURPLE with a placement algorithm. After the assessment of the time needed to develop a technology-dependent algorithm, the use of an already coded algorithm is evaluated as the best solution. The research was oriented to a timing-driven algorithm, technology independent and open source tool. The Versatile Place and Route (VPR) FPGA CAD tool meets the requirements [3]. After a deep work to set the tool for the Atmel architecture the results show that the route capability for the solution is the most critical limitation of the tool. Although VPR is proposed as a generic tool for different FPGA architectures, the target architectures used for the placement algorithm development are more complex than the Atmel one, so they do not present some limitations as in our case. In fact the idea of VPR is based on Altera and Xilinx architectures that are more complicated than the Atmel one. In particular, they contain complex routing elements that can route large implementation on the device. For these reasons some architectural limitations are not taken into account in the VPR algorithm, this leads to have some solution for the Atmel devices that present the problem of *congestion*.

In this final report the the three main tasks carried out on the Atmel FPGA devices are presented:

- The extension of static analysis tool with debug capabilities –Susanna and Jonathan –.
- The analysis of utilization of logic architecture capabilities –Use of Macro in Atmel architecture –.
- The development of a platform to apply a new placement algorithm to the implementation flow –PURPLE–.

It is organized as follows. Each task is presents in one chapter. For each chapter, Section 1 presents the introduction to the task. Section 2 gives the motivations of the task. In the Section 3 the developed study is presented, while the experimental results are described in section 4. Finally, the conclusion are depicted in Section 6.

Contents

List of Tables

List of Figures

Chapter 1

Susanna and Jonathan

1.1 Introduction

Due to the many advantages of the reconfigurability of SRAM-based FPGAs, their use is increasing even in systems requiring a high level of dependability (safety, availability, security, etc.). Moreover the missions being a collaboration between several space agencies exportation rules must be fulfilled. In particular these rules - International Traffic in Arms Regulation (ITAR) - restrict drastically component availability. This is especially critical for FPGAs since most of the manufacturers are in US. Under this conditions the Atmel FPGA are becoming more and more interesting for the European market.

The main issue for such systems is their working conditions: they often have to operate under harsh environment, such as ionizing radiations, or they may have to resist to voluntary fault-based attacks, creating similar perturbations by using for example a laser. Single-event effects (SEEs) induced by the interaction of particles with integrated circuits are a well-known threat for space systems, which are directly exposed to cosmic rays. With the shrinking of the transistor sizes in modern technologies, systems are also sensitive to atmospheric particles at sea-level.

The most probable effect, when we consider SRAM-based FPGA at sea-level, is the single-event upset (SEU), i.e. a bit-flip in the embedded memories [7]. Faults in the configuration memory of a SRAM-based FPGA directly modify the definition of its function, dangerously impacting its ability to operate properly [8]. These errors usually last until the configuration memory is refreshed. Moreover, detecting and/or correcting these errors induce, in most cases, a high cost, that can still increase if multiple-bit upsets (MBUs) must also be considered. Protecting the system against faults in the configuration memory is an important issue at design time. Several design-level solutions exist to develop fault-tolerant architectures from SRAM-based FPGAs. An example using the triple modular redundancy (TMR) technique is given

in [9]. In all cases, the designer has to make a compromise between cost (area, power and performance overheads) and fault-tolerance.

At design time, the evaluation of the effects of faults in the device and the choice of the best protection strategy require realistic fault models. To achieve this, it is necessary to use results from actual fault injections on a test device to develop the fault models. The better the models are, the more accurate the evaluation of the dependability will be. An on-going collaborative effort has allowed us to develop a software tools and associated methodologies for performing fault injections in Atmel FPGA devices [10], see Annex A. The software for the static analysis was developed in collaboration with Politecnico di Torino and extended during the YGT project at ESA. The static analysis tool is named Susanna, while the extended version done in ESA is called Jonathan.

1.2 Motivation

Modern FPGAs have been designed with advanced integrated circuit techniques that allow high speed performance, joined to reconfiguration capabilities. This makes new FPGA devices very advantageous for space and avionics computing. However critical environments makes FPGA's configuration memory a critical part of the system. Different solutions are taken into account from the different vendors. The Atmel company developed radiation hardened SRAM-based reprogrammable FPGAs. It has been especially designed for space application by implementing hardened cells and permanent self integrity check mechanism [11].

Anyway RHBD devices, although robust against SEEs, they still have a not-null cross section. The most recent product, the ATF280E has a susceptibility threshold (LETth) of 30 MeVcm2/mg at Vcc min and saturation cross-section of $3\text{x}10^{-9}$ bit^{-1}. As a result, designers must take into account the contribution of ionizing radiation on such devices when evaluating the robustness of FPGA–system [10].

The the impact of radiation is evaluating on the configuration memory. The designers do not have visibility on the low-level information stored in the configuration memory. For this reason, making handling the information about the sensitiveness of the system for the designer, becomes a basic necessity. The analysis results can lead the strategy for fail-safe design. In order to have a correlation between each single bit of the configuration memory and the design instance it belongs to, Susanna is enhance with design description reading capability. This makes the reliability data correlated to the instances of the design. This feature, given by Jonathan, offers to the designer a set of statistics in order to detect the most critical parts of the design under test.

1.3 Proposed Approach:An Overview

1.3.1 Background terminology: netlist elements

In order to give a general view and a referment for the terminology used lately, this section describe the design database according to the Precision Synthesis Reference Manual [12]. In a synthesis tool the elements in the design database are:

- *Library.* All design and technology information resides in a library. A library can contain design data, technology cells, or primitive cell. The primitive cell library contains a generic set of combinational and sequential logic cells that the synthesis tool uses to represent Hardware Description Language (HDL) as gate-level networks.

- *Cells.* A cell represents either technology information or levels of design hierarchy. A set of cell create a library. A cell can be composed by different views.

- *Views.* A view contains interface information and might also contain a netlist.

In analogy to VHDL, a *cell* is equivalent to an *ENTITY* and a *view* is equivalent to an *architecture*. In summary, the following objects are typically contained within a view and are used to represent netlists and hierarchies in a design:

- A *view* has ports, nets and instances.

- A *port* is a terminal of a view.

- An *instance* is a pointer to a view.

- A *net* is a connection between ports and/or port instances (pointer to the port of the view under an instance).

- An *elementary instance* is a leaf instance that does not contain hierachy.

1.3.2 Susanna

Susanna is a static analysis too aimed at evaluating the sensitiveness of designs implemented on RHBD FPGAs from Atmel. The tool analyzes designs implemented on such device to identify the portion of the configuration memory that is sensitive, i.e., all those configuration memory bits that, if affected by soft errors, result in modifications to the FPGA resource that lead to application failure. Thanks to this tool a precise design dependent dependability analysis can be performed, and

alternative design solutions can be evaluated easily. To have more details about the algorithm see Annex B.

The report of the static analysis tool contains a list of all the critical bits. From the configuration memory, the configuration bits for the logical (Core Cell) and routing (Repeater) physical resources are taken into account; while the configuration bits for the I/O and the RAM resources are considered out of scope. For each configuration bit classified as critical are showed the address in the configuration memory and the physical resource it programs, see Figure 1.3. The information are reported as follow:

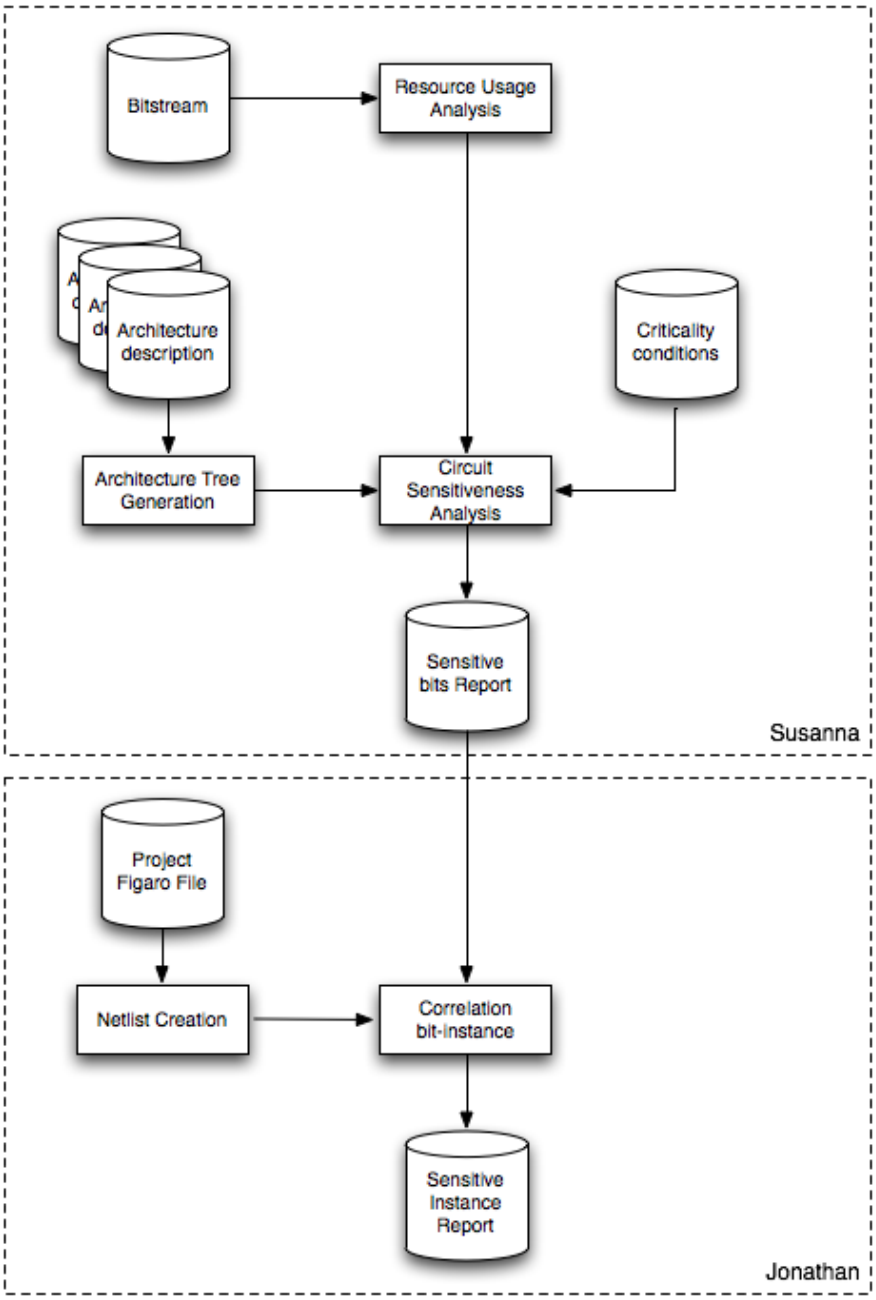
1. Byte address: byte address in the configuration memory.
2. Bit number: bit of the byte address.
3. Bit value: bit value of the critical bit under test.
4. Resource: type and coordinate of the physical resource that the bit under test belongs to. The type can be Core Cell (CC) or Horizontal Repeater (HR) or Vertical Repeater (VR). The coordinate detect the position of the resource in the matrix representation for island-style FPGA.

1.3.3 Jonathan

The tool Jonathan is an enhancement part of the static analysis tool Susanna, see Figure 1.1. Using the combination of Susanna and Jonathan the whole design flow is investigated. From the RTL description to the bitstream data, the circuit implementation is explored and correlated.

While Susanna works on the last step of the implementation flow, where the circuit information are ready to be stored in the configuration memory of the device, Jonathan has got the visibility of all the different descriptions of the design during the several stages of design flow, see Figure 1.2. Starting from an RTL description of a design, the synthesis tool constructs a corresponding network of gates in a given technology. The Place and Route tool provide to map these gates on the real architecture of the device and place and route the resources used to implement the circuit. Finally, the Place and Route Tool generates the bitstream in order to program the configuration memory of the device. After the analysis of the configuration memory of the design, the criticalities are lead back across the design flow. Jonathan adds design level details to the information in the report of Susanna. As showed in Figure 1.3 the information about the part of the design programmed by the critical bit are added to the configuration memory and physical resource info. This design information are showed by all the hierarchy of the design, from the most external instance to the elements implemented in the physical resource programmed by the

Figure 1.1. Susanna and Jonathan flow



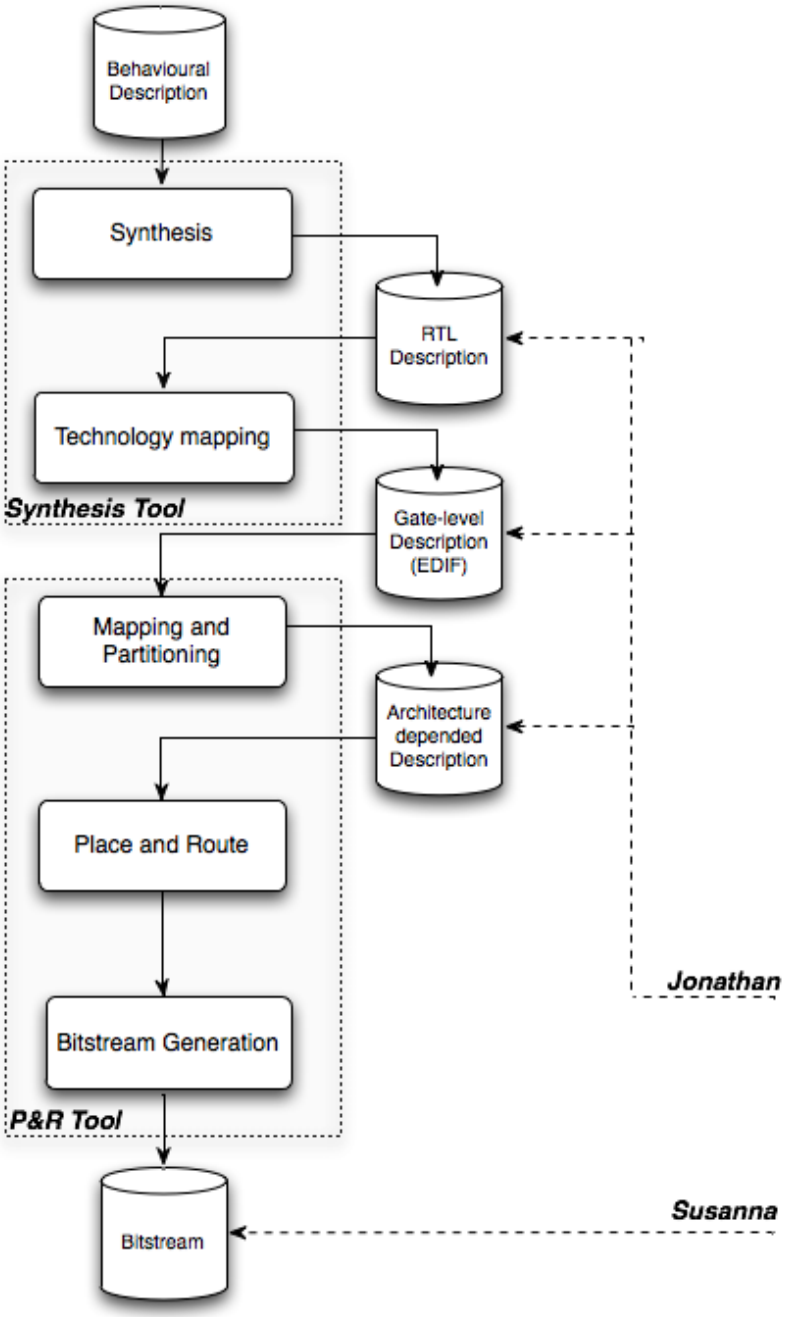


Figure 1.2. Design Implementation flow

bit. The information cover all the steps of the implementation, from the RTL design model, given by the synthesis tool Precision, to the mapping information given by the place and route tool Figaro.

The information are reported as follow:

1. Hierarchical Design Instance: all the hierarchical path from the most external *instance* to the most internal hierarchical *instance*, the hierarchy follows the RTL model of the design.
2. Map instance: name of the physical resource after the mapping.
3. Design instance: name of the RTL *elementary instance* contained in the physical resource.

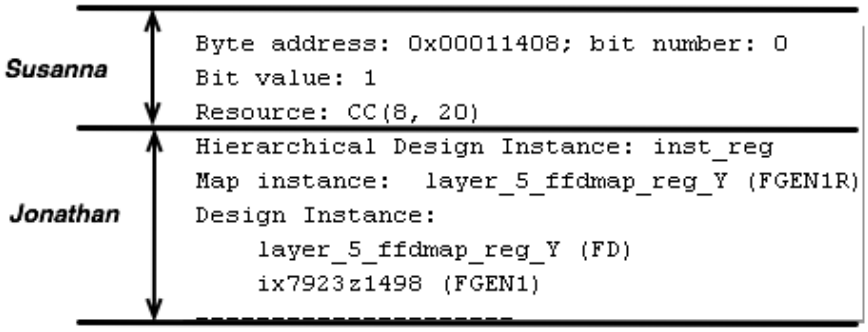


Figure 1.3. Critical bit information

The basic idea to get all the information about the implementation is to extract the data from the project file of the place and route tool Figaro. The Figaro tool takes in input the description of the netlist in the Electronic Design Interchange Format (EDIF), then map, place and route the design on a specific FPGA and finally gives the bitstream in order to program the device. The EDIF file is producted by the synthesis tool Precision. Figaro mainly works on 4 steps:

- Open is the process to open an existing project file or the .edif file.
- Mapping is the process of optimizing design logic and adapting it to a specific architecture.
- Partitioning is the process of allocating the design logic to parts on the board.

- Compilation is the combination of placement and routing.

All the information about the execution of each of the above steps are stored in the project file. At the end of all the implementation process, in the project file are stored: the information at RTL level from the EDIF file, some architectural information, the mapping between the netlist and the library of the target device, the position of the logic/routing/IO used elements in the FPGA and the routing information. The project file is in a readable but not described format. By means of a deep reverse engineer work the information are decoded and tagged, see section 1.4 for the classification of the information.

In order to extract the data a parser is implemented. The parser can read the project file after the *Compile* step and get the information at the different levels. The parser is able to get the RTL information and connect them with the physical used resources stored in the project file after the place and route steps. Moreover, the parser create a graph to represent the netlist by means of the extracted routing information. This graph represents the final netlist implemented on the bitstream after the last optimization done by Figaro. In fact, after a careful analysis we find out that Figaro performs a further optimization step after the ones performed by the synthesis tool.

The connection to the Susanna report is done by the physical resource position. In fact, from Susanna the correlation between the bit and the physical resource is given. On the other hand, Jonathan provides the correlation between the netlist and the physical implementation. Finally the data are merged by the physical resource field.

1.4 Experimental results

1.4.1 Reverse engineering: Figaro Project File

The Figaro project file (.fgd) is created during all the process of place and route. A long reverse engineering work led to decode the information stored in this file and classify them according to the implementation design flow.

The Figaro project file is structured in five main parts:

- General information
- Technology mapped circuit description
- Implementation placement description
- Implementation routing description
- Project features

The *general information* part contains the data relative to the device selected, the library for the macros, the working directory and the design flow information.

The *technology mapped circuit description* contains the description of the circuit as imported from the .edif file. This description comes from the synthesis tool and it has to undergo a phase of technology optimization by Figaro. In this optimization phase the redundant part of the circuit are deleted and some boolean functions are redistributed among the instances. The circuit is described as hierarchy of *Cells*, see section 1.3.1. The external cell contains all the design is unique and it is identified by the keyword **EXTERNAL**. The format for the description of each hierarchical cell of the circuit is:

- list of the inputs
- list of the outputs
- list of the technology instances
- list of the nets

The hierarchical cell description is identified by the tag **HirarchicalCell**.

In the *Implementation placement description* section are described all the instances mapped and placed on the selected device. During the mapping process the design is optimized and adapted to a specified vendor architecture. In this phase, mapping takes the instance from the netlist design and:

1. perform some optimization for area (to reduce the space needed by the design);
2. converts the instances to a technology-indipendent form;
3. convers them to instance which are specific to selected device;

So, after mapping, more RTL instances from the *technology mapped circuit description* level can be put together in a single mapped instance. For each mapped instance the most significative description fields are:

- *name*: specifies the name of the instance;
- *technology macro*: specifies the type of macro implemented in the instance;
- *functioning*: specifies the logic function implemented in the instance;
- *assoc*: specifies the RTL instances of the *technology mapped circuit description* level included in this mapped instance. The representation is in a coordinate format [x y], where x is the index of the most internal hierarchical cell that contains the RTL instance, whose index is y;

- *grid origin*: specifies the position of the Core Cells [13] assigned during placement phase to the instance;

- *net table*: specifies the assignment of the boolean function variable to the input of the Core Cells.

The *implementation* part starts with the keyword **implementation**; each mapped instance description is labeled with **DynamicMacroInstance** or **MacroInstance**. The first one describes functional or dynamical macros [14], the second one describes a Library of Parameterized Module (LPM) macro or an user macro [15].

The following part is the *implementation routing description*. In this part is stored the routing information. For each net the routing is described by means of a tree structure. This tree structure describe the net from the output of a Core Cell to the input of all the fanout Core Cells, go through Local/Local turns, Express/Express turns, Repeaters, Core Cells used to implementing routing functions [13]. The format to describe the net is particular and complicated, see the PURPLE User Guide [17] for more details.

Finally, the *Project features* part contains some other information about the Figaro GUI, the pin out, the design flow, etc...

1.4.2 Test results

In order to evaluate the Susanna and Jonathan tools we have run the tool on several benchmark circuits. The circuits we used range from a simple gate sets up to complex processor cores. The experiments are taken on both the architectural configuration AT40K and ATF280E. For each circuit under test the VHDL description is synthesized using Precision RTL 2010a_Update1.228OEM_Atmel. The devices AT40KEL040KMQFP256 and ATF280EMQFP256 are respectively selected for the synthesis and the mapping of FGEN2 macro cell is enabled. The outcome .edif file is processed by Figaro version ids9.0.2, in order to implement the logical synthesis on a real device. Figaro is used in a complete automatic way to generate the bitstream. Finally the bitstream and the project file of Figaro are given in input to the Susanna/Jonathan tool in order to detect the sensitiveness of the configuration memory on the basis of the implemented circuit.

For each architecture, the data are organized as follow:

- One table shows the results of Susanna tool

- One table shows the results of Jonathan tool

Susanna experimental results

The Susanna experimental results are presented showing the characteristic of the benchmark circuits and the sensitiveness of the configuration memory. For each benchmark are reported the name and the identification of the circuit, the functionality implemented, the number of occupied logic resource, the number of programmed bits in the configuration memory and the number of the configuration bits detected as critical.
The Susanna experiment results are showed in Table 1.1 for the AT40K architecture and in Table 1.2 for the ATF280E architecture.

Moreover, the critical bits are classified in logical and routing critical bits depending on the functionality of the switch programmed by that bit. All the bits that program the LUT, the mux for the selection of the LUT inputs and outputs and the D Flip Flop inside the Core Cell architecture are classified as logic bit. On the other hand, the Repeater configuration bit and the Core Cell configuration bits used to perform Express/Express Turn or Local/Local turn are classified as Routing bits. The impact of the routing on the sensitiveness of the circuit increase with the area occupied by the circuit. Anyway its value is not much bigger than the logic ones and it heavily depends on the place and routing strategies, as showed in Figure 1.4 for the AT40k architecture.

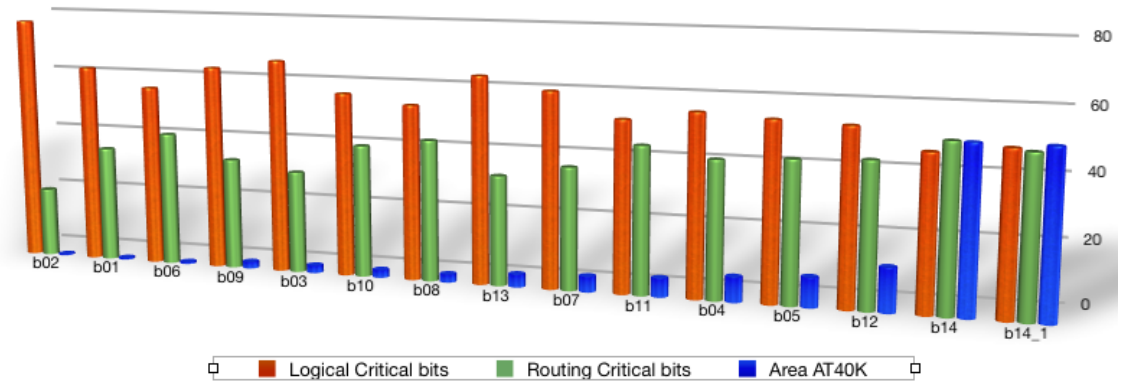


Figure 1.4. Classification criticality AT40K

Different sensitiveness metrics can be compared to analyze how application-oriented analysis differs from other methods. Figure 1.5 and Figure 1.6 show the sensitiveness, with respect three possible metrics. The first one is based on the device cross-section and is basically the area occupied by the design. The second one is based on the number of programmed bits, bits that assumes a value different from

the one in an empty bitstream. Finally, the metric estimated by the static analysis. All the data are expressed in percentage respect the whole configuration memory or area occupied. As showed in both the figures, the metric based on the number of programmed bits lead to completely unrealistic estimations, leaving out all those bits that, even if not used, can induce misbehaviors [10] [18]. On the other hand, the occupied area without any index of the spread of the circuit is a rough metric. In fact, as showed in Figure 1.5 for the circuits implementing a subset of the Viper processor and the optimization of the same circuit (b14 and b14_1), although they occupy more or less the same area of the AT40K device the sensitiveness is different. This can be associated, beyond the optimization of the logic, with the spread of the circuit, the different placement strategy and the consequent different routing.

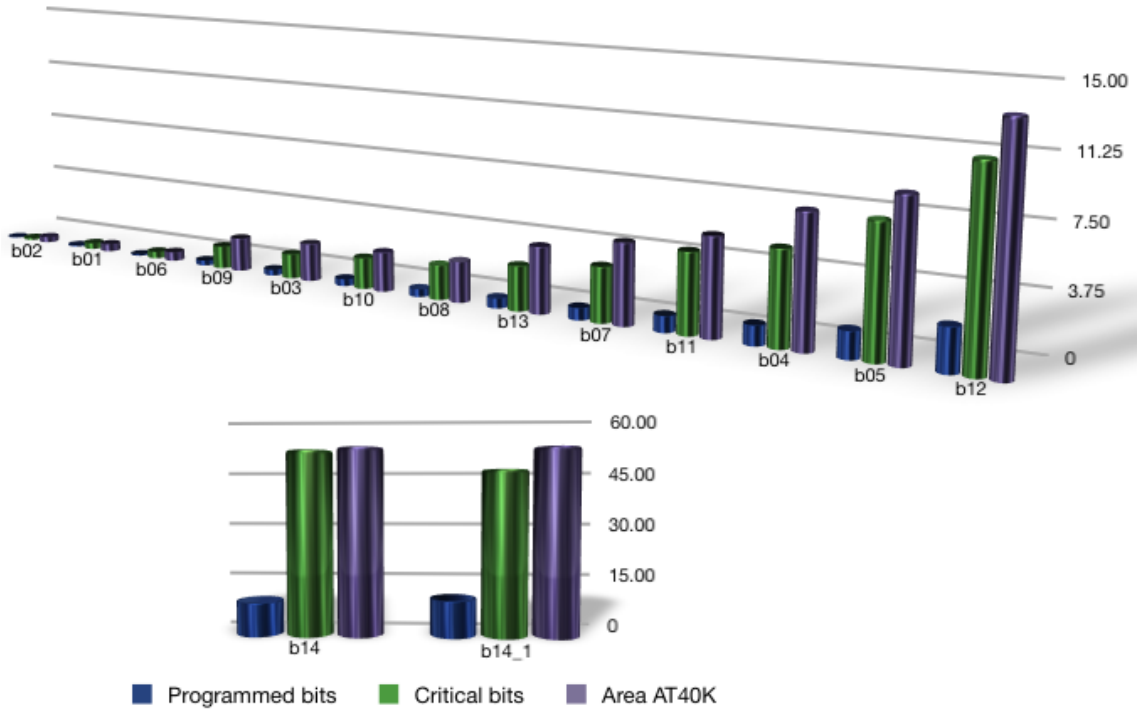


Figure 1.5. Experimental results AT40K

Jonathan experimental results

On the Jonathan side, several statistics can be extracted from the experimental results depending on the complexity of the circuit hierarchy. Jonathan capabilities

Circuit		Figaro logic resources	Programmed bits	Critical bits		
				Tot	Logical	Routing
b02	FSM recognizes BCD num	8	90	387	302	85
b01	FSM compares serial flow	11	242	1095	693	402
b06	Interrupt handler	13	237	1240	713	527
b09	Voting system	48	911	3981	2580	1401
b03	Serial converter	54	1037	4320	2927	1393
b08	Sub-sequences finder	58	1372	6040	3346	2694
b10	Arbiter	56	1220	5452	3171	2281
b13	Interface meteo sensor	93	1747	7754	5071	2683
b07	Count points on a line	113	2122	9506	5851	3655
b11	Scramble string	136	2854	13783	7418	6365
b04	Compute min and max	181	3407	16060	9150	6510
b05	Elaborate contents of memory	215	4635	22091	12312	9779
b12	1-player game	319	7393	32716	17962	14754
b14	Viper processor (subset)	1188	29078	144226	69568	74658
b14_1	Viper processor (subset) optimized	1188	26463	129198	65174	64024

Table 1.1. Experimental results ATF40K -Susanna-

Circuit		Figaro logic resources	Programmed bits	Critical bits		
				Tot	Logical	Routing
b02	FSM recognizes BCD num	8	97	423	316	107
b01	FSM compares serial flow	11	226	1157	650	507
b06	Interrupt handler	13	237	1099	713	386
b03	Serial converter	60	1110	4376	3147	1616
b09	Voting system	49	915	4111	2681	1430
b10	Arbiter	82	1617	7362	5745	1617
b08	Sub-sequences finder	78	1708	7469	4264	3205
b13	Interface meteo sensor	93	1737	7613	5876	1737
b07	Count points on a straight line	120	2226	10039	6077	3962
b11	Scramble string	145	3104	15146	7724	7422
b04	Compute min and max	181	3669	18629	9547	9082
b12	1-player game	399	8327	37484	21349	16135
b14_1	Viper processor (subset) optimized	1532	35509	182860	83391	99469
b14	Viper processor (subset)	1188	35410	182303	82827	99476
	I8051 processor	7518	186170	860712	448458	412254

Table 1.2. Experimental results ATF280E -Susanna-

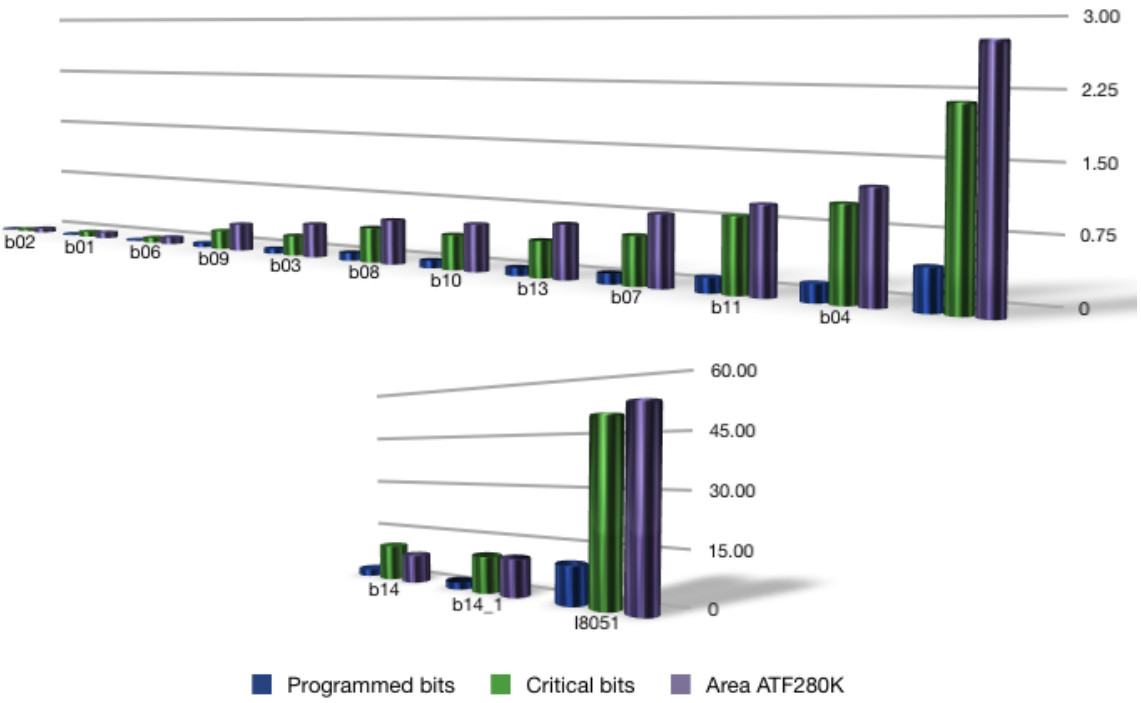


Figure 1.6. Experimental results AT280

can create different statistics on the different level of the design flow. For a circuit would be possible extract the statistic about the criticality of

- the RTL modules that compose the RTL hierarchical description of the circuit;
- the technology cells used after the synthesis phase;
- the macros used after mapping by the place and routing tool.

For the benchmark circuits, the Table 1.3 and the Table 1.4 reported the most critical macro. For the small circuit no macro are used. The criticality of the macros is calculated as sensitiveness bits as the set of logic and routing ones.

Another possible statistic given by Jonathan is the sensitiveness associated to the different modules that compose a hierarchical circuit. As showed in Table 1.5 for the implementation of the I8051 processor are detected the different modules that form the design. The *External* module represents the highest level module that include all the others modules. The results show that the most critical instance is the *U_RAM* module for both the routing functionality and the complex sensitiveness

Circuit		Most critical macro
b02	FSM recognizes BCD num	no macro
b01	FSM compares serial flow	no macro
b06	Interrupt handler	no macro
b03	Serial converter	no macro
b09	Voting system	ix48175z63998
b10	Arbiter	no macro
b08	Sub-sequences finder	no macro
b13	Interface meteo sensor	modgen_counter_tx_conta_ix12439z26862
b11	Scramble string	cont1_addsub9_2i3_ix35655z57342
b04	Compute min and max	ix47767z25108
b05	Elaborate contents of memory	ix47180z35732
b12	1-player game	count_dec6_11i4_ix57040z58209
b14_1	Viper processor (subset) optimized	ix4821z23561
b14	Viper processor (subset)	ix314z23561

Table 1.3. Experimental results AT40K -Jonathan-

Circuit		Most critical macro
b02	FSM recognizes BCD num	no macro
b01	FSM compares serial flow	no macro
b06	Interrupt handler	no macro
b03	Serial converter	no macro
b09	Voting system	ix48175z63998
b10	Arbiter	no macro
b08	Sub-sequences finder	no macro
b13	Interface meteo sensor	modgen_counter_tx_conta_ix12439z26862
b11	Scramble string	cont1_addsub9_2i3_ix35655z57342
b04	Compute min and max	ix47767z25123
b12	1-player game	count_dec6_11i4_ix57040z58209
b14_1	Viper processor (subset) optimized	r_addsub32_2i7_ix22593z23571
b14	Viper processor (subset)	reg2_addsub32_2i10_ix314z23576

Table 1.4. Experimental results ATF280E -Jonathan-

that adds to the sensitiveness of the routing the one of the logic; while the most critical module only for the logic is the *External* one. Finally in Table 1.6 are showed all the statistic results for the I8051 processor.

Instance	Critical bits		
	Tot	Logic	Routing
External	278968	162204	116764
U_ALU	54377	27340	27037
U_CTR	203903	95983	107920
U_DEC	17191	9352	7839
U_RAM	290771	153264	137507

Table 1.5. Instances I8051

Fault injection results

In order to validate the static analysis algorithm we performed fault injection in a design, with both combinational and sequential logic. During a first step the bitstream was analyzed by the Susanna/Jonathan tool, in order to identify the sensitive bits; during a second step the injection was performed in every bit identified as critical by the static analysis and in a set of 20.000 "not-critical" bits randomly chosen. Table 1.7 depicts the obtained results. In this table, for each type of resource are reported the number of sensitive bits identified by the static analysis (S.A.), the number of detected faults after the fault injection campaign (F.I.) and the corresponding percentage. As expected the results show that the static analysis is pessimistic. However, the analysis is quite accurate; indeed in four out of five cases about the 63% of the configuration memory bits identified as critical are actually critical during the fault injection experiments. Pessimistic prediction of Express-CC resources are due to the fact that global routing spans over a big part of the FPGA. Moreover, no one of the 20.000 "not-critical" bits is detect in a fault by the fault injection campaign, this means that the cover of the static analysis tool is 100%.

Once verified the accuracy of the static analysis tool, the SEU sensitiveness is evaluated for the I8051 processor soft core implemented on the ATF280E device. The results of this experiment are reported in Table 1.8. These results are accepted for publication in IEEE RADECS 2010 [10], see AnnexA.

User Module	Critical bits		
	Tot	Logic	Routing
des_1_add16_0i3_ix27920z20265	995	797	198
des_1_add3_0i77_a3_a3	187	151	36
des_1_add4_0i74_a4_a4	246	197	49
des_1_addsub16_0i4_ix27920z28386	1027	815	212
des_1_multu16_0i2_modgen_add_0_ix27920z42477	430	368	62
des_1_multu16_0i2_modgen_add_1_ix27920z42495	454	388	66
des_1_multu16_0i2_modgen_add_2_ix27920z20001	605	494	111
des_1_multu16_0i2_modgen_add_3_ix27920z42514	481	411	70
des_1_multu16_0i2_modgen_add_4_ix27920z42531	500	401	99
des_1_multu16_0i2_modgen_add_5_ix27920z23662	628	528	100
des_1_multu16_0i2_modgen_add_6_ix27920z41938	724	615	109
des_1_sub2_0i1_a2_a2	116	94	22
des_1_sub3_0i76_a3_a3	181	145	36
ix198z48278	455	356	99
ix198z48307	449	350	99
ix198z48310	473	374	99
ix20941z40508	491	401	90
ix21938z40524	450	384	66
ix22935z40513	510	418	92
ix23932z40512	442	400	42
ix24929z40518	428	393	35
ix25926z40511	436	394	42
ix26923z40518	456	408	48
ix27920z29683	405	357	48
ix27920z48374	437	347	90
ix27920z64154	364	297	67
ix60420z64055	363	289	74
ix60420z64062	379	304	75
ix60420z64115	369	296	73
v_add7_0i75_ix53773z34473	392	314	78

Table 1.6. User macros I8051

Type	Detailed Type	S.A [#]	FI [#]	Percentage [%]
Logic	Core Cell	284	154	54.22
Routing	Local - CC	36	22	61.11
Routing	Express - CC	114	2	1.75
Routing	HR	110	75	68.18
Routing	VR	124	86	69.35

Table 1.7. Fault Injection Results

Type	Detailed Type	Critical Bits [#]
Logic	Core Cell	448.458
Routing	Local - CC	33.436
Routing	Express - CC	96.586
Routing	HR	142.099
Routing	VR	140.133

Table 1.8. I8051 Critical Bits

1.5 Conclusion

The SEU cross-section static analysis approach we proposed has been evaluated with respect to dynamic analysis perfomed by fault injection [10], see Annex A. Though being more pessimistic, the static analysis tool is able to predict the actual application sensitiveness with an accuracy of the 60% on the average and without any false negative. The Jonathan capability gives a good estimation of the parts of the design that results more critical. Moreover Jonathan is able to distinguish between the hierarchical levels given by the designer project solution and the macro used by the mapping phase. This means that also for the library macros is possible make an estimation of the reliability. This could open the possibility to estimate the sensitiveness of the existing libraries or define a library of user macros optimized for the reliability fields.

Chapter 2

Use of macros in Atmel architecture

2.1 Introduction

The macros are components designed to perform a functionality, fixed or definable. The scope of the macros is to offer better performance in term of timing and area. The Atmel macro libraries for the ATF40K and ATF280E families of FPGAs contains two kinds of macros: functional and dynamical. Functional macros are components with fixed functionality, such as the 2 input AND gate. Dynamic macros are designed to allow user specification of any desired functionality attached as an attribute. The dynamic macros are provided to give the user better control over the implementation of specific functions in a single Core Cell.
The study was aimed to prove that the efficient use of the logic cell architecture, by means of dynamic macros, could improve the area and the reliability aspects. The results of this study were sent to the Atmel company that provided this documentation to the Mentor company in order to request the use of the macros during the synthesis by Precision tool.

2.2 Motivation

As detected from other Atmel costumers, the use of the logic resource architectures did not exploit all the capability of the element [4]. In particular, the macro detection problems lead to a leak of performance in term of area and, as it has been demonstrated, of reliability. Since the capability of the Atmel FPGA are restricted due the size of the devices a further leak means a considerable reduction of the device capacity. The use of macros has a profound impact on the design.

2.3 Proposed Approach: An Overview

The analysis of the implementation of the circuits on the device shows that some logic functions, that implement two different functions in a single Core Cell, are not used neither during the synthesis phase nor the place and route one. The purpose of this study is to illustrate the improvements that would be brought using all the logic function available in the AT40k and AFT280E technology mapping.

The study was developed in two parts.

- First step: manual.
  It confirmed the manual effort benefits.
- Second step: automatic.
  To prove the benefits on more and bigger circuits.

During the first step some circuits were tested, in terms of area and sensitiveness, before and after an manual optimization. This optimization provided the optimized use of the logic cell architecture by means of user macros implemented specifically for the design under test. This means that for each circuit under test, the design implementation on the target device was studied. The use of the Core Cell architecture was manually examined.
If the single functionalities implemented on two different Core Cells could be implemented on a single one Core Cell performing two functions, the functionality of this Core Cell is implemented on a user macro. Figaro tool allows the user to create own user libraries and store the macros for current and future designs. The user macro is used like a black box during the synthesis and mapped in the design during the place and route phase. So for each circuit under test a specific user macro is created. The VHDL code of the circuit is modified in order to use the user macro in place of the two functionalities, the new circuit description is synthesized. At this point, the user macro is mapped in the circuit using Figaro that performs also the *Compilation* stage. The two versions of the same circuit are analyzed by Susanna and Jonathan tools. The results show that, beyond the area gain, the optimized use of the Core Cell architecture improve the sensitiveness. In fact, the optimization leads to occupy as much as possible of the capability of a logic cell. For this reason the amount of configuration data not directly used to implement the function, but that surround the logic and could induce an error on it, is thus reduced.

In a second phase an automatic tool was developed in order to detect the amount of resources that could be optimized by means of the correct use of dynamic macros. This automatic tool can recognize the couple of Core Cells which functions can be collapsed in a single one Core Cell. This tool performs the analysis of the EDIF netlists of realistic circuits implemented on Atmel FPGAs. The analysis process has been developed on the basis of the Polito Automatic Hardening Tool (PAHT).
See Annex C and Annex D for all the details about this study.

2.4 Conclusion

This study showed, in both the steps, manual and automatic estimation, the gain in term of area. In particular, for the manual experiments the sensitiveness gain was also showed. In average the gain in term of area is around 10%, with a pick of 15% for a simplified implementation of the Leon2 (without cache). This study has contributed to have the use of optimized macro in the new release of synthesis tool Precision RTL Synthesis 2010a_Update1.228OEM_Atmel.

Chapter 3

Platform to Upgrade and Redo Placement Leading Efficiency

3.1 Introduction

Placement is the process by which a netlist of circuit blocks (which are whether I/O or logic blocks) are mapped onto physical location in an FPGA. Placement is one of the most important steps in the post-RTL synthesis process as it directly defines the interconnects, which have now become the bottleneck in circuit and system performance. This task is aimed to study and development a platform to apply a new placement algorithm in order to improve the timing/criticality aspects of designs implemented on the Atmel AT40K and ATF280E series. The Platform to Upgrade and Redo Placement Leading Efficiency (PURPLE) idea was born to provide an independent solution in order to improve the performance of the implementation of a design on an Atmel FPGA device. PURPLE can be used to evaluate the different characteristics of a circuit using different placement strategy. Moreover, the placement strategy could be lead to improve the performance of an implementation or improve the reliability of it.

3.2 Motivation

The place and route is responsible for producing a physical implementation of an application netlist on the FPGA devices. More specifically, the placement tool determines the actual physical location of each netlist logic block in the FPGA layout, and the routing tool assigns the signals that connect the placed blocks to routing resources in the FPGA's interconnect structure. Due to the finite nature of an FPGA's interconnected structure, the success of the routing tool is heavily reliant on the quality of the solutions produced by the placement tool. Placement

tools are not only used for subsequent routing step but a timing-driven router can only produce routings that are as good as the placement on which the routing is performed, so to extract more speed out on an FPGA it is essential that timing-driven placement algorithms be used [4] [5] [6].

The Atmel Figaro tool performs the place and route in a complete automatic way. Moreover the manual placement is supported in order to optimize placement manually. The use of this strategy can not be considerate ordinary, the Figaro Help warn the designer off using the manual placement if he is not familiar with the architecture [16]

> Detailed knowledge of the architecture could be used to improve the timing along the paths in the design by manual editing.

In a first feasibility study, some experiments are done to evaluate the benefits of the optimized placement. The results of the tests show that it is possible to reach a gain in frequency of about 30% with manual placement, although the device utilization increase due the growth of the routing complexity, see Table 3.1. Another aspect is that the time needed to achieve this improvement in performance weigths heavily on the results. Up to 8 hours were needed to achieve a good solution. Then after a certain value of improvement further effort is not worth, as shown in the second entry in Table 3.1.

The experiments are taken on small circuits (from 6% to 15% of the device area is used), the complexity of the operation increases with the size of the circuit. In fact optimize placement manually for big circuit can be used to make specific adjustments not to improve the global performance of the implementation. The manual placement can be used to:

- Resolve contention.
- Make easier to route a specific net.
- Squeeze the design that the automatic placement has not succeeded in fitting.
- Improving timing on a path.
- Reserve a particular area for an instance.

Although the manual placement is presented as a powerful technique to refine upon the place and routing implementation, it is the Figaro Help itself that warns on use of this technique.

> Only optimize placement manually if you are familiar with the device architecture and know how your design should us it.

Circuit	Atmel's P&R		Manual Placement		Gain		Time
	Max frequency	Device utilization	Max frequency	Device utilization	Frequency	Device utilization	$[h]$
Memory elaboration	7.1MHz	11.4%	10.4 MHz	12%	32%	5%	8
			10.5 MHz	11.8%			+2
Scruble string	10.9MHz	6%	15 MHz	8.5%	27%	29%	6
1-player game	11.5MHz	15%	15.3 MHz	20.3%	25%	26%	5

Table 3.1. Manual placement

Furthermore, no external constrains are supported to drive the placement. This means that the development of an independent platform becomes necessary to automate the re-place changes. The independent platform can allow both of the methods:

- Use of other placement algorithms in order to improve a particular features of the design.
- Use of a set of constrains to drive the placement algorithm.

3.3 Proposed Approach: An Overview

The platform should work in parallel with Figaro tool in order to let Figaro provide the routing step and the bitstream generation. Moreover it might be possible use the Figaro solution as starting point for the re-placement. The basic idea is that the platform would be able to:

- read a Figaro project file;
- give the information about the Figaro placement solution;
- contain a re-placement engine;
- create a Figaro project file implementing the new solution in order to feed Figaro with the the new implementation and complete the design flow.

As showed in Figure 3.1 the PUPLE platform takes as input a Figaro project file and produce another Figaro project file implementing the new placement solution. The platform is composed by three main blocks:

- Front End.
- Placement Engine.
- Back End.

The *Front End* block extracts the netlist of the circuit under test from a Figaro project file that implements the circuit on a specific device. The information is fed to a placement engine to obtain a new placement. The placement solution is transformed in a new Figaro project file by means of the *Back End* block.
The *Front End* block implements a parser able to read the Figaro project file and give as output the netlist description. The parser is basically the same used for the Jonathan tool with some modifications for the netlist storing. The *Back End* block implements part of the same parser extended with the capabilities of recognize

the part of the project file implementing only the placement step and change the physical resource positions according with the output of the placement engine.

The *Placement Engine* block contains the coded placement algorithm that the user wants to apply. This block is movable, it is possible exchange this block with several placement engines. The independency of the blocks that compose PURPLE is given by intermediate files, as showed in Figure 3.1. The data passed by the single blocks are write on a file. In particular, for the two intermediate files used in PURPLE to describe the netlist and the placement solution, it is defined a specific format for each of them, see following section.

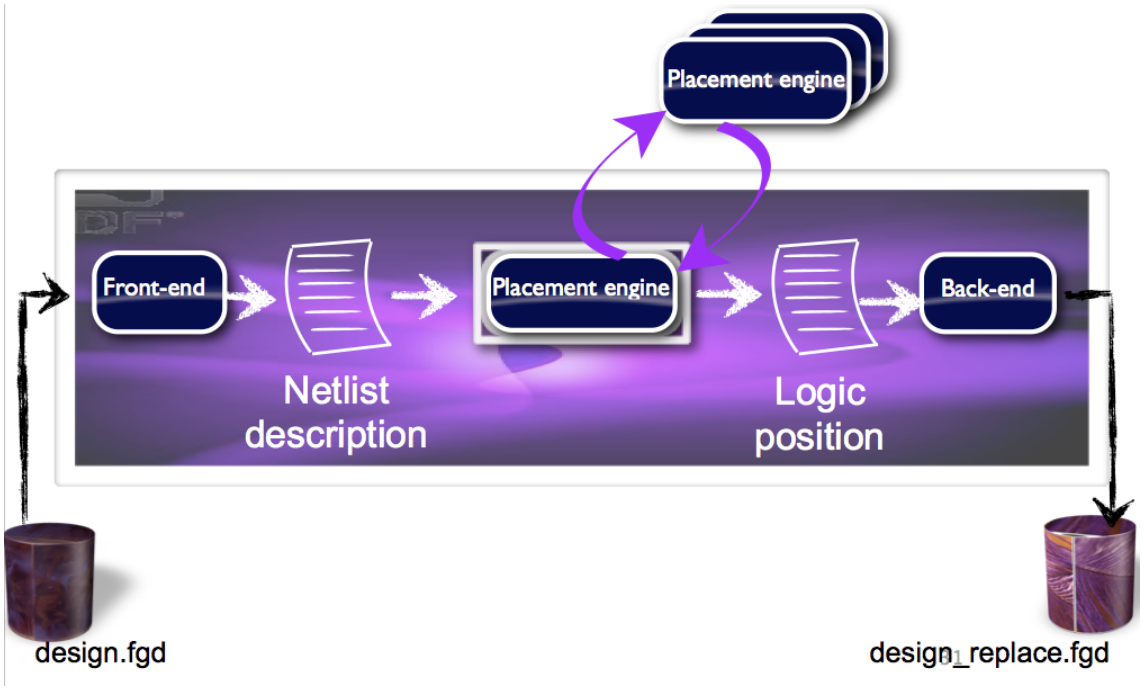


Figure 3.1. PURPLE scheme

3.3.1 Intermediate Files

To allow the complete independency between the modules that composed PURPLE, two intermediate files are used:

- Netlist description file
- Logic Position file

The format of these files is simplified as much as possible in order to give the maximum of handling to the developer of the placement engine.

The *netlist description* file is composed by three parts:

- Pin input, specified using the keyword *PI*, lists all the input pins
- Pin output, specified using the keyword *PO*, list all the output pins
- Gates, specified using the keyword *GATES*, list all the gates
- Fan out, specified using the keyword *fan-out*, describe the single nets

All the keywords are preceded by the semicolon simbol (*;*). For each net the description is

- Name of the net, specified using the symbol semicolon (*;*) in front of the name
- Number of fan-out elements
- Name of elements in fanout, as many as the previous field indicates

An example netlist described with this format is given below. The netlist described is represented in Figure 3.2.

```
; PI
inputA
inputB
inputC
; PO
outputX
outputY
; Gate
gateG
gateH
gateK
; fan-out
; inputA_net
3
gateG
gateH
gateK
; inputB_net
2
```

```
gateG
gateH
; inputC_net
1
gateK
; gateG_Xnet
2
gateH
; gateG_Ynet
1
gateK
; gateK_Xnet
1
outputX
; gateH_Xnet
1
outputY
```

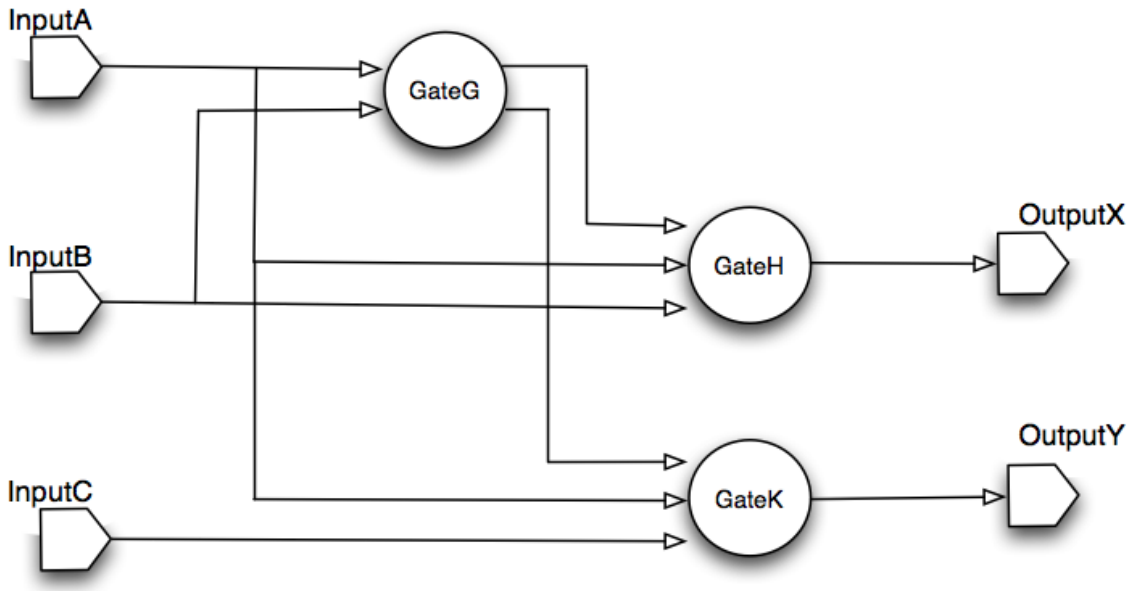


Figure 3.2. Netlist example

This *netlist description* file is generated by the Front End block processing the Figaro project file, where the gates are the logic cells used to implement the circuit

and the nets are the connections among them.

The other intermediate file is the *position description* file. In this file are reported the new positions for the logic cells and I/O elements. The format is shown below:

For logic cell:

```
[xOld yOld] --> [xNew yNnew]
```

For I/O element:

```
[xOld yOld label] --> [xNew yNnew label]
```

where *xOld* and *yOld* are the coordinates of the initial placement given by the Figaro solution, stored in the Figaro project file in input; while *xNew* and *yNew* are the coordinates assigned by the new placement. The label for the I/O specified the type of I/O.

3.4 A placement engine: Versa-Tile Place and Route

Three factors combine to determine the performance of an FPGA: the quality of the CAD tools used to map circuits into the FPGA, the quality of the FPGA architecture and the electrical (i.e. transistor level) design of the FPGA [23]. In order to validate the existing CAD tool and to improve the performance of the Atmel FPGAs we decide to work also on a *placement engine* for the PURPLE platform.
During the assessment phase for the *placement engine* task two main alternatives are evaluated to implement the placement engine:

- Develop a specific placement engine for the Atmel FPGA devices.
- Use a well-know generic placement engine for FPGA devices.

Since the limited time and resources the final decision was to not implement a new dedicated placement engine for the Atmel architecture but use an existing, timing-driven, well-known one. There are several works on generating timing-driven placement examples [21] [22]. However, not everyone can match our requirements. After a deep research in the literature we selected the Versa-Tile Place and Route (VPR) tool from University of Toronto. VPR is a placement and routing tool for array-based FPGAs, VPR incorporates also a timing-driven router. A timing-driven router can only produce routings that are as good as the placement on which the

routing is performed, so to extract more speed out of an FPGA it is essential that timing-driven placement algorithms are used.

In brief, the reasons that led us to choose the VPR tool are:

- The expectation of being a general and flexible tool like described by the authors [25],
- The use of timing-driven algorithm (simulating annealing) [24],
- The open-source code [3],
- The robustness of the code (VPR project started in 1996 and it has been added to the SPEC 2000 suite of computer benchmarcks)
- Largely used in the research community

The inputs of VPR consist of a technology-mapped neltlist and a text file describing the FPGA architecture, see section 3.1. VPR can place the circuit and also perform either a global router or a combined global/detailed route of the placement. VPR's output consist of the placement and routing, as well as statistics such as routed wirelength, track count, etc. etc. For our purpose we use only the placement capability of the tool.

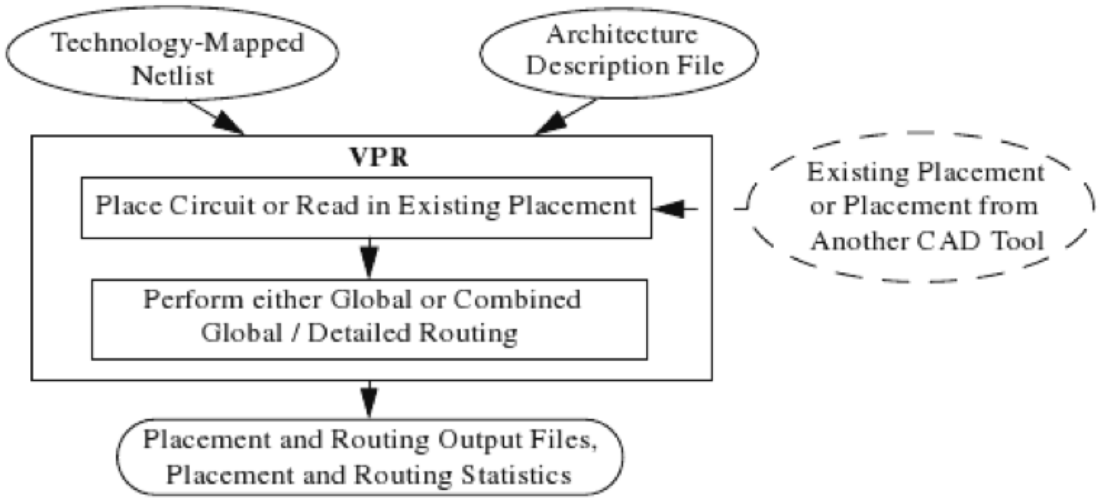


Figure 3.3. VPR CAD flow

Some of the architectural parameters that can be specified into the architecture description file are:

- the number of logic block input and outputs,
- the side(s) of the logic block from which each input and output is accesible
- the logical equivalence between various input and output pins
- the number of I/O pads that fit into one row or one column of the FPGA
- the dimensions of the logic block matrix

In addition, if the routing is to be performed, one can also specify the characteristic of the routing lines.

The details of the algorithm used on this tool and the relatives works are out of the purpose of this report, please check the website [3] or the web page of one of the authors Vaughn Betz [26] for more information.

In order to interface the PURPLE platform to the VPR tool the format conversions are needed. Two converter tools are developed in order to convert from and to the intermediate files of PURPLE platform, see section 3.3.1. One tool converts the netlist input file from the intermediate *netlist description* format to VPR format, while the other tool translate the placement output file of VPR to the *logic position file* of PURPLE. The respective converters are developed in C/C++ and each one contains a parser, developed in flex and bison languages, for the file in input.

3.4.1 The congestion factor

For large circuit ($\geq 25\%$ area of the FPGA occupied), the congestion of the solution is becoming an important factor for the effectiveness of the routing step. If the congestion is such that the contentions routing resource are so many, the placement solution can still be implemented by Figaro but the tool is not able to route it. Several measures are taken in order to solve this issue.

A first research led to a couple of works on the congestion factor for the VPR tool [19] [20]. These works introduce a new congestion driven placement algorithm for FPGA, based on the VPR algorithm, modified with a congestion factor. To reduce the routing channel width, a placement algorithm has to pay attention to both the resource consumed by each net, and the interaction (congestion) among different nets. The idea is to reduce the amount of nets in a small fraction on the chip.

The cost function in VPR is:

$$\Delta C = \lambda \frac{\Delta C_T}{PreviousC_t} + (1 - \lambda)\frac{\Delta C_W}{PreviousC_W} \tag{3.1}$$

where C_T is the timing cost, C_W is the wiring cost and λ is a constant between 0 and 1 which trades off between cost and wiring cost.

$$C_T = \sum_{\bigvee_{i,j} \subset circuit} C_T(i,j) \tag{3.2}$$

$$Crit(i,j) = 1 - \frac{Slack(i,j)}{D_{max}} \tag{3.3}$$

where $C_T(i,j)$ is the timing for each edge of the netlist and β is a constant from 1 to user-defined maximal value which is 8 by default. The wiring cost is defined as:

$$C_W = \sum_{i=1}^{N_{nets}} q(i)(bb_x(i) + bb_y(i)) \tag{3.4}$$

In the congestion works is used the same top-level cost function in Equation 3.1 and the top-level timing cost function in Equation 3.2. The formulas to compute the timing cost for each edge and the witing cost are modified as shown below:

$$C_T(i,j) = History(i,j) * Delay(i,j) * Crit(i,j)^{\beta} \tag{3.5}$$

$$C_W c = Congestion * \sum_{i=1}^{N_{nets}} q(i)(bb_x(i) + bb_y(i)) \tag{3.6}$$

Where the *History* factor assign more weight to a recent stable solution since it is more reliable. This is applicable when the rollback function is activated due the fact that the last set of movements have increased the cost of the solution. The *Congestion* factor take into account the congestion of the solution, following the bounding box strategy. The *Congestion* is computed by the following formula:

$$Congestion = \left(\frac{\frac{\sum_{x,y} U_{x,y}^2}{n_x * n_y}}{\left(\frac{\sum_{x,y} U_{x,y}}{n_x * n_y} \right)^2} \right)^k \tag{3.7}$$

where $U_{x,y}$ is the number of bounding boxes covering the logic cell, k is a small positive integer, and the whole chip consists of nx by ny logic cells.

In both the works [19] [20] the*Congestion* factor is introduced. This factor is multiplied for the wiring cost used by VPR to estimate the cost of a swap elaboration of the placement algorithm. In particular, this factor is modified between the first and the second work on the aspects that in the seconds one it takes into account also the empty logic cells in the swapping. The scope of this is to bring in the middle of a congestion area some empty logic cells. Moreover, the assignment of some parameter to calculate the wiring cost are changed between the two solutions

in order to improve the efficiency of the algorithm according with some experimental results.

The code of these others two version was asked to the authors and tested as well. Only one of the several versions received proved to be running with the proper cost factor.

3.5 Experimental Results

The VPR algorithm is applied to several circuits in order to evaluate the capability of this tool on the Atmel FPGA architecture. The circuits are synthesized as flat netlist circuits using Precision RTL Synthesis. The flat attribute on the modules of the circuit allows to flatten the design hierarchy. This is necessary because VPR does not support the hierarchical circuits. Then EDIF description is given in input to Figaro that place and route the circuit on the AT280E device. At this point the Figaro project file is fed to the PURPLE platform.

During the run, for each circuit:

- The *Front End* block, extracts the data from the Figaro project file and give a *netlist description file*
- A converter tool converts the netlist description from the format of the *netlist description file* to the VPR netlist format
- VPR tool performs the placement
- A converter tool converts the output placement file of VPR in the *position description* file.
- The *Block End* block creates the Figaro project file that implement the VPR solution.

At the end of the run, Figaro is used to route the VPR solution. If the route success, for both the project, the original and the one implementing the VPR solution, the timing is evaluate using the Figaro Timing Analyzer. A preliminary set of experiments show that the VPR solution, in most cases, is better in term of Delay Longest path, see Table 3.2. These results are considerate preliminary for two reason:

- The placement of the IO, for the VPR solution, is not optimized. This could improve the timing results for the path that involve an IO pin as endpoint of the path.
- The Figaro Timing Analyzer tool is not completely reliable. To improve the accuracy of the estimation the experiments should be analyzed by a different timing analysis tool.

Circuit	Used logic resources	Delay Longest path[ns]	
		VPR	Figaro
b01	11	22.98	30.09
b02	8	20.49	29.52
b03	54	33.1	41.23
b06	13	33.64	37.85
b13	93	32.75	32.03

Table 3.2. Simple VPR test experiments

The experiments show that the circuit tested are not bigger than the 10% of the FPGA. The test on larger circuit failed due the congestion of the VPR placement solution. In fact, for circuits which occupied area is between 10% and 25% of the total area of the device, Figaro was not able to route the VPR placement solution.

The cost factors taken into account in the VPR algorithm tends to place the connected resource as much close as possible, as showed in Figure 3.4_a. Since the limited routing resource in the Atmel FPGA architecture, Figaro is not able to connect congested areas of the device, as showed in Figure 3.4_b. Several measures are taken to try to avoid high density of used logic resource in restricted area of the FPGA. The congestion solutions presented in the two works [19] [20] are used for the un-routed circuits. The results have shown that the solution given by [20] is better than the solution given by [19] but it is still not enough strong to allow Figaro to route the placement solution.

The parameters of the total cost in Equation 3.1 and in the *Congestion* cost in Equation 3.7 are changed in order to increase the weight of the *Congestion* in the cost function. The parameters are scaling from the default values to the maximum value that still guarantee the balancing with the other cost factors (wiring cost and timing cost). For example, the value of λ in Equation 3.1 is decrease down to 0.2 in order to maintain the minimum influence of the timing cost.

The last measure considering unoccupied logic cell. The objective is to force the placement of empty logic cell close to the ones with hight fanout. This objective is achieved increasing the weight of empty logic cell in selection of the core cell to swap in the placement algorithm. The effect of this measure give support to the Congestion one, as showed in Figure 3.4_c.

These changing are tested on a circuit implementing an acdc converter. The circuit occupy the 22% of the logic resource of the ATF280E device.
The results of the experiments are shown in Table 3.3. The first three columns show the cost parameters: λ in Equation 3.1, the nx and ny in Equation 3.7. The fourth column shows the percentage of empty logic cell respect the total occupied cells by the circuit, inserted in the most congested area. The last column shows

the route contentions given by Figaro after the routing step. The tests show that the only modification of the costs parameters can not lead to the solution of the route contentions. Although the changing of the parameters led to decrease the contentions of about 50% the amount of total route contentions is still too high (465 contentions).
The measure of force the placing of empty logic cell in congestion areas is the most efficient approach although it is not the definitive solution.

λ	nx	ny	empty logic cell	route contentions
0.5	Auto	Auto	0	953
0.3	+10%	+10%	0	651
0.3	+30%	+30%	0 655	
0.2	+10%	+10%	0	465
0.3	+10%	+10%	20%	370
0.2	+10%	+10%	20%	250
0.3	+10%	+10%	37%	73
0.2	+10%	+10%	37%	120

Table 3.3. Routed contentions for *acdc* circuit

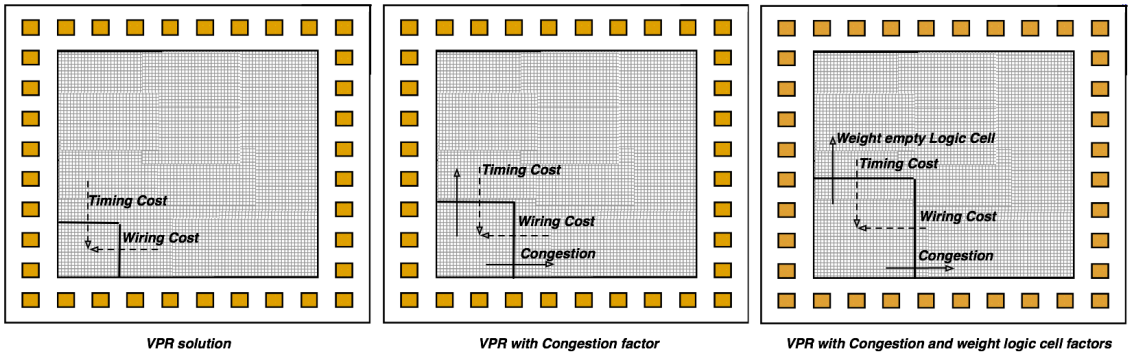


Figure 3.4. VPR solutions

We can conclude that the experiments showed that the best parameters to use are:

- Area FPGA under test ≥30% more than the minimum required to fit the design

- Timing-tradoff between bouding box minimization and delay minimization in the placer around 0.3%
- The weight for the unused logic cell in the quantification of the alleviation of congestion brought by an unused block should be almost the weight of an uses logic cell.

3.5.1 Why does VPR not work properly for Atmel?

The experimental results show how VPR is not able to reach a good solution for all the circuits under test. The mainly problem is the congestion factor. The VPR solution seems to be too much congested for the architecture of Atmel. There are several points that can cause this leak:

- VPR was developed for Altera and Xilinx architecture that are more complex and powerful.
- From VPR support.

  > VPR takes into account some of the routing description in placement. It is far too time consuming to make full use of detailed routing information so instead, we chose to use simplified models of routing during placement. As you can imagine, there are many different ways to simplify the routing model and the -place_cost_type parameters affects which simplified model to use.

- VPR does not implement direct connections with the diagonal neighbors.
- VPR does not support the express bus in Atmel architecture.
- VPR does not support black-box, so it is not possible use optimized structure like *lpm* elements that could decrease the routing between the placed logic cells.
- VPR does not support 3-state IOs, so we have to define a "fake" IO for the enable signal of the 3-state IO, this increase the routing.

Moreover, VPR works normally with a packer tool. It is proved the efficiency of the use of the cluster to improve FPGA speed and Density [27]. The logic cluster "packed" different LUTs together followings some rules. The logic cluster are a generalized version of the Logic Array Blocks used in Altera's FLEX 8k and FLEX 10K parts and the Xilinx 5200 and Virtex FPGAs. With the logic cluster structure the local routing is reduced by the packer, because the placement algorithm will work on the clusters and the routing inside the cluster is ignored by the placer.

So the complexity of the routing is reduced by the use of the cluster structure. Unfortunately, the Atmel architecture does not support the cluster. This means that more routing will effect the placement. The use of clusters and a more powerful architecture of routing justify the choice of a simplified models during the placement obtained also gain in time performance.

3.6 Conclusion

The PURPLE platform allows the use of others placement algorithms on Atmel FPGA devices. The platform can work in parallel with Figaro tool in order to hook the normal flow of the design implementation and use the features of the commercial tool. The independence of the blocks inside the platform allows the use of different placement engines. The use of VPR showed the importance of the estimation of the routing capability for the placement solution. The congestion problems due the fact that VPR was born for different and more powerful architectures lead to detect the most critical points for a placement engine used on to Atmel FPGA devices. Add some congestion metrics to the cost function of the placement code so that is spreads out blocks in congested area could improve the efficiency of VPR.

Bibliography

[1] http://www.actel.com/products/milaero/rtpa3/default.aspx

[2] C. Carmichael, *Triple Module Redundancy Design Techniques for Virtex FPGAs*, Xilinx Application Notes XAPP197, 2001.

[3] http://www.eecg.toronto.edu/~vaughn/vpr/vpr.html

[4] , CNES feedback on the ATF280 presentation, Atmel FPGA User Group Workshop, 3rd of March, 2010 ESTEC, http://spacefpga.atmel-nantes.fr/spacefpga/files/Meetings/03march2010/P0.2_CNES_ATMEL\%20FPGA\%20User\%20Group\%20Workshop\%20\textendash\%20ESTEC\%2003\textendash03\textendash10.pdf

[5] , Thales Alenia Space feedback on the ATF280 presentation, Atmel FPGA User Group Workshop, 3rd of March, 2010 ESTEC, http://spacefpga.atmel-nantes.fr/spacefpga/files/Meetings/03march2010/P6_Workshop\%20ESA\%20ATF280E\textendashTAS\textendashF.pdf

[6] , Institut d'Astrophysique Spatiale feedback on the Atmel FPGA presentation, Atmel FPGA User Group Workshop, 3rd of March, 2010 ESTEC, http://spacefpga.atmel-nantes.fr/spacefpga/files/Meetings/03march2010/P6_Workshop\%20ESA\%20ATF280E\textendashTAS\textendashF.pdf

[7] M. Alderighi, A. Candelori, F. Casini, S. DÕAngelo, M. Mancini and A. Paccagnella et al., *SEU sensitivity of virtex configuration logic*, IEEE T Nucl Sci 52 (6) (2005), pp. 2462–2467.

[8] K. Morgan, M. Caffrey, P. Graham, E. Johnson, B. Pratt and M. Wirthlin, *SEU-induced persistent error propagation in FPGAs*, IEEE T Nucl Sci 52 (6) (2005), pp. 2438–2445.

[9] FL. Kastensmidt FL, L. Sterpone, L. Carro, MS. Reorda,*On the optimal design of triple modular redundancy logic for SRAM-based FPGAs*, Proceedings of design, automation and test in Europe (DATE) 2005, vol. 2; 2005. p. 1290–5.

[10] B. Barcelin, N. Battezzati, DS. Codinachs, F. Decuzzi, F. Margaglia,M. Violante, ,*Application–oriented SEU cross–section of processor soft core for Atmel RHBD*, IEEE Radecs 2010 [Accepted for publication]

[11] www.atmel.com

[12] Precision Synthesis Reference Manual, 2003c Update1, March 2004

[13] Rad Hard Reprogrammable FPGA, ATF280E, Advanced Information, Atmel
[14] AT40K IO Generator Guide, June 2002
[15] Figaro tool Help, Atmel
[16] Integrated Development System - Figaro Tutorial, June 2002
[17] PURPLE user guide, Decuzzi Filomena November 2010
[18] Application-oriented SEU sensitiveness analysis of Atmel rad-hard FPGAs, N. Battezzati, F. Decuzzi, M. Violante, M. Briet, 15th IEEE International On–Line Testing Symposium, 24–26 June, 2009, pp. 89–94
[19] Y. Zhou, H. Li, S.P. Mohanty, *A congestion driven placement algorithm for FPGA synthesis*, In Proceedings of the International Conference on Field Programmable Logic and Applications, 2006,pp. 683–686.
[20] Y. Zhuo,H. Li,Q .Zhou,Y .Cai,and X. Hong,*New timing and routability driven placement algorithms for FPGA synthesis* in Proc. GLSVLSI, 2007, pp. 570–575.
[21] M. Hutton,J.P. Grossman, J. Rose, and D. Corneil, *Synthetic benchmark circuits for timing-driving physical design applications* in Proc. Design Automation Conf, ACM Press, 2002, pp.94–99
[22] P. Verplaetse, D. Stroobandt, and JV. Campenhout,*Synthetic benchmark circuits benchmark circuits for timing-driven physical design applications*, in Proc. International Conference on VLSI, CSREA Press, 2002, pp.31–37
[23] V. Betz, J. Rose, A. Marquardt, *Architecture and CAD for Deep-Submicron FPGAs*, Kluwer Academic Publishers, February 1999
[24] A. Marquardt,V. Betz, and J. Rose, *Timing-Driven Placement for FPGAs*, ACM/SIGDA International Symposium on Field Programmable Gate Arrays, Monterey, CA, February 2000, pp. 203–213.
[25] V. Betz and J. Rose, '*VPR: A New Packing, Placement and Routing Tool for FPGA Research*, Seventh International Workshop on Field-Programmable Logic and Applications, London, UK, 1997, pp. 213–222.
[26] `http://www.eecg.toronto.edu/~vaughn/`
[27] V. Betz, J. Rose, A. Marquardt, *Using Cluster-Based Logic Blocks and Timing-Driven Packing to Improve FPGA Speed and Density*, ACM/SIGDA International Symposium on Field Programmable Gate Arrays, Monterey, CA, February 1999, pp. 37–46

Appendix A

Annex

Application-oriented SEU cross-section of a processor soft core for Atmel RHBD FPGAs

N. Battezzati, F. Margaglia, M. Violante
Politecnico di Torino, Dip. Automatica e Informatica, Torino, ITALY

F. Decuzzi, D. Merodio Codinachs
European Space Agency, ESA/ESTEC, Noordwijk, THE NETHERLANDS

B. Bancelin
Atmel Corporation, Nantes, FRANCE

35-WORD ABSTRACT:

Approximating SEU sensitiveness by the device cross-section for applications implemented in FPGAs is very pessimistic. We propose and validate a static analysis approach to asses the application-oriented cross-section, providing evidence on a soft processor core.

Corresponding and Presenting Author:

Niccolò Battezzati, Dip. Automatica e Informatica, Politecnico di Torino, C.so Duca degli Abruzzi 24, 10129 Torino, ITALY, phone: +39 011 564 7221, fax: +39 011 564 7099, email: niccolo.battezzati@polito.it

Session Preference: Single event effect 2: devices and integrated circuits

Presentation Preference: poster

Abstract — **Approximating SEU sensitiveness by the device cross-section for applications implemented in FPGAs is very pessimistic. We propose and validate a static analysis approach to asses the application-oriented cross-section, providing evidence on a soft processor core.**

Index Terms — **RHBD, FPGA, Single Event Upset (SEU), fault injection, static analysis, processor.**

I. Introduction

FIELD Programmable Gate Arrays (FPGAs) that have the capability of being reconfigured when already deployed in the field are drawing more and more attention from designers of space applications. On the one hand, re-configurable FPGAs make possible fixing bugs when the device is already deployed in a satellite. On the other hand, re-configurable FPGAs are the enabling technology for implementing the configurable computing paradigm, which is gaining popularity as an effective mean to share the same hardware resource among different applications or different parts of the same application, or to upgrade a design as soon as the user requirements change (e.g., to implement a new transmission standard as soon as it is available).

Nowadays, re-programmable FPGAs can be SRAM-based FPGAs, when the information defining the configuration of the device (i.e. the configuration memory) is stored using SRAM memory cells, or Flash-based FPGAs, when the configuration memory is implemented resorting to floating-gate memory cells.

As far as space applications have to survive to harsh environments, where ionizing radiation may interact with silicon devices and induce Single Event Effects, like Single Event Upsets (SEUs) in the FPGA configuration and user memory (i.e., flip-flops), and Single Event Transients (SETs) in the FPGA routing resources, silicon vendors proposed different solutions. On the one hand radiation-hardened technologies [1] can be used to build robust devices, but they have much higher cost and performance overhead than Commercial-Of-The-Shelf (COTS) devices, and they are not re-programmable. On the other hand it is possible to use commercial devices and implement some hardening techniques at the application level [2]. Some vendors, like Xilinx, proposed to use COTS devices for mainstream applications in conjunction with proper SEE mitigation techniques that, by exploiting hardware/information redundancy and memory scrubbing, can alleviate the SEE problem [3]. Other vendors adopted configuration memory cells that are less likely to undergo SEEs. Actel adopted floating gate cells based on a 130 nm process that radiation testing experiments performed so far shown to be insensitive to heavy ions [4]. In the middle of these two approaches Radiation-Hardened-By-Design (RHBD) [5] FPGAs offer a robust device even if using the same technology as COTS devices. Atmel developed RHBD configuration memory cells and flip-flops that are less sensitive to ionizing radiation; Xilinx is also developing an RHBD version of its Virtex-5 device that promises unprecedented level of immunity to SEEs.

As far as Atmel RHBD devices are considered, although robust against SEEs, they still have a not-null cross section. The most recent product, the ATF280E has a susceptibility threshold (LET_{th}) of 30 MeV·cm2/mg at V_{cc} min and saturation cross-section of $3x10^{-9}$ bit^{-1}. As a result, designers must take into account the contribution of ionizing radiation on such devices when evaluating the robustness of FPGA-systems. As a matter of fact, designs make seldom use of the whole configuration memory bits. Indeed, only a portion of the entire configuration memory is programmed to implement a given functionality. As result, in order to accurately predict the robustness of FPGA-based systems, it is mandatory to quantify the application-oriented sensitiveness of the device, which can be defined as the number of configuration memory bits that is sensitive to SEE for a certain application mapped on the device.

In this paper we describe the tool, introduced in [6], we developed for evaluating the application-oriented sensitiveness of designs based on ATF280E devices. Moreover, we present the results of the validation process we performed using fault injection, discussing the achieved results. Finally, we present the benefits stemming from the adoption of the application-oriented sensitiveness analysis in evaluating the robustness of a realistic soft processor mapped on the ATF280E device.

II. SEU Cross-section Static Analysis

In order to perform a workload-independent analysis of the configuration memory given a certain design (application), we implemented a static SEU analysis algorithm. The algorithm is based on the configuration memory bits (bitstream) analysis and is aimed at detecting which bits are sensitive for the application once upset. A bit is considered sensitive (or critical) if a change in its value can induce modifications in the implemented circuit that result in application failures. The reliability is then estimated as the number of sensitive bits over the total number of bits in the configuration memory.

The proposed flow is composed by two steps: the resource usage analysis and the circuit sensitiveness analysis. The first step extracts the programmed configuration bits from the bitstream in order to identify which resources are actually used. The second step provides the set of configuration bits that are actually sensitive, on the basis of the considered design and device architecture.

A. Resource Usage Analysis phase

The starting point of the first step is a generic bitstream of a generic circuit. First of all, the data within the bitstream are divided in two parts: the first one containing general information about the device and the configuration mode and the second one that describes the resources configuration. Data contained in the latter are then divided according to the resource type they program. The bits that configure such resources are thus classified in two groups, logic controlling and routing controlling bits.

In order to recognize used resources, the bitstream is split in different windows, each containing the configuration data of a programmed portion of the device. For each window are then

detected the resources used to implement the design. In this manner, the data of each window are again split in data blocks, each of which contains the configuration bits of one resource. Each resource is thus classified as used or not and the correspondent data block is tagged accordingly.

The device is finally divided in sectors, defined as the set of resources that:

- Results equally repeated within the entire device structure.
- Contains all the logic resources that are interconnected by local buses not interrupted by repeaters.

For example a sector could include 4x4 logic cells surrounded by 4 horizontal and 4 vertical repeaters. Adjacent cells are connected by means of direct hard-wires, while longer connections are implemented by local buses. Repeaters connect local buses of neighboring sectors. Global buses (called Express) are used to connect far cells, spanning through multiple sectors.

B. Circuit Sensitiveness Analysis phase

The second step of the proposed flow performs the actual sensitiveness analysis of the application. A map is built, containing the sectors, which the FPGA array is divided in, and, for each resource type involved in the analysis, a tree structure is created to model its architecture. Such a structure is used to verify the criticality conditions for the configuration bits that program a certain resource.

The algorithm runs on every sector with at least one programmed resource. For each resource within the sector, the tree structure of the correct type is filled with the data extracted from the bitstream during the first step, verifying, for each configuration bit, a set of rules that identify the conditions for which an upset bit is critical. In general, a configuration bit is considered critical if, once upset, at least one of the following conditions is verified:

- it modifies the circuit functionality
- it modifies the circuit topology
- it modifies a signal that is an input for another used resource.

The criticality can be further specified as internal or external, depending on the fact that the upset bit causes a modification in the resource it controls or to another resource. While an internal criticality can be ineffective for the whole system, in such a way that it could not be possible observing its effect, an external criticality affects the entire circuit so the considered bit is always classified as critical.

The internal criticality depends on the resource architecture. To detect the internal criticality it is necessary to determine the paths between the point within the resource architecture controlled by the potential critical bit and a resource output. For this reason, we described the internal architecture of the considered resources defining a simple language, the Atmel Resource Description (ARD) language. The description is uniquely based on the FPGA cells architecture and is completely independent both from the device size and from the implemented design. The above mentioned tree structure is built on the basis of this file and describes all the possible paths between a point of the resource architecture and an output. Multiple roots of the tree model the programmable points of the resource, while the leaves are the outputs. The internal nodes represent the conditions according to which the fault may reach an output. So an internal node specifies the bits that have to be configured in order to let the fault pass. Fig. 1 depicts a simple example of a logic cell and the correspondent tree for some of its programming bits.

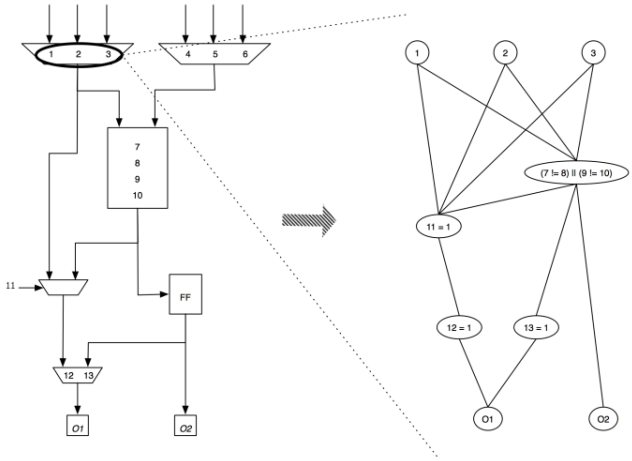


Fig. 1 Logic cell architecture and correspondent tree structure.

An upset bit is considered internally critical either if it changes the logic function of the resource it belongs to or it propagates an error to an output of the resource it belongs to. In the first case, the bit is directly classified as critical. This change indeed corrupts the function implemented by the resource, in such a way that it affects the whole application behavior. In the second case, instead, in order to define whether the bit is critical or not, the external criticality conditions must be verified. In particular, the conditions for which an internal criticality for logic or local routing bits becomes an external criticality are:

- the output corrupted by the upset bit is connected to a bus to which are connected other used resources
- the output corrupted by the upset bit is an input of another used resource through a direct connection.

III. Fault Injection Validation

In order to validate the proposed algorithm, we developed a fault injection platform for Atmel FPGAs in order to perform dynamic analysis on circuits implemented in such devices and compare the results with the ones obtained by static analysis. The aim of this platform is to evaluate the actual effects of SEUs in the configuration memory with respect to a predefined workload for the implemented design.

The fault injection platform is composed by 3 modules, two of which are hardware modules and one is software. Fig. 2 depicts the general architecture. The first module is a board that hosts the Device Under Test (DUT). The FPGA is programmable through an on-board EEPROM that can be accessed by a USB hardware cable and the correspondent software control interface. The second module is the

Monitoring Board that hosts the Monitoring FPGA (MF). These two modules are connected in order to have the test clock and test reset generated by the MF sent to the DUT, and the output of the DUT sent back to the MF. On the latter, a copy of the Design Under Test is mapped along with a comparing module that, clock cycle by clock cycle, compares the outputs of the two designs. While the design implemented in the DUT is corrupted by injecting faults in the configuration memory, the design implemented in the MF is always correct (golden design), thus providing the right outputs. As soon as a discrepancy is detected, the injected fault is classified as critical and we pass to the following one.

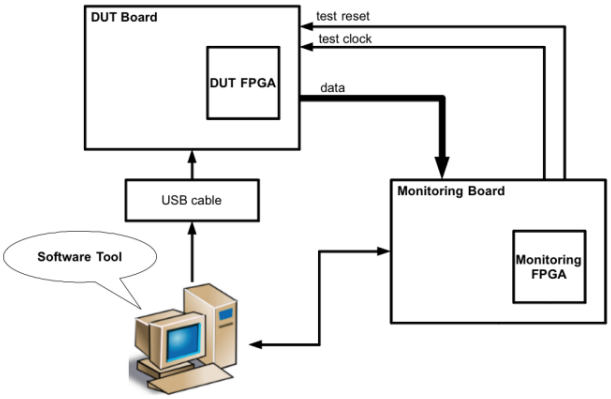


Fig. 2 Fault injection platform architecture.

The orchestration of the whole process is performed by an automatic software tool running on the PC. The software is composed by two components: the "Bitstream Corruption Module" (BCM) and the "Injection and Comparison Module" (ICM).

The BCM takes the golden bitstream as produced by the Atmel tool chain, the fault list and a configuration file. The bitstream corruptor generates a set of faulty bitstreams, each of which contains an SEU fault, located in one of the configuration bits, and a temporary file containing the names of every faulty bitstream along with the corresponding fault number in the reliability report.

The ICM has the control over the two FPGAs. It loads the monitoring design (containing both the golden design and the comparator) on the MF, it loads a bitstream on the DUT and finally it controls the test process by enabling and disabling the DUT master reset, resetting the monitoring design and reading back the result of the test. Fig. 3 shows the flow diagram followed to execute a fault injection campaign.

IV. Experimental Results

A. Fault injection

In order to validate the proposed static analysis algorithm we performed fault injection in a simple design, with both combinational and sequential logic. The Monitoring Board is a Xilinx development kit with a Virtex-II 1000 device. The software we used to program the MF is a tool we already used in [7], which is able to program the FPGA, reset its internal registers and read back their value too. The DUT board has been designed by Atmel and hosts an ATF280E device. The software that programs it has been developed on the basis of the Space Programmer tool by Atmel. It is able to erase and program the on-board EEPROM, as well as to enable and disable the FPGA reset in batch mode.

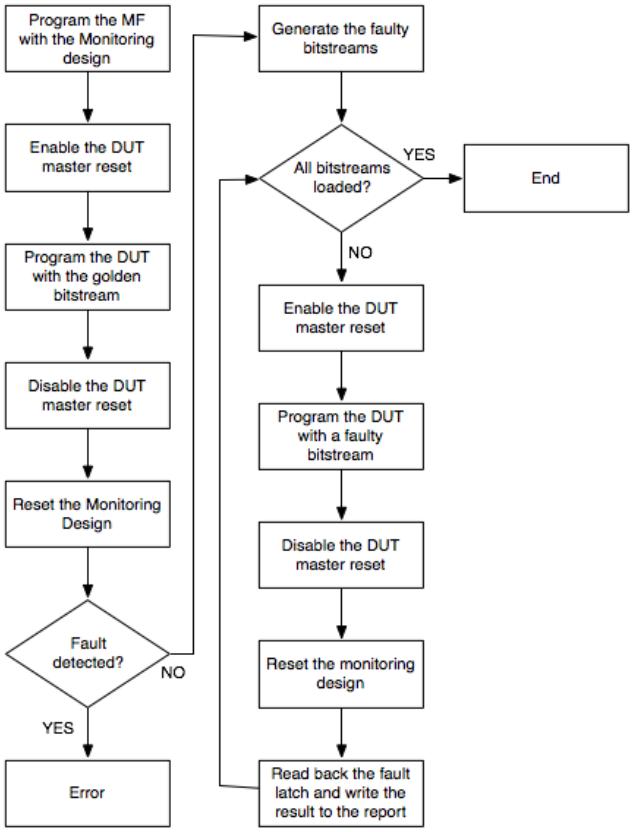


Fig. 3 Fault injection flow-diagram.

We first performed a static analysis of the circuit, independent from the workload, to identify the sensitive bits; we then injected faults in every bit identified as critical by the static analysis and in a set of other 20.000 "not-critical" bits randomly chosen. TABLE I depicts the obtained results. In particular, for each resource type (Detailed Type) we report the number of critical bits identified by the static analysis (S.A.), the number of detected faults after the fault injection campaign (F.I.) and the corresponding percentage. From this experiment we can state that the static analysis is pessimistic, as it overestimates the number of configuration memory bits that are actually critical. However, the analysis is quite accurate; indeed in four out of five cases about 63% of the configuration memory bits identified as sensitive are actually critical during fault injection experiments. Pessimistic predictions for the Express–CC resources are due to the fact that global routing spans over several sectors. In order not to miss some critical bits, and keep low the memory usage of the algorithm analyzing just one sector at a time, all the used

global routing resources are considered critical, without verifying the external criticality conditions. The set of critical bits identified by the fault injection is completely contained in the set of bits recognized as critical by the static analysis. This means that no one of the about 20.000 randomly injected faults is missed by the static analysis algorithm.

TABLE I
FAULT INJECTION RESULTS

Type	Detailed Type	S.A. [#]	F.I. [#]	Percentage [%]
Logic	Core Cell	284	154	54.22
Routing	Local - CC	36	22	61.11
Routing	Express - CC	114	2	1.75
Routing	H.R.	110	75	68.18
Routing	V.R.	124	86	69.35

B. *SEU-sensitiveness analysis for a complex processor core*

Once verified the accuracy of the static analysis algorithm we used it to evaluate the actual SEU sensitiveness for a processor soft core. In particular we implemented the I8051 processor for the ATF280E FPGA and performed the analysis. TABLE II shows the number of critical bits for every resource type, logic Core Cells, local (Local-CC) and global (Express-CC) buses, and for the Horizontal and Vertical Repeaters.

TABLE II
I8051 CRITICAL BITS

Type	Detailed Type	Critical Bits [#]
Logic	Core Cell	448,458
Routing	Local - CC	33,436
Routing	Express - CC	96,586
Routing	H.R.	142,099
Routing	V.R.	140,133

We then compared several sensitiveness estimation metrics, to analyze how application-oriented analysis differs from other methods, based for example on the device cross-section. TABLE III shows the sensitiveness, expressed as the percentage of critical bits over the number of total bits in the configuration memory, with respect to each analyzed metric. In particular, we used four different methods. The first one (Configuration bits) is based on the device cross-section and is basically the number of bits in the configuration memory. The second one (Programmed bits) is based on the number of programmed bits, bits that assumes a value different from the one in an empty bitstream. Finally we used the metric estimated by the static analysis (Sensitive bits) and a slightly more accurate version of the same, that takes into account possible differences in the sensitiveness of bits programmed with a '0' or with a '1'. As reported in [6] and [8] we considered the cells programmed with a '0' 50 times more sensitive than the others. The results show how the first two metrics are respectively too pessimistic and, worse, too optimistic, thus not considering bits that are actually critical. In the second two metrics, instead, we still obtain a pessimistic result, as showed by the fault injection, that has the advantage of being a conservative result, but much more realistic that in the first case.

TABLE III
I8051 SENSITIVENESS ANALYSIS METRICS

Metric	Sensitiveness [%]
Configuration bits	100.00
Programmed bits	10.66
Sensitive bits	49.26
Weighted sensitiveness	38.82

To put the obtained results into the perspective of a realistic mission, we considered an L2 orbit and we evaluated two possible scenarios. If the dependability analysis is performed considering the whole configuration memory as sensitive, due to the characteristics of the considered device, and the L2 orbit, according to CREEME'96 we have that expected upset rate is 1.25×10^{-11} upset/s. On the other hand, if the dependability analysis is performed considering the application-oriented analysis, and thus by taking into account only sensitive bits for the processor core mapped on the FPGA, the expected upset rate is 4.84×10^{-12} upset/s.

From this example we can see that by considering the whole configuration memory worst-case prediction are obtained, which are 2.5x more pessimistic than the predictions that can be obtained by using application-oriented sensitivity analysis.

V. CONCLUSIONS

The SEU cross-section static analysis approach we proposed has been evaluated with respect to dynamic analysis performed by fault injection. Though being more pessimistic, and then conservative, than dynamic analysis, it is able to predict the actual application sensitiveness with an accuracy of 60% on the average and without any false negative. Moreover, we adopted the proposed approach to analyze the SEU cross-section of a complex soft processor, showing that it leads to an estimation improvement of 2.5 times, considering a realistic mission profile.

REFERENCES

[1] L. Rockett, D. Patel, S. Danziger, B. Cronquist, J.J. Wang, "Radiation Hardened FPGA Technology for Space Applications", Proceedings of the IEEE Aerospace Conference, 3-10 March 2007, pp. 1 – 7.

[2] M. Berg, "Fault Tolerance Implementation within SRAM Based FPGA Designs based upon the Increased Level of Single Event Upset Susceptibility", Proceedings of the 12th IEEE International On-Line Testing Symposium 2006.

[3] C. Carmichael, "Triple Module Redundancy Design Techniques for Virtex FPGAs", Xilinx Application Notes XAPP197, 2001.

[4] http://www.actel.com/products/milaero/rtpa3/default.aspx

[5] R. Lacoe, "CMOS scaling, design principles and hardening-by-design methodologies"; Proceedings of the IEEE NSREC Short Course, 2003.

[6] N. Battezzati, F. Decuzzi, M. Violante, M. Briet, "Application-oriented SEU sensitiveness analysis of Atmel rad-hard FPGAs", Proceedings of the 15th IEEE International On-Line Testing Symposium, 2009. IOLTS 2009.

[7] N. Battezzati, S. Gerardin, A. Manuzzato, D. Merodio, A. Paccagnella, C. Poivey, L. Sterpone, M. Violante, "Methodologies to Study Frequency-Dependent Single Event Effects Sensitivity in Flash-Based FPGAs", IEEE Transactions on Nuclear Science, Volume: 56 , Issue: 6 , Part: 1, 2009.

[8] N. Renaud, M. Briet, S. Hachad, G. Rouxel, J.-M. Vrignaud, "ATMEL AT40KEL040 re-programmable FPGA SEU hardened configuration memory", Proceedings of the 8th European Workshop on Radiation and its Effects on Components and Systems (RADECS 2004), 2004.

Appendix B

Annex

Application-oriented SEU sensitiveness analysis of Atmel rad-hard FPGAs

N. Battezzati, F. Decuzzi, M. Violante
Politecnico di Torino

M. Briet
Atmel Corp.

***Abstract*—Radiation-hardened-by-design (RHBD) SRAM-based FPGAs will play a crucial role in providing new generations of satellites with reliable in-flight reconfiguration ability, which is mandatory to enable the successful use of configurable computing in space. RHBD SRAM-based FPGAs sensitiveness against ionizing radiation is normally evaluated resorting to radiation testing, which provides the device cross-section. However, as a matter of fact, applications implemented on such devices use only a portion of the available resources, and the corresponding configuration memory. As a result, application-oriented sensitiveness analysis tools are needed that, by analyzing how the FPGA resources are actually used by a given application, produce application cross-section that is a reliability figure more accurate than device cross section.**

This paper presents a novel application-oriented sensitiveness analysis tool we are developing for the new generation of SRAM-based FPGAs from Atmel: the ATF280E.

I. Introduction

FPGAs are becoming more and more interesting in space and avionic applications, where reconfigurability, high performance and low-power consumption can be fruitfully used to develop innovative systems. However, missions take place in a harsh environment, rich in radiation, which can induce errors within electronic devices. Several approaches have been developed in order to mitigate this problem, from radiation-hardened technologies to make the more robust devices [1], up to design hardening to make the more robust applications implemented using non-robust devices [2]. On the one side, radiation-hardened FPGAs implemented with ad-hoc technologies offer a solid solution to the radiation problem but they have several drawbacks. In particular, their cost with respect to the Commercial-Off-The-Shelf (COTS) counterpart is one or two magnitude order higher and the different technology cannot reach the same area, speed and power performance with respect to COTS devices. For this reasons, even if less robust, it is becoming common practice to use COTS devices [3]. On the other hand, hardening the design using redundancy techniques on a COTS device can lead to a higher cost in terms of area, power and speed. Moreover, implementing known hardening techniques is not trivial, especially for complex designs. No commercial tools, except for [4], are available to the designer for automatically implementing hardening techniques, thus giving the designer the whole burden of hardening the circuits.

In the middle of these two solutions, hardened-by-design [5] FPGAs offer a more robust device even if using the same technology as COTS devices. In particular, redundancy is added in the basic cells that compose the FPGA, thus being transparent to the designer that can implement designs as in rad-hard devices, but with the advantages of COTS ones. However, the commercial technology, is still less robust then the ad-hoc ones, and, even if redundancy is added to the basic FPGA cells, particles over a certain energy can result in soft errors [6]. For this reason, a dependability estimation process is always needed in order to evaluate the actual reliability of a design, taking into account the maximum environmental conditions it can undergo on the basis of the application specifications. It is common use to evaluate the application reliability taking into account only the device reliability, i.e. the robustness of the whole device, independently from the design it implements. However, in the most cases, the application does not occupy the whole device thus leading to pessimistic reliability estimations that consequently may lead to overdesigned applications.

The main contribution of this paper is a novel application-oriented analysis tool aimed at evaluating the sensitiveness of designs implemented on RHBD FPGAs from Atmel. The tool analyzes designs implemented on such device to identify the portion of the configuration memory that is sensitive, i.e., all those configuration memory bits that, if affected by soft errors, result in modifications to the FPGA resource that lead to application failure. Thanks to this tool a precise design-dependent dependability analysis can be performed, and alternative design solutions can be evaluated easily.

In the following sections, after some brief background information (Section II), we detail the proposed approach (Section III and IV) and present experimental results used for evaluating the tool (Section V). Finally, some conclusions and considerations on future works are drawn (Section VI).

II. Background

FPGA on-the-fly configurability is obtained through an on-chip memory that can be volatile or non-volatile, storing device-programming information inside the FPGA chip. SRAM is the technology used for implementing volatile configuration memories, as the ones used in Xilinx, Altera, and Atmel devices, offering high speed and fine granularity

for reconfiguration. However, such memories are sensitive to Single Event Upsets (SEUs) induced by radiation [7]. On the other side, Flash technology is used in non-volatile configuration memories, which being based on floating-gate configuration memory cells, are less sensitive to upsets. However, charge injection can produce transient faults that may temporarily alter the correct operations of the circuit the FPGA implements [8].

In SRAM-based FPGAs, both the configuration and the user memory are sensitive to SEUs. To cope with this problem, developers of space-oriented applications proposed to make their devices more robust using RHBD techniques, obtaining hardened devices according to hardware redundancy techniques [9][10][11].

Several approaches have been then proposed to evaluate the actual memory robustness, as described in [12]. In such devices, however, the SEU cross-section is dominated by the SRAM configuration cells, because user memory elements like flip-flops or hardwired memories occupy a very small area with respect to the whole configuration memory. For this reason sensitiveness evaluation methods focus on this part of the device.

Various approaches can be used in order to evaluate the SEU sensitiveness of an SRAM-based FPGA. We can classify them in three main groups. First of all, radiation testing [13] that consists in irradiating a device with a particle beam and evaluating the SEU cross-section usually reading back the configuration memory looking for the upset cells. This method emulates a real environment but is very expensive. Another possible solution is fault injection [14] that consists in emulating a fault within the configuration memory injecting it manually. Such an approach is cheaper than radiation testing but is very slow. Finally, static analysis methods [15] can be used in order to evaluate which bits of the configuration memory are actually sensitive to SEUs. These methods are very fast but being independent from the input stimuli of the circuit can provide a pessimistic estimation. In several cases this feature can be useful, in order to be conservative and, if carefully designed and with an accurate model of the device such methods are a good approximation of the reality.

III. The Proposed Approach: An Overview

In this section we present an overview of the approach we propose for evaluating SEU sensitiveness of applications implemented in RHBD Atmel FPGAs. The algorithm is based on the configuration memory bits (bitstream) analysis and is aimed at detecting which bits are sensitive for the application once upset. A bit is considered sensitive (or critical) if a change in its value can induce modifications in the implemented circuit that result in application failures. The reliability is then estimated as the number of sensitive bits over the total number of bits in the configuration memory. Moreover, configuration bits are weighted according to their actual sensitiveness that can change for technological reasons.

The proposed flow is composed by two steps: the resource usage analysis and the circuit sensitiveness analysis. The first step extracts the programmed configuration bits from the bitstream in order to identify which resources are actually used. The second step provides the set of configuration bits that are actually sensitive, on the basis of the considered design and device architecture. Fig. 1 depicts the proposed flow, divided in the two above mentioned phases. In the first step, the application bitstream is analyzed looking for the used resources. In the second step, the actual sensitiveness of every configuration bit is evaluated, on the basis of the device architecture, modeled by means of a tree structure, a set of rules, that define the conditions such that a bit can be considered critical, and the set of used resources, identified in the previous step.

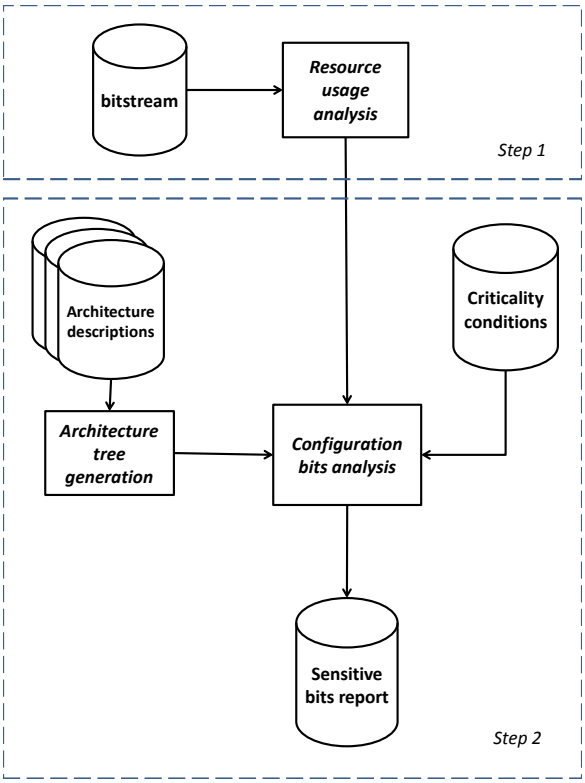


Fig. 1. The proposed flow

More in detail, during the sensitiveness analysis phase, the configuration bits are classified according to the effect they may provoke in the considered design if upset. In general, using a tree representation of the resources architecture and a set of conditions that defines the effect of a bit flip within the resource, a configuration bit is classified as sensitive in two cases. First of all, a bit is considered critical if, once upset, it corrupts the function implemented by the resource it belongs to. Moreover, a bit is also classified as critical if, once upset, although not directly affecting the resource itself, the error it produces can affect one of the connected resources.

IV. Tool Implementation Description

As mentioned in the previous section, the software process is composed by two main steps. The first step detects the resources used to implement the design and it generates the information about the used device area and the corresponding

configuration memory. The output of the first step becomes an input to the second one that evaluates the sensitiveness of the desired circuit. In this section the two phases are detailed.

A. Resource Usage Analysis phase

The starting point of the first step is a generic bitstream of a generic circuit. First of all, the data within the bitstream are divided in two parts: the first one containing general information about the device and the configuration mode and the second one that describes the resources configuration. Data contained in the latter are then divided according to the resource type they program. Two types of resources are considered in this analysis, logic cells and bus repeaters. Logic cells implement combinational and sequential logic functions and partially control local routing structures. Bus repeaters regenerate and switch local and global connection signals thus being at the base of the routing architecture of the FPGA.

In order to recognize programmed resources, the bitstream is split in different blocks, each containing the configuration data of one resource. Each resource is thus classified as programmed or not and the correspondent data block is tagged accordingly. In this manner, we can generate a map of the device, with the information about programmed and un-programmed resources.

Finally, the device map is divided in sectors, defined as the set of resources that:

- results equally repeated within the entire device structure
- contains all the logic resources that are interconnected by local buses not interrupted by repeaters.

Fig. 2 shows a basic sector. In the example a 2x2 logic cells matrix is surrounded by two repeaters for each side. Each repeater connects local bus segments (thin lines) of two neighboring sectors. Thick lines represent instead direct connections among adjacent logic cells. The difference between local buses and direct connections lies in the fact that the first ones are programmable through configuration memory bits while the second ones are not. Long global interconnections are not represented in Fig. 2 not to confuse the picture, but they can be considered as local interconnections spanning two adjacent sectors.

A sector is considered used, and thus analyzed in the second step, if at least one of its resources is programmed.

B. Circuit Sensitiveness Analysis phase

The second step of the proposed flow performs the actual sensitiveness analysis of a circuit. Using the map described in the previous section, the FPGA array is divided in sectors and, for each resource type involved in the analysis, a tree structure is created to model its architecture. Such a structure is used to verify the criticality conditions for the configuration bits that program a certain resource.

The algorithm runs on every sector with at least one programmed resource. For each resource within the sector, the tree structure of the correct type is filled with the data extracted from the bitstream during the first step, verifying, for each configuration bit, a set of rules that identify the conditions for which an upset bit is critical. In general, a configuration bit is considered critical if, once upset, at least one of the following conditions is verified:

- it modifies the circuit functionality
- it modifies the circuit topology
- it modifies a signal that is an input for another used resource.

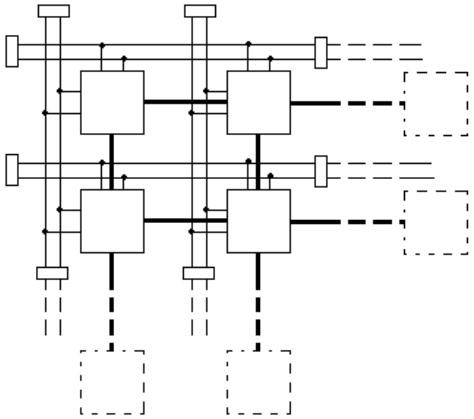


Fig. 2. The elementary sector architecture

The criticality can be further specified as *internal* or *external*, depending on the fact that the upset bit causes a modification in the resource it controls or to another one. While an internal criticality can be ineffective for the whole system, in such a way that it could not be possible observing its effect, an external criticality affects the entire circuit so that the considered bit is always classified as critical. Indeed, considering that designs mapped in RHBD devices usually do not implement any additional hardening strategy, if an error propagates outside a resource it can not be further masked, unless by logical masking, which is not considered in a static analysis.

The analysis algorithm acts on two levels. First of all, the internal criticalities for the resource under test are identified. In the second level are analyzed the connections between the resource under test and the other ones, upgrading the internal criticalities to external criticalities when required. This flow is applied to every configuration bit of each resource in the sector.

The internal criticality depends on the resource architecture. To detect the internal criticality it is necessary to determine the paths between the point within the resource architecture controlled by the potential critical bit and a resource output. For this reason, the Atmel Resource Description (ARD) files are used to describe the internal architecture of the considered resources. The description is uniquely based on the FPGA cells architecture and is completely independent both from the device size and from the implemented design. The tool is flexible enough that new kinds of resources can be added later, just describing their architecture by an ARD file. Five kinds of elements have been defined in order to describe a resource:

- *SET*: describes a programmable element whose output exclusively depends on one input, that is selected by its configuration bits
- *OUT*: element that labels a resource output

- *LUT*: describes a programmable element whose output depends on one or more inputs, according to a function defined by its configuration bits
- *Common path (CP)*: describes the path between two points of the resource architecture
- *Root (R)*: describes, for each programmable point, the set of conditions that determine its criticality.

Once this file is parsed, for each resource type we create a tree structure that describes all the possible paths between a point of the resource architecture and an output. Multiple roots of the tree model the programmable points of the resource, while the leaves are the outputs. The internal nodes represent the points of the architecture a signal has to traverse in order to reach the output from the point controlled by the potentially critical bit. An internal node thus specifies the bits that have to be configured in order to let the signal pass. An upset bit is considered internally critical if at least one of the following conditions is verified:

1. it changes the logic function of the resource it belongs to
2. it propagates an error to an output of the resource it belongs to.

In the first case the bit is directly classified as critical. This change indeed corrupts the function implemented by the resource, in such a way that is not necessary to check if another one is connected to it. In particular, the potential directly critical bits are:

a. all the bits used to configure the resource in the current design
b. all the bits that, if upset, cause a violation of an ARD element definition.

In the second case, instead, in order to define whether the bit is critical or not, the external criticality conditions must be verified. In particular, the conditions for which an internal criticality becomes an external criticality are:

1. the output corrupted by the upset bit is connected to a bus to which are connected other used resources
2. the output corrupted by the upset bit is an input of another used resource through a direct connection.

Fig. 3 shows an example of logic cell architecture, based on ARD elements, highlighting the configuration bits that control each of them. The picture does not represent the Atmel logic cell but is a simplification that contains all the blocks needed to explain the use of ARD elements involved in the proposed analysis flow. Programmable switches are controlled by configuration memory bits that can be critical. The SET element in the upper part of the cell is configured by 4 bits that can be active only one at a time. Each memory configuration in which two or more of those bits are programmed at the same time violates the SET definition thus being critical. The same happens for the SET element on the lower part of the cell. The LUT element is controlled by 8 bits that define the output value on the basis of the three input signals. Every bit of this element, if upset, is critical because changes the function implemented by the LUT. In the example, the sequential functions are implemented by hardwired Flip Flops (FF) that are not programmable. Finally, the two bits in the top-right corner of the cell are not directly critical, because, once upset, do not necessarily induce an error in the resource. If the LUT input they control is not used, indeed, an upset in those bits is not critical. To determine if they are sensitive bits or not, is thus necessary to verify if the error they induce can be propagated toward one of the resource OUT elements. This operation is performed traversing all the *common paths* in the resource tree structure, until a leaf is reached. In particular, a path is opened or closed according to the values of the conditions in the root element correspondent to the bit under analysis. If the error can propagate to an output, the external criticality rules are then verified, in order to define whether the bit is actually sensitive or not.

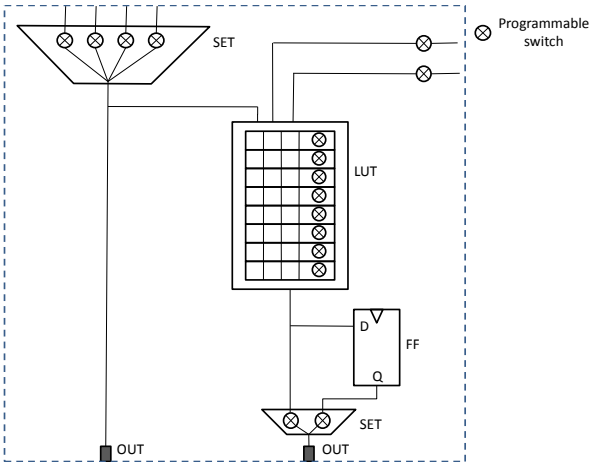


Fig. 3. Example logic cell described with ARD elements

V. Experimental Results

In order to evaluate the proposed algorithm we ran the tool on several benchmark circuits of the ITC'99 set [16]. Such circuits have been designed for evaluating Design For Testability (DFT) techniques but, because of the heterogeneity of the logic structures they implement, can be very useful for validating and stimulating the algorithm we propose with real designs. The circuits we used range from simple gate sets up to complex processor cores.

As a case study we selected two RHBD FPGA architectures from Atmel Corporation, the AT40KEL040 and the ATF280E. We then implemented the ITC circuits on one device from both the two families using the FIGARO tool presented in [17]. Table I details the characteristics of the benchmark circuits implemented in each device, as number of occupied logic cells, flip flops and routing resources (local buses).

After implementing the benchmark circuits in the two devices, we analyzed the produced bitstreams with the proposed algorithm. Besides validating the approach against real and complex designs, this operation allows also comparing different sensitiveness analysis strategies. The SRAM bits, even if hardened-by-design, over a certain radiation energy become sensitive and the mitigation techniques can fail. For this reason they are the real critical points of the application and should be taken into account during a reliability estimation process. In particular we compared three possible metrics for evaluating design

sensitiveness: the percentage of programmed bits in the configuration memory, the percentage of used resources, as the set of logic and routing ones, and the percentage of weighted sensitive bits as produced by the proposed approach.

In Table II we present the results of the three sensitiveness estimation strategies obtained on the implemented benchmarks from ITC'99. The percentage of programmed bits is simply the ratio between the number of bits set to '1' within the design's bitstream and the total number of bits in the configuration memory. The percentage of used resources is obtained by the reports of the FIGARO tool. The result of the proposed algorithm is instead expressed by the formula:

$$sens = \frac{b_{S\max} \cdot \sigma + b_{S\min}}{b_{tot} \cdot \sigma}$$

where b_{Smax} is the number of sensitive bits with a higher probability to be upset and b_{Smin} is the number of sensitive bits with a lower susceptibility to upset. The *sensitiveness factor* (σ) is then defined as the ratio between the sensitiveness of the most sensitive bits and the sensitiveness of the least sensitive ones, as expressed in the following formula:

$$\sigma = \frac{\max\{ w_0, w_1\}}{\min\{ w_0, w_1\}}$$

where w_0 is the sensitiveness of configuration SRAM cells containing a '0' (not programmed) and w_1 is the sensitiveness of configuration SRAM cells containing a '1' (programmed). Indeed, as reported in [18], for technological reasons un-programmed configuration cells are 50 times more sensitive than programmed ones.

In this scenario, the metric based on the number of programmed bits leads to completely unrealistic estimations, leaving out all those bits that, even if not used, can induce misbehaviors. Moreover, just these bits are the most sensitive. On the other hand, resource usage estimation based on FIGARO reports provides a still rough result. It is better than the first one because it intrinsically takes into account both programmed and not programmed bits, but can not provide a fine analysis of the actual sensitiveness, not being able to tell apart the contribution of programmed and not programmed bits. Finally, the finest estimation is provided by the proposed approach, that considers only really critical bits and it can also weight them according to their actual measured sensitiveness. In particular, the reader can observe how the last benchmark design, built as a chain of 40 B08 circuits, occupies almost the entire logic area of the used AT40KEL040 device; however, the number of sensitive logic bits estimated by the proposed analysis flow is more than 25% lower than the number of used logic resources. Indeed, it is not completely correct estimating the actual design sensitiveness with respect to used resources because critical parts of the FPGA are the configuration bits and not the logic and routing resources themselves. For this reason, the approach we have proposed evaluates the application SEU cross-section on the basis of actually critical configuration bits.

In the end, Table III shows the computation time requested by the developed tool to perform the analysis on the proposed benchmark circuits.

VI. Conclusions And Future Works

In this paper we developed a novel software flow for evaluating the actual SEU sensitiveness of designs implemented in RHBD SRAM-based FPGAs. RHBD FPGA applications still need reliability estimation and, in order to be as more precise as possible and not overdesign the application, an accurate analysis of the design is needed. The approach we propose is based on static analysis, that is faster and cheaper than radiation testing and fault injection techniques. Moreover it provides a more conservative estimation. Experimental results proved the effectiveness of the proposed flow on both AT40KEL040 and AT280E Atmel devices highlighting the differences between a rough evaluation based on resource usage estimation and a more accurate method based on configuration bits analysis.

Future works will be aimed at developing a fault injection platform for validating the proposed approach and integrating it to perform dynamic analysis.

References

[1] L. Rockett, D. Patel, S. Danziger, B. Cronquist, J.J. Wang, "Radiation Hardened FPGA Technology for Space Applications", Proceedings of the IEEE Aerospace Conference, 3-10 March 2007, pp. 1 – 7.

[2] M. Berg, "Fault Tolerance Implementation within SRAM Based FPGA Designs based upon the Increased Level of Single Event Upset Susceptibility", Proceedings of the 12th IEEE International On-Line Testing Symposium 2006.

[3] A.H. Johnston, "Radiation effects in advanced microelectronics technologies", IEEE Transactions on Nuclear Science, Volume 45, Issue 3, Part 3, June 1998, pp. 1339 – 1354.

[4] C. Carmichael, "Triple Module Redundancy Design Techniques for Virtex FPGAs", Xilinx Application Notes XAPP197, 2001.

[5] R. Lacoe, "CMOS scaling, design principles and hardening-by-design methodologies"; Proceedings of the IEEE NSREC Short Course, 2003.

[6] J. E. Knudsen, L. T. Clark, "An Area and Power Efficient Radiation Hardened by Design Flip-Flop", IEEE Transactions on Nuclear Science, Volume 53, Number 6, December 2006, pp. 3392 – 3399.

[7] JEDEC standard JESD89, Measurement and Reporting of Alpha Particles and Terrestrial Cosmic Ray-Induced Soft Errors in Semiconductor Devices, Aug. 2001.

[8] N. Battezzati, S. Gerardin, A. Manuzzato, A. Paccagnella, S. Rezgui, L. Sterpone, M. Violante, "On the Evaluation of Radiation-Induced Transient Faults in Flash-Based FPGAs", Proceedings of the 14th IEEE International On-Line Testing Symposium, 7-9 July 2008, pp.135-140.

[9] Z. Huang, H. Liang, "A New Radiation Hardened By Design Latch for Ultra-Deep-Sub-Micron Technologies", Proceedings of the 14th IEEE International On-Line Testing Symposium 2008, pp. 175 – 176.

[10] L. Jacunski, S. Doyle, D. Jallice, S. Haddad, T. Scott, "SEU Immunity: The Effects of Scaling on the Peripheral Circuits of SRAMs", IEEE Transactions on Nuclear Science, Volume 41, Number 6, December 1994, pp. 2272 – 2276.

[11] T. Calin, M. Nicolaidis, R. Velazco, "Upset Hardened Memory Design for Submicron CMOS Technology", IEEE Transactions on Nuclear Science, Volume 43, Number 6, December 1996, pp. 2274 – 2278.

[12] R.N. Nowlin, C.S. Begay, R.R. Parker, M.P. Garrett, T.D. Penner, "Radiation Hardness of Hardened-By-Design SRAMs in Commercial Foundry Technologies", Proceedings of IEEE Radiation Effects Data Workshop, July 2006, pp. 136 – 143.

[13] M. Alderighi, F. Casini, S. D'Angelo, F. Faure, M. Mancini, S. Pastore, G.R. Sechi, R. Velazco, " Radiation test methodology for SRAM-based FPGAs by using THESIC+", Proceedings of the 9th IEEE On-Line Testing Symposium, 7-9 July 2003, pp. 162.

[14] M. Alderighi, F. Casini, S. D'Angelo, M. Mancini, S. Pastore, G.R. Sechi, "Evaluation of Single Event Upset Mitigation Schemes for SRAM based FPGAs using the FLIPPER Fault Injection Platform", Proceedings of the

22nd IEEE International Symposium on Defect and Fault-Tolerance in VLSI Systems, 26-28 September 2007, pp. 105 – 113.
[15] L. Sterpone, M. Violante, "Static and Dynamic Analysis of SEU Effects in SRAM-Based FPGAs" Proceedings of the 12th IEEE European Test Symposium, 20-24 May 2007, pp. 159 – 164.
[16] D. Bhovsar, "Itc 99 panels", IEEE Design & Test of Computers, Volume 16, Issue 4, Oct.-Dec. 1999, pp. 96 – 99.
[17] K. Nasi, T. Karouhalis, M. Danek, Z. Pohl, "FIGARO - an automatic tool flow for designs with dynamic reconfiguration", Proceedings of the International Conference on Field Programmable Logic and Applications, 2005. 24-26 Aug. 2005, pp. 590 – 593.
[18] N. Renaud, M. Briet, S. Hachad, G. Rouxel, J.-M. Vrignaud, "ATMEL AT40KEL040 re-programmable FPGA SEU hardened configuration memory", Proceedings of the 8th European Workshop on Radiation and its Effects on Components and Systems (RADECS 2004), 2004.

TABLE I
BENCHMARK DESIGNS CHARACTERISTICS ON ATMEL DEVICES

Circuit	AT40KEL040			AT280E		
	Logic cells[#]	Flip Flops[#]	Routing resources[#]	Logic cells[#]	Flip Flops[#]	Routing resources[#]
B01	11	10	20	13	10	43
B03	58	34	127	65	34	158
B04	176	66	669	185	66	761
B08	60	21	239	79	21	306
B11	139	35	592	152	35	606
B12	347	108	1335	420	108	1676
B14	1302	215	7020	1651	215	10233
B08_chain	1923	840	6298	1948	840	7000

TABLE II
COMPARISON BETWEEN DIFFERENT SENSITIVENESS ESTIMATION STRATEGIES

Circuit	AT40KEL040			AT280E		
	Programmed bits[%]	Figaro resources[%]	Weighted Sensitiveness[%]	Programmed bits[%]	Figaro resources[%]	Weighted Sensitiveness[%]
B01	0.07	0.32	0.22	0.01	0.07	0.05
B03	0.36	1.79	1.09	0.07	0.33	0.20
B04	1.17	6.71	4.15	0.20	1.16	0.80
B08	0.46	2.33	1.43	0.10	0.48	0.31
B11	0.98	5.57	3.44	0.17	0.77	0.60
B12	2.50	13.20	8.03	0.48	2.61	1.62
B14	9.42	58.23	33.98	2.03	12.77	7.78
B08_chain	14.32	68.88	42.64	2.40	11.58	7.46

TABLE III
PROPOSED METHOD EXECUTION TIME FOR PROPOSED BENCHMARK CIRCUITS

Circuit	AT40KEL040 [ms]	AT280E [ms]
B01	0.028	0.100
B03	0.084	0.116
B04	0.172	0.624
B08	0.112	0.152
B11	0.136	0.432
B12	0.228	0.576
B14	0.436	1.780
B08_chain	0.452	1.248

Appendix C

Annex

Use of macros for At40k FPGAs

Study about the area and reliability optimization

DATA	AUTHORS	REVISION
26/01/2010	Decuzzi Filomena Merodio Codinachs David	1.0

Reference documentation

[1] Atmel, “AT40K Macro Library”, June 2002
[2] Decuzzi Filomena, Massimo Violante, “Configuration memory analyzer tool”

Introduction

This study is aimed at analyzing the performance of the use of macros for the AT40k family of FPGAs. With respect to [1] the study is performed on dynamic macros. These ones, differently from functional macros, are designed to allow user specification of any desired functionality attached as an attribute, via an equation string on the symbol. Contrary to the description in [1] of the macro library for the AT40K family, the FGEN2xx components are not used to implement a circuit.

However the use of FGEN2xx and the other macros is allowed by means of the Figaro’s “macro generator”. This tool allows designers creating their user-defined macros by specifying parameters and storing them in their own user libraries.

Dynamic Macro

The use of macros gives the user better control over the implementation of specific functions in a single core cell. It can also be used to simplify the design entry process. According to [1] the dynamic macros available for the At40k FPGAs are showed in Table 1.

Logical Function	**Description**
FGEN1	*n* input function generator ($1 \leq n \leq 4$)
FGEN1F	*n* input function generator with combinatorial feedback ($1 \leq n \leq 3$)
FGEN1FT	*n* input function generator with combinatorial feedback followed by tri-state buffer ($1 \leq n \leq 3$)
FGEN1R	*n* input function generator followed by a register ($1 \leq n \leq 4$)
FGEN1RF	*n* input function generator with registered feedback ($1 \leq n \leq 3$)
FGEN1RFT	*n* input function generator with registered feedback followed by tri-state buffer($1 \leq n \leq 3$)
FGEN1RT	*n* input function generator followed by a register and tri-state buffer ($1 \leq n \leq 4$)
FGEN1T	*n* input function generator followed by a tri-state buffer ($1 \leq n \leq 4$)
FGEN2	Two *n* input function generators ($1 \leq n \leq 3$)
FGEN2F	Two *n* input function generators with combinatorial feedback on 1-output($1 \leq n \leq 2$)
FGEN2FT	Two *n* input function generators with combinatorial feedback followed by tri-state buffer on 1-output ($1 \leq n \leq 2$)
FGEN2R	Two *n* input function generators with 1-output registered and the other combinatorial ($1 \leq n \leq 3$)
FGEN2RF	Two *n* input function generators with 1-output registered and feedback ($1 \leq n \leq 2$)
FGEN2RFT	Two *n* input function generators with 1-output registered, tri-stated and feedback ($1 \leq n \leq 2$)
FGEN2RT	Two *n* input function generators with 1-output registered and tri-stated ($1 \leq n \leq 3$)
FGEN2T	Two *n* input function generator with 1-output tri-stated ($1 \leq n \leq 3$)
MGEN	Two *3*-input function generators
MGENR	Two 3-input function generators with 1-output registered
MGENRT	Two 3-input function generators with 1-output registered and tri-stated
MGENT	Two 3-input function generator with 1-output tri-stated

Table 1: Logical Function

The study shows that the logical functions FGEN2xx and MGENxx are not used neither during the synthesis phase nor the place and route one. In particular, in the technology library of Precision only the FGEN1 macro is defined; although the Figaro tool is able to use other macros of FGEN1xx kind during the placement step.
In this way Figaro performs some optimization like using only one FGEN1R macro instead two FGEN1 macros produced by Precision, in order to implement a combinational function followed by a FFD.

Use of the FGEN2 MACRO

In some cases, two FGEN1 macros can be optimized in one FGEN2 macro; in particular, if the resulting functions use at most 3 inputs this substitution is possible and brings to a 100% area improvement (see Figure 1). Table 2 shows all the different configurations for that allow the use of one FGEN2 macro in place of two FGEN1 macros. Black dots represent the function inputs, while white circles are the functions themselves. Each input is connected to the functions it feeds by means of an arrow.

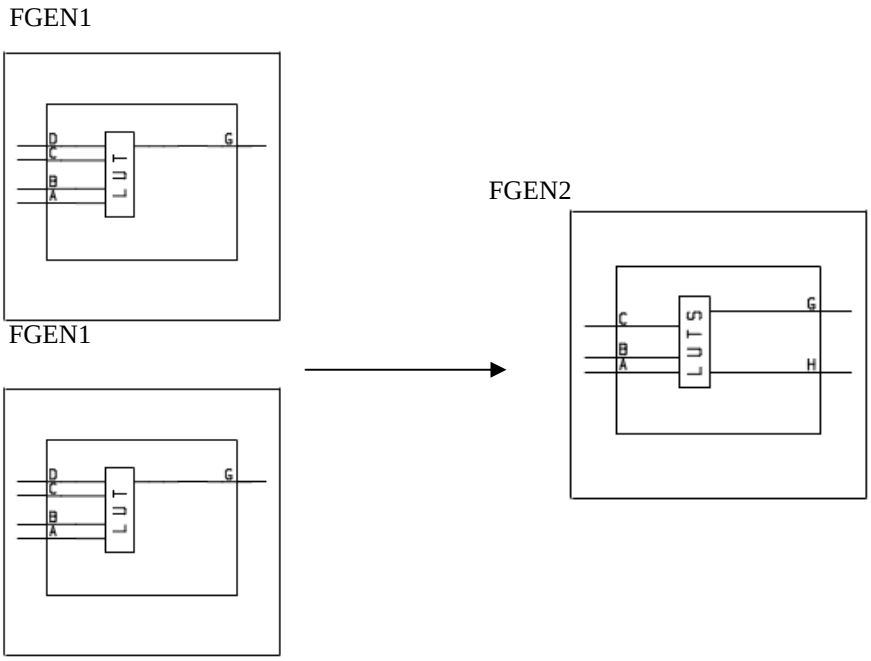


Figure 1: Optimization scheme

Purpose of study

The purpose of this study is to illustrate the improvements that would be brought using FGEN2xx macros. MGENxx macros are out of the scope of this report.
The analysis and the comments are made on the results of:

- Figaro statistics
- Memory usage and criticality from Susanna
- Use of logic cell from Susanna

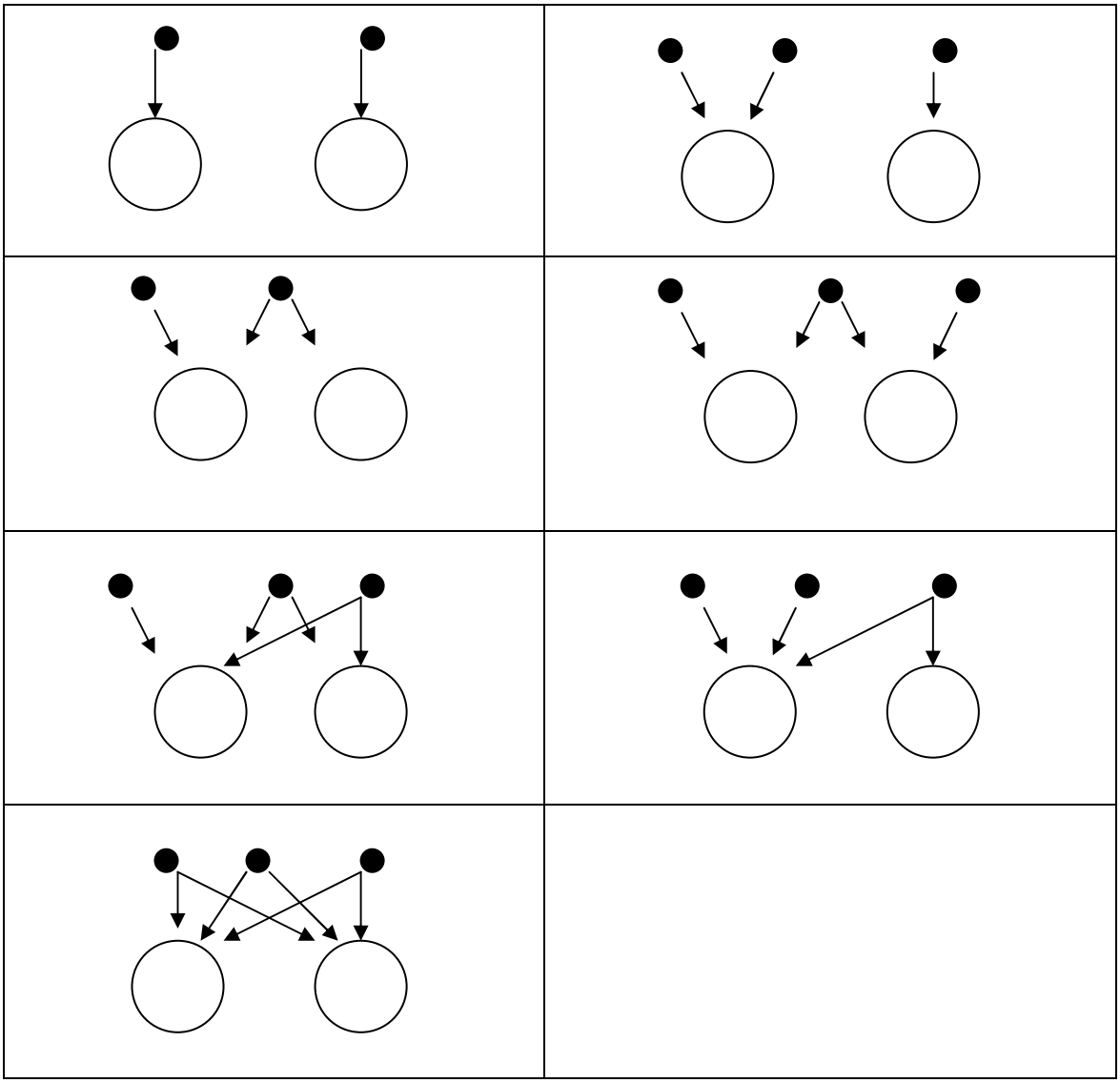


Table 2: Configurations enabling the FGEN1 to FGEN2 substitution

Experimental results

The study has been completed with the analysis of three circuits, showing the application of the ideas explained before. For each circuit under test statistics about the device resources usage and configuration memory sensitivity are reported. The configuration memory is analyzed by the Susanna tool [2] in order to detect the critical bits and the occupied area of the device. The results are organized in three tables:

- Table A: Figaro statistic
- Table B: Memory usage and criticality from Susanna
- Table C: Use of logic cell from Susanna

Table A summarizes the static usage of the device provided by the Figaro statistics file. Table B shows the configuration memory usage, in order to implement the design, and the results of the static analysis as the number of critical bits, then classified on basis of the resources they belong to.

Table C shows the task of the critical Core Cells. They are mainly classified as programmed or not programmed. A not programmed Core Cell is critical if a bit flip in its configuration bits induces an error on a used resource. Besides, the programmed Core Cells are classified according the implemented function. In particular are reported the Routing Core Cells, that use the LUT to perform the routing.

And-Or circuit

The first circuit implement a couple of combinational gate with shared inputs. Figure 2 shows the circuit.

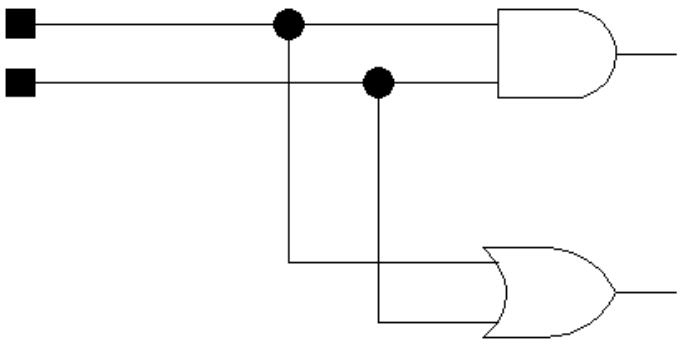


Figure 2: And-Or circuit

The implementation of this circuit needs two Core Cells to implement the gates in two distinct resources where each Core Cell uses only one LUT to perform the function. While the optimized implementation use a user-defined FGEN2 macro, where the output are:

G= A*B H=A+B

AND_OR macro functions

The two designs are synthesized and the relative bitstreams are analyzed by Susanna. Table A-1 shows that the used Logic Cell is only one in the optimized implementation

versus the 2 Core Cells used in the original one. This area saving corresponds to a decrease of programmed bits in the configuration memory. Table B-1 shows that the sensitive bits for the CCs estimated using Susanna decrease by 15 bits while the programmed bits decrease by only 3 bits.

	Original	**Optimized**
Number of Macros	6	5
Number of IO Macros	4	4
Number of Logic only Macros	2	1
Number of used Logic Cells	2	1
Number of Flip-Flops	0	0
Number of Gates	2	1

Table A-1:Figaro statistics

	Original	**Optimized**
Total programmed bits	15	12
Total sensitive bits	52	29
CC's bits	44	29
HR's bits	8	0
VR's bits	0	0

Table B-1: Susanna reliability estimation

	Original	**Optimized**
Total number of critical CCs	2	1
Programmed critical CCs	2	1
Used for Routing	0	0
Using the LUT	0	0
Used for Logic	2	1
Used for Logic and Routing	0	0
Not Programmed critical CCs	0	0

Table C-1: Core Cell functions

Bits_counter circuit

The second used circuit implements a bits counter. Figure 3 shows the RTL schematics. The *counter* block implements a 4 bits counter. Figure 4 shows the technology implementation after the synthesis, reported by Precision.

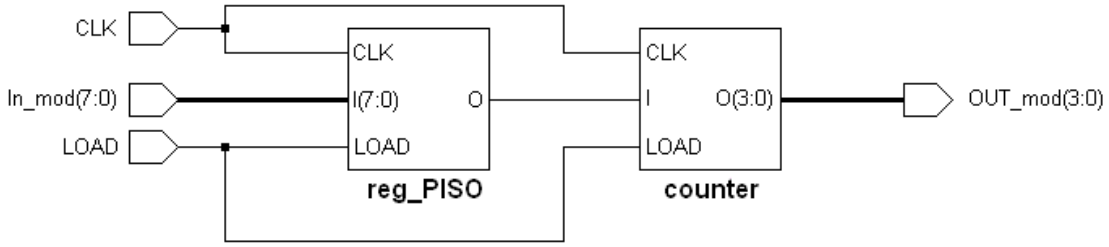


Figure 3: Bits_counter RTL schematic

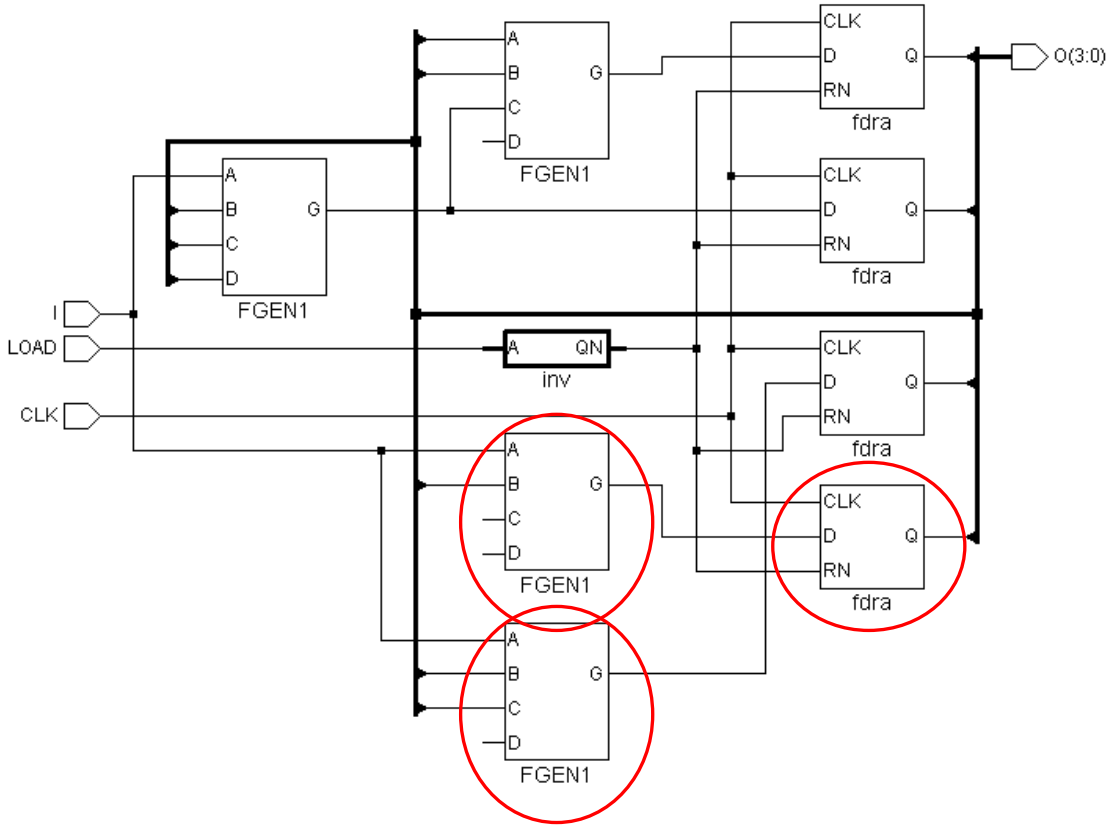


Figure 4: Counter technology schematic

During the place and route phase Figaro performs an optimization using FGEN1R macros to implement each couple of FGEN1 and FDRA given by Precision. The implemented circuit occupy 16 Core Cells for the logic. In particular, the counter occupies 5 Core Cells, where 4 CCs implement FGEN1R macros and 1 a FGEN macro; so each CC use only one LUT.

The optimization joins together the two combinational functions that share two inputs and one FDRA, as highlighted in Figure 4. In this case the optimization is aimed at evaluating the device cross-section with different use of logical resources. In fact, although the macro joins two combinational functions, the number of used CCs depends on the implemented DFF, because the Core Cell architecture allows to register only one LUT output. By means of a second FGEN2R macro the DFF for the second LUT of the first macro and the **inv** component are implemented in the same Core Cell.

The functions in the two FGEN2R macros are the following:

```
G= B*!C+A*!B*C+!A*B (.FF)
H= A*!C+!A*C'
```

MACRO1CLK_COUNT macro functions

```
G= A (.FF)
H=!B
```

DFF_NOT macro functions

Table A-2 and Table B-2: Susanna reliability show how the optimization, even if doesn't reduce the amount of used resources to implement the logic, in terms of number of CCs, it reduce the amount of sensitive configuration bits by means of a redistribution of the logic in the CCs. In fact, the optimization leads to occupy as much as possible of the capability of a logic cell. For this reason the amount of configuration data not directly used to implement the function, but that surround the logic and could induce an error on it, is thus reduced.

The results in Table B-2 shows that for the optimized circuit the amount of sensitive bits for the Core Cells is reduced even on this small circuit that occupies less then 1% of the total FPGA area. Bigger circuits could offer a much higher possibility of optimization.

	Original	**Optimized**
Number of Macros	27	27
Number of IO Macros	14	14
Number of Logic only Macros	13	13
Number of used Logic Cells	16	16
Number of Flip-Flops	12	12
Number of Gates	1	1

Table A-2:Figaro statistics

	Original	**Optimized**
Total programmed bits	232	223
Total sensitive bits	977	969
CCs	740	727
HRs	144	131
VRs	93	111

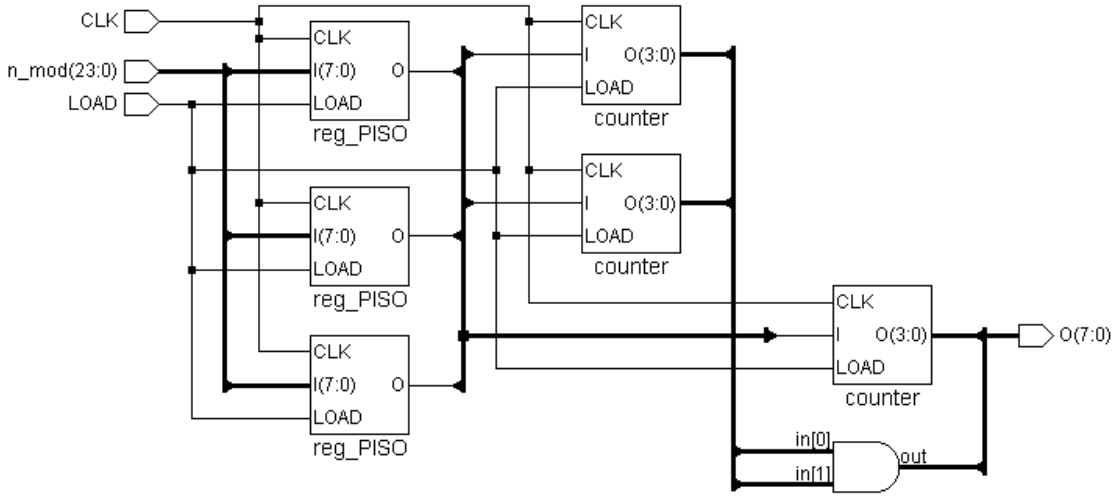
Table B-2: Susanna reliability estimation

	Original	**Optimized**
Total number of critical CCs	25	28
Programmed critical CCs	19	19
Used for Routing	6	6
Using the LUT	3	2
Used for Logic	10	9
Used for Logic and Routing	3	4
Not Programmed critical CCs	6	9

Table C-2: Core Cell functions

Multiple bits_counter circuit

The third case of study is composed by a set of three circuits bits_counter described above and a set of four AND gate that perform the logic product of the outputs of the first two circuits.



In this case the optimization is still performed on two logical functions belonging to the **counter** circuit and, moreover, between one DFF t and one AND gate used for the outputs. The functions of the two FGEN2R user macros are the following:

G= B*!C+A*!B*C+!A*B (.FF) H= A*!C+!A*C'

MACRO1CLK_COUNT macro functions

G= A (.FF) H= B*C

FF_AND macro functions

The optimized circuit use 2 Core Cells less than the original one to implement the logic, see Table C-3. This result leads to a decrease of more than 6% of criticality estimated by Susanna on 1.31% of critical area of the configuration memory, see Table B-3: Susanna reliability estimation. This reliability gain is underestimated because the optimized implementation uses a number of logic resources to perform routing larger than the original one. Different trials have showed that a programmed LUT have a wide impact on the reliability of the device. Table C-3 shows than the CCs programmed to perform the routing are 9 in the optimized version versus 3 of the original one.

	Original	Optimized
Number of Macros	77	75
Number of IO Macros	34	34
Number of Logic only Macros	43	41
Number of used Logic Cells	46	50
Number of Flip-Flops	36	36
Number of Gates	7	5

Table A-3: Figaro statistics

	Original	Optimized
Total programmed bits	758	780
Total sensitive bits	3721	3487
CCs	2701	2555
HRs	491	370
VRs	529	562

Table B-3: Susanna reliability estimation

	Original	Optimized
Total number of critical CCs	357	226
Programmed critical CCs	63	65
Used for Routing	20	24
Using the LUT	3	9
Used for Logic	30	25
Used for Logic and Routing	13	16
Not Programmed critical CCs	294	161

Table C-3: Core Cell functions

Conclusions

The different experiments show the advantage of the use of optimized macro in terms of area and sensitiveness. In particular, for the **And_or** and the **Multiple bits_counter** circuits the use of FGEN2xx macro lead to both area and estimated criticality reduction. While the **Bits_counter** circuit shows that though the use of FGEN2xx doesn't modify the area occupied from the logic it takes advantage on the cross-section.

Possible causes

On of the possible causes of such a sub-optimal behaviour of the Precision-Figaro flow is that the FGEN2xx macros are not described in the technology library used by Precision to perform the synthesis. In fact, the command **get_libs_cells** doesn't show dynamic macros other than FGEN1.

Future works

Developing a tool to automatically explore the circuit netlist looking for possible optimizations, it would be possible to predict the real impact of the ideas illustrated in this study, also in the case of realistic designs.

Appendix D

Annex

Study on the Optimization of Circuits Implemented in Atmel FPGAs

Filomena Decuzzi, David Merodio Codinachs
ESA/ESTEC

Niccolò Battezzati
Politecnico di Torino

March 15, 2010

1 Introduction

According to the analysis reported in [1], we developed an automatic process to analyze EDIF netlists of realistic circuits implemented in Atmel FPGAs and to report possible optimizations that can be taken into account before the mapping step. The analysis process has been developed on the basis of the Polito Automatic Hardening Tool (PAHT).

In particular, the EDIF netlist generated by Mentor Precision RTL tool [2] is analyzed by PAHT, identifying all the possible pairs of instances that could be optimized substituting an FGEN2xx macro. The pairs of instances to be optimized are chosen in order to maximize the final number of optimizations.

2 Experimental results

We evaluated the process on a set of benchmark circuits, from simple combinational ones up to complex processors.
The first designs have been taken from the ITC'99 benchmark [3]. Table 1 reports their name and the respective description. The EDIF netlist has been mapped in an AT40K device using Atmel Figaro ids8.2.2 tool. Besides, we analyzed the Intel 8051 micro-controller core and the Leon2 core without the cache memory, mapping the EDIF netlists in an ATF280E device using Atmel Figaro ids9.0.2 tool.

We then analyzed the EDIF netlists by means of the automatic process based on the PAHT tool, independently from the device they are mapped in. The results are presented in Table 2 for the designs mapped on the AT40K and in Table 3 for the cores mapped on the ATF280E.
For each analyzed circuit we report the total number of Core Cells used both for implementing routing and logic (All CCs), the number of Core Cells used

Circuit Name	*Description*
b01	FSM that compares serial flows
b02	FSM that recognizes BCD numbers
b03	Resource arbiter
b04	Compute min and max
b05	Elaborate the contents of a memory
b06	Interrupt handler
b07	Count points on a straight line
b08	Find inclusions in sequences of numbers
b09	Serial to serial converter
b10	Voting system
b11	Scramble string with variable cipher
b12	1-player game (guess a sequence)
b13	Interface to meteo sensors
b14	Viper processor (subset)

Table 1: ITC'99 benchmark circuits

to implement just netlist logic instances (Logic CCs), the number of memory elements (FFs) and the percentage of the Logic Area computed before the optimization analysis (Pre-OPT L.A.). These data have been produced by the Atmel Figaro tool used to perform the implementation. Finally we report the area gain, if the optimizations identified by the PAHT tool would be implemented. For designs that occupies a relevant portion of the FPGA area, the

AT40K	Figaro ids8.2.2				Analysis
Circuit	All CCs [#]	Logic CCs [#]	FFs [#]	Pre-OPT L.A. [%]	Area gain [%]
b01	11	11	10	0.48	9.09
b02	8	8	8	0.35	25.00
b03	58	54	34	2.34	5.56
b04	176	119	66	5.16	5.04
b05	225	144	36	6.25	18.06
b06	14	13	11	0.56	15.38
b07	117	74	45	3.21	12.16
b08	60	58	21	2.52	8.62
b09	48	43	28	1.87	6.98
b10	58	56	24	2.43	12.50
b11	143	118	35	5.12	12.71
b12	347	298	108	12.93	10.74
b13	105	66	60	2.86	4.55
b14	1299	905	215	39.28	10.28

Table 2: Optimization results

optimizations of the Logic Core Cells could lead to a maximum gain of about

ATF280E	Figaro ids9.0.2				Analysis
Circuit	All CCs [#]	Logic CCs [#]	FFs [#]	Pre-OPT L.A. [%]	Area gain [%]
I8051	10179	7867	1334	54.63	8.91
leon2 (no cache)	6195	5556	1264	38.58	14.38

Table 3: Optimization results

15% for the Leon2 processor with an average gain of 10%.

3 Conclusions

In conclusion, the automatic analysis process developed led to evaluating possible area optimizations for circuits implemented in Atmel FPGAs. Realistic circuits have been analyzed as case study and the results show that an average 10% gain can be achieved on relevant circuits with a peak of about 15% on the Leon2 processor core.

References

[1] Filomena Decuzzi, David Merodio Codinachs, *Use of macros for At40k FPGAs*, 2010.

[2] *Mentor Precision RTL Synthesis 2007a3.40EM_Atmel.*

[3] Various authors, ITC'99 Benchmark homepage: http://www.cerc.utexas.edu/itc99-benchmarks/bench.html, *Benchmark Circuits*, 1999.