Czech Technical University in Prague
Faculty of Information Technology
Department of Digital Design

Reliability Analysis of SRAM-based Field-Programmable Gate Arrays

by

Jiří Kvasnička

A thesis submitted to
the Faculty of Information Technology, Czech Technical University in Prague,
in partial fulfilment of the requirements for the degree of Doctor.

PhD programme: Informatics

Prague, August 2013

Thesis Supervisor:
Hana Kubátová
Department of Digital Design
Faculty of Information Technology
Czech Technical University in Prague
Thákurova 9
160 00 Prague 6
Czech Republic

Abstract and contributions

This doctoral thesis deals with bit error effects in the configuration memory of FPGA, particularly those induced by heavy ions or particles, called SEU. These bit errors may have an effect on the design functionality and may lead to the failure, even when the design is secured with security codes. On an average, these bit errors do not affect the design at all and most bit errors have no influence on the design functionality.

This thesis presents two methods of evaluation of the bit error effects: 1) software-based fault effect prediction, that is based on the bitstream of analyzed design, FPGA architectures knowledge and knowledge of bitstream associations to the FPGA resources; 2) hardware emulator, that emulates soft errors in the configuration memory by a dynamic reconfiguration and classifies the fault effect on the basis of exhaustive testing. As those tasks require a precise knowledge of the FPGA architecture and bitstream addressing, this information was extracted from the FPGA by methods that are also presented in this doctoral thesis. The precise prediction requires fault models of the bit error, which are introduced in this doctoral thesis.

In particular, the main contributions of the doctoral thesis are as follows:

1. Introduction of fault models for single bit errors in the configuration memory with regards to the placed&routed design in the FPGA.
2. Prediction method for the single bit error effect of the design mapped in the FPGA, which can evaluate the number of sensitive bits in the configuration memory.
3. An emulator platform, which evaluates the SEU effect and even the CED (Concurrent Error Detection) reliability properties.
4. Experimental results of the soft error effect on several benchmarks mapped in the FPGA.
5. Described FPSLIC structure with bitstream addressing with guidelines of bitstream analysis of any FPGA device.

Keywords:
FPGA, SEU, soft error, fault model, fault effect, emulation, dependability, CED.

Acknowledgments

First of all, I would like to express my gratitude to my dissertation thesis supervisor, Dr. Hana Kubátová. She has been a constant source of encouragement and insight during my research and helped me with numerous problems and professional advancements.

I would like to thank to Dr. Petr Fišer and Dr. Jan Schmidt for giving me a valuable feedback and for having pregnant discussions. I would also like to thank Dr. Pavel Kubalík, who has been a great source of motivation during the early stage of the emulator birth and FPGA analysis.

Special thanks go to the staff of the Department of Digital Design, who maintained a pleasant and flexible environment for my research. My research has also been partially supported by the Ministry of Education, Youth, and Sport of the Czech Republic under research program MSM 6840770014, and by the Czech Science Foundation as project No. 102/04/2137 (53/04003/13136).

Finally, my greatest thanks go to my beloved wife, who has been a constant source of endless support, and to my family, for their help with saving enough time for writing out of this thesis.

Dedication

To Veronika

Contents

List of Figures

List of Tables

List of Abbreviations

Abbreviations

ASIC	Application-Specific Integrated Circuit
AT40K	A commercial FPGA series from Atmel
AT94K	A commercial SoC from Atmel, which contains an FPGA and an AVR
AVR	An 8-bit microcontroller from Atmel
CED	Concurrent Error Detection
CMOS	Complementary Metal Oxide Semiconductor
CRAM	Configuration RAM: Memory, that holds the configuration of FPGA
CRC	Cyclic Redundancy Code
DDD	Displacement Damage Dose
DFF	D flip-flop
DRAM	Dynamic Random Access Memory
ECC	Error Correction Code
EDA	Electronic Design Automation
EDIF	Electronic Design Interchange Format
EEPROM	Electrically Erasable Programmable Read-Only Memory
FIT	Failures in time. Number of faults during 10^9 h
FPGA	Field Programmable Gate Array
FPSLIC	Field Programmable System Level Integrated Circuit. A SoC from Atmel
FS	Fault Security
GCR	Galactic Cosmic Rays
GND	Ground
HDL	Hardware Description Language
I/O	Input/Output
JTAG	Joint Test Action Group (refers to an IEEE 1149.1 standard)
LET	Linear Energy Transfer
LUT	LookUp Table
MBU	Mutiple Bit Upset
MOSFET	Metal Oxide Semiconductor Field Effect Transistors
MPGA	Mask Programmable Gate Array
MTBF	Mean Time Between Failures

MUX	Multiplexer
NDA	Non-Disclosure Agreement
NIEL	Non-Ionizing Energy Loss
NMOS	N-channel Metal Oxide Semiconductor Field Effect Transistor
PIP	Programmable Interconnection Point
PKA	Primary Knock on Atom
PLA	Programmable Logic Array / PLA physical description format
PLD	Programmable Logic Device
PMOS	P-channel Metal Oxide Semiconductor Field Effect Transistor
RAM	Random-Access Memory
σ	Cross section
SBU	Single Bit Upset
SECDED	Single Error Correcting and Double Error Detecting code
SEE	Soft Error Effect
SEFI	Single Event Functional Interrupt
SEM	Soft Error Mitigation IP core (controller)
SEMU	Single Event Multiple Upset
SER	Soft Error Rate
SET	Single Event Transient
SEU	Single Event Upset
SI	The International System of Units. Le Système International d'unités
SoC	System on Chip
SPE	Solar Particle Events
SRAM	Static Random-Access Memory
ST	Self-Testing
TID	Total Ionizing Dose
TMR	Triple Modular Redundancy
TSC	Totally Self-Checking
UART	Universal Asynchronous Receiver and Transmitter
V_{DD}	Voltage Supply for the Drain of the transistor (typicaly positive)
V_{SS}	Voltage Supply for the Source of the transistor (typically ground)
VHDL	VHSIC (Very-High-Speed Integrated Circuits) HDL
VLSI	Very Large Scale logic Integration

FPGA labels

C	Inner Logic Cell net, selects registered or direct signal (Fig. A.4)
D	Inner product of the LUTs. Also an input of the register (Fig. A.4)
E1* ... E5*	Express routing nets, that are located South (E1S...E5S), North (E1N...E5N), West (E1W...E5W) or East (E1E...E5E) from the repeater. Sec. A.9
FPGAD	AVR register, Data register for Cache Logic
FPGAX	AVR register, X part of the Cache Logic address
FPGAY	AVR register, Y part of the Cache Logic address

FPGAZ	AVR register, Z part of the Cache Logic address
H1 ... H5	5 horizontal local routing nets (Fig. A.5)
H1B ... H5B	5 horizontal bottom express nets (Fig. A.5)
H1T ... H5T	5 horizontal Top express nets (Fig. A.5)
iX, iY, iW, iZ	Products of the multiplexers, that drive the LUT inputs (Fig. A.4)
L	Product net of Logic Cell output driver (Fig. A.4)
L1 ... L5	5 local routing nets of Logic Cell (Fig. A.4)
L1* ... L5*	Local routing nets, that are located South (L1S...L5S), North (L1N...L5N), West (L1W...L5W) or East (L1E...L5E) from the repeater. Sec. A.9
LUT_X	A 3-input LUT in the Logic Cell (drives the XL net, Fig. A.4)
LUT_Y	A 3-input LUT in the Logic Cell (drives the YL net, Fig. A.4)
oX, oY	Direct LUT outputs to the neighbor Cell (Fig. A.4)
P	Inner Logic Cell net, selects registered or direct signal (Fig. A.4)
Q	An output of the register in the Logic Cell (Fig. A.4)
V1 ... V5	5 vertical local routing nets (Fig. A.5)
V1L ... V5L	5 vertical left express nets (Fig. A.5)
V1R ... V5R	5 vertical right express nets (Fig. A.5)
W	Input to the LUT (Fig. A.3)
X	Input to the LUT (Fig. A.3)
XL	Output of the LUT_X (Fig. A.3)
Y	Input to the LUT (Fig. A.3)
YL	Output of the LUT_Y (Fig. A.3)
ZM	A net driving the ZM Multiplexer (serves as a 4th LUT intput, Fig. A.3)

Math symbols and operators

$\lceil x \rceil$	Ceil function. $\lceil x \rceil$ is the smallest integer not less than x
$\lfloor x \rfloor$	Floor function. $\lfloor x \rfloor$ is a largest integer not greater than x
$\mathbb{N}$	Set of Natural numbers. $\mathbb{N} = \{1, 2, 3 \ldots\}$
σ	Standart deviation of a normal (Gaussian) distribution
$x \bmod y$	Remainder of the integer division. $x \bmod y = x - y \lfloor x/y \rfloor$
$x \div y$	Integer division. $x \div y = \lfloor x/y \rfloor$

Chapter 1

Introduction

This doctoral thesis studies an effect of soft errors in the FPGA configuration memory on the functionality of designs implemented in that FPGA.

The FPGAs became a universal and cost effective platform for a lot of applications, including prototyping, small productions and applications requiring reprogramming during lifetime or dynamic reconfiguration. The price for the universality of FPGAs is paid by a vulnerability of the configuration memory to the soft errors, which modify the implemented FPGA design itself, instead of changing the inner state of the design only, as traditionally happen in all digital circuits, even non-FPGA based.

As the technology of FPGA devices evolve rapidly, as well as the size of configuration memory, the bit errors in the configuration memory are becoming a significant vulnerability issue of system based on such modern FPGAs, as the soft error rate, caused by an elementary particle or ion, is increasing as well and is more probable to happen during the lifetime of the device.

The configuration bit error mechanism (SEU, explained in section 2.3.2) is very well described, as well as the error rates in different environments. But only few studies directly address the effect on design, that is already mapped onto the FPGA and rarely are those studies supported by the exact knowledge of the FPGA structure and the fault models of the design causing the error.

The traditional fault analysis and the ATPG (Automatic Test Pattern Generation) rely on the stuck-at fault models, which are extremely useful for the fault testing and analyzing in the ASICs, but do not cover all faults in the FPGA (especially interconnection), as the FPGA structure is naturally redundant and only a subset of resources is used by the design and different faults are possible in the FPGA. Fault models, that better reflect the configuration bit errors, are therefore necessary for dependability analysis.

An FPGA, as a universal device, does not typically require all the programmable resources of the FPGA to be active. On the contrary, the majority of configuration memory is not used and a single error in the unused part has no effect. The classification of those bits is a task that requires a precise knowledge of the FPGA structure and an exact assignment of configuration bits to the FPGA resources. For smaller circuits running in the FPGA, the number of sensitive bits (bits that cause an error on the output) can be dramatically smaller, even by factor 10 or more.

The FPGA structure knowledge is the crucial limiting factor for any study dealing with errors in bitstream and only a few studies address the configuration bit errors in higher detail, than in LUTs. The content of LUTs represent typically only 10 ∼ 15% of the total bitstream of the FPGA. Unfortunately, the FPGA structure and bitstream information is an FPGA vendor's intellectual property, which is kept secret for many reasons.

In order to overcome the basic problem of the FPGA structure knowledge, one FPGA (FPSLIC device from Atmel) has been deeply analyzed and this knowledge was obtained. Not only the result, the bitstream description, is described in this doctoral thesis, but also the procedure of analysis itself has been documented, including all algorithms and methods used, touching also the field of the FPGA security against the reverse engineering analysis.

Main interest of this research is the bit error effect on the combinational circuits and its effect on same circuits that are equipped with the error detection code. For this reason, a fault classification was used and an experimental platform evaluating the class and the real impact was built.

1.1 Motivation

The great motivation was the existence of SEU (Single Event Upset – a mechanism of changing flip-flop state due to radiation losses of traversing ion/particle). The SEU can occur in the configuration memory, which, if unprotected, may alter the design in the FPGA. Before planning or suggesting methods for SEU mitigation, there was a need to know what is actually happening during the SEU in configuration memory of mapped design and what are the quantities of various types of faults and FPGA resources. Because of lack of similar studies, the motivation for this research was quite straightforward and can be summarized by following points:

- Demand of knowing the exact number of bits that are capable of harming the design.
- Lack of experimental data. Currently all bit-error data is obtained either by methods not considering the mapped design, or by LUT testing only.
- A need of platform for in-hardware bit error testing for the validation of fault effect
- Missing fault models for bit-flips in the configuration memory. Only a little attention is paid to the fault models in FPGA, though they are different from the traditional stuck-at faults.

This research can be characterized as a basic research, which has an impact in following fields that were also motivations for performing this study:

- Reliability calculations of design implemented in FPGA
- Efficiency of CED methods in FPGA

1.2 Problem Statement

This doctoral thesis should answer some very fundamental questions:

- What exactly happens to the designs in SRAM-based FPGA, which suffers from a bit error in the configuration memory?
- Can the soft error effect be predicted and how?
- What is the exact number of sensitive bits in the configuration memory for a specific design?
- How does the physical implementation differ from similar circuits not being implemented in the FPGA by the metrics of number of possible faults and fault classification within the scope of CED techniques?

1.3 Related Work/Previous Results

This topic was studied since 2005 as a diploma thesis [A.5] by the author. That work focused on the soft errors in LUT. A hardware emulator platform built on the FPSLIC FPGA device was introduced, though only LUTs were tested in 2006.

The emulator was further improved and the range of tested bits was extended from the LUT subset to a complete bitstream. Further, the classification algorithm was implemented, after the knowledge of FPGA structure and bitstream meaning was acquired. Those results were partially published in between 2007 and 2009 in publications listed in the list of author's publications.

As the problem of soft error effect increases, both leading FPGA manufacturers implemented some solutions of soft error mitigation to the commercial-grade FPGAs in the years 2010-2012. These solutions are designed for the newest FPGA devices (improved scrubbing) and EDA tools, which allow to pre-select bits that are sensitive to the design. Moreover, an emulation of soft errors became possible lately, however no information about the FPGA resource or fault type is provided. These solutions are described in section 3.1.

Some studies exist for aerospace and military applications, but those application do not typically use cheap commercial-grade FPGAs (especially the SRAM-based FPGAs, which are the target of this study), and different techniques of soft error mitigation are used there. Those devices are designed from the beginning to cope with the soft errors.

1.4 Contributions of the Thesis

1. Fault models for single bit error in the configuration memory with regards to the placed&routed designs in the FPGA.
2. Prediction method for the single bit error effect of the design mapped in the FPGA, which can evaluate the number of sensitive bits.

3. An emulator platform, which evaluates the SEU effect and even CED reliability properties.

4. Experimental results of the soft error effect on designs mapped in the FPGA.

5. Described FPSLIC structure with bitstream addressing with guidelines of bitstream analysis of any FPGA device.

1.5 Structure of the Thesis

The thesis is organized into several chapters as follows:

1. *Introduction*: Describes the motivation behind my efforts together with my goals. There is also a list of contributions of this doctoral thesis.

2. *Theoretical Background*: Introduces the reader to the necessary theoretical background: Digital circuits, FPGA, soft error mechanism and radiation as a primary source of the soft errors.

3. *State-of-the-Art*: Surveys the methods of soft error mitigation and suppressing and surveys other works related to the problem of this doctoral thesis.

4. *Overview of My Approach*: describes the fault classification of circuits with CED, introduces FPGA resource classes and fault models, shows how to predict the fault effect and introduces the hardware SEU emulator.

5. *Main Results*: Show the results of software analysis and compare it with the hardware emulation results, including the experimental result of parity-secured benchmark

6. *Conclusions*: Summarizes the results of my research, suggests possible topics for further research, and concludes the thesis.

7. *Appendices A and B*: Describe the bitstream of FPSLIC, associate the bitstream address with FPGA resources and demonstrate how this knowledge can be acquired

8. *Appendix C*: Lists the results for testing of all benchmark for eventual further analysis

Chapter 2

Theoretical Background

This chapter will describe all necessary terms that will be operated within this work later on. Its intention is to explain all necessary topics to understand the motivation and the impact of this thesis. Digital circuits basics will be described with basic logic gates in CMOS (section 2.1), which will continue into the description of the structures used in the FPGAs (section 2.2).

Following sections will focus on the radiation (section 2.3), its quantities (section 2.3.1), principals of damage to the integrated circuits (section 2.3.2), single particle effects (section 2.3.2.3) and sources, which cause these effects (section 2.3.3) and occurrences of those various sources in various environments (section 2.4).

Final section 2.5 will explain some terms used to describe the dependability of systems.

2.1 Digital circuits

Digital electronic circuit is a special case of an (analog) electronic circuit, that operates at discrete (=digital) signal levels. Typical digital circuit uses 2 signal levels and this digital signal is mostly represented by a voltage.

Since the digital signal is discrete, there is a native abstraction of it, called a *logic value*, which is defined for a specific voltage range of a certain *logic family*. The commonly used abstraction of digital signal is *Boolean logic*, which operates with two values: “0” and “1”, referred also as “False” and “True”, or “Low” and “High”.

Logic family provides specifications of its “High” and “Low” acceptance voltage ranges and also its guaranteed output voltage levels, which are always more strict than the input levels (giving some space to noise to be safely superposed). These voltage ranges can be found in the datasheet and are referred as V_{IH} (minimal voltage, that is accepted as “High”), V_{IL}(maximal voltage, that is accepted as “Low”), V_{OH}(minimal guaranteed voltage generated by the “High value”) and V_{OL}(guaranteed maximal voltage generated by the “Low” value) parameters. The old famous TTL logic family is defined by V_{IL}=0.8 V and V_{OH}=2 V for example.

2.1.1 Basic Logic Gates

The Boolean algebra provides some *Boolean functions* ($f : \mathbb{B}^n \rightarrow \mathbb{B}$, where $\mathbb{B}$ is the Boolean domain {0,1} and n is a natural number defining number of inputs): AND, OR, NOT, XOR and others. These primitives are implemented by the *logic gates* (AND gate, OR gate, inverter, XOR gate and others). The "logic gate" term may however refer both to the Boolean function and the physical implementation.

2.1.1.1 Combinational[1] Circuits

Circuits, where the output is a function of the inputs only, are called *combinational.* Any implementation of Boolean function is therefore a combinational circuit by the definition. Number of inputs is not limited, but combinational circuits cannot contain any feedback, which would make them sequential. Combinational circuits can be composed of several basic logic gates, such as NAND, NOR, XOR, AND, OR, NOT (inverter), buffer, multiplexer.

In the very old history, these elementary logic gates were available as standalone integrated circuits (typically containing small number of gates in a single package). Nowadays, elementary gates are directly used inside the ASIC using a *standard cell* library (containing a complete layout of each gate) or using a *full custom* design, which is done at the transistor level.

Implementation of combinational circuits in FPGA is done by the cascade of *logic cells* (containing LUTs,) which are connected together and evaluate the desired combinational function.

2.1.1.2 Sequential Circuits

Besides the combinational circuits there are also logic gates that define their output not only by their inputs, but also by the *state* of the circuit. These logic circuits are called *sequential logic circuits* and are capable to store the information inside.

According to their behavior, sequential circuits can be *asynchronous*, *gated (latched)* or *synchronous (clocked).* Some widely used sequential gates are important to understand (especially SRAM cell) and their description will be given in sections 2.1.3, where they will be discussed with relation to the technology, which influences the implementation significantly. Mechanism of soft error in sequential circuits is explained in section 2.7.

The simplest, elementary sequential circuits are often called flip-flops. Complex sequential circuits typically utilize elementary flip-flops, such as DFF (D flip-flops), as a basic structural components. Sequential logic gates are available as single integrated circuits as well as the combinational gates; however modern integrated circuits integrate millions of them on a single chip.

[1]term combinatorial is also used

2.1.2 Technology and Logic Families

Today's market of digital integrated circuits is dominated by the CMOS (Complementary Metal Oxide Semiconductor) process. CMOS process combines both N-MOSFET (N-channel metal oxide semiconductor field effect transistors, PMOS) and P-MOSFET (p-channel MOSFET, PMOS) on a single chip in a single manufacturing process. The properties of CMOS technology (low leakage, low static consumption, small transistor size) allow the logic gates to be easily implemented by few transistors.

The CMOS logic families started in 1968 by the CD4000 logic family, which was further improved in the 74HC (High-speed CMOS) series in 1981 and further in AHC and many low-voltage/low-power series(for example 74LVC, 74AUP and 74AUC from Texas Instruments)

There are also other technologies based on the BJTs (Bipolar Junction Transistors): ECL (Emitter Coupled Logic, 1962), DTL (Diode Transistor Logic, 1962), RTL (Resistor Transistor Logic, 1959) and TTL (Transistor – Transistor logic, 1964) with its famous 7400 series, which got many derived (improved) series: 74S, AS, ALS, F, ABT and others.

Unlike the bipolar technology, the CMOS technology is well capable of *very large scale logic integration* (VLSI) and is the current leading technology in the digital design market, allowing integration billions of transistors in a single chip . It is basically the only technology, in which the FPGA (described in 2.2) chips are fabricated and that is the reason, why a space will be given to the CMOS gate description in section 2.1.3.

2.1.3 Logic Gates in CMOS

Detailed transistor-level design of the CMOS technology primitives will be shown in this section in order to understand the fault effects later.

The target of this thesis is the reliability of the FPGAs, which are fabricated in CMOS technology. Therefore primitives used at this technology need to be explained. The technology, that applies to the result of this work, is 0.35 µm (introduced in 1995), which is more than 1 order less advanced, than current leading process for FPGA manufacturing, which is 28 nm [8, 72]. 0.35 µm and 28 nm parameters of the technology in simplicity refer to a minimum transistor gate length (L) that is possible to fabricate in that technology.

Almost all CMOS gates provide a rail-to-rail output, have low static consumption (leakage) and therefore consume low standby current.

The simplest gate in CMOS is an **inverter**. The Inverter is composed of just 2 transistors, as shown in Fig. 2.1a. **NAND** gate is shown in Fig. 2.1d, which requires 4 transistors only. The NOR gate can be implemented in a similar way. Note, that the NAND implementation is simpler, than the AND, which requires an additional inverter.

Transmission gate is shown in Fig. 2.1b in a simple form. Transfmission gate serves as a bi-directional switch, which connects or disconnects input with output. The current can flow each direction, so the input and output are fully interchangeable. The transistors have the bulk connected separately to the ground (V_{SS}) or supply voltage (V_{DD}), unlike the usual MOS transistors that connect the substrate to the source (NMOS) or drain

(PMOS). This allows passing any signal within the supply voltage in both directions. The full connection of transmission gate is shown in Fig. 2.1c.

However full transfer gates are seldom used in FPGA [41] for their higher space requirements and higher capacitance.

The **pass transistor** (shown in Fig. 2.3) uses only single NMOS transistor to pass the signal. This represents a problem for a high-level signal to pass the pass transistor because of the lack of the PMOS transistor. The output of the pass gate will then never reach the rail. The voltage drop is high enough to partially open the PMOS transistors and create an enormous leakage current [40], which is a great concern in FPGAs with billions [72] of transistors.

Pass transistors are used intensively for routing, but they can be also used for logic gates implementation with some advantages and disadvantages [79, 50]. Pass transistors are often used in the multiplexer selection trees [41, 10], as shown in Fig. 2.2

There are two approaches to overcome the pass gate output level problem, that are relevant for the 0.35 µm technology: using **gate boosting** [19, 41], where the gate[2] control voltage of the pass gate is increased above the VDD (shown in Fig. 2.3a), or **level restoration**, where an extra PMOS transistor driven by inverted signal will pull the signal to rail, as shown in Fig. 2.3b.

The level restoration circuit can be implemented together with a following buffer, as shown in Fig. 2.3c. The level restoration PMOS transistor is driven by the first inverting stage of the buffer as a control signal.

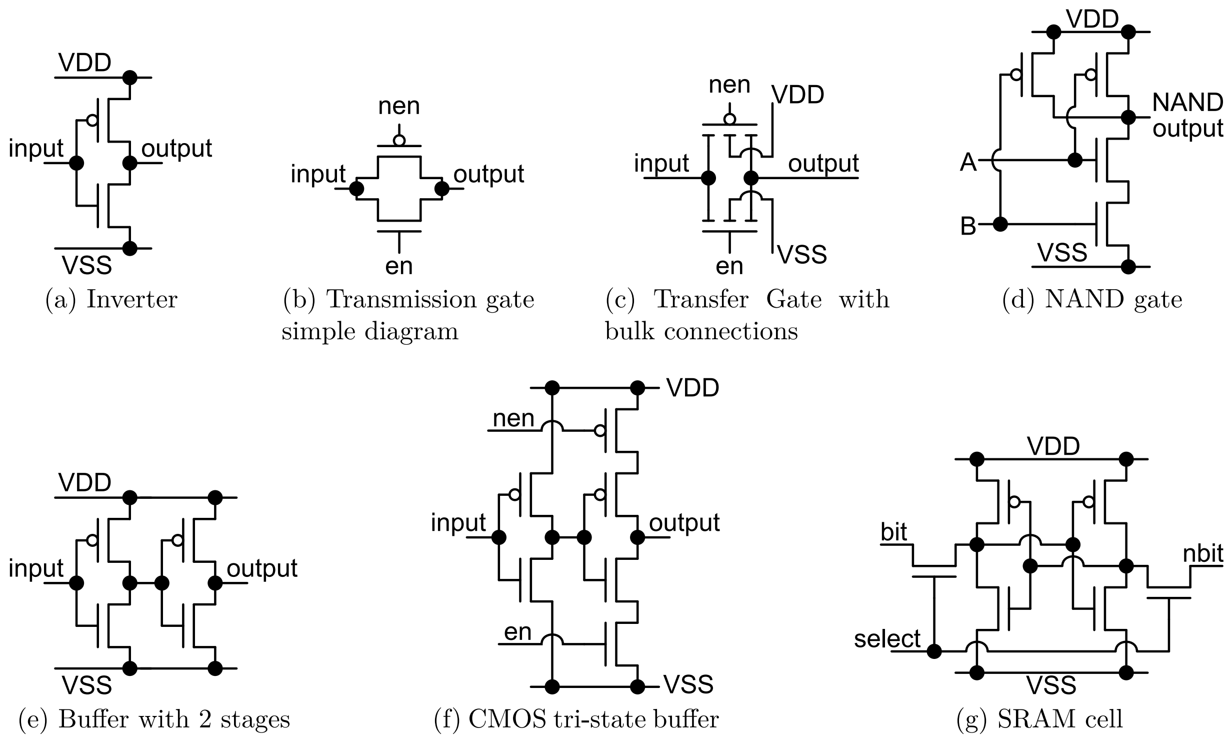
The **buffer** is a gate that follows the input, has minimal input capacitance and provides the required current at the output. Its primary use is therefore a strengthening of the signal and a separation of the load capacitance from the signal wire. A two-staged buffer is shown in Fig. 2.1e. The Real buffer is created using several stages of inverters, typically increasing the buffer strength (transistor gate widths) exponentially to provide the best delay, gate area and output strength.

Sometimes it is useful to share a wire (bus), for which a **Tri-State Buffer** is used. There are several ways of implementing the tri-state buffer. It can be built of already described gate: a buffer and a transfer gate (or pass transistor). It can be also implemented as a macro shown in Fig. 2.1f, which takes an advantage in physical layout size saving, when one contact can be saved in each final drive transistor by merging both PMOSes (NMOSes respectively) into a single transistor with a dual gate. Other implementations can employ low-power and sleep modes [10].

SRAM cell (Static Random Access Memory) is a basic CMOS element that is capable of storing information – one bit. It is typically made of 6 transistors, as shown in Fig. 2.1g. 4 transistors (in the center part) make two inverters that connect their outputs to each other's inputs. The other 2 are used for writing into or reading from the SRAM.

What were not shown in this brief overview of gates in CMOS are the sizes of the transistors. The choice of transistor sizes cannot be generally given, because of different requirements on the gates (input capacitance, delay, fan-out, output strength, etc.), apart

[2]Gate refers to the gate of transistor in this context, not the logic gate



(a) Inverter (b) Transmission gate simple diagram (c) Transfer Gate with bulk connections (d) NAND gate

(e) Buffer with 2 stages (f) CMOS tri-state buffer (g) SRAM cell

Figure 2.1: Implementation of basic gates in CMOS

from the fact that sizing of the transistor is highly technology specific. More information about transistor sizing can be found in [19, 41].

2.1.4 Description of Digital Circuits

The digital circuit description can be given in more ways. The most used and convenient is the HDL (Hardware Description Language). The commonly used HDLs are VHDL and Verilog. VHDL is used extensively in this thesis. The modern trend is to move the hardware description language towards a system level modeling language like SystemC or

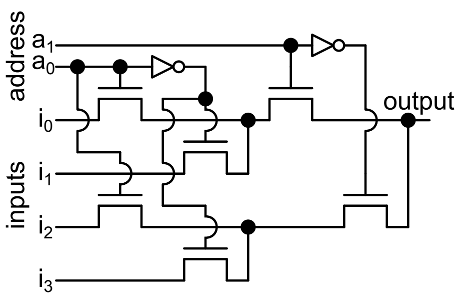
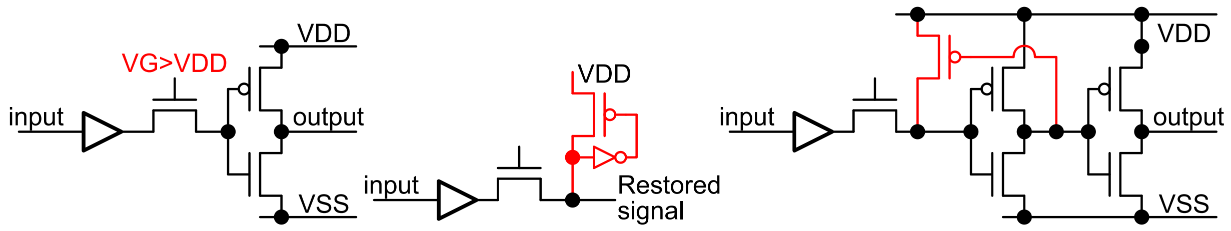


Figure 2.2: Multiplexer implemented by pass transistors



(a) pass transistor with gate boosting

(b) Level restoration

(c) pass transistor followed by level-restoration buffer

Figure 2.3: Solutions for the pass transistor level problem

SystemVerilog.

Another possibility of the digital circuit description is to use a netlist – a file, that contains only nets, instances of some set of blocks and a description, how to connect them. This is considered to be quite a low-level description. This is a convenient way for describing a circuit, which is composed only of commonly known logic blocks, such as FPGA primitives. A widely used standardized format of netlist is EDIF (Electronic Design Interchange Format). Version EDIF 2 0 0 (ANSI/EIA-548-1988 standard) is used in this thesis.

The most low-level format of the combinational circuit description format used in this thesis is the PLA format [22], which is suitable for Boolean minimization [22] of the given circuit description, for example by tools like Espresso [22] or BOOM [29]. This PLA format and minimization was used for programming of the PLA devices (Programmable Logic Arrays, programmable logic device, that is able to implement combinational logic circuit by the AND-OR matrix). Benchmarks, which were used in this work, were provided in this format.

The description in HDL usually doesn't reflect the final topology of the target application – the FPGA structure. The process, that converts the HDL design into netlist of logic gate primitives, is called **logic synthesis**.

When the target technology is precisely known, the design can be translated (mapped) to the target logic gate primitives (logic cells in FPGA, AND-OR equations for PLA, etc.). This step is called **technology mapping**.

Finding an optimal result (in terms of area, speed or both) is a NP-hard problem.

2.1.5 Simulation

Digital circuits can be easily simulated. The *digital simulation*, unlike the analog circuits, operates only with logical (discrete) values and the propagation delay model. The *analog simulation* of the complete electrical circuit represents in many orders of magnitude more difficult task with much more time required for the simulation, especially for designs counting tens of thousands transistors or more.

The simulation of the logic circuit can be made at different levels of design stage: At a HDL level (fastest and with only a little detail of delay), at a synthesized netlist, at a mapped netlist and at a placed&routed design (will be discussed later), which contains precise delay information.

The final result of the digital simulation should be the same for all levels of description, assuming that all combinatorial parts of the circuits will have enough time to produce a result (critical path delay) and all timing requirements for the sequential parts are met (setup time, hold time).

2.1.6 Scalability of Digital Circuits

Logic gates are the simplest elements, from which any digital electronic circuit can be built. All mentioned logic gates are commercially available as a single *integrated circuit*, incorporating some reasonable amount of (usually same) gates into a single *chip* and *package* with less than 20 pins (for example 7400, a quad NAND gate).

As already mentioned, all basic logic gates are commercially available as integrated circuits since 1960s. At the beginning, the digital systems were built from these simple-logic-gate chips. Requirements on most applications overcame the ages of building large systems from single gate IC (Integrated Circuits) long time ago. The space, speed, manufacture cost, manufacture yield and reliability parameters forced the digital system to be composed of fewer components and to be more integrated on a single chip.

There are 3 basic approaches, how the digital logic systems can be scaled to a very large and complex system:

1. Create a custom chip, which would perform the desired functionality directly in hardware. Such a chip is called ASIC (Application Specific Integrated Circuit).

2. Use a universal logic device, which can be configured by a user to perform the desired functionality in the hardware. Such devices are GALs (Generic Array Logic devices, since), CPLDs (Complex Programmable Logic devices) and FPGAs (Field Programmable Gate Arrays, described in section 2.2). Today's most popular devices are CPLDs and FPGAs.

3. Use a universal logic device, that is "programmed" by creation of a special metal layer mask, which than used during a production of the device. This approach was historically first available in 1970s for the PLA devices, later for example for the MPGA (Mask Programmable Gate Arrays) from Actel company. Today these devices are being pushed out of market by the flash and antifuse PLDs.

4. Use a processor, which provides the functionality by running a program (a sequence of instructions) in the processor.

The selection of the approach is driven by following factors and the final approach selection is a tradeoff among them: price, volume, time for development, power constraints, speed, size and reliability

2.2 FPGAs

FPGA (Field Programmable Gate Array) is the most flexible pre-fabricated logic device, that can contain more than a million of effective gates, leaving the end user an opportunity to configure the device by programming the connections between gates and by modifying their (gate's) function.

The configuration of an FPGA has a form of a file containing the configuration states (bits) of all resources (connections, drivers, LUTs, etc.). This configuration file is stored outside of the FPGA (for the SRAM-based FPGAs) in a non-volatile memory, such as flash memory. During the startup, the configuration is streamed into the FPGA, thus the configuration file is often referred as a *bitstream*.

The FPGA market is experiencing revolutionary growth since 1987. FPGA became an affordable platform for a development and a debugging of applications, keeping the flexibility in reprogramming by a new bitstream, as well as a platform for small- and middle-volume production of digital electronics.

FPGA devices are affordable in a term of unit price (especially in low-quantity production), because of the versatility of the device. FPGAs are fabricated usually within the leading CMOS technology. The performance and the number of gates give the FPGAs a possibility of implementation of almost any application of digital logic imaginable.

The price for a universal pre-fabricated device over the ASIC is paid by the chip area ($\sim 35\times$), static and dynamic ($\sim 14\times$) power consumption and speed ($3.4\times \sim 4.6\times$) [38].

A wide range of applications also covers applications, where the reliability is an issue due to hostile environment (radiation, electromagnetic noise) or a long-term fault-free operation is required. This counts applications, which laid in the ASIC domain exclusively, like satellites.

2.2.1 Architecture

The FPGA architecture is mainly defined by its routing architecture. It can be hierarchical, island-style or heterogeneous. This section will describe the **island-style** FPGA architecture briefly and explains the terms that are used in this work. The reason for only island-style architecture being described is that representative of this style only has been studied. Complete FPGA architecture overview can be found in [39] or [21].

The basic topology of an island-style FPGA is shown in Fig. 2.4. It is a regular two-dimensional mesh of logic cells, connections and switch blocks.

The programmable **Switch block** contains **wires** and **switches** (buffered, unbuffered, unidirectional, bidirectional or a combination of different types), that connect wire segments. The activated combination of those switches form the desired connections of the switch block. The connecting capabilities of switch blocks are limited (only few pre-routed connection combinations are possible), such as Disjoint, Wilton, Universal [24] and Subset [60] switch block connection patterns [45]. Typical fan-out of the switch block is 3, meaning, that signal wire can be connected to 3 other wires.

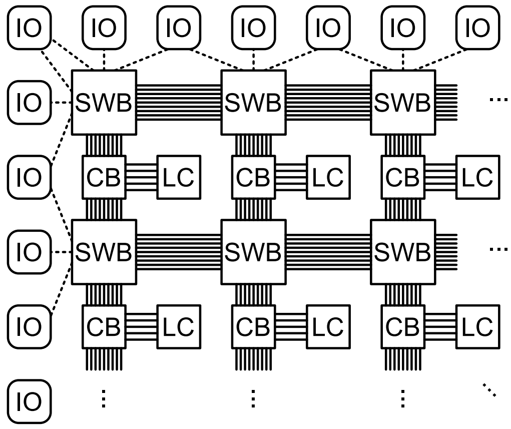


Figure 2.4: An island-style FPGA topology overview. IO=Input/Output block. SWB=SWitch Block. LC=Logic Cell. CB=logic cell Connection block.

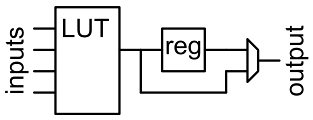


Figure 2.5: FPGA logic cell - a simplified view

Some wires can span over more logic blocks, creating longer wire segments of length of 2, 4, 8, etc. FPGA usually contains a mixture of wire segments with different lengths. The advantage of having longer wire segments is a better performance in terms of critical path delay, as discussed in [20]. This approach will be shown on the “express wires” of the device under the study in the section 2.2.3.

The programmable switch in the switch block can be hard-wired as a pass gate or a tri-state buffer and is controlled by a value stored in the configuration memory.

The **Connection block** connects the logic cell to the routing wires, as shown in Fig. 2.4. The number of connection combinations is usually also limited to only small subset of pre-wired connections. The flexibility of the logic cell's inputs and outputs connections is also limited to a subset of pre-routed connections, so only some wires can be connected to the logic cell inputs. The connections within the switch block are commonly referred as **global interconnection**.

The **Logic Cell** represents the basic programmable block, which provides a logic function based on the inputs and/or the clock&reset signals. A simplified view of the logic cell is shown in Fig. 2.5. The logic cell contains one, two (typically) or more **LUT**s (Look-Up Tables) generating desired functions (each LUT evaluates one logic function). The LUT size is typically 3-7 inputs [4]. The function product can be sent to the output directly (asynchronously) or through the register (synchronized by not-in-figure-shown clock), as shown in Fig. 2.5.

Logic cells can be grouped into **clusters** [21], which take an advantage of a fast intra-cluster **local interconnection** (connection within logic cell and within the cluster among logic cells) in order to produce a better delay for complex functions.

The **Input/Output block** (I/O block) is a general mean of communication with outer space. The I/O block is usually connected to a physical pin, which protect the chip from the ESD (ElectroStatic Discharge), clamps the signal to a valid range, restores the logic value, optionally debounces the signal, defines the input delay (which is selectable), protects from metastability, synchronizes the signal and controls the output slew rate and delay.

2.2.2 FPGA Configuration

The configuration of FPGA is usually stored in an external flash memory, which contains the configuration file known as a **bitstream** and which is downloaded into the FPGA during the power-up. The bitstream is streamed into a special volatile SRAM memory (called configuration memory), which holds the configuration a whole power cycle, until the power is removed or special actions for a content replace are taken.

The content of the FPGA configuration SRAM is typically routed bit-by-bit directly to the FPGA resources, where it controls its function and connections. The function of 1 configuration bit can be 1 bit of LUT, single connection (a logic cell to net connection for example) or a predefined set of connections (for example in the switch block).

The bitstream is usually prepared from a circuit description in a higher level description language and a set of constraints, as described in section 2.1.4, by the tools provided by an FPGA manufacturer, or by a third-party such as Mentor Graphics or Cadence.

A mapping of the bitstream to the FPGA resources is shown in Appendix A in detail on the Atmel AT40K FPGA device.

2.2.3 FPSLIC Architecture Overview

This section will briefly introduce the architecture of the FPGA under study, the Atmel AT40K FPGA architecture. The physical used device was an AT94KAL, which is in principle a SoC with AT40K FPGA and AVR processor. Complete architecture overview with the bitstream explanation is given in the Appendix A.

A very brief description of the FPGA part of the FPSLIC: The FPGA is an island-style FPGA, which contains up to 48×48 logic cells. Cells have direct connections to the neighboring cells. Each cell can be connected to one of 5 “local bus planes”, which span over 4 logic cells both in a horizontal and a vertical direction. A repeater row/column is inserted every 4 cells, which allows routing of the signal to the “express buses”, which span over 8 cells and end in another repeater. 10 express wires pass the logic cell in both directions (horizontal and vertical). The logic cell has two 3-input LUTs and rich intra-cell connection capabilities.

2.3 Radiation

A **radiation** is in this work referred to as a physical process of radiation of charged particles (electrons, protons and ions), or radiation of an uncharged neutrons or electromagnetic waves (photons) [34]. The region of interest are particles and particle energies, that are capable of depositing enough energy into the material (the chip in this publication) by the particle (or energy) passing through it or by the particle. The energy ranges over many decades. Natural sources of radiation are described in section 2.3.3.

The primary concern in this work is an energy deposition induced by an **ionizing radiation**. The **ionization** is a process, by which one or more electrons are liberated in a collision of a particle (or an ion) with an atom [65]. The deposited energy has to be high enough; otherwise an *excitation* will occur instead of ionization.

When a charged particle travels through the matter, the particle is being stopped by the matter and it gives some of its energy away into the matter (or is completely stopped there).

Quantities being used for the stopping power losses are: *stopping power*, *mass stopping power* and *linear energy transfer.*

2.3.1 Units and Quantities

Useful radiation units and quantities are described in this section. More detailed information can be found [65], which is the main reference for the labels and letters used in this work.

The SI unit for energy is Joule [J], but electron volt [eV] is rather used for describing the energy of a particle, with a conversion constant $1\,\mathrm{eV} = 1.602\,176\,487 \times 10^{-19}\,\mathrm{eV}$. 1 eV is a too small energy for particles in radiation environments (except for thermal neutrons), therefore keV, MeV and GeV are used more often.

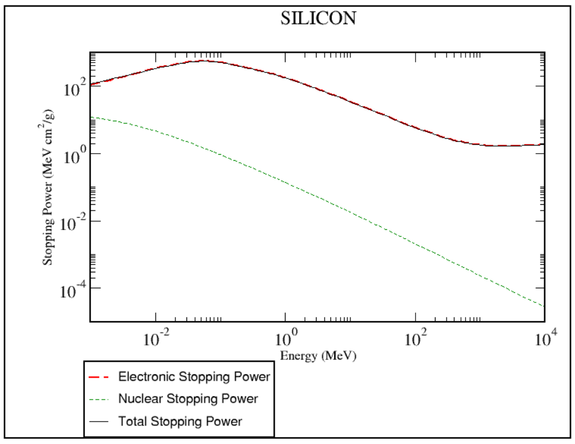
The **stopping power** S is defined as

$$S = \frac{\mathrm{d}E}{\mathrm{d}l} \quad \left[\mathrm{J\,m^{-1}}, \mathrm{eV\,m^{-1}}, \mathrm{MeV\,cm^{-1}}\right], \tag{2.1}$$

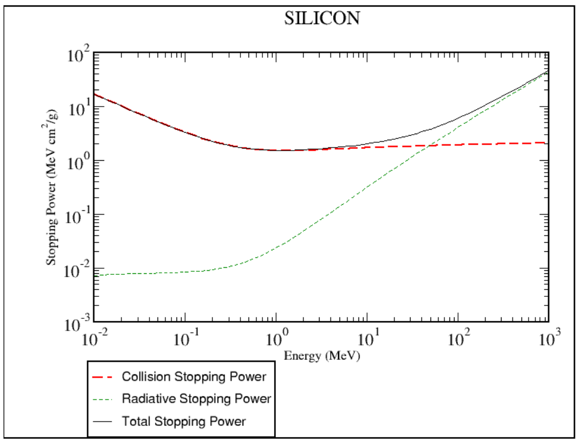
where E is the kinetic energy of the particle and l is the distance of the particle trajectory in the matter. There are 3 principles of an energy transfer:

1. **Electronic energy loss** by Coulomb inelastic collision, i.e. the ionization and excitation of the electron in the target.
2. **Nuclear energy loss** by elastic collisions with the nuclei of target atoms.
3. **Radiative energy loss** by an emission of energy by means of bremsstrahlung emission in the electric fields of atomic nuclei or electrons.

The **mass stopping power**, $\frac{S}{\rho}$, is a material property for charged particles of a given type and energy, is the quotient of $\mathrm{d}E$ and $\rho \mathrm{d}l$, where $\mathrm{d}E$ is the mean energy lost by the



(a) proton



(b) electron

Figure 2.6: Mass stopping power in Silicon. Credit: physics.nist.gov [54]

charged particles in traversing a distance dl in the material of density ρ, thus

$$\frac{S}{\rho} = \frac{1}{\rho}\frac{\mathrm{d}E}{\mathrm{d}l} \quad \left[\mathrm{J\,m^2\,kg^{-1}, eV\,m^2\,kg^{-1}, MeV\,cm^2\,g^{-1}}\right]. \tag{2.2}$$

The mass stopping power can be obtained from the several sources [78, 62, 54] with reasonable uncertainty, that was confirmed by many experiments. Example of electronic component and nuclear component of the stopping power of electron and proton in silicon are shown in Fig. 2.6.

The (unrestricted) **Linear Energy Transfer** (**LET**), L, is a material property for charged particles of a given type and energy. It is expressed as a quotient of dE_{el} by dl, where dE is the mean electronic energy lost by the charged particles due to electronic interactions in traversing a distance dl, thus

$$L = \frac{\mathrm{d}E_{el}}{\mathrm{d}l} \quad \left[\mathrm{J\,m^{-1}, eV\,m^{-1}, keV\,\mu m^{-1}}\right]. \tag{2.3}$$

The LET can be restricted (L_Δ), where the restriction comes from not subtracting the mean sum of the kinetic energies in excess of Δ: $L_\Delta = \left(\frac{\mathrm{d}E}{\mathrm{d}l}\right)_{el} - \frac{\mathrm{d}E_{ke\Delta}}{\mathrm{d}l}$, where d$E_{ke\Delta}$ is the mean sum of kinetic energies, greater than Δ, of all electrons released by the target particle. Secondary electrons with larger energy than Δ are therefore excluded from the deposited energy L_Δ.

Analogically to the *stopping power* and the *mass stopping power*, the LET is often given with respect to the material density ρ:

$$\frac{L}{\rho} = \frac{1}{\rho}\frac{\mathrm{d}E_{el}}{\mathrm{d}l} \quad \left[\mathrm{J\,kg\,m^{-2}, MeV\,g^{-1}\,cm^2}\right].$$

.

The **cross section** σ is an interaction coefficient, that indicates the incidence likelihood of the interaction, where N is the mean number of the interactions at the given particle fluence Φ:

$$\sigma = \frac{N}{\Phi} \quad \left[\mathrm{m}^2, \mathrm{b}\right], \tag{2.4}$$

where [b] is a non-SI unit barn, where $1\,\mathrm{b} = 10^{-28}\,\mathrm{m}^2$. The $1\,\mathrm{b}$ is approximately the cross section of a uranium nucleus.

The **particle fluence** Φ is a quantity, which refers to how many particles incidented (passed through) a unit area:

$$\Phi = \frac{\mathrm{d}N}{\mathrm{d}a} \quad \left[\mathrm{m}^{-2}, \mathrm{cm}^{-2}\right], \tag{2.5}$$

where N is a total number of particles incidented on a sphere of cross-sectional area $\mathrm{d}a$.

The **particle flux** $\dot{N}$ refers to a number of particles in time unit: particle fluence per unit of time, therefore

$$\dot{N} = \frac{\mathrm{d}N}{\mathrm{d}t} \quad \left[\mathrm{s}^{-1}\right], \tag{2.6}$$

where $\mathrm{d}N$ is an increment of the particle number and $\mathrm{d}t$ is the time interval.

The unit that describes particle fluence per time is called a **fluence rate** $\dot{\Phi}$. It is defined as follows:

$$\dot{\Phi} = \frac{\mathrm{d}\Phi}{\mathrm{d}t} \quad \left[\mathrm{m}^{-2}\,\mathrm{s}^{-1}, \mathrm{cm}^{-2}\,\mathrm{s}^{-1}\right], \tag{2.7}$$

where $\mathrm{d}\Phi$ is an increment of particle fluence and $\mathrm{d}t$ is the time interval. The *particle flux* term is often misused on places, where the *fluence rate* as defined above should be used instead.

There are several units describing the acquired **dose** from particle fluence: kerma [Gy], cema [Gy], exposure [$\mathrm{C\,kg^{-1}}$], energy deposit [J], energy imparted [J], specific energy [Gy] and absorbed dose [Gy], as described in [65].

As shown, most dose quantities share the same SI derived unit *gray* [Gy], which has a following relation to the SI unit: $1\,\mathrm{Gy} = 1\,\mathrm{J\,kg^{-1}}$. Older unit rad [rad, rd] is still often used, with a simple conversion factor $1\,\mathrm{rad} = 0.01\,\mathrm{Gy}$, eventhough its usage is discouraged in favour of the Gy unit.

Doses are defined as an energy E deposited in the mass m:

$$D = \frac{\mathrm{d}E}{\mathrm{d}m} \quad \left[\mathrm{J\,kg^{-1}}, \mathrm{Gy}, \mathrm{rad}\right], \tag{2.8}$$

where the energy $\mathrm{d}E$ evaluation is defined differently for kerma, cema and an absorbed dose.

Kerma (for ionizing uncharged particles): the $\mathrm{d}E$ is defined as a mean sum of the initial kinetic energies (including the kinetic energy of the charged particles emitted in the decay of excited atoms/molecules or in the nuclear de-excitation or disintegration) of all the charged particles liberated in a mass dm of a material by the uncharged particles incident on dm.

Cema (for ionizing charged particles): the dE is defined as a mean energy lost in electronic interactions in a mass dm of a material by the charged particles, except secondary electrons, incident on dm.

Absorbed dose: the dE is defined as a mean energy imparted by ionizing radiation to matter of mass dm [65].

Since the dose is a material property, it is directly referred to a material, which is exposed to the radiation, such as silicon: Gy (Si) and rad (Si), or different materials such as silicon dioxide Gy (SiO_2) , Silicon gallium arsenide Gy (GaAs) and many others.

The **dose rate** $\dot{D}$ describes how fast the dose is being acquired. At some applications the result of the irradiation depends on it, not on the dose itself. The dose rate is defined as

$$\dot{D} = \frac{\mathrm{d}D}{\mathrm{d}t} \quad \left[\mathrm{J\,kg^{-1}\,s^{-1}, Gy\,s^{-1} rad\,s^{-1}}\right], \tag{2.9}$$

where dD is the dose increment and dt is the time interval.

2.3.2 Radiation Effects

The radiation is one of the factors, that can drastically decrease the reliability parameters of the electronic component [32]. Apart from radiation, there are other factors decreasing reliability, such as temperature, pressure, supply voltage, EM noise of chemical environment, which are not discussed in this work.

The technology, in which the electronic component is fabricated, plays the key role of the radiation susceptibility and its selection and improvement is the most straightforward way to improve the reliability. This section will focus on the radiation effect on the CMOS devices. There are other technologies (such as GaAs, GaN, SiC), that are more resistant to the radiation effects, but they are not the technologies, that commercial FPGAs are fabricated in.

Radiation effects, which affect the functionality of electronic components, can be divided into two groups:

- Immediate effects caused by a single particle or particle shower, generally referred as *Single Event Effect (SEE)*, which is described in section 2.3.2.3.
- Cumulative doses, which integrate over the time that the component was exposed to the radiation. This includes *Total Ionizing Dose (TID)* , which is described in section 2.3.2.1.

More detailed reading about the radiation effect can be found in book [34]

2.3.2.1 Total Ionizing Dose (TID)

When the electronic component is exposed to an ionizing radiation, it accumulates the energy lost by the mean of ionizing energy loss. The acquired dose is called Total Ionizing

Dose (TID), which is related to the component's material – typically silicon.

The primary mechanism of the dose accumulation and CMOS transistor degradation is a charge collection in the gate oxide, or a bulk oxide, as described in [55]. This charge collection causes a voltage threshold of the transistors. The threshold shift also depends on the dose rate showing a low dose rate (LDR) effect [36, 30] and voltage applied during irradiation [55].

The contributors to the TID are charged hadrons, electron, ions and also gamma (photons) and neutrons.

TID affects the MOS devices by implanting an electric field into the gate oxide or by creating side leakage paths by charge accumulation in lateral oxide. The TID affects bipolar transistors as well, degrading the gain, leakage and also the noise. TID affects the bipolar transistor as well due to charge trapping into oxide.

Irradiation tests for an assurance of a sufficient TID resistance of the electronic device is usually performed by gammas from ^{60}Co source. This method is preferred because of its cheap availability; however it is valid only for environments and devices, where the displacement damage doesn't matter.

2.3.2.2 Displacement Damage Dose (DDD)

The Displacement damage dose is a dose that quantizes the displacement damage to the semiconductor lattice by the energetic particles, which can penetrate into a crystal lattice of the semiconductor. An atom can be displaced through several mechanisms [32]. The resulting crystal contains empty positions of knocked-out atoms, which are stable positioned or clustered elsewhere in the crystal.

Primary Knock on Atom (PKA, the atom released from its position in the lattice by the incident particle) results in a silicon vacancy-interstitials pair, which recombines (90%), or migrates and forms a stable defect (interstitial, vacancy, divacancy, vacancy-interstitial Frenkel defect or vacancy-interstitial Schottky defect). However, the original PKA can only be displaced if its energy is higher than the binding energy of $20 \sim 25\,\mathrm{eV}$.

The displaced atom can be caused by neutrons, protons [58], neutrons and high energy electrons and photons. For and higher proton energy >10MeV and neutrons (1MEV) there is an increasing chance of an inelastic collision, which produce a cascade of lattice defects, which tent to be grouped in clusters. Electrons usually produce isolated point defects, since they can impart only a small energy to the PKA. The ionizing component of the proton energy loss is much greater, than displacement component. Only 0.1% of proton's energy produces displacement [61].

Much alike the TID effect, the degradation is long-term and often has similar long-term degradation characteristics to the TID, although it is based on different physical mechanism.

Typical effect of the displacement dose is a degradation of the bipolar transistor gain and leakage current, as the places of knocked-out atoms serve as recombination centers, especially for the minority carriers in the base of bipolar transistors. Optoelectronic components (optical sensors, transceivers and optocouplers) are more sensitive to the displacement damage, including photovoltaic solar cells.

The displacement damage energy deposition per unit mass of material is given by the product of NIEL and the particle fluence. DDD can be therefore defined also as a displacement kerma (defined in sec.2.3.1) and it directly reflects the displacement damage dose.

The displacement damage dose is often given as an equivalent *fluence* of 10MeV protons or 1MeV SiEq neutrons. For some application like solar cells damage models, 1MeV electron equivalent fluence is also considered. Irradiation tests are usually performed by protons.

Non-Ionizing Energy Loss (NIEL) is a quantity that describes the rate of energy loss due to atomic displacement due to a particle traveling through a material.

The damage to the bulk silicon, according to NIEL concept, is proportional to the final concentration of the defects and not on the particle type or particle energy. This concept (that displacement damage scales with the NIEL) has been found very useful for many semiconductor devices and optical sensors. It was demonstrated by many studies, that the NIEL deposited energy is linearly correlated to the displacement damage, but few studies showed, that it is not valid always.

There are more processes involved in the NIEL - coulomb interaction and both elastic and inelastic nuclear scattering. Coulomb interactions dominate in the production of displaced atoms at proton energies below 10 MeV. At higher energies (above $30 \sim 50$ MeV), nuclear reactions dominate in displaced atom production.

2.3.2.3 Single event effect (SEE)

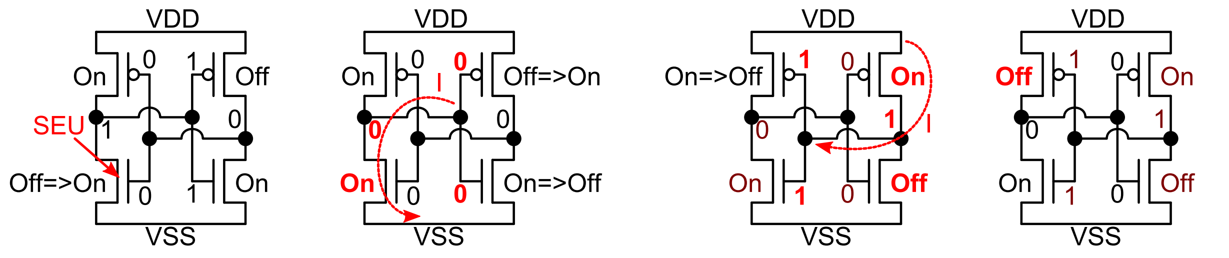
The single event effect is an effect caused by a single particle, which generates an electrical charge in the material. This charge can be caused by the electronic energy loss from the charged particles, ions or by other secondary mechanism initiated by neutral particles, as described in [61, 17, 32].

The charge is deposited along the track in a cylinder with a $< 1\,\mu$m in diameter, where electron-hole pairs are generated. The energy needed to free an electron from the Si lattice is ~1 µm. The deposited charge can be calculated directly from the LET parameter for a given energetic particle (ion) and material, as shown in Fig. 2.6.

The charge collected by the junction, Q_{coll}, which creates a voltage&current spikes, is however different from the deposited charge. The collected charge depends on many factors, such as size, various biasing, substrate, doping, hit spot and other factors. The collected charge Q_{coll} typically varies from 1 to several 100s of fC and is collected in a range of picoseconds up to hundreds of ps. More information about the mechanism of the charge collection, depletion region funnel–shaped change and time scale can be found in [17, 51].

The sensitivity of the electronic component to the SEU is expressed as function of SEE cross section versus the LET, displaying 2 key properties:

LET threshold (LET_{th}) is the minimum LET to cause the SEE. Below this threshold the Q_{coll} is too low to induce an SEE.



(a) Particle hit (b) Propagation (c) Further propagation (d) result

Figure 2.7: Smplified mechanism of SEU in SRAM cell. a) particle hits a transistor in off state; b) charge is collected by the collector of left NMOS and creates current I, which discharge gates of right transistors; c) right transistors toggles and enable current to charge gates of left transistors, as shown by red arrow; d) left PMOS switches off and the circuit reaches stable condition. Note, that only some notes are susceptible to SEU: the drain of NMOS and the source of PMOS (both nodes in non-conducting state)

Saturation cross section (σ_{sat}) – the cross section value, which will not increase its value with higher LET ion/particle.

The SEE can be soft or hard, as described in sections 2.3.2.3 and 2.3.2.3

Soft SEE Soft effects are not destructive to the device and do not persist after a power cycle. Soft SEEs can be classified into following categories:

- **SET (Single Event Transient)** is the voltage or current spike both in the analog and digital circuits. The spike is referred as “glitch” and propagates further in the circuit up to the output. Generally analog circuits are more sensitive to SET by their asynchronous nature. Digital circuits are less affected, because the fast glitch in asynchronous logic is filtered by the synchronization at the input of flip-flops. The SET may be caught by the flip-flop, when its timing matches the latch moment. When SET is latched, one may consider this event as SEU.

- **SEU (Single Event Upset)** is a type of SEE, where the collected charge Q_{coll} is greater than the critical charge Q_{crit}, which is required to cause a change of state of the flip-flop (e.g. SRAM memory, DFF). The SEU directly affects all flip-flops and SRAM memory. The source of the upset is the voltage and current spike, induced by the Q_{coll}, as in SET. This spike has to be localized in sensitive spots (transistors of flip-flops) and has to be large enough to trigger the change of state. Principle of SEU in SRAM cell is shown in Fig. 2.7. The change is not destructible and is easily eliminated by writing a new value or by resetting the device.

- **MBU (Multiple Bit Upset)** is an event or series of events, where more than 1 bit is flipped during a measurement or clock cycle.

- **SEMU (Single Event Multiple Upset)** is an MBU incident, where all bits are upset by a single particle. This event affects typically topologically neighboring memory cells. There are 3 main principles, how SEMU is originated [59]: a) the particle impact angle allow the particle to pass through more cells; b) the diameter of the cylinder, where the charge is deposited, crosses more memory cells and SEUs occur there; c) memory cells are upset by the products of spallation reactions from the primary particle in the chip.

- **SEFI (Single Event Functional Interrupt)** is a special case of SEU, where the SEU occurs in a control logic and control over the device and the function of the device is lost. Unless stucked in the reset state, the device can be reset without the power cycle to restore the correct operation of the device. Affected logic is for example JTAG controller or programming logic, debug controller, bootloader controller, configuration downloader, reset logic and reset generators, etc.

Hard SEE Hard SEEs cause permanent functional change to the device, which is irreversible. Hard errors are often physically destructive to the device. Hard SEEs are classified according to the mechanism of destruction:

- **SHE (Single Hard Error)** is the common designation for the irreversible change in the operation, thus permanent damage, such as gate oxide rupture.

- **SEL (Single Event Latchup)** is a short circuit, triggered by the particle/ion that can happen in the parasitic n-p-n-p semiconductor structure, similar to the thyristor. Once triggered, the structure sustain in the latchup state because of its positive feedback. The short circuit between power supply and ground starts to dissipate heat. Depending on the resistance, the latchup can be a) fatal, when the current density exceeds safe current limits, or b) temporal (soft), when a latchup generates heat, increased current consumption, but after the power cycle the device recovers. The SEL typically requires to power cycle the device (when the latchup occurs between the supply voltage and ground), but can occur also within signals, where the latchup can be stopped by the change of values. The latchup can be detected by a sudden increase of the current consumption. SEL immune devices are typically considered to be SEL-free for $\mathrm{LET_{th}} > 120\,\mathrm{MeV\,cm^2\,mg^{-1}}$ (for silicon, the particle/ion would have to deposit more than $280\,\mathrm{MeV\,cm^{-1}}$ to the sensitive region in order to cause the latchup).

- **SEB (Single Event Burnout)** is a destructive event, where the structure is destroyed by an excessive current and heat, which is generated. Susceptible devices are power MOSFETs and BJTs and also some CMOS structures.

- **SEGR (Single Event Gate Rupture)** is a destructive event, when the gate oxide is broken. This effect can manifest itself by an increase of the gate leakage, or as a complete failure of the device. It can happen, when high gate voltage is applied and a particle/ion hits the stressed gate. The most susceptible devices are therefore power MOSFETs and also some programmable devices and very thin oxide VLSI structures, such as EEPROM during writing and erasing.

2.3.3 Sources of Radiation

The sources can be natural or artificial. The natural sources can be terrestrial or extraterrestrial (cosmic). The **terrestrial** sources includes a natural occurrence of primordial[3] radionuclide at the earth's surface, such as uranium ($^{238}_{92}$U), thorium ($^{232}_{90}$Th) and potassium ($^{40}_{19}$K), which suffer from the decay (α, β) or spontaneous fission and contributes to the background by the worldwide average human-body-effective dose of 0.48 mSv a^{-1}. Products of the decay are radium ($^{226}_{88}$Ra) and radon ($^{222}_{86}$Rn), which is the highest contributor to the background radiation with 1.1 mSv a^{-1}. Overall average dose from the background is 2.4 mSv a^{-1} [66, 67].

On the Earth there is also a circulation of the radioactive carbon ^{14}C, which does not contribute to the radiation background significantly (only 12 µSv a^{-1}), which is not a primordial radionuclide, but an cosmogenic radionuclide created in the troposphere and stratosphere (from 9 to 15 km) by a reaction of thermal neutron from the products of cosmic rays and a nitrogen: $\mathrm{n} +^{14}_{7}\mathrm{N} \rightarrow^{14}_{6}\mathrm{C} + \mathrm{p}$. Then the ^{14}C goes into the cycle of living organism and is often used for radiocarbon dating.

There are two main sources of the **extraterrestrial** radiation: the cosmic rays and solar activity. It is believed, that the cosmic rays are mainly originated from the Galaxy, therefore it is commonly referred as a GCR (Galactic Cosmic Rays), the solar activity is usually referred as SPE (Solar Particle Events). The radiation from GCR varies with the altitude (increase with altitude) with and latitude (increase in the proximity of Earth's poles), because it is being shielded by the atmosphere and deflected by the geomagnetic field.

The products of extraterrestrial radiation contribute to the background radiation with the dose of 0.39 mSv a^{-1}in average [67].

The GCR are composed mainly of protons (90%), α particles (9%) and heavy ions (1%). When the GCR energetic particle reaches Earth's atmosphere, it collide with the air and creates a cascade of interactions, which finally form a shower of particles, which is further stopped in the atmosphere. Depending on the primary energy, the air shower is absorbed in the athmosphere, or it reaches the surface of the Earth, spreaded typically within with diameter of ~100 m.

The artificial, created by human, can be nuclear, medical and experimental. Nuclear sources are mainly nuclear power plants and nuclear scientific reactors for irradiation, activation of radiopharmaceuticals and production of radiation sources $^{60}_{27}$Co, $^{90}_{38}$Sr and many others [35]. Other nuclear sources are the primary nuclear explosion, nuclear accidents and contamination of the environment by these accidents and nuclear weapon tests, which still contribute to the background radiation by the dose of 5 µSv a^{-1}.

[3]Nuclides, that exist on Earth since the Earth was formed

Medical sources are devices used for diagnostic radiography, such as CT (Computed Tomography) scans, X-ray radiography, positron emission tomography (PET), or radiosources used for radiotreatment.

Radiation also occur in the high energy physics accelerator in the form of radiation from products of collisions (either fixed targets or colliding beam) or any particle beam interaction (such as interaction of the beam with residual gas in the vacuum), synchrotron radiation (where particles are being accelerated) or at the irradiation test facilities, where samples of electronics are being tested for their radiation hardness.

2.4 Radiation Environments

The radiation environment has an extreme impact on the reliability of electronic system. Various sources of radiation were described in section 2.3.3.This section will show the basic classification of radiation environments, where the reliability of electronics is in concern, and the main radiation contribution in those environments.

2.4.1 Ground Level

In the late 1970s, first issue with SEUs was observed in DRAM (Dynamic RAM) chips at the ground level [46]. It was pointed out, that the sources of the SEUs were alpha particles, which cannot penetrate the package and have to be originated from the package itself by contamination of radioactive impurities (such as uranium and thorium), which emit the alpha particles by the radioactive decay. The emission of alpha particles can be significantly reduced by purifying the packaging material below the level of $0.001\,\alpha/\mathrm{cm}^2/\mathrm{h}$.

Another source of alpha particles in packaging is the lead solder in solder bumps in flip-chip packages. Isotope $^{210}_{82}\mathrm{Pb}$ and its decay product, $^{210}_{84}\mathrm{Po}$, is strong emitter of α particles at energy 5.3 MeV. α particle from within package is often produced from the boron ($^{10}_{5}\mathrm{B}$) which captures a neutron: $^{10}_{5}\mathrm{B} + \mathrm{n} \rightarrow {}^{7}_{4}\mathrm{Li} + \alpha$. Boron is used as a dopant of P-type semiconductor and in BPSG (Boro-Phospho-Silicate Glass,), which is often used as an insulating layer between the metal layers of the IC. The boron issue can be solved by using another isotope of boron, $^{11}_{5}\mathrm{B}$, which has up to 6 orders of magnitude low thermal neutron cross section compared to $^{10}_{5}\mathrm{B}$.

The cosmic ray contributes to the SEU significantly. The primary particle of cosmic rays usually interacts in the Earth's atmosphere and created secondary particles, or particle shower. Secondary particle capable of SEE generation are energetic neutrons and protons. The ground level flux of neutrons is typically $10\,\mathrm{neutrons/cm}^2/\mathrm{h}$ and increases by factor of 2.2 for every 1 km in altitude above the sea level. The typical increase factor for radiation in airplane cruising altitude$\sim$ 12 km is $95 \sim 561$, depending on latitude and longitude [26].

Cosmic ray flux varies also with the latitude (higher on poles, lower near equator), longitude (due to misalignment of geomagnetic and Earth's poles by 5°) and solar cycle activity [JEDC89]. The solar activity has an opposite effect – high solar activity decreases the amount of GCR radiation on the ground. The relative flux according to the location and activity can be easily calculated [1, 2].

The SEE rates are low, but contribute to the FIT rate of memories and FPGAs significantly. It has been shown by many studies [44, 43, 42, 68], that the SEU-induced error rate (SER) is typically 100~500 FIT/Mb (Failures In 1×10^9 h per 1 Megabit of memory) depending on the technology, while configuration RAM having usually lower FIT typically than fast block RAM. The contribution of α particles from the package is typically below 100 FIT.

The LET of the primary particles is rather low, compared to other environments, and primary source of SEU is a secondary emission from the radioactive decay of nucleus, which capture the neutron.

Ground level does not usually have any radiation dose issue, since the acquired dose is fairly low compared to doses (TID, NIEL) that initiate functional changes.

2.4.2 Space

The space radiation outside Earth's atmosphere is very hostile for electronics. Proper shielding is necessary, though complete shielding is not feasible due to extreme amount of material needed. Aluminium and hydrogen-rich composites are often used today to shield from cosmic rays and solar activity.

The distribution of radiation varies according to the Earth's orbit, such as Low Earth Orbits (LEO, 160 km $\sim$ 2000 km), Medium Earth Orbit (MEO, 2000 km $\sim$ 35 786 km), Geostationary orbit (GEO, 35 786 km), High Earth Orbit (HEO, above 35 786 km) and elliptical orbits, which passes many altitudes. The geomagnetic field is capable of deflecting some radiation in the close Earth proximity, but the magnetic field hold two layers of cloud of particles, generally known as Van Allen Belts. The inner belt traps particles between 10 000 km $\sim$ 6000 km, the outer belt between 13 000 km $\sim$ 60 000 km. The radiation in the Van Allen Belts consist of omnidirectional fluxes of protons of various energies (10 keV $\sim$ 400 MeV) and fluxes up to $1 \times 10^5\,\mathrm{cm^{-2}\,s^{-1}}$, and electrons and electrons of energy up to 7 MeV and fluxes in the order of $1 \times 10^8\,\mathrm{cm^{-2}\,s^{-1}}$ (for the low energy electrons).

The acquired dose varies with the trajectory. The maximal dose rate can be acquired in the inner Van Allen Belt, $25\,\mathrm{Gy\,a^{-1}}$. The proper way to estimate the fluxes and SEE rates at the Earth orbit (and beyond) is to use data from the model, such as AP-8/AE-8 RADBELT models for protons/neutrons in Van Allen Belts, JPL92 model (JPL Solar Energetic Particle Event Environment) and many other models like CREME, HZETRN and Monte-Carlo models FLUKA, KORSIKA, CREME-MC, GEANT4 and others.

2.5 Dependability terms

This section summarizes basic terms, which are being used in this work.

Fault is a logical manifestation of a physical defect, or a logical manifestation of a SEE in the scope of this work. Faults may or may not have influence on the behavior inside the system for which another term is used, an error.

Error is a manifestation of the fault in the behavioral alternation of the system within its boundaries. In FPGA, most common errors are modification of the datapath. Other examples from digital logic are data words may have a bit error, state machines that enter wrong state etc. Errors may not be necessarily propagated to the output of the system boundaries, especially when the system is designed to cope with errors: wrong data words can be corrected using an error recovery techniques, state machine can recover.

When the error is propagated to the output without fixing, it may lead the system into a failure. **Failure** is a state, where the system exhibits a different behavior, than is supposed to and that does not conform to the specification anymore. The failures are usually in concern and dependability attributes are being referred to them.

Reliability according to the IEC standard [27] is an ability of the item to perform a required function under given condition for a given time interval. There are more quantities describing the reliability of the system:

Failure In Time (FIT) is a count of failures, that will occur during a period of $1 \times 10^9\,\mathrm{h}$ (billion hours) of device operation. This is equivalent to number of failures during 114077 years of continuous operation.

Failure rate is the frequency of failures of the system, which is being denoted as λ.

Mean Time Between Failures (MTBF) is another quantity of reliability, which express arithmetic mean time between failures, assuming, that the failure is immediately repaired. MTBF is usually given in hours or years of device operation.

Mean Time To Repair (MTTR) is an arithmetic mean of time required to repair the device, where the time interval of repair starts at the moment when the failure is discovered and ends when correct operation of the device is recovered. Sometimes is the time interval measured since the repair starts, instead of the discovery (occurrence) of the failure.

Availability is a ratio of a the functional (failure-free) time and the total operating time, which can be also expressed as a $1-U$, where U is unavailability (probability in time that the device will not operate correctly due to failure or repair. The availability is also given by the number of nines in the availability number. For example 6 nines are equal to the availability of $0.999999x$ (or $99.9999x\%$), where x is a sequence of digits not starting with 9.

The **Soft Error Rate (SER)** is a rate of SEE in the system, which is usually being expressed as FIT or MTBF quantities. The typical SER of modern FPGA from XILINX is between $80 \sim 400\,\mathrm{FIT/Mb}$ [76].

Digital circuits can implement error detection schemes, so-called **CED (Concurrent Error Detection)** [56]. The method of CED is to calculate (predict) check bits in parallel to the circuits, sharing only the same inputs. The predictor is independent, i.e. concurrent to the original circuit.

Error in the circuit is detected, when the predicted check bits and check bits calculated from output of the original circuit do not match. A **codeword** is any valid combination of the output and predicted bits, for which the predicted check bits and check bits calculated from the output do match.

There are 3 formal quantitative properties of electronic circuits supporting the CED (Concurrent Error Detection), which can help with circuit classification:

Fault Secure (FS) is a circuit, which for every fault the circuit either generates an invalid codeword, or a proper output. Correct codeword for erroneous output is not acceptable for fault security. In other words, fault secure circuits do not produce incorrect output vectors that would belong to a codeword. This has a consequence, that the error will be always detected by the codeword and no error can pass undetected by the codeword checking

Self-Testing (ST) is a property declaring, that for each fault there is at least one input vector), that produces output vector not belonging to the proper codeword. This has a consequence for the testing of the circuit, that the fault in the circuit can be identified just by checking the validness to the codeword. For some input vector combination, the fault may pass undetected by the means of codeword checking.

totally self-checking (TSC) circuit is a circuit that is both fault secure (FS) and self-testing (ST).

Chapter 3

State-of-the-Art

This section will present current State-of-the-Art in fields of:

- Reliable FPGA devices
- Soft error mitigation techniques in standard SRAM-based FPGAs
- Fault models for SEU in the configuration memory of an FPGA

3.1 Reliable FPGA Devices

There are several topics to be addressed when one needs to increase the reliability of FPGAs. The most eminent issue of FPGA reliability is the susceptibility of the SRAM cells (mechanism of SEU in SRAM shown in Fig. 2.7), registers and all flip-flops. In modern FPGA, configuration memory (referred also as CRAM or CSRAM) typically represent the largest number of bits, block memories second largest number of bits, distributed memories in CLBs third largest number of bits and core Flip-Flops the fourth largest amount of susceptible bits. Other, low count bits are I/O registers and FPGA control registers, which may cause the SEFI.

The configuration memory is typically less susceptible to SEU, than other user memories. The reason for that is different design and different purpose (Configuration memory does not need to be so fast). Measurements done at the accelerated beam test with protons showed, that cross section of SBU typically $10^{-14}\,\mathrm{cm}^2/\mathrm{bit}$ (depends strongly on technology, varies from $5 \times 10^{-14}\,\mathrm{cm}^2/\mathrm{bit}$ to $5 \times 10^{-15}\,\mathrm{cm}^2/\mathrm{bit}$, the smaller technology has smaller cross section), cross section of MBU is smaller than SBU by factor $2000\times$(for older technologies) to $30\times$ smaller (the newer technology) and the SEFI cross section is smaller than SBU by factor from $10^{-5}\times$ to $3 \times 10^{-4}\times$ (for the modern technology) [57].

Reliable applications address two problems: the protection of the FPGA configuration and protection of user memory elements. Section 3.1.1 describes methods, how the SEU susceptibility of configuration memory can be overcome. One cannot completely omit usage of registers in FPGA, but those registers can be hardened by hardware and also by design, as described in section 3.1.3.

3.1.1 Hardening of Configuration Memory

The mainstream of commercial FPGA devices has the configuration stored in the SRAM type memory, which holds the configuration since the start-up, where it is streamed from the external EEPROM memory. Once the EEPROM or flash EEPROM is programmed, the data itself is much less SEU susceptible, than in the SRAM memory. A simple idea would be to replace the SRAM cell in the FPGA by another form of storage elements. This idea has been realized by following commercially available devices:

- Replacement of the SRAM with **antifuse**. Antifuse is a special structure, which is made conductive, using an excessive current pushed through the antifuse. Nowadays, two methods are used: Polysilicon-to-Diffusion (leaving typically $600\,\Omega$ connection) and Metal-to-Metal (leaving typically $80\,\Omega$ connection). It is only one-time programmable, it is therefore kind of PROM. Antifuses are used in many radiation tolerant PLDs from Microsemi (former Actel), like the old RT [3] and SX FPGA series, ACT 1, ACT2 and ACT3 FPGA series and FPGA series from current production of Microsemi: AX, RTSX, RTAX FPGA series.

- **Flash**- or **EEPROM**-based PLD: This storage of configuration is common for CPLD devices. Flash FPGAs are produced by the Microsemi, namely ProASIC3, IGLOO2 and IGLOO.

- **Mask-programmable FPGA**. Technically speaking, these devices are not “field programmable” devices anymore, since they have to be configured before they enter the manufacturer foundry. Better designation of such devices is MPGA (Mask-Programmable Gate Array). There is typically only one (but can be more) metal layer, which configure the interconnection. This allows a cost reduction compared to full custom ASIC, because only one (or more) lithographic mask instead of full set of mask is needed. Disadvantage of MPGA is that they cannot be reprogrammed nor programmed after the manufacturing. Every MPGA is typically bound to a SRAM-based FPGA, on which the application is developed and then is transferred to the MPGA by a simple transition without need of proceeding to the FPGA-to-Custom-ASIC design flow. Timing constraints are kept from the FPGA designs and typically even improved. Representatives of such approach are Xilinx Hardwire Logic Cell Arrays [28]. Altera HardCopy ASIC can be considered also as a MPGA, though having more structural differences between them and associated FPGAs and being faster [6]. Others are ISSP platform from NEC, via configurable ViaASIC from Triad Semiconductors and Nextreme devices from eASIC company.

All solutions solve the issue of SEU susceptibility of the SRAM configuration memory and configuration loss due to SEU. All presented configuration “memories” have a zero FIT reliability and zero SEU cross section of that memories.

3.1.2 Configuration Scrubbing

Apart from the in-hardware hardening of the configuration memory, software or semi-hardware techniques for securing of the SRAM configuration is being offered for many FPGA. Basic idea is to keep the traditional SRAM configuration memory, but actively take care for any changes that occur in that memory. CRC code is used for detection of SEU and ECC may be also incorporated in order to provide easy repair functionality. Techniques, which are currently being used in commercial FPGAs, are:

- **External readback and configuration scrubbing** are techniques for detecting and repairing faults in the bitstream memory of running FPGA from outside of the FPGA using the programming interface, such as JTAG or SelectMAP interface at Xilinx FPGAs [23]. Configuration scrubbing is de-facto a partial reconfiguration, which has to be done carefully, since it is performed on running designs and values stored in the reconfiguration region. This technique does not prevent from SEFI in the configuration logic, which can be recovered using dedicated pin. This technique also need an external device controlling the readback and scrubbing, as well as the system power-up configuration download, which should be radiation hardened.

- **Internal configuration readback and scrubbing** uses similar techniques as the external one, but can run standalone an may automatically repair single bit errors and also a couple of adjacent bit errors using the CRC checksums and/or ECC protection. Unlike the external readback, internal configuration scrubbing does not require any external controller. It can be controlled completely by hard IP, or via dedicated soft IP implemented in the user logic of the FPGA. The Stratix V FPGA from Altera is capable of detecting up to 5 bit error and fix single-bit and double-adjacent errors without the need to reconfigure the device [9]. Such functionality is also provided by the Xilinx SEM (Soft Error Mitigation) IP core, providing also testing functionality, which is described in section 3.2.

All methods over the SRAM configuration memory repair have a disadvantage of the latency before the error is detected and fixed. The configuration is checked periodically, block by block. The latency is a function of the recovery solution clock frequency and FPGA size, typically $10\,\mathrm{ms} \sim 100\,\mathrm{ms}$ [75, 9], which might be too late for some applications. The disadvantage of the checking circuitry is the SEU sensitivity of the checking and scrubbing circuitry itself, especially when is implemented as another IP core in the FPGA. Such IP core has its limitation in high radiation environment.

Almost all FPGA devices provide build-in CRC circuitry for the configuration memory, typically 16-bit or 32-bit CRC hashes. The CRC can be generated for each configuration frame individually, and/or for larger blocks. The CRC is recalculated during the FPGA configuration process and verified with the precalculated CRC.

The functionality of bitstream self-repair is relatively new. The Xilinx SEM IP core was released in 2009 and works only with the new devices.

3.1.3 Hardening of User Memory Resources of FPGA

Second largest SEU susceptible memory is the **user memory** inside the FPGA. Typical sizes are up to tens of megabits. The user memory represents typically $\sim$ 10% of all flip-flops (flip-flops in CLBs only $\sim$ 0.5%, distributed RAM $\sim$ 5%, but the distributed RAM is counted as configuration memory, since it is built from LUTs). The On-chip memory error checking is incorporated into FPGA from the Altera company since Stratix II device, which has been improved in Stratix 5 by the interleaving and providing enhanced multi-bit correction [7]. Stratix V solution from Altera incorporates hardware IP cores for various memory configurations, for example 32 bit $\rightarrow$ 40 bit encoder and 40 bit $\rightarrow$ 32 bit decoder with correction, which is being used for M20K user memory blocks [7]. Xilinx provides similar error correcting functionality using the SECDED Hamming(72,64) code only for the 512×64 RAM configuration, which is available for Virtex-7 and some older FPGAs [73]. Securing of the user memory by hardware encoder and decoder can be found in many other FPGAs, such as Igloo2 FPGA from Microsemi. Memory organization allowing parity checking (1, 2, 4, 9, 18, 36 and 72) is very common even for FPGAs, which do not incorporate hardware correction.

User registers are not commonly subjected to SEU hardening, leaving basically only 2 options of implementation:

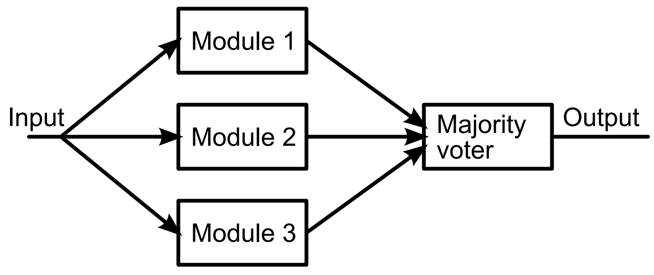
- Usage a FPGA, which offer SEU-hardened flip-flops. Such devices are RTSX, RTAX FPGA from Microsemi, which provides in-cell TMR hardened registers.
- Usage of redundancy. A simple approach is a replacement of a cell with register by 3 cells (with 3 registers) and a voter. This can be done manually, but many tools are available for automatic triplication. Automatic design triplication is usually provided directly by the manufacturer, like TMRtool from Xilinx [70], or by third parties, such as Precision Hi-Rel Synthesis software from Mentor Graphics or FPGA design tools from Synopsis for the Altera and Microsemi FPGAs.

3.1.4 SEU Mitigation

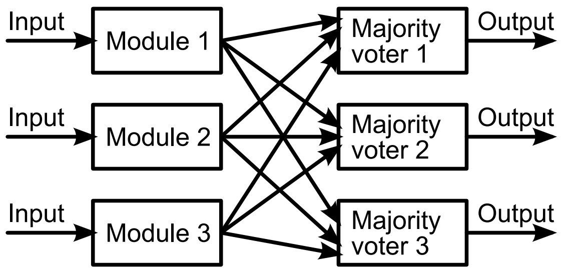
Previous section showed, what technologies are used, when one needs a reliable solution based on the FPGA. The selection of optimal FPGA for the target application is a tedious process of balancing between the cost and provided SEU resistance, which has to be done carefully, when the application will be running in the radiation, such as in space, avionics or high energy physics experiments.

There are many techniques for improving the design reliability just at the design level, which usually contains some sort of redundancy. These techniques are called Concurrent Error Detection (CED) [52]. Such CED techniques are for example parity coding, duplex system or triple modular redundancy.

The **TMR (Triple Modular Redundancy)** is the well-known and most commonly used technique for SEU mitigation. The idea of TMR is triplicating the functional block and those 3 results are compared by majority voter, as shown in Fig. 3.1a. Obviously, the



(a) Simple TMR principle



(b) TMR with triplicated voters

Figure 3.1: Triple Modular Redundancy (TMR)

failure of a single module would be outvoted by the 2 remaining modules. The bottleneck is the majority voter itself, which has to be hardened against SEU.

When TMR is implemented in a single FPGA, any single (non-triplicated) net is a second bottleneck, since path can be easily broken by the SEU. Both issues can be solved by triplication of everything, as shown in Fig. 3.1b, including majority voters, inputs and outputs. Even complete triplication in single FPGA does not safeguard the SEU bulletproofness. Following study showed that even a single SEU can lead to the TMR failure:

L. Sterpone and M. Violante from Politechnico di Torino presented an important study of dependability of TMR designs implemented in SRAM-based FPGA, based on a detailed knowledge of FPGA architecture and configuration memory [64]. They proposed an analytical computation, which analyzes the design as soon, as the placed and routed model of the designed circuit is available. Analysis is based on a graph model, which has vertices composed of logic cells and routing input/output port, and edges composed of hard-wired routing segments and programmable interconnection points represented as routing edges. The vertices are than colored by the affiliation to the tripled module. SEUs are modeled by insertion/removal of the edges to/from the graph. At certain condition, a single SEU can violate the TMR. Violation list of those bits was made further subjected to fault injection. Results surprisingly show, that for some designs (Mul8, Add16, Add8 and elliptic Filter) have failure rate up to 13%, even when implemented as TMR architecture [63]. They also proposed a methodology to improve the failure rate, which is based on iterative process of analyzing the routed TMR design and creating large set of constraints to avoid the violation of TMR in the next place&route iteration. The static analysis of the SEU impact requires few minutes to analyze the impact, which is still factor $10^2 \sim 10^3$ faster, than the fault injection [64].

The most bulletproof TMR design would be that one composed of 3 independent FPGA devices, each implementing one module and majority voter. Even more, triple FPGA design is able to solve the SEFI issue and can continue working seamlessly even when one FPGA needs complete reset or power cycle.

It should be noted, that the TMR requires more FPGA more resources, because of a) $3\times$ more flip-flops; b) $3\times$ more I/Os; c) $> 3\times$ user logic (can be more than $4\times$, because

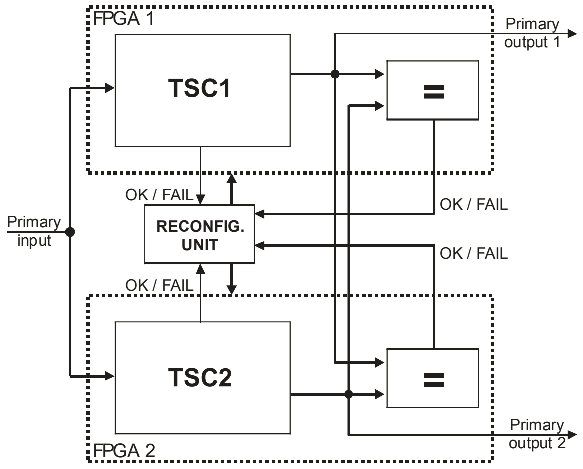


Figure 3.2: Duplex system

voters have to be added); d) > 20% on reducing of clock frequency.

Other solution form improving is a duplex system, such as one proposed in [A.2] and shown in Fig. 3.2. The design has to be duplicated and split into 2 different FPGAs. Unlike the TMR, the duplex system lay a strong requirement on the secured design: It has to be totally self-checking (explained in section 2.5), which basically means, that the design has to be modified, typically by adding a security code predictor, which leads to a further area requirements.

3.2 Previous Results and Related Works

Some works have been done, but only few of them before 2006, where started this work, which originally began even 2 years before 2006. FPGA manufacturers were forced to take the SEU issue into the account, since the growth of size of FPGA is almost exponential and despite decreasing SEU cross-section per Mbit, the overall FIT rate is also increasing exponentially. There is an increasing probability of errors due to SEUs even at the ground level, caused by factors described in Sections 2.3.3 and 2.4.1. The hardening of the physical devices was described in previous section, 3.1. Following sections will describe the State-of-the-Art in current estimation of SEU impact and mitigation techniques.

3.2.1 SEU Impact Derating

All producers of big FPGAs are aware of the SEU sensitivity of the configuration RAM. SEU represent a reliability issue especially for the large FPGA, where the susceptibility of the configuration memory is below 100 FIT/Mb, but due to enormous size of the FPGA and the configuration memory (up to 500 Mb), the SER may drop to a few incidents per year, deeply below the acceptable level. It is also known, that only some bits in the configuration memory cause an error. This was not only expected, but also proved empirically by many tests, including results of this work. As a result, a **derating factor** for the SEU impact is being used for estimation of overall system reliability.

Xilinx claims, that as little as 2% $\sim$ 10% of all SEUs actually cause a system error [74] and even designs occupying 100% of CLBs use only $\sim$ 20% of the configuration bits [26, 25]. Other sources from Xilinx claim, that less than 30% of all configuration bits are used in any design and the SEU impact rate is often less than 10% and typically less than 5% [33].

Altera also claims that in reality only 10% of configuration bits typically affect a given design due to low routing utilization in even full designs [5].

Xilinx came with a term **essential bits**, which is a general classification of bits in bitstream. Essential bits are bits that may have effect on the functionality of the design in FPGA, non-essential bits are “don’t care” bits. This classification helps to determine the SEU susceptibility and overall FIT rate. Moreover, the information about essentiality of all bits can be stored in the look-up memory outside the FPGA (next to the configuration itself in the external flash) so that the SEM core can directly look the essentiality up. The only requirement is a double sized flash chip and a program that predetermines the essentiality. The SEU essential bits calculation is provided by both the ISE Design Suite and the Vivado Design Suite.

Previous approach, that Xilinx used on Virtex-II designs, was the Xilinx **SEUPI** (SEU probability impact) special tool flow, which estimated the ratio of configuration bits used in a design (such bits were labeled “care” bits). This tool flow used the JBits software [69]. SEUPI is also referenced by Xilinx as a derating factor rather than designation of the tool chain [42].

SEU FIT Rate Calculator is a tool from Xilinx company, which offers an easy calculation of the FIT rate based on the following parameters: device being used, elevation, longitude, latitude, solar activity, SEM frequency, essential bits amount and usage of error correction in Block RAMs. This tool made just the calculation very user friendly, but does not provide any calculation, that would not be possible without this tool, since all reliability data is publicly provided.

Common approach for calculating a SEU impact on the reliability of the device can be summarized to following procedure:

1. Obtain reliability data. This data is often provided by the manufacturer by the means of neutron cross section σ per one bit, α-induced FIT and overall FIT at ground level.

2. Obtain derating factor by one of following methods:

   (a) Emulation of the SEU by bit fault injection and recording number of sensitive bits.

   (b) Accelerated beam test (neutron, proton or heavy ions), where both total number of SEUs and numbers of those having functional effect and those leading to SEFI are being recorded. This is the closest method to the real world application. It can be applied even on parts lacking the FIT rate specifications. This method is most expensive.

(c) Estimation performed by the EDA software, which has a direct overview about used resources of the FPGA and therefore is capable of precise prediction of the fault effect. The estimation is always pessimistic.

SEUs are not the only contributors to the overall system reliability. The other factors are high temperature during operation, package reliability itself, moisture and mechanical issues from soldering and mechanical stresses due to temperature variations.

Xilinx pro provide information that it typically takes from 10 to 100 upsets to actually cause a functional failure and recommends to use the worst-case estimate factor of 10 as a derating factor [42]. Overall, typical factor derating the impact of the SEU in FPGA is considered to be 10.

3.2.2 SEU Impact Simulation and Testing

Bellato at al. [18] showed an analysis of the effect of SEU in the SRAM-based FPGA. Analysis was based on 2 principles: by making hypotheses on the meaning of every bit in the configuration memory and by irradiation testing, that was to validate the hypotheses. Hypotheses on the bitstream meaning were based on exploiting of the available information and tools dealing with the device configuration memory. The paper considered the XCV300 Virtex FPGA from Xilinx. This paper also suggested following classification of FPGA bitstream faults, specifically *designed for the interconnection matrices* (which have several inputs and outputs) named Programmable Interconnection Points (PIPs):

- *Open*: The connection in the datapath is removed;
- *Bridge*: 1st PIP net is deleted and new bridge from 2nd PIP net, which drive the same output node as the 1st PIP net, is established, while the 2nd net is kept;
- *Input Antenna*: An unused PIP net is connected to a used output node of the interconnection matrix. The resulting behavior is unknown, since the output pad driven by an unknown logic value;
- *Output Antenna*: A PIP net, which is connected to an unused output node of the interconnection matrix, is attached to another net. This has no influence on behavior;
- *Conflict*: A new PIP net is created, that connect used input node of the interconnection matrix and another used output node of the interconnection matrix;
- *None*: A new PIP net is created, but does not connect to any used node of the interconnection matrix;
- *Others*: Cannot be classified in any of the above classes,

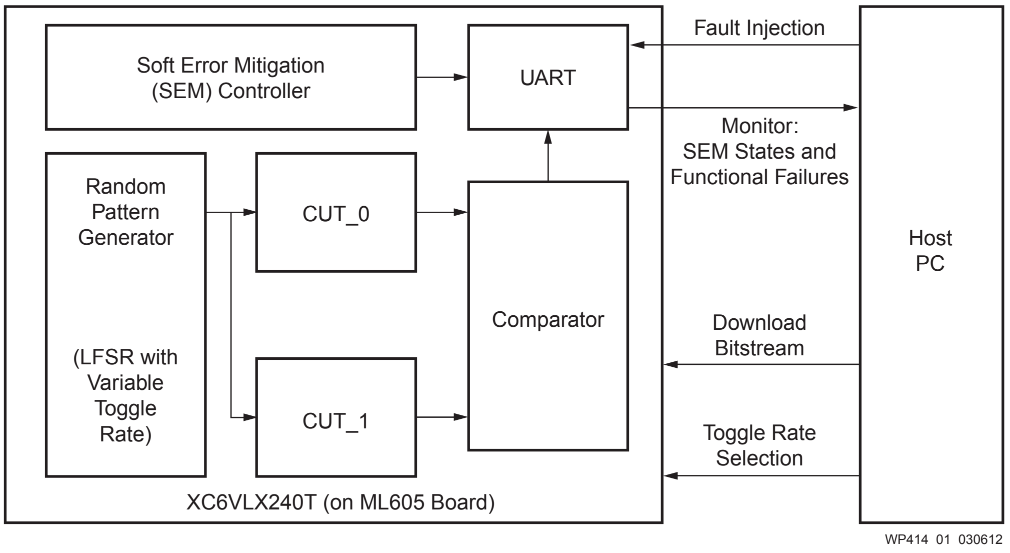


Figure 3.3: SEU emulator solution presented by Xilinx in [74]

while the CLB resources are classified as follows:

- *LUT defect*: LUT content modification;
- *MUX defect*: MUX selection bit changes the path of the input to the CLB;
- *Initialization defect*: Initialization bit produces a modification of the behavior of the internal components of the CLB.

The result of the irradiation showed, that in interconnection matrix, *Conflict* (31.9%) is the most frequent fault causing a failure, followed by *Open* (23.8%), *Bridge* (14.5%), *Others* (7.0%), *Input Antenna* (2.8%),*Output Antenna* (0%) and *None* (0%). The largest contributor to CLB failure is the *MUX* category (11.9%), followed by *LUT* (7.9%) and Initialization (0%). The results were obtained from the irradiation, where 454 bits in total caused the device failure.

A white paper, demonstrating how the essential bit can be further derated to those having functional effect, was presented by Xilinx in 2012 [74]. An emulator structure, very similar to the emulator structure suggested in this thesis, was proposed. Schematic diagram of the emulator is shown in Fig. 3.3. The emulator incorporated 2 circuits under test, TPG and comparator on the ML605 board with Virtex-6 [74]. The Emulator design also contains the SEM controller [75], which was used for injecting errors into the bitstream. Impact on the functionality is then observed by the comparator. Results are sent to the host PC through UART . Only the essential bits of the bitstream were used for random injection. Measurements were performed only by limited amount of bits (3000), which should be enough to determine the derating factor from ratio of the failures to number of tested essential bits. 14.13% bits from bitstream of the design were essential and 761 failures were observed with 3 of them unrecoverable. The further derating factor is 3000/761 = 3.94, which derates the count of essential bits, resulting in failure ratio $14.13\%/3.94 = 3.58\%$ of total amount of bits from the whole bitstream, that cause a failure.

Chapter 4

Overview of My Approach

The main purpose of the research is to obtain precise information about the effect of SEU and improve the fault model over the traditional stuck-at fault models, which are used extensively by the automatic test pattern generators. The approach, which will be described in this section, aims at the highest understanding of SEU effect in the configuration memory for the combinatorial circuits implemented in the FPGA.

Following sections will present 3 different classifications of different subjects:

1. Fault classification (section 4.1), which is necessary to quantize reliability parameters and reliability improvements of circuits secured with parity code.
2. FPGA resource classes (section 4.2), that inform about the responsible resources, that the error (or failure) occurred in.
3. FPGA fault models (section 4.3), which is the essential condition for the prediction of the effect.

The classifications in sections 4.2 and 4.3 would have never be possible without the sufficient knowledge of the FPGA structure and without the exact bitstream mapping of the FPGA resources and the bitstream. Because the FPGA structure is not publicly known (it wasn't at the beginning of the work) and it is a highly guarded vendor secret, the FPGA structure and bitstream had to be reverse engineered. The procedure (Appendix B) and results of that FPGA analysis with a complete bitstream description (Appendix A) are described outside of the main text in detail, because their difficulty might easily mislead the reader from the main topic, and because they are not necessary to understand for understanding results of this work. The FPGA analysis was however more time consuming, than the work described in this chapter.

After all classes are introduced, a more practical section will follow and which will show methods for quick SEU effect prediction on a mapped circuit in the FPGA (section 4.4). The final section (section 4.5) will reveal the unique hardware emulator platform, which executes in hardware an exhaustive testing and implements the prediction method validation using all the shown classifications. Results will follow in a next chapter, chapter 5.

4.1 Fault Classification

Every fault from the set of bit has a specific effect on behavior of the circuit. There is therefore a natural need to classify the fault effect by categories. The fault category is evaluated upon the full statistics of set of input test vectors. The response of the faulty circuit to a single input test vector can be characterized by 2 signals:

1. Validity of the output codeword. A codeword is in the field of CED any vector, which has the check bits part the same as the check bits calculated from the result part of the vector. The codeword validity can be checked by the code **checker** (for example parity checker).

2. Correctness of the output. The correctness of the output vector has to be compared with the dictionary of precalculated results, by comparison to results of another instance ("golden copy") of the circuit, which is driven by same input, or by other method. The golden copy of the circuit is not subjected to the testing and therefore always has to give correct results. The golden copy also contains CED predictor, as the circuit being tested. The circuit, that provides the information about the correctness circuit, is therefore called a **comparator**, because the results are being compared to the correct ones, regardless the method used for obtaining the correct results.

The combination of signals from the checker and the comparator has actually only 3 valid combinations, because the correct result indicated by a comparator implies, that the result is a valid codeword:

1. The result is correct and the codeword is valid.

2. The result is incorrect and the output vector does not belong to a valid codeword.

3. The result is incorrect, but the output vector belongs to a valid codeword.

When results of all input vectors from the test vector set are collected, one of following 4 classes can be assigned to the fault:

Class A: The fault is hidden. These are faults that do not affect the circuit output for any allowed input vector. Faults belonging to this class have no impact to the FS property. A circuit cannot be ST, when this fault occurs.

Class B: The fault is detectable by at least one input vector. Faults from this class do not produce an incorrect codeword (valid code word, but incorrect one) for other input vectors. These faults do not have any impact to the FS and ST properties.

Class C: The fault causes an incorrect codeword of at least one input vector and the fault is not detectable by the code checker by any other input vector. Faults from this class cause undetectable errors. If any fault in the circuit belongs to this class, the circuit is neither FS, nor ST.

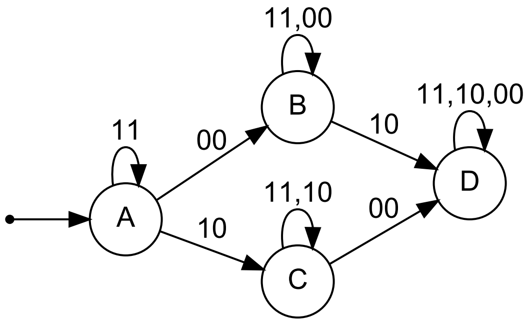


Figure 4.1: Fault class evaluation FSM. Transition edges are coded by 2 bits: `<codeword?,result_ok?>`

Class D: A fault that causes an undetectable error for at least one vector and a detectable error for at least one another vector. Although these faults are detectable, they do not satisfy the FS property and therefore they are also undesirable.

The in-hardware experimental evaluation of the class during testing of the fault by application of test vectors to the input can be described by a Finite State Machine (FSM), which is shown in Fig. 4.1. The procedure of the experimental class evaluation is following: The class is initially assumed to be a class A. A transition is made for every test vector according to the fault response to the output. The response is evaluated by the checker (whether the output vector belong to the codeword) and the comparator (whether the output vector match with the reference circuit). After all test vectors applied, the final state corresponds to the fault class.

The classification of all faults in the circuit can be further used for decision, whether the circuit is fault secure (FS), self-testing (ST) and totally self-checking (TSC), as described in section 2.5. The circuit is fault-secure, when all faults belong to classes A, or B. Circuit is self-testing only when all faults belong to classes B, or D. Totally self-checking circuit is therefore a circuit, whose faults belong all to class B.

This classification also allows to determine the effect of the SEU faults in the FPGA on the behavior and distinguish between *don't care* bits and those bits having an effect on the functionality.

There is another metrics, that can be used for expressing the dependability: by the Not Fail Safe (NFS) metrics and Not Self-Testing (NFS) metrics, which in the means of described ABCD categories can be described as NFS = C + D and NST = C.

How this fault classification is implemented in the hardware emulator is shown at section 4.5.

4.2 FPGA Resource Classes

Unique results of this work are based on detailed information of the FPGA resources being used. Traditional approach of exhaustive bit in-hardware testing (and also irradiation) gives an overall error rate, which does not contain the associated resource error rate statistics.

The FPGA resources were divided into following disjoint classes, which are specified by their location and function:

LUT: It stores the logic function in SRAM memory.

Cell interconnection: These bits control the configuration inside the logic cells. These bits are responsible for the LUT correct inputs selection, feedback in logic cell and correct output selection (registered/non-registered function, 3-input or 4-input LUT organization). These resources are often referred as *local interconnection.*

BUS to Cell/Cell to BUS: These are bidirectional connections, which connect the logic cell to the routing wires (to a bus plane). These resources belong to the *global interconnection.*

BUS crossing: These bits connect the perpendicular crossings of the horizontal and vertical routing wires. These resources belong to the *global interconnection.*

BUS repeater: The repeater separates or connects 4 routing wires by basically every possible combination and direction. These resources also belong to the *global interconnection.*

Unexplored: Other resources, which were not listed above and which have not been explored. Includes Clock, reset, I/Os, block RAM

These resource classes cover the requirements for studying of SEU effect on the combinational circuits. The bitstream contains several bits that do not provide any functionality and do not have a SRAM cell, although it is possible to address those bits as a part of configuration byte, which contains otherwise used bits. Proof of non-existence requires completely different approach, than proof of affiliation to one of the resource class, as shown at Appendix B. Such bits are therefore classified by the *unexplored* class. Some bits were reported by other references as non-existing [31, 47]. Those bits were later in this work re-classified as a "*Fake*" bit class, which is used in the results.

Sequential resources are not within the scope this thesis, which studies the effect of combinational circuits, therefore were not deeply analyzed and classified by the *unexplored* category, except for bits able to change the logic cell behavior from combinational to sequential. These bits and resources were studied and are all included in the *Cell interconnection* class.

Clock and reset resources are essential for sequential circuits – any alternation in the clock tree (the clock source is disconnected, a different clock source used, a change of polarity or a change of frequency) may be fatal. These resources are required only for sequential circuit effect studies, because these resources do not affect combinational circuits at all. If there is a need for a detailed classification of clock&reset resources, the only action required is to analyze the bitstream with focus on those resource, which would require few

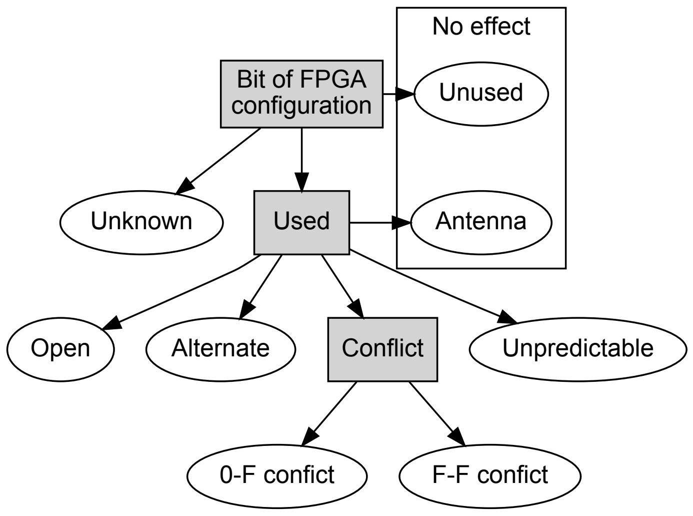


Figure 4.2: Faults classes naming and hierarchy. Grayed boxes include more fault models.

sequential test designs to be analyzed by very similar procedures to those procedures described in Appendix B.

Almost similar is the situation with I/Os, which do not interfere with the design that is completely contained within the FPGA and does not use thembecause it is com

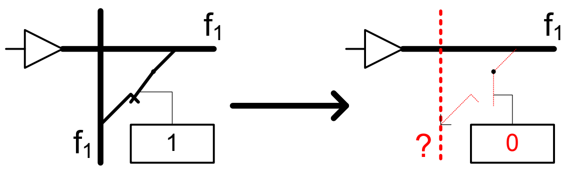
Although resources from the unexplored class are not directly used by the benchmarks, the SEU in those areas may still result to an error or a system failure. Examples of such resources can be an activation of tri-state buffer on the L output of the logic cell, activation of I/O block to drive a global wire, or sudden connection of RAM output to the global wire. Even though not classified in detail, those bits from *unexplored* class are completely tested in the emulator. The expected low contribution to the number of error was confirmed in the results and no system failure was observed (the emulator was still running) .

The time needed to determine the category, to which the bit belongs, corresponds to $O(1)$ time complexity, because the bit is assigned to the resource category regardless to the bitstream.

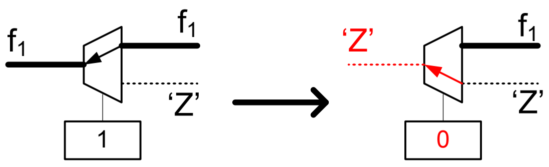
4.3 FPGA Fault Models

This section introduces the FPGA SEU fault models, which can be assigned to each bit of the configuration bitstream. The design has to be already placed & routed for this classification, since the resource utilization plays an essential role in the classification process.

In the 1st approximation, the bit can be marked as *used* or *unused* by a simple rule: Where the observed bit connects/disconnects/drives/joins/modify resources (wires), that are used by the design, the bit is marked as *used*, otherwise it is *unused*. If the decision cannot be made (due to a limited knowledge of the bitstream and FPGA structure), the bit is assigned to an *unknown* fault model class.

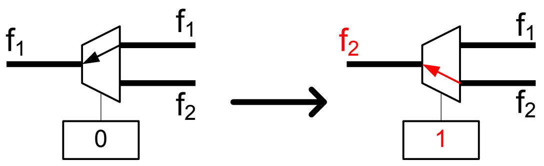


(a) Open fault at the bus crossing

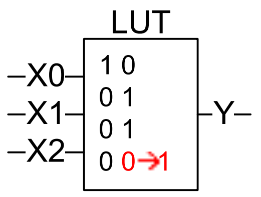


(b) Open fault at 2:1 multiplexer

Figure 4.3: Examples of *open* fault class



(a) Alternate fault at 2:1 multiplexer



(b) Alternate fault at LUT

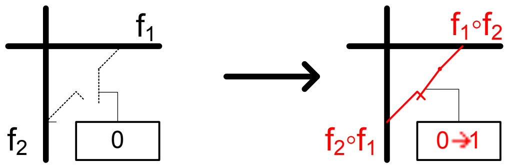
Figure 4.4: Examples of *alternate* fault class

Fig. 4.2 shows the detailed fault classification of bitstream bit, including detailed sub-categories of the used bits, which will be described below. The hierarchy in Fig. 4.2 also shows, which bits have no (static) functional effect on the design. Those classes are unused and antenna fault models. The static functional effect means, that it does not affect the function of the design, but the timing parameter of the design. Consequences are discussed in the antenna category description below.

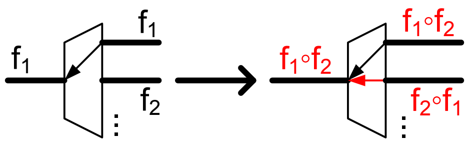
Open: The data-path has to be broken without a substitution of a different signal. This fault is equivalent to a broken wire in the non-integrated circuit. As a result, the signal net is not driven anymore and the destination of the signal receives high impedance "Z" value or "1" if the restoration buffer is present further on the signal path. There are more different origins of this fault from different resources. Typical examples are shown at Figs. 4.3a and 4.3b.

Alternate: The data path is modified as a result of this fault, without creating any *conflict* or *open* faults (although the data-path is broken, the input isn't left undriven, when compared to the *open* fault). There are only few possibilities of alternation. Two of them are shown at Figs. 4.4a and 4.4b. The FPGA under study does allow an alternation in the repeater, nor global interconnect, since at least 2 bits need to be altered there. Other FPGA may offer this fault in the global interconnect, especially when more resources are connected by a single bit. The alternated bit in LUT has to be used; otherwise is this bit classified as *unused* instead. More details are in section 4.4.2.

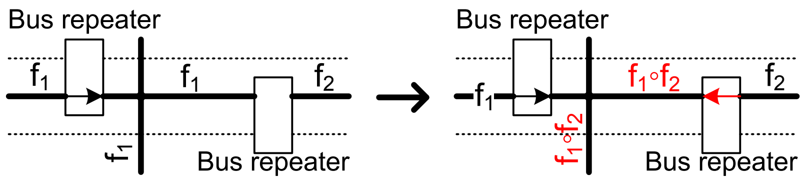
Conflict: This is a special category defined by connecting two or more driven wires. This conflict may lead to a weak short circuit between a power supply and a ground



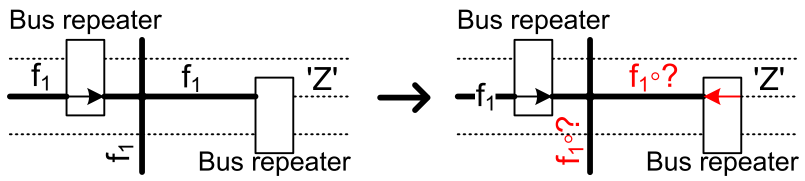
(a) Conflict at the bus crossing



(b) Conflict at the multiplexer

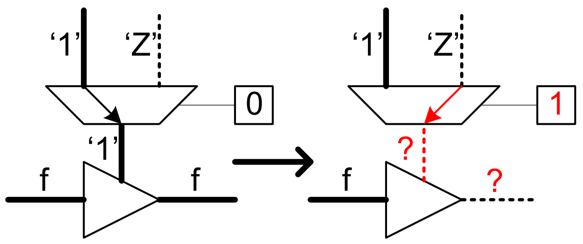


(c) Conflict at the bus repeater

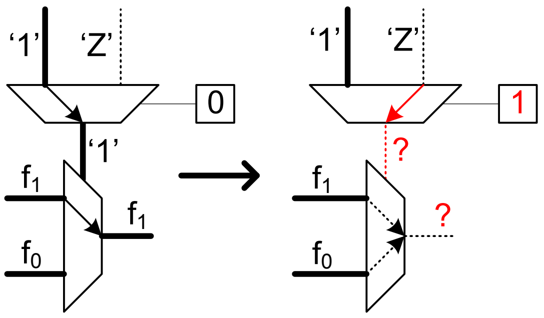


(d) Conflict at the bus repeater with undriven net

Figure 4.5: Examples of *conflict* fault class



(a) Unpredictable fault at output enable buffer



(b) Unpredictable fault at mux selecting a mux

Figure 4.6: Examples of *Unpredictable* fault class

through the drivers. A different strength of drivers can determine the driver to be a dominant driver and a recessive driver. The result is hard to predict, unless a detailed FPGA layout is known. A simple conflict can occur on BUS crossings, when both horizontal and vertical wires are driven (Fig. 4.5a). The "$\circ$" operator in Figures 4.5 should be interpreted as dominance: An operand capable of delivering more current will define the result of the "$\circ$" operator. Conflicts can also occur in multiplexers when selecting more than 1 input, as shown in Fig. 4.5b. Conflicts at multiplexers are possible only when the selector encoding allows multiple inputs to be selected. Such a fault is possible in FPSLIC. Another possibility of the signal conflict is in the bus network. During a normal operation, only one repeater can drive the wire. In the altered state, a conflict on a common bus wire can occur (as shown in Fig. 4.5c), with the same current strength of the drivers. The conflict can be separated into 3 subcategories:

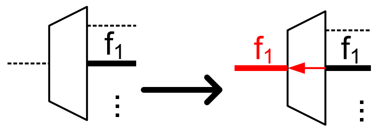
"F-F" conflict is between two non-constant functions. More detailed conflict subcategories list could have been obtained if FPGA technology is known, especially the strength of drivers. Otherwise we have no sufficient information for creating more conflict categories, which would be helpful in fault influence decision.

"0-F" conflict is a conflict, where any function conflicts with constant "0". This category may look redundantly, because it is a special case of "F-F" category. The reason for including this subcategory is a simple fact that in FPSLIC this conflict is frequent, because constant "0" is the default value of unused LUT output.

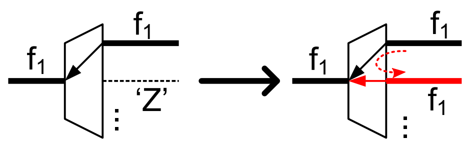
"1-F" conflict is a conflict, where any function conflicts with constant "1". Such a conflict could be considered a situation shown in Fig. 4.5d, but this situation was rather classified as unpredictable. It is highly probable, that the logic value of an undriven routing net is a weak one ("H") due to usage of level restoration. The driver of the repeater will than strengthen the "weak one" to "strong one" and so the "1-F conflict" can be caused.

Unpredictable: There are special cases, where the undriven net is forced to the control input of the resources, or other faults, that can be neither predicted, nor assigned to any other category. Two cases in FPSLIC architecture are presented at Figs. 4.6a and 4.6b. Depending on the level restoration circuit presence, these faults may have no effect on the circuit at all. These unpredictable faults were separated from other faults only because of unknown physical layout of the FPGA.

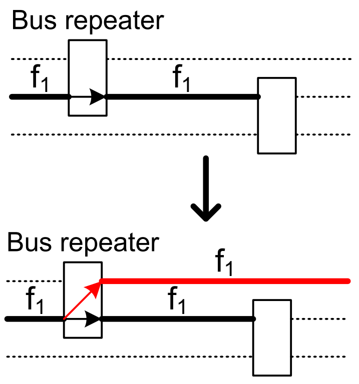
Antenna: It is a bit, where an unused wire is connected to the data-path. This fault has no static influence on the design function, since no *conflict*, *alternate* or *open* occurs. The only consequence will be increased delays, because an extra load capacity is appended, or an extra level restoration buffer (if used) is connected, that has to be overpowered during the $1 \rightarrow 0$ transition. This fault can influence the design only if the antenna is appended to the time critical path. A typical example of the antenna



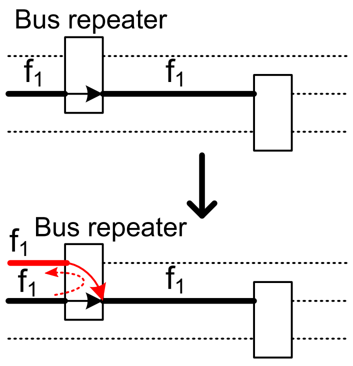
(a) Antenna at the bus crossing



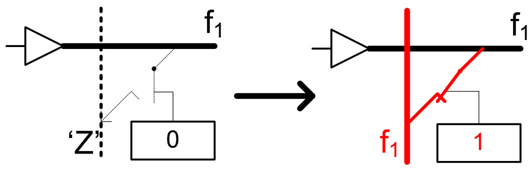
(b) Antenna fault at mux input



(c) Antenna fault at the bus repeater output



(d) Antenna fault at the bus repeater input



(e) Antenna at the bus crossing

Figure 4.7: Examples of *antenna* fault class

fault is shown in Fig. 4.7e, where the vertical bus is connected to the horizontal through the turned-on switch (transfer gate). A lot of multiplexers in FPSLIC have more than 2 inputs. All multiplexers select each input by a separate bit. This technically allows more than 1 input to be selected, as shown in Fig. 4.7b. The unused multiplexer can be driven by some signal. If the output of the multiplexer is not used, no *conflict* will rise and the antenna fault is born, as shown in Fig. 4.7a. The output wire of the multiplexor was not driven nor read before the fault. Selecting the input will not harm the design, it append an unused wire to the data-path. Similar situation can happen in a repeater. Another unused input of the repeater can be attached to the used repeater input, which leads in propagation of that signal to the other input, as shown in Fig. 4.7d. The repeater can also accidentally drive another output, as shown in Fig. 4.7c, not creating any trouble. Better understanding of transistors involved in the repeater antenna fault can be gained from the Fig. B.12.

Unknown: Special category of bits that cannot be assigned to any fault class, because the bit function is not known.

These fault models are suitable for the FPGAs, which control the resources by direct encoding (each bit directly controls a programmable switch or specify a LUT value), and which do not decode more than one connection in FPGA from one bit.

The time needed to determine the category, to which a single bit belongs in, surprisingly corresponds also to $O(1)$ time complexity, at least in AT40 FPGA. Although the computation time does not depend on the FPGA size, the time constant, however, strongly depends on the FPGA architecture and analyzed bit location. The problem of complete bitstream classification has $O(n)$ time complexity - it scales linearly with the size of FPGA.

4.4 How to Predict Bitstream Error Effect

Requirements for predicting the SEU effect are following:

- The prediction calculation should be fast.
- The prediction should not require the detailed model of the implemented circuit.
- The prediction must select all bits that can cause an error. No erroneous bit is allowed to escape the accusation of being errorneous by the prediction.
- The prediction can be pessimistic: Harmless bits are allowed to be predicted erroneous, but erroneous bits must be all predicted.
- False prediction (declaration of harmless bit as a erroneous) should be kept in reasonable levels.

The prediction should not substitute the simulation, or the formal methods of fault effect determination. Those methods which will give the exact answer, but may require more computation time – more, than hardware emulation, which give even more competent result.

The prediction gives an answer, to which class of fault model (as defined in section 4.3) any bit from FPGA configuration belongs. The further decision, by which fault class (A, B, C or D, as defined in section 4.1) will the fault manifest itself, is not predicted in this work, since an exact method of hardware testing is implemented for the fault class decision (A, B, C, or D). This method of hardware testing is described in section 4.5.

The fault prediction, however, is able to indicate those bits, that don't have any effect on the design and therefore must belong to fault class A. Bits classified as those having an effect, can finally represents fault from any class (A,B,C or D). This statistics is described in results (section 4.5).

Both fault simulation and prediction have in common one requirement: The FPGA structure has to be precisely known, as well as the association of bitstream to the FPGA resources (bitstream knowledge). This basically limits any research on the bitstream effect, since the FPGA structure is a guarded secret of the FPGA vendors.

The structure of FPGA has to be known up to the transistor level, transistor sizes, and buffer location. The knowledge precision of the structure has to be even deeper, than would require a simulation of the FPGA works without errors. The example of the buffers presence impact on the fault propagation is shown in Fig. 4.8.

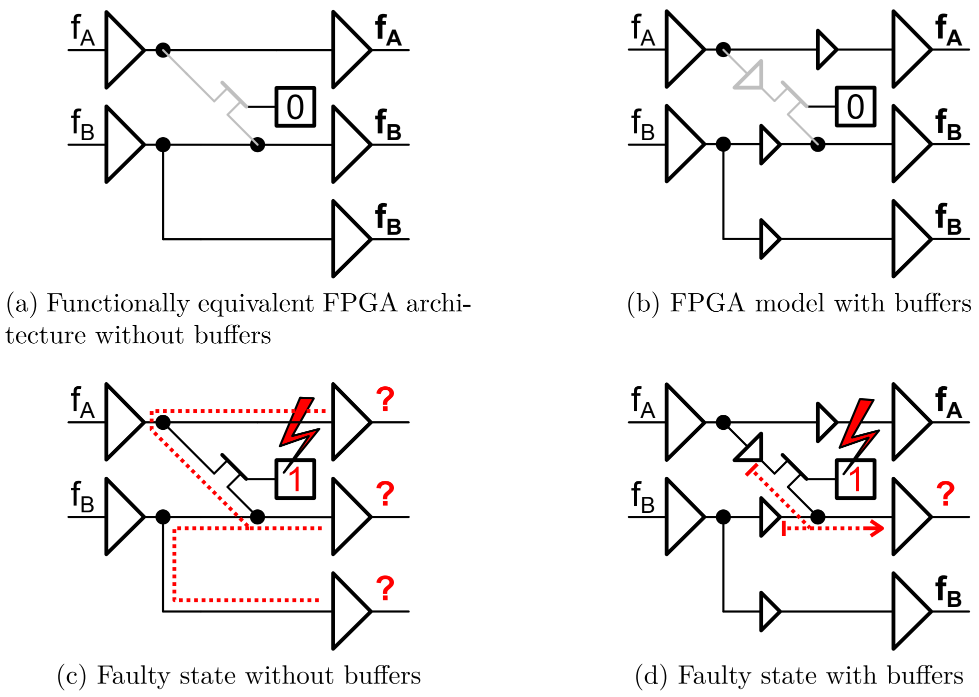
The prediction is based on the resource usage information. Three implementations are possible, although the algorithms for prediction are very similar. Method 1 is not available outside the place&route program. Methods 2 (partially) and 3 (fully) were successfully tested in this thesis:

1. Prediction in EDA after (or during) the place & route process. A structure storing information about the resource utilization has to be in the memory in some form. The prediction can be then made directly by observing the analyzed bit and the status of the resources connected/controlled by the bit.

2. Compute the resource utilization in advance in a dedicated memory structure containing a complete FPGA. In principle, a similar resource utilization structure, as in the 1st method, but it has to be recreated from the bitstream.

3. Extraction of formulas (equations) from the FPGA model, which contain both the resources and the resource connections. Each bit then has its own formula (equation). Since the structure is regular, those equations can be parameterized by the location in the FPGA (by coordinates).

The method 1 is completely out of the scope of this thesis, but the EDA vendors might benefit from the results of this thesis and implement the prediction algorithm inside their tools. Method 2 requires a memory structure, that will hold the information about used resources.



(a) Functionally equivalent FPGA architecture without buffers

(b) FPGA model with buffers

(c) Faulty state without buffers

(d) Faulty state with buffers

Figure 4.8: Bitstream fault back propagation effect, showing different fault propagation for unbuffered (*a*, *c*) and buffered (*b*, *d*) FPGA model. Simplified drawing representing the routing buses, that can be connected by a pass gate.

4.4.1 Graph Representation of the FPGA Used Resources

The FPGA can be transformed into a semi-directional graph, and the design can be transformed onto the coloring of that graph (both edges and nodes). Semi-directional graph means, that both directed and undirected edges are present in the graph. The graph has a following structure:

- Each node represents a physical wire or a more complex resource (such as LUT or logical gate).
- Each edge represents a programmable connection. If the connection is bidirectional (by the means of signal passing), the edge is undirected. If the connection contains a driver, the edge is directional in the direction of signal propagation.

In principle, edges use a very simple coloring (“`active`”), copied directly from the responsible bit in most cases. Some cases require a calculation: for example inputs of 2:1 multiplexer (controlled by 1 bit) by the bit negation (the first input) and by the bit itself (the other bit).

The coloring of nodes is possible by many keys, but “`is_driven`” was found to be easily implementable, though a different approach, “`is_read`” might also be a (non verified) possibility. The `is_driven` color is related to the existence of a *function* (a LUT product), by which the node is driven. There are special colors for **`is_driven_by_0`** and **`is_-driven_by_1`**, which informs about the constant strong 0 or constant strong 1 driving the node. This coloring method could be easily enhanced with the `is_restored` coloring, which indicates, that the level is being restored on the particular node (wire). This would lead to weak 1 on the wire, which could lead to the strong 1, when path through the buffer is established.

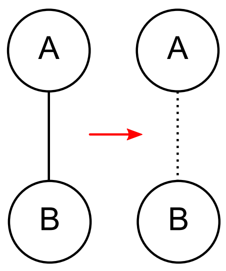
The process of a full graph coloring is following: Each LUT, that is used (how this is evaluated is shown later), is marked as `is_driven`. The graph is then traversed and `is_driven` color is spread through the `active` edges in both directions (if the edge is undirected) or in one way only (in the way of arrow, if the edge is directed). As a result, a colored graph that allows the fault classification is available.

In practice, the directional edges with shared buffer have to be grouped together (by a color or an attribute) for proper fault back-propagation evaluation and distinguishing between *antenna* and *conflict* (*unpredictable* resp.). That is actually quite a common configuration, where the multiplexer is followed by the buffer, which is enabled when any input is selected. Depending on the implementation, the fault may or may not propagate to the other input

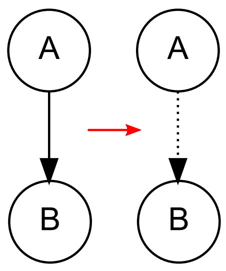
The fault category can be evaluated from the graph coloring, in particular from the edge (bit affected by SEU) and neighboring nodes (typically wires):

- *Open*: When 2 used nodes with an active edge are disconnected by that edge (shown at Figs. 4.9a and 4.9b).
- *Alternate*: When a used LUT bit (as defined by Table 4.1) is modified, or another edge from a used node is activated instead of the original (shown in Fig. 4.9c), but without a change of the signal direction through the edge.
- *Conflict*: When an inactive edge between 2 used nodes is activated (shown at Figs. 4.9d and 4.9e).
- *Unpredictable*: Another edge from unused node is activated instead of the original one, but the new node is undriven. Example is shown in Fig. 4.9f. Unlike *open* fault, the edge is activated instead of deactivation (used in *open* fault).
- *Antenna*: an undriven node is attached to the driven node (shown at Figs. 4.9g and 4.9h), or another node is attached to the input using an edge, that share the same driver as another active edge, as shown in Fig. 4.9i. For easier understanding, two edges sharing the same driver can be redrawn as two undirected edges and one additional edge, whose coloring is based on the colors of the undirected edges, as shown in Fig. 4.10.

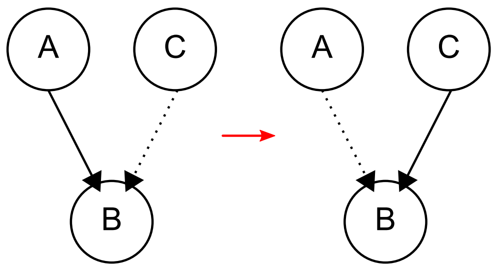
When the graph is created and colored, a fault prediction decision diagram can be made. Simplified diagram is shown in Fig. 4.11. It uses an assumption, that an active edge always connects used nodes (both are colored by `is_driven`). This decrease the



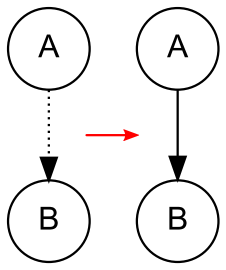
(a) Open



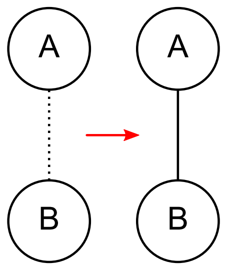
(b) Open



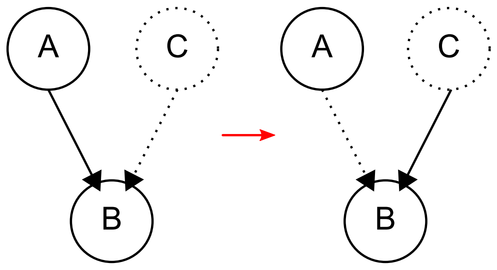
(c) Alternate



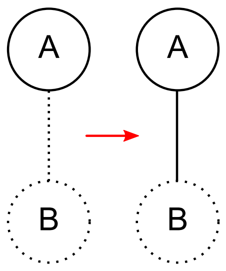
(d) conflict



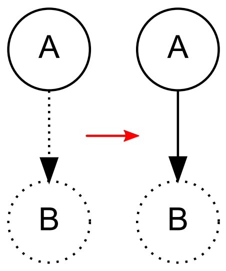
(e) conflict



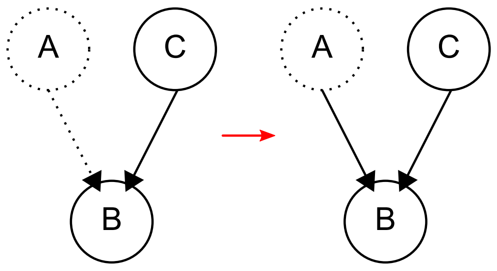
(f) unpredictable



(g) antenna



(h) antenna



(i) antenna (edges share same driver)

Figure 4.9: Graph representation of the fault models. Nodes and edges colored by is_driven and active colors are drawn by full lines. Unused nodes and edges are dashed

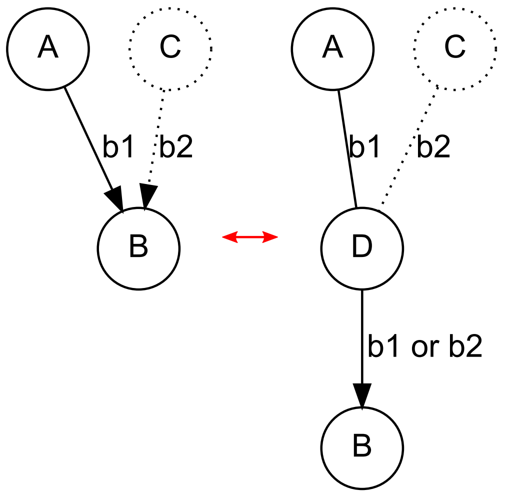


Figure 4.10: Antenna fault with 2 input directed edges transformation, where a buffer is shared between inputs

number of decisions in the active edge branch. Second simplification is a missing LUT bit decision branch, which is described in a separate chapter (chapter 4.4.2).

The “Another edge?” decision diamond in Fig. 4.11 means a condition, whether there exists another edge, that is activated while the original edge is deactivated due to the same SEU. In FPSLIC there is only one such situation and how such edge is oriented with respect to the 3rd node (node C), as shown in Fig. 4.9c. Nodes A and B refer always to the source (A) and destination (B) of the analyzed edge (analyzed bit).

4.4.2 LUT Partial Usage

The LUT memory is not always fully used. The usage of LUT can be defined either by:

- number of connected inputs, or
- list of possible input vector combinations.

The number of connected inputs can be quickly calculated, since bits connecting the input to the LUT are known and if anyone is active, the LUT input is used (F is asserted to the LUT input in table 4.1). When an input of the LUT is unconnected, it is driven high by default. The usage map of the LUT can be extracted from that information, as shown in Table 4.1.

The LUT output, however, cannot be unconnected at the output – it always drives the output by a value. The default value (when no input is used) is “0”. Technically it is meaningless to route a constant 0 to another LUT, but there is a special case (though never seen in use by any design), where both LUTs are unconnected (not driven by any input) and only the multiplexor selecting between those 2 LUTs is active. The 1st LUT would be filled with 0s, the other with 1s, effectively generating a buffer or inverter by the ZM signal at the D net (logic cell connection details shown in Fig. A.4).

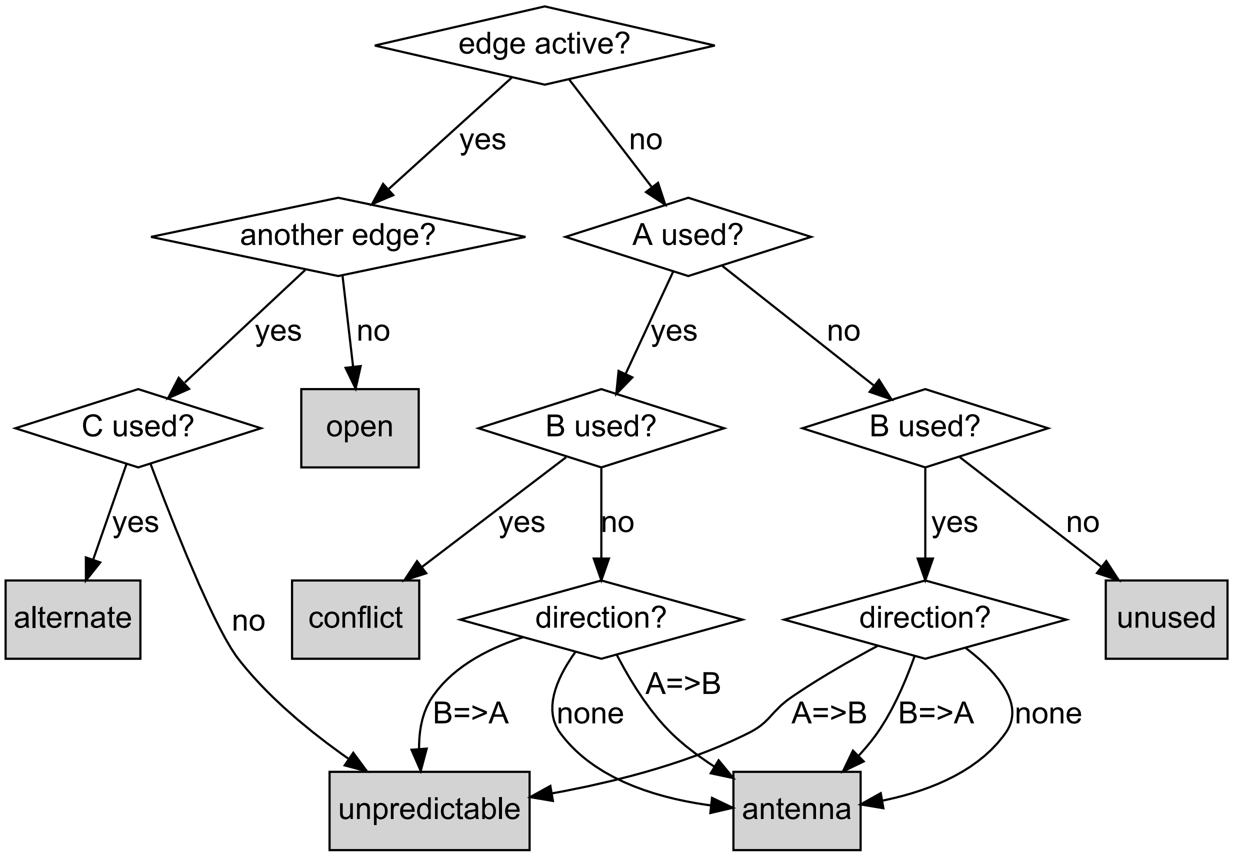


Figure 4.11: Fault prediction decision diagram

	LUT input 0	1[a]	F	F	F	1[a]	1[a]	F	1[a]
	LUT input 1	1[a]	1[a]	F	F	F	1[a]	1[a]	F
	LUT input 2	1[a]	1[a]	1[a]	F	F	F	F	1[a]
input combination	000	-	-	-	u	-	-	-	-
	001	-	-	-	u	u	-	-	-
	010	-	-	-	u	-	-	u	-
	011	-	-	-	u	u	u	u	-
	100	-	-	u	u	-	-	-	-
	101	-	-	u	u	u	-	-	u
	110	-	u	u	u	-	-	u	-
	111	-[b]	u	u	u	u	u	u	u

[a]When no input is selected, "1" is asserted
[b]Almost always unused, but some evil constructions might use even this bit

Table 4.1: LUT partial usage table. "u" states for the used bit, "-" states for the unused bit and "F" at the LUT input means, that some function is routed to the input. LUT inputs names do not match the real input name, since a different permutation of inputs for each LUT (LUTX or LUTY) is given

The second issue (will be also shown in results in the section 5) is that usually some input vector are not possible at the input. LUT bit for those combinations will be technically unused, since they are unreachable by the LUT inputs combination. Addressing those bits would require deeper analysis of the complete implemented design, which is out of the scope of this work.

4.4.3 Prediction Formulas Extraction

This section will focus on the equations extraction from the graph presented in previous section. An uncolored graph from the previous section is used and few tricks to limit the necessary bit set required for the fault class prediction are used.

Let's focus on the inputs and outputs of the formulas. The formula can in principle have access to the complete graph representation and complete bitstream, but that would not be a very efficient formula. The goal is to find a smallest graph and a minimum set of bits, which is sufficient for a fault class prediction.

The primary information source is:

- status of the edge, controlled by the bit
- if needed, status of another edges that are controlled by the analyzed bit
- Usage information about the adjacent nodes (wires)

The nodes color information is not available here, because the colored graph is not constructed. Since the information of node status is essential, these states have to be acquired.

This can be done by the extraction of bit conditions prior to the analysis. The extraction simply picks all direct neighbor edges to the node by the graph traversal for depth 1, but ignoring the edge directions. The depth 1 is sufficient for all FPGA structures of FPSLIC, since it is assumed, that any connected node is automatically driven somewhere, therefore can be colored by the `is_driven` color.

An example of condition extraction is shown in Fig. 4.12. The extracted edge is shown in the middle. Nodes with all adjacent edges and nodes to those nodes are also shown. This method benefits from the bitstream implementation, where almost all bits drive only one programmable connection. The partial result of this extraction is a formula for the V4[4,7] node: OR (b1…b15); and for the L4[4,7] node: OR(b17…b22).

The final formula of the bit effect is a formula implementing the passage through the decision diagram presented in previous section and shown in Fig. 4.11, which uses coloring (`is_driven`) evaluated from the extracted partial formula.

This extraction method for prediction formula was implemented for all bits of the bitstream and successfully tested. Some bits (tri-state buffers) were added manually and some bits inside the logic cell required to be written manually, since some inputs of 2:1 multiplexers are being actively selected, even though the signal is not routed through that multiplexer for any purpose.

4.5 How to Emulate the SEU in FPGA

The emulation of SEU can be performed by a simple change of one bit in the configuration memory. Preparation of such modified bitstream may require also CRC checksums recalculation in order to be accepted by the FPGA. If the FPGA supports partial reconfiguration, only fraction of the bitstream needs to be modified and the time needed for FPGA reconfiguration is also fractional to the complete configuration.

Emulator is a hardware platform that implements the manipulation of the configuration memory on a physically implemented design and evaluates the effect of the injected fault on the design functionality. The bitstream is manipulated by the same way, as the SEU would - by a single bit flip, one at the time.

There are several reasons for building an emulator and for testing circuits in there: 1) emulator can be much faster than simulation; 2) Results obtained from emulation are more trustworthy; 3) Understanding of the FPGA and bitstream is not necessary for obtaining the results. The FPGA and bitstream knowledge only help in obtaining more detailed statistics, but the “OK/fail” results are absolutely sufficient for obtaining of number of sensitive bits and further reliability calculations.

4.5.1 The Emulator

The emulator presented in this thesis was built completely on top of single FPSLIC SoC device. The structure of emulator and signal paths are outlined in Fig. 4.13. Functional part of this emulator is split into following 4 sections:

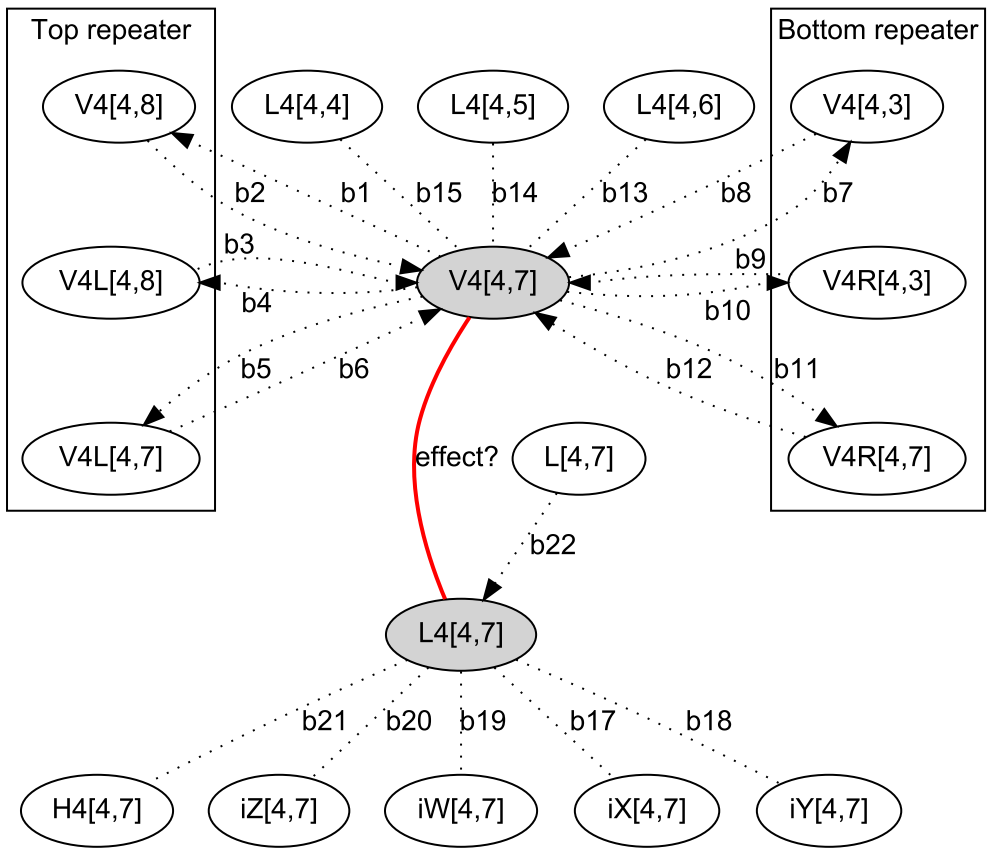


Figure 4.12: Example of prediction formula extraction from two wires (V4[4,7] and L4[4,7]) and programmable connection (marked red and labeled "effect?")

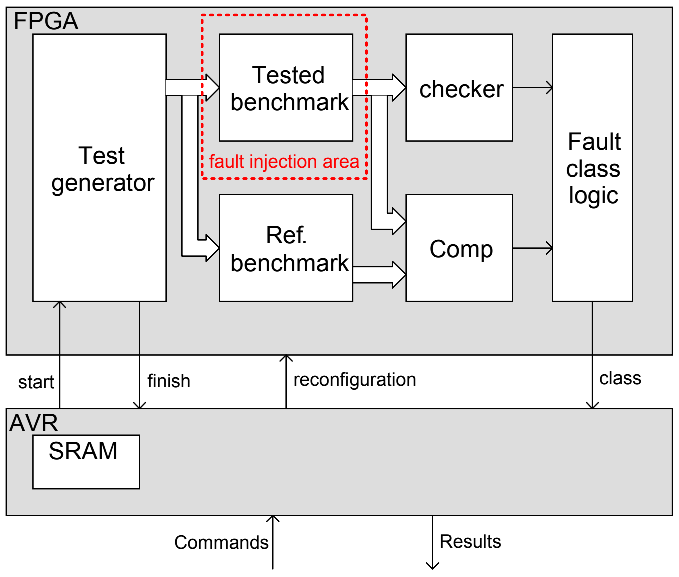


Figure 4.13: The architecture of SEU fault emulator implemented in the FPSLIC device

- FPGA part of the FPSLIC: Test vectors are generated there. Vectors are further split to drive both copies of tested circuit (1st is for the testing, 2nd for the reference). The 1st copy of the tested circuit is placed to the fault injection zone of the FPGA, while the 2nd is kept untouched, out of the fault injection zone. Response of the 1st test circuit, subjected to fault injection, is checked (if the test circuit supports error detection coding) and compared with the 2nd copy of the tested circuit. Checking and comparison is performed for each test vector and results are summarized by the fault class evaluation logic, which finally assign the fault effect class (as defined in section 4.1) to the bit.

- The embedded AVR microcontroller in the FPSLIC performs the fault injections and testing. The fault injection is done by the dynamic reconfiguration, which rewrite only 1 byte (byte is the smallest addressing granularity capability of FPSLIC) of the configuration memory with the modified byte with 1 bit flipped. The testing procedure itself consist of resetting the fault evaluation logic, initiating test pattern generation, waiting for the test vector generation to end and reading out the class from the FPGA.

- Control PC with control SW, which sends to the microcontroller addresses and configuration bytes to test with testing mask. The bitstream addresses and testing masks are computed from the bitstream by the PC, where also unused bits can be excluded from the testing. Excluding unused bits from testing saves time of testing, but all unused bits were in fact tested anyway to validate, that the selection algorithm provides a correct prediction.

- Physical board with FPSLIC device and serial port transceiver. Atmel development kit STK594 with AT94K40AL chip (The largest FPSLIC device) was used without any special modification.

Two different methods of test pattern generators were implemented: an exhaustive test by all possible vectors to the benchmark, which has n inputs, which is implemented by a simple counter. This test generator restricts the usage of the emulator only for benchmarks, having $<\sim 20$ bits. An advanced test pattern generator based on LFSR and hybrid CA was also implemented for use with benchmark with higher number of inputs. The precision of the results was the primary concern in this work; therefore the full exhaustive was always performed.

The FPGA part design tool flow is shown in Fig. 4.14. Tested circuits were picked from the MCNC benchmark library [77]. The benchmark circuit is than split. One branch is minimized, converted to multi-level network format and single parity predictor is generated, the second branch is minimized. Both the benchmark and parity predictor are combined. The benchmark processes are drawn by white boxes in Fig. 4.14

The benchmark VHDL code is then split into 2 copies. A macro (placed, but unrouted design in restricted area) is created from the benchmark copy, which will be subjected to testing. The code of the other copy of benchmark (the reference) is embedded into the

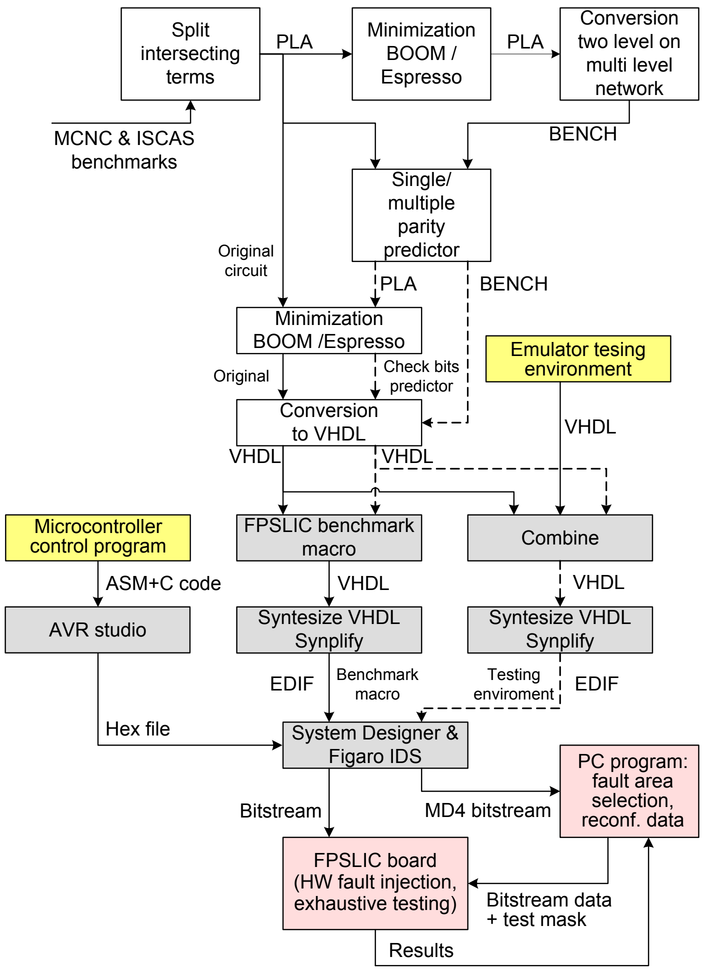

Split intersecting terms
PLA
Minimization BOOM / Espresso
PLA
Conversion two level on multi level network
MCNC & ISCAS benchmarks
BENCH
Single/ multiple parity predictor
Original circuit
PLA
BENCH
Minimization BOOM /Espresso
Emulator tesing environment
Original
Check bits predictor
Conversion to VHDL
VHDL
VHDL
VHDL
Microcontroller control program
FPSLIC benchmark macro
Combine
ASM+C code
VHDL
VHDL
AVR studio
Syntesize VHDL Synplify
Syntesize VHDL Synplify
EDIF
Benchmark macro
Testing enviroment
EDIF
Hex file
System Designer & Figaro IDS
PC program: fault area selection, reconf. data
Bitstream
MD4 bitstream
FPSLIC board (HW fault injection, exhaustive testing)
Bitstream data + test mask
Results

Figure 4.14: Tool flow of the hardware emulator, displaying chain of used tools

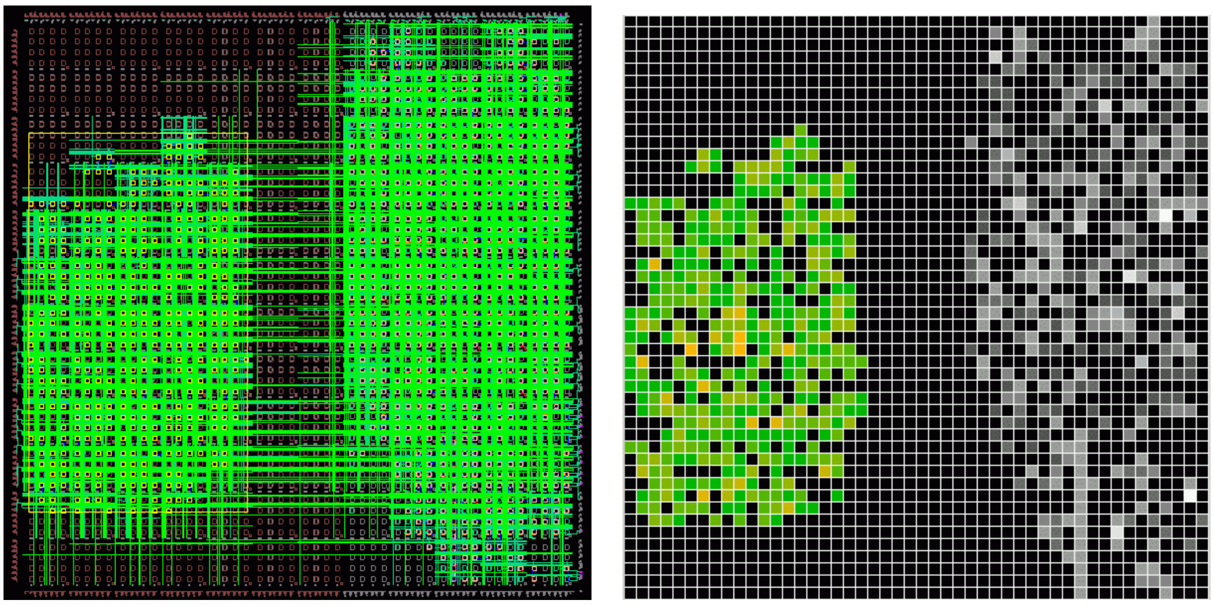

(a) EDA floorplan view (b) Predictor view of fault injection areas

Figure 4.15: Emulator floorplan and fault injection zone

emulator testing environment. The emulator environment is parameterized by the size of input vector, size of output vector and number of parity bits. Once these parameters are known, following blocks are generated: checker, comparator and test pattern generator.

The benchmark macro is placed in the left part of the FPGA (shown in Fig. 4.15), while the complete left half is locked from placing the emulator environment (can be seen by brown color of unused resources on Fig. 4.15a). More information about the floorplan follows in the next section. The emulator environment is than placed and complete design is routed. Basically all emulator processing steps are shown in gray boxes in Fig. 4.14. All the processing steps have to be done manually for every benchmark, except for the microcontroller code.

Essential emulator code, that the emulator is based on, is shown in Fig. 4.14, while the hardware, on which is the emulator operated, is described by the pink color. For the full operation, the emulator requires the FPSLIC board, and a control PC, which communicates with the FPSLIC board via the RS-232 interface.

4.5.2 Emulator Floorplan and Fault Injection Area

The emulator was implemented completely in a single FPGA, which brings a problem: How to separate the fault injection zone with tested circuit from the rest of the emulator logic implemented in FPGA? Solution used by this emulator is to split the FPGA in 2 parts, as shown in Fig. 4.15.

The complete FPGA splitting into 2 halves is possible under following conditions:

- All logic (except for the testing circuit) has to be outside of the fault injection zone
- All connections paths (except for routes from and to the tested circuit) have to be routed outside of the fault injection zone

Such conditions are feasible only in the presented configuration, where fault injection zone is the complete left half of the FPGA, because the right I/Os of the FPGA are used for communication with the microcontroller. The emulator also restricted placing of logic cells 1 sector column (4 columns of logic cells) from the middle in each direction; otherwise the separation would be not possible. Express bus nets skip every second repeater; therefore such used connections might leak into the other half, which might affect also the emulator control logic and golden copy of the tested circuit.

4.5.3 Software Equipment of the Emulator

The software provides several functionalities:

- Implements the bit fault model prediction and FPGA resource classification of the bitstream
- Provides a possibility of individual area selection, as well as specific FPGA resources and specific fault models selection
- Communicates with the FPSLIC via the serial port
- Sends the bitstream address, content and testing mask to the AVR
- Records the results of measurements (A, B, C or D classification) in the form presented in Appendix C

The software is completely written in Java. The most valuable functionality is the implementation of classification of the fault models, as described in section 4.4.3. The algorithm benefits from the extremely regular structure of the FPGA, therefore many formulas for fault class prediction equations were parameterized by the coordinates. Some modifications were however necessary at the border, where non-existing resources might be references (resources outside the FPGA corner) and different routing at the edge had to be implemented to the formulas.

More details about the emulator and control software can be found in [A.4, A.5, A.9].

Chapter 5

Main Results

The primary results are results of SEU effect emulation of the combinational circuits, which are implemented (i.e. placed&routed) in the FPGA. The conventional fault effect analysis does not count any other resources than LUTs, even though LUTs typically represents only $<\sim$ 15% of the sensitive bits in the FPGA. This value is even more critical for modern FPGAs ($<\sim$ 10%). Moreover, a detailed FPGA resource class and an FPGA fault model are assigned to every bit of the bitstream, which provides an exceptional insight into a mechanism of faults inside the FPGA due to SEU.

The fault model prediction presented in this doctoral thesis provides an effective way of prediction of number of sensitive bits in the placed&routed designs. The prediction of number of sensitive bits is an indispensable and essential source for precise reliability calculation of designs based on FPGAs. The fault prediction itself therefore represents one of the most important results of the work, probably affecting many other studies.

The FPGA bitstream analysis procedure might be considered as an important secondary result, which provides algorithms of FPGA analysis, which is necessary for anyone studying fault effect on any SRAM-based FPGA, which does not provide the bitstream information (most of the cases).

This section will describe results, which were measured in the emulator. There are some important findings in the statistics of used logic cells number, used LUT bits, measured number of sensitive bits in LUTs and in the whole FPGA, that will be presented in section 5.1. More statistics about the resource usage and predicted faults will be given in section 5.2.

Results of the fault categories provided by the means of CED will be evaluated in section 5.3. The CED allows a classification of the fault (A, B, C, or D), from which the satisfaction of FS and ST properties can be evaluated. Information about the measurement time will be given in section 5.5. Chapter 5.4 is dedicated to the uncertainty of the measurement, which was studied as an instability among the measurement results, which were measured many times with the same design.

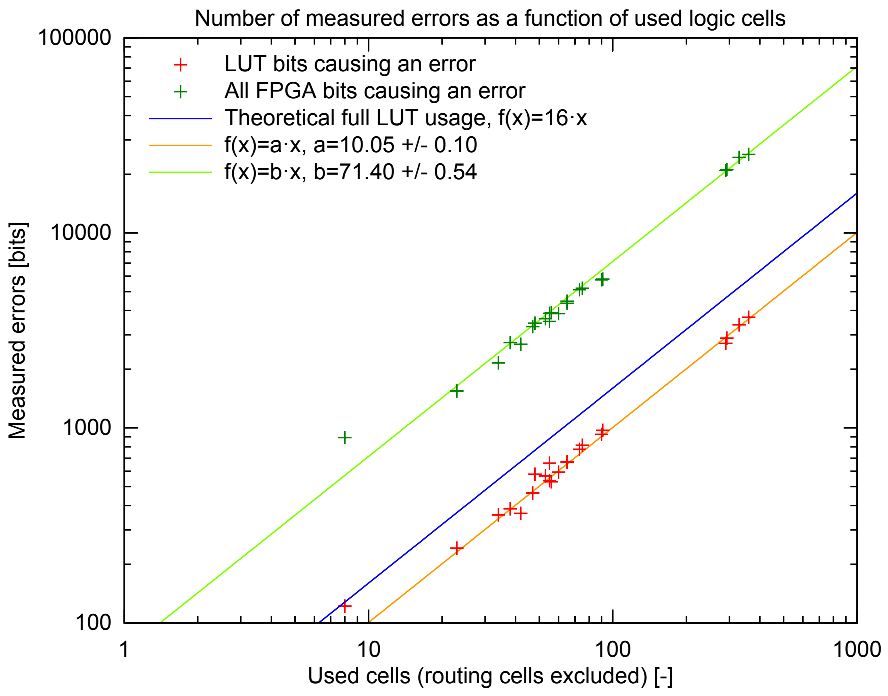


Figure 5.1: Overall and LUT only measured errors as a function of used logic cells

5.1 Global Statistics of Sensitive Bits

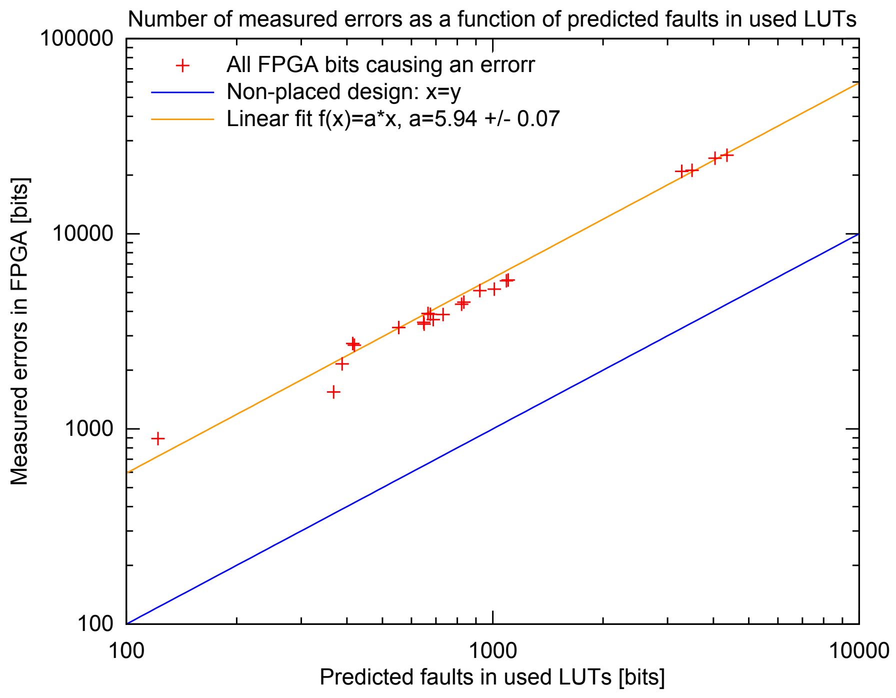
The graph in Fig. 5.1 shows statistics of all tested benchmarks, both with and without the parity predictor. Though some designs contain parity, these results do not reflect it. A raw number of bits, that functionaly modify the whole design (regardless of which part was hit, whether the benchmark or predictor) is displayed as a number of errors. The output vector differs from the expected one during an error.

The X-axis display the number of logic cells, which contained the benchmark macro before it has been placed; therefore no routing cells are included in the number of used cells. The graph shows 5 values on the y-axis:

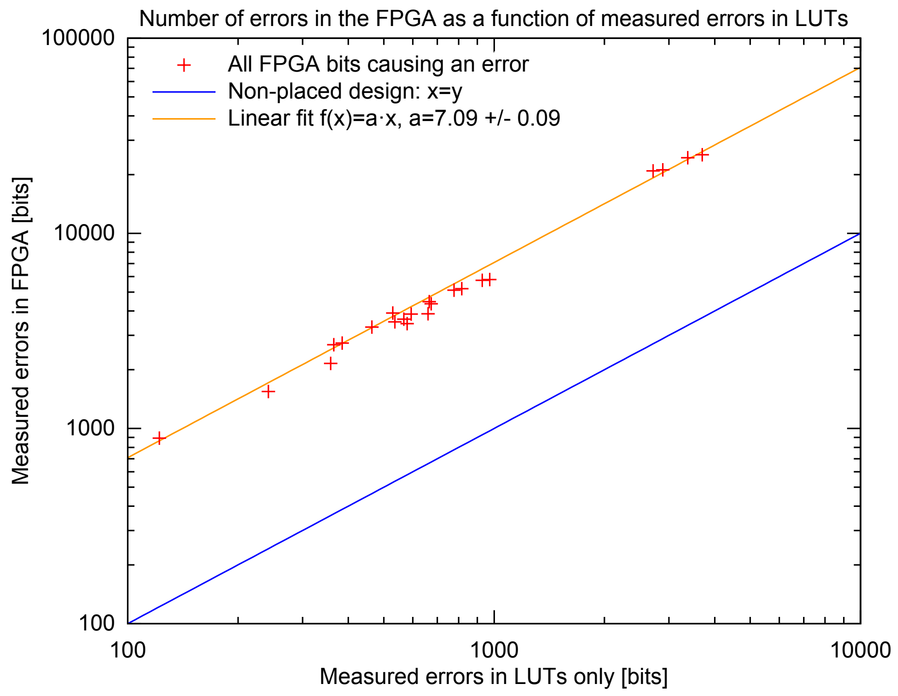
1. Measured numbers of LUT bits, which have functional influence on the design. As expected, the LUT utilization is typically smaller than the theoretical full LUT utilization. Measured numbers are still increased a little over the unrouted design, because errors originated from the routing cells are also included to the number of LUT bit errors.

2. Measured numbers of errors in all FPGA resources as a function of number of used cells. This result includes all classes of fault models and FPGA resources.

3. Theoretical limit for design, which fully uses LUTs (all 4 inputs are used in each LUT) and does not have any routing cells. Such unrealistic design would utilize all 16 bits for each LUT, as shown by the blue line in Fig. 5.1. A nearest benchmark to this limit is ALU1 with 8 used cells.



(a)



(b)

Figure 5.2: Total errors in the FPGA as a function of predicted sensitive (a) and measured (B) LUT bits

[bits]	LUT	Cell interconnection	Bus to cell	Bus crossing	Bus repeater	Fake	Unexplored
Unused	13304	23286	6155	6434	18916	1104	–
Alternate	4360	3417	–	–	–	–	–
Open	–	730	1643	363	1077	–	–
Conflict 0F	–	2228	–	–	–	–	–
Conflict FF	–	5133	817	1181	9403	–	–
Antenna	–	12108	2425	3062	5824	–	–
Unpredictable	–	570	–	–	–	–	–
Unknown	–	–	–	–	–	–	11740

Table 5.1: Total bit counts in s1488 benchmark

4. A linear fit of the number of sensitive LUT bits with regards to the number of used cells. The result of the fit is 10 errors per 1 used logic cell.

5. A linear fit of the total number of sensitive bits with regards to the number of used cells, showing an estimation of 71.4 errors per 1 used cell.

Another useful result is the relation of LUT sensitive bits to the total number of predicted sensitive bits in LUTs (Fig. 5.2a) and to the actual number of measured errors in LUT (Fig. 5.2b). The only difference between these 2 graphs is a different x-axis, where predicted or measured values are used. As expected, the overall error count is much higher, than LUT error count. The factor of all bit errors vs. LUT errors was fitted and drawn onto both graphs, resulting in factor 5.94 for predicted sensitive bits in LUT and 7.09 for measured errors in LUT.

5.2 Resource Utilization and Fault Distribution

Example of The absolute numbers of resource utilization can be found in Table 5.1, which shows the numbers for the s1488 benchmark. The table also shows the prediction of a bit effect. The *unused* and *antenna* classes are those that should not have any influence on the design functionality.

The next Table 5.2 shows the in-hardware testing of the complete bitstream. Based on the knowledge of the previous bitstream analysis, faults are grouped into the respective categories. The absolute number of errors is given together with the percentage of the

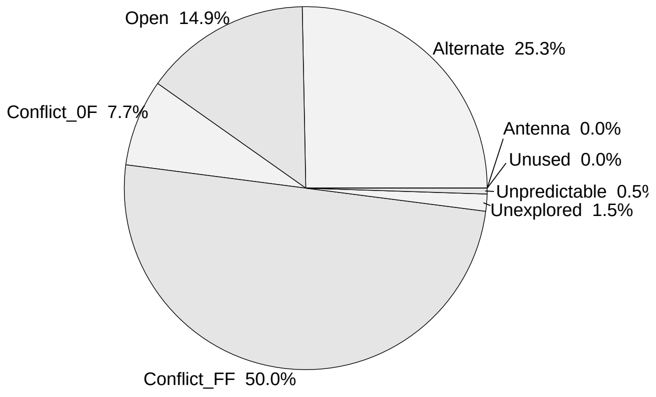


Figure 5.3: Fault model class distribution in the s1488 benchmark

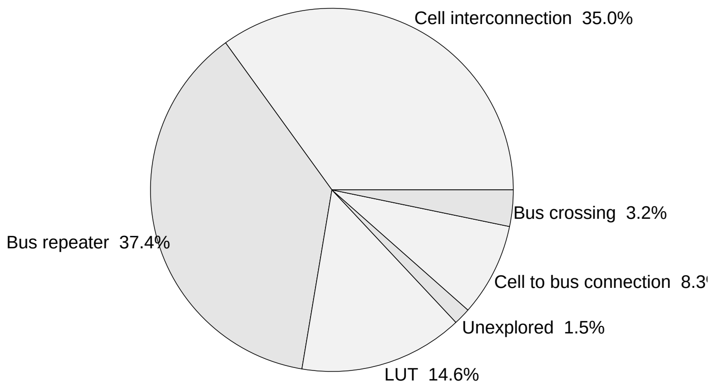


Figure 5.4: Used FPGA resource distribution of the s1488 benchmark

predicted bits shown in table 5.1, that have caused an error. The most important assumption, that *antenna* and *unused* faults never cause an error, has been confirmed during the testing of all benchmarks.

Bitstream usage overview of the s1488 benchmark is given at figures for fault models (5.3) and for the resource distribution (5.4). The resource distribution show, that cell local interconnection and bus repeaters are the major sources of the errors of the design.

The only unexpected result is the *conflict* fault fairly low rate of manifesting itself as an error. The reason could be different driver strength, different FPGA structure, than expected, or an error in the prediction algorithm, which exaggerate the fault effect.

Relative number of sensitive bits to the predicted number of those bits is shown in Table 5.3 for different benchmarks.

5.3 CED Fault Categories Results

Results presented in this section will show the result of the fault classification into 4 categories (A, B, C and D, as defined in section 4.1) for circuits equipped with a parity

Errors [bits] Errors [%]	LUT	Cell interconn.	Bus to cell	Bus crossing	Bus repeater	fake	Unexplored
Unused	0 0 %	0 0 %	0 0 %	0 0 %	0 0 %	0 0 %	–
Alternate	3695 84.8 %	2700 79.0 %	–	–	–	–	–
Open	–	721 98.9 %	1604 97.6 %	363 100 %	1077 100 %	–	–
Conflict 0F	–	1958 87.9 %	–	–	–	–	–
Conflict FF	–	3329 64.9 %	487 59.6 %	456 38.6 %	8376 89.1 %	–	–
Antenna	–	0 0 %	0 0 %	0 0 %	0 0 %	–	–
Unpredictable	–	112 19.6 %	–	–	–	–	–
Unknown	–	–	–	–	–	–	383 3.3 %

Table 5.2: Measured bit effects in s1488 benchmark for fault models (rows) and FPGA resources (columns). The first (top) number indicates an absolute measured count of bits causing an error on the output. The second (bottom) number indicates the prediction effeciency – the ratio of a predicted sensitive bit count and a measured error count

[%]	5xp1	alu1	alu2	alu3	b11	b12	br1	bw	s1488	s1494
used LUT bits	388	676	1102	1090	420	650	822	834	4360	4042
Unused	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
Alternate	90.0	92.3	84.7	84.5	83.6	86.0	81.0	80.5	82.3	81.0
Open	97.6	99.3	98.1	97.8	98.4	98.2	99.6	98.9	98.7	98.9
Conflict 0-F	93.6	94.4	87.2	87.4	89.2	86.1	87.7	88.1	87.9	88.1
Conflict F-F	84.7	86.0	80.6	79.5	79.9	82.2	77.7	81.5	76.5	76.4
Antenna	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
Unpredictable	24.7	19.8	10.9	15.3	25.8	48.9	11.1	20.4	19.6	22.6
Unknown	0.2	0.3	0.6	0.5	0.2	0.3	0.5	0.4	3.3	3.0

Table 5.3: Functional effect of predicted faults in different benchmarks

	A	B	C	D
Cell interc.	38652	7198	245	1377
Bus repeater	25767	6037	174	3242
LUT	13966	3091	86	521
Bus to Cell	8949	1614	59	418
Bus crossing	10221	520	11	288
Unexplored	11357	256	7	120

	A	B	C	D
Unused	69199	0	0	0
Antenna	23419	0	0	0
Conflict FF	3886	8713	242	3693
Alternate	1379	5318	156	924
Open	48	2732	102	931
Conflict 0F	270	1605	67	286
Unpredict.	458	92	8	12
Unexplored	11357	256	7	120

Table 5.4: Fault categories in the 1488 benchmark according to the FPGA resource (left table) and fault model (right table)

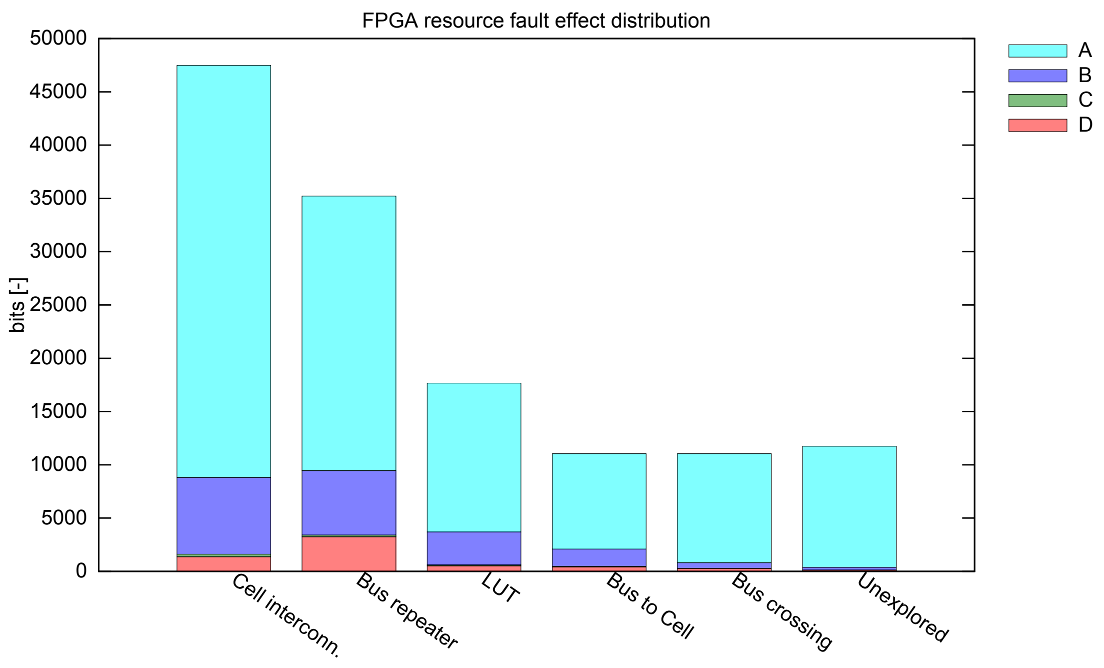
predictor. Complete distribution description requires a lot a space; therefore only 1 circuit (s1488) will be described in detail and then a summary statistics will be shown for all tested circuits. Other detailed results can be plot using the full result set in Appendix C.

The distribution of fault classes according to the FPGA resources model is shown in Fig. 5.5. The s1488 benchmark belongs to the one of the bigger benchmarks that fit into the emulator and occupies ~50% of the testing area, as shown in Fig. 4.15a. The distribution of fault classes according to the FPGA fault models is shown in Fig. 5.6. This figure also shows a clear separation of the A fault class locations, that can be predicted and safely excluded from testing, having no functional effect on the circuit. Same results are also shown in Table 5.4.

Fault class category distributions for more benchmarks are shown in Fig. 5.7, where a relative distribution of the A, B, C and D faults is displayed for benchmarks, that have different sizes. The graph includes all the predicted sensitive bits (100%), excluding the *unknown* class (*unexplored* FPGA resource), which contains a great amount of A faults, which depends on the size of benchmark only faintly. The *unknown* fault model statistics is included in the Table 5.4.

Table 5.5 shows, which benchmarks were tested with and without parity and the absolute number of bits that propagate the error to the output. Several values are indicated in the table:

- Number of inputs of benchmark.
- Number of outputs and number of parity bits of the predictor (parity has always 1 bit).
- Checkmarks, which combinations of benchmarks and parity was tested. Basically all benchmarks were tested, except for alu2 and alu3, which were tested only with the parity.
- Number of logic cells, which is measured before the routing, thus does not contain the routing cells, which are shown in the next results.



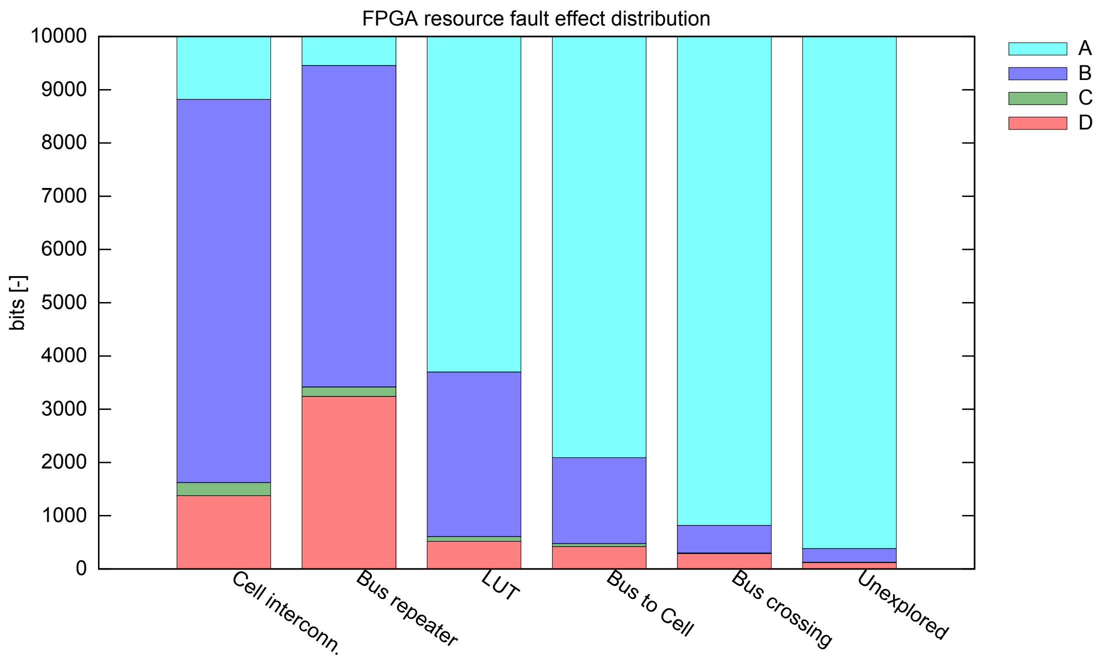
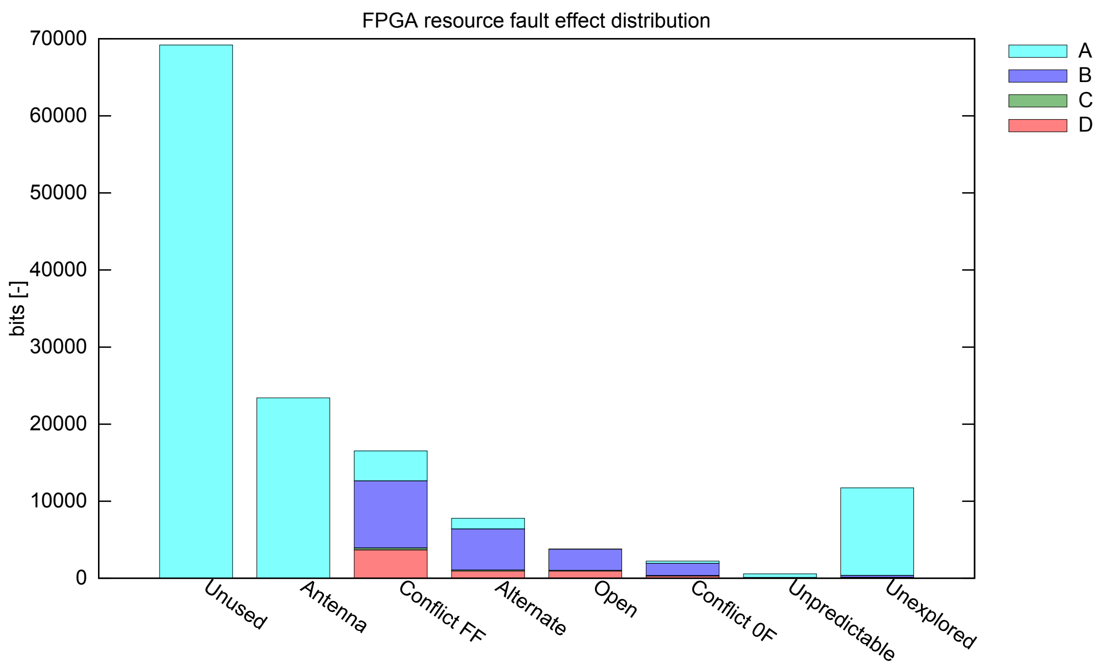


Figure 5.5: s1488 detailed class fault distributions based on the FPGA resource classification. Both figures show the same result. The bottom figure has zoomed y-axis



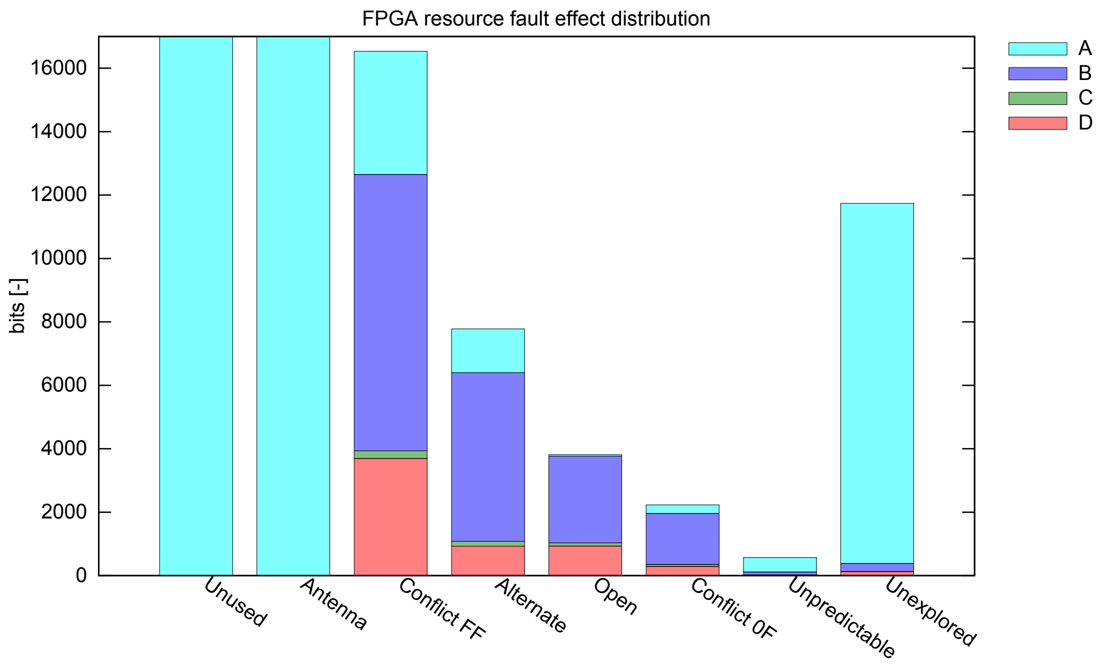


Figure 5.6: s1488 detailed class fault distributions based on the fault model prediction. Both figures show the same result. The bottom figure has zoomed y-axis

benchmark	inputs	outputs + parity	without parity	with parity	Cells w/o parity	Cells with parity	LUT bits w/o parity	LUT bits with parity	errors w/o parity (C)	errors with parity (B+C+D)	may be undetected (NFS=C+D)	always undetected (NST=C)
5xp1	7	10+1	✓	✓	23	34	368	388	1547	2156	548	65
alu1	12	8+1	✓	✓	8	55	122	676	893	3873	420	64
b11	8	31+1	✓	✓	38	42	415	419	2741	2685	933	326
b12	15	9+1	✓	✓	53	48	688	650	3635	3442	520	58
s1494	14	25+1	✓	✓	293	329	3500	4042	21131	24411	6225	529
s386	13	13+1	✓	✓	56	75	666	1010	3901	5203	1565	229
br1	12	8+1	✓	✓	55	65	648	822	3516	4352	1868	388
apla	10	12+1	✓	✓	47	73	554	922	3306	5115	1450	79
bw	5	28+1	✓	✓	60	65	732	834	3857	4466	1278	180
s1488	14	25+1	✓	✓	290	360	3278	4360	20904	25264	6548	582
alu2	10	8+1	–	✓	–	91	–	1102	–	5793	886	41
alu3	10	8+1	–	✓	–	90	–	1090	–	5743	782	73

Table 5.5: Tested benchmarks

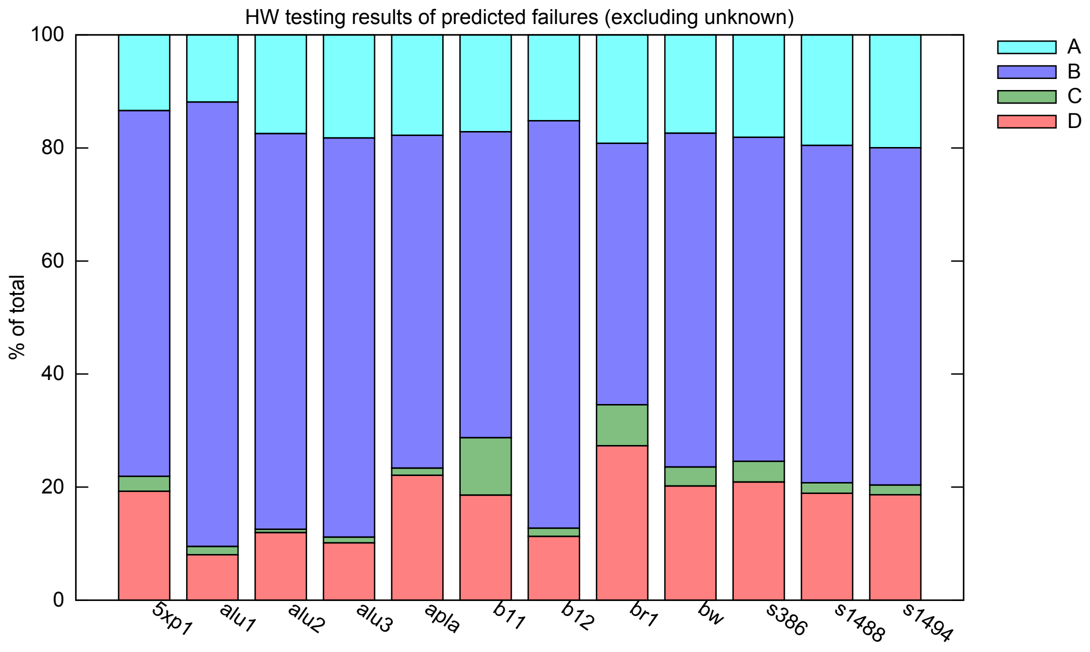


Figure 5.7: Relative distribution of fault classes among different benchmarks. *Unused*, *antenna* and *unknown* fault classes are excluded from this graph

- Number of sensitive bits in all LUTs.
- Total number of error for benchmarks with and without parity. This includes all bits that cause an error. For benchmarks without parity, this is the number of bits in the C class. For benchmarks with parity predictor, B, C and D classes are counted.
- Number of bit errors, that can pass undetected by the parity checker (C and D classes are counted). Whether the error passes depends only on the input vectors.
- Number of bit errors, that always pass undetected by the parity checker, because the output vector is wrong, but always belong to the codeword.

5.4 Measurement Variations

During the measurement some instability in certain resources and faults was observed. Though not having an exact explanation, this behavior was subjected to an intensive testing, which was repeated 130 times on a single.

Assuming, that the distribution of measurement instability has a Gaussian distribution, a fit of those values was also made, however the statistics is too low to make a qualified statement about the distribution. Histograms of propagated errors according to the FPGA resource and fault models are shown in Fig. 5.8 and summarized in Table 5.6. The

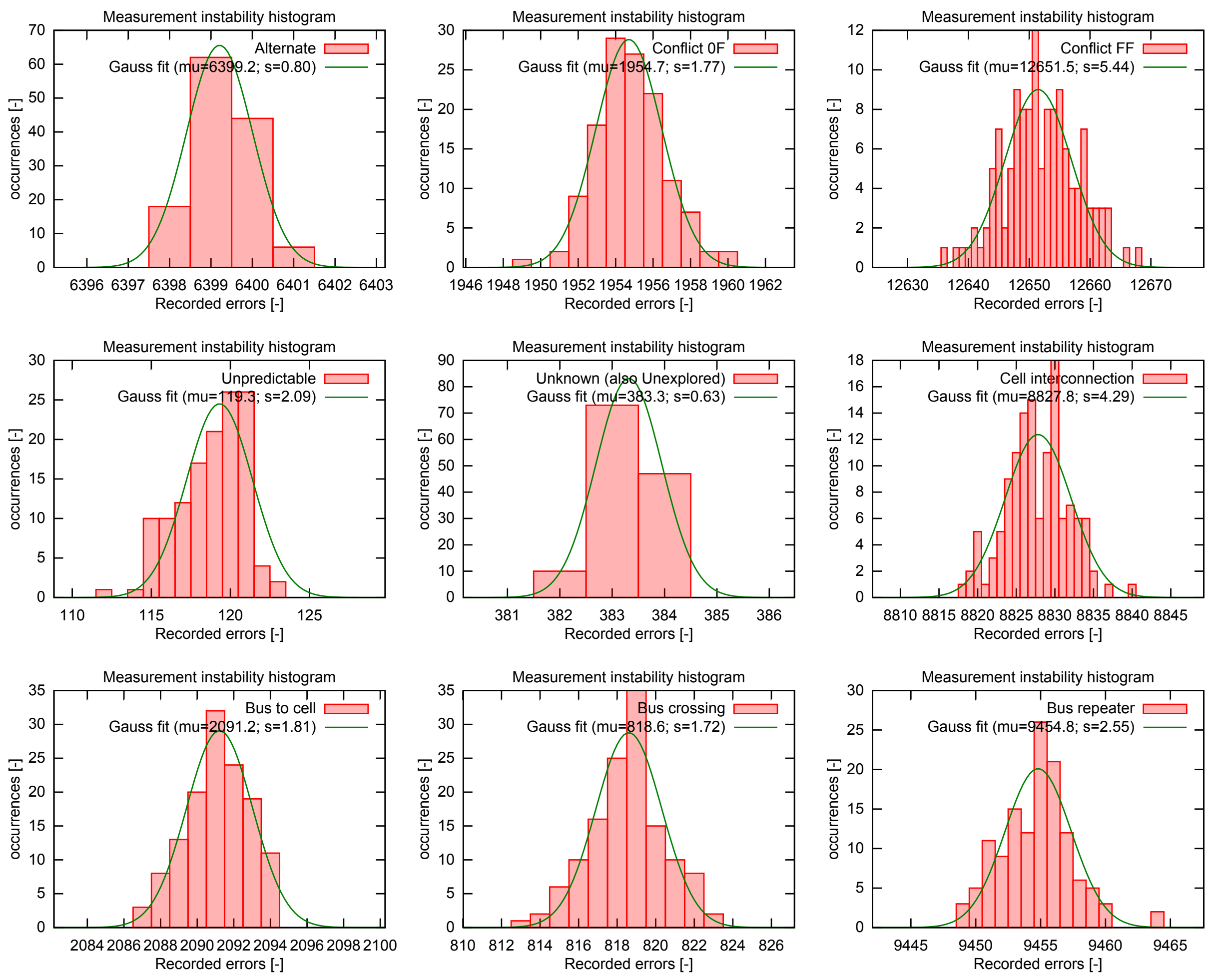


Figure 5.8: Measurement stability according to the FPGA resources and fault models

Conflict F-F fault histograms was rebinned by factor 4 for the Gaussian fit and the *cell interconnection* resource histogram was rebinned by factor 2 for the fit.

A global statics, which include all resources (either all faults or all FPGA resources), is shown in Fig. 5.9. The data for fitting were also rebinned by factor 4 in order to get a better fit, however a perfect agreement with the Gaussian distribution was not obtained, mainly because of the small statistics. The standard deviation for the total measured number of errors is $\sigma = 5.9$ bits, which represents 0.023% variation in the measurement.

There is an important observation, that the *Open* faults are completely stable, as well as *LUT*s. *Unused* and *antenna* faults do not suffer from measurement variation, because such errors are not recorded at all.

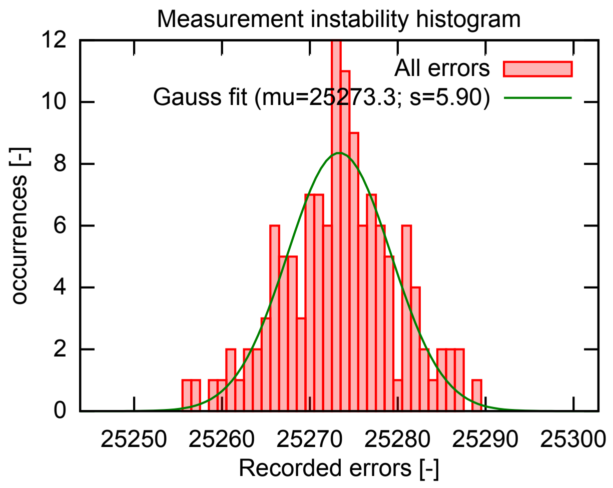


Figure 5.9: Overall measurement instability

Fault model	μ	σ	σ [%]
Unused	0	-	-
Alternate	6399.2	0.8	0.01
Open	3765	0	0
Conflict 0-F	1954.7	1.77	0.09
Conflict F-F	12651.5	5.44	0.04
Antenna	0	-	-
Unpredictable	119.3	2.09	1.8
Unknown	383.3	0.63	0.16

FPGA resource	μ	σ	σ [%]
LUT	3698	-	-
Cell interconn.	8827.8	4.29	0.05
Bus-to-cell conn.	2091.2	1.81	0.09
Bus crossing	818.6	1.72	0.2
Bus repeater	9454.8	2.55	0.03
Unexplored	383.3	0.63	0.16

Table 5.6: Measurement stability according to the FPGA resources and fault models

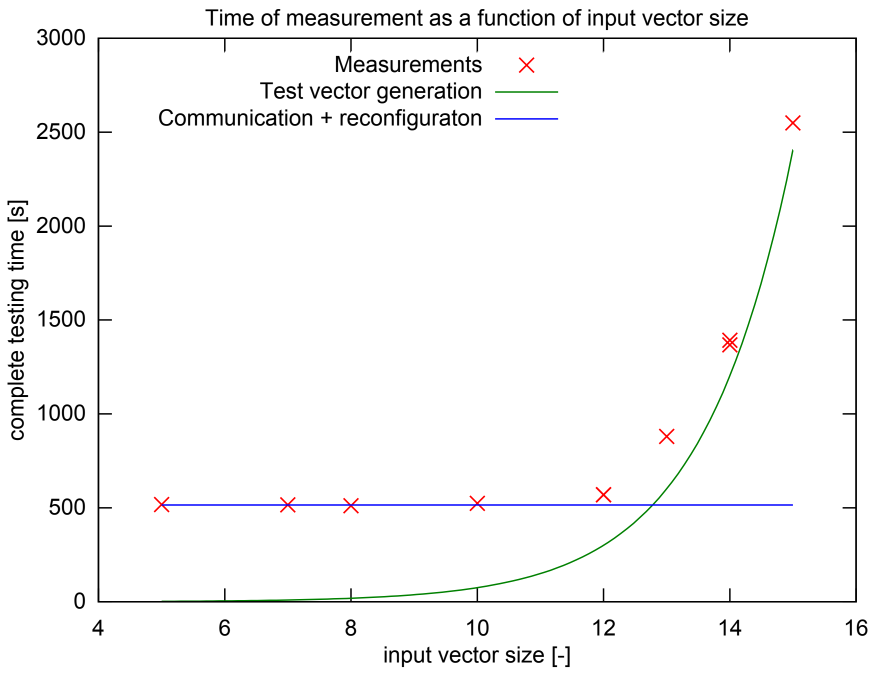


Figure 5.10: Time of measurement

5.5 Time of Measurement

The total time of measurement is shown at 5.10. The graph does not include the programming time and bitstream preparation time. The bottleneck for low-input-count benchmarks is the communication with the PC (most limiting), reconfiguration and FPGA $\leftrightarrow$AVR synchronization. The test vector generation take over the dominant part of the time for circuits with $n > 14$, where n is the number of inputs. Then the time increase exponentially by factor 2 for every other input. This makes the testing impractical for circuits with $n > 16$ and extremely time consuming for $n > 20$, when the measurement takes >1 day. The frequency used for testing is 1.8432 MHz, more than 1 order of magnitude below the maximal frequency allowed by the benchmark.

5.6 Discussion

The measurement instability (described in section 5.4) was quite unexpected. Though the standard deviation is only 0.023%, the measured values should be exact, since the measurements were taken during the same conditions, by same timing and by same input vectors applied. The sequence of results was also studied, whether the results exhibit any pattern that would indicate any time or temperature dependence of the results. No dependence was found and the measurement results seem to fluctuate completely randomly.

There is a possibility to compare the here measured results with the results predicted in the SW simulation [37], which is shown in Fig. 5.11. The results are not directly

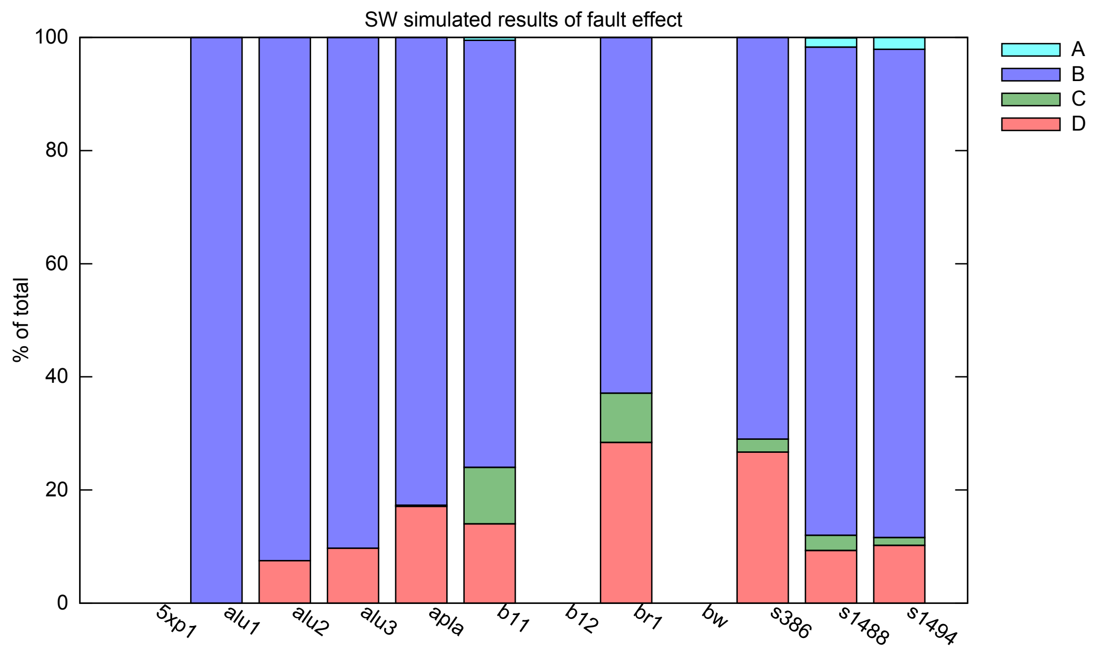


Figure 5.11: Results of the SW simulation of the same benchmark

comparable to results from complete emulator (shown in Fig. 5.7), since a different tool flow was used for synthesis. The software simulation is based on a simple LUT-representation of the circuit. Similar results can be also extracted from results presented in this work. The only-used-LUTs result of the fault emulator is shown in Fig. 5.12. The direct comparison of SW and HW effects (Figs. 5.7 and 5.11) indicates a generally increased number of A faults. Higher count of C classes has been measured also in the hardware emulator. The correlation is shown in Fig. 5.13. The number of compared circuits is too small to provide a decisive result, but the correlation between C faults in SW simulation and HW emulation restricted to the LUTs only provides best agreement, though a different synthesis tool was run on those benchmarks.

The biggest problem of circuits implemented in the enormous presence of A category faults, which do not affect the circuit directly, but when more of them became active, the combination may result in an error of the circuit. Any presence of the A fault break the ST (Self-Testing) property as well as the TSC property. There is still a question how to deal with the class A faults. Those faults may be originated:

1. From the design, where some form of redundancy may be present, or only some combinations can occur at the input of some LUTs, which leaves some bits in LUTs never used.

2. From the FPGA, which will always contain some unused resources.

While the source 1 can be very well identified by the ATPG, the source 2 depends mainly on the FPGA architecture and result of the place & route process and creates an extreme

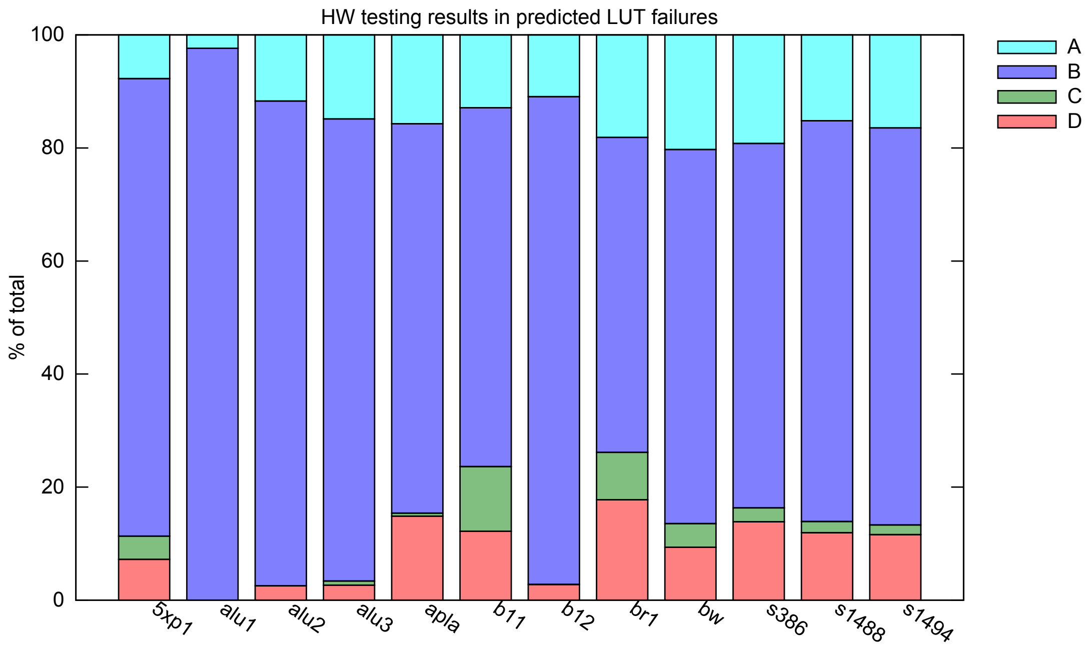


Figure 5.12: LUT-only fault category, which includes only used LUTs of the benchmark and also those LUTs used for routing

amount of faults A, which would imply, that a circuit implemented in the FPGA can almost never be a ST or TSC.

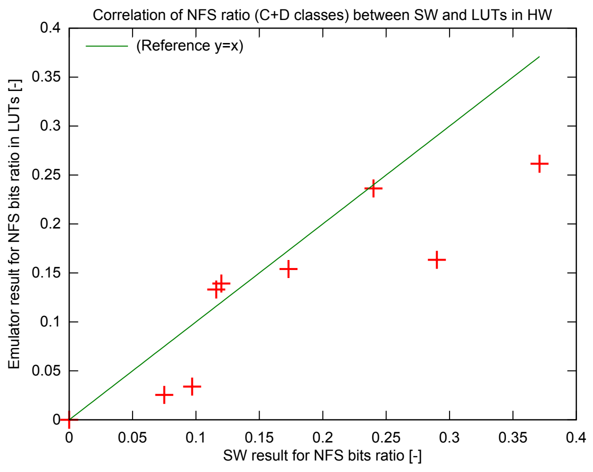
5.7 Summary

12 different benchmarks were used. Those benchmarks were picked by their size in order to fit reasonable in the FPSLIC. Two benchmarks (s1488 and s1494) reached the limits of the FPGA. All these benchmarks were tested with the parity predictors and 10 of them were also tested without the parity predictor, as shown in Table 5.5. All these 22 circuits have been subjected to a software analysis and fault prediction. Since all tested circuits were mapped, the results provide valuable information about the routing resources sensitivity.

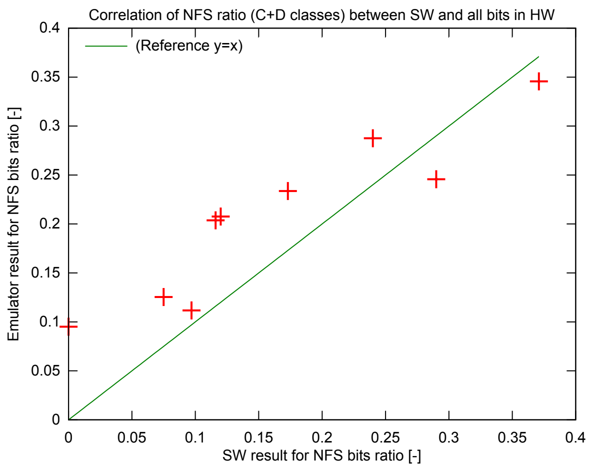
Several statistics have been presented in this section showing, that an average LUT utilization is $\sim$ 10 bits and the total number of sensitive bits $\sim 7\times$ higher compared to the LUT sensitive bits

The average prediction efficiency is $\sim$ 84%, meaning that the rest of the predicted faults ($\sim$ 16%) has no effect on the design due to known hidden faults in the benchmarks (which manifest themselves mostly in the LUTs) and other faults in the other resources of the FPGA. The false prediction rate (faults predicted as harmless, but causing an error) is 0, which was one of the requirements.

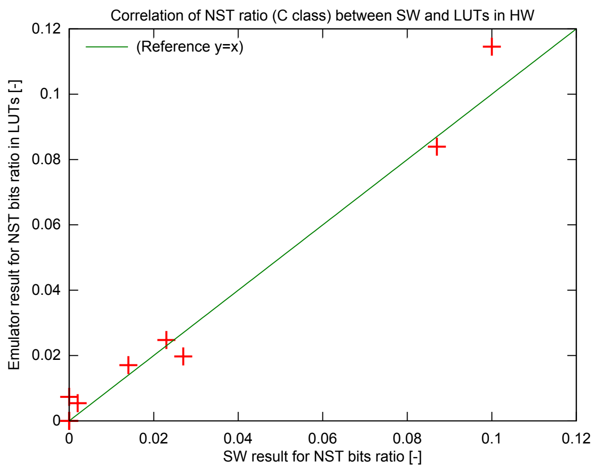
An interesting variation of the measurement of predicted sensitive bits has been ob-



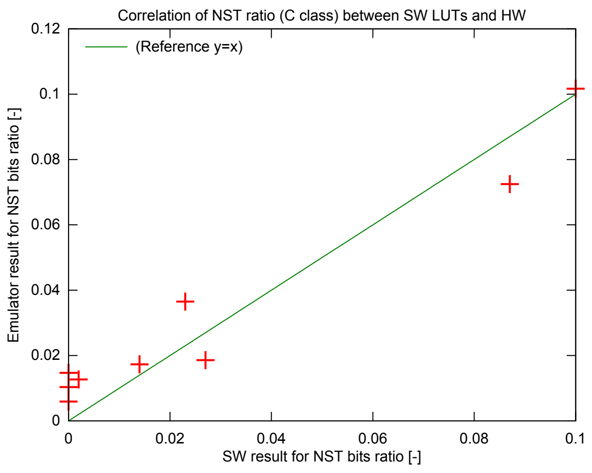
(a) NFS value, LUTs only



(b) NFS value, complete FPGA



(c) NST, LUTs only



(d) NST, complete FPGA

Figure 5.13: Correlation between SW and HW testing. Software-computed NFS (C+D faults) or NST (C faults) values compared to the LUTs only or complete FPGA testing results

served, especially for the *conflicts*, but no variations of *LUT alternation* or *open* faults were observed.

Benchmarks with parity predictors had been subjected to detailed measurements of fault classes according to the CED classification. Unique results grouped by the fault models and FPGA resources were measured and presented here.

Chapter 6

Conclusions

Single Event Upsets contribute significantly to the long-term reliability of system based on the SRAM-based FPGAs. A particle can cause a bit-flip of any flip-flop, depending on the affected area. Largest amount of flip-flips contains the configuration memory (up to 90%). The configuration memory is therefore most susceptible to SEU in terms of total number of errors.

The SEU is a serious operational problem for environments with higher radiation, such as avionics, spaces, or other artificial radiation environments. The reliability problem arises with the bigger devices, where the growth in number of transistors is not compensated by an equal decrease of the SEU cross-section per bit. As a result the overall SEU rate may rise up to several errors in year for the largest devices.

As SEU susceptibility problem became more imminent, the FPGA vendors are starting to address it in their latest models and EDA tools. An illustrative example is the Soft Error Mitigation IP core released in 2010 by Xilinx, which is able to detect errors in the configuration memory by a periodical checking and also provide a possibility to fix those errors completely from within the FPGA.

This doctoral thesis studies in detail the SEU in the configuration memory on the combinational circuits, which are placed and routed in the FPGA. Most of the related studies focus on the global statistics of the SEU effect, but results obtained in this research are quite unique, as every bit error in the configuration memory is assigned to one of the fault models and one of the FPGA resources. This provides a unique statistics, which is further improved by the monitoring of an effect on circuits, that are equipped with parity, a simplest CED mechanism. These combined results provide an important about the SEU manifestation in circuits secured with the parity predictors, which is different for the HDL-level description (even technology mapped) and the same designs placed and routed in the FPGA.

The SEU in the configuration memory can be emulated directly inside the FPGA by a single-bit alternation of the whole bitstream, or more effectively, by a dynamic partial reconfiguration of only few resources in the FPGA. The platform for the fault injection was implemented in the FPSLIC SoC, which is capable of dynamic reconfiguration with fine granularity of 1 byte. The control, testing and CED class evaluation was completely contained within a single FPSLIC device.

Fault models of placed&routed design were introduced as well as methods, how to classify (almost) any bit of the configuration file. The classification was the hardest part of this doctoral thesis, since a complete knowledge of the FPGA structure is needed, as well as an exact bitstream knowledge (which bit controls which resource). This information is generally not publicly available and sometimes even some precautions against obtaining such information are taken in order not to compromise intellectual property of the FPGA itself or IP cores of the loaded design. Separate appendices are dedicated to a description of a method, how the knowledge can be obtained and to a description of the bitstream itself.

Based on the precise knowledge of the FPGA architecture, a procedure, how to predict these faults, has been implemented for the FPSLIC and has been described in this thesis in detail. The *unused* and *antenna* fault models represent one of the greatest achievements of this work, since a prediction of these faults basically sort out bits having no functional influence on the design at all. FPGA resource classes were also defined for a detailed statistical overview of the results.

As a result of in-hardware testing, various parameters of various circuits have been observed. The findings can be summarized by following points:

- Emulator that emulates the SEU in the configuration memory of the FPGA has been developed and used extensively.

- Bit-error effect in the configuration memory has been precisely measured for all bits of the configuration memory.

- Prediction of the bit-error effect, based on the bitstream analysis, has been implemented and experimentally validated by the emulator.

- The prediction method is capable of sorting out the harmless bits from the whole bitstream and thus determines the exact number of sensitive bits. Even for the largest designs, there is $\sim 80\%$ of bits, that do not propagate to the output.

- Valuable results, showing in average a $7\times$ higher number of sensitive bits, that would one expect by the LUT-level analysis.

- Variation of the *conflict* fault effects with the standard deviation 5.9 bits (0.023%) has been observed and studied.

- Measurements of fault classes (those defined in 4.1) in the parity secured benchmarks were performed and globally showed an increased number of A class faults over the software evaluation.

6.1 Contributions of the Thesis

The ollowing list summarizes the contributions of this doctoral thesis, which are valuable and might be useful for other studies in the field of reliable FPGA systems:

1. Introduction of fault models for single bit error in the configuration memory with regards to the placed&routed designs in the FPGA.
2. Prediction method for the single bit error effect of the design mapped in the FPGA, which can evaluate the number of sensitive bits.
3. An emulator platform, which evaluates the SEU effect and even the CED reliability properties.
4. Experimental results of the soft error effect on designs mapped in the FPGA.
5. Described FPSLIC structure with bitstream addressing with guidelines of bitstream analysis of any FPGA device.

6.2 Future Work

Such basic research has so many spinoffs to the real world application that some works have been already started and some results have been already presented. There is an abits library [48] – a bitstream manipulation library, which represents a perfect tool for playing with bitstream of FPSLIC device.

The FPGA vendors recently implemented tools and cores for SEU effect mitigation, as described in chapter 3 with the SEM IP core as a bright example from Xilinx of such effort.

The author of the doctoral thesis suggests exploring the following:

- It would be interesting to have an FPGA, that exists as a *modern, commercially available* chip and has a *known structure* (up the transistor lever ideally) and *known bitstream.* Existance of such commercial FPGA goes straight against the intellectual-property-preserving effort of the vendors, so it is not expected, that there will be a research headed to a way of complete structure knowledge based on an modern FPGA. However, it would be the best way for developing reliable designs and validating their reliability.
- It would very useful to move the emulator on a modern FPGA platform. An emulator is newly directly provided by the FPGA vendor [74], but only "blind" testing is possible without further information of what resource the bit controls and what type fault is caused by the bit-flip. It is possible, that additional fault models would have to be introduced, as more advanced structures are used. The biggest problem is the detailed bitstream and FPGA knowledge.

- Application of the SEU sensitivity bits prediction to the place&route algorithm of the fixed design may improve the design reliability basically just at a cost of time spent on the increased place&route effort. It would be interesting to see the relation between the reliability improvement (in units of sensitive bits) and area overhead due to the "reliability place&route".

- Since the configuration memory fault mechanisms are different, than the stuck-at, it would be interesting to see, what would be the coverage of test vectors generated by ATPG (which should cover all faults) compared to the exhaustive test.

- The fault prediction is very fast and might be used as a driving force for the routing algorith, instead of other routing algorithm, such as timing-driven.

- It would be interesting to see the results from the synchronous designs (benchmarks). There are actually two problems to be addressed:

  1. Bit-flips configuration memory, that control the clock and reset resources. The prediction method is expandable quite easily, but the emulator is not designed to cope with sequential circuits. Depending on the design, the evaluation might require much more time, because it may take a lot of clock cycles until the bit-flip manifest itself
  2. Bit-flips in the registers and RAMs, where the configuration memory of the FPGA is untouched. This problem does not have to deal with the FPGA structure knowledge at all and the simulation can substitute the emulation. However, there might be cases, where the register upsetting is faster, than the simulation. The "injector" of fault can be directly synthesized with the design (area is increased), or the fault can be "injected" by a dynamic reconfiguration of the reset wire to the affected register and then by activation of the reset.

Bibliography

[1] SEU Flux calculation.
http://www.seutest.com/FluxCalculator.htm.

[2] JEDEC JESD89A standard. Measurement and reporting of alpha particle and terrestrial cosmic ray-induced soft errors in semiconductor devices, 2006.

[3] Actel. RadTolerant FPGAs.
http://www.microsemi.com/document-portal/doc_download/130711-radtolerant-fpgas, 2004.

[4] E. Ahmed and J. Rose. The effect of LUT and cluster size on deep-submicron FPGA performance and density. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 12(3):288–298, march 2004.

[5] Altera. White paper WP-01012-1.0: Robust SEU mitigation with Stratix III FPGAs.
http://www.altera.com/literature/wp/wp-01012.pdf, 2007.

[6] Altera. White paper WP-01095-1.2: Generating functionally equivalent FPGAs and ASICs with a single set of RTL and synthesis/timing constraints.
http://www.altera.com/literature/wp/wp-01095-rtl-synthesis-timing.pdf, February 2009.

[7] Altera. White paper WP-01135-1.0, enhancing robust SEU mitigation with 28-nm FPGAs.
http://www.altera.com/literature/wp/wp-01135-stxv-seu-mitigation.pdf, July 2010.

[8] Altera. About Stratix family high-end FPGAs and SoCs.
http://www.altera.com/devices/fpga/stratix-fpgas/about/stx-about.html, 2013.

[9] Altera. SEU mitigation for Stratix V devices.
http://www.altera.com/literature/hb/stratix-v/stx5_51011.pdf, 2013.

[10] J.H. Anderson and F.N. Najm. Low-power programmable routing circuitry for FPGAs. In *Proceedings of the 2004 IEEE/ACM International conference on Computer-aided*

design, ICCAD '04, pages 602–609, Washington, DC, USA, 2004. IEEE Computer Society.

[11] Atmel. Application note on AT94K series configuration (doc2313), web archived entry.
`http://web.archive.org/web/20101214160317/http://atmel.com/dyn/resources/prod_documents/DOC2313.PDF`.

[12] Atmel. Web pages.
`http://www.atmel.com/`.

[13] Atmel. Application note on AT40K series configuration (doc1009).
`http://www.atmel.com/Images/DOC1009.PDF`, March 2002.

[14] Atmel. FPGA integrated development systems (IDS).
`http://www.atmel.com/tools/fpgaintegrateddevelopmentsystems_ids_.aspx`, 2005.

[15] Atmel. AT40K05/10/20/40AL complete datasheet (doc2818).
`http://www.atmel.com/Images/doc2818.pdf`, Jul 2006.

[16] Atmel. AT94KAL series field programmable system level integrated circuit datasheet (doc1138), web archived entry.
`http://web.archive.org/web/20101121233705/http://atmel.com/dyn/resources/prod_documents/doc1138.pdf`, January 2008.

[17] R.C. Baumann. Radiation-induced soft errors in advanced semiconductor technologies. *IEEE Transactions on Device and Materials Reliability*, 5(3):305–316, 2005.

[18] M. Bellato, P. Bernardi, D. Bortolato, A. Candelori, M. Ceschia, A. Paccagnella, M. Rebaudengo, M.S. Reorda, M. Violante, and P. Zambolin. Evaluating the effects of SEUs affecting the configuration memory of an SRAM-based FPGA. *Design, Automation and Test in Europe Conference and Exhibition, 2004. Proceedings*, 1:584–589 Vol.1, Feb. 2004.

[19] V. Betz and J. Rose. Circuit design, transistor sizing and wire layout of FPGA interconnect. In *Custom Integrated Circuits, 1999. Proceedings of the IEEE 1999*, pages 171–174, 1999.

[20] V. Betz and J. Rose. FPGA routing architecture: segmentation and buffering to optimize speed and density. In *Proceedings of the 1999 ACM/SIGDA seventh international symposium on Field programmable gate arrays*, FPGA '99, pages 59–68, New York, NY, USA, 1999. ACM.

[21] V. Betz, J. Rose, and A. Marquardt. *Architecture and CAD for Deep-Submicron FPGAs*. Kluwer Academic Publishers, Norwell, MA, USA, 1999.

[22] R. Brayton, G. Hachtel, C. McMullen, and A. Sangiovanni-Vincentelli. *Logic minimization algorithms for VLSI synthesis*, volume 2 of *Kluwer international series in engineering and computer science: VLSI, computer architecture, and digital signal processing*. Kluwer Academic Publishers, 1984.

[23] C. Carmichael and W. Chen. Xilinx application note XAPP1088 (v1.0) correcting single-event upsets in Virtex-4 FPGA configuration memory. `http://vts.uni-ulm.de/docs/2007/5956/vts_5956_7989.pdf`, October 2009.

[24] Yao-Wen Chang, D. F. Wong, and C. K. Wong. Universal switch modules for FPGA design. *ACM Transactions on Design Automation of Electronic Systems (TODAES)*, 1(1):80–101, January 1996.

[25] K. Chapman and L. Jones. Xilinx application note XAPP864 (v1.0.1): SEU strategies for Virtex-5 devices. `http://tec.icbuy.com/uploads/2010/5/24/xapp864.pdf`, 2009.

[26] Hu Ching and Zain Suhail. Xilinx application note XAPP1073: NSEU mitigation in avionics applications. `http://www.xilinx.com/support/documentation/application_notes/xapp1073_NSEU_Mitigation_Avionics.pdf`, June 2010.

[27] International Electrotechnical Commission et al. IEC 60050-191 international electrotechnical vocabulary, Chapter 191: Dependability and quality of service, 1990.

[28] B.K. Fawcett, N. Sawyer, and T. Williams. Hardwire: A risk-free FPGA-to-ASIC migration path. In *Field-Programmable Logic Architectures, Synthesis and Applications*, volume 849 of *Lecture Notes in Computer Science*, pages 280–282. Springer Berlin Heidelberg, 1994.

[29] P. Fiser and H. Kubatova. Flexible two-level Boolean minimizer BOOM-II and its applications. In *9th EUROMICRO Conference on Digital System Design: Architectures, Methods and Tools DSD 2006*, pages 369–376, 2006.

[30] D.M. Fleetwood., S.L. Kosier, R.N. Nowlin, R.D. Schrimpf, R.A. Reber, Jr., M. DeLaus, P.S. Winokur, A. Wei, W.E. Comb, and R.L. Pease. Physical mechanisms contributing to enhanced bipolar gain degradation at low dose rates. *IEEE Transactions on Nuclear Science*, 41(6):1871–1883, 1994.

[31] W. S. Gosset. Atmel AT40K/94K Configuration Format Documentation. Usenet comp.fpga.arch, `https://groups.google.com/d/msg/comp.arch.fpga/RWy5ojQU1gM/K47-qoLKD6kJ`, August 2005.

[32] Andrew Gordon Holmes-Siedle and Len Adams. *Handbook of radiation effects, second edition*. Oxford University Press, 2002.

[33] J. Hussein and G. Swift. Xilinx white paper WP395 (V1.0) mitigating single-event upsets.
`http://www.xilinx.com/support/documentation/white_papers/wp395-Mitigating-SEUs.pdf`, April 2012.

[34] Krzysztof Iniewski. *Radiation detection and measurement.* CRC Press, Taylor & Francis Group, 2010. ISBN 978-1-4398-2694-2.

[35] J. Beringer et al. (Particle Data Group). Review of particle physics. *Phys. Rev. D*, 86:010001, Jul 2012.

[36] A.H. Johnston, B.G. Rax, and C.I. Lee. Enhanced damage in linear bipolar integrated circuits at low dose rate. *Nuclear Science, IEEE Transactions on*, 42(6):1650–1659, 1995.

[37] L. Kafka, P. Kubalík, H. Kubátová, and O. Novák. Fault classification for self-checking circuits implemented in FPGA. In *Proceedings of IEEE Design and Diagnostics of Electronic Circuits and Systems Workshop. Sopron University of Western Hungary*, pages 228–231, 2005.

[38] I. Kuon and J. Rose. Measuring the gap between fpgas and asics. *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, 26(2):203–215, 2007.

[39] I. Kuon, R. Tessier, and J. Rose. FPGA architecture: Survey and challenges. *Found. Trends Electron. Des. Autom.*, 2(2):135–253, February 2008.

[40] G. Lemieux and D. Lewis. Circuit design of routing switches. In *Proceedings of the 2002 ACM/SIGDA tenth international symposium on Field-programmable gate arrays*, FPGA '02, pages 19–28, New York, NY, USA, 2002. ACM.

[41] G. Lemieux and D. Lewis. *Design of Interconnection Networks for Programmable Logic.* Kluwer Academic Publishers, Norwell, MA, USA, 2004.

[42] A. Lesea. Xilinx white paper WP286 (v1.1): Continuing experiments of atmospheric neutron effects on deep submicron integrated circuits.
`http://www.xilinx.com/support/documentation/white_papers/wp286.pdf`, 2011.

[43] A. Lesea and K. Castellani-Coulie. Experimental study and analysis of soft errors in 90nm Xilinx FPGA and beyond. In *Radiation and Its Effects on Components and Systems, 2007. RADECS 2007. 9th European Conference on*, pages 1–5, 2007.

[44] A. Lesea, S. Drimer, J. Fabula, C. Carmichael, and P. Alfke. The Rosetta experiment: atmospheric soft error rate testing in differing technology FPGAs. *IEEE Transactions on Device and Materials Reliability*, 5(3):317–328, September 2005.

[45] M.I. Masud and S.J.E. Wilton. A new switch block for segmented FPGAs. In Patrick Lysaght, James Irvine, and Reiner Hartenstein, editors, *Field Programmable Logic and Applications*, volume 1673 of *Lecture Notes in Computer Science*, pages 274–281. Springer Berlin Heidelberg, 1999.

[46] T.C. May and M.H. Woods. Alpha-particle-induced soft errors in dynamic memories. *Electron Devices, IEEE Transactions on*, 26(1):2–9, 1979.

[47] A. Megacz. AT94K bitstream description, last modification 2007.09.01.
`http://git.megacz.com/?p=slipway.git;a=blob_plain;f=doc/gosset.txt`.

[48] A. Megacz. Slipway and abits.
`http://research.cs.berkeley.edu/project/slipway/`.

[49] A. Megacz. A library and platform for FPGA bitstream manipulation. In *Field-Programmable Custom Computing Machines, 2007. FCCM 2007. 15th Annual IEEE Symposium on*, pages 45 –54, april 2007.

[50] R. Mehrotra, M. Pedram, and Xunwei Wu. Comparison between nMOS pass transistor logic style vs. CMOS complementary cells. In *Computer Design: VLSI in Computers and Processors, 1997. ICCD '97. Proceedings., 1997 IEEE International Conference on*, pages 130–135, 1997.

[51] G.C. Messenger. Collection of charge on junction nodes from ion tracks. *Nuclear Science, IEEE Transactions on*, 29(6):2024–2031, 1982.

[52] S. Mitra and E.J. McCluskey. Which concurrent error detection scheme to choose? In *Test Conference, 2000. Proceedings. International*, pages 985–994, 2000.

[53] A. Moradi, A. Barenghi, T. Kasper, and C. Paar. On the vulnerability of FPGA bitstream encryption against power analysis attacks: Extracting keys from Xilinx Virtex-II FPGAs. In *Proceedings of the 18th ACM Conference on Computer and Communications Security*, CCS '11, pages 111–124, New York, NY, USA, 2011. ACM.

[54] NIST, Physical Measurement Laboratory. PSTAR and ESTAR – stopping power and range tables for protons and electrons in various material.
`http://physics.nist.gov/PhysRefData/Star/Text/PSTAR.html`
`http://physics.nist.gov/PhysRefData/Star/Text/ESTAR.html`.

[55] T.R. Oldham and F.B. McLean. Total ionizing dose effects in mos oxides and devices. *Nuclear Science, IEEE Transactions on*, 50(3):483–499, 2003.

[56] Dhiraj K. Pradhan, editor. *Fault-tolerant computer system design*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1996.

[57] H. Quinn, P. Graham, J. Krone, M. Caffrey, and S. Rezgui. Radiation-induced multi-bit upsets in SRAM-based FPGAs. *Nuclear Science, IEEE Transactions on*, 52(6):2455–2461, 2005.

[58] B.G. Rax, A.H. Johnston, and C.I. Lee. Proton damage effects in linear integrated circuits. *Nuclear Science, IEEE Transactions on*, 45(6):2632–2637, 1998.

[59] R.A. Reed, M. A. Carts, P.W. Marshall, C.J. Marshall, O. Musseau, P.J. McNulty, D. R. Roth, S. Buchner, J. Melinger, and T. Corbiere. Heavy ion and proton-induced single event multiple upset. *Nuclear Science, IEEE Transactions on*, 44(6):2224–2229, 1997.

[60] H. Schmit and V. Chandra. FPGA switch block layout and evaluation. In *Proceedings of the 2002 ACM/SIGDA 10th international symposium on Field-programmable gate arrays*, FPGA '02, pages 11–18, New York, NY, USA, 2002. ACM.

[61] R.D. Schrimpf and D.M. Fleetwood. *Radiation effects and soft errors in integrated circuits and electronic devices*, volume 12. World Scientific, 2004.

[62] K. Shibata et al. JENDL-4.0: A new library for nuclear science and engineering. *Journal of Nuclear Science and Technology*, 48(1):1–30, 2011.

[63] L. Sterpone and M. Violante. Analysis of the robustness of the TMR architecture in SRAM-based FPGAs. *Nuclear Science, IEEE Transactions on*, 52(5):1545–1549, 2005.

[64] L. Sterpone and M. Violante. A new analytical approach to estimate the effects of SEUs in TMR architectures implemented through SRAM-based FPGAs. *Nuclear Science, IEEE Transactions on*, 52(6):2217–2223, 2005.

[65] D.J. Thomas. ICRU report 85: Fundamental quantities and units for ionizing radiation. *Radiation Protection Dosimetry*, 150(4):550–552, 2012.

[66] M.C. Thorne. Background radiation: natural and man-made. *Journal of Radiological Protection*, 23(1):29, 2003.

[67] United Nations, Scientific Committee on the Effects of Atomic Radiation. *Effects of Ionizing Radiation: Report to the General Assembly, with scientific annexes*, volume 1. United Nations Publications, 2008.
`http://www.unscear.org/docs/reports/2008/09-86753_Report_2008_GA_Report_corr2.pdf`.

[68] Fan Wang and V.D. Agrawal. Single event upset: An embedded tutorial. In *21st International Conference on VLSI Design, 2008. VLSID 2008.*, pages 429–434, 2008.

[69] Xilinx. JBits SDK for Virtex-II.
`http://www.xilinx.com/labs/projects/jbits/`.

[70] Xilinx. The industry's first development tool to automatically generate Triple Module Redundancy (TMR) for space-grade re-programmable FPGAs. `http://www.xilinx.com/ise/optional_prod/tmrtool.htm`.

[71] Xilinx. Application note XAPP452: Spartan-3 FPGA family advanced configuration architecture. `http://www.xilinx.com/support/documentation/application_notes/xapp452.pdf`, 2008.

[72] Xilinx. Moving a generation ahead with all programmable FPGAs, SOCs, and 3D ICs. `http://www.xilinx.com/publications/prod_mktg/Generation-Ahead-Backgrounder.pdf`, 2012.

[73] Xilinx. User guide UG473 (v1.7): 7 series FPGAs memory resources. `http://www.xilinx.com/support/documentation/user_guides/ug473_7Series_Memory_Resources.pdf`, October 2012.

[74] Xilinx. White paper WP414: SEU emulation environment. `http://www.xilinx.com/support/documentation/white_papers/wp414-SEU-Emulation.pdf`, April 2012.

[75] Xilinx. Product guide PG036: LogiCORE IP Soft Error Mitigation Controller v4.0. `http://www.xilinx.com/support/documentation/ip_documentation/sem/v4_0/pg036_sem.pdf`, June 2013.

[76] Xilinx. User guide UG116 (v9.4): Device reliability report. `http://www.xilinx.com/support/documentation/user_guides/ug116.pdf`, May 2013.

[77] Saeyang Yang. *Logic synthesis and optimization benchmarks user guide: version 3.0.* Citeseer, 1991.

[78] J.F. Ziegler, M.D. Ziegler, and J.P. Biersack. SRIM – the stopping and range of ions in matter (2010). *Nuclear Instruments and Methods in Physics Research Section B: Beam Interactions with Materials and Atoms*, 268(11–12):1818 – 1823, 2010.

[79] R. Zimmermann and W. Fichtner. Low-power logic styles: CMOS versus pass-transistor logic. *Solid-State Circuits, IEEE Journal of*, 32(7):1079–1090, 1997.

Publications of the Author

Reviewed Relevant Publications of the Author

[A.1] J. Kvasnička, P. Kubalík, and H. Kubátová. Experimental SEU Impact on Digital Design Implemented in FPGAs. In *Proc. 11th Euromicro Conference on Digital System Design (DSD 2008)*, pp. 100–103, Los Alamitos: IEEE Computer Society, 2008, ISBN 978-0-7695-3277-6, doi:10.1109/DSD.2008.119.

[A.2] P. Kubalík, J. Kvasnička, and H. Kubátová. Fault injection and simulation for fault tolerant reconfigurable duplex system. In *Proc. IEEE Design and Diagnostics of Electronic Circuits and Systems DDECS '07*, pp. 357–360, 2007, Los Alamitos: IEEE Computer Society, doi:10.1109/DDECS.2007.4295312.

[A.3] J. Kvasnička and H. Kubátová. Emulation of SEU Effect In Bitstream of FPGA. In *Proc. 4th Doctoral Workshop on Mathematical and Engineering Methods in Computer Science*, pp. 140-147 Brno, ISBN 978-80-7355-082-0, 2008.

[A.4] J. Kvasnička, P. Kubalík, and H. Kubátová. Experimental emulation of FPGA bitstream faults in combinatorial circuits. In *Proc. CSE 2008 International Scientific Conference on Computer Science and Engineering*, pp. 328-335, Košice: Department of Computers and Informatics of FEI, Technical University Košice, vol. 1. ISBN 978-80-8086-092-9, 2008.

Remaining Relevant Publications of the Author

[A.5] J. Kvasnička. Vysoce spolehlivý systém založený na bázi hradlových polí (in czech). *Master's thesis.* Czech Technical University in Prague, Faculty of Electrical Engineering, 2006.

[A.6] J. Kvasnička. Vliv poruch v bitstreamu na funkci obvodů v FPGA. In *Proc. Počítačové architektúry a diagnostika*, pp. 89–94, Ústav informatiky SAV, Bratislava, ISBN 80-969202-2-7, 2006.

[A.7] J. Kvasnička, P. Kubalík, and H. Kubátová. An FPGA based fault emulator. In *Proc. Work in Progress Session held in connection with the EUROMICRO Confer-*

ences SEAA and DSD 2007, pp. 42–43. Linz: Johannes Kepler University, ISBN 978-3-902457-16-5, 2007.

[A.8] J. Kvasnička and H. Kubátová. Emulation of SEU Effect In Bitstream of FPGA. In *Proc. 4th Doctoral Workshop on Mathematical and Engineering Methods in Computer Science*, pp. 140-147 Brno, ISBN 978-80-7355-082-0, 2008.

[A.9] J. Kvasnička. Fault-tolerant circuit – FPGA fault analysis. *Doctoral Study Report*, Faculty of Electrical Engineering, CTU Prague, Czech Republic, 2008.

[A.10] J. Kvasnička. FPGA bitstream analysis and emulation of SEU effect. In *Proc. Počítačové architektury & diagnostika*, pp. 63-68, Liberec: Technická univerzita v Liberci, 2008, ISBN 978-80-7372-378-1, 2008.

[A.11] J. Kvasnička and H. Kubátová. Single Event Upset Tolerant FPGA Design. In *Proc. Work in Progress Session SEAA 2009 and DSD 2009*, pp. 37-38, Linz: J. Kepler University - FAW, ISBN 978-3-902457-25-7, 2009.

Relevant Citations of Author's Publications

[C.1] J. Borecký, M. Kohlík, H. Kubátová. How to Measure Dependability Parameters of Programmable Digital Circuits – A Survey. In *Proc. of 6th Doctoral Workshop on Mathematical and Engineering Methods in Computer Science.* Brno: NOVPRESS, 2010, pp. 28-35, ISBN:978-80-87342-10-7. Cites [A.8]

[C.2] Jančik M., Dynamic Reconfiguration with Atmel FPSLIC (in czech), *Master's Thesis*, Brno University of Technology, Faculty of Electrical Engineering and Communication Department of Radio Electronics, 2010. Cites [A.5]

[C.3] J. Fan, Z. Zhang, Speeding up Fault Simulation using Parallel Fault Simulation, CEIS 2011, *Procedia Engineering*, Volume 15, Pages 1817–1821, 2011, ISSN 1877-7058, doi:10.1016/j.proeng.2011.08.338. Cites [A.2]

[C.4] J. Borecký, M. Kohlík, P. Kubalík, H. Kubátová, Fault Models Usability Study for On-line Tested FPGA. In *Proc. 14th Euromicro Conference on Digital System Design (DSD 2011)*, pp. 287–290, 2011, Los Alamitos: IEEE Computer Society, ISBN:978-1-4577-1048-3, doi:10.1109/DSD.2011.42. Cites [A.1]

[C.5] Vojtěch Hovorka. Systém pro detekci dočasných poruch v hradlových polích (in Czech), *Bachelor Thesis*, Faculty of Electrical Engineering, CTU Prague, 2011. Cites [A.9]

Remaining Publications of the Author

[A.12] I. Polak, J. Kvasnicka. Fast Calibration UV LED System for CALICE Scintillator Based Tile Hadron Calorimeter. *IEEE Nuclear Science Symposium and Medical*

Imaging Conference Record (NSS/MIC) , pp.1240-1244, ISBN: 978-1-4673-2028-3 or 978-1-4673-2030-6/12, 2012, doi:10.1109/NSSMIC.2012.6551304.

[A.13] J. Kvasnicka and I. Polak. LED calibration systems for CALICE hadron calorimeter. *Physics Procedia*, 37(0):402–409, 2012. Proceedings of the 2nd International Conference on Technology and Instrumentation in Particle Physics (TIPP 2011).

[A.14] J. Kvasnicka and I. Polak on behalf of the CALICE collaboration. LED Calibration Systems for CALICE Hadron Calorimeter, *Physics Procedia*, Volume 37, 2012, Pages 402-409, ISSN 1875-3892, doi:10.1016/j.phpro.2012.02.379.

[A.15] I. Polak et al.,An LED calibration system for the CALICE HCAL, *Proc. IEEE Nuclear Science Symposium (NSS10)*, Knoxville, Tennessee, USA, Nov. 2010.

[A.16] The CALICE collaboration et al. Construction and commissioning of the calice analog hadron calorimeter prototype. *Journal of Instrumentation*, 5(05):P05004, 2010, ISSN:1748-0221, doi:10.1088/1748-0221/5/05/P05004.

[A.17] The CALICE collaboration et al. Tests of a Particle Flow Algorithm with CALICE test beam data. *Journal of Instrumentation*, 6(04):P04003, 2011, ISSN:1748-0221, doi:10.1088/1748-0221/6/04/P04003.

[A.18] The CALICE collaboration. Construction and performance of a silicon photomultiplier/extruded scintillator tail-catcher and muon-tracker. *Journal of Instrumentation*, 7(04):P04015, 2012, ISSN:1748-0221, doi:10.1088/1748-0221/7/04/P04015.

[A.19] The CALICE collaboration. Hadronic energy resolution of a highly granular scintillator-steel hadron calorimeter using software compensation techniques. *Journal of Instrumentation*, 7(09):P09017, 2012, ISSN:1748-0221, doi:10.1088/1748-0221/7/09/P09017.

[A.20] C Adloff et al. Validation of GEANT4 Monte Carlo models with a highly granular scintillator-steel hadron calorimeter. *Journal of Instrumentation*, 8(07):P07005, 2013, ISSN:1748-0221, doi:10.1088/1748-0221/8/07/P07005.

[A.21] J. Cvach et al. Magnetic field tests of the QRLED driver. *Eudet memo*, Eudet-Memo-2009-05.

[A.22] J.Cvach et al. Beam test of the QMB6 calibration board and HBU0 prototype. *Eudet memo*, Eudet-Memo-2010-21.

[A.23] J. Cvach et al. Calibration prototype for the EUDET HCAL. *Eudet report*, .EUDET-Report-2008-07.

[A.24] The CALICE collaboration, CALICE Report to the DESY Physics Research Committee, 2010.

Appendix A

Bitstream of AT94K FPGA

A.1 Statement

The information presented in this this Appendix has been solely obtained by observing the bitstream, observing the device response on the bitstream error insertions and by observing the mapped layout. The author did not have access to any confidential information related to this topic. No non-disclosure agreement was signed during this research and this research was not supported by anyone bound with any related non-disclosure agreement or licensing contract.

A.2 Introduction

This part describes the bitstream of the FPSLIC in detail. Only asynchronous logic (big combinational benchmarks) was in the scope of this thesis, therefore **a limited bitstream subset is described here** – only bits that can corrupt the combinational benchmarks were studied and verified. Description of the synchronous logic elements is limited only to registers in logic cells. Clocks, reset, I/Os and block RAMs were not investigated, nor tested, and are therefore not verified.

A complete bitstream description exists [31, 47] and differences between those versions are discussed at section A.11.

A.3 The FPSLIC

The FPSLIC (AT94KAL) embeds an FPGA core together with AVR microcontroller. The largest device from the FPSLIC family (AT94K40AL) embeds an AT40K40AL FPGA core. This SoC benefits from the direct FPGA bitstream access and from the shared data bus between the microcontroller and the FPGA. Both features were used extensively in this project. The data bus was used for fast control and fast result acquisition. The low level bitstream access was used for bitstream bit-error injections.

FPSLIC is already an obsolete (mature) product since 2011 and is therefore not recommended for new design since then. Information (datasheet and some software) was backdrawn from the public Atmel site [12] as of January 2013. Information about AT94K device can be obtained at archived pages [16].

The FPSLIC bitstream itself was a target for a bitstream revealing effort since a long time ago. First publicly available publication of the FPSLIC bitstream was released on comp.arch.fpga Usenet discussion group [31] in 2005. Later, this description was bug-fixed by Adam Megacz [47], who also created an open-hardware board with FPSLIC and wrote an FPSLIC bitstream modification open-source library *abits* [48, 49].

According to [11], the AT94K40AL (the largest, 48×48 –cell device) needs 520920 bits (65115 bytes) for a complete configuration [11], while the AT40K40 FPGA has only 336504 bits (42063 bytes) [13].

All FPGAs from Atmel (AT94K, AT40K, AT40KAL) have a possibility to directly access the FPGA bitstream in so-called *Synchronous RAM Configuration Downloads (Mode 4)* [11]. In this mode, the configuration (bitstream) SRAM is accessed as an 8-bit memory (16-bit in special mode) with 24-bits wide address. The *Mode 4* does not require the FPGA to enter any specific state during the download process [11] and can be accessed instantly. Detailed description of this mode (which should also a complete bitstream description) is available from Atmel in exchange of the signed NDA [11].

The bitstream can be accessed in a similar mode from the AVR within the FPSLIC device. AVR uses 4 dedicated AVR 8-bits registers FPGAX, FPGAY, FPGAZ and FPGAD, referred as a *FPGA Cache Logic* [16]. This mode has been used for the bitstream alternation in this doctoral thesis. In this mode, 1 byte of bitstream configuration can be downloaded into the FPGA by writing the FPGAD register after the FPGAZ, FPGAY and FPGAX registers (the address) are written.

A.4 AT40K Topology

It is important to understand the FPGA topology in order to read the bitstream easily. AT40K is a typical island topology, which has 48 horizontal cells times 48 vertical cells[1]. The MD4 bitstream of FPSLIC preserves the topology by keeping the x and y coordinates in the bitstream address. This is a key property, which makes the bitstream easy to analyze and understand.

Figures in this section show the correspondence of the bitstream values of x and y and the FPGA structure. The detailed structure (also for clock and reset distribution, I/O and RAM) is shown in [15, 16] in more detail. Complete view of the structure is provided by [14].

Logic cells are addressed by coordinates (x, y), where $(x, y) \in (\langle 0, 47\rangle, \langle 0, 47\rangle)$; $(x, y) \in (\mathbb{N}, \mathbb{N})$. Addressing of logic cells is shown in Fig. A.1.

[1]Differs according to the device size: AT94K40AL has 48×48 cells, AT94K10AL has 24×24 cells, AT94K05AL has 16×16 cells.

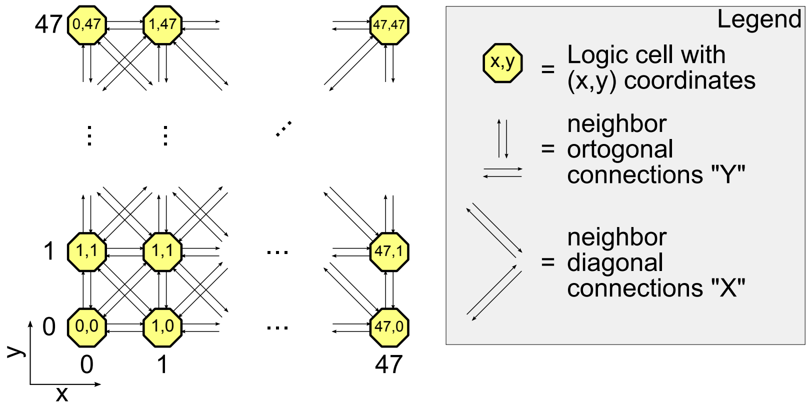

47
0,47
1,47
47,47
1
1,1
1,1
47,1
0
0,0
1,0
47,0
0
1
47
y
x
Legend
x,y
= Logic cell with (x,y) coordinates
= neighbor ortogonal connections "Y"
= neighbor diagonal connections "X"

Figure A.1: FPSLIC Cell coordinates and cell neighbor connections.

FPGAZ	FPGAY	FPGAX	FPGAD

Table A.1: MD4 bitstream configuration word byte order

The **repeater** is inserted every 4 cells (either in horizontal and vertical direction) into the bus, creating a sector of 4 × 4 cells. At a given cell coordinates (x, y), the nearest left (l), right (r), below (b) and above (a) repeater coordinates are calculated as follows: $(x_l, y_l) = (\lfloor x/4 \rfloor, y)$, $(x_r, y_r) = (\lfloor x/4+1 \rfloor, y)$, $(x_b, y_b) = (x, \lfloor y/4 \rfloor)$, $(x_a, y_a) = (x, \lfloor y/4+1 \rfloor)$.

The range of horizontal coordinates is therefore $(x, y) \in (by\langle 0, 12\rangle, \langle 0, 47\rangle)$; $(x, y) \in (\mathbb{N}, \mathbb{N})$. Range of vertical repeater is $(x, y) \in (\langle 0, 47\rangle, \langle 0, 12\rangle)$; $(x, y) \in (\mathbb{N}, \mathbb{N})$.

Repeater alternates “top” and “bottom” position in the network (“left” and “right” respectively for a vertical repeater), as shown in Fig. A.2. This makes the *local bus* (the middle bus) resource always 4 cell long and the *express bus* (corner bus) 8 cells long.

Both *local* and *express buses* are bidirectional buses and repeater can be programmed at any possibly pathways

Clock and **reset** sector resources share the coordinates with vertical repeaters: $(x, y) \in (\langle 0, 47\rangle, \langle 0, 12\rangle)$; $(x, y) \in (\mathbb{N}, \mathbb{N})$.

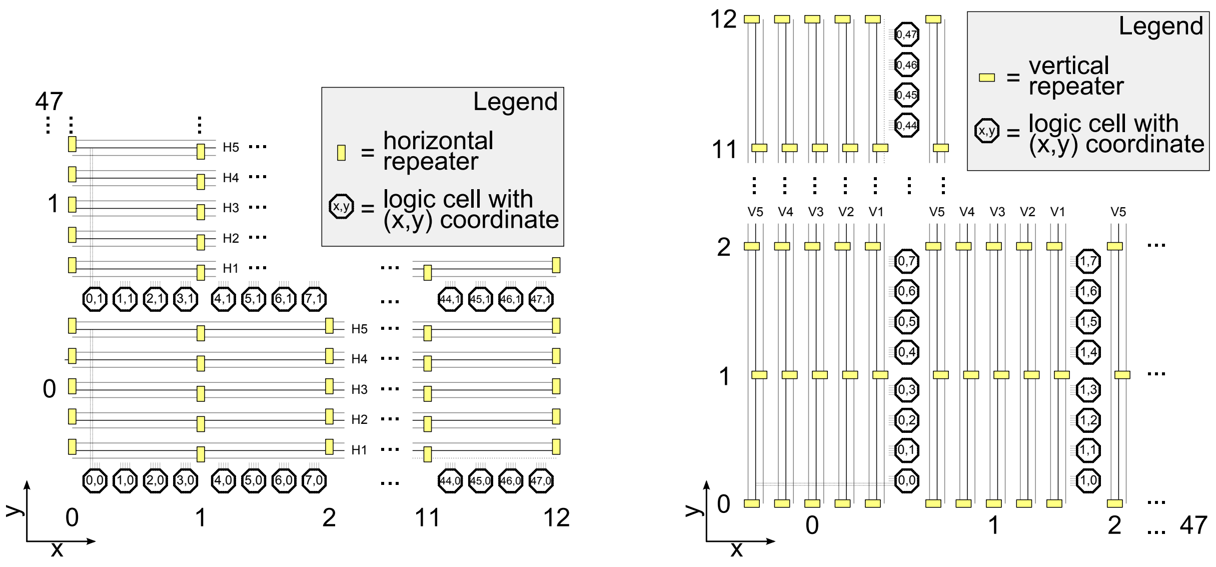
I/O resources are divided into primary and secondary I/Os. Coordinates are: $(x, y) \in (\langle 0, 1\rangle, \langle 0, 47\rangle)$; $(x, y) \in (\mathbb{N}, \mathbb{N})$ for west and east; $(x, y) \in (\mathbb{N}, \mathbb{N})$ for north and south I/Os.

Clock distribution is addressed only by$x$ coordinate, $x \in \langle 0, 47\rangle$; $x \in \mathbb{N}$. The y coordinate is always $y = 0$.

A.5 MD4 Bitstream Format

The easiest way to read the bitstream and to understand the FPGA topology is working with the Mode 4 (MD4) [11] format. The MD4 format is generated in Figaro IDS [14] tool.

The MD4 bitstream word (32 bits) consist of 8-bit configuration word (byte), that is addressed by z, x and y bytes. The byte order of the MD4 configuration word is shown in Table A.1. The z value addresses the set of configuration bits within the regular structure.



(a) Horizontal repeaters (b) Vertical repeaters

Figure A.2: FPSLIC repeater placing and coordinates.

The structure can be a logic cell, line repeaters, I/Os, clocks, reset, block RAM or others. Each structure has a different granularity, therefore a different (x, y) range, as shown at section A.4.

The z, y and x values correspond directly to the FPGAZ, FPGAY and FPGAX registers. The FPGAD register is dedicated for the bitstream data.

Example code of the MD4 bitstream:

```
001D1F01 //Cell (31,29) configuration + connection
011D1F00 //Cell configuration
021D1F00 //Cell configuration
031D1F00 //Cell config + express bus cross pass gates
041D1F01 //Cell configuration
051D1F02 //Cell configuration
061D1F00 //LUTY
071D1F22 //LUTX
081D1F6C //Cell connection
091D1F00 //Express bus crossing pass gates
...
```

The Bitstream is organized into following groups, which are loaded in following order:

1. Head (Z=0xD0..0xD3, Tab.A.7),
2. Logic Cell and bus crossing (Z=0x00-0x09, Tab. A.2),
3. Minor cell configuration (Z=0x10-0x11, Tab. A.3),

4. Horizontal repeaters (Z=0x20-0x29, Tab. A.5),
5. Vertical repeaters (Z=0x30-0x39, Tab. A.4),
6. Ram (Z=0x40-0x41, Tab. A.6),
7. Vertical I/O columns Z=(0x60-0x67, Tab. A.8),
8. Horizontal I/O rows (Z=0x70-0x77, Tab. A.8),
9. Clock (Z=0x50-0x51, Tab. A.7)

As shown at bitstream example, a complete Z group from the list is loaded for given coordinates at once. The iteration order is 1) y in valid range; 2) x in valid range; 3) z from the group.

Fast clocks are missing in the bitstream description. Only a GCK5 clock, which is a shared clock for AVR and FPGA was tested and revealed its position in the bitstream. Other clock sources seem to have a straightforward position in the bitstream at address 0x50, where a column clock source is selected. It also corresponds to [47].

A.6 LUT

The LUT is shown in Fig. A.3. FPSLIC has two independent 3-bit LUTs, each having 3 address lines (X, Y and W). The LUT can be implemented cascade tree of multiplexers, as shown for the ZM input stage of the LUT, which combines signal from both LUT into one 4-bit LUT (ZM input and D output in Fig. A.3).

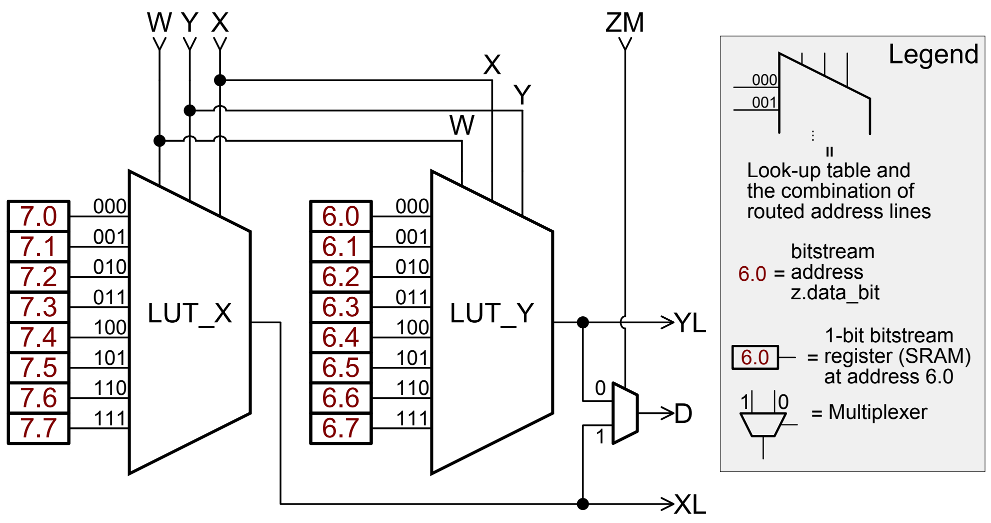
LUT input signal (X, Y, W) sequence is different for each LUT (WYX for LUT_X and WXY for LUT_Y). The correct input order was taken into account in the schematic drawing of LUTs in Fig. A.3, where the corresponding LUT bitstream bits are shown for appropriate input signal vector value.

Bitstream addresses for LUTs is Z=0x06 for LUT_Y and 0x07 for LUT_Y.

A.7 Logic Cell Inner Routing

A possible implementation of logic cell is shown in Fig. A.4. This schematic is proven to be functionally equivalent with the physical device. However there can be some buffer inserted in the signal lines, that would not change the function, but which would have an impact on fault propagation in the interconnecting network.

The picture is slightly modified from the datasheet version [16], according to observed bit meaning in the bitstream. The iW “multiplexer” was joined with the default ’1’ selection “multiplexer” in [16]. The Y and iY “multiplexers” can be joined the same way, as well as X and iX “multiplexes”. They represent a single “multiplexer” from the bitstream perspective. Names were slightly renamed in order to minimize usage of duplicitous labels.



W Y X
ZM
X
Y
W
7.0
7.1
7.2
7.3
7.4
7.5
7.6
7.7
000
001
010
011
100
101
110
111
LUT_X
6.0
6.1
6.2
6.3
6.4
6.5
6.6
6.7
LUT_Y
YL
0
1
D
XL
Legend
Look-up table and the combination of routed address lines
6.0 = bitstream address z.data_bit
6.0 = 1-bit bitstream register (SRAM) at address 6.0
= Multiplexer

Figure A.3: FPSLIC LUT explained

Z	bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0
0x0	L1↔V1	L1↔H1	L→L3	L→L2	L→L4	L→L5	L→L1	1 (?)
0x1	iZ→ZM	'0'→ZM	Z→FB	P→FB	D→P	D→C	C→oX	C→oY
0x2	L1→iZ	L1→iY	L2→iZ	L3→iZ	L4→iZ	L5→Z	V5→OE$_V$	H5→OE$_H$
0x3	L3→iW	L2→iW	L1→iW	L1→iX	L4→iW	L5→iW	H3T↔V3L	H2B↔V2R
0x4	*N*→Y	*W*→Y	*S*→Y	*E*→Y	L5→iY	L4→iY	L3→iY	L2→iY
0x5	*SW*→X	*NE*→X	*SE*→X	*NW*→X	L5→iX	L4→iX	L3→iX	L2→iX
0x6	LUT_Y (address bits 2,1 and 0: W, X, Y)							
0x7	LUT_X (address bits 2,1 and 0: W, Y, X)							
0x8	V2↔L2	H2↔L2	H3↔L3	V3↔L3	V4↔L4	H4↔L4	V5↔L5	H5↔L5
0x9	H4T↔V4L	H5T↔V5L	H5B↔V5R	H1T↔V1L	H1B↔V1R	H4B↔V4R	H2T↔V2L	H3B↔V3R

Table A.2: Bitstream table for cell and cell connection. Z=0x0..0x09. Direction shortcuts are emphasized (*N*,*E*,*S*,*W*,*NE*,*SE*,*SW*,*NW*) in order to distinguish them from net labels (*W* vs. W). Grayed entries were not verified, but just cited from [47].

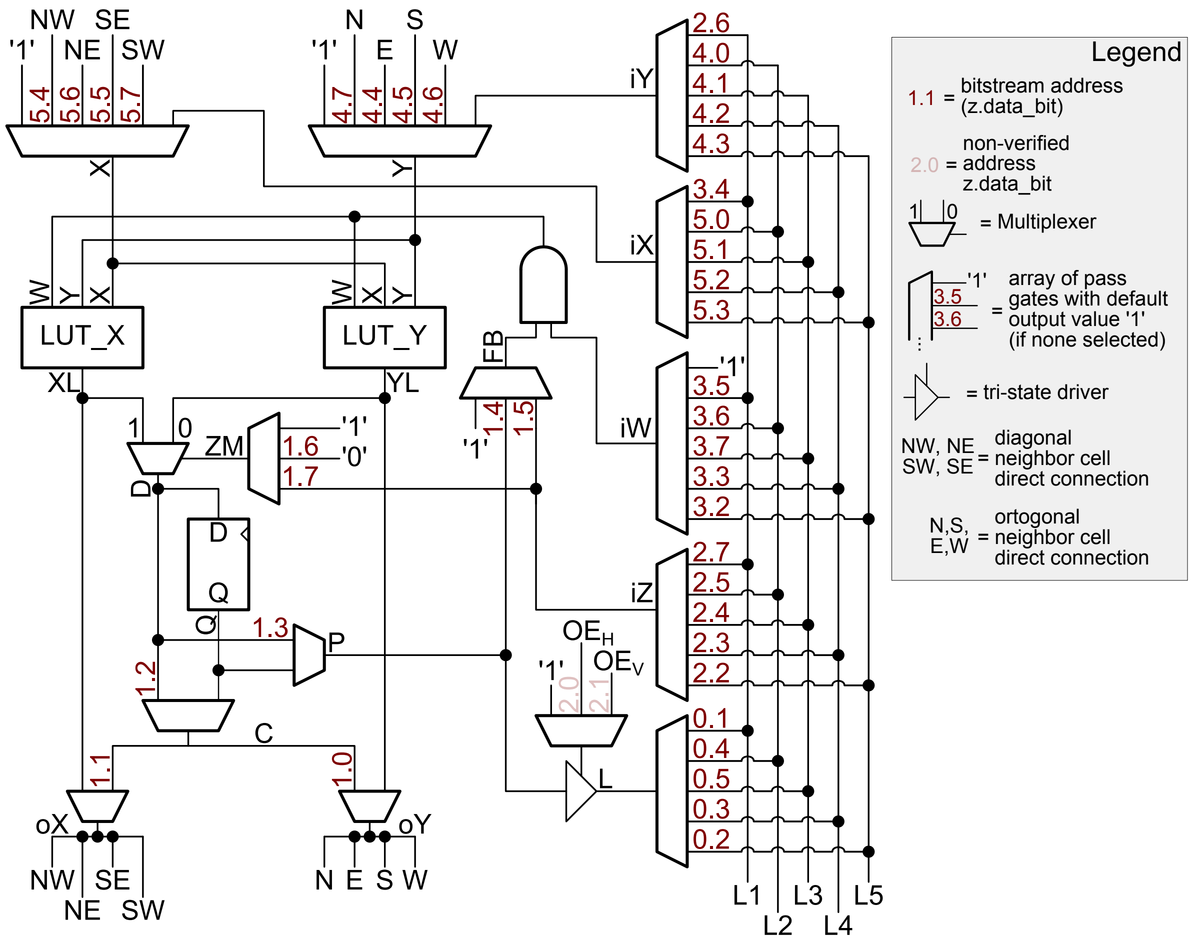


Figure A.4: FPSLIC cell inner structure

The “multiplexer” are hot-one encoded and are probably implemented as an array of pass gates. Some “Multiplexers” have an input with constant ’1’, which is not selected by any bit of the bitstream. These inputs are selected by default, when none of the other inputs are selected.

Overview of labels, that are used both in Fig. A.4 and Tab. A.2:

N, E, S, W indicates the direction, from which the cell receives direct signal (shown at Fig) a to which direction is the signal sent. North, East, South and West directions correspond to “Y” input and output.

NW, NE, SE, SW same as above, corresponds to X input and output

X, Y, W LUT inputs at the point they enter the LUT

iX, iY, iW, iZ product nets of the local bus selection “multiplexers”

XL ,YL LUT product of the X LUT and Y LUT

Z	bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0
0x10	1 (?)	1 (?)	1 (?)	1 (?)	1 (?)	1 (?)	1 (?)	1 (?)
0x11	1	1	1	1	$\overline{\text{set}}$/reset	1	1	1

Table A.3: Bitstream table for cell and cell connection. Z=0x10..0x11. Grayed entries were not directly tested by author, but cited from [47].

L1..L5 Local bidirectional wires, that is used to connect the LUT input and cell product output to the horizontal or vertical local bus

ZM Wire, that drives the Z multiplexer (product is D). It can be connected to a signal, or be a constant

D, Q Product of LUTs and Z multiplexer. D is an asynchronous signal and can either pass, or enter the Register (generating the Q output)

C,P Products of multiplexer, selecting asynchronous or registered version.

FB “feedback” line, entering the AND gate together with iW

L Output of the buffer

’0’, ’1’ logical constants

The clock and set/reset input are not shown at the Fig.A.4. The tri-state buffer is shown, but was not verified. Only clue to appropriate bit address is given by [47, 31].

Cell inner routing corresponding bitstream addresses are: Z=0x08, 0x09, and partly 0x00, 0x02, 0x03.

A.8 BUS to Cell Connection, BUS Turn Points

Each cell is placed at the point of bus crossing. At this crossing point, a vertical express bus can be connected to a horizontal one. A cell can be connected to a local bus at this crossing.

A horizontal local bus can be also connected to the vertical, but Cell’s wires L1..L5 are blocked for this connection and the connection requires both L1..L5 horizontal and vertical pass gates to be programmed. This consume 2 pass gates for this connection. This double pass gate local bus connection is shown as a single switch point at the Figaro IDS.

Figaro IDS show the cell connection in a different way, which provide direct visual information about the inner cell configuration (which input is connected to the L1..L5 net). That view displays more switch points, than exist physically on the FPGA.

Overview of labels, that are used both in Fig. A.5 and Tab. A.2:

L1..L5 Cell internal bidirectional wires (details in section A.7)

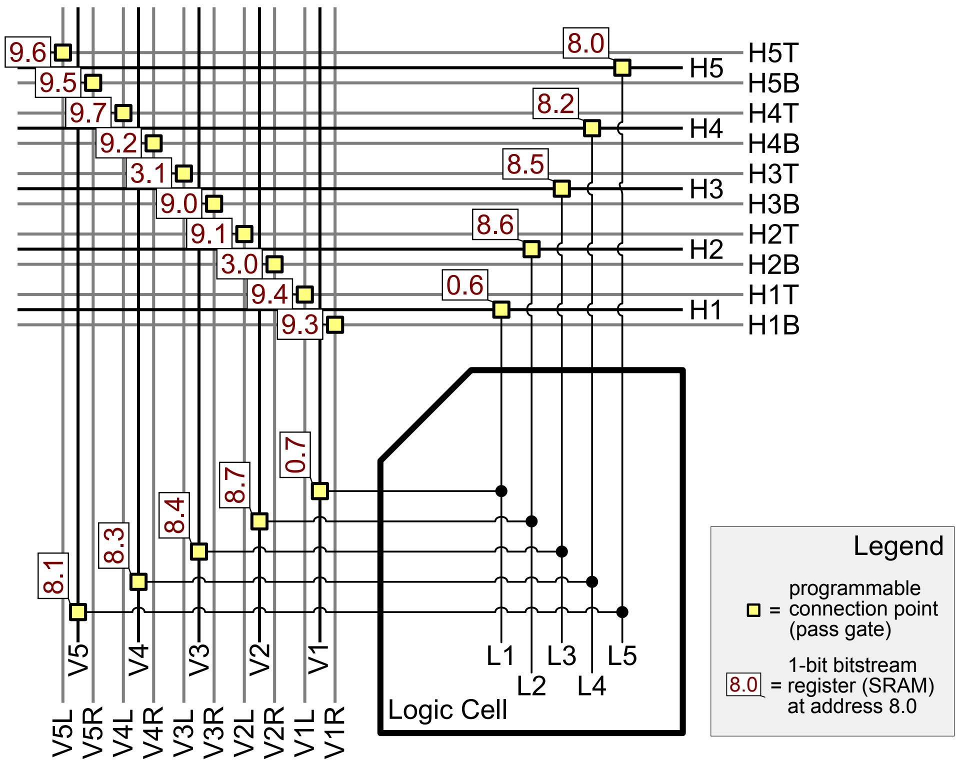


Figure A.5: FPSLIC logic cell to local bus. Express bus turn points

V1..V5 Vertical local buses

V1L..V5L Express buses left to V1..V5

V1R..V5R Express buses right to V1..V5

H1..H5 Horizontal local buses

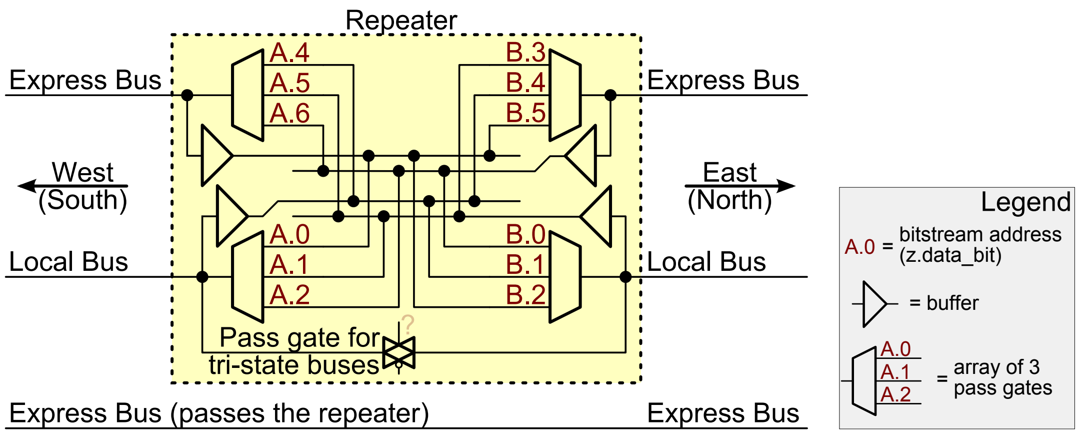
H1T..H5T Express buses above the H1..H5

H1B..H5B Express buses below H1..H5

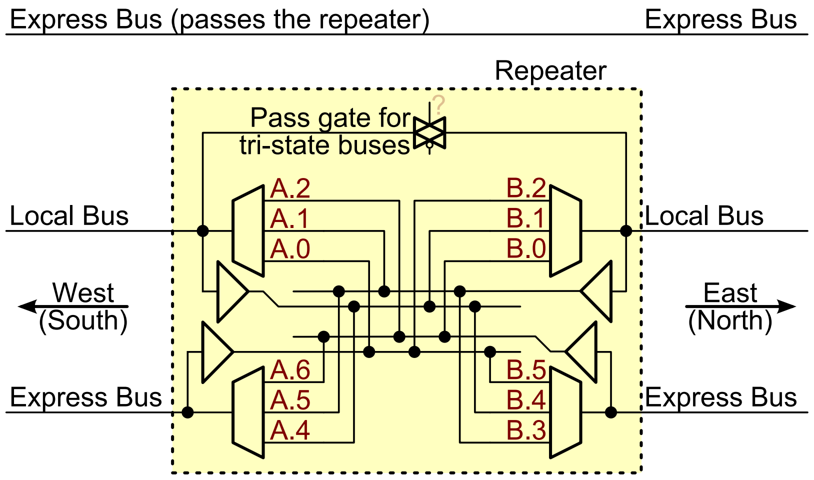
All switches in Fig. A.5 seem to be a passive bi-directional pass gates without buffers. These pass gates require only 1 bit of bitstream for connection, which is exactly FPSLIC case.

There are Output enable signals for the cell tri-state buffer that can be connected to V5 or H5. This connection was not verified and is not shown in Fig. A.5.

Bus-to-cell connection corresponding bitstream addresses are: Z=0x08, 0x09 and partly 0x00, 0x02, 0x03.



(a) Repeater at even position (bottom express bus passes the repeater)



(b) Repeater at odd position (upper express bus passes the repeater)

	A	B		A	B
V1	0x20	0x21	H1	0x30	0x31
V2	0x22	0x23	H2	0x32	0x33
V3	0x24	0x25	H3	0x34	0x35
V4	0x26	0x27	H4	0x36	0x37
V5	0x28	0x29	H5	0x38	0x39

(c) Hexadecimal addresses of vertical (V1-V5) and horizontal (H1-H5) repeater.

Figure A.6: Implementation of FPSLIC repeaters. West↔East orientation for horizontal repeaters. South↔North orientation for vertical repeaters. A and B are the Z bitstream address. $B = A + 1$.

Z	bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0
0x20	1 (?)	E1S→E1N	L1S→E1N	L1N→E1N	0 (?)	E1S→L1N	L1S→L1N	E1N→L1N
0x21	0 (?)	V5R→CR	E1N→E1S	L1N→E1S	L1S→E1S	E1N→L1S	L1N→L1S	E1S→L1S
0x22	1 (?)	E2S→E2N	L2S→E2N	L2N→E2N	0 (?)	E2S→L2N	L2S→L2N	E2N→L2N
0x23	1 (?)	1 (?)	E2N→E2S	L2N→E2S	L2S→E2S	E2N→L2S	L2N→L2S	E2S→L2S
0x24	1 (?)	E3S→E3N	L3S→E3N	L3N→E3N	0 (?)	E3S→L3N	L3S→L3N	E3N→L3N
0x25	CC→SC+	V4R→SC	E3N→E3S	L3N→E3S	L3S→E3S	E3N→L3S	L3N→L3S	E3S→L3S
0x26	1 (?)	E4S→E4N	L4S→E4N	L4N→E4N	0 (?)	E4S→L4N	L4S→L4N	E4N→L4N
0x27	1 (?)	1 (?)	E4N→E4S	L4N→E4S	L4S→E4S	E4N→L4S	L4N→L4S	E4S→L4S
0x28	1 (?)	E5S→E5N	L5S→E5N	L5N→E5N	0 (?)	E5S→L5N	L5S→L5N	E5N→L5N
0x29	SC=$\overline{\text{SC}}$	CC→SC	E5N→E5S	L5N→E5S	L5S→E5S	E5N→L5S	L5N→L5S	E5S→L5S

Table A.4: Bitstream table for vertical repeaters and sector clock. Z=0x20..0x29. Label structure is [EL][1-5][NS], where E is Express bus, L is Local bus, 1..5 is a bus index (bus plane), N is North and S is South. Grayed entries were not directly tested by author, but cited from [47]. CC=Column Clock; CR=Column Reset SC=Sector Clock; SC+=Sector Clock of the sector below coordinates.

A.9 Line Drivers/Repeaters

Repeaters are located in the bus every 4 cells in both vertical and horizontal direction. The bitstream reflects this fact in a new coordinates numbers, which are described at section A.4 and shown in Fig. A.2 .

The position of the repeater differs for even and odd columns/rows. The repeater position is shown in Fig. A.2 and A.6. The bitstream address does not distinguish between odd and even position, which makes the labeling a little confusing – it has to fit both even and odd position. Therefor a different labeling scheme has to be introduced to the same nets, which were already labeled in previous section:

[EL][1-5][NESW] Bus label structure

- **L**** Local bus
- **E**** Express bus
- ***1*..*5*** Bus index (bus "plane")
- ****S** Southern side of the repeater (Vertical Repeater)
- ****N** Northern side of the repeater (Vertical Repeater)
- ****E** Eastern side of the repeater (Horizontal Repeater)
- ****W** Western side of the repeater (Horizontal Repeater)

V4R (V5R) 4th (5th) right vertical express bus, 4th (5th) column, right side.

Z	bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0
0x30	1 (?)	E1E→E1W	L1E→E1W	L1W→E1W	0 (?)	E1E→L1W	L1E→L1W	E1W→L1W
0x31	0 (?)	0 (?)	E1W→E1E	L1W→E1E	L1E→E1E	E1W→L1E	L1W→L1E	E1E→L1E
0x32	1 (?)	E2E→E2W	L2E→E2W	L2W→E2W	0 (?)	E2E→L2W	L2E→L2W	E2W→L2W
0x33	1 (?)	1 (?)	E2W→E2E	L2W→E2E	L2E→E2E	E2W→L2E	L2W→L2E	E2E→L2E
0x34	1 (?)	E3E→E3W	L3E→E3W	L3W→E3W	0 (?)	E3E→L3W	L3E→L3W	E3W→L3W
0x35	0 (?)	0 (?)	E3W→E3E	L3W→E3E	L3E→E3E	E3W→L3E	L3W→L3E	E3E→L3E
0x36	1 (?)	E4E→E4W	L4E→E4W	L4W→E4W	0 (?)	E4E→L4W	L4E→L4W	E4W→L4W
0x37	1 (?)	1 (?)	E4W→E4E	L4W→E4E	L4E→E4E	E4W→L4E	L4W→L4E	E4E→L4E
0x38	1 (?)	E5E→E5W	L5E→E5W	L5W→E5W	0 (?)	E5E→L5W	L5E→L5W	E5W→L5W
0x39	0 (?)	0 (?)	E5W→E5E	L5W→E5E	L5E→E5E	E5W→L5E	L5W→L5E	E5E→L5E

Table A.5: Bitstream table for horizontal repeaters. Z=0x30..0x39. Label structure is [EL][1-5][WE], where E is Express bus, L is Local bus, 1..5 is a bus index (bus plane), W is West and E is East.

CC Column Clock

CR Column Reset

SC Sector clock

SC+ Sector clock of the sector below the coordinates

At the FPGA edges, all bus repeater output are routed to a shared 2 buses (**EEB** – Edge Express Bus and **ELB** – Edge Local Bus, instead of former Express buses E1*..E5* and Local buses L1*..L5*), that are shared for all 5 repeater at the corner side.

This allows the signal to be routed to a different bus indexes (routing planes). For example, a horizontal repeaters at coordinate (1,0) has west outputs L1W..L5W and E1W..E5W, as shown in Fig. A.6b. The horizontal repeater at the west corner (coordinate (0,0)), does not have these independent 5 express and 5 local buses, but a single local bus output (ELB) instead of L1W..L5W and a single express bus (EEB) instead of E1W..E5W, which are shared among all 5 repeaters.

According, the documentation [16, 15] the repeater allows tri-state bus to pass the repeater. There has to be therefore a pass gate inside the repeater. The pass gate is drawn in Fig. A.6, but the bit position is not known. A natural candidate is the bit 3 at addresses 0x20, 0x22, 0x24, 0x26, 0x28, 0x30, 0x32, 0x34, 0x36 and 0x38, which contains '0' consistently at the same position for all repeaters.

The repeater architecture is shown in Fig. A.6. It is one of the possible implementations. This one, which is a valid repeater architecture [40], seems to be closest to the bitstream structure. The bitstream does not use any bit of bitstream for the buffer enable, which however can be generated as an OR product of 3 source select (not output select in

Z	X	Y	bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0
0x40	0..11 (even)	0..11	1 (?)	1 (?)	1 (?)	1 (?)	USECLK	DUAL	$\overline{\text{DUAL}}$	ENABLE
0x40	0..11 (odd)	0..11	1 (?)	1 (?)	1 (?)	1 (?)	1 (?)	1 (?)	1 (?)	1 (?)
0x41	0..11 (even)	0..11	1 (?)	1 (?)	1 (?)	1 (?)	1 (?)	1 (?)	1 (?)	1 (?)
0x41	0..11 (odd)	0..11	1 (?)	1 (?)	1 (?)	1 (?)	USECLK	$\overline{\text{ENABLE}}$	ENABLE	ENABLE

Table A.6: Bitstream table for RAM logic. Z=0x40,0x41. Not tested by the author at all. Grayed entries were not directly tested by author, but cited from [47]

Z	X	Y	bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0
0x50	0..47	0	GCK8	GCK7	GCK6	GCK5	GCK4	GCK3	GCK2	GCK1
0x51	0,23,47	0	1?	1?	1?	1?	1?	1?	GCK SRC?	
0x51	1..22,24..46	0	1 (?)	1 (?)	1 (?)	1 (?)	1 (?)	1 (?)	1 (?)	1 (?)
0xD0	0	0	1 (?)	1 (?)	0 (?)	0 (?)	0 (?)	0 (?)	0 (?)	0 (?)
0xD1	0	0	0 (?)	0 (?)	0 (?)	0 (?)	0 (?)	0 (?)	0 (?)	0 (?)
0xD2	0	0	0 (?)	0 (?)	0 (?)	0 (?)	0 (?)	0 (?)	0 (?)	0 (?)
0xD3	0	0	0 (?)	0 (?)	0 (?)	0 (?)	1 (?)	1 (?)	0 (?)	1 (?)

Table A.7: Bitstream table for clock source and other unknown Z=0x50,0x51, 0xD0-0xD3. Grayed entries were not directly tested by author, but cited from [47]

suggested architecture) on the chip. The architecture has a fundamental impact on fault propagation.

A.10 Unexplored

There are many bits in the bitstream, which were not explored in this work. Mainly because they control resources, that have nothing to do with combinational circuits:

Distributed FreeRAM is not used by any benchmark, but connects to resources, that can be occupied by benchmarks. Most critical bits are the ones responsible for memory data out. The data out signal connects to V4 local buses. Bitstream taken from [47] is shown in Table A.6.

Clock & reset were not used by any benchmark and a single bit should not have any effect on the benchmarks, but would have impact on a synchronous design. Bits for these resources were reported at Z=0x11, 0x25, 0x29, 0x50 and 0x51, which are shown in Table A.7, A.3, A.4 and A.7.

I/Os can have effect on bigger benchmarks (especially on hard-to-route benchmarks, which consume a lot of routing nets), that use routing nets outside the corner re-

Z	bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0
Primary cell:								
0x60 (0x70)	SCHMITT	SLEW		$\overline{G2} \to CR$	0	PULL-up/down		0
0x61 (0x71)	$REG \to OUT$	0	OE	Output Mux				
0x62 (0x72)	Added delay				PRI→S-	PRI→G+	PRI→G	PRI→S
0x63 (0x73)	OEM	USEOEM	OEM					
Secondary cell:								
0x64 (0x74)	SCHMITT	SLEW		$\overline{G2} \to CR$	0	PULL		0
0x65 (0x75)	$REG \to OUT$	0	OE	Output Mux				
0x66 (0x76)	Added delay				SND→S-	SND→G+	PRI→G	SND→S
0x67 (0x77)	OEM	USEOEM	OEM					

Table A.8: Bitstream table for I/Os according to [47]. Addresses 0x60..0x67 correspond to East I/O column (x=1) and West I/O column (x=0). Addresses 0x70..0x77 correspond to North I/O row (y=1) and South I/O row (y=0). Not tested by the author. Legend: S = Sector wires of this cell; S+ = Sector wires of next cell; S- = Sector wires of previous cell; G = Global wires of this cell; G+ = Global wires of next cell; Output = Allow output from this IOB; OE = when low, output is always enabled; OEM = 7 bits, one-hot encoded, chooses input to output-enable mux; USEOEM = when low, ignore the output enable mux; Delay = amount of delay to add; can be 0, 1, 3, or 5; Slew = slew time: 11=fast, 10=med, 01=slow; Pull = 00=pullup, 11=pulldown, 01=none.

peater. These routing resources connect directly to I/O are therefore sensitive to get driven by the I/O accidentally.

There are many bits that have unknown bitstream SRAM mapping – if they have any. For example bit 0 at address z=0x0, many "constant 1" bits at repeaters, etc. This does not mean that they can't be tested. On the contrary, it is very easy to test them. They are marked as "UNEXPLORED" bit class.

Other bits were not explored, since they were not used for the purpose of this work.

A.11 Discussion on Description Differences

The bitstream descriptions (my and [47]) differ in the net labels, which were used:

- Bus index has completely different numbering (here $\leftrightarrow$ [47])
  - V1, L1, H1 $\leftrightarrow$ V4, L4, H4
  - V2, L2, H2 $\leftrightarrow$ V3, L3, H3
  - V3, L3, H3 $\leftrightarrow$ V2, L2, H2
  - V4, L4, H4 $\leftrightarrow$ V1, L1, H1
  - V5, L5, H5 $\leftrightarrow$ V0, L0, H0
- Express buses (at the crossing point) differ in numbering explained above, shorten here as a transformation $i \leftrightarrow (5 - i)$. The express line designation, which distinguish among them:
  - HiT $\leftrightarrow$ H$(5 - i)$a, where i is the net index, $i \in \{0, 1, 2, 3, 4\}$
  - ViL $\leftrightarrow$ V$(5 - i)$a
  - HiB $\leftrightarrow$ H$(5 - i)$b
  - ViR $\leftrightarrow$ V$(5 - i)$b
- Output of the generated function (L, P $\leftrightarrow$ FB), which is only a minor, functionally almost equivalent difference. FB is called the product of the FB multiplexer in this work.

More important, there are differences in the bitstream position:

- LUT_X – the location was found at Z=0x07, rather than 0x06 as in [47, 31].
- LUT_Y – the location was found at Z=0x06, rather than 0x07.
- Order of LUT inputs (address lines) is different for each LUT (WYX for LUT_X, WXY for LUT_Y, as shown in Fig. A.3).

- The P product is selected by bit 1.3, rather than 1.2 as in [47].
- The C product is selected by bit 1.2, rather than 1.3 as in [47].
- The L5→iW connection is controlled by bit 3.2, rather than 3.3 as in [47].
- The L4→iW connection is controlled by bit 3.3, rather than 3.2 as in [47].
- The South neighbor input Y is selected by bit 4.5, rather than 4.4 as in [47].
- The West neighbor input Y is selected by bit 4.5, rather than 4.4 as in [47].
- The H5↔L5 connection is controlled by bit 8.0 rather than 8.2 as in [47].
- The H4↔L4 connection is controlled by bit 8.2 rather than 8.0 as in [47].
- The repeater bits I found completely scrambled and even 3-bit block (0x20.4-0x20.6) are shifted left by 1 bit (from the position suggested by [47]) from (0x20.3-0x20.5) position for all even repeater addresses (0x20, 0x22, 0x24, 0x26, 0x28, 0x30, 0x32, 0x34, 0x36, 0x38). The shift creates a gap of "0"s at position 0x20.3 and further (0x22.3, etc.).

Summary of Bitstream

As stated before, this bitstream description contains all necessary bits for purpose of SEU testing of combinational circuits, which does not require a complete description.

To know more is always better: The more accurate is the bitstream knowledge, the more accurate prediction can be evaluated for the bit change effect. The SEU emulator however does not require this knowledge at all – faults can be injected completely blindly, as long as the emulated SEU does not interfere with the control logic.

The route to complete bitstream understanding is open and clear and it is only a matter of little manpower to get there. The structure is known (drawings are provided here and documented) and the analysis is straightforward, as shown at the Bitstream analysis (Chapter B). It waits to somebody who will need the complete bitstream understanding.

Eventhough the reverse engineering was performed twice in order to cross-check the bitstream description, an unwanted mistake can be still included in the description.

Appendix B

Bitstream Analysis

This section describes general procedures, how can be the FPGA architecture analyzed using non-invasive methods of analysis, based on the bitstream analysis. Aim of the procedures is a hand-made analysis, rather than an automated analysis. Subjects of the analysis are bitstreams of several suitably chosen designs. Product of this analysis is a functionally equivalent model of the FPGA architecture and a bit mapping of the bitstream and FPGA's programmable resources. Example of the method application is given for the Atmel's AT90K40AL FPGA.

As a side result of this analysis, any bitstream for that given FPGA can be transformed back into the design netlist (or HDL language).

The complete bit understanding of the bitstream is usually considered to be confidential information by the FPGA vendors. The end user does not need to know the bits, since the bitstream is completely generated by a software provided either by the FPGA vendors or a third parties.

The bitstream knowledge is kept secret for the FPGA design security reason. The known FPGA and bitstream structures can be used for an attack on the bitstream encryption [53] with a many security consequence with practical impact, such as compromising of an encrypted design, malicious design modification, etc.

The bitstream analysis is a complex task, where many sub-tasks have to be explored together:

FPGA architecture The architecture is usually very well described from the manufacturer. The Place&route tools from the design flow software suite also often provides detailed insight into the FPGA structure and into the actual resource utilization by the mapped design. The architecture understanding can be from the functionally equivalent level up to the detailed model with timing information.

FPGA configuration logic The FPGA logic, that controls the loading of the bitstream is usually very well described. Encryption represents a biggest problem to the analysis, if can't be turned off.

Bitstream architecture The bitstream in the binary form consist of headers, commands and packets with add address and data. This architecture can be documented by the FPGA vendor, for example Xilinx [71].

FPGA frame granularity The FPGA resources are usually grouped into bigger frames containing several resources (mostly very close together), that are organized into frames. These frames are in bitstream represented as a single configuration block of data. The granularity can be hierarchical; For example many similar CLB inside the bigger frame.

Addressing The addressing of the frames often reflect its topology (x and y coordinates). The addressing is often different for different resources (I/O, RAM, switching matrices and CLBs).

B.1 Bitstream modes

The FPGA can load the bitstream itself in a master mode, or it can be loaded externally by a microcontroller in a slave mode. In the Master mode, the bitstream contains all necessary initial sequences, headers and commands. In the Slave mode, the commands and data frame creation can be operated by the microcontroller. The bitstream can be similar for both modes.

The analyzed device, AT94K40AL, provides a special slave mode (MD4), which does not group the bitstream into frames, but uses a unique address for each byte of the bitstream data. The "address" effectively reveals both the frames and addressing by its format and therefore provides a very useful hint for the *FPGA frame granularity*, *Bitstream architecture* and *Addressing* tasks.

B.2 Precision of the analysis

The goal of the bitstream analysis can have a different level:

1. Obtaining a functionally equivalent FPGA model.
2. Obtaining a detailed FPGA structure with correct allocation of buffers and pass gates.
3. Obtaining a detailed timing of the FPGA structure.

At the first stage of analysis, looking for any functionally equivalent implementation is sufficient. Once a working bitstream description and FPGA model is found, the analysis can be further iterated, in order to improve the precision of structure knowledge.

Analysis for the reliability analysis purpose requires a very high precision of the bitstream knowledge, especially locations of buffers, which affect the fault back propagation and buffer locations. Buffers have no influence on the function fault-free functionally equivalent FPGA mode (except for timing), but they do affect the fault back-propagation, as shown in Fig. 4.8.

The delay model extraction method was not within the scope of this work. All benchmarks were tested safe below the maximal operating frequency. The easiest way of delay model extraction would be directly from the EDA tool. Time model applications would aim toward testing of the own Place & route algorithms or designing an asynchronous logic.

B.3 Analysis counter-measurements

This section summarizes the obstacles for the analysis. Every non-regularity makes the analysis more time consuming. Here are some observations how simple counter-measurements would make the presented analysis difficult:

- Bitstream framing (sectioning) - not to use at all. This would extend the bitstream analysis from a single frame to the whole chip.
- Programmable switch on-state representation. Switch on-state that is always represented by '1' in the bitstream and '0' for off-state is easy to analyze.
- Bitstream encoding. The reverse engineering would be focused on catching the data from the software flow tool chain.
- Mask the topology information from the bitstream frame addresses.
- Resources input scrambling. The more irregularities exist, the more the analysis requires manual interventions.
- In-Frame bit scrubbing, which is individual for each frame.
- Disallowing the floorplan view of the design or disallowing the insight into the CLB in the EDA tools.
- Transformation of the design view (from the design flow) into an equivalent structure that does not reflects the real one.

B.4 Tools and Methods of analysis

This section summarizes the methods, how the bitstream was analyzed. Described procedures describe some methods, which assume that:

- The bitstream is not encoded,
- The bitstream is regular,

- The bitstream addressing reflects the chip topology.

Following list summarizes methods that are useful in the analysis process:

FPGA architecture survey The FPGA architecture is an intensively studied research field, which generates a lot of publications and books. The literature provides many suggestions for the possible logic implementations. It is therefore a valuable source of the guesses, which type of programmable pass gates and buffers is used at the given foundry process.

Design Tools and Design Tools files The Vendor's Design Tools can provide an architecture insight. The design tools files save the work design into a file, which, if not encrypted, may be analyzed and may contain valuable information about names and addressing within the file format, which is not shown at the Design Tools view.

Visualization The visualization gives an easy overview at many stages of analysis. It can show, which bit are "used" (set to '1') and this can be cross-checked with EDA view of the design. Details in section B.5.

Specific designs Simplest designs that are moved all over the FPGA can reveal much of its structure and bitstream representation. The design can be simple like 4-input AND or 4-input OR. The less resources utilized, the better. Bigger designs are useful for verification of the bitstream description.

Bitstream compare When very similar designs are played with, bitwise bitstream compare can limit the bitstream to a limited number of bits, which can be responsible for utilized resources.

Set operations and selection Set operations at the bitstream frame level is useful at the verification phase, where there is a need for select bitstream frames with a set of resources in use or without being used by other resources.

Timing characteristic The delays (either from datasheet or extracted from EDA) can reveal information about the structure, because each element adds a delay to the signal path.

All those methods do not actually require the physical device in the hand. Analysis can be done completely in the software. However, the verification phase of the buffer placement analysis (the problem presented in Fig. 4.8) does require the physical device to be analyzed.

B.5 Visualization

A very simple method for discovering the addressing and granularity in the first approach to the bitstream understanding is to create a 2D image of the bitstream. The image for the *mode 4* (MD4) bitstream format of the analyzed FPGA is unnecessary, because the

MD4 address contains all the necessary information for decoding its addressing (the x and y coordinates and the "function" part of the address).

The bitstream structure of the "mode 0" does not embed topological address for each byte of bitstream. For such bitstream, an image can be created by an algorithm B.1.

Algorithm B.1 Bitstream analysing by the image observation

```
input    : A bitstream file bst of size bSize
output   : A width of the image width
optional : An offset optionally scrools the image horizontally.
output   : A 2D bitmap bmp of size $width \times \lceil bSize/width \rceil$

width ← 1;
repeat                                                      //iterate the image width
    width ← width + 1;
    create an empty bmp [width ][$\lceil bSize/width \rceil$];
    choose offset ∈ $\langle 0, width)$;                    //optional, otherwise offset ← 0
    for $i$ ← offset to bSize do                            //fill the image
        bmp[$i$ mod width][$i \div$ width] ← Color(bst[$i$ mod width]);
    end
    Show bmp;
until *A new pattern is seen* or (width > bSize/2);
```

The `Color` function in the Algorithm B.1 defines the coloring of the bitmap. It depends on the bit granularity, which has to be displayed. For a bit granularity, a black and white mapping is natural. For a byte (8 bits) display granularity (which were used in this example), color based on the count of ones in bitstream byte (range (0,8) expanded to a gray scale is shown in Fig. B.1). It can be combined with color variation according to a specific bit (as shown in Fig. B.13)

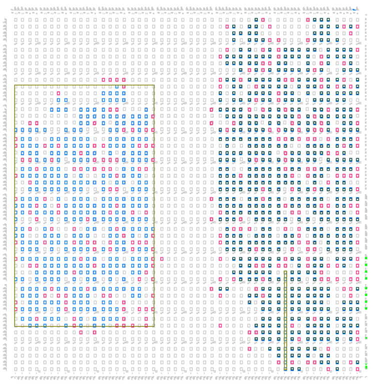
The resulting image pattern provides a guideline about the bitstream organization. Images for the *mode 0* bitstream of medium sized, unevenly-shaped design are shown in Fig. B.1

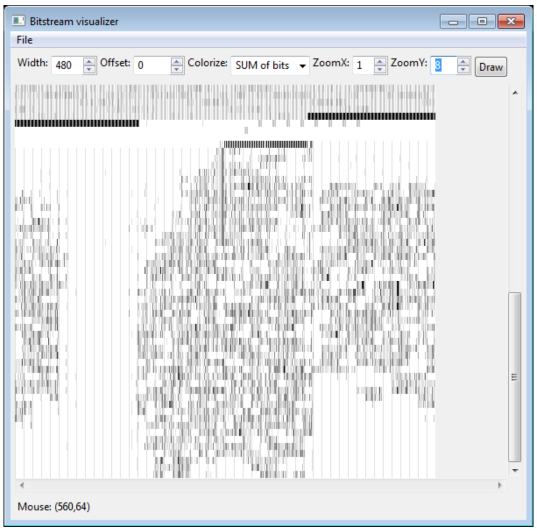
B.6 Finding out the regularities

Visualization is an effective method for discovering the *bitstream addressing* and *bitstream granularity* via observing of the regularities in bitstream. Visual analysis is used extensively for many tasks of the bitstream analysis.

This section is divided in two bitstream formats, *mode 0* and *mode 4*, since each bitstream mode needs different approach. Mode 0 is byte oriented stream, while mode 4 contains an address for each byte of the bitstream.

Analysis of *mode 0* bitstream provides a guideline to a generic bitstream analysis. *Mode 4* analysis is Atmel specific and the analysis can be hardly transferred to an generic bitstream analysis method.


(a) EDA design overview

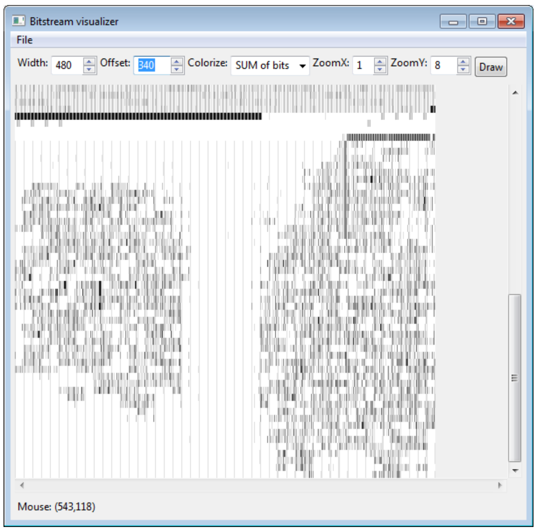

Bitstream visualizer

File

Width: 480 Offset: 0 Colorize: SUM of bits ZoomX: 1 ZoomY: 8 Draw

Mouse: (560,64)

(b) Logic cells. Width = 480; Offset = 0

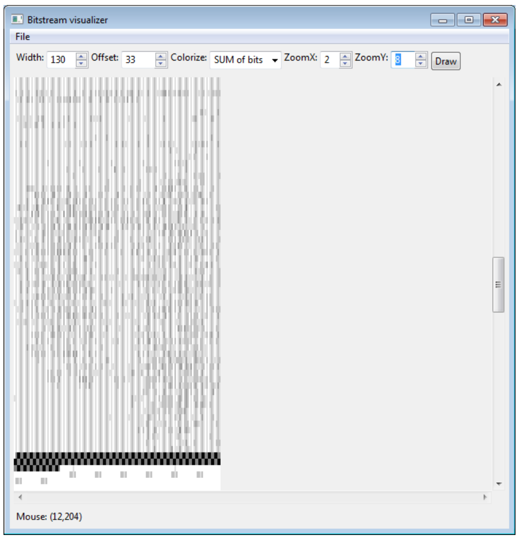

Bitstream visualizer

File

Width: 480 Offset: 340 Colorize: SUM of bits ZoomX: 1 ZoomY: 8 Draw

Mouse: (543,118)

(c) Logic cells shifted. Width = 480; Offset = 340

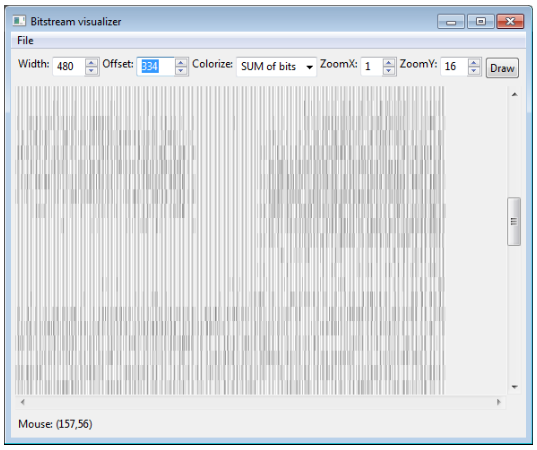

Bitstream visualizer

File

Width: 130 Offset: 33 Colorize: SUM of bits ZoomX: 2 ZoomY: 8 Draw

Mouse: (12,204)

(d) Horizontal repeaters. Width = 130


Bitstream visualizer

File

Width: 480 Offset: 334 Colorize: SUM of bits ZoomX: 1 ZoomY: 16 Draw

Mouse: (157,56)

(e) Vertical repeaters. Width = 480

Figure B.1: Initial hunt for the *mode 0* bitstream frames and addressing. Note the ZoomX and ZommY values, that stretch the image for easier pattern recognition.

B.6.1 Mode 0 bitstream visual analysis

Based on the knowledge of the FPGA topology from the datasheet (48×48 logic cells, 5 horizontal/vertical repeaters in each row/column every 4 cells, etc.), following information can be obtained from the images:

- How the *logic cells* are stored in the bitstream:
  - Logic Cells **are streamed sequentially** from the bottom left cell (0,0) to the bottom right cell (47,0). Rows are streamed from the bottom row (0) to the top row (47). This was identified by a comparison of patterns from Figures B.1a and B.1c, which has a visible 14-bit counter (right bottom part)
  - **10 bytes per logic** cell is required (derived from 480 bytes per one row of 48 cells)
- How the *repeaters* are stored in the bitstream. Main identification key for the repeaters was their count (48×13 or 13×48). 13 is a prime number, which makes the search for repeaters easy.
  - 13 rows of *vertical repeaters*, that copy the design structure (Fig. B.1a), were identified in Fig. B.1e. Image had to be stretched in the y direction. Blocks of 5 repeaters **are streamed sequentially** in similar way as logic cells: starting from the left bottom vertical repeater (0,0) to the right bottom vertical repeater (47,0). 48 rows of vertical repeaters are streamed from the bottom row (0) to the top row (13).
  - 48 rows of *horizontal repeaters* were identified in Fig. B.1d, streamed sequentially likewise the vertical repeaters: starting from the left bottom horizontal repeaters (0,0) to the right bottom horizontal repeaters (12,0). 13 rows of horizontal repeaters are streamed from the bottom row (0) to the top row (47).
  - Each row of horizontal (vertical) repeaters has 480 (130) bytes. This suggests a theory, that **10 bytes represents 5 repeaters**, which are topologically located close together. It can potentially lead to **2 bytes per repeater**.
- Other granularities without the structure were seen:
  - granularity 2
  - One of following structures: 144×2, 96×3, 72×4, 48×6, 36×8,32×9, 24×24,18×16,16×18, ,12×24, 8×36, 6×48, 4×72 or 2×144. Which one of these organizations is valid is not always easy to see at a first look ,

Having the addressing and granularity information is essential for further structure investigation.

B.6.2 Mode 4 bitstream

At first, it is important to be able to make visual maps, that corresponds to the bitstream granularity (frames).

A search for a frame structure and regularities does not represents a problem at all for the mode 4 bitstream of AT94K40AL device, since the addressing is contained in the bitstream configuration word.

The initial granularity and addressing search is shown in Fig. B.2. The visualization of $z \in \langle 0; 9\rangle$ according to the x and y address bytes directly corresponds to the topology of the design in Fig. B.2a.

Some bitstream z addresses ($z \in \langle 0x20; 0x29\rangle$ – shown in Fig. B.2c, $z \in \langle 0x30; 0x39\rangle$ – shown in Fig. B.2d) have only a limited range ($y \in \langle 0; 12\rangle$, or $x \in \langle 0; 12\rangle$). The visual image is scaled to the limited range. That indicates, that those resources are located every 4 CLBs in x and y directions.

As a result of the first look at the *mode 4* bitstream, following was discovered:

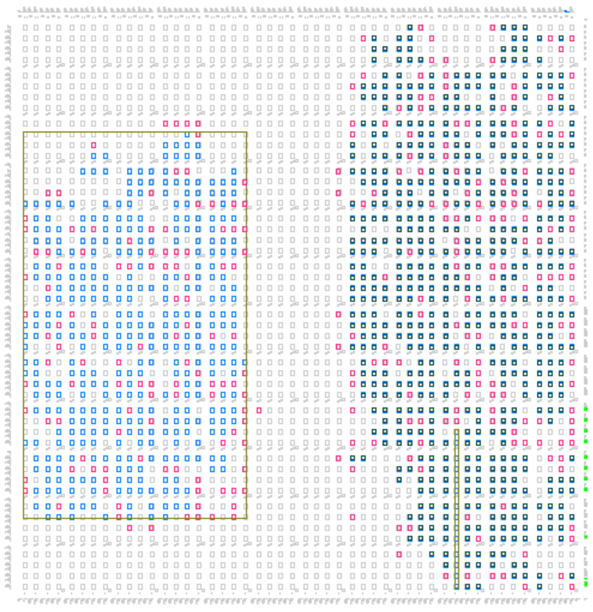
- The mapping of the Cartesian coordinate system to the address.
- Groups of addresses, that
  - Use the coordinates as a logic cells;
  - Use a *sector* coordinate in one direction (sector means the increment ever 4 logic cells). Utilization of the bitstream is still visually related to the placed design;
  - Use a *sector* coordinates in both directions. These do not provide any correlation with the placed design;
  - Use a *corner* coordinates in one direction (the coordinate is 1 or 0). These resources visually correspond to the I/O pad utilization;
  - Used only x coordinate ($y = 0$). The utilization map corresponds directly to the clock resource utilization. This was possible to see only because of the design floorplan, that used registers only in the right half of the design;
  - Some bitstream frames do not use address (neither x, nor y), or use only very few combinations.

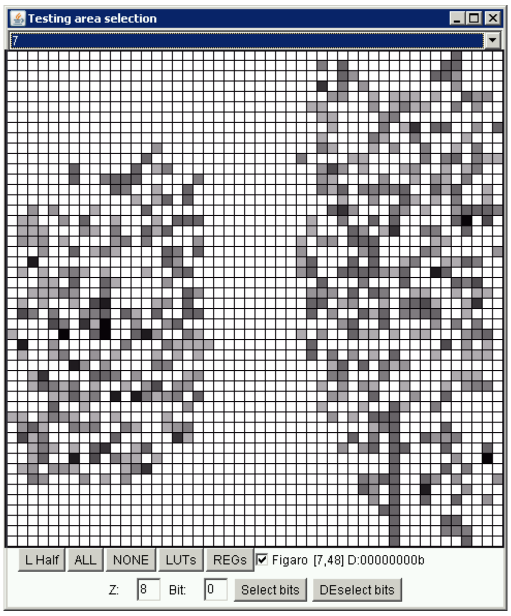
Understanding the bitstream granularity requires less effort in the *mode 4*, than in the *mode 0*. Further analysis in next sections is based on the *mode 4* bitstream format.

B.7 LUT search

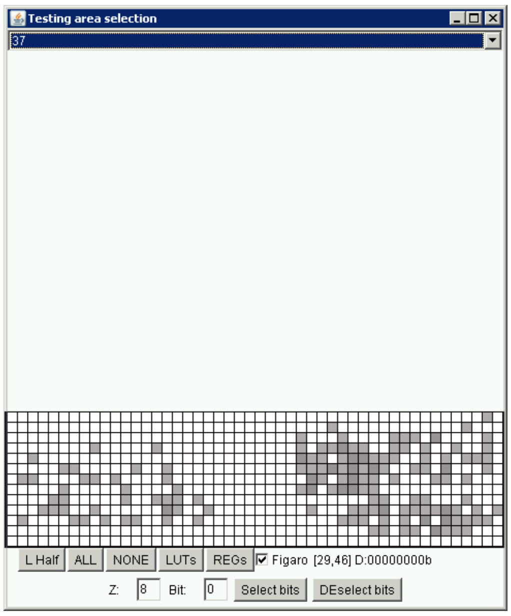
The task of the LUT bitstream revelation is the easiest one. The LUT architecture does not provide much degree of freedom in the implementation. Let's start with the assumption based on the AT94KAL datasheet information, that each of the two LUTs is represented as a truth table in the bitstream and is implemented by a 3-bit multiplexer.

The task of analysis contains and should answer:

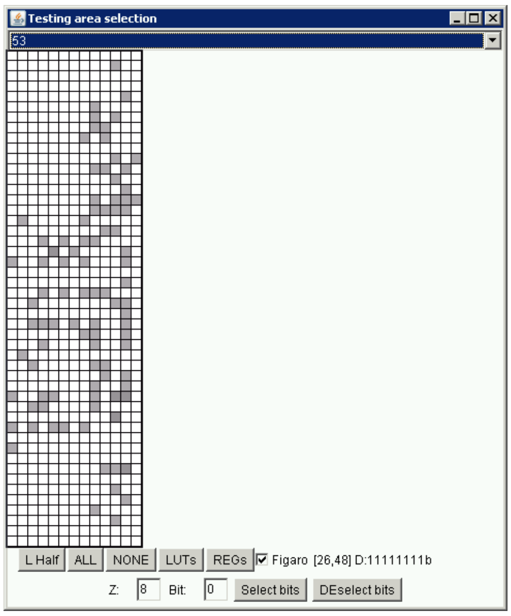

(a) EDA design overview



(b) Z = 7 map



(c) Z = 37 map



(d) Z = 53 map

Figure B.2: Initial hunt for the *mode 4* bitstream frames and addressing. All subfigures show the same design. *b*, *c* and *d* show different addresing for different frames (different Z address part). The gray level of the picture corresponds to number of bits set to '1' in the related bitstream address

- Where in the bitstream are the LUT tables stored;
- How are the LUT inputs organized;
- Advanced: what is the physical structure of the LUT (where are the transistors and how do the errors propagate).

B.7.1 Guess of structure

The **timing characteristics** may provide valuable information about the cell structure. According to the timing information provided by [16] (shown also in Table B.1 right), the X/Y→X/Y is the fastest path (when going through the Z multiplexor) and it adds 0.5 ∼ 0.6 ns when the W input (X/Y/W→X/Y) is used. The average propagation delay timing information provided by the Figaro IDS is smaller. Simple designs (4-input XOR and inverter) were used for the delay extraction. Results are shown in Table B.1.

4-input XOR				inverter				Datasheet	
path	delay	path	delay	path	delay	path	delay	path	delay
X→X	1.05	X→Y	1.08	X→X	0.99	X→Y	0.96	XY→XY	2.9
Y→X	1.09	Y→Y	1.13	Y→X	0.95	Y→Y	0.92	XYZ→XY	2.8
W→X	1.65	W→Y	1.68	W→X	1.54	W→Y	1.51	XYW→XY	3.4
Z→X	1.61	Z→Y	1.63	Z→X	1.19	Z→Y	1.16	XYWZ→XY	3.4

Table B.1: Extracted LUT timing for two different designs, compared to a datasheet values (for 1 unit load). Delays units are ns.

These timing values do not directly answer the question, how the LUT is implemented, because there is no clear delay step between X and Y inputs (at least at the first approach). It shows, that both X/Y input stages is fastest. The W input adds another 0.5 ns, as written in [16].

So let's start with assumption, that the LUT is organized as a cascade of multiplexers, where the W stage is the first stage (because its propagation delay is largest) and the X/Y inputs drive the last stage of multiplexers.

B.7.2 Search method

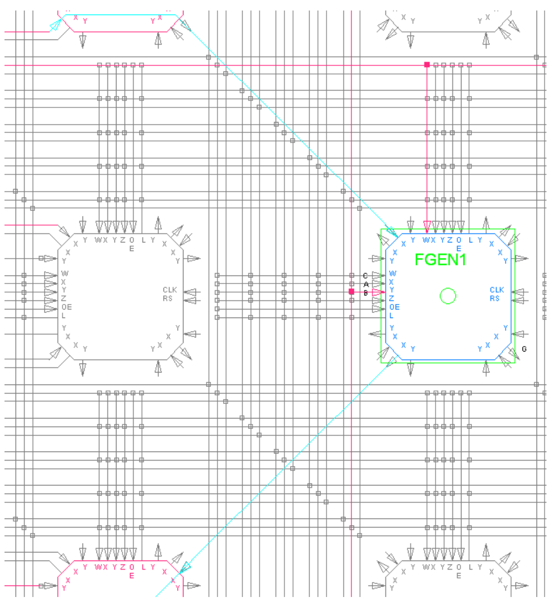
The AT94 logic cell contains two 3-bit LUTs; therefore the search is set for 16 bits in the bitstream, probably organized into two groups of 8 bits. It is expected, that those bits will be scrambled in the bitstream. The reason for scrambling is the optimization of the chip layout and metal routing.

8 test designs (LUT0 . . . LUT7) should be able to completely reveal the single LUT in the scrambled bitstream. There are 2 LUTs in the logic cell, therefore there 8 tests have to be performed for each LUT separately.

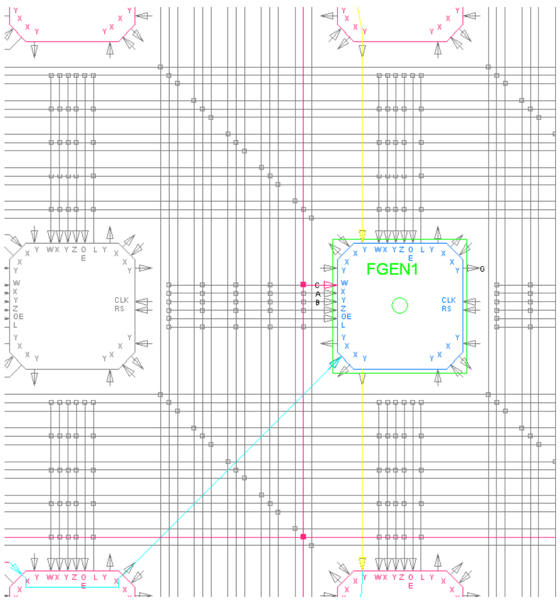
The input order was selected according to the findings in section B.7.1: The W input is a first, since it has a longest delay, than the Y and X inputs. Y and X inputs were ordered randomly without a timing clue.

The test designs LUT0...LUT7 are defined by equations B.1...B.8. They define the truth tables (Tab. B.2), which should be easily identified in the bitstream, because they are effectively encoded as "moving one".

The test designs have to be placed manually on a fixed place and force the EDA to keep the inputs unchanged (scrambling of the inputs would destroy the search) and keep the output of the logic cell. The output of the logic cell (can be mapped either to X or Y output) defines which LUT is used (LUT_X of LUT_Y). Second reason for keeping the input, output and routing the same for all 8 test designs is to minimize number of changes in the bitstream. Then only the LUT part of the bitstream will vary. Example of such a placement and routing is given in Fig. B.3.



(a) Design for testing LUT_X



(b) Design for testing LUT_Y

Figure B.3: Test design placement for the LUT analysis: The Logic cell is fixed at the position and the EDA is forced to drive the X or Y outputs from the X,Y and W inputs

Equations LUT0...LUT7 were created by the property, that each LUT{n} equation is true only at a given combination of inputs, which is the n in a binary code at the given order of W, Y and X inputs.

$$\mathrm{LUT0} = \overline{\mathrm{W}} \cdot \overline{\mathrm{Y}} \cdot \overline{\mathrm{X}} \tag{B.1}$$

$$\mathrm{LUT1} = \overline{\mathrm{W}} \cdot \overline{\mathrm{Y}} \cdot \mathrm{X} \tag{B.2}$$

$$\mathrm{LUT2} = \overline{\mathrm{W}} \cdot \mathrm{Y} \cdot \overline{\mathrm{X}} \tag{B.3}$$

$$\mathrm{LUT3} = \overline{\mathrm{W}} \cdot \mathrm{Y} \cdot \mathrm{X} \tag{B.4}$$
$$\mathrm{LUT4} = \mathrm{W} \cdot \overline{\mathrm{Y}} \cdot \overline{\mathrm{X}} \tag{B.5}$$
$$\mathrm{LUT5} = \mathrm{W} \cdot \overline{\mathrm{Y}} \cdot \mathrm{X} \tag{B.6}$$
$$\mathrm{LUT6} = \mathrm{W} \cdot \mathrm{Y} \cdot \overline{\mathrm{X}} \tag{B.7}$$
$$\mathrm{LUT7} = \mathrm{W} \cdot \mathrm{Y} \cdot \mathrm{X} \tag{B.8}$$

The truth table for the equations B.1...B.8 is simply a "hot one", which is moving across the LUT truth table in the bitstream (shown in Table B.2).

W	Y	X	LUT0	LUT1	LUT2	LUT3	LUT4	LUT5	LUT6	LUT7
0	0	0	1	0	0	0	0	0	0	0
0	0	1	0	1	0	0	0	0	0	0
0	1	0	0	0	1	0	0	0	0	0
0	1	1	0	0	0	1	0	0	0	0
1	0	0	0	0	0	0	1	0	0	0
1	0	1	0	0	0	0	0	1	0	0
1	1	0	0	0	0	0	0	0	1	0
1	1	1	0	0	0	0	0	0	0	1

Table B.2: LUT test designs truth tables

B.7.3 Procedure of LUT search

The LUT_X bits were easy to find. They are all grouped together at the bitstream address Z = 7 and corresponding x and y coordinates of the logic cell, where was the design placed:

```
Z:     ...    Z=6       Z=7      ...
LUT0:  ...  00000000  00000001  ...
LUT1:  ...  00000000  00000010  ...
LUT2:  ...  00000000  00000100  ...
LUT3:  ...  00000000  00001000  ...
LUT4:  ...  00000000  00010000  ...
LUT5:  ...  00000000  00100000  ...
LUT6:  ...  00000000  01000000  ...
LUT7:  ...  00000000  10000000  ...
```

The bit position (n) at the bitstream address Z = 7 directly corresponds to the LUT$\{n\}$. This means, that the Z = 7 configuration byte is a content of the LUT_X truth table with the input vector (W, Y, X).

The LUT_Y search has a similar result. All bits are contained within the configuration byte Z = 6. Unlike the LUT_X, bits are systematically scrambled:

```
Z:     ...   Z=6       Z=7      ...
LUT0: ... 00000001 00000000 ...
LUT1: ... 00000100 00000000 ...
LUT2: ... 00000010 00000000 ...
LUT3: ... 00001000 00000000 ...
LUT4: ... 00010000 00000000 ...
LUT5: ... 01000000 00000000 ...
LUT6: ... 00100000 00000000 ...
LUT7: ... 10000000 00000000 ...
```

The scrambling can be explained in 2 ways, which are both functionally equivalent:

1. The X and Y inputs of the LUT_Y are connected reversely to LUT_X connection. The correct input vector for LUT_Y should be (W, X, Y)

2. Bits are simply scrambled in order to match the chip layout.

Since it is easier not to deal with data reordering during the analysis and to modify the LUT input order instead, the case 1 was selected as a result of the LUT search. Following hypothesis can be made as a result of the search:

Hypothesis B.1 *The LUT_X is represented by a truth table at* $\mathrm{Z} = 7$ *with signal input vector order (W, Y, X). The LUT_Y is represented by a truth table at* $\mathrm{Z} = 6$ *with signal input vector order (W, X, Y).*

B.8 Logic cell connections

Finding out all the connection inside the logic cell requires big amount of time both for testing and verification. But it can still be done manually for the AT94K40AL device.

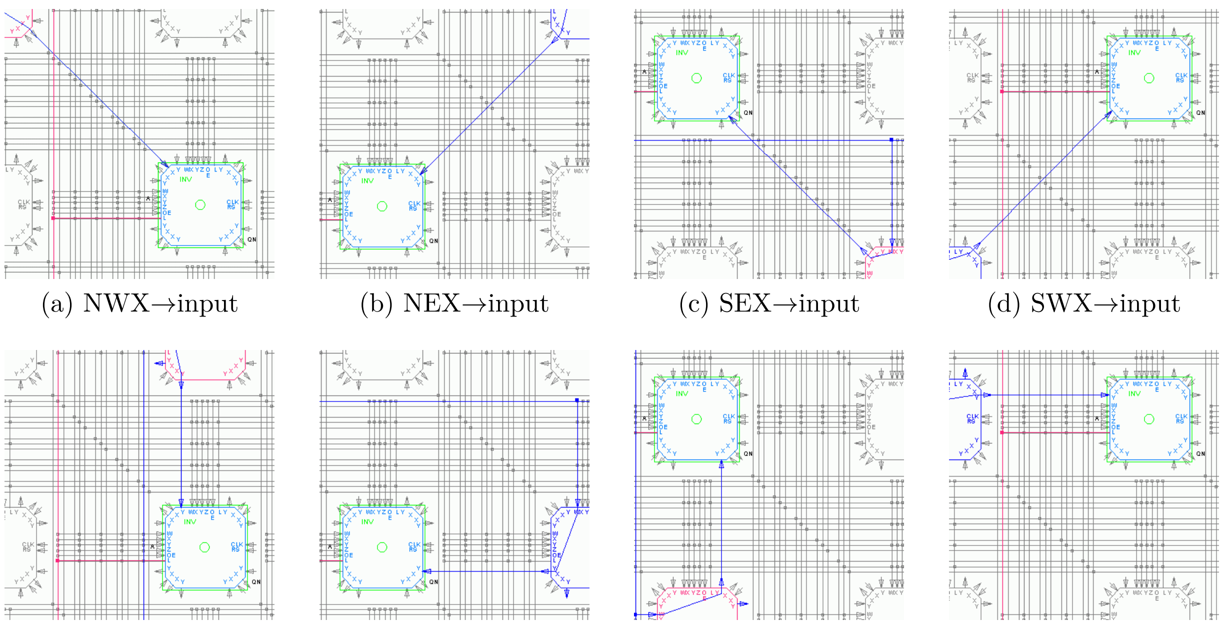
Up to now, only bitstream addressing and LUTs are known.

Analysis in this section uses a method of the simple design that is connected using different connection, via different signal path. Than are the bitstream files compared and the bit location revealed.

B.8.1 Neighbor connections

The test design was a simple inverter (“INV” macro). The cell has 8 direct neighbor inputs. 4 connect to the X input, 4 to the Y input. Neighbor connections are selected by a “multiplexer” in the datasheet [16], that is also shown in Fig. A.4. Coding of the “multiplexer” is unknown.

Bitstreams were created for all 8 designs (Fig. B.4) and they were compared among each other. For a given (x,y) coordinate of the design, following differences were found at addresses $\mathrm{Z} = 4$ and $\mathrm{Z} = 5$:


(a) NWX→input (b) NEX→input (c) SEX→input (d) SWX→input

(e) NY→input (f) EY→input (g) SY→input (h) WY→input

Figure B.4: Neighbor cell connection test designs

```
        Z=0      Z=1      Z=2      Z=3      Z=4      Z=5      Z=6      Z=7      Z=8
NWX: 00000101 00001000 00000000 00000000 00000000 00010000 00000000 01010101 00000010
NEX: 00000101 00001000 00000000 00000000 00000000 01000000 00000000 01010101 00000010
SEX: 00000101 00001000 00000000 00000000 00000000 00100000 00000000 01010101 00000010
SWX: 00000101 00001000 00000000 00000000 00000000 10000000 00000000 01010101 00000010
 NY: 00000101 00001000 00000000 00000000 10000000 00000000 00000000 00110011 00000010
 EY: 00000101 00001000 00000000 00000000 00010000 00000000 00000000 00110011 00000010
 SY: 00000101 00001000 00000000 00000000 00100000 00000000 00000000 00110011 00000010
 WY: 00000101 00001000 00000000 00000000 01000000 00000000 00000000 00110011 00000010
```

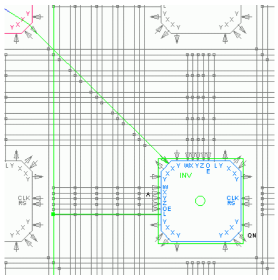
Since only the input source for the cell varied, responsible bits are clearly visible (marked red). The pattern of the LUT_X changed according to the input (X/Y), consistently with hypothesis B.1 in section B.7.3.

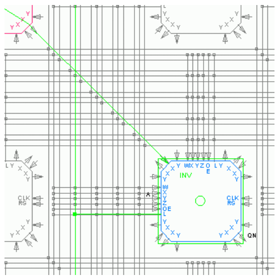
Hypothesis B.2 *The Neighbor X and Y connection are selected by dedicated single bits* 5.4 . . . 5.7 *and* 4.4 . . . 4.7.

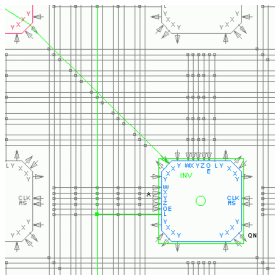
B.8.2 Cell L output

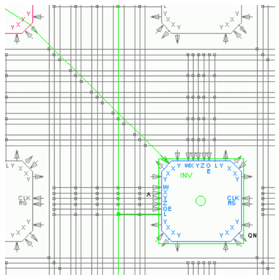
The output can be connected to the local buses: 5 horizontal local buses (H1 . . . H5) and 5 vertical local buses (V1 . . . V5). Bitstream for each output connection was generated using the inverter for all possible connections (Fig. B.5)

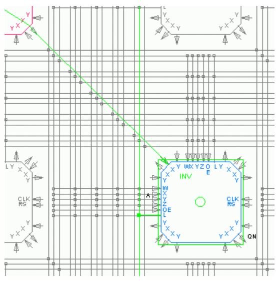
Bitstream is not as straightforward as in previous section was. More differences were found at addresses Z = 0, Z = 1 and Z = 8:

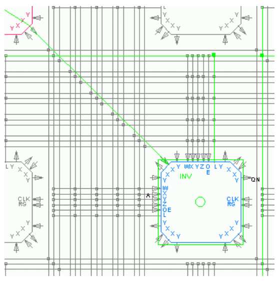

(a) Output→V5

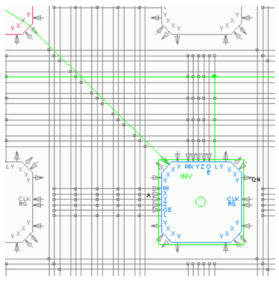

(b) output→V4

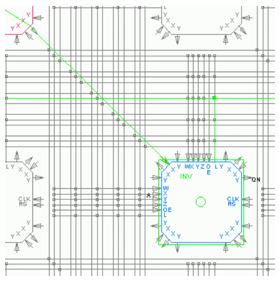

(c) output→V3

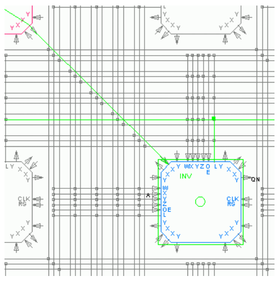

(d) output→V2


(e) Output→V1


(f) output→H5


(g) output→H4


(h) output→H3


(i) output→H2

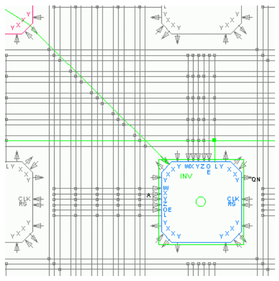

(j) output→H1

Figure B.5: Cell output test designs

```
          Z=0      Z=1      Z=2      Z=3      Z=4      Z=5      Z=6      Z=7      Z=8
O→V5 00000101 00001000 00000000 00000000 00000000 00010000 00000000 01010101 00000010
O→V4 00001001 00001000 00000000 00000000 00000000 00010000 00000000 01010101 00001000
O→V3 00100001 00001000 00000000 00000000 00000000 00010000 00000000 01010101 00010000
O→V2 00010001 00001000 00000000 00000000 00000000 00010000 00000000 01010101 10000000
O→V1 10000011 00001000 00000000 00000000 00000000 00010000 00000000 01010101 00000000
O→H5 00000101 01001000 00000000 00000000 00000000 00010000 00110011 00000000 00000001
O→H4 00001001 01001000 00000000 00000000 00000000 00010000 00110011 00000000 00000100
O→H3 00100001 01001000 00000000 00000000 00000000 00010000 00110011 00000000 00100000
O→H2 00010001 01001000 00000000 00000000 00000000 00010000 00110011 00000000 01000000
O→H1 01000011 01001000 00000000 00000000 00000000 00010000 00110011 00000000 00000000
```

Three sets of bits were observed:

1. Bits that are common for V5 and H5, V4 and H4, V3 and H3, V2 and H2, V1 and H1. These bits are marked by blue "1" in the results and connect the L net to the L5, L4, L3, L2 and L1 nets.

   Hypothesis B.3 *The L→L{1..5} connections are represented by bits 0.1, 0.4, 0.5, 0.3 and 0.2..*

2. Bits that are unique for each design. These bits are marked by red "1". There is only one unique bit in each design, therefore these bits have to represent the L{1..5}→H{1..5} and L{1..5}→V{1..5} connections. There is an open question about the directionality of this connection, which should be sorted out in the next section B.8.3.

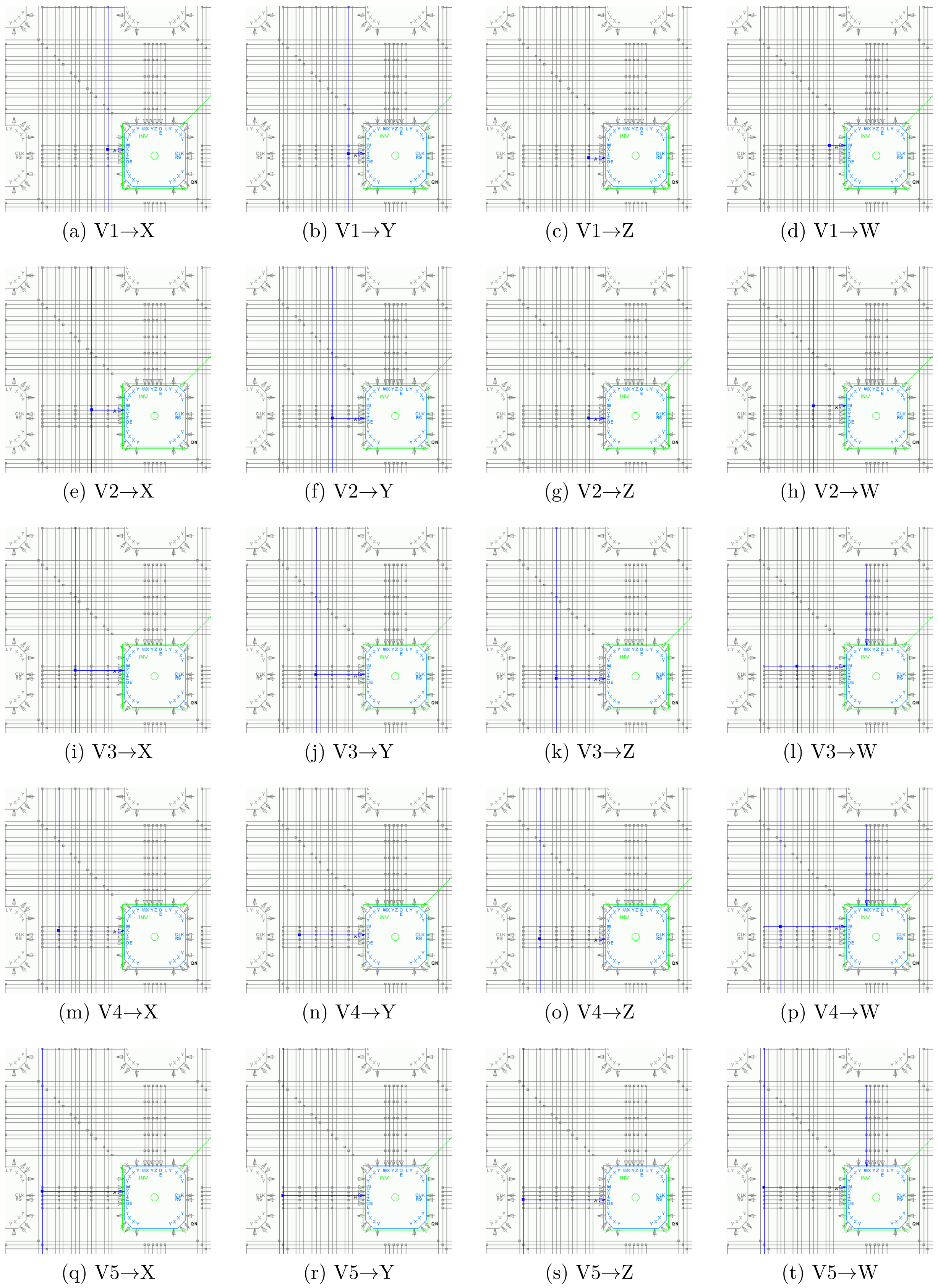
   Hypothesis B.4 *The L{1...5}→V{1...5} connection is represented by bits 0.7, 8.7, 8.4, 8.3 and 8.1. The L{1...5}→H{1...5} connection is represented by bits 0.6, 8.6, 8.5, 8.2 and 8.0.*

3. Bit that is marked by green "1" is only present when the output is mapped to Y. The LUT_Y is used instead of LUT_X. The LUTs signals are selected by the ZM multiplexer. .

   Hypothesis B.5 *The bit 1.6 selects "0"→ZM multiplexer. The "1"→ZM is the multiplexer's default value*

B.8.3 Cell inputs

There are 40 possibilities in total of how to connect a single signal to the LUT input: using the X, Y, Z and W input and using one of 5 horizontal local buses H{1...5} of one of 5 vertical local buses V{1...5}. The datasheet [16] indicates, that same switches (observed in section B.8.2 and written into the hypothesis B.4) are used for output and input. If



(a) V1→X (b) V1→Y (c) V1→Z (d) V1→W

(e) V2→X (f) V2→Y (g) V2→Z (h) V2→W

(i) V3→X (j) V3→Y (k) V3→Z (l) V3→W

(m) V4→X (n) V4→Y (o) V4→Z (p) V4→W

(q) V5→X (r) V5→Y (s) V5→Z (t) V5→W

Figure B.6: Cell input test designs

that confirms, only one direction of the local bus connection (either horizontal or vertical) is sufficient for complete input bit set revelation. So let's start with 20 designs.

Used test designs (a simple inverter with different connections) for vertical bus connection are shown in Fig. B.6. Output is always connected to NE X output.

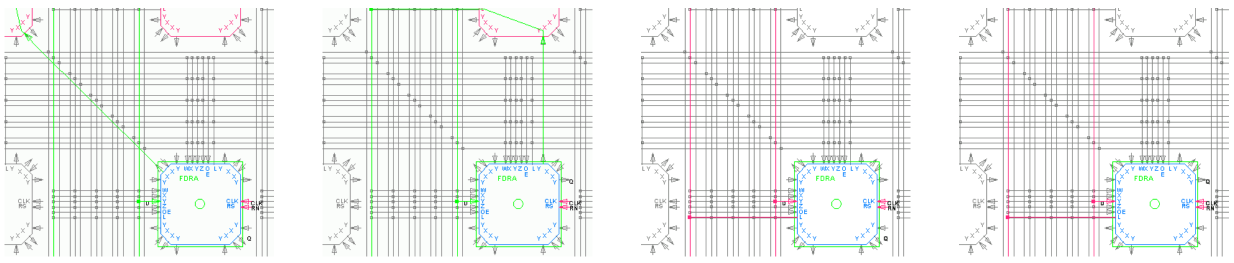
```
         Z=0      Z=1      Z=2      Z=3      Z=4      Z=5      Z=6      Z=7      Z=8
V1→X 10000001 00000000 00000000 00010000 00000000 00000000 00000000 01010101 00000000
V1→Y 10000001 00000000 01000000 00000000 00000000 00000000 00000000 00110011 00000000
V1→Z 10000001 00100000 10000000 00000000 00000000 00000000 00000000 00001111 00000000
V1→W 10000001 00000000 00000000 00100000 00000000 00000000 00000000 00001111 00000000
         Z=0      Z=1      Z=2      Z=3      Z=4      Z=5      Z=6      Z=7      Z=8
V2→X 00000001 00000000 00000000 00000000 00000000 00000001 00000000 01010101 10000000
V2→Y 00000001 00000000 00000000 00000000 00000001 00000000 00000000 00110011 10000000
V2→Z 00000001 00100000 00100000 00000000 00000000 00000000 00000000 00001111 10000000
V2→W 00000001 00000000 00000000 01000000 00000000 00000000 00000000 00001111 10000000
         Z=0      Z=1      Z=2      Z=3      Z=4      Z=5      Z=6      Z=7      Z=8
V3→X 00000001 00000000 00000000 00000000 00000000 00000010 00000000 01010101 00010000
V3→Y 00000001 00000000 00000000 00000000 00000010 00000000 00000000 00110011 00010000
V3→Z 00000001 00100000 00010000 00000000 00000000 00000000 00000000 00001111 00010000
V3→W 00000001 00000000 00000000 10000000 00000000 00000000 00000000 00001111 00010000
         Z=0      Z=1      Z=2      Z=3      Z=4      Z=5      Z=6      Z=7      Z=8
V4→X 00000001 00000000 00000000 00000000 00000000 00000100 00000000 01010101 00001000
V4→Y 00000001 00000000 00000000 00000000 00000100 00000000 00000000 00110011 00001000
V4→Z 00000001 00100000 00001000 00000000 00000000 00000000 00000000 00001111 00001000
V4→W 00000001 00000000 00000000 00001000 00000000 00000000 00000000 00001111 00001000
         Z=0      Z=1      Z=2      Z=3      Z=4      Z=5      Z=6      Z=7      Z=8
V5→X 00000001 00000000 00000000 00000000 00000000 00001000 00000000 01010101 00000010
V5→Y 00000001 00000000 00000000 00000000 00001000 00000000 00000000 00110011 00000010
V5→Z 00000001 00100000 00000100 00000000 00000000 00000000 00000000 00001111 00000010
V5→W 00000001 00000000 00000000 00000100 00000000 00000000 00000000 00001111 00000010
```

Several observation was made from these 20 designs:

1. Same bits that connect the L{1...5} output to corresponding V{1...5} were used for the V{1...5}→L{1...5} connection, consistently with hypothesis B.4. These bits are marked by red "1". This findings also answers the question from section B.8.2 (observation 2) about the directionality of the L{1...5}↔V{1...5} connection: It is bidirectional.

2. Bits that are unique for each design. These bits are marked by blue "1". There is only one unique bit per per each design, therefore this bits have to represent the L{1...5}→{X, Y, Z or W} connections.

   Hypothesis B.6 *The L1→{X,Y,Z,W} connection is represented by bits 3.4, 2.6, 2.7 and 3.5. The L2→{X,Y,Z,W} connection is represented by bits 5.0, 4.0, 2.5 and 3.6. The L3→{X,Y,Z,W} connection is represented by bits 5.1, 4.1, 2.4 and 3.7. The L4→{X,Y,Z,W} connection is represented by bits 5.2, 4.2, 2.3 and 3.3. The L4→{X,Y,Z,W} connection is represented by bits 5.3, 4.3, 2.2 and 3.2.*



(a) D→X (b) D→Y (c) D→L(X) (d) D→L(Y)

Figure B.7: Cell inner structure test designs aimed on oX, oY and P multiplexers analysis

3. Bit that is marked by green "1" is only present when the Z input is selected. According to the datasheet, there is an multiplexer (FB), that has the Z signal to pass in order to get to the W LUT input through the AND gate (FB and W). The other possible path (through the Z multiplexer) would probably require more bits to set the path through the D, C and Y-output multiplexers.

   Hypothesis B.7 *The Z→FB path is selected using bit 1.5. The "1"→FB is the multiplexer's default value.*

4. There is no special bit used for selecting the local bus connection and neighbor connection for the X and Y input. The Y and iY "multiplexers" (see Fig. A.4) are therefore made of a single "multiplexer".

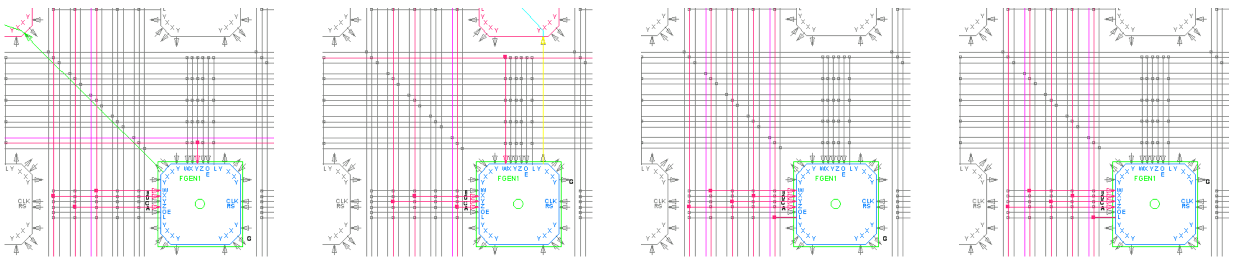
The B.6 and B.4 hypotheses should be tested also with a *horizontal* bus connection. That requires 20 more (almost similar) tests that *were performed* during the analysis. Results confirmed those hypotheses.

B.8.4 Cell inner connections

This section will answer how the FB, ZM, P, C, oX and oY multiplexers are controlled.

The asynchronous cell output can be in principal connected in two ways (though the delay is not same). The first path is direct from LUT to oX/oY, the second passes the D and C path above the first one.

The sequential cell (like the simple DFF) however has to pass the D path and is therefore more suitable for hypotheses testing. The 4-input LUT configuration also has to pass the D path, but uses more connection than DFF, therefore the DFF is used for the oX/oY bits analysis, as shown in Fig. B.7.



(a) XOR4→NWX (b) XOR4→NY (c) XOR4→L(X) (d) XOR4→L(Y)

Figure B.8: Cell inner structure 4-input LUT test designs aimed on oX, oY and P multiplexers analysis. The (X) and (Y) labels denotes the mapping of the output

	Z=0	Z=1	Z=2	Z=3	Z=4	Z=5	Z=6	Z=7	Z=8
D→X[a]	10000001	00000010	01000000	00000000	00000000	00000000	00000000	11001100	00000000
D→Y[b]	10000001	01000001	01000000	00000000	00000000	00000000	10101010	00000000	00000000
D→L[a]	10000101	00000010	01000000	00000000	00000000	00000000	00000000	11001100	00000010
D→L[b]	10000101	01000001	01000000	00000000	00000000	00000000	10101010	00000000	00000010

[a]Product is mapped to the X output
[b]Product is mapped to the Y output

The bitstream contains other differences, related to the clock and reset network. These bits have different addressing (sector-wise) and therefore should not interfere with individual cell configuration.

Assuming, that hypotheses B.3, B.4 and B.5 are valid (green bits "1" are understood), there are only bits 1.0 and 1.1, that were not yet understood (marked by red "1"). This has following consequences:

Hypothesis B.8 *The C→oX connection is selected by bit 1.1. The C→oY connection is selected by bit 1.0.*

Hypothesis B.9 *The Q→C connection is the C multiplexer's default selection. The Q→P connection is the P multiplexer's default selection.*

Hypothesis B.10 *The P→L buffer is enabled by default.*

Why bits 1.0 and 1.1 remain used when the L output is used instead, is unclear.

These findings do not reveal the C,P,oY and oX bitstream representation completely. Another test designs are necessary. The 4-input LUT configuration was proposed at the beginning of this chapter and are shown in Fig. B.8. The test circuits are different combination of the 4-input LUT configuration, buffered and unbuffered version.

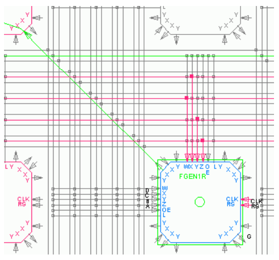


Figure B.9: Registered 4-input XOR →NWX. Test designs aimed on D→C connection analysis.

```
              Z=0      Z=1      Z=2      Z=3      Z=4      Z=5      Z=6      Z=7
XOR4→L[a]     10000011 10001110 00000100 00001000 00000010 00000001 10010110 01101001 ...
XOR4→L[b]     10000011 10001101 00000100 00001000 00000010 00000001 10010110 01101001 ...
XOR4→NY[b]    10000001 10000101 10000000 00000100 00000100 00000010 10010110 01101001 ...
XOR4→NWX[a]   01000001 10000110 01001000 10000000 00000000 00001000 10010110 01101001 ...
```

[a]Product is mapped to the X output
[b]Product is mapped to the Y output

Results of the oX and oY multiplexers (marked green “1”) are consistent with previous findings in this section and confirm the hypothesis B.8.

Assuming, that hypotheses B.3, B.4, B.6 and B.10 are valid, the only used bits left are bits 1.2, 1.3 (marked “1”) and 1.7 (marked “1”) left. Bit 1.3 is the only bit that is active when the product goes to the L line.

Hypothesis B.11 *The D→P connection is selected by bit 1.3.*

One of the two blue bits (“1”) has to be responsible for the D→C connection, the other has to be responsible for the iZ→ZM connection. Which one is which can be answer by comparison of the unbuffered 4-input LUT design (shown in Fig. B.8a) with the buffered one (shown in Fig. B.9).

```
              Z=0      Z=1      Z=2      Z=3      Z=4      Z=5      Z=6      Z=7
XOR4 →NWX 10000011 10001110 00000100 00001000 00000010 00000001 10010110 01101001 ...
XOR4R→NWX 01000001 10000010 10000000 10000000 00000001 00000100 10010110 01101001 ...
```

Up to now, the bitstream is quite understood. Assuming hypotheses B.3, B.4, B.6 and B.8 are valid, following 2 observations can be made from these 2 additional design comparison: bit 1.2 is active only for unbuffered design, revealing the D→C connection (marked “1”). This leaves the last unknown active bit to be revealed as the iZ→ZM connection.

Hypothesis B.12 *The D→C connection is selected by bit 1.2.*

Hypothesis B.13 *The iZ→ZM connection is selected by bit 1.7.*

With the knowledge of already discovered bits, a new look at already analyzed designs (in section B.7.3) can be made in order to see, if there is a bit selecting the XL→oX or YL→oY connections. The designs from Fig. B.3 can be taken again for the analysis:

```
    Z=0      Z=1      Z=2      Z=3      Z=4      Z=5      Z=6      Z=7      Z=8      Z=9
X: 10000001 00000000 01000000 00000100 00000000 00010000 00000000 00000001 00000001 00000000
Y: 10000001 00000000 00000000 00100000 10000000 10000000 00000001 00000000 00000000 00000000
```

By comparison of these 2 bitstream, there is clearly no bit selecting the XL→oX or YL→oY connections. Already known bits are marked green (“`1`”), the not yet discovered marked red. From this comparison, the hypothesis B.8 can be extended:

Hypothesis B.14 *The XL→oX is the oX multiplexer's default selection. The YL→oY is the oY multiplexer's default selection.*

Test design with the P→FB connection was not successfully synthesized. The signal path is always wired through some L wire (L1 for example). This bit has to be proved experimentally with the physical device response on the manual bit activation.

B.8.5 Bus crossing search

According to the datasheet [15] and EDA tool [14], a programmable switch is located at each bus crossing next to the logic cell. There are 5 bus planes, each having 2 express buses and 1 local bus. This makes 15 bits, that are expected to be found in the bitstream.

Since many bits are known from previous search, the set of remaining unknown bits and unknown connection is limited. At this point:

- Unknown bits are: 0.0, 1.4, 2.0, 2.1, 3.0, 3.1, 9.0. . .9.7
- Unknown structures are: bus crossing switches, P→FB, OE_H→L, OE_V→L, V{1. . .5}↔H{1. . .5}, V{1. . .5}T↔H{1. . .5}L and V{1. . .5}B↔H{1. . .5}R.

This counts 14 unknown bits and 18 undiscovered connections. The number of unique connection cannot be higher than the number of unknown bits, unless coding for the connection is used.

At this point of bitstream understanding, it is more effective to use another method of bitstream analysis, which us a large design that is spread all over the FPGA area. Then 3 approaches are possible:

1. **Search by connection**. Find (visually) desired activated connection c_d at different locations in the EDA tool. The location (l) with c_d usually contain also other connections active. It is necessary to find just enough locations l, that do all share only the c_s connection: $\bigcap C_l = c_s$, where C_l is the set of active connections in the location l. Than the bitstream comparison show exactly the bit position (b_s) of the searched connection c_s.The search is summarized by the Algorithm B.2.

Algorithm B.2 Search for connection's bit by used connections

```
input  : List of non-analyzed connections naConn
input  : List of non-analyzed bits naBits
input  : Connection under study cs
output: Bit position bitPos of the cs

Locs ← empty;                                  //clean the list of selected locations
repeat                                                     //add location to the list
    pick a location loc with cs, loc $\notin$ Locs;
    add loc to Locs;
    UsedConn ← naConn;
    foreach location il in Locs do                               //itereate locations
        UsedConn ← UsedConn $\cap$ il.getUsedConnections();  //make conn. intersection
    end
until UsedConn contains cs only;
cBits ← naBits;
foreach location il in Locs do                                   //itereate locations
    cBits ← cBits $\cap$ il.getBits();                          //make bits intersection
end
bitPos ← cBits;                                                      //The result(s)
```

2. **Search by bit**. Find all locations in the bitstream that have the searched bit b_s active. Then pick locations l, that share only the b_s bit: $\bigcap B_l = b_s$. Than the intersection ($\bigcap$) of connections over selected locations shows the connection c controlled by the bit b_s. It can be shown visually in the EDA. The resulting Algorithm B.3 has the structure very much alike the Algoritm B.2, but the procedure is different.

3. **Show the connection directly**. Make the list of all location that use the searched bit b. Than for all unknown bit b_u remove the location, where b_u is active. This method is a very fast, simple and effective one, yet not very likely to work out, especially when many unknown bits are present. This method can be formally written as Algorithm B.4.

Sometimes the use of algorithms B.4 and B.3 is trivial - a single design directly reveals the correspondence of the bit with a connection.

There are relations among successes of B.2, B.3 and B.4 Algorithms runs:

$$\begin{aligned}
\text{Alg. B.2 is successful} &\iff \text{Alg. B.3 is successful}\\
\text{Alg. B.4 is successful} &\implies \text{Alg. B.3 is successful}\\
\text{Alg. B.3 is successful} &\not\Longrightarrow \text{Alg. B.4 is successful}
\end{aligned}$$

All these algorithms assume, that the coordinates addressing of the resources are perfectly understood. These algorithms are most effective in the late phase of the analysis,

Algorithm B.3 Search for bit's connection by used bits

```
input  : List of non-analyzed bits naBits
input  : List of non-analyzed connections naConns
input  : Bit under study bs
output: Connection Conn controlled by the bs

Locs ← empty;                                //clean the list of selected locations
repeat                                               //add location to the list
    pick a location loc with bs active, loc $\notin$ Locs;
    add loc to Locs;
    UsedBits ← naBits;
    foreach location il in Locs do                        //itereate locations
        UsedBits ← UsedBits $\cap$ il.getUsedBits();         //make bits intersection
    end
until UsedBits contains bs only;
bConns ← naConns;
foreach location il in Locs do                            //itereate locations
    bConns ← bConns $\cap$ il.getConnections();             //make conn. intersection
end
Conn ← bConns;                                                  //The result(s)
```

where some bits are already understood. In the early stage, the intersection operations may be difficult due to the large number of unknown bits and connections. The probability of successful search is increasing with the number of known bits. The unsuccessful search can be therefore repeated each time a new bit and connection is found.

First, let's look at use coverage of the unknown bits.all bits, whether are they even used in the design. This will eliminate search for bits that are not used or are meaningless:

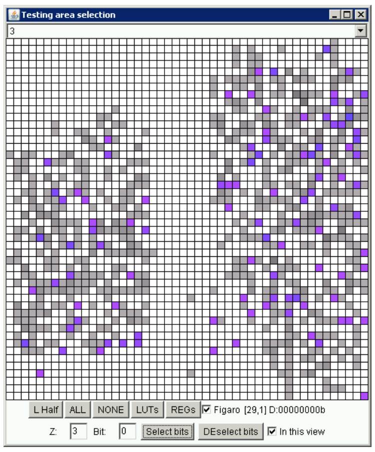
- Bit 0.0 is used in all cells;
- Bits 1.4, 2.0, 2.1, are not used in a single location in the whole design.

Application of Algoritm B.4 on the remaining bit set (3.0, 3.1, 9.0. . .9.7) is shown in Fig. B.10. It was successful for all these bits (though only bits 3.0 and 3.1 are shown). Both directions H→V and V→H were observed, when the appropriate bit was active.

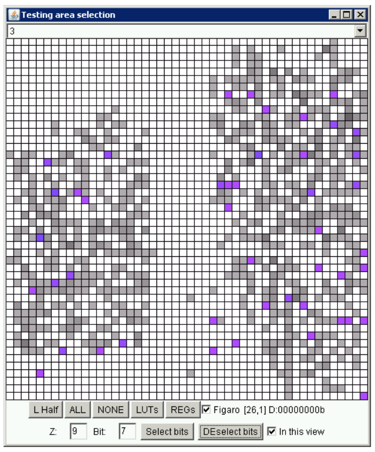
Hypothesis B.15 *Express bus H{1. . .5}B↔V{1. . .5}R connections ares controlled by bits 9.3, 3.0, 9.0, 9.2 and 9.5. Express bus H{1. . .5}T↔V{1. . .5}L connections ares controlled by bits 9.4, 9.1, 3.1, 9.7 and 9.6.*

Connections H{1. . .5}↔V{1. . .5} are still missing. It looks like that there is not a direct bit responsible for this connection. Connections (shown in Fig. B.11) can be analyzed by Algorithm B.2. The set of known connection has to be extended to all bits, even those already known. The algorithm would stop after the set of (V{123}↔H{123}[1],

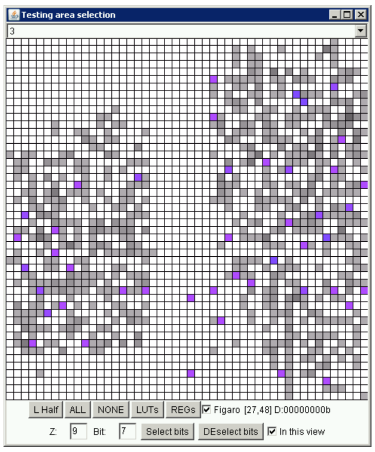
[1]V{123}↔H{123} means location, which has V1↔H1, V2↔H2 and V3↔H3 connections


Testing area selection
3
L Half ALL NONE LUTs REGs Figaro [29,1] D:00000000b
Z: 3 Bit: 0 Select bits DEselect bits In this view

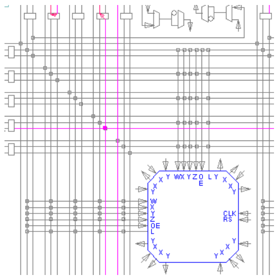
(a) All locations with bit 3.0


Testing area selection
3
L Half ALL NONE LUTs REGs Figaro [26,1] D:00000000b
Z: 9 Bit: 7 Select bits DEselect bits In this view

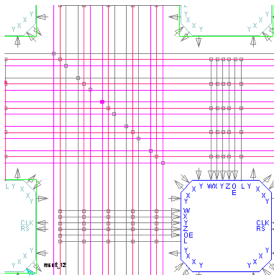
(b) Algorithm B.4 results, bit 3.0


Testing area selection
3
L Half ALL NONE LUTs REGs Figaro [27,48] D:00000000b
Z: 9 Bit: 7 Select bits DEselect bits In this view

(c) Algorithm B.4 results, bit 3.1



(d) Connection at bit 3.0 locations



(e) Connection at bit 3.1 locations.

Figure B.10: Search for bus crossing connections. Highlighted locations (blue color) indicate (a) all occurances of bit 3.0, (b) occurances of bit 3.0 that does not share any other unknown bit, (c) same for bit 3.1. (d) and (e) show EDA view of randomly picked location from (b) and (c)

Algorithm B.4 Reveal locations uniquely used by bit's connection

```
input  : List of non-analyzed bits naBits
input  : Bit under study bs
output: location(s) of the Connection Conn controlled by the bs

Locs ← all locations where bs is active;
foreach bit bi in naBits do                                  //itereate locations
    Locs ← Locs ∩ (∼ bi.getLocations();             //remove bi's Locations from list
end
return Locs;
```

V{12}↔H{12} and V{13}↔H{13}), but the V{14}↔H{14} eliminates further bits from selection, so it therefore wise to take more location bitstream snapshot.

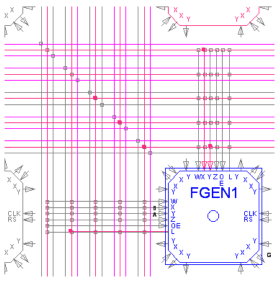
```
                  Z=0      Z=1      Z=2      Z=3      Z=4      Z=5      Z=6      Z=7      Z=8      Z=9
V{123}↔H{123}: 11001001 00001000 01000000 00000000 00000000 00001000 00000000 00010001 11111001 00000000
  V{13}↔H{13} : 11000001 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00110000 00000000
  V{12}↔H{12} : 11100001 10001101 00001000 00100000 00100000 00000001 00110011 00010101 11010100 00000000
  V{14}↔H{14} : 11000001 10000110 01000100 00001000 00000000 00000001 01000100 00010011 10001110 00000001
--------------------------------------------------------------------------------------------------------------
  bitwise AND : 11000001 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
```

It was revealed at the lines "bitwise AND", that the V1↔H1 connection is controlled by bits 0.6 and 0.7 (marked "1"). These bits are already known from previous analysis (H1↔L1, V1↔L1). The V1↔H1 connections in the EDA are therefore just a phantom connections provided by L1 horizontal and vertical connections described in Hypothesis B.4.

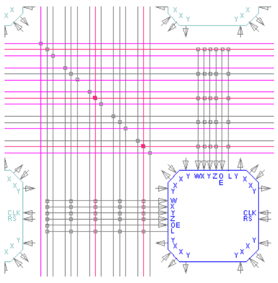
```
                  Z=0      Z=1      Z=2      Z=3      Z=4      Z=5      Z=6      Z=7      Z=8      Z=9
  V{13}↔H{13} : 11000001 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00110000 00000000
  V{14}↔H{14} : 11000001 10000110 01000100 00001000 00000000 00000001 01000100 00010011 10001110 00000001
  V{23}↔H{23} : 00000001 00000000 00000000 00000000 00000000 00000000 00000000 00000000 11110000 01000000
  V{24}↔H{24} : 00000001 00000000 00000000 00000000 00000000 00000000 00000000 00000000 11001100 00000000
  V{25}↔H{25} : 00000001 00000000 00000000 00000000 00000000 00000000 00000000 00000000 11000011 00000000
V{135}↔H{135}: 11000001 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00110011 00000000
--------------------------------------------------------------------------------------------------------------
 {23} AND {24} : 00000001 00000000 00000000 00000000 00000000 00000000 00000000 00000000 11000000 00000000
 {13} AND {23} : 00000001 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00110000 00000000
 {14} AND {24} : 00000001 00000000 00000000 00000000 00000000 00000000 00000000 00000000 10001100 00000000
 {25} AND {135}: 00000001 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000011 00000000
```

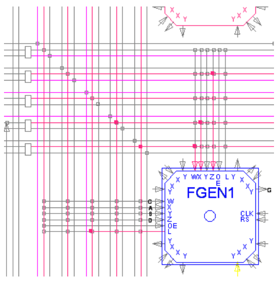
Further analysis (shown above) with same approach for V{2345}↔H{2345} proved, that connections V{2...5}↔H{2...5} is provided by the L{2...5} nets and its vertical and horizontal connections, as shown for V1↔H1 connection. .

Discovered connections are used for both signal direction (either H→V or V→H). This was checked at the EDA view and the compared configuration share exactly the same bits. This is a second proof of bidirectionality of the {VH}{1...5}→L{1...5} connections, which was questioned at section B.8.2.

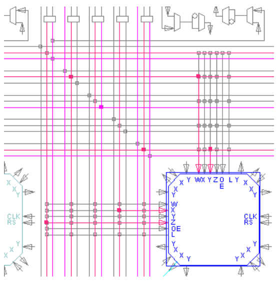


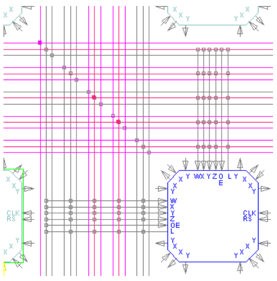
(a) V{123}↔H{123}

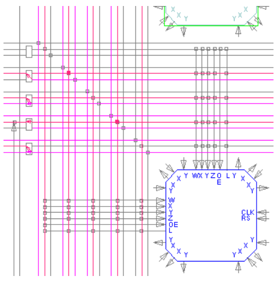

(b) V{13}↔H{13}

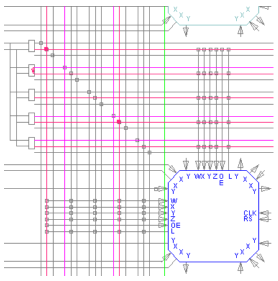


(c) V{12}↔H{12}


(d) V{14}↔H{14}


(e) V{23}↔H{23}


(f) V{24}↔H{24}


(g) V{25}↔H{25}

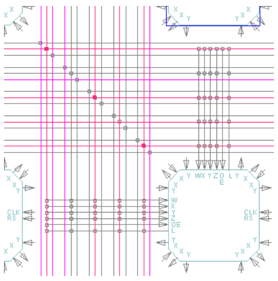

(h) V{135}↔H{135}

Figure B.11: H{1...5}↔V{1...5} connection analysis.

B.9 Repeaters/sector resources

Repeaters have different addressing, than the cell-related resources, as already discovered at section and shown in Fig. B.2. Repeaters are sector resources – they are placed every 4 cells. There is a difference between horizontal sectors and vertical sectors. Vertical sectors contain clock and reset distribution, whereas horizontal do not.

The repeater structure is completely unknown. The datasheet [15] says, that all configurations of 4-input/output are possible, plus the possibility to pass the. Possible configurations are shown in Fig. B.12, all of them are functionally equivalent. The purpose of the bit labels is to provide the transformation among the implementations, not to show the exact bit locations, which are not known at the moment.

The tri-state bus switch presented at Figs. B.12a and B.12b is not needed in implementations at Figs. B.12c and B.12d. The tri-state bus can be passed by the input "multiplexers" if they are implemented as an array of transfer gates, for example A.4 and A.5 (or B.3 and B.4) pass the upper bus left↔right without a buffer.

Repeater structures are very regular (on the contrary of the logic cell, which was more complex). It can be therefore expected, that the search will be easier, than the cell configuration search in previous section B.8.

The analysis uses basically the same techniques used in previous section. Routing is less convenient to control in the EDA, which may lead to a big effort in preparing the test bitstreams. There are many combinations of the repeater configuration; some of them

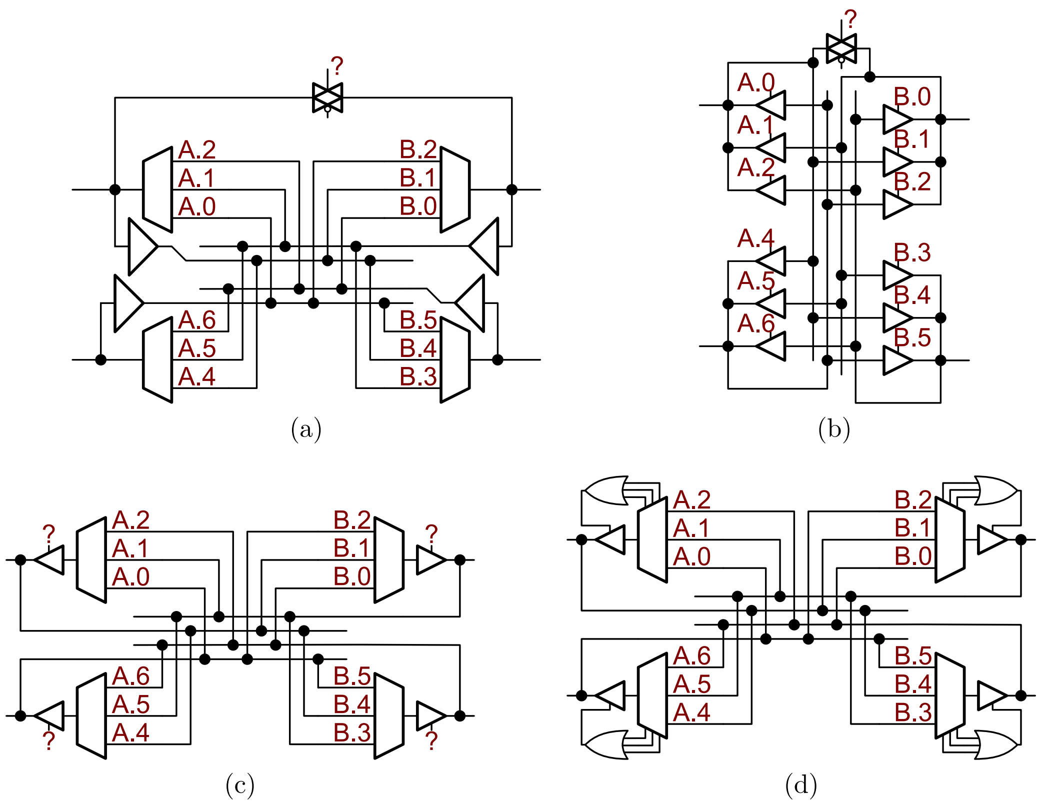


Figure B.12: Example of 4 different (but functionally equivalent) implementations of the repeater. All implementations are functionally equivalent.

might not even be seen on a small design. It is therefore effective to choose a large design and use one of the algorithms B.2, B.3, or B.4 to locate the bits and connections.

It would be also wise analyze only one resource at once, not horizontal and vertical together. Horizontal sector resources might be easier to analyze due to lack of clock&reset resources. Let's start with horizontal repeaters first, then continue with vertical repeaters and then finish with remaining resources.

B.9.1 Horizontal sector resources bitstream analysis methods

At first, it is useful to look at the map of bits {0x30... 0x39}{0...7} , 80 maps in total. There were discovered some bits, that are always "1" (marked "1") and always zero (marked "0"), as shown at the following portion of bitstream. Bits, that have at least one occurrence, are marked as ".".

```
           Z=0x30     Z=0x31     Z=0x32     Z=0x33     Z=0x34     Z=0x35     Z=0x36     Z=0x37     Z=0x38     Z=0x39
 always:  1...0...   00......   1...0...   11......   1...0...   00......   1...0...   11......   1...0...   00......
```

Some of these constant bits (at least 5 of them) should be responsible for passing the

Byte \ bit	7	6	5	4	3	2	1	0
0x30	all	44	25	16	-	55	15	23
0x31	-	-	17	11	5	27	7	37
0x32	all	39	18	17	-	57	34	22
0x33	all	all	7	20	7	21	4	34
0x34	all	53	21	9	-	65	33	14
0x35	-	-	20	10	5	24	10	29
0x36	all	38	22	9	-	57	21	18
0x37	all	all	20	12	10	28	6	46
0x38	all	31	23	8	-	34	32	19
0x39	-	-	27	13	14	34	7	30

Table B.3: Bit usage of horizontal repeaters. Values Indicate how many locations is the bit used on

tri-state buffers, which were not used in this test design. This bit count should be doubled, if pass of the tri-state bus is possible.

An observation was made here: The test design is large enough to cover all possible connections. This was guessed from the average number of bit occurrence location (24 locations) with maximum 65 locations and minimum 4 locations, as shown in Table B.3.

The count of the locations reveals a clear structure in the bitstream, when looked at not-allocated bits ("-") and always-one bits ("all"). Bits 3 and 7 in even addresses, alternating bits 7 and 6 ("11" $\leftrightarrow$"00") at odd address suggest, that there might be 5 repeated structures every 2 bytes, which exactly correspond with 5 repeaters in the FPGA.

The 2-byte structure can be also seen on the bit occupancy map of the used bits (".") in Fig. B.13. The structure is dominated by the occupancy of bits 0, 1, 2 and very slightly on bit 4. This supports the theory, that 2 bytes could represent one repeater.

Theoretically, if I knew the exact bit occupancy and exact connection occupancy, **I would be able to assign the bits to connections directly using the bit counts**. Unfortunately, comparison of the bit occupancy with the real connection occupancy is not an easy way of analysis for this FPGA, since the count of used connection is not provided by EDA and it would have been counted manually, which expands to a tremendous amount of manual work to do. The knowledge of real connection occupancy of a single design would not be sufficient for bit assignment anyway, since occupancy counts are not unique (many bits have same counts, as can be seen in Table B.3). More designs would be required in order to reveal the bitstream structure from bit occupancies and connection occupancies.

Another problem: The repeater position alternates (Shown in Fig. A.2 and further in Fig. B.14). The bit assignment can be different for odd and even locations, since it is not known, whether the repeater is physically shifted, flipped or completely different.

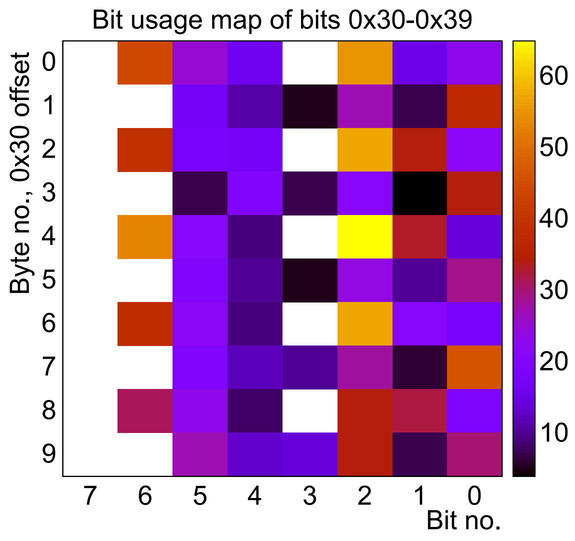


Figure B.13: Horizontal sector resources bit occupancy map

In next 3 subsection 3 examples of the Algorithms (B.2...B.4) will be given. If possible, same data and locations will be used in order to provide differences between the algorithms.

B.9.1.1 Search by bit

The Alg. B.3 has following inputs:

- List of non-analyzed bits (shown at B.9.1)
- List of non-analyzed connections (all connections at this point)
- A bit under study b_s: 0x30.0 (addres 0x30, bit 0)

There are 23 locations in total, that have the b_s active (also shown in Table B.3). Let's start with location [3,17], which has following connections active (Bits previously shown to be constant are marked "`1`". The bit under study b_s is marked by red "`1`" and all other unknown connections are marked "`1`"):

```
               Z=0x30   Z=0x31   Z=0x32   Z=0x33   Z=0x34   Z=0x35   Z=0x36   Z=0x37   Z=0x38   Z=0x39
(1) [03,17]: 10000001 00000000 11000000 11000000 10000010 00000000 10100100 11000000 11000100 00000000
```

This location is not sufficient to make a conclusion about the b_s connection, since there are 6 more connections (marked "`1`"). Another location has to be picked: for example location [11,15]:

```
               Z=0x30   Z=0x31   Z=0x32   Z=0x33   Z=0x34   Z=0x35   Z=0x36   Z=0x37   Z=0x38   Z=0x39
(2) [11,15]: 10000001 00000000 10000000 11010000 10000000 00100100 10000000 11000000 11000000 00000010
(1) AND (2): 10000001 00000000 10000000 11000000 10000000 00000000 10000000 11000000 11000000 00000000
```

The intersection of the used unknown bits gives 2 products: the b_s(0x30.0) and 0x38.6, as shown at the (`(1) AND (2)`) line. Another location has to be picked: for example [4,19]:

```
               Z=0x30   Z=0x31   Z=0x32   Z=0x33   Z=0x34   Z=0x35   Z=0x36   Z=0x37   Z=0x38   Z=0x39
(3) [04,19]: 10000001 00000000 10000000 11000000 10000100 00000000 11000000 11000001 10000000 00000000
AND (1,2,3): 10000001 00000000 10000000 11000000 10000000 00000000 10000000 11000000 10000000 00000000
```

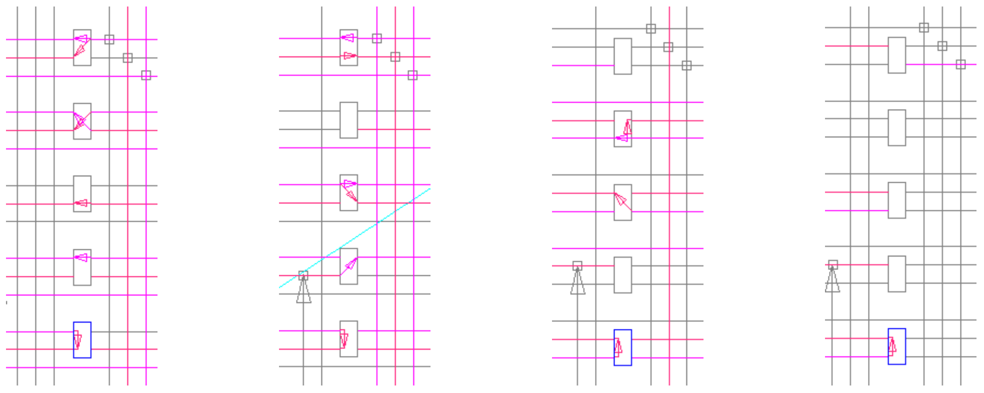

(a) [3,17] (b) [11,15] (c) [4,19] (d) [6,26]

Figure B.14: Search for repeater connection at the bit position 0x30.0

Now the bit intersection is just the b_s, therefore no other location is necessary for connection identification.

The output of the algorithm B.3 is a connection, which is a product of connections used at selected 3 locations. Connections have to be observed at EDA and they are displayed at Figures B.14a, B.14b and B.14c and here is the procedure, how the connections are intersectioned (T means Top input, B means Bottom input, E means East, W means West):

(1) uses: $\underline{\texttt{T1W}\rightarrow\texttt{B1W}}$, `T2E→T2W`, `B3E→B3W`, `T4E→B4W`, `B4E→T4W`, `T5E→B5W`, $\underline{\texttt{T5E}\rightarrow\texttt{T5W}}$

(2) uses: $\underline{\texttt{T1W}\rightarrow\texttt{B1W}}$, `B2W→T2E`, `T3W→T3E`, `T3W→B3E`, $\underline{\texttt{T5E}\rightarrow\texttt{T5W}}$, `B5W→B5E`

(3) uses: `B1W→T1W`, `B3E→T3W`, `B4E→B4W`, `B4E→T4E`

Intersection of these connections gives an empty set, clearly indicating something is wrong.

The visual check at Figures B.14a, B.14b and B.14c gives an quick answer. The connection of b_s cannot be at plane 5 (the top repeater) because of the unused repeater at [4,19] location. Because of the same reason it can't be the repeater at plane 4 (2nd from top), nor at plane 2 (2nd from bottom). It can't be the at plane 3 uses different connections at cases B.14a and B.14b. The only remaining option is therefore repeater which has only one connection.

The result leads to a new naming, which reflects the express/local bus instead of the top/bottom position. Intersection of the connections with the new labeling:

(1) uses: $\underline{\texttt{E1W}\rightarrow\texttt{L1W}}$, `E2E→E2W`, `L3E→L3W`, `E4E→L4W`, `L4E→E4W`, $\underline{\texttt{E5E}\rightarrow\texttt{E5W}}$, `E5E→E5L`

(2) uses: $\underline{\texttt{E1W}\rightarrow\texttt{L1W}}$, `L2W→E2E`, `E3W→E3E`, `E3W→L3E`, $\underline{\texttt{E5E}\rightarrow\texttt{E5W}}$, `L5W→L5E`

(3) uses: $\underline{\texttt{E1W}\rightarrow\texttt{L1W}}$, `E3E→L3W`, `E4E→E4W`, `E4E→L4E`

There is a possibility to cross-check the result with one of the 23 locations with b_s, which has only the b_s active. This location is shown in Fig. B.14d.

The **primary result** is that bit 0x30.0 controls the E1W→L1W connection for both odd and even locations. Some guesses based on current knowledge can be made:

- Repeater on plane 1 is controlled by 2 bytes from address 0x30, plane 2 from address

0x32, plane 3 from address 0x34, plane 4 from address 0x36 and plane 5 from address 0x38

- addressing of the odd and even repeaters is the same, but the repeaters are horizontally flipped (they do no keep the top/bottom input, but keep the express/local bus addressing)

Once the primary result is accepted, an immediate **side results** (which is not part of the algorithm though) can be made from the analyzed location: The E5E→E5W connection is controlled by bit 0x38.6.

B.9.1.2 connection search

This chapter will follow the algorithm B.2 and will try to find a connection L1W→E1W

The task of selecting proper location with this connection active requires quite a lot of manual effort to actually find such locations. At this case, about a 1/4 of all connection had to manually viewed in order to find 3 locations

The first location [12,27] (shown in Fig. B.15a) uses following 4 connections:

(a) uses: `L1W→E1W, E2W→L2W, L3E→L3W, E5W→L5E`

Another location [11,36] (shown in Fig. B.15b) uses following 5 connections (shared connections with the (a) case are `highlighted` by underlining):

(b) uses: `L1W→E1W, L3E→E3E, E4E→L4W, E5E→L5W, E5W→L5E`

There are 2 connections shared (L1W→E1W and E5W→L5E). Another location is picked (shown in Fig. B.15c) at coordinates [11,43]:

(c) uses: `L1W→E1W, L1W→L1E, E2W→L2E, E4E→L4W`

The L1W→E1W connection is the only left. The bit responsible for this connection can be analyzed by intersection if bitstream of these 3 locations:

```
               Z=0x30   Z=0x31   Z=0x32   Z=0x33   Z=0x34   Z=0x35   Z=0x36   Z=0x37   Z=0x38   Z=0x39
(a) [12,27]: 10010000 00000000 10000001 11000000 10000010 00000000 10000000 11000000 10000000 00000100
(b) [11,36]: 10010000 00000000 10000000 11..0000 10000000 0000.000 10000100 11000000 10000100 00000100
(c) [11,43]: 10010000 00000010 10000000 11000100 10000000 00000000 10000100 11000000 10000000 00000000
```

```
AND (a,b,c): 10010000 00000000 10000000 11000000 10000000 00000000 10000000 11000000 10000000 00000000
(a) AND (b): 100P0000 00000000 10000000 11000000 10000000 00000000 10000000 11000000 10000000 00000100
(b) AND (c): 10010000 00000000 10000000 11000000 10000000 00000000 10000100 11000000 10000000 00000000
```

The primary result is the address of the L1W→E1W connection: 0x30.4 (as seen on the line AND (a,b,c) above). Once this is accepted, another side results can be immediately observed by comparing the locations (a) with (b) and (b with (c), as shown at the bitstream code above. The connection E3W->L5E is controlled by bit 0x39.2 and connection E4E->L4W is controlled by bit 0x36.2.

B.9.1.3 Direct bit revelation

This section will show some examples of successful and unsuccessful runs of Algorithm B.4.

The algorithm starts with a list of locations of searched bit b_s. The first example will repeat the bit at address 0x30.1. There are 15 locations, in which the b_s is active:

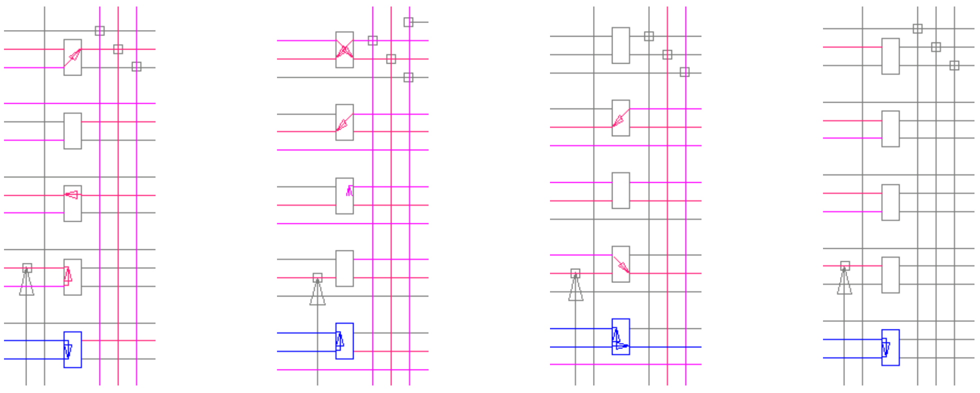

(a) [12,27] (b) [11,36] (c) [11,43] (d) [6,19]

Figure B.15: Search for connection L1W→E1W in horizontal repeater

```
          Z=0x30   Z=0x31   Z=0x32   Z=0x33   Z=0x34   Z=0x35   Z=0x36   Z=0x37   Z=0x38   Z=0x39
  known: 1..x0..x 00...... 1...0... 11...... 1...0... 00...... 1...0x.. 11...... 1x..0... 00...x..
[13,09]:  10000010 00000000 10000000 11000000 10000000 00000000 10000000 11000000 10000000 00000000
[05,11]:  10000010 00100000 11000000 11000000 11000100 00000000 10000100 11000001 10000000 00000000
[08,12]:  11000010 00000000 11000000 11000000 10000000 00000100 10000100 11000000 10000000 00000000
[04,13]:  10000010 00000000 10010000 11010000 11000010 00000000 10000000 11100000 11000000 00000000
[05,13]:  10000010 00000000 10000001 11000000 10100100 00000000 10000000 11100000 10000000 00010000
[11,13]:  10000010 00100000 10000000 11000000 10000000 00010010 10000000 11000000 10000000 00000001
[04,15]:  10000010 00000000 11000000 11000000 10000000 00000001 10000000 11000010 10000010 00000000
[05,15]:  10000010 00000000 10000010 11000000 10000000 00000100 10000000 11000000 10000000 00000000
[06,16]:  10000010 00000000 10000010 11000000 11000000 00000000 10000000 11100000 10000000 00000000
[11,17]:  11000010 00000000 10000000 11000000 10000000 00000000 10100000 11000000 11000000 00000000
[04,22]:  10000010 00000000 10100010 11000000 10000000 00000001 10100000 11000000 11000000 00000000
[11,25]:  10000010 00000000 10000001 11000000 11000000 00000000 10000000 11000001 10000001 00000001
[03,31]:  10000010 00000000 10000000 11000000 10000000 00000000 10000100 11000000 10000000 00000000
[12,35]:  11000010 00000000 11000000 11000000 10000000 00000100 10000000 11000000 11000000 00000000
[09,45]:  10000010 00000000 10000000 11000001 10000000 00000000 10000000 11100000 10000010 00000000
```

This list of location will be reduced by occurrences of unknown bits locations. The bit 0x30.6 removes locations [08,12], [11,17] and [12,35] from the list, keeping following locations:

`0x30.6: [13,09], [05,11], [04,13], [05,13], [11,13], [04,15], [05,15], [06,16], [04,22], [11,25], [03,31], and [09,45].`

This list is further reduced by bits 0x31.5 (removes [5,11] and [11,13]) and 0x32.0 (removes[5,15],[11,25]). Bit 0x32.1 remove locations [05,15] and [06,16], but the location [05,15] was already removed at previous step by bit 0x32.0. The list if further reduced by bits 0x32.4 (removes [4,13]), 0x32.5 (removes [04,22]), 0x32.6 (removes among others locations [04,15]), 0x33.0 (removes [09,45]) and 0x34.2 (removes locations [05,13]. No other bit reduces the list of location. Progress of the reduction is shown below:

```
0x30.6:[13,09],[05,11],[04,13],[05,13],[11,13],[04,15],[05,15],[06,16],[04,22],[11,25],[03,31],[09,45]
0x31.5:[13,09],[04,13],[05,13],[04,15],[05,15],[06,16],[04,22],[11,25],[03,31],[09,45]
0x32.0:[13,09],[04,13],[05,13],[04,15],[06,16],[04,22],[03,31],[09,45]
0x32.1:[13,09],[04,13],[05,13],[04,15],[04,22],[03,31],[09,45]
0x32.4:[13,09],[05,13],[04,15],[04,22],[03,31],[09,45]
0x32.5:[13,09],[05,13],[04,15],[03,31],[09,45]
0x32.6:[13,09],[05,13],[03,31],[09,45]
0x32.6:[13,09],[05,13],[03,31]
0x34.2:[13,09],[03,31]
```

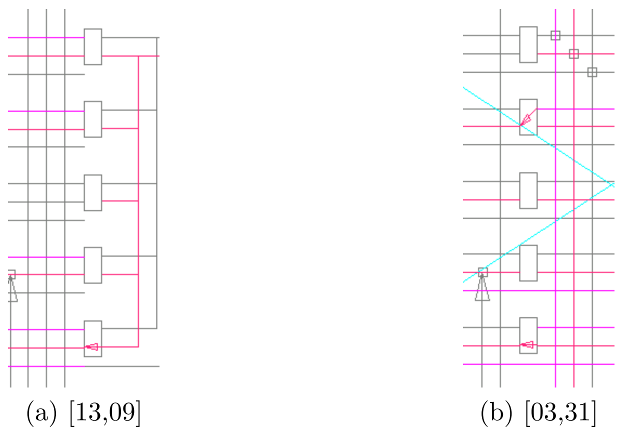

(a) [13,09] (b) [03,31]

Figure B.16: Search for bit 0x31.1. .

The algorithm B.4 succeeded and found two locations: [13,09] and [03,31], which are shown in Fig. B.16 directly. Bit 0x30.1 controls the L1E->L1W connection. The E4E->L4W connection (shown in Fig. B.16b and controlled by bit 0x36.2) was already discovered in previous section.

The algorithm B.4 is not always successful. Such a bit is 0x30.2, which was active on 55 locations, but never alone (or with known bits only).

B.9.2 Clock and reset

Although not part of this research, clock and reset distribution network were partly observed by a simple method: On half of the FPGA was filled with the clocked resources (registers), the other was filled only by the combinational logic. As a result, a map revealing the clock-related resource location was following map was observed.

B.9.3 Vertical sector resources

The vertical sector resources could be tested by exactly the same methods, that were used in previous section for the horizontal resources (section B.9.1). A brief look at the bitstream structure showed extreme similarities to the horizontal repeaters, which indicated, that vertical resource control might be the same, as for the horizontal resource. This observation was found very useful and the mapping from the horizontal repeater to the vertical repeater was found very quickly, though differences were found for the non-repeater part of the bitstream, which should be related to the sector clock/reset resources, as indicated in A.9. The clock and reset resources were not part of this study.

B.10 Which Algorithm to choose

Basically all algorithms B.2. . .B.4 will solve the task. The design itself affects the speed of analysis. Algorithms B.4 was found to be the fastest and most useful, however not always can give a result, either because the searched connection does not exist alone at any location due to too dense design, or the design is too small and the bit is not use at all.

Speed of the algorithms B.2 and B.3 is comparable, but the total speed depends on the available tools to do the analysis. For this analysis was done mostly by hand with the bitstream printed in 1s and 0s (as shown throughout this Appendix many times), the faster algorithm was algorithm B.3.

B.11 Verification

The verification of the bitstream knowledge was done completely manually, since the FPGA is not very large and the FPGA is highly regular. A medium sized design, different from the analyzed, was loaded both into the EDA and bitstream analysis tool. Every single bit was selected in the bitstream analysis tool and locations of occurrences were highlighted. Highlighted locations were crosschecked with the EDA views, whether the correct resource is active. Some bits were not possible to verify by this way. Anther design was used, or (rarely) one of the algorithms B.2. . .B.4 was used again, but the different one, than lead to the bit discovery.

Second assurance, that the bitstream analysis result was correct, comes from the validation during the classification and in-hardware emulation. Wrong analysis result would lead to wrong prediction.

Appendix C

Full results

This section contains full set of results from the emulator, where all bits are grouped by in the fault classes and resources. Even more detailed response for every bit is provided by the emulator, but amount of that information is huge and basically meaningless to publish.

Results have a structure of log recorded during testing and they are provided in several parts:

1. A detailed statistics based on the combination of both FPGA resource and fault model. This statistics can be fully expanded to the 2D table for the resource vs. fault table by filling unlisted class combinations by zero.
2. Statistics grouped by the FPGA resources
3. Statistics grouped by the fault model
4. Cumulative result of the complete half of the FPGA
5. Time of measurement (complete testing, but without the initial FPGA programming)

At the beginning, pairs of designs are shown next to each other's. The pair is the same circuit tested, but once without the parity predictor and second time with the parity encoder. Since no parity checking is available with unsecured design, the error is indicated only by the comparison and can be therefore classified by the A and C categories. For the parity encoded designs, a full A, B, C and D category statistics is given.

Some benchmarks were tested only with the parity encoder. These benchmarks are shown at the end.

Full testing of those circuits included 16910 bytes to test, which results in 135280 bits that were all tested. The amount of predicted bits, which have no functional influence on the circuit, is extracted from the *unused* and *antenna* categories (result lines `cumulative7`, `cumulative12` and `cumulative13`.

5xp1 without parity

```
Full0:  A:17296 B:0     C:0     D:0     LUT+UNUSED
Full1:  A:126   B:0     C:242   D:0     LUT+ALTERNATE
Full2:  A:32212 B:0     C:0     D:0     CELL_INTERCONN.+UNUSED
Full3:  A:125   B:0     C:168   D:0     CELL_INTERCONN.+ALTERNATE
Full4:  A:24    B:0     C:48    D:0     CELL_INTERCONN.+OPEN
Full5:  A:70    B:0     C:98    D:0     CELL_INTERCONN.+CONFLICT_OF
Full6:  A:249   B:0     C:243   D:0     CELL_INTERCONN.+CONFLICT_FF
Full7:  A:14173 B:0     C:0     D:0     CELL_INTERCONN.+OUTPUT_ANTENNA
Full8:  A:56    B:0     C:6     D:0     CELL_INTERCONN.+UNPREDICTABLE
Full9:  A:10638 B:0     C:0     D:0     BUS_TO_CELL+UNUSED
Full10: A:50    B:0     C:94    D:0     BUS_TO_CELL+OPEN
Full11: A:37    B:0     C:19    D:0     BUS_TO_CELL+CONFLICT_FF
Full12: A:202   B:0     C:0     D:0     BUS_TO_CELL+OUTPUT_ANTENNA
Full13: A:10518 B:0     C:0     D:0     BUS_CROSSING+UNUSED
Full14: A:8     B:0     C:19    D:0     BUS_CROSSING+OPEN
Full15: A:13    B:0     C:2     D:0     BUS_CROSSING+CONFLICT_FF
Full16: A:480   B:0     C:0     D:0     BUS_CROSSING+OUTPUT_ANTENNA
Full17: A:33596 B:0     C:0     D:0     BUS_REPEATER+UNUSED
Full18: A:14    B:0     C:75    D:0     BUS_REPEATER+OPEN
Full19: A:172   B:0     C:521   D:0     BUS_REPEATER+CONFLICT_FF
Full20: A:160   B:0     C:0     D:0     BUS_REPEATER+INPUT_ANTENNA
Full21: A:682   B:0     C:0     D:0     BUS_REPEATER+OUTPUT_ANTENNA
Full22: A:1104  B:0     C:0     D:0     FAKE+UNUSED
Full23: A:11728 B:0     C:12    D:0     UNKNOWN+UNEXPLORED
---- FPGA RESOURCES -----
Cumulative0:    A:17422 B:0     C:242   D:0     LUT
Cumulative1:    A:46909 B:0     C:563   D:0     CELL_INTERCONN.
Cumulative2:    A:10927 B:0     C:113   D:0     BUS_TO_CELL
Cumulative3:    A:11019 B:0     C:21    D:0     BUS_CROSSING
Cumulative4:    A:34624 B:0     C:596   D:0     BUS_REPEATER
Cumulative5:    A:1104  B:0     C:0     D:0     FAKE
Cumulative6:    A:11728 B:0     C:12    D:0     UNKNOWN
---- FAULT MODELS -----
Cumulative7:    A:105364 B:0    C:0     D:0     UNUSED
Cumulative8:    A:251   B:0     C:410   D:0     ALTERNATE
Cumulative9:    A:96    B:0     C:236   D:0     OPEN
Cumulative10:   A:70    B:0     C:98    D:0     CONFLICT_OF
Cumulative11:   A:471   B:0     C:785   D:0     CONFLICT_FF
Cumulative12:   A:160   B:0     C:0     D:0     INPUT_ANTENNA
Cumulative13:   A:15537 B:0     C:0     D:0     OUTPUT_ANTENNA
Cumulative14:   A:56    B:0     C:6     D:0     UNPREDICTABLE
Cumulative15:   A:11728 B:0     C:12    D:0     UNEXPLORED
Result: All: A:133733    B:0     C:1547  D:0
```

5XP1 with parity

```
Full0:  A:17276 B:0     C:0     D:0     LUT+UNUSED
Full1:  A:30    B:314   C:16    D:28    LUT+ALTERNATE
Full2:  A:32239 B:0     C:0     D:0     CELL_INTERCONN.+UNUSED
Full3:  A:37    B:222   C:8     D:13    CELL_INTERCONN.+ALTERNATE
Full4:  A:1     B:67    C:2     D:3     CELL_INTERCONN.+OPEN
Full5:  A:11    B:127   C:0     D:34    CELL_INTERCONN.+CONFLICT_OF
Full6:  A:115   B:280   C:11    D:47    CELL_INTERCONN.+CONFLICT_FF
Full7:  A:14178 B:0     C:0     D:0     CELL_INTERCONN.+OUTPUT_ANTENNA
Full8:  A:58    B:19    C:0     D:0     CELL_INTERCONN.+UNPREDICTABLE
Full9:  A:10597 B:0     C:0     D:0     BUS_TO_CELL+UNUSED
Full10: A:7     B:109   C:3     D:34    BUS_TO_CELL+OPEN
Full11: A:25    B:23    C:0     D:5     BUS_TO_CELL+CONFLICT_FF
Full12: A:237   B:0     C:0     D:0     BUS_TO_CELL+OUTPUT_ANTENNA
Full13: A:10578 B:0     C:0     D:0     BUS_CROSSING+UNUSED
Full14: A:0     B:7     C:0     D:8     BUS_CROSSING+OPEN
Full15: A:3     B:0     C:0     D:1     BUS_CROSSING+CONFLICT_FF
Full16: A:443   B:0     C:0     D:0     BUS_CROSSING+OUTPUT_ANTENNA
Full17: A:33550 B:0     C:0     D:0     BUS_REPEATER+UNUSED
Full18: A:0     B:54    C:4     D:33    BUS_REPEATER+OPEN
Full19: A:43    B:373   C:21    D:269   BUS_REPEATER+CONFLICT_FF
Full20: A:166   B:0     C:0     D:0     BUS_REPEATER+INPUT_ANTENNA
Full21: A:707   B:0     C:0     D:0     BUS_REPEATER+OUTPUT_ANTENNA
Full22: A:1104  B:0     C:0     D:0     FAKE+UNUSED
Full23: A:11719 B:13    C:0     D:8     UNKNOWN+UNEXPLORED
---- FPGA RESOURCES -----
Cumulative0:    A:17306 B:314   C:16    D:28    LUT
Cumulative1:    A:46639 B:715   C:21    D:97    CELL_INTERCONN.
Cumulative2:    A:10866 B:132   C:3     D:39    BUS_TO_CELL
Cumulative3:    A:11024 B:7     C:0     D:9     BUS_CROSSING
Cumulative4:    A:34466 B:427   C:25    D:302   BUS_REPEATER
Cumulative5:    A:1104  B:0     C:0     D:0     FAKE
Cumulative6:    A:11719 B:13    C:0     D:8     UNKNOWN
---- FAULT MODELS -----
Cumulative7:    A:105344 B:0    C:0     D:0     UNUSED
Cumulative8:    A:67    B:536   C:24    D:41    ALTERNATE
Cumulative9:    A:8     B:237   C:9     D:78    OPEN
Cumulative10:   A:11    B:127   C:0     D:34    CONFLICT_OF
Cumulative11:   A:186   B:676   C:32    D:322   CONFLICT_FF
Cumulative12:   A:166   B:0     C:0     D:0     INPUT_ANTENNA
Cumulative13:   A:15565 B:0     C:0     D:0     OUTPUT_ANTENNA
Cumulative14:   A:58    B:19    C:0     D:0     UNPREDICTABLE
Cumulative15:   A:11719 B:13    C:0     D:8     UNEXPLORED
Result: All: A:133124    B:1608  C:65    D:483
```

alu1 without parity

```
Full0:  A:17542 B:0     C:0     D:0     LUT+UNUSED
Full1:  A:0     B:0     C:122   D:0     LUT+ALTERNATE
Full2:  A:32837 B:0     C:0     D:0     CELL_INTERCONN.+UNUSED
Full3:  A:17    B:0     C:70    D:0     CELL_INTERCONN.+ALTERNATE
Full4:  A:0     B:0     C:24    D:0     CELL_INTERCONN.+OPEN
Full5:  A:1     B:0     C:50    D:0     CELL_INTERCONN.+CONFLICT_OF
Full6:  A:30    B:0     C:141   D:0     CELL_INTERCONN.+CONFLICT_FF
Full7:  A:14298 B:0     C:0     D:0     CELL_INTERCONN.+OUTPUT_ANTENNA
Full8:  A:2     B:0     C:2     D:0     CELL_INTERCONN.+UNPREDICTABLE
Full9:  A:10876 B:0     C:0     D:0     BUS_TO_CELL+UNUSED
Full10: A:2     B:0     C:59    D:0     BUS_TO_CELL+OPEN
Full11: A:4     B:0     C:15    D:0     BUS_TO_CELL+CONFLICT_FF
Full12: A:84    B:0     C:0     D:0     BUS_TO_CELL+OUTPUT_ANTENNA
Full13: A:10663 B:0     C:0     D:0     BUS_CROSSING+UNUSED
Full14: A:0     B:0     C:8     D:0     BUS_CROSSING+OPEN
Full15: A:0     B:0     C:6     D:0     BUS_CROSSING+CONFLICT_FF
Full16: A:363   B:0     C:0     D:0     BUS_CROSSING+OUTPUT_ANTENNA
Full17: A:34372 B:0     C:0     D:0     BUS_REPEATER+UNUSED
Full18: A:0     B:0     C:51    D:0     BUS_REPEATER+OPEN
Full19: A:13    B:0     C:344   D:0     BUS_REPEATER+CONFLICT_FF
Full20: A:87    B:0     C:0     D:0     BUS_REPEATER+INPUT_ANTENNA
Full21: A:353   B:0     C:0     D:0     BUS_REPEATER+OUTPUT_ANTENNA
Full22: A:1104  B:0     C:0     D:0     FAKE+UNUSED
Full23: A:11739 B:0     C:1     D:0     UNKNOWN+UNEXPLORED
---- FPGA RESOURCES -----
Cumulative0:    A:17542 B:0     C:122   D:0     LUT
Cumulative1:    A:47185 B:0     C:287   D:0     CELL_INTERCONN.
Cumulative2:    A:10966 B:0     C:74    D:0     BUS_TO_CELL
Cumulative3:    A:11026 B:0     C:14    D:0     BUS_CROSSING
Cumulative4:    A:34825 B:0     C:395   D:0     BUS_REPEATER
Cumulative5:    A:1104  B:0     C:0     D:0     FAKE
Cumulative6:    A:11739 B:0     C:1     D:0     UNKNOWN
---- FAULTS -----
Cumulative7:    A:107394 B:0    C:0     D:0     UNUSED
Cumulative8:    A:17    B:0     C:192   D:0     ALTERNATE
Cumulative9:    A:2     B:0     C:142   D:0     OPEN
Cumulative10:   A:1     B:0     C:50    D:0     CONFLICT_OF
Cumulative11:   A:47    B:0     C:506   D:0     CONFLICT_FF
Cumulative12:   A:87    B:0     C:0     D:0     INPUT_ANTENNA
Cumulative13:   A:15098 B:0     C:0     D:0     OUTPUT_ANTENNA
Cumulative14:   A:2     B:0     C:2     D:0     UNPREDICTABLE
Cumulative15:   A:11739 B:0     C:1     D:0     UNEXPLORED
Result: All: A:134387    B:0     C:893   D:0
```

alu1 with parity

```
Full0:  A:16988 B:0     C:0     D:0     LUT+UNUSED
Full1:  A:16    B:660   C:0     D:0     LUT+ALTERNATE
Full2:  A:31642 B:0     C:0     D:0     CELL_INTERCONN.+UNUSED
Full3:  A:75    B:429   C:0     D:0     CELL_INTERCONN.+ALTERNATE
Full4:  A:1     B:121   C:0     D:0     CELL_INTERCONN.+OPEN
Full5:  A:14    B:229   C:0     D:9     CELL_INTERCONN.+CONFLICT_OF
Full6:  A:170   B:651   C:0     D:18    CELL_INTERCONN.+CONFLICT_FF
Full7:  A:13997 B:0     C:0     D:0     CELL_INTERCONN.+OUTPUT_ANTENNA
Full8:  A:93    B:23    C:0     D:0     CELL_INTERCONN.+UNPREDICTABLE
Full9:  A:10357 B:0     C:0     D:0     BUS_TO_CELL+UNUSED
Full10: A:3     B:216   C:0     D:18    BUS_TO_CELL+OPEN
Full11: A:45    B:67    C:0     D:7     BUS_TO_CELL+CONFLICT_FF
Full12: A:327   B:0     C:0     D:0     BUS_TO_CELL+OUTPUT_ANTENNA
Full13: A:10222 B:0     C:0     D:0     BUS_CROSSING+UNUSED
Full14: A:0     B:40    C:1     D:11    BUS_CROSSING+OPEN
Full15: A:30    B:22    C:0     D:2     BUS_CROSSING+CONFLICT_FF
Full16: A:712   B:0     C:0     D:0     BUS_CROSSING+OUTPUT_ANTENNA
Full17: A:32498 B:0     C:0     D:0     BUS_REPEATER+UNUSED
Full18: A:0     B:106   C:10    D:33    BUS_REPEATER+OPEN
Full19: A:70    B:860   C:53    D:252   BUS_REPEATER+CONFLICT_FF
Full20: A:245   B:0     C:0     D:0     BUS_REPEATER+INPUT_ANTENNA
Full21: A:1093  B:0     C:0     D:0     BUS_REPEATER+OUTPUT_ANTENNA
Full22: A:1104  B:0     C:0     D:0     FAKE+UNUSED
Full23: A:11705 B:29    C:0     D:6     UNKNOWN+UNEXPLORED
---- FPGA RESOURCES -----
Cumulative0:    A:17004 B:660   C:0     D:0     LUT
Cumulative1:    A:45992 B:1453  C:0     D:27    CELL_INTERCONN.
Cumulative2:    A:10732 B:283   C:0     D:25    BUS_TO_CELL
Cumulative3:    A:10964 B:62    C:1     D:13    BUS_CROSSING
Cumulative4:    A:33906 B:966   C:63    D:285   BUS_REPEATER
Cumulative5:    A:1104  B:0     C:0     D:0     FAKE
Cumulative6:    A:11705 B:29    C:0     D:6     UNKNOWN
---- FAULT MODELS -----
Cumulative7:    A:102811 B:0    C:0     D:0     UNUSED
Cumulative8:    A:91    B:1089  C:0     D:0     ALTERNATE
Cumulative9:    A:4     B:483   C:11    D:62    OPEN
Cumulative10:   A:14    B:229   C:0     D:9     CONFLICT_OF
Cumulative11:   A:315   B:1600  C:53    D:279   CONFLICT_FF
Cumulative12:   A:245   B:0     C:0     D:0     INPUT_ANTENNA
Cumulative13:   A:16129 B:0     C:0     D:0     OUTPUT_ANTENNA
Cumulative14:   A:93    B:23    C:0     D:0     UNPREDICTABLE
Cumulative15:   A:11705 B:29    C:0     D:6     UNEXPLORED
Result: All: A:131407    B:3453  C:64    D:356
```

b11 without parity

```
Full0:  A:17249 B:0     C:0     D:0     LUT+UNUSED
Full1:  A:30    B:0     C:385   D:0     LUT+ALTERNATE
Full2:  A:32056 B:0     C:0     D:0     CELL_INTERCONN.+UNUSED
Full3:  A:74    B:0     C:300   D:0     CELL_INTERCONN.+ALTERNATE
Full4:  A:3     B:0     C:91    D:0     CELL_INTERCONN.+OPEN
Full5:  A:12    B:0     C:136   D:0     CELL_INTERCONN.+CONFLICT_OF
Full6:  A:194   B:0     C:388   D:0     CELL_INTERCONN.+CONFLICT_FF
Full7:  A:14166 B:0     C:0     D:0     CELL_INTERCONN.+OUTPUT_ANTENNA
Full8:  A:45    B:0     C:7     D:0     CELL_INTERCONN.+UNPREDICTABLE
Full9:  A:10495 B:0     C:0     D:0     BUS_TO_CELL+UNUSED
Full10: A:6     B:0     C:168   D:0     BUS_TO_CELL+OPEN
Full11: A:48    B:0     C:38    D:0     BUS_TO_CELL+CONFLICT_FF
Full12: A:285   B:0     C:0     D:0     BUS_TO_CELL+OUTPUT_ANTENNA
Full13: A:10181 B:0     C:0     D:0     BUS_CROSSING+UNUSED
Full14: A:0     B:0     C:42    D:0     BUS_CROSSING+OPEN
Full15: A:46    B:0     C:17    D:0     BUS_CROSSING+CONFLICT_FF
Full16: A:754   B:0     C:0     D:0     BUS_CROSSING+OUTPUT_ANTENNA
Full17: A:32864 B:0     C:0     D:0     BUS_REPEATER+UNUSED
Full18: A:2     B:0     C:141   D:0     BUS_REPEATER+OPEN
Full19: A:95    B:0     C:1005  D:0     BUS_REPEATER+CONFLICT_FF
Full20: A:203   B:0     C:0     D:0     BUS_REPEATER+INPUT_ANTENNA
Full21: A:910   B:0     C:0     D:0     BUS_REPEATER+OUTPUT_ANTENNA
Full22: A:1104  B:0     C:0     D:0     FAKE+UNUSED
Full23: A:11717 B:0     C:23    D:0     UNKNOWN+UNEXPLORED
---- FPGA RESOURCES -----
Cumulative0:    A:17279 B:0     C:385   D:0     LUT
Cumulative1:    A:46550 B:0     C:922   D:0     CELL_INTERCONN.
Cumulative2:    A:10834 B:0     C:206   D:0     BUS_TO_CELL
Cumulative3:    A:10981 B:0     C:59    D:0     BUS_CROSSING
Cumulative4:    A:34074 B:0     C:1146  D:0     BUS_REPEATER
Cumulative5:    A:1104  B:0     C:0     D:0     FAKE
Cumulative6:    A:11717 B:0     C:23    D:0     UNKNOWN
---- FAULT MODELS -----
Cumulative7:    A:103949 B:0    C:0     D:0     UNUSED
Cumulative8:    A:104   B:0     C:685   D:0     ALTERNATE
Cumulative9:    A:11    B:0     C:442   D:0     OPEN
Cumulative10:   A:12    B:0     C:136   D:0     CONFLICT_OF
Cumulative11:   A:383   B:0     C:1448  D:0     CONFLICT_FF
Cumulative12:   A:203   B:0     C:0     D:0     INPUT_ANTENNA
Cumulative13:   A:16115 B:0     C:0     D:0     OUTPUT_ANTENNA
Cumulative14:   A:45    B:0     C:7     D:0     UNPREDICTABLE
Cumulative15:   A:11717 B:0     C:23    D:0     UNEXPLORED
Result: All: A:132539    B:0     C:2741  D:0
```

b11 with parity

```
Full0:  A:17245 B:0     C:0     D:0     LUT+UNUSED
Full1:  A:54    B:266   C:48    D:51    LUT+ALTERNATE
Full2:  A:32041 B:0     C:0     D:0     CELL_INTERCONN.+UNUSED
Full3:  A:79    B:240   C:21    D:52    CELL_INTERCONN.+ALTERNATE
Full4:  A:1     B:62    C:5     D:8     CELL_INTERCONN.+OPEN
Full5:  A:23    B:120   C:27    D:43    CELL_INTERCONN.+CONFLICT_OF
Full6:  A:197   B:220   C:32    D:85    CELL_INTERCONN.+CONFLICT_FF
Full7:  A:14154 B:0     C:0     D:0     CELL_INTERCONN.+OUTPUT_ANTENNA
Full8:  A:46    B:15    C:0     D:1     CELL_INTERCONN.+UNPREDICTABLE
Full9:  A:10495 B:0     C:0     D:0     BUS_TO_CELL+UNUSED
Full10: A:5     B:111   C:16    D:45    BUS_TO_CELL+OPEN
Full11: A:31    B:21    C:4     D:7     BUS_TO_CELL+CONFLICT_FF
Full12: A:305   B:0     C:0     D:0     BUS_TO_CELL+OUTPUT_ANTENNA
Full13: A:10265 B:0     C:0     D:0     BUS_CROSSING+UNUSED
Full14: A:0     B:25    C:6     D:10    BUS_CROSSING+OPEN
Full15: A:38    B:9     C:9     D:3     BUS_CROSSING+CONFLICT_FF
Full16: A:675   B:0     C:0     D:0     BUS_CROSSING+OUTPUT_ANTENNA
Full17: A:32976 B:0     C:0     D:0     BUS_REPEATER+UNUSED
Full18: A:1     B:81    C:20    D:29    BUS_REPEATER+OPEN
Full19: A:74    B:565   C:138   D:262   BUS_REPEATER+CONFLICT_FF
Full20: A:198   B:0     C:0     D:0     BUS_REPEATER+INPUT_ANTENNA
Full21: A:876   B:0     C:0     D:0     BUS_REPEATER+OUTPUT_ANTENNA
Full22: A:1104  B:0     C:0     D:0     FAKE+UNUSED
Full23: A:11712 B:17    C:0     D:11    UNKNOWN+UNEXPLORED
---- FPGA RESOURCES -----
Cumulative0:    A:17299 B:266   C:48    D:51    LUT
Cumulative1:    A:46541 B:657   C:85    D:189   CELL_INTERCONN.
Cumulative2:    A:10836 B:132   C:20    D:52    BUS_TO_CELL
Cumulative3:    A:10978 B:34    C:15    D:13    BUS_CROSSING
Cumulative4:    A:34125 B:646   C:158   D:291   BUS_REPEATER
Cumulative5:    A:1104  B:0     C:0     D:0     FAKE
Cumulative6:    A:11712 B:17    C:0     D:11    UNKNOWN
---- FAULT MODELS -----
Cumulative7:    A:104126 B:0    C:0     D:0     UNUSED
Cumulative8:    A:133   B:506   C:69    D:103   ALTERNATE
Cumulative9:    A:7     B:279   C:47    D:92    OPEN
Cumulative10:   A:23    B:120   C:27    D:43    CONFLICT_OF
Cumulative11:   A:340   B:815   C:183   D:357   CONFLICT_FF
Cumulative12:   A:198   B:0     C:0     D:0     INPUT_ANTENNA
Cumulative13:   A:16010 B:0     C:0     D:0     OUTPUT_ANTENNA
Cumulative14:   A:46    B:15    C:0     D:1     UNPREDICTABLE
Cumulative15:   A:11712 B:17    C:0     D:11    UNEXPLORED
Result: All: A:132595   B:1752  C:326   D:607
```

b12 without parity

```
Full0:  A:16976 B:0     C:0     D:0     LUT+UNUSED
Full1:  A:121   B:0     C:567   D:0     LUT+ALTERNATE
Full2:  A:31695 B:0     C:0     D:0     CELL_INTERCONN.+UNUSED
Full3:  A:102   B:0     C:407   D:0     CELL_INTERCONN.+ALTERNATE
Full4:  A:4     B:0     C:108   D:0     CELL_INTERCONN.+OPEN
Full5:  A:37    B:0     C:285   D:0     CELL_INTERCONN.+CONFLICT_OF
Full6:  A:276   B:0     C:536   D:0     CELL_INTERCONN.+CONFLICT_FF
Full7:  A:13937 B:0     C:0     D:0     CELL_INTERCONN.+OUTPUT_ANTENNA
Full8:  A:38    B:0     C:47    D:0     CELL_INTERCONN.+UNPREDICTABLE
Full9:  A:10387 B:0     C:0     D:0     BUS_TO_CELL+UNUSED
Full10: A:8     B:0     C:234   D:0     BUS_TO_CELL+OPEN
Full11: A:45    B:0     C:75    D:0     BUS_TO_CELL+CONFLICT_FF
Full12: A:291   B:0     C:0     D:0     BUS_TO_CELL+OUTPUT_ANTENNA
Full13: A:10221 B:0     C:0     D:0     BUS_CROSSING+UNUSED
Full14: A:0     B:0     C:44    D:0     BUS_CROSSING+OPEN
Full15: A:29    B:0     C:22    D:0     BUS_CROSSING+CONFLICT_FF
Full16: A:724   B:0     C:0     D:0     BUS_CROSSING+OUTPUT_ANTENNA
Full17: A:32616 B:0     C:0     D:0     BUS_REPEATER+UNUSED
Full18: A:1     B:0     C:134   D:0     BUS_REPEATER+OPEN
Full19: A:54    B:0     C:1145  D:0     BUS_REPEATER+CONFLICT_FF
Full20: A:223   B:0     C:0     D:0     BUS_REPEATER+INPUT_ANTENNA
Full21: A:1047  B:0     C:0     D:0     BUS_REPEATER+OUTPUT_ANTENNA
Full22: A:1104  B:0     C:0     D:0     FAKE+UNUSED
Full23: A:11709 B:0     C:31    D:0     UNKNOWN+UNEXPLORED
---- FPGA RESOURCES -----
Cumulative0:    A:17097 B:0     C:567   D:0     LUT
Cumulative1:    A:46081 B:0     C:1391  D:0     CELL_INTERCONN.
Cumulative2:    A:10731 B:0     C:309   D:0     BUS_TO_CELL
Cumulative3:    A:10974 B:0     C:66    D:0     BUS_CROSSING
Cumulative4:    A:33941 B:0     C:1279  D:0     BUS_REPEATER
Cumulative5:    A:1104  B:0     C:0     D:0     FAKE
Cumulative6:    A:11709 B:0     C:31    D:0     UNKNOWN
---- FAULT MODELS -----
Cumulative7:    A:102999 B:0    C:0     D:0     UNUSED
Cumulative8:    A:223   B:0     C:974   D:0     ALTERNATE
Cumulative9:    A:13    B:0     C:520   D:0     OPEN
Cumulative10:   A:37    B:0     C:285   D:0     CONFLICT_OF
Cumulative11:   A:404   B:0     C:1778  D:0     CONFLICT_FF
Cumulative12:   A:223   B:0     C:0     D:0     INPUT_ANTENNA
Cumulative13:   A:15999 B:0     C:0     D:0     OUTPUT_ANTENNA
Cumulative14:   A:38    B:0     C:47    D:0     UNPREDICTABLE
Cumulative15:   A:11709 B:0     C:31    D:0     UNEXPLORED
Result: All: A:131645    B:0     C:3635  D:0
```

b12 with parity

```
Full0:  A:17014 B:0     C:0     D:0     LUT+UNUSED
Full1:  A:71    B:561   C:0     D:18    LUT+ALTERNATE
Full2:  A:31746 B:0     C:0     D:0     CELL_INTERCONN.+UNUSED
Full3:  A:85    B:361   C:0     D:22    CELL_INTERCONN.+ALTERNATE
Full4:  A:1     B:103   C:0     D:4     CELL_INTERCONN.+OPEN
Full5:  A:38    B:206   C:0     D:29    CELL_INTERCONN.+CONFLICT_OF
Full6:  A:229   B:492   C:0     D:31    CELL_INTERCONN.+CONFLICT_FF
Full7:  A:14037 B:0     C:0     D:0     CELL_INTERCONN.+OUTPUT_ANTENNA
Full8:  A:45    B:41    C:0     D:2     CELL_INTERCONN.+UNPREDICTABLE
Full9:  A:10361 B:0     C:0     D:0     BUS_TO_CELL+UNUSED
Full10: A:7     B:191   C:0     D:27    BUS_TO_CELL+OPEN
Full11: A:34    B:55    C:0     D:2     BUS_TO_CELL+CONFLICT_FF
Full12: A:363   B:0     C:0     D:0     BUS_TO_CELL+OUTPUT_ANTENNA
Full13: A:10265 B:0     C:0     D:0     BUS_CROSSING+UNUSED
Full14: A:0     B:25    C:1     D:7     BUS_CROSSING+OPEN
Full15: A:26    B:12    C:0     D:6     BUS_CROSSING+CONFLICT_FF
Full16: A:698   B:0     C:0     D:0     BUS_CROSSING+OUTPUT_ANTENNA
Full17: A:32720 B:0     C:0     D:0     BUS_REPEATER+UNUSED
Full18: A:1     B:97    C:9     D:36    BUS_REPEATER+OPEN
Full19: A:73    B:754   C:48    D:270   BUS_REPEATER+CONFLICT_FF
Full20: A:227   B:0     C:0     D:0     BUS_REPEATER+INPUT_ANTENNA
Full21: A:985   B:0     C:0     D:0     BUS_REPEATER+OUTPUT_ANTENNA
Full22: A:1104  B:0     C:0     D:0     FAKE+UNUSED
Full23: A:11708 B:24    C:0     D:8     UNKNOWN+UNEXPLORED
---- FPGA RESOURCES -----
Cumulative0:    A:17085 B:561   C:0     D:18    LUT
Cumulative1:    A:46172 B:1209  C:0     D:91    CELL_INTERCONN.
Cumulative2:    A:10765 B:246   C:0     D:29    BUS_TO_CELL
Cumulative3:    A:10989 B:37    C:1     D:13    BUS_CROSSING
Cumulative4:    A:34006 B:851   C:57    D:306   BUS_REPEATER
Cumulative5:    A:1104  B:0     C:0     D:0     FAKE
Cumulative6:    A:11708 B:24    C:0     D:8     UNKNOWN
---- FAULT MODELS -----
Cumulative7:    A:103210 B:0    C:0     D:0     UNUSED
Cumulative8:    A:156   B:922   C:0     D:40    ALTERNATE
Cumulative9:    A:9     B:416   C:10    D:74    OPEN
Cumulative10:   A:38    B:206   C:0     D:29    CONFLICT_OF
Cumulative11:   A:362   B:1313  C:48    D:309   CONFLICT_FF
Cumulative12:   A:227   B:0     C:0     D:0     INPUT_ANTENNA
Cumulative13:   A:16083 B:0     C:0     D:0     OUTPUT_ANTENNA
Cumulative14:   A:45    B:41    C:0     D:2     UNPREDICTABLE
Cumulative15:   A:11708 B:24    C:0     D:8     UNEXPLORED
Result: All: A:131838   B:2922  C:58    D:462
```

s1494 without parity

```
Full0:  A:14164 B:0     C:0     D:0     LUT+UNUSED
Full1:  A:614   B:0     C:2886  D:0     LUT+ALTERNATE
Full2:  A:24836 B:0     C:0     D:0     CELL_INTERCONN.+UNUSED
Full3:  A:671   B:0     C:2316  D:0     CELL_INTERCONN.+ALTERNATE
Full4:  A:5     B:0     C:617   D:0     CELL_INTERCONN.+OPEN
Full5:  A:178   B:0     C:1568  D:0     CELL_INTERCONN.+CONFLICT_OF
Full6:  A:1616  B:0     C:2727  D:0     CELL_INTERCONN.+CONFLICT_FF
Full7:  A:12447 B:0     C:0     D:0     CELL_INTERCONN.+OUTPUT_ANTENNA
Full8:  A:359   B:0     C:132   D:0     CELL_INTERCONN.+UNPREDICTABLE
Full9:  A:6855  B:0     C:0     D:0     BUS_TO_CELL+UNUSED
Full10: A:23    B:0     C:1372  D:0     BUS_TO_CELL+OPEN
Full11: A:289   B:0     C:473   D:0     BUS_TO_CELL+CONFLICT_FF
Full12: A:2028  B:0     C:0     D:0     BUS_TO_CELL+OUTPUT_ANTENNA
Full13: A:7274  B:0     C:0     D:0     BUS_CROSSING+UNUSED
Full14: A:0     B:0     C:300   D:0     BUS_CROSSING+OPEN
Full15: A:545   B:0     C:318   D:0     BUS_CROSSING+CONFLICT_FF
Full16: A:2603  B:0     C:0     D:0     BUS_CROSSING+OUTPUT_ANTENNA
Full17: A:21374 B:0     C:0     D:0     BUS_REPEATER+UNUSED
Full18: A:0     B:0     C:890   D:0     BUS_REPEATER+OPEN
Full19: A:766   B:0     C:7157  D:0     BUS_REPEATER+CONFLICT_FF
Full20: A:868   B:0     C:0     D:0     BUS_REPEATER+INPUT_ANTENNA
Full21: A:4165  B:0     C:0     D:0     BUS_REPEATER+OUTPUT_ANTENNA
Full22: A:1104  B:0     C:0     D:0     FAKE+UNUSED
Full23: A:11365 B:0     C:375   D:0     UNKNOWN+UNEXPLORED
---- FPGA RESOURCES -----
Cumulative0:    A:14778 B:0     C:2886  D:0     LUT
Cumulative1:    A:40112 B:0     C:7360  D:0     CELL_INTERCONN.
Cumulative2:    A:9195  B:0     C:1845  D:0     BUS_TO_CELL
Cumulative3:    A:10422 B:0     C:618   D:0     BUS_CROSSING
Cumulative4:    A:27173 B:0     C:8047  D:0     BUS_REPEATER
Cumulative5:    A:1104  B:0     C:0     D:0     FAKE
Cumulative6:    A:11365 B:0     C:375   D:0     UNKNOWN
---- FAULT MODELS -----
Cumulative7:    A:75607 B:0     C:0     D:0     UNUSED
Cumulative8:    A:1285  B:0     C:5202  D:0     ALTERNATE
Cumulative9:    A:28    B:0     C:3179  D:0     OPEN
Cumulative10:   A:178   B:0     C:1568  D:0     CONFLICT_OF
Cumulative11:   A:3216  B:0     C:10675 D:0     CONFLICT_FF
Cumulative12:   A:868   B:0     C:0     D:0     INPUT_ANTENNA
Cumulative13:   A:21243 B:0     C:0     D:0     OUTPUT_ANTENNA
Cumulative14:   A:359   B:0     C:132   D:0     UNPREDICTABLE
Cumulative15:   A:11365 B:0     C:375   D:0     UNEXPLORED
Result: All: A:114149    B:0     C:21131 D:0
```

s1494 with parity

```
Full0:  A:13622 B:0     C:0     D:0     LUT+UNUSED
Full1:  A:665   B:2839  C:69    D:469   LUT+ALTERNATE
Full2:  A:23814 B:0     C:0     D:0     CELL_INTERCONN.+UNUSED
Full3:  A:727   B:2110  C:60    D:383   CELL_INTERCONN.+ALTERNATE
Full4:  A:6     B:579   C:19    D:97    CELL_INTERCONN.+OPEN
Full5:  A:237   B:1439  C:47    D:265   CELL_INTERCONN.+CONFLICT_OF
Full6:  A:1773  B:2556  C:76    D:547   CELL_INTERCONN.+CONFLICT_FF
Full7:  A:12184 B:0     C:0     D:0     CELL_INTERCONN.+OUTPUT_ANTENNA
Full8:  A:428   B:95    C:4     D:26    CELL_INTERCONN.+UNPREDICTABLE
Full9:  A:6275  B:0     C:0     D:0     BUS_TO_CELL+UNUSED
Full10: A:36    B:1166  C:38    D:284   BUS_TO_CELL+OPEN
Full11: A:313   B:390   C:6     D:90    BUS_TO_CELL+CONFLICT_FF
Full12: A:2442  B:0     C:0     D:0     BUS_TO_CELL+OUTPUT_ANTENNA
Full13: A:6289  B:0     C:0     D:0     BUS_CROSSING+UNUSED
Full14: A:0     B:237   C:7     D:129   BUS_CROSSING+OPEN
Full15: A:844   B:298   C:8     D:180   BUS_CROSSING+CONFLICT_FF
Full16: A:3048  B:0     C:0     D:0     BUS_CROSSING+OUTPUT_ANTENNA
Full17: A:19182 B:0     C:0     D:0     BUS_REPEATER+UNUSED
Full18: A:0     B:709   C:19    D:339   BUS_REPEATER+OPEN
Full19: A:967   B:5521  C:167   D:2794  BUS_REPEATER+CONFLICT_FF
Full20: A:968   B:0     C:0     D:0     BUS_REPEATER+INPUT_ANTENNA
Full21: A:4554  B:0     C:0     D:0     BUS_REPEATER+OUTPUT_ANTENNA
Full22: A:1104  B:0     C:0     D:0     FAKE+UNUSED
Full23: A:11391 B:247   C:9     D:93    UNKNOWN+UNEXPLORED
---- FPGA RESOURCES -----
Cumulative0:    A:14287 B:2839  C:69    D:469   LUT
Cumulative1:    A:39169 B:6779  C:206   D:1318  CELL_INTERCONN.
Cumulative2:    A:9066  B:1556  C:44    D:374   BUS_TO_CELL
Cumulative3:    A:10181 B:535   C:15    D:309   BUS_CROSSING
Cumulative4:    A:25671 B:6230  C:186   D:3133  BUS_REPEATER
Cumulative5:    A:1104  B:0     C:0     D:0     FAKE
Cumulative6:    A:11391 B:247   C:9     D:93    UNKNOWN
---- FAULT MODELS -----
Cumulative7:    A:70286 B:0     C:0     D:0     UNUSED
Cumulative8:    A:1392  B:4949  C:129   D:852   ALTERNATE
Cumulative9:    A:42    B:2691  C:83    D:849   OPEN
Cumulative10:   A:237   B:1439  C:47    D:265   CONFLICT_OF
Cumulative11:   A:3897  B:8765  C:257   D:3611  CONFLICT_FF
Cumulative12:   A:968   B:0     C:0     D:0     INPUT_ANTENNA
Cumulative13:   A:22228 B:0     C:0     D:0     OUTPUT_ANTENNA
Cumulative14:   A:428   B:95    C:4     D:26    UNPREDICTABLE
Cumulative15:   A:11391 B:247   C:9     D:93    UNEXPLORED
Result: All: A:110869    B:18186 C:529   D:5696
```

s386 without parity

```
Full0:  A:16998 B:0     C:0     D:0     LUT+UNUSED
Full1:  A:137   B:0     C:529   D:0     LUT+ALTERNATE
Full2:  A:31585 B:0     C:0     D:0     CELL_INTERCONN.+UNUSED
Full3:  A:101   B:0     C:413   D:0     CELL_INTERCONN.+ALTERNATE
Full4:  A:0     B:0     C:113   D:0     CELL_INTERCONN.+OPEN
Full5:  A:35    B:0     C:270   D:0     CELL_INTERCONN.+CONFLICT_OF
Full6:  A:285   B:0     C:543   D:0     CELL_INTERCONN.+CONFLICT_FF
Full7:  A:14033 B:0     C:0     D:0     CELL_INTERCONN.+OUTPUT_ANTENNA
Full8:  A:72    B:0     C:22    D:0     CELL_INTERCONN.+UNPREDICTABLE
Full9:  A:10276 B:0     C:0     D:0     BUS_TO_CELL+UNUSED
Full10: A:8     B:0     C:261   D:0     BUS_TO_CELL+OPEN
Full11: A:44    B:0     C:69    D:0     BUS_TO_CELL+CONFLICT_FF
Full12: A:382   B:0     C:0     D:0     BUS_TO_CELL+OUTPUT_ANTENNA
Full13: A:10067 B:0     C:0     D:0     BUS_CROSSING+UNUSED
Full14: A:0     B:0     C:54    D:0     BUS_CROSSING+OPEN
Full15: A:27    B:0     C:13    D:0     BUS_CROSSING+CONFLICT_FF
Full16: A:879   B:0     C:0     D:0     BUS_CROSSING+OUTPUT_ANTENNA
Full17: A:32108 B:0     C:0     D:0     BUS_REPEATER+UNUSED
Full18: A:0     B:0     C:162   D:0     BUS_REPEATER+OPEN
Full19: A:100   B:0     C:1403  D:0     BUS_REPEATER+CONFLICT_FF
Full20: A:237   B:0     C:0     D:0     BUS_REPEATER+INPUT_ANTENNA
Full21: A:1210  B:0     C:0     D:0     BUS_REPEATER+OUTPUT_ANTENNA
Full22: A:1104  B:0     C:0     D:0     FAKE+UNUSED
Full23: A:11691 B:0     C:49    D:0     UNKNOWN+UNEXPLORED
---- FPGA RESOURCES -----
Cumulative0:    A:17135 B:0     C:529   D:0     LUT
Cumulative1:    A:46111 B:0     C:1361  D:0     CELL_INTERCONN.
Cumulative2:    A:10710 B:0     C:330   D:0     BUS_TO_CELL
Cumulative3:    A:10973 B:0     C:67    D:0     BUS_CROSSING
Cumulative4:    A:33655 B:0     C:1565  D:0     BUS_REPEATER
Cumulative5:    A:1104  B:0     C:0     D:0     FAKE
Cumulative6:    A:11691 B:0     C:49    D:0     UNKNOWN
---- FAULT MODELS -----
Cumulative7:    A:102138 B:0    C:0     D:0     UNUSED
Cumulative8:    A:238   B:0     C:942   D:0     ALTERNATE
Cumulative9:    A:8     B:0     C:590   D:0     OPEN
Cumulative10:   A:35    B:0     C:270   D:0     CONFLICT_OF
Cumulative11:   A:456   B:0     C:2028  D:0     CONFLICT_FF
Cumulative12:   A:237   B:0     C:0     D:0     INPUT_ANTENNA
Cumulative13:   A:16504 B:0     C:0     D:0     OUTPUT_ANTENNA
Cumulative14:   A:72    B:0     C:22    D:0     UNPREDICTABLE
Cumulative15:   A:11691 B:0     C:49    D:0     UNEXPLORED
Result: All: A:131379    B:0     C:3901  D:0
```

s386 with parity

```
Full0:  A:16654 B:0     C:0     D:0     LUT+UNUSED
Full1:  A:194   B:651   C:25    D:140   LUT+ALTERNATE
Full2:  A:30978 B:0     C:0     D:0     CELL_INTERCONN.+UNUSED
Full3:  A:122   B:447   C:24    D:89    CELL_INTERCONN.+ALTERNATE
Full4:  A:4     B:134   C:6     D:32    CELL_INTERCONN.+OPEN
Full5:  A:35    B:298   C:17    D:62    CELL_INTERCONN.+CONFLICT_OF
Full6:  A:432   B:519   C:26    D:134   CELL_INTERCONN.+CONFLICT_FF
Full7:  A:13968 B:0     C:0     D:0     CELL_INTERCONN.+OUTPUT_ANTENNA
Full8:  A:124   B:13    C:2     D:6     CELL_INTERCONN.+UNPREDICTABLE
Full9:  A:9948  B:0     C:0     D:0     BUS_TO_CELL+UNUSED
Full10: A:6     B:279   C:13    D:87    BUS_TO_CELL+OPEN
Full11: A:44    B:87    C:1     D:32    BUS_TO_CELL+CONFLICT_FF
Full12: A:543   B:0     C:0     D:0     BUS_TO_CELL+OUTPUT_ANTENNA
Full13: A:10005 B:0     C:0     D:0     BUS_CROSSING+UNUSED
Full14: A:0     B:34    C:3     D:23    BUS_CROSSING+OPEN
Full15: A:42    B:11    C:0     D:12    BUS_CROSSING+CONFLICT_FF
Full16: A:910   B:0     C:0     D:0     BUS_CROSSING+OUTPUT_ANTENNA
Full17: A:31414 B:0     C:0     D:0     BUS_REPEATER+UNUSED
Full18: A:0     B:122   C:11    D:73    BUS_REPEATER+OPEN
Full19: A:132   B:999   C:101   D:621   BUS_REPEATER+CONFLICT_FF
Full20: A:290   B:0     C:0     D:0     BUS_REPEATER+INPUT_ANTENNA
Full21: A:1457  B:0     C:0     D:0     BUS_REPEATER+OUTPUT_ANTENNA
Full22: A:1104  B:0     C:0     D:0     FAKE+UNUSED
Full23: A:11671 B:44    C:0     D:25    UNKNOWN+UNEXPLORED
---- FPGA RESOURCES -----
Cumulative0:    A:16848 B:651   C:25    D:140   LUT
Cumulative1:    A:45663 B:1411  C:75    D:323   CELL_INTERCONN.
Cumulative2:    A:10541 B:366   C:14    D:119   BUS_TO_CELL
Cumulative3:    A:10957 B:45    C:3     D:35    BUS_CROSSING
Cumulative4:    A:33293 B:1121  C:112   D:694   BUS_REPEATER
Cumulative5:    A:1104  B:0     C:0     D:0     FAKE
Cumulative6:    A:11671 B:44    C:0     D:25    UNKNOWN
---- FAULT MODELS -----
Cumulative7:    A:100103 B:0    C:0     D:0     UNUSED
Cumulative8:    A:316   B:1098  C:49    D:229   ALTERNATE
Cumulative9:    A:10    B:569   C:33    D:215   OPEN
Cumulative10:   A:35    B:298   C:17    D:62    CONFLICT_OF
Cumulative11:   A:650   B:1616  C:128   D:799   CONFLICT_FF
Cumulative12:   A:290   B:0     C:0     D:0     INPUT_ANTENNA
Cumulative13:   A:16878 B:0     C:0     D:0     OUTPUT_ANTENNA
Cumulative14:   A:124   B:13    C:2     D:6     UNPREDICTABLE
Cumulative15:   A:11671 B:44    C:0     D:25    UNEXPLORED
Result: All: A:130077    B:3638  C:229   D:1336
```

br1 without parity

```
Full0:  A:17016 B:0     C:0     D:0     LUT+UNUSED
Full1:  A:112   B:0     C:536   D:0     LUT+ALTERNATE
Full2:  A:31626 B:0     C:0     D:0     CELL_INTERCONN.+UNUSED
Full3:  A:94    B:0     C:415   D:0     CELL_INTERCONN.+ALTERNATE
Full4:  A:1     B:0     C:125   D:0     CELL_INTERCONN.+OPEN
Full5:  A:32    B:0     C:231   D:0     CELL_INTERCONN.+CONFLICT_OF
Full6:  A:298   B:0     C:491   D:0     CELL_INTERCONN.+CONFLICT_FF
Full7:  A:14056 B:0     C:0     D:0     CELL_INTERCONN.+OUTPUT_ANTENNA
Full8:  A:87    B:0     C:16    D:0     CELL_INTERCONN.+UNPREDICTABLE
Full9:  A:10352 B:0     C:0     D:0     BUS_TO_CELL+UNUSED
Full10: A:8     B:0     C:244   D:0     BUS_TO_CELL+OPEN
Full11: A:40    B:0     C:64    D:0     BUS_TO_CELL+CONFLICT_FF
Full12: A:332   B:0     C:0     D:0     BUS_TO_CELL+OUTPUT_ANTENNA
Full13: A:10255 B:0     C:0     D:0     BUS_CROSSING+UNUSED
Full14: A:0     B:0     C:37    D:0     BUS_CROSSING+OPEN
Full15: A:24    B:0     C:8     D:0     BUS_CROSSING+CONFLICT_FF
Full16: A:716   B:0     C:0     D:0     BUS_CROSSING+OUTPUT_ANTENNA
Full17: A:32488 B:0     C:0     D:0     BUS_REPEATER+UNUSED
Full18: A:1     B:0     C:137   D:0     BUS_REPEATER+OPEN
Full19: A:90    B:0     C:1175  D:0     BUS_REPEATER+CONFLICT_FF
Full20: A:220   B:0     C:0     D:0     BUS_REPEATER+INPUT_ANTENNA
Full21: A:1109  B:0     C:0     D:0     BUS_REPEATER+OUTPUT_ANTENNA
Full22: A:1104  B:0     C:0     D:0     FAKE+UNUSED
Full23: A:11703 B:0     C:37    D:0     UNKNOWN+UNEXPLORED
---- FPGA RESOURCES -----
Cumulative0:    A:17128 B:0     C:536   D:0     LUT
Cumulative1:    A:46194 B:0     C:1278  D:0     CELL_INTERCONN.
Cumulative2:    A:10732 B:0     C:308   D:0     BUS_TO_CELL
Cumulative3:    A:10995 B:0     C:45    D:0     BUS_CROSSING
Cumulative4:    A:33908 B:0     C:1312  D:0     BUS_REPEATER
Cumulative5:    A:1104  B:0     C:0     D:0     FAKE
Cumulative6:    A:11703 B:0     C:37    D:0     UNKNOWN
---- FAULT MODELS -----
Cumulative7:    A:102841 B:0    C:0     D:0     UNUSED
Cumulative8:    A:206   B:0     C:951   D:0     ALTERNATE
Cumulative9:    A:10    B:0     C:543   D:0     OPEN
Cumulative10:   A:32    B:0     C:231   D:0     CONFLICT_OF
Cumulative11:   A:452   B:0     C:1738  D:0     CONFLICT_FF
Cumulative12:   A:220   B:0     C:0     D:0     INPUT_ANTENNA
Cumulative13:   A:16213 B:0     C:0     D:0     OUTPUT_ANTENNA
Cumulative14:   A:87    B:0     C:16    D:0     UNPREDICTABLE
Cumulative15:   A:11703 B:0     C:37    D:0     UNEXPLORED
Result: All: A:131764    B:0     C:3516  D:0
```

br1 with parity

```
Full0:  A:16842 B:0     C:0     D:0     LUT+UNUSED
Full1:  A:149   B:458   C:69    D:146   LUT+ALTERNATE
Full2:  A:31336 B:0     C:0     D:0     CELL_INTERCONN.+UNUSED
Full3:  A:123   B:331   C:53    D:104   CELL_INTERCONN.+ALTERNATE
Full4:  A:1     B:103   C:15    D:33    CELL_INTERCONN.+OPEN
Full5:  A:40    B:176   C:21    D:87    CELL_INTERCONN.+CONFLICT_OF
Full6:  A:396   B:369   C:62    D:159   CELL_INTERCONN.+CONFLICT_FF
Full7:  A:13964 B:0     C:0     D:0     CELL_INTERCONN.+OUTPUT_ANTENNA
Full8:  A:88    B:8     C:2     D:1     CELL_INTERCONN.+UNPREDICTABLE
Full9:  A:10144 B:0     C:0     D:0     BUS_TO_CELL+UNUSED
Full10: A:2     B:172   C:36    D:93    BUS_TO_CELL+OPEN
Full11: A:60    B:45    C:7     D:29    BUS_TO_CELL+CONFLICT_FF
Full12: A:452   B:0     C:0     D:0     BUS_TO_CELL+OUTPUT_ANTENNA
Full13: A:10083 B:0     C:0     D:0     BUS_CROSSING+UNUSED
Full14: A:0     B:24    C:6     D:17    BUS_CROSSING+OPEN
Full15: A:50    B:13    C:6     D:10    BUS_CROSSING+CONFLICT_FF
Full16: A:831   B:0     C:0     D:0     BUS_CROSSING+OUTPUT_ANTENNA
Full17: A:31930 B:0     C:0     D:0     BUS_REPEATER+UNUSED
Full18: A:0     B:89    C:10    D:89    BUS_REPEATER+OPEN
Full19: A:110   B:668   C:98    D:683   BUS_REPEATER+CONFLICT_FF
Full20: A:281   B:0     C:0     D:0     BUS_REPEATER+INPUT_ANTENNA
Full21: A:1262  B:0     C:0     D:0     BUS_REPEATER+OUTPUT_ANTENNA
Full22: A:1104  B:0     C:0     D:0     FAKE+UNUSED
Full23: A:11680 B:28    C:3     D:29    UNKNOWN+UNEXPLORED
---- FPGA RESOURCES -----
Cumulative0:    A:16991 B:458   C:69    D:146   LUT
Cumulative1:    A:45948 B:987   C:153   D:384   CELL_INTERCONN.
Cumulative2:    A:10658 B:217   C:43    D:122   BUS_TO_CELL
Cumulative3:    A:10964 B:37    C:12    D:27    BUS_CROSSING
Cumulative4:    A:33583 B:757   C:108   D:772   BUS_REPEATER
Cumulative5:    A:1104  B:0     C:0     D:0     FAKE
Cumulative6:    A:11680 B:28    C:3     D:29    UNKNOWN
---- FAULT MODELS -----
Cumulative7:    A:101439 B:0    C:0     D:0     UNUSED
Cumulative8:    A:272   B:789   C:122   D:250   ALTERNATE
Cumulative9:    A:3     B:388   C:67    D:232   OPEN
Cumulative10:   A:40    B:176   C:21    D:87    CONFLICT_OF
Cumulative11:   A:616   B:1095  C:173   D:881   CONFLICT_FF
Cumulative12:   A:281   B:0     C:0     D:0     INPUT_ANTENNA
Cumulative13:   A:16509 B:0     C:0     D:0     OUTPUT_ANTENNA
Cumulative14:   A:88    B:8     C:2     D:1     UNPREDICTABLE
Cumulative15:   A:11680 B:28    C:3     D:29    UNEXPLORED
Result: All: A:130928    B:2484  C:388   D:1480
```

apla without parity

```
Full0:  A:17110 B:0     C:0     D:0     LUT+UNUSED
Full1:  A:90    B:0     C:464   D:0     LUT+ALTERNATE
Full2:  A:31799 B:0     C:0     D:0     CELL_INTERCONN.+UNUSED
Full3:  A:82    B:0     C:367   D:0     CELL_INTERCONN.+ALTERNATE
Full4:  A:0     B:0     C:113   D:0     CELL_INTERCONN.+OPEN
Full5:  A:21    B:0     C:201   D:0     CELL_INTERCONN.+CONFLICT_OF
Full6:  A:227   B:0     C:478   D:0     CELL_INTERCONN.+CONFLICT_FF
Full7:  A:14101 B:0     C:0     D:0     CELL_INTERCONN.+OUTPUT_ANTENNA
Full8:  A:70    B:0     C:13    D:0     CELL_INTERCONN.+UNPREDICTABLE
Full9:  A:10401 B:0     C:0     D:0     BUS_TO_CELL+UNUSED
Full10: A:6     B:0     C:220   D:0     BUS_TO_CELL+OPEN
Full11: A:43    B:0     C:74    D:0     BUS_TO_CELL+CONFLICT_FF
Full12: A:296   B:0     C:0     D:0     BUS_TO_CELL+OUTPUT_ANTENNA
Full13: A:10252 B:0     C:0     D:0     BUS_CROSSING+UNUSED
Full14: A:0     B:0     C:45    D:0     BUS_CROSSING+OPEN
Full15: A:31    B:0     C:23    D:0     BUS_CROSSING+CONFLICT_FF
Full16: A:689   B:0     C:0     D:0     BUS_CROSSING+OUTPUT_ANTENNA
Full17: A:32564 B:0     C:0     D:0     BUS_REPEATER+UNUSED
Full18: A:0     B:0     C:142   D:0     BUS_REPEATER+OPEN
Full19: A:102   B:0     C:1123  D:0     BUS_REPEATER+CONFLICT_FF
Full20: A:223   B:0     C:0     D:0     BUS_REPEATER+INPUT_ANTENNA
Full21: A:1066  B:0     C:0     D:0     BUS_REPEATER+OUTPUT_ANTENNA
Full22: A:1104  B:0     C:0     D:0     FAKE+UNUSED
Full23: A:11697 B:0     C:43    D:0     UNKNOWN+UNEXPLORED
---- FPGA RESOURCES -----
Cumulative0:    A:17200 B:0     C:464   D:0     LUT
Cumulative1:    A:46300 B:0     C:1172  D:0     CELL_INTERCONN.
Cumulative2:    A:10746 B:0     C:294   D:0     BUS_TO_CELL
Cumulative3:    A:10972 B:0     C:68    D:0     BUS_CROSSING
Cumulative4:    A:33955 B:0     C:1265  D:0     BUS_REPEATER
Cumulative5:    A:1104  B:0     C:0     D:0     FAKE
Cumulative6:    A:11697 B:0     C:43    D:0     UNKNOWN
---- FAULT MODELS -----
Cumulative7:    A:103230 B:0    C:0     D:0     UNUSED
Cumulative8:    A:172   B:0     C:831   D:0     ALTERNATE
Cumulative9:    A:6     B:0     C:520   D:0     OPEN
Cumulative10:   A:21    B:0     C:201   D:0     CONFLICT_OF
Cumulative11:   A:403   B:0     C:1698  D:0     CONFLICT_FF
Cumulative12:   A:223   B:0     C:0     D:0     INPUT_ANTENNA
Cumulative13:   A:16152 B:0     C:0     D:0     OUTPUT_ANTENNA
Cumulative14:   A:70    B:0     C:13    D:0     UNPREDICTABLE
Cumulative15:   A:11697 B:0     C:43    D:0     UNEXPLORED
Result: All: A:131974    B:0     C:3306  D:0
```

apla with parity

```
Full0:  A:16742 B:0     C:0     D:0     LUT+UNUSED
Full1:  A:145   B:635   C:5     D:137   LUT+ALTERNATE
Full2:  A:31074 B:0     C:0     D:0     CELL_INTERCONN.+UNUSED
Full3:  A:123   B:475   C:1     D:96    CELL_INTERCONN.+ALTERNATE
Full4:  A:5     B:130   C:0     D:25    CELL_INTERCONN.+OPEN
Full5:  A:50    B:283   C:0     D:70    CELL_INTERCONN.+CONFLICT_OF
Full6:  A:403   B:570   C:4     D:156   CELL_INTERCONN.+CONFLICT_FF
Full7:  A:13902 B:0     C:0     D:0     CELL_INTERCONN.+OUTPUT_ANTENNA
Full8:  A:92    B:10    C:0     D:3     CELL_INTERCONN.+UNPREDICTABLE
Full9:  A:10073 B:0     C:0     D:0     BUS_TO_CELL+UNUSED
Full10: A:10    B:245   C:3     D:78    BUS_TO_CELL+OPEN
Full11: A:74    B:85    C:1     D:13    BUS_TO_CELL+CONFLICT_FF
Full12: A:458   B:0     C:0     D:0     BUS_TO_CELL+OUTPUT_ANTENNA
Full13: A:9795  B:0     C:0     D:0     BUS_CROSSING+UNUSED
Full14: A:0     B:43    C:2     D:34    BUS_CROSSING+OPEN
Full15: A:40    B:17    C:1     D:7     BUS_CROSSING+CONFLICT_FF
Full16: A:1101  B:0     C:0     D:0     BUS_CROSSING+OUTPUT_ANTENNA
Full17: A:31314 B:0     C:0     D:0     BUS_REPEATER+UNUSED
Full18: A:0     B:120   C:6     D:82    BUS_REPEATER+OPEN
Full19: A:150   B:1008  C:55    D:658   BUS_REPEATER+CONFLICT_FF
Full20: A:300   B:0     C:0     D:0     BUS_REPEATER+INPUT_ANTENNA
Full21: A:1527  B:0     C:0     D:0     BUS_REPEATER+OUTPUT_ANTENNA
Full22: A:1104  B:0     C:0     D:0     FAKE+UNUSED
Full23: A:11683 B:44    C:1     D:12    UNKNOWN+UNEXPLORED
---- FPGA RESOURCES -----
Cumulative0:    A:16887 B:635   C:5     D:137   LUT
Cumulative1:    A:45649 B:1468  C:5     D:350   CELL_INTERCONN.
Cumulative2:    A:10615 B:330   C:4     D:91    BUS_TO_CELL
Cumulative3:    A:10936 B:60    C:3     D:41    BUS_CROSSING
Cumulative4:    A:33291 B:1128  C:61    D:740   BUS_REPEATER
Cumulative5:    A:1104  B:0     C:0     D:0     FAKE
Cumulative6:    A:11683 B:44    C:1     D:12    UNKNOWN
---- FAULT MODELS -----
Cumulative7:    A:100102 B:0    C:0     D:0     UNUSED
Cumulative8:    A:268   B:1110  C:6     D:233   ALTERNATE
Cumulative9:    A:15    B:538   C:11    D:219   OPEN
Cumulative10:   A:50    B:283   C:0     D:70    CONFLICT_OF
Cumulative11:   A:667   B:1680  C:61    D:834   CONFLICT_FF
Cumulative12:   A:300   B:0     C:0     D:0     INPUT_ANTENNA
Cumulative13:   A:16988 B:0     C:0     D:0     OUTPUT_ANTENNA
Cumulative14:   A:92    B:10    C:0     D:3     UNPREDICTABLE
Cumulative15:   A:11683 B:44    C:1     D:12    UNEXPLORED
Result: All: A:130165    B:3665  C:79    D:1371
```

bw without parity

```
Full0:  A:16932 B:0     C:0     D:0     LUT+UNUSED
Full1:  A:138   B:0     C:594   D:0     LUT+ALTERNATE
Full2:  A:31470 B:0     C:0     D:0     CELL_INTERCONN.+UNUSED
Full3:  A:96    B:0     C:461   D:0     CELL_INTERCONN.+ALTERNATE
Full4:  A:3     B:0     C:122   D:0     CELL_INTERCONN.+OPEN
Full5:  A:41    B:0     C:302   D:0     CELL_INTERCONN.+CONFLICT_OF
Full6:  A:294   B:0     C:585   D:0     CELL_INTERCONN.+CONFLICT_FF
Full7:  A:13986 B:0     C:0     D:0     CELL_INTERCONN.+OUTPUT_ANTENNA
Full8:  A:78    B:0     C:34    D:0     CELL_INTERCONN.+UNPREDICTABLE
Full9:  A:10265 B:0     C:0     D:0     BUS_TO_CELL+UNUSED
Full10: A:11    B:0     C:269   D:0     BUS_TO_CELL+OPEN
Full11: A:55    B:0     C:43    D:0     BUS_TO_CELL+CONFLICT_FF
Full12: A:397   B:0     C:0     D:0     BUS_TO_CELL+OUTPUT_ANTENNA
Full13: A:10156 B:0     C:0     D:0     BUS_CROSSING+UNUSED
Full14: A:0     B:0     C:50    D:0     BUS_CROSSING+OPEN
Full15: A:51    B:0     C:10    D:0     BUS_CROSSING+CONFLICT_FF
Full16: A:773   B:0     C:0     D:0     BUS_CROSSING+OUTPUT_ANTENNA
Full17: A:32278 B:0     C:0     D:0     BUS_REPEATER+UNUSED
Full18: A:0     B:0     C:153   D:0     BUS_REPEATER+OPEN
Full19: A:127   B:0     C:1200  D:0     BUS_REPEATER+CONFLICT_FF
Full20: A:254   B:0     C:0     D:0     BUS_REPEATER+INPUT_ANTENNA
Full21: A:1208  B:0     C:0     D:0     BUS_REPEATER+OUTPUT_ANTENNA
Full22: A:1104  B:0     C:0     D:0     FAKE+UNUSED
Full23: A:11706 B:0     C:34    D:0     UNKNOWN+UNEXPLORED
---- FPGA RESOURCES -----
Cumulative0:    A:17070 B:0     C:594   D:0     LUT
Cumulative1:    A:45968 B:0     C:1504  D:0     CELL_INTERCONN.
Cumulative2:    A:10728 B:0     C:312   D:0     BUS_TO_CELL
Cumulative3:    A:10980 B:0     C:60    D:0     BUS_CROSSING
Cumulative4:    A:33867 B:0     C:1353  D:0     BUS_REPEATER
Cumulative5:    A:1104  B:0     C:0     D:0     FAKE
Cumulative6:    A:11706 B:0     C:34    D:0     UNKNOWN
---- FAULT MODELS -----
Cumulative7:    A:102205 B:0    C:0     D:0     UNUSED
Cumulative8:    A:234   B:0     C:1055  D:0     ALTERNATE
Cumulative9:    A:14    B:0     C:594   D:0     OPEN
Cumulative10:   A:41    B:0     C:302   D:0     CONFLICT_OF
Cumulative11:   A:527   B:0     C:1838  D:0     CONFLICT_FF
Cumulative12:   A:254   B:0     C:0     D:0     INPUT_ANTENNA
Cumulative13:   A:16364 B:0     C:0     D:0     OUTPUT_ANTENNA
Cumulative14:   A:78    B:0     C:34    D:0     UNPREDICTABLE
Cumulative15:   A:11706 B:0     C:34    D:0     UNEXPLORED
Result: All: A:131423    B:0     C:3857  D:0
```

bw with parity

```
Full0:  A:16830 B:0     C:0     D:0     LUT+UNUSED
Full1:  A:169   B:552   C:35    D:78    LUT+ALTERNATE
Full2:  A:31302 B:0     C:0     D:0     CELL_INTERCONN.+UNUSED
Full3:  A:119   B:429   C:24    D:69    CELL_INTERCONN.+ALTERNATE
Full4:  A:1     B:125   C:5     D:17    CELL_INTERCONN.+OPEN
Full5:  A:39    B:199   C:16    D:74    CELL_INTERCONN.+CONFLICT_OF
Full6:  A:313   B:590   C:32    D:109   CELL_INTERCONN.+CONFLICT_FF
Full7:  A:13901 B:0     C:0     D:0     CELL_INTERCONN.+OUTPUT_ANTENNA
Full8:  A:86    B:17    C:0     D:5     CELL_INTERCONN.+UNPREDICTABLE
Full9:  A:10207 B:0     C:0     D:0     BUS_TO_CELL+UNUSED
Full10: A:7     B:225   C:10    D:71    BUS_TO_CELL+OPEN
Full11: A:67    B:53    C:4     D:13    BUS_TO_CELL+CONFLICT_FF
Full12: A:383   B:0     C:0     D:0     BUS_TO_CELL+OUTPUT_ANTENNA
Full13: A:10006 B:0     C:0     D:0     BUS_CROSSING+UNUSED
Full14: A:0     B:29    C:0     D:28    BUS_CROSSING+OPEN
Full15: A:33    B:8     C:0     D:8     BUS_CROSSING+CONFLICT_FF
Full16: A:928   B:0     C:0     D:0     BUS_CROSSING+OUTPUT_ANTENNA
Full17: A:31988 B:0     C:0     D:0     BUS_REPEATER+UNUSED
Full18: A:0     B:112   C:7     D:60    BUS_REPEATER+OPEN
Full19: A:94    B:818   C:47    D:547   BUS_REPEATER+CONFLICT_FF
Full20: A:280   B:0     C:0     D:0     BUS_REPEATER+INPUT_ANTENNA
Full21: A:1267  B:0     C:0     D:0     BUS_REPEATER+OUTPUT_ANTENNA
Full22: A:1104  B:0     C:0     D:0     FAKE+UNUSED
Full23: A:11690 B:31    C:0     D:19    UNKNOWN+UNEXPLORED
---- FPGA RESOURCES -----
Cumulative0:    A:16999 B:552   C:35    D:78    LUT
Cumulative1:    A:45761 B:1360  C:77    D:274   CELL_INTERCONN.
Cumulative2:    A:10664 B:278   C:14    D:84    BUS_TO_CELL
Cumulative3:    A:10967 B:37    C:0     D:36    BUS_CROSSING
Cumulative4:    A:33629 B:930   C:54    D:607   BUS_REPEATER
Cumulative5:    A:1104  B:0     C:0     D:0     FAKE
Cumulative6:    A:11690 B:31    C:0     D:19    UNKNOWN
---- FAULT MODELS -----
Cumulative7:    A:101437 B:0    C:0     D:0     UNUSED
Cumulative8:    A:288   B:981   C:59    D:147   ALTERNATE
Cumulative9:    A:8     B:491   C:22    D:176   OPEN
Cumulative10:   A:39    B:199   C:16    D:74    CONFLICT_OF
Cumulative11:   A:507   B:1469  C:83    D:677   CONFLICT_FF
Cumulative12:   A:280   B:0     C:0     D:0     INPUT_ANTENNA
Cumulative13:   A:16479 B:0     C:0     D:0     OUTPUT_ANTENNA
Cumulative14:   A:86    B:17    C:0     D:5     UNPREDICTABLE
Cumulative15:   A:11690 B:31    C:0     D:19    UNEXPLORED
Result: All: A:130814    B:3188  C:180   D:1098
```

s1488 without parity

```
Full0:  A:14386 B:0     C:0     D:0     LUT+UNUSED
Full1:  A:563   B:0     C:2715  D:0     LUT+ALTERNATE
Full2:  A:25046 B:0     C:0     D:0     CELL_INTERCONN.+UNUSED
Full3:  A:623   B:0     C:2262  D:0     CELL_INTERCONN.+ALTERNATE
Full4:  A:10    B:0     C:589   D:0     CELL_INTERCONN.+OPEN
Full5:  A:221   B:0     C:1441  D:0     CELL_INTERCONN.+CONFLICT_OF
Full6:  A:1475  B:0     C:2723  D:0     CELL_INTERCONN.+CONFLICT_FF
Full7:  A:12615 B:0     C:0     D:0     CELL_INTERCONN.+OUTPUT_ANTENNA
Full8:  A:332   B:0     C:135   D:0     CELL_INTERCONN.+UNPREDICTABLE
Full9:  A:7096  B:0     C:0     D:0     BUS_TO_CELL+UNUSED
Full10: A:20    B:0     C:1260  D:0     BUS_TO_CELL+OPEN
Full11: A:333   B:0     C:465   D:0     BUS_TO_CELL+CONFLICT_FF
Full12: A:1866  B:0     C:0     D:0     BUS_TO_CELL+OUTPUT_ANTENNA
Full13: A:7110  B:0     C:0     D:0     BUS_CROSSING+UNUSED
Full14: A:0     B:0     C:323   D:0     BUS_CROSSING+OPEN
Full15: A:776   B:0     C:491   D:0     BUS_CROSSING+CONFLICT_FF
Full16: A:2340  B:0     C:0     D:0     BUS_CROSSING+OUTPUT_ANTENNA
Full17: A:21712 B:0     C:0     D:0     BUS_REPEATER+UNUSED
Full18: A:0     B:0     C:910   D:0     BUS_REPEATER+OPEN
Full19: A:887   B:0     C:7273  D:0     BUS_REPEATER+CONFLICT_FF
Full20: A:773   B:0     C:0     D:0     BUS_REPEATER+INPUT_ANTENNA
Full21: A:3665  B:0     C:0     D:0     BUS_REPEATER+OUTPUT_ANTENNA
Full22: A:1104  B:0     C:0     D:0     FAKE+UNUSED
Full23: A:11423 B:0     C:317   D:0     UNKNOWN+UNEXPLORED
---- FPGA RESOURCES -----
Cumulative0:    A:14949 B:0     C:2715  D:0     LUT
Cumulative1:    A:40322 B:0     C:7150  D:0     CELL_INTERCONN.
Cumulative2:    A:9315  B:0     C:1725  D:0     BUS_TO_CELL
Cumulative3:    A:10226 B:0     C:814   D:0     BUS_CROSSING
Cumulative4:    A:27037 B:0     C:8183  D:0     BUS_REPEATER
Cumulative5:    A:1104  B:0     C:0     D:0     FAKE
Cumulative6:    A:11423 B:0     C:317   D:0     UNKNOWN
---- FAULT MODELS -----
Cumulative7:    A:76454 B:0     C:0     D:0     UNUSED
Cumulative8:    A:1186  B:0     C:4977  D:0     ALTERNATE
Cumulative9:    A:30    B:0     C:3082  D:0     OPEN
Cumulative10:   A:221   B:0     C:1441  D:0     CONFLICT_OF
Cumulative11:   A:3471  B:0     C:10952 D:0     CONFLICT_FF
Cumulative12:   A:773   B:0     C:0     D:0     INPUT_ANTENNA
Cumulative13:   A:20486 B:0     C:0     D:0     OUTPUT_ANTENNA
Cumulative14:   A:332   B:0     C:135   D:0     UNPREDICTABLE
Cumulative15:   A:11423 B:0     C:317   D:0     UNEXPLORED
Result: All: A:114376    B:0     C:20904 D:0
```

s1488 with parity

```
Full0:  A:13304 B:0     C:0     D:0     LUT+UNUSED
Full1:  A:662   B:3091  C:86    D:521   LUT+ALTERNATE
Full2:  A:23286 B:0     C:0     D:0     CELL_INTERCONN.+UNUSED
Full3:  A:717   B:2227  C:70    D:403   CELL_INTERCONN.+ALTERNATE
Full4:  A:9     B:599   C:20    D:102   CELL_INTERCONN.+OPEN
Full5:  A:270   B:1605  C:67    D:286   CELL_INTERCONN.+CONFLICT_OF
Full6:  A:1804  B:2675  C:80    D:574   CELL_INTERCONN.+CONFLICT_FF
Full7:  A:12108 B:0     C:0     D:0     CELL_INTERCONN.+OUTPUT_ANTENNA
Full8:  A:458   B:92    C:8     D:12    CELL_INTERCONN.+UNPREDICTABLE
Full9:  A:6155  B:0     C:0     D:0     BUS_TO_CELL+UNUSED
Full10: A:39    B:1220  C:47    D:337   BUS_TO_CELL+OPEN
Full11: A:330   B:394   C:12    D:81    BUS_TO_CELL+CONFLICT_FF
Full12: A:2425  B:0     C:0     D:0     BUS_TO_CELL+OUTPUT_ANTENNA
Full13: A:6434  B:0     C:0     D:0     BUS_CROSSING+UNUSED
Full14: A:0     B:232   C:8     D:123   BUS_CROSSING+OPEN
Full15: A:725   B:288   C:3     D:165   BUS_CROSSING+CONFLICT_FF
Full16: A:3062  B:0     C:0     D:0     BUS_CROSSING+OUTPUT_ANTENNA
Full17: A:18916 B:0     C:0     D:0     BUS_REPEATER+UNUSED
Full18: A:0     B:681   C:27    D:369   BUS_REPEATER+OPEN
Full19: A:1027  B:5356  C:147   D:2873  BUS_REPEATER+CONFLICT_FF
Full20: A:1013  B:0     C:0     D:0     BUS_REPEATER+INPUT_ANTENNA
Full21: A:4811  B:0     C:0     D:0     BUS_REPEATER+OUTPUT_ANTENNA
Full22: A:1104  B:0     C:0     D:0     FAKE+UNUSED
Full23: A:11357 B:256   C:7     D:120   UNKNOWN+UNEXPLORED
---- FPGA RESOURCES -----
Cumulative0:    A:13966 B:3091  C:86    D:521   LUT
Cumulative1:    A:38652 B:7198  C:245   D:1377  CELL_INTERCONN.
Cumulative2:    A:8949  B:1614  C:59    D:418   BUS_TO_CELL
Cumulative3:    A:10221 B:520   C:11    D:288   BUS_CROSSING
Cumulative4:    A:25767 B:6037  C:174   D:3242  BUS_REPEATER
Cumulative5:    A:1104  B:0     C:0     D:0     FAKE
Cumulative6:    A:11357 B:256   C:7     D:120   UNKNOWN
---- FAULT MODELS -----
Cumulative7:    A:69199 B:0     C:0     D:0     UNUSED
Cumulative8:    A:1379  B:5318  C:156   D:924   ALTERNATE
Cumulative9:    A:48    B:2732  C:102   D:931   OPEN
Cumulative10:   A:270   B:1605  C:67    D:286   CONFLICT_OF
Cumulative11:   A:3886  B:8713  C:242   D:3693  CONFLICT_FF
Cumulative12:   A:1013  B:0     C:0     D:0     INPUT_ANTENNA
Cumulative13:   A:22406 B:0     C:0     D:0     OUTPUT_ANTENNA
Cumulative14:   A:458   B:92    C:8     D:12    UNPREDICTABLE
Cumulative15:   A:11357 B:256   C:7     D:120   UNEXPLORED
Result: All: A:110016    B:18716 C:582   D:5966
```

alu2 with parity

```
Full0:  A:16562 B:0     C:0     D:0     LUT+UNUSED
Full1:  A:129   B:945   C:0     D:28    LUT+ALTERNATE
Full2:  A:30785 B:0     C:0     D:0     CELL_INTERCONN.+UNUSED
Full3:  A:170   B:646   C:0     D:32    CELL_INTERCONN.+ALTERNATE
Full4:  A:3     B:168   C:0     D:5     CELL_INTERCONN.+OPEN
Full5:  A:60    B:379   C:0     D:31    CELL_INTERCONN.+CONFLICT_OF
Full6:  A:386   B:857   C:0     D:65    CELL_INTERCONN.+CONFLICT_FF
Full7:  A:13710 B:0     C:0     D:0     CELL_INTERCONN.+OUTPUT_ANTENNA
Full8:  A:156   B:18    C:0     D:1     CELL_INTERCONN.+UNPREDICTABLE
Full9:  A:9975  B:0     C:0     D:0     BUS_TO_CELL+UNUSED
Full10: A:13    B:314   C:0     D:40    BUS_TO_CELL+OPEN
Full11: A:60    B:107   C:0     D:5     BUS_TO_CELL+CONFLICT_FF
Full12: A:526   B:0     C:0     D:0     BUS_TO_CELL+OUTPUT_ANTENNA
Full13: A:10011 B:0     C:0     D:0     BUS_CROSSING+UNUSED
Full14: A:0     B:52    C:0     D:29    BUS_CROSSING+OPEN
Full15: A:63    B:49    C:0     D:19    BUS_CROSSING+CONFLICT_FF
Full16: A:817   B:0     C:0     D:0     BUS_CROSSING+OUTPUT_ANTENNA
Full17: A:31440 B:0     C:0     D:0     BUS_REPEATER+UNUSED
Full18: A:0     B:149   C:6     D:56    BUS_REPEATER+OPEN
Full19: A:170   B:1171  C:35    D:518   BUS_REPEATER+CONFLICT_FF
Full20: A:286   B:0     C:0     D:0     BUS_REPEATER+INPUT_ANTENNA
Full21: A:1389  B:0     C:0     D:0     BUS_REPEATER+OUTPUT_ANTENNA
Full22: A:1104  B:0     C:0     D:0     FAKE+UNUSED
Full23: A:11672 B:52    C:0     D:16    UNKNOWN+UNEXPLORED
---- FPGA RESOURCES -----
Cumulative0:    A:16691 B:945   C:0     D:28    LUT
Cumulative1:    A:45270 B:2068  C:0     D:134   CELL_INTERCONN.
Cumulative2:    A:10574 B:421   C:0     D:45    BUS_TO_CELL
Cumulative3:    A:10891 B:101   C:0     D:48    BUS_CROSSING
Cumulative4:    A:33285 B:1320  C:41    D:574   BUS_REPEATER
Cumulative5:    A:1104  B:0     C:0     D:0     FAKE
Cumulative6:    A:11672 B:52    C:0     D:16    UNKNOWN
---- FAULT MODELS -----
Cumulative7:    A:99877 B:0     C:0     D:0     UNUSED
Cumulative8:    A:299   B:1591  C:0     D:60    ALTERNATE
Cumulative9:    A:16    B:683   C:6     D:130   OPEN
Cumulative10:   A:60    B:379   C:0     D:31    CONFLICT_OF
Cumulative11:   A:679   B:2184  C:35    D:607   CONFLICT_FF
Cumulative12:   A:286   B:0     C:0     D:0     INPUT_ANTENNA
Cumulative13:   A:16442 B:0     C:0     D:0     OUTPUT_ANTENNA
Cumulative14:   A:156   B:18    C:0     D:1     UNPREDICTABLE
Cumulative15:   A:11672 B:52    C:0     D:16    UNEXPLORED
Result: All: A:129487    B:4907  C:41    D:845
```

alu3 with parity

```
Full0:  A:16574 B:0     C:0     D:0     LUT+UNUSED
Full1:  A:162   B:891   C:8     D:29    LUT+ALTERNATE
Full2:  A:30730 B:0     C:0     D:0     CELL_INTERCONN.+UNUSED
Full3:  A:132   B:637   C:7     D:27    CELL_INTERCONN.+ALTERNATE
Full4:  A:4     B:189   C:2     D:8     CELL_INTERCONN.+OPEN
Full5:  A:54    B:343   C:9     D:24    CELL_INTERCONN.+CONFLICT_OF
Full6:  A:398   B:787   C:12    D:34    CELL_INTERCONN.+CONFLICT_FF
Full7:  A:13866 B:0     C:0     D:0     CELL_INTERCONN.+OUTPUT_ANTENNA
Full8:  A:177   B:31    C:0     D:1     CELL_INTERCONN.+UNPREDICTABLE
Full9:  A:9913  B:0     C:0     D:0     BUS_TO_CELL+UNUSED
Full10: A:13    B:334   C:4     D:31    BUS_TO_CELL+OPEN
Full11: A:68    B:105   C:3     D:7     BUS_TO_CELL+CONFLICT_FF
Full12: A:562   B:0     C:0     D:0     BUS_TO_CELL+OUTPUT_ANTENNA
Full13: A:9836  B:0     C:0     D:0     BUS_CROSSING+UNUSED
Full14: A:1     B:54    C:1     D:20    BUS_CROSSING+OPEN
Full15: A:57    B:27    C:1     D:4     BUS_CROSSING+CONFLICT_FF
Full16: A:1039  B:0     C:0     D:0     BUS_CROSSING+OUTPUT_ANTENNA
Full17: A:31074 B:0     C:0     D:0     BUS_REPEATER+UNUSED
Full18: A:2     B:159   C:3     D:71    BUS_REPEATER+OPEN
Full19: A:199   B:1349  C:22    D:448   BUS_REPEATER+CONFLICT_FF
Full20: A:330   B:0     C:0     D:0     BUS_REPEATER+INPUT_ANTENNA
Full21: A:1563  B:0     C:0     D:0     BUS_REPEATER+OUTPUT_ANTENNA
Full22: A:1104  B:0     C:0     D:0     FAKE+UNUSED
Full23: A:11679 B:55    C:1     D:5     UNKNOWN+UNEXPLORED
---- FPGA RESOURCES -----
Cumulative0:    A:16736 B:891   C:8     D:29    LUT
Cumulative1:    A:45361 B:1987  C:30    D:94    CELL_INTERCONN.
Cumulative2:    A:10556 B:439   C:7     D:38    BUS_TO_CELL
Cumulative3:    A:10933 B:81    C:2     D:24    BUS_CROSSING
Cumulative4:    A:33168 B:1508  C:25    D:519   BUS_REPEATER
Cumulative5:    A:1104  B:0     C:0     D:0     FAKE
Cumulative6:    A:11679 B:55    C:1     D:5     UNKNOWN
---- FAULT MODELS -----
Cumulative7:    A:99231 B:0     C:0     D:0     UNUSED
Cumulative8:    A:294   B:1528  C:15    D:56    ALTERNATE
Cumulative9:    A:20    B:736   C:10    D:130   OPEN
Cumulative10:   A:54    B:343   C:9     D:24    CONFLICT_OF
Cumulative11:   A:722   B:2268  C:38    D:493   CONFLICT_FF
Cumulative12:   A:330   B:0     C:0     D:0     INPUT_ANTENNA
Cumulative13:   A:17030 B:0     C:0     D:0     OUTPUT_ANTENNA
Cumulative14:   A:177   B:31    C:0     D:1     UNPREDICTABLE
Cumulative15:   A:11679 B:55    C:1     D:5     UNEXPLORED
Result: All: A:129537    B:4961  C:73    D:709
```