

DELIVERABLE  
IST-2001-38930  
FT-EA

Deliverable D6  
FTEA-Philips-26May04-D6

# Prototyping for Fault Tolerance and Simultaneously Switching Outputs



<b>Project Number</b>	:	IST-2001-38930
<b>Project Title</b>	:	FT-EA

<b>Internal Project Number</b>	:	FTEA-philips-26may04-D6
<b>Date</b>	:	May 26, 2004
<b>Title</b>	:	Prototyping for Fault Tolerance and simultaneously Switching Outputs
<b>Contributing Work Package</b>	:	WP3
<b>Deliverable type</b>	:	Report + Software
<b>Security</b>	:	Report (Public), Software (Confidential)
<b>Author(s)</b>	:	Damien Leroy, Atul Katoch, André Nieuwland

**Abstract:**

This document is deliverable D6 of Work package 3 of the FT-EA project. The aim of the project is to research the application of Fault Tolerance Techniques to combat electrical problems in very deep sub-micron semi-conductor designs.

The public part of this deliverable is the description of the prototype tool for evaluating the options for fault tolerant busses, in particular with respect to the noise generated by simultaneous switching outputs (SSO).

The confidential part of deliverable D4 is added as an appendix (CD-ROM) containing the software of the prototyping tool.

**Keywords:**

Fault Tolerance, Power consumption, VDSM, Error correcting code, Dual Rail, Hamming, simultaneous switching outputs, switching noise.



# Table of Content

1	Introduction .....	4
2	SSO module description .....	5
	2.1 Overview .....	5
	2.2 Inputs/Outputs .....	5
	2.3 SSO noise computation .....	5
3	Bus inversion description .....	7
	3.1 Quick description .....	7
	3.2 Implementation .....	7
4	General flow changes .....	9
5	B-RoCKIT additional features .....	10
6	B-RoCKIT output.....	11
7	Benchmarking .....	13
8	Bus segmentation .....	15
9	Conclusion on SSO.....	16
10	References.....	17
11	Appendix A : BRoCKIT user guide.....	18
	11.1 Introduction: A glance at B-RoCKIT .....	18
	11.2 Supported platforms .....	18
	11.3 Package installation.....	18
	11.4 Installed directories.....	19
	11.5 Quick Start.....	19
	11.6 B-RoCKIT setup file .....	21
	11.7 Schematic view of resulting protected bus.....	22
	11.8 Technical support and upgrades.....	22
12	Appendix B : Description of B-RoCKIT directory structure.....	23
13	Appendix C : BRoCKIT Software .....	24



# 1 Introduction

This document contains the description of the B-RoCKIT tool at the end of WP3 of the FT-EA project. (As the prototype B-RoCKIT has to be upgraded at each waypoint, this document only contains the changes brought during the current waypoint.)

The FT-EA project is dedicated to the study of ECC-based fault-tolerant bus architectures, with a focus on power consumption, crosstalk, and Simultaneously Switching Outputs (SSO) noise, the final goal of the project being to propose an optimal bus architecture regarding all these parameters.

In this deliverable, we will present a quick glance on our studies on SSO noise, the way we modeled it and how we compute its relative influence easily. These studies have also led us to a way for reducing these SSO noise in a bus architecture. This 'bus inversion' feature is based on the elimination of the worst cases transitions in the current bus architecture. This function has been implemented in B-RoCKIT to fit with all the ECC developed and studied in this project (Hamming, Dual-Rail, and modified Dual-Rail codes).

We first discuss briefly the topic of SSO noise, which will be presented extensively in deliverable D7, since this work forms the start for the implementation in the prototype tool.

Thereafter, we describe the new ECC optimization (bus inversion), the current flow and the new modules. Then we will present the validation tests that have been conducted to debug the tool, and finally we will show the use of B-RoCKIT on the test vehicle, a 32 bits bus taken from a fault-tolerant RISC processor designed by iRoC Technologies.

Finally we will discuss the results of the tests and show the results on power consumption, crosstalk, and SSO noise, via the example of the test vehicle.



## 2 SSO module description

### 2.1 Overview

Simultaneous switching noise (SSN) in reference to on-chip busses, refers to the noise generated due to changes in voltages and currents caused by the switching of many drivers at the same time. It can be of major concern to signal integrity where fast changing currents can cause large voltage drop on on-chip power supply network. SSN on on-chip busses is not a random process and therefore deterministic and can be modeled and simulated. However, SSO depends on a number of parameters which makes it extremely complicated to be accurately modeled. Therefore, a simplified model is developed which helps in evaluating the relative performance of different coding schemes. This module computes the SSO noise impact on the design.

### 2.2 Inputs/Outputs

The inputs are the technology parameters and the current design parameters. Design parameters include frequency, footprint, bus segmentation, length, width etc. Technology parameters include minimum and maximum spacing, minimal wire pitch, Vdd, capacitance information and power information. These parameters are needed for the earlier developed modules of B-RoCKIT, and only a few of them are actually used in the SSO module.

The output contains the description of the bus and since the update of this work package also the relative SSO noise parameter.

### 2.3 SSO noise computation

In order to compute the SSO noise parameter, we have modeled and simulated the power grid of a bus [1]. This model is very accurate but also quite hard to compute from analytical electrical formulas.

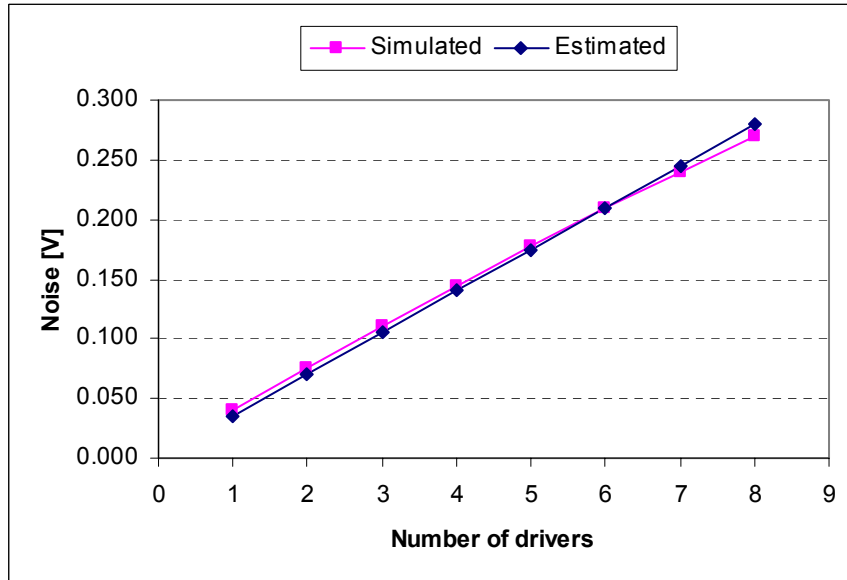
Based on these results, the partners have created a function that approximates the real value but is by far easier to compute. This function is shaped like:  $SSO = N * S_w + M * S_{wm} * R$ , where:

- SSO is the relative SSO noise level
- $N$  is the number of switching wires,
- $S_w$  is the switching factor,
- $M$  is the inter wire coefficient,
- $S_{wm}$  is the inter wire contribution which can be either positive or negative depending upon wire geometry,
- $R$  is the spacing ratio against minimum spacing.

The accuracy of this approximated formula is better than 10% for 10 wires and wire spacing less than ten times the minimum spacing allowed by the technology. If the number of wires is very large then the formula is no longer accurate as in real situations the noise would saturate due to negative feedback from the power supply to the drivers. The reduced power supply level will reduce the strength of the drivers and reduced the additional SSO noise. Realise that at these saturation levels, the supply level has been decreased beyond acceptable integrity levels. However, if any measure like decoupling capacitors is used to mitigate the effect of SSO the saturation effect is postponed again to even higher numbers of drivers and the approximate formula used here becomes valid again. This means that this formula can help in qualitative evaluation of the impact of various schemes on the noise due to simultaneous switching drivers.



Moreover, we have seen that the  $S_{wm}$  factor is very small as compared to  $S_w$ , such that the formula can be simplified to  $SSO = N * S_w$  without much loss in accuracy. This holds quite true for the kind of wire geometries we are interested in where the wire resistance and capacitance are quite large compared to driver resistance and capacitance. This implies that the drivers remain in the saturation region for significant amount of time while switching, which leads to a very limited dependence on the size of load on simultaneous switching noise. This is also shown in Figure 1. Therefore, we used the approximated formula in B-RoCKIT in order to compute the SSO noise.



**Figure 1: Simulated v/s estimated noise due to SSO for a 10mm wire, minimum wire spacing and width, on Metal-2 over FOX in 0.13um CMOS technology.**

Finally, considering that  $S_w$  will be approximately the same for all the encoding schemes, we will only consider the  $N$  factor.

This factor is representing the number of switching wires for the worst-case transitions.

The worst case is all 0 to all 1 for Hamming, and all 0 to all 1 for dual rail and modified dual rail with odd data width.

For dual-rail and modified dual rail codes the worst case is all 0 to all 1 minus parity if data width is even.

The only way to reduce the SSO noise by means of coding is to reduce the amount of simultaneous transitions. Therefore, we incorporated bus invert coding, which reduces the worst case number of simultaneous transitions by 50%. The application of bus invert coding is briefly described in the next chapter.



### 3 Bus inversion description

#### 3.1 Quick description

The bus inversion feature is the method we propose to reduce the SSO noise problem. It is based on the studies made in D5 [1]. These studies have proven that the SSO noise is directly related to the number of switching wires in the worst case.

The main idea is to check the number of transitions between two consecutive words. If the number of transitions is larger than half the bus size, we invert the word in a binary way (ie. Invert each bit of the word). This approach ensures that the number of transitions on the bus is less than half the bus size. For more information on bus-invert coding, the advantages and disadvantages, see [2] and the references made in that article to prior art.

The advantage of this method is that the worst case noise of the bus architecture is reduced by half, though the implementation will cost additional area.

#### 3.2 Implementation

This bus inversion feature is implemented in the current version of B-RoCKIT. If the '-inv' option is used in the command line, B-RoCKIT implements after the encoder a distance checker between the current and the last transmitted words. If the Hamming distance between these two words is greater than half the word length, the current word will be inverted in order to minimize the SSO noise.

If the bus is not synchronized, the data bits and the check bits will not be sent at the same time because of the delay of the ECC circuitry, thus the check bits won't be added in the worst case computation. On the other hand, if the signals are synchronized, all the bits are sent at the same time and contribute to the worst case SSO noise. This option can be set through the '-sync' option.

The current architecture for the bus invert feature is shown in Figure 2:

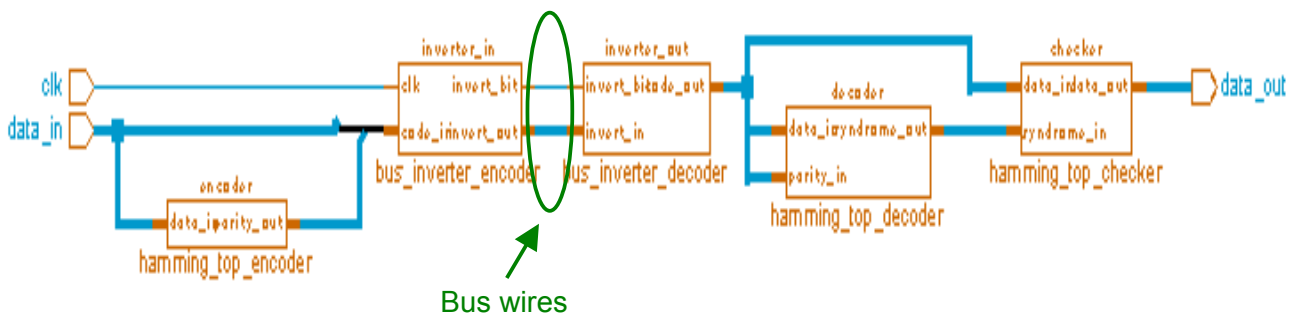


Figure 2: Bus Invert Architecture

The main disadvantage of this implementation is the area of the distance checker, which is about 64% of the whole architecture (encoder, decoder and checker) in a 0.13 um technology, for a 32 bit bus protected by Hamming code.



A new parallel architecture based on the concurrent computation of Hamming distance and parity bits will be implemented in the next version of the tool. This new architecture will decrease the design critical path, and thus will increase its speed.



## 4 General flow changes

During the development of B-RoCKIT for WP3, a new module (sso) has been added to the flow. This module is in charge of computing the SSO noise parameter for the current configuration. This module can be called independently from the others through the use of the -sso parameter in the command line.

The general data structure has been modified to accept Bus-invert code.

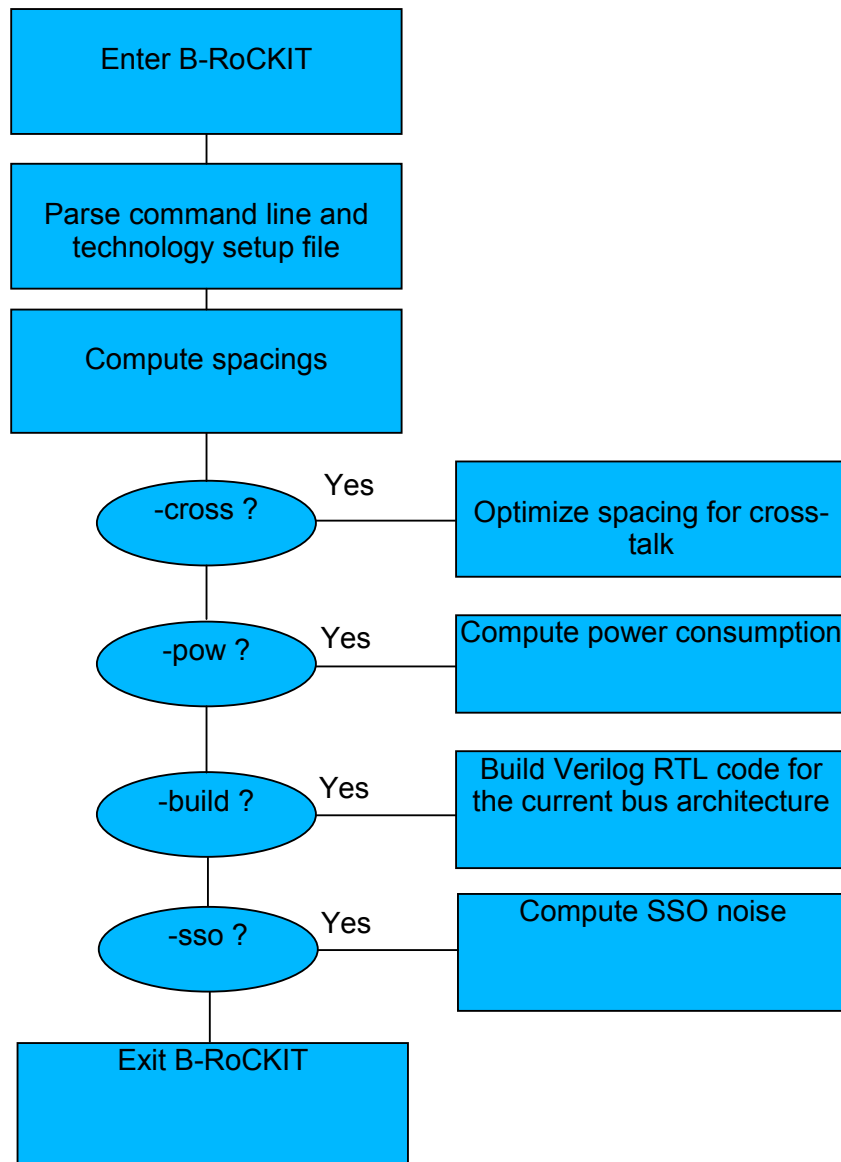


Figure 3: B-RoCKIT flow diagram



## 5 B-RoCKIT additional features

Several changes have been included in B-RoCKIT during WP3 development, mainly due to architectural demands from the new module. The main change is the addition of the sso module to the tool, but the command line has been updated and the flow has been rearranged. All these changes take precedence before B-RoCKIT description in [3].

- B-RoCKIT now supports the bus invert feature and some associated extensions.
- Fixed some bugs in the wire spacing computation.
- B-RoCKIT now uses a more user-friendly report template with some figures, in order to quickly evaluate the relative architecture efficiency. (See below)

The new report template displays some ASCII pictures. (see below).

```

-----
info:          Overall Power Estimation: 294.32 EGP
-----
info:          End of printing current_ecc fields

sso:          Entering sso module

sso:          Computing the formula Noise = N*Sw

sso:          the noise parameter is N = 19

sso:          Exiting sso module
Relative power consumption for the current_architecture
power = 294.319422      worst_power = 420.372642
      |-----X-----|
Relative crosstalk for the current_architecture
crosstalk = 9.590038   worst_crosstalk = 30.216029
      |-----X-----|
Relative sso noise for the current_architecture
sso = 19.000000 worst_sso = 66.000000
      |-----X-----|

builder:      dumping of Verilog code for Hamming
builder:      implementation of bus inversion

__ end of brockit (iRoC Technologies) __ ran by "dleroy"

```

**Figure 4: ASCII graphs, taken from B-RoCKIT output**

In Figure 4, you can see the three ASCII graphs that give the estimates of power, cross talk and SSO noise of the chosen architecture after optimization.

- The cross (X) is the position of the current optimized architecture,
- The left '|' is 0,
- The right '|' is the worst case value, computed for the same number of bits and bus segmentation, but for the minimum spacing allowed by the technology design rules.

Due to these graphs, one is immediately able to compare in a single view the efficiency of the chosen architecture with alternative architectures.



## 6 B-RoCKIT output

For the benchmarking example we have displayed below the output of B-RoCKIT interface :

```
sun4:>./brockit -s setup.txt -f 166 -foot 134 -map [0 31] -l 1000 -buf 0 -cross -power -build -o out.v -ecc hc -inv -sso -verb -graph

__ brockit (iRoC Technologies) __ ran by "dleroy"
__ Bus-RoCKIT Platform (Verilog RTL)

brockit: FlexLM licensed version
brockit: technology parameters list
      Cbsq = 2.230000e-17 F/um^2
      Vdd = 1.20 V
      SPmin = 0.21 um
      Wmin = 0.20 um
      Pinputdriver = 0.62 EGP
      Pinternaldriver = 3.88 EGP
      Pinputreceiver = 0.08 EGP
      Pinternalreceiver = 0.33 EGP
      SPmax = 2.00 um
      EGP = 3.096000e-14 W/Hz
      Cecm = 1.100000e-16 F*um
      Cebmax = 1.980000e-17 F/um
      Cebmin = 5.330000e-18 F/um

code_computer: Computing Hamming code size

space_computer: Footprint = 134 wires      Footprint = 54.73 um
space_computer: Computing Hamming code spacings
      index: 0 SP = 1.198205      SP0 = 1.198205

crosstalk_comp: Entering crosstalk module
crosstalk_comp: Before optimisation NC_worst_eff = 7.81      TEC = 49.91
crosstalk_comp: Optimizing spacings for each bus partition.
      index: 0 SP = 1.233076      SP0 = 0.553085
crosstalk_comp: After optimisation NC_worst_eff = 7.68      TEC = 49.99
crosstalk_comp: Exiting crosstalk module

power_comp: Entering power module
power_comp: Computing Hamming code power consumption
power_comp: Ptot = 282.677490 EGP;      Ptot= 1.452781e-03 W
power_comp: Exiting power module

sso_comp: Entering sso module
sso_comp: Computing the formula Noise = N*Sw
sso_comp: Noise parameter: N = 19
sso_comp: Exiting sso module

info: prints global results
```



```

info:          ecc type: Hamming
info:          total bus size: 38
info:          total data size: 32
info:          total parity size: 6
info:          critical path: 14
info:          number of bus subpartitions: 1
info:          printing datas, parity widths and spacings for each bus subpartition
index: 0 data: 32 parity: 6
                maximum spacing between wires:          SP = 1.23
                spacing on the edge (near shields):      SP0 = 0.55
info:          Cbot_min: 1.506023e-17 F/mm
info:          Pwires: 70.03 EGP
info:          Pcc: 166.00 EGP
info:          Pbuff: 46.65 EGP
-----
info:          Overall Power Estimation:          282.68 EGP
info:          Overall Crosstalk Estimation:        7.68
info:          Overall SSO Estimation:             19.00
-----
info:          End of printing global results

ascii:        Displaying bus architecture efficiency
ascii:        Relative power consumption for the current_architecture
ascii:        power = 282.68          worst_power = 420.37
                |0-----X-----|
ascii:        Relative crosstalk for the current_architecture
ascii:        crosstalk = 7.68          worst_crosstalk = 30.22
                |0-----X-----|
ascii:        Relative sso noise for the current_architecture
ascii:        sso = 19.00          worst_sso = 66.00
                |0-----X-----|
ascii:        End of displaying bus architecture efficiency

builder:      Entering Verilog code builder
builder:      dumping of Verilog code for Hamming
builder:      implementation of bus inversion
builder:      Exiting Verilog code builder

___ end of brockit (iRoC Technologies) ___ ran by "dleroy"

```



## 7 Benchmarking

Within FT-EA we decided to use an existing bus for benchmarking the tool, for which the specifications are listed below:

- 0.13 μm CMOS technology,
- 1 millimeter long, unbuffered,
- 166 MHz running frequency,
- 32 data bits.

This bus is a 32 bits ARM bus, which connects a RISC fault-tolerant processor and a protected memory. The bus uses the AMBA AHB protocol (Advanced Microcontroller Bus Architecture, Advanced High-performance Bus).

The non-protected bus has been implemented in a Fault-Tolerant SoC by iRoC, and protecting this bus is a further improvement of this design since it carries the fault-tolerant achievement between the CPU and the memory, which are both already designed to protect the integrity of the data they process. The resulting architecture is now able to maintain the integrity of the data during the whole processing time (data processing, data storage and data transfers).

The benchmarking phase has been greatly reduced related to previous deliverables, because B-RoCKIT data structure and main algorithms have already been validated.

In Figure 5, we find results about this 32 bits bus, regarding different footprints and coding techniques.

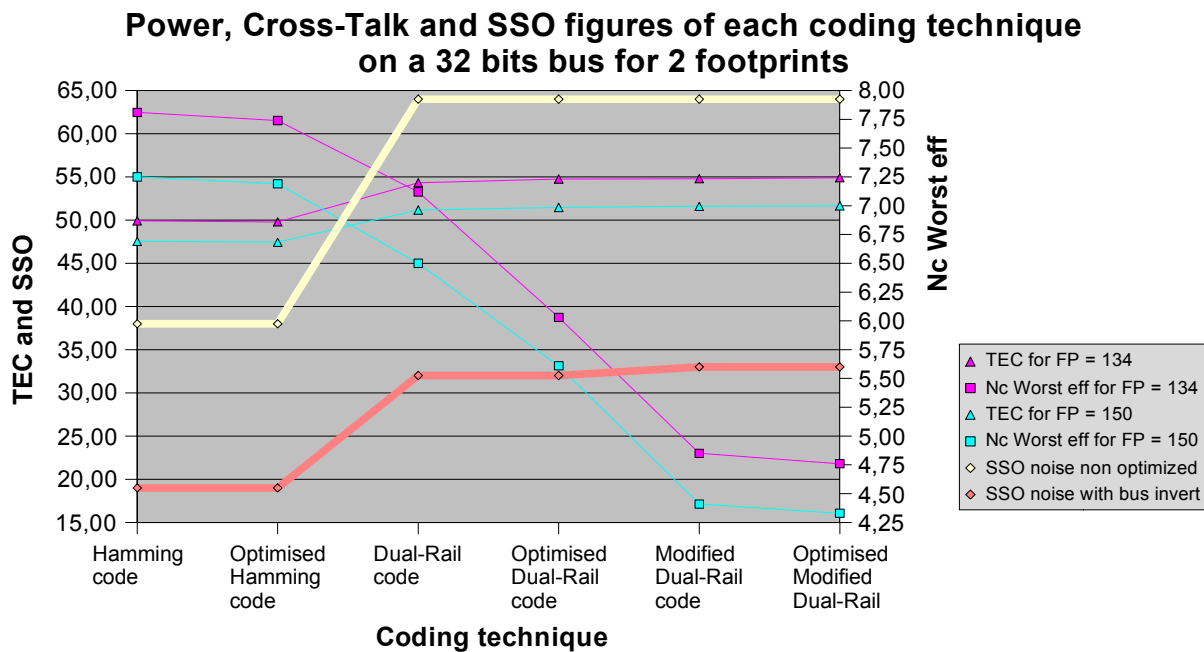


Figure 5: Power, cross-talk and SSO figures for each ECC

TEC means Total Energy Cost. It is related to the power dissipated in the bus wires.

$$TEC = P_{wires} / (C_{bot\ min} * V_{dd}^2 * f)$$



It is a normalized factor, which can be compared to  $N_{c \text{ Worst}}^{\text{eff}}$ .

$N_{c \text{ Worst}}^{\text{eff}}$  represents the Cross-Talk influence of the worst-case wire(s) as described in D3.

$N_{c \text{ Worst}}^{\text{eff}}$  is homogenous to TEC.

SSO is the number of switching wires in the worst case.



## 8 Bus segmentation

B-RoCKIT contains a feature allowing exploring different segmentation of the input bus. For example a 32 bits bus may be divided in 4 times 8 bits, 2 times 16 bits and so on. The main advantage is a simplification of the ECC circuitry and a diminution of the critical path. Moreover, the bus segmentation does heighten the fault coverage by allowing the detection and correction of more than 1 fault per word.

The drawbacks are more wires, more power consumption, and so on.

Considering that the sso noise comes only from the number of bus wires, we can discard bus segmentation effects (other than the increase in number of bus wires).



## 9 Conclusion on SSO

This waypoint of the FT-EA project has been dedicated to the modeling of SSO noise effects on bus design. This task has been achieved, and a mean to minimize SSO noise has been proposed: the bus-invert architecture.

B-RoCKIT has been updated according to this new feature in prevision of the next waypoint, the unified aspects (WP4).

We observed that dual rail-based codes suffer from their larger number of wires compared to a Hamming code, and that the design tricks used so far (duplication, intelligent spacing) have no impact on the SSO noise parameter.



## 10 References

- [1]: Deliverable D5 of FT-EA project:  
*"Optimal ECC for Simultaneously Switching Outputs (SSO) and Encoding/Decoding Circuitry for the Optimal ECC"*, March 2004.
- [2]: *"Why Transition Coding for power minimization of On-Chip Buses does not work"*  
C. Kretzschmar, A.K. Nieuwland, D. Müller,  
Proceedings of Design, Automation, and Test in Europe (DATE'04), Paris, France,  
February 2004, pp. 512-517.
- [3]: Deliverable D4 of FT-EA project:  
*"Prototyping for Fault Tolerance and Crosstalk induced peak voltage reduction"*,  
December 2003.



## 11 Appendix A : BRoCKIT user guide

### 11.1 Introduction: A glance at B-RoCKIT

The goal of B-RoCKIT is to perform and manage error correcting code implementation in data busses. Roughly, B-RoCKIT operates at RT-level and is organized as a set of binary files achieving the following tasks:

- Computing power consumption of a specified bus for different (Hamming, Dual-Rail and modified Dual-Rail) code types,
- Optimizing (limit) the cross talk impact on the design by intelligently arranging the bus wires,
- Compute the Simultaneously Switching Outputs (SSO) noise in the bus, and propose a solution (Bus Inversion) to minimize it,
- Implementing these codes in Verilog RTL.

B-RoCKIT offers the following features:

- Processing data bus according to many parameters,
- Adaptation to any technology,
- different flow options allowing a complete exploration of the design space,
- User-friendly interface that helps the user to drive the protection process.

### 11.2 Supported platforms

Sun SPARC workstations with operating systems Solaris 7, 8, 9.  
LINUX based Operating Systems.

### 11.3 Package installation

B-RoCKIT can be installed into a directory of your choice. Depending on the directory permissions, you may need to log in as a super user. Let `install_dir` be the path of the directory in which the user would like to install B-RoCKIT. The installation steps are:

- Move to `install_dir` by entering the following command on the shell prompt:

```
> cd install_dir
```

- Get the B-RoCKIT archive file (file `brockit.tar`) from the distribution support. For instance, if the distribution is on a CD, enter

```
> cp /cdrom/cdrom0/ brockit.tar .
```

- Install the package:

```
> /usr/bin/tar -xvf brockit.tar
```



## 11.4 Installed directories

The installation of B-RoCKIT creates a directory `install_dir/brockit` organized as follows:

Directory/file	Description
<code>./bin</code>	Directory of executable binary file B-RoCKIT: <b>brockit</b>
<code>./doc</code>	Documentation directory containing the user guide in PDF.
<code>./src</code>	Directory containing C source of B-RoCKIT. This directory will NOT be included in a commercial release of B-RoCKIT.
<code>./templates</code>	Directory containing templates of B-RoCKIT setup file ( <b>brockit_setup.txt</b> ) and typical examples of B-RoCKIT command line.
<code>./samples</code>	Directory containing generic VERILOG RTL examples.

Once the package is installed, check the installation by running the tool by entering:

```
> brockit
```

If the package is correctly installed, the B-RoCKIT help file is displayed on the screen.

```
sun4::test_dir > brockit
___ brockit (iRoC Technologies) ___ ran by "dleroy"
___ Bus-RoCKIT Platform (Verilog RTL)

brockit_args:  options and arguments missing

usage: brockit [-s | --setup setupfilename] [-h | --help] [-verb | --verbose] [-sil | --silent]
[-f | --frequency frequency] [-foot | --footprint footprint]
[-depth | --critical_path logicaldepth] [-buf | --buffer number_of_buffers_per_wire]
[-ecc | --ecc_type ecc_type] [-build | --builder] [-o | --output_file output_file_name]
[-power | --power_estimation] [-map | --sub_bus_map [ bus_partition ]]
[-l | --length bus_length] [-cross | --crosstalk] [-sso | --sso_noise] [-inv | --invert]
[-ver | --version] [-sync | --synchronized_process] [-graph | --graphical]

brockit arguments:
-s | --setup          setup filename
-h | --help          print this help file
-verb | --verbose    verbose mode
-sil | --silent      silent mode
-graph | --graphical graphical mode (prints architecture efficiency)
-f | --frequency     running frequency of the bus (MHz)
-foot | --footprint  maximum footprint allowed for the bus layout
-map | --sub_bus_map description of the range of the bus and its sub partitions
-depth | --crit_path maximum logical depth allowed without speed penalty
-l | --length        Total length of the bus (um)
-buf | --buffer      number of repeaters per wire
-ecc | --ecc_type    ecc type between HC (hamming code) and DRC (dual-rail code)
-cross | --crosstalk minimize crosstalk effects option
-sso | --sso_noise  minimize sso effects option
-build | --builder  builder output file name
-power | --power_estim estimation of the power dissipated by the protected bus described here
-inv | --invert include the bus inversion feature in the design in order to minimize SSO noise
-ver | --version    Displays the current version of B-RoCKIT, then exits
-sync | --synchronize Synchronizes all the bits when entering bus wires

bus_partition      [ boundary_1 boundary_2 boundary_3 ... final_boundary ]

___ end of brockit (iRoC Technologies) ___ ran by "dleroy"
```

## 11.5 Quick Start

B-RoCKIT is headed toward the finding of the “best” protected bus. This goal is achieved by choosing carefully the Error Correcting Code (ECC). This choice allows to reduce the power consumption and the cross talk impact on the protected bus by influencing different parameters.

B-RoCKIT is a front-end tool that has to be used as soon as the bus specification are written in order to generate the RTL code for the ECC circuitry and the back-end information needed by the



place&route process.

As the FT-EA project goes on, the tool will be upgraded in order to widens the proposed solutions, with solutions heading for the Simultaneously Switching Outputs (SSO) and some unified aspects.

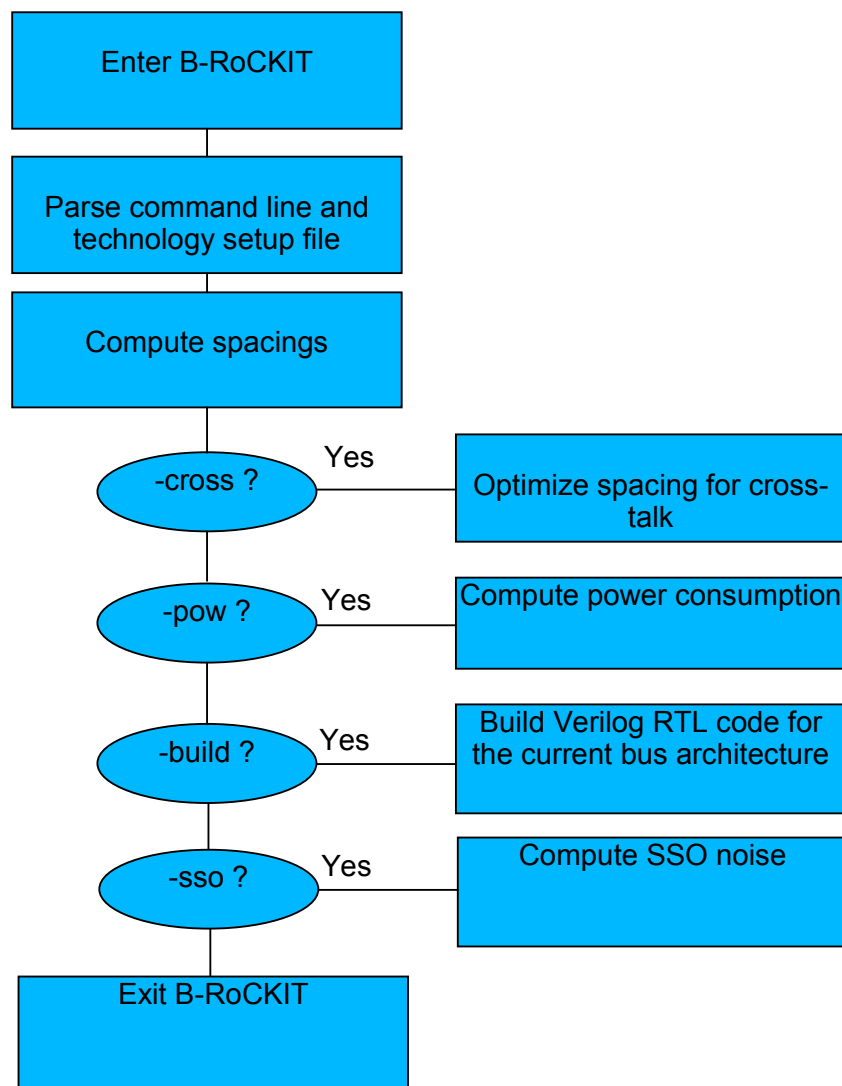
B-RoCKIT starts by reading its command line, then reading the setup file. It then begins the flow accordingly to the command line options. For a power estimation, B-RoCKIT will begin to compute the power dissipated in the wires, then in the ECC circuitry, and finally in the buffers. B-RoCKIT then displays the overall power dissipation for the chosen code.

For the cross talk optimization, B-RoCKIT begins with computing power and crosstalk parameters for the current bus architecture, then it optimizes spacing between wires, and when the optimal solution has been defined, it returns the new parameters.

If the sso option has been defined, B-RoCKIT computes a parameter for evaluating the impact of SSO noise on the architecture. Then you can use the bus inversion feature to reduce its impact on the design.

If the build option has been selected, B-RoCKIT then dump the Verilog RTL description of the chosen bus in a file specified in the command line. If no output file has been specified, B-RoCKIT uses the default "out.v" file in the current directory.

The overall flow of B-RoCKIT is displayed in the picture below:



B-RoCKIT flow will be described in the following sections using these parameters:

- 32 bits bus
- 400 MHz frequency
- 10 mm long
- 4 repeaters per wire
- 100 wires wide footprint.

The complete brockit command line description may be found by typing:

```
> brockit -h
```

The brockit setup file is structured as:

```
# comments
name_of_value = value
```

Make sure you respect the unit given in the template.

## 11.6 B-RoCKIT setup file

B-RoCKIT uses a special setup file in order to specify the technology on which we are working. The setup file must be specified through the use of `-s` option.

There is a sample of a setup file used for our example:

```
# beginning of technology setup file parsing
# Cbsquare in fF/um^2
Cbsq = 0.0223

# Vdd in Volts
Vdd = 1.2

# SPmin in micrometers
SPmin = 0.21

# Wmin in micrometers
Wmin = 0.2

# P(inp,dr) in EGP
Pinputdriver = 0.62

# P(int,dr) in EGP
Pinternaldriver = 3.88

# P(inp,rc) in EGP
Pinputreceiver = 0.08

# P(int,rc) in EGP
Pinternalreceiver = 0.33

# SPmax in micrometers (maximal value keeping the model validity)
SPmax = 2.

# Value of the EGP measure unit in fW/Hz
EGP =30.96

# value of Cecm in fF/um
Cecm = 0.110

# value of Cebmax in fF/um
Cebmax = 0.0198

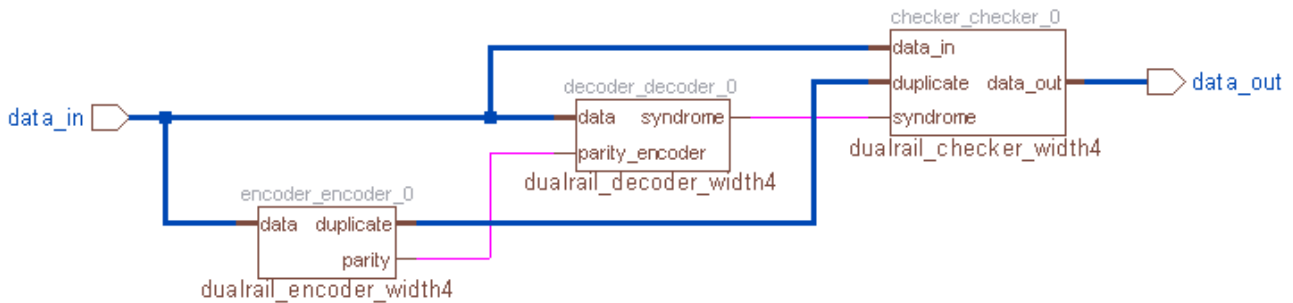
# value of Cebmin in fF/um
Cebmin = 0.00533
```



## 11.7 Schematic view of resulting protected bus

B-RoCKIT generates a VERILOG file, named dualrail4.v as specified in the command line. This file contains the functional description of the protected bus, including the ECC logical part at RT-Level and written in VERILOG 95.

The snapshot below shows the architecture of the resulting bus (visualization provided by Ambit Navigates of Cadence Design Systems, Inc.):



## 11.8 Technical support and upgrades

Bugs should be reported to iRoC Technologies support team at [support@iroctech.com](mailto:support@iroctech.com).

When reporting B -RoCKIT bugs, it is important to include:

- A reliable way to reproduce the bug.
- The version number of B-RoCKIT (and if possible version numbers of needed software).
- OS name and version.
- And any other relevant specifications.

See complete form (in PDF and MS Word formats) included in the installed `install_dir/brockit/doc` directory.



## 12 Appendix B : Description of B-RoCKIT directory structure

The directory structure of this B-RoCKIT release is organized as follows:

- Release/bin: contains the .o files resulting from the compilation of the source codes, before linking them. It also contains brockit\_local, the main executable file.
- Release/bin.ix86lin: contains the main B-RoCKIT executable file for X86 linux OS.
- Release/bin.sun4sol: contains the main B-RoCKIT executable file for Sun OS Solaris 4.
- Release/lib.ix86lin: contains the library files for X86 linux OS.
- Release/lib.sun4sol: contains the library files for Sun OS Solaris 4.
- Release/readme.txt: contains a short description of B-RoCKIT with a sample command line.
- Release/src: contains all the source files for compiling B-RoCKIT. It includes a makefile.
- Release/test\_dir: contains a sample command line (brockit.sh) for B-RoCKIT and some common outputs like a verilog file. It also contains a sample setup file (setup.txt).



## 13 Appendix C : BRoCKIT Software

The hardcopy version contains on this page a CD-ROM with the confidential software.

