



DELIVERABLE  
IST-2001-38930  
FT-EA

Deliverable D4  
FTEA-Philips-17dec03-D4

# Prototyping for Fault Tolerance and Crosstalk induced peak voltage reduction



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

<b>Internal Project Number</b>	:	FTEA-Philips-17dec03-D4
<b>Date</b>	:	17 December 2003
<b>Title</b>	:	Prototyping for Fault Tolerance and Crosstalk induced peak voltage reduction
<b>Contributing Work Package</b>	:	WP2
<b>Deliverable type</b>	:	Report
<b>Security</b>	:	Public
<b>Author(s)</b>	:	D. Leroy

**Abstract:**

This document is deliverable D4 of Work package 2 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 cross-talk, and the generation of the hardware description of the selected bus in a high-level hardware description language (HDL).

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

**Keywords:**

Fault Tolerance, Power consumption, VDSM, Error correcting code, Dual Rail, Hamming, cross talk



## Table of contents

1.	Introduction .....	4
2.	Modified Dual Rail code description .....	5
3.	General flow changes .....	6
4.	B-RoCKIT additional features .....	7
5.	Cross-Talk module description.....	8
5.1	Overview .....	8
5.2	Inputs/Outputs.....	8
5.3	Algorithm.....	8
6.	Validation tests.....	9
7.	Benchmarking .....	10
8.	Bus segmentation .....	12
9.	Conclusion on cross-talk.....	14
10.	Appendix A: B-RoCKIT user guide.....	15
10.1	Introduction: A glance at B-RoCKIT .....	15
10.2	Supported platforms.....	15
10.3	Package installation .....	15
10.4	Installation directories .....	16
10.5	Quick installation test .....	16
10.6	Quick Start: B-RoCKIT flow.....	17
10.7	An example .....	18
10.8	Launching B-RoCKIT .....	18
10.9	B-RoCKIT setup file .....	19
10.10	Schematic view of resulting protected bus.....	20
10.11	Technical support and upgrades.....	20
11.	Appendix B: B-RoCKIT readme.txt .....	21
12.	Appendix C: CD-ROM with sources (Confidential) .....	23
13.	References:.....	24



# 1. Introduction

This document contains the description of the B-RoCKIT tool at the end of WP2 of the FT-EA project. As B-RoCKIT has to be upgraded at each waypoint, this document contains only the changes brought during the current waypoint.

We will first describe the new ECC (modified Dual-Rail), the current flow and the new modules, then we will present the validation tests that have occurred in order to debug the tool, and finally we will try to use B-RoCKIT on the test vehicle, that is a 32 bits bus implemented on an iRoC Technologies design.



## 2. Modified Dual Rail code description

The modified Dual-Rail code has been described in deliverable D3 of FT-EA, and consists in:

- The normal data (n wires)
- The duplication of the data (n wires)
- One parity wire which is the logical XOR of each data bit (1 wire)
- The duplication of the parity wire (1 wire)

This code has a  $2n+2$  width, and is specially designed toward the minimization of cross-talk effects and power consumption. It derives from the Dual-Rail code introduced in D1 of FT-EA for power consumption minimization [1].

In this paper we will use the 'footprint' word to describe the width of the bus on the layout. This footprint is calculated by placing wires and shields at minimal spacing.

If the bus is segmented, we place a shield between each part of the bus.

As an example, for  $FP = 16$ , we can place 16 wires, and 15 minimal spacing (in 0.13um technology, a wire is 0.20 um wide, and the minimal spacing is  $S_{pmin} = 0.21$  um), so the footprint is  $FP = 6.35$  um.

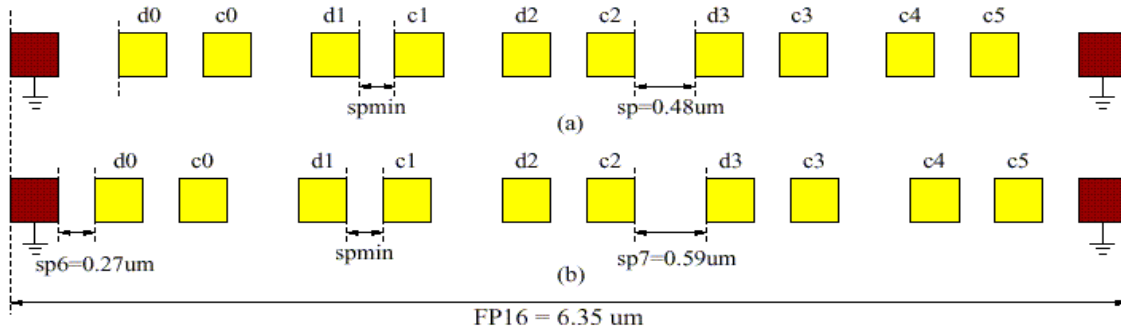


Figure 1. Example of Footprint use in a protected bus.

In Figure 1, you can figure out the Footprint ( $FP_{16}$ ), and the wire positions for a 4-bits modified Dual-Rail encoded bus, first non-optimized, then optimized.



### 3. General flow changes

During the development of B-RoCKIT for WP2, a new module (crosstalk) has been added to the flow. This module is in charge of computing the cross-talk parameter for the current configuration, then optimizing the inter-wire spacing in order to minimize this parameter.

This module can be called independently from the others through the use of the `-cross` parameter in the command line.

The general data structure has been modified to accept Modified Dual-Rail code, and the estimator module has been modified accordingly.

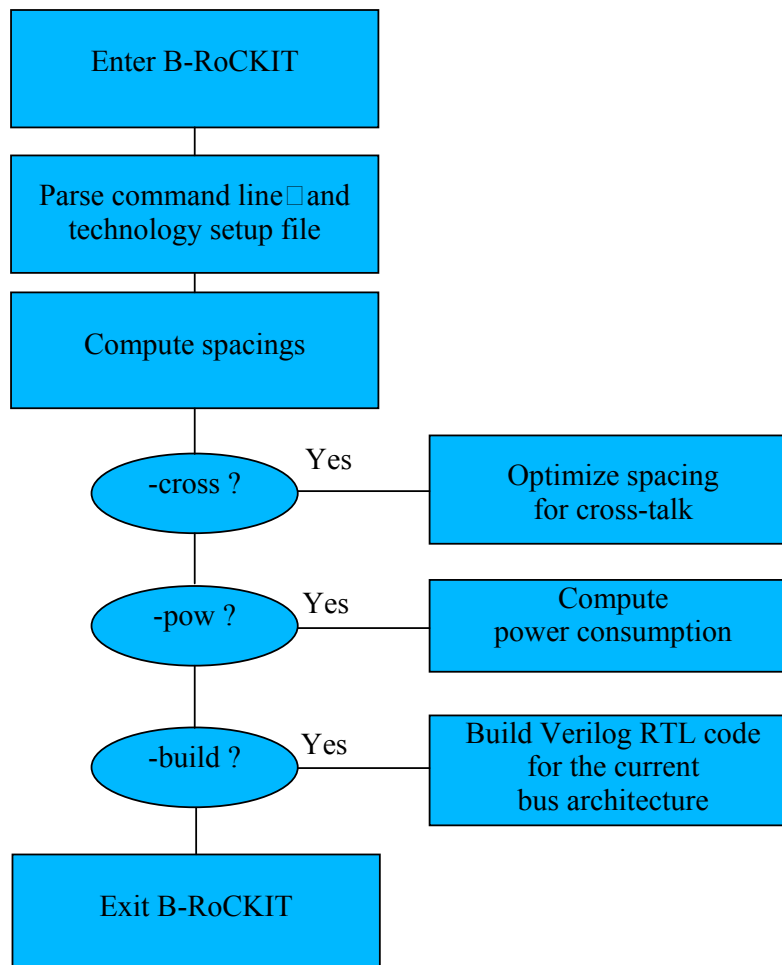


Figure 2. B-RoCKIT flow diagram



## 4. B-RoCKIT additional features

Several changes have been included in B-RoCKIT during WP2 development, mainly due to architectural demands from the new crosstalk module. The main change is the addition of the crosstalk 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 [2].

- B-RoCKIT doesn't use anymore float type objects, but double ones. It is mandatory for mathematical functions to work without forcing object type.
- A module for checking in and out licenses (on the FlexLM model) has been implemented.
- The computation of wire spacing has been removed from the power\_estimator module, and has been implemented as an independent function in order to allow its calling by every module.
- Two utility functions have been written, in order to compute TEC and  $N_{c\_Worst}^{eff}$  for benchmarking.
- The outputs of the tool have been changed in order to compel with iRoC Technologies programming rules.
- The command line has changed in order to simplify the syntax and to separate each task of the flow. Moreover some default parameters have been included, such as a default output file name for the builder module.



## 5. Cross-Talk module description

### 5.1 Overview

This module computes the best wire placement regarding Cross-Talk impact on the design.

### 5.2 Inputs/Outputs

Its inputs are the technology parameters and the current design parameters.

The outputs contain the description of the bus, modified spacing values and the cross-talk parameter.

### 5.3 Algorithm

The minimization of cross-talk effects consists in manipulating wire spacing in order to equalize the cross talk impact on each wire. The final goal is to eradicate the worst case by moving away nearby wires (and thus reducing spacing between non worst case wires).

The algorithm used for this task works step by step, by checking design rules and the cross-talk equation at each modification of spacing.

Procedure CROSSTALK\_OPTIMIZATION

Step 1: INITIALIZE the loop, Spwc := Sp

Step 2: DO

INCREASE Spwc

DECREASE Sp

WHILE [(design rules are valid) AND ( $N_{c\text{ Worst}}^{\text{eff}}(\text{Spwc}) > N_{c\text{ Worst}}^{\text{eff}}(\text{Sp})$ )]

Step 3: STORE new spacing Spwc and Sp

Please note that « design rules » means minimal and maximal spacing.

Moreover, the increase and decrease of spacing is regulated by the code structure, because each code has its own worst case, and spacing has to be distributed between every wire. The step used here is Spmin / 1000, ie. around 0.2 nm

The  $N_{c\text{ Worst}}^{\text{eff}}$  function returns the parameter for the input spacing and the current code.



## 6. Validation tests

Tool modifications have been checked in order to ensure the non-regression, and the crosstalk module has been validated through 4, 8 and 16 bits busses, which figures have been given by FT-EA partner Bologna in [3].

All FT-EA members have approved the validation phase, and have allowed the beginning of the Benchmark phase. The results of the validation process are summed in Figure 3.

	Code type	NC_worst_eff brockit	NC_worst_eff Deliv. D3	TEC brockit	TEC Deliv. D3
4 bits bus FP = 16	Hamming	12.95	13,0	13.22	13,3
	Hamming optimized	11.62	11,7	13.33	13,4
	Dual-Rail	10.48	10.5	10.62	10.7
	Dual-Rail optimized	8.54	8,5	11.17	11,2
	Modified Dual-Rail	7.85	7,9	11.69	11,7
	Modified Dual-Rail optimized	6.81	6,8	11.80	11,9
8 bits bus, FP = 27,667	Hamming	12.29	12,3	21.77	21,8
	Hamming optimized	11.46	11,5	21.80	22,0
	Dual-Rail	10.86	10,9	19.86	19,9
	Dual-Rail optimized	8.86	7,9	20.36	20,8
	Modified Dual-Rail	7.85	7,9	21.04	21,0
	Modified Dual-Rail optimized	7.22	6,8	21.18	20,2
16 bits bus, FP = 48	Hamming	11.39	11,4	35.96	36,0
	Hamming optimized	10.94	11,0	35.98	36,1
	Dual-Rail	11.14	11,2	38.48	38,5
	Dual-Rail optimized	9.10	8,0	38.95	39,5
	Modified Dual-Rail	7.85	7,9	39.75	39,8
	Modified Dual-Rail optimized	7.50	7,6	39.89	40,0

Figure 3. Results of the validation process for WP2.

The minor differences between the model from D3 and the results of B-RoCKIT can be explained by a definition of the footprint that differs (D3 actually states the spacing between wires, while B-RoCKIT only states footprint as a number of wires).



## 7. Benchmarking

FT-EA decided to use an existing bus for benchmarking the tool, and you can read its specifications 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 from previous deliverable, because B-RoCKIT data structure and main algorithms have already been validated.

In Figure 4, we find results about this 32 bits bus, regarding different footprints and coding techniques.

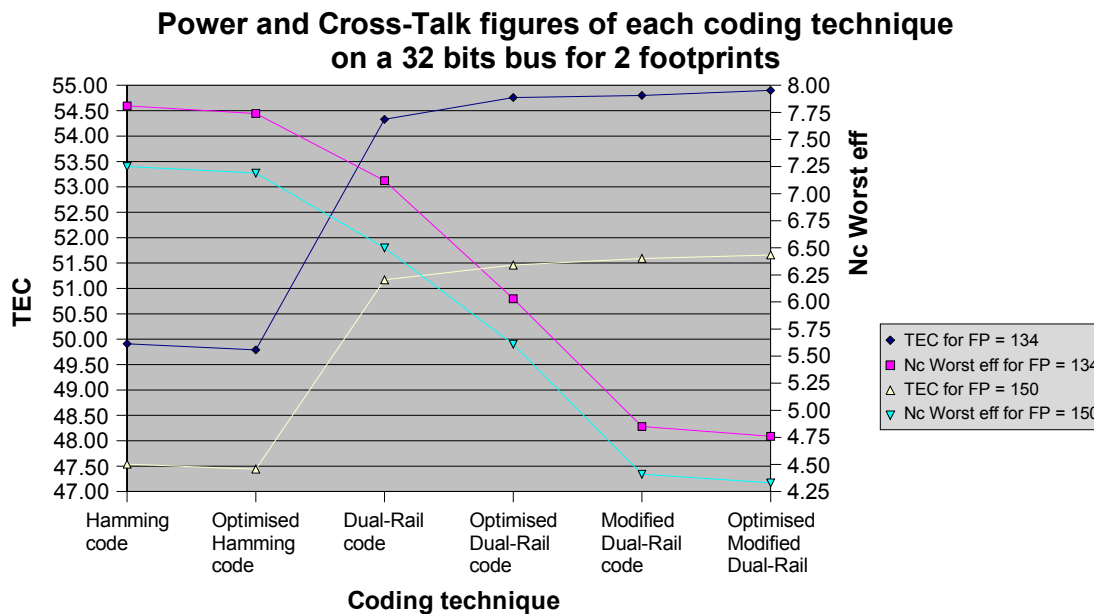


Figure 4. Power and cross-talk figures for each ECC.

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  $N_{c\ Worst}^{eff}$ .

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

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



A more complete footprint exploration for this 32 bits bus has been compiled in Figure 5, using  $TEC * N_{c \text{ Worst}}^{\text{eff}}$  in order to represent the global efficiency of each ECC regarding both power consumption and cross talk impact minimization:

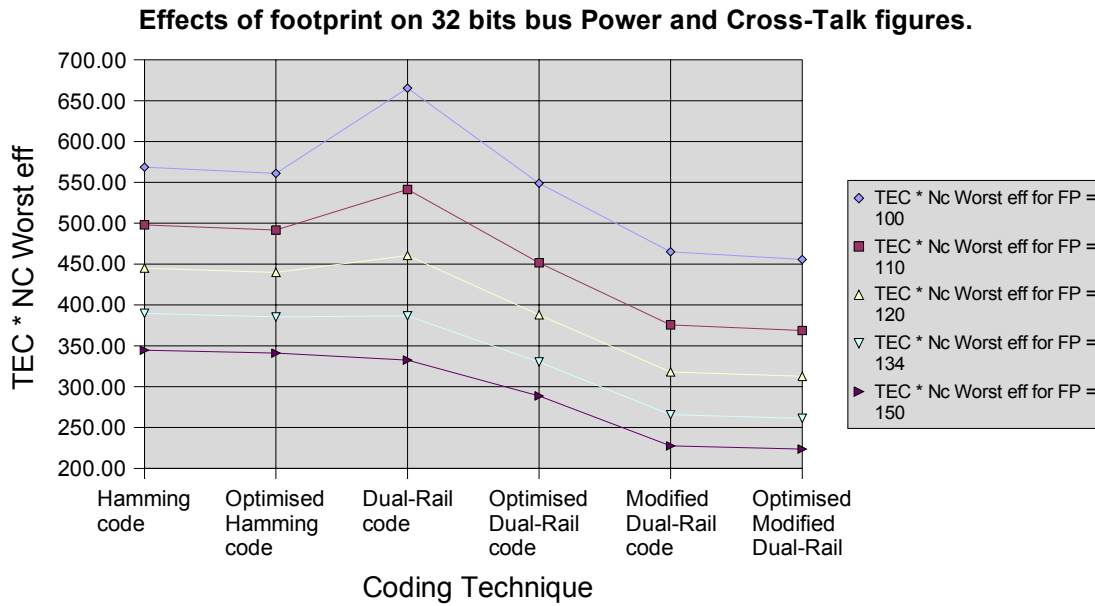


Figure 5. Effects of footprint on power and cross-talk figures for each ECC.

In Figure 5, we are able to see that the modified Dual-Rail code is simply the best code available regarding power consumption and Cross-Talk impact minimization. We also see that the Dual-Rail code, even with its very interesting power consumption figures, has a negative impact on Cross Talk, and thus is worse than Hamming code regarding the metric  $TEC * N_{c \text{ Worst}}^{\text{eff}}$  for tight footprints. But if we optimize the wire positions, even the Dual-Rail code becomes more efficient than Hamming code.



## 8. Bus segmentation

B-RoCKIT contains a feature allowing exploring different segmentation of the input bus. For example a 32 bits bus may be cut 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.

The critical path of the bus is defined as the number of gates through which the data has to go. Typically, there is a XOR tree in the encoder, the same in the decoder, the syndrome computation and finally the correction of the data.

In Dual-Rail based codes, the encoder and the decoder each contain the XOR reduction from  $n$  to 1 bits ( $\sim$  binary log of  $n$  depth) then we have the comparison between the parity bit and the reconstructed parity (one XOR), and finally the correcting multiplexer.

In the graph below appear power and cross-talk figures for each code considering several bus segmentation on the same footprint for the generic 32 bits bus used above.

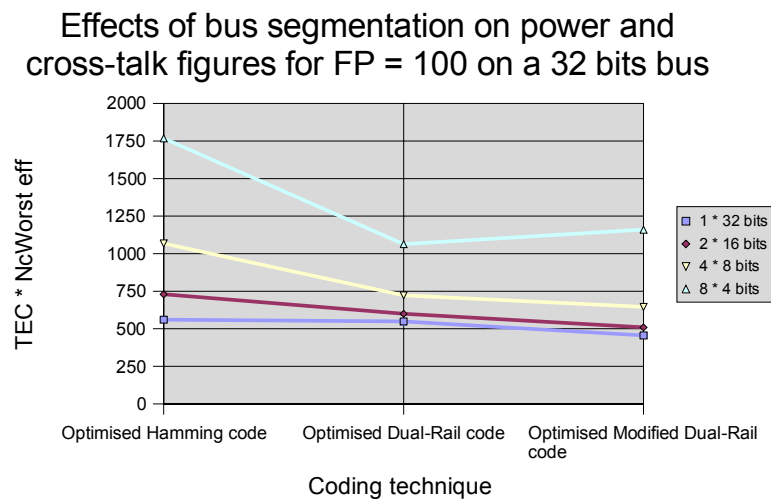


Figure 6. *Effects of bus segmentation on each ECC.*

In Figure 6, we are able to see that bus segmentation has a major impact on cross talk and power figures, mainly due to the increase of wire number. We have chosen a tight footprint in order to point the main problem of bus segmentation, that is the decrease of inter-wire spacing.



However, in Figure 7 we are able to see the positive effects of bus segmentation: Bus segmentation is able to decrease the critical path, thus allowing greater bus speed. Moreover, in the basic bus protection, the ECC allows the correction of one fault per 32 bits word, when the 4\*8 bits segmentation allows the correction of 4 errors per 32 bits word.

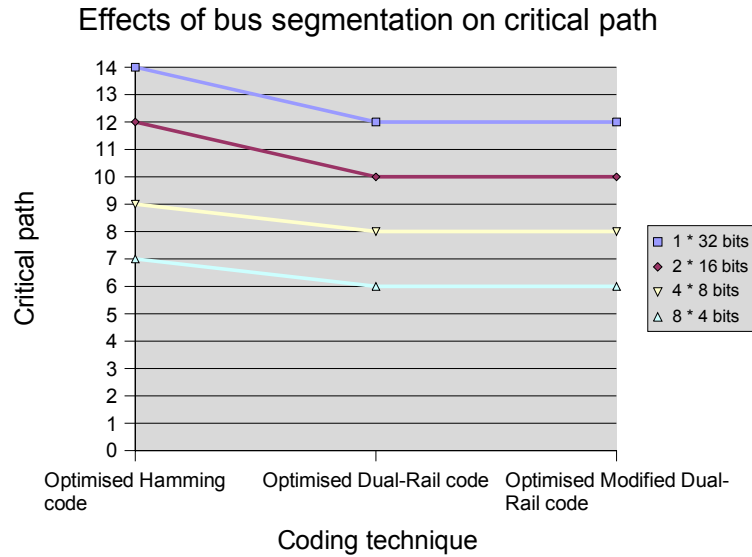


Figure 7. Effects of bus segmentation on critical path for each ECC.

Finally, we can say that the Hamming code as a whole is slower than Dual-Rail and modified Dual-Rail codes, because of its bigger critical path.



## 9. Conclusion on cross-talk

This waypoint of the FT-EA project has been dedicated to the minimization of Cross-Talk impacts on bus design, but the project has already been geared toward the final goal of unified electrical aspects, mostly with the use of the metric  $TEC * N_{c\_Worst}^{eff}$  that takes into account both cross-talk and power consumption impacts on the design.

Regarding the modified Dual-Rail code, we have seen that its power consumption figures are quite as efficient as these of the Dual-Rail code, and its Cross-Talk impact is much better than any other studied code.



## 10. Appendix A: B-RoCKIT user guide

### 10.1 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-RoCKIT1.2 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,
- 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.

### 10.2 Supported platforms

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

### 10.3 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
```



## 10.4 Installation directories

The installation of B-RoCKIT creates a directory **install\_dir/brockit** organized as follows:

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

## 10.5 Quick installation test

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.

```
sun15::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] [-ver | --version]

brockit arguments:
-s | --setup          setup filename
-h | --help          print this help file
-verb | --verbose    verbose mode
-sil | --silent      silent mode
-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
-build | --builder   builder output file name
-power | --power_estim estimation of the power dissipated by the protected bus described here
-ver | --version     Displays the current version of B-RoCKIT, then exits
bus_partition        [ boundary_1 boundary_2 boundary_3 ... final_boundary ]

___ end of brockit (iRoC Technologies) ___ ran by "dleroy"
sun15::test_dir >
```

Figure 8. B-RoCKIT help file



## 10.6 Quick Start: B-RoCKIT flow

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 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 (Fig. 9):



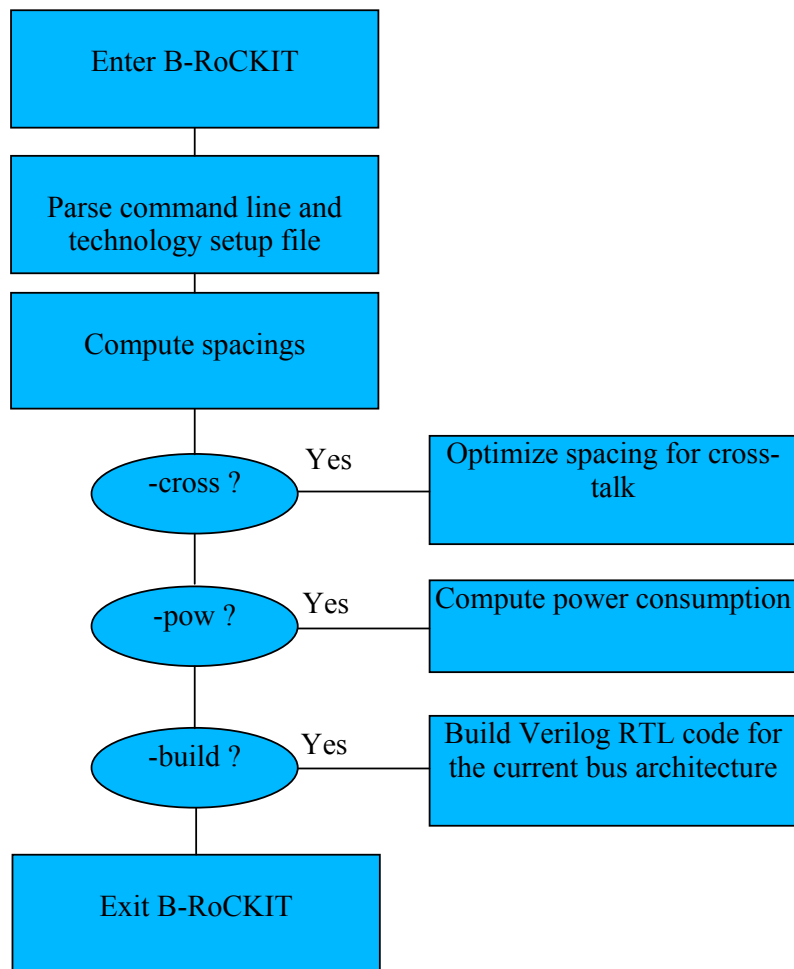


Figure 9. B-RoCKIT flow

## 10.7 An example

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.

## 10.8 Launching B-RoCKIT

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.

## 10.9 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
```



## 10.10 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.) (see fig. 10):

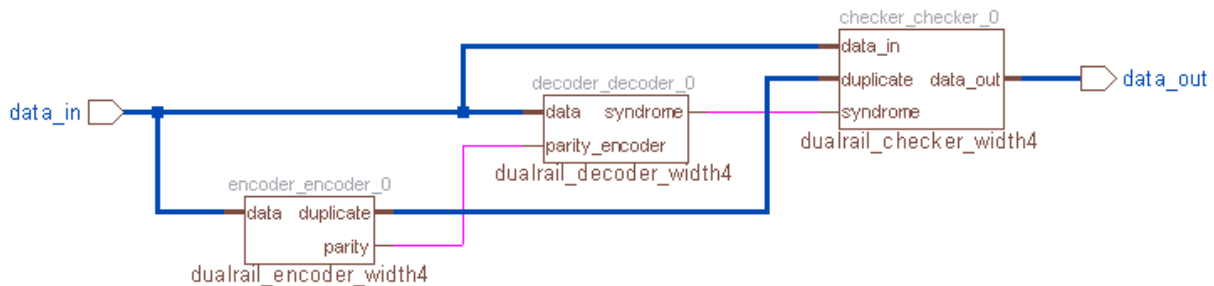


Figure 10. Architecture of generated bus

## 10.11 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.



## 11. Appendix B: B-RoCKIT readme.txt

B-RoCKIT readme.txt

Author: D. LEROY

iRoC Technologies

B-RoCKIT is a tool that computes the power dissipation of a fault tolerant bus and then produces the Verilog RTL code for the chosen architecture. It is also able to optimize a design regarding the cross talk effects.

We have three choices of architecture:

- Hamming code,
- Dual-Rail code.
- modified Dual-Rail code.

The implementation of these codes are described in Deliverable 2 and 4.

B-RoCKIT uses 2 types of parameters: Command line and setup file parameters.

-Command line parameters: these are parameters that change with each new design, such as bus length, data width, and so on.

-Setup file parameters: these are technology parameters that will remain unchanged between 2 designs on the same technology, such as the minimal wire spacing, the capacitances, and so on.

In the command line you will also find flow controls, that are -build, -cross and -power. -build will dump the Verilog code for the chosen bus configuration, -cross will optimize spacings (and then crosstalk impact on the architecture) for the current parameters, and -power will compute the power consumption of the chosen architecture.

If you select -power, with the wanted bus segmentation, B-RoCKIT will compute the power dissipation of the chosen Bus, and will return the result.

If you select -cross, with the correct bus segmentation, B-RoCKIT will compute the power and cross talk parameters of the current bus architecture, then it will optimize the architecture for cross talk impact, and it will finally display the new cross talk and power parameters.

If you select the -build option, with the wanted output file, B-RoCKIT will dump the Verilog RTL code of the chosen protected bus into this file.

(Note: B-RoCKIT will accept a default "out.v" for the output file if you don't specify the -o option).

A typical command line appears as:



```
brockit -s setup.txt -verb -f 166 -foot 83 -power -map [0 15] -ecc hc -l 1000 -buf 2 -cross -build -o design.v ;
```

This command line reads as: launch B-RoCKIT, with the setup file 'setup.txt', in verbose mode, at frequency 166 MHz, footprint 83, for Hamming code and a 1 mm length, with 2 repeaters per wire. Compute the power consumption for a non segmented 16 bits bus, then optimize the cross talk parameter for this bus, and finally dump the Verilog RTL code for the bus in the 'design.v' file.



## 12. Appendix C: CD-ROM with sources (Confidential)



### **13. References:**

[1]: Further studies on Power Consumption of Fault Tolerant codes: the Active Elements  
D. Rossi, V.E.S. Van Dijk, R.P. Kleihorst, A. K. Nieuwland, C. Metra

[2]: Deliverable 2 of FT-EA project

[3]: Optimal ECC for Crosstalk Impact Minimization  
D. Rossi, A. K. Nieuwland, V. E. S. van Dijk, R. P. Kleihorst, C. Metra

