



DELIVERABLE
IST-2001-38930
FT-EA

Deliverable
FTEA-Philips-30June04-D8

Prototyping for Fault-tolerance and Combined Electrical Aspects



Project Number	:	IST-2001-38930
Project Title	:	FT-EA

Internal Project Number	:	FTEA-Philips-30June04-D8
Date	:	June 2004
Title	:	Prototyping for Fault-tolerance and Combined Electrical Aspects
Contributing Work Package	:	WP4
Deliverable type	:	Report + Software
Security	:	Report (Public), Software (Confidential)
Author(s)	:	Damien Leroy

Abstract:

This document is deliverable D8 of Work package 4 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.

This document is the 8th deliverable of the FT-EA project. It reports the work done by iRoC Technologies for the 4th work package of the project, ie. the Unified Electrical Aspects of fault tolerant bus design, such as power consumption, crosstalk delays, and SSO noise. This work is based on the earlier project way points where each aspect has been studied and dealt with at once [D2], [D4], [D6]. This last part of the project has the objective of finding a fault tolerant coding technique able to minimize the impact of all these electrical aspects.

We will see that we haven't found a brand new ECC able to minimize all these parameters, but we will propose a design methodology and the tools that will help the designer to find the best bus architectures regarding its main design constraints. This flow allows the designer to explore the design space, and B-RoCKIT features have been developed to ease this task and to make these decisions very efficient.

First we will present the B-RoCKIT tool, its features and its goal. Then the tool inputs and outputs and the tool flow, and finally we will propose a design flow and the corresponding tools able to guide the designer toward the fault-tolerant bus architecture that suits best his design.

The confidential part of deliverable D8 is added as an appendix (CD-ROM) containing the software of the prototyping tool.

Keywords:

Fault Tolerance, Simultaneously Switching Output (SSO) noise, Power consumption, Cross talk, VDSM, Error correcting codes, Dual Rail, Hamming



Table of contents

1.	Introduction	4
2.	B-RoCKIT overview.....	5
3.	B-RoCKIT System Specification & Architecture.....	6
	3.1 Inputs	6
	3.1.1 Setup File	6
	3.1.2 Command Line Options	6
	3.2 Outputs	7
	3.3 B-RoCKIT flow	8
4.	Unified electrical aspects	9
	4.1 The Trade-off issue.....	9
	4.2 Scripting B-RoCKIT	9
	4.2.1 Result option	9
	4.2.2 TCL examples	11
5.	Running B-RoCKIT on RoC-S81.....	13
	5.1 The Trade-off issue.....	13
	5.2 Results.....	13
6.	Conclusion	14
7.	References.....	15
8.	Appendix A : BRoCKIT user guide.....	16
	8.1 Introduction: A glance at B-RoCKIT.....	16
	8.2 Supported platforms	16
	8.3 Package installation.....	16
	8.4 Installed directories.....	17
	8.5 Quick Start.....	18
	8.6 B-RoCKIT setup file	20
	8.7 Schematic view of resulting protected bus	20
	8.8 Design exploration	21
	8.9 Technical support and upgrades	22
9.	Appendix B : Description of B-RoCKIT directory structure.....	23
10.	Appendix C : BRoCKIT Software	24



1. Introduction

This document is the 8th deliverable of the FT-EA project. It reports the work done by iRoC Technologies for the 4th way point of the project, ie. the Unified Electrical Aspects of fault tolerant bus design, such as power consumption, crosstalk delays, and SSO noise.

This work is based on the earlier project way points where each aspect has been studied and dealt with separately [D1-D7]. This last part of the project has the objective of finding a single fault tolerant coding technique able to minimize the impact of all these electrical aspects.

We will see that we haven't found a brand new ECC able to minimize all these parameters, but we will propose a design methodology and the tools that will help the designer to find the best bus architectures regarding its main design constraints. This flow allows the designer to explore the design space, and B-RoCKIT features have been developed to ease this task and to make these decisions very efficient.

First we will present the B-RoCKIT tool, its features and its goal. Then the tool inputs and outputs and the tool flow, and finally we will propose a design flow and the corresponding tools able to guide the designer toward the fault-tolerant bus architecture that suits best his design.



2. B-RoCKIT overview

This section presents a rapid functional overview of B-RoCKIT. Given a fault-tolerant bus specification together with a parameter set describing the technology features, B-RoCKIT implements the processes of:

- Estimating the power consumption of the bus when an error-correcting code (ECC) is embedded. Actually, B-RoCKIT delivers estimations made with 3 different types of ECC: (1) Hamming code, (2) Dual-rail (with single parity) code, and (3) modified Dual-Rail code (dual parity). B-RoCKIT is capable of processing either the genuine bus configuration or a partition of that bus into a set of sub-busses that are functionally equivalent.
- Estimating the crosstalk impact of bus wires on the design. B-RoCKIT first estimates then minimizes this impact by rearranging spacing between wires. This optimization is processed with constant footprint.
- Estimating SSO noise effects on the bus architecture. The tool estimates an SSO noise parameter and propose a solution to handle it (e.g. Bus inversion). The user may then ask B-RoCKIT to implement this feature.
- Generating and connecting ECC logic to busses. For the best characterized ECC and bus partition solution, M-RoCKIT must deliver an HDL output file that fully describes the ECC logic (mainly the encoders/decoders) at RT-level. It also includes the optimizations made by B-RoCKIT, mainly by back-end information and RTL modifications.

B-RoCKIT automates these hardware design and decision-making stages and supports a user-friendly option set that enables the user to rapidly specify her/his queries and use the tool in both step-by-step mode and single-run mode. B-RoCKIT is designed to be quickly scripted and incorporated in sophisticated fault-tolerant design flows.

TCL scripts are available in the tool package to show design exploration possibilities, such as the search of the best footprint, or an overview of the most efficient ECC configuration.



3. B-RoCKIT System Specification & Architecture

This section details B-RoCKIT specification and describes the overall flow it implements.

3.1 Inputs

B-RoCKIT starts reading the user's options and preferences from both a setup file and the command line.

3.1.1 Setup File

The setup file specifies features that are technology-dependent:

- Parameters of capacitance model, e.g., normalized capacitance parameters λ , γ , and δ for minimum spacing.
- Bus geometric features, e.g., minimal spacing between wires and minimal wire width
- Voltage supply and input/internal power dissipation of wire driver/receiver.
- Features of logic gates used for the ECC logic (e.g., XOR gate dissipation).
-

3.1.2 Command Line Options

Command line options enable the user to quickly specify B-RoCKIT mode (i.e., estimation or architecture generation), together with the specification of the bus configuration and constraints:

- Bus configuration:
 - Number of data bus wires, and number of repeater along each wire.
 - Frequency.
 - Bus partition into sub-busses expressed as a list of wire groups.
- Constraints:
 - Footprint formulated as the maximal number of wires that can be implemented.
 - Length of the bus.
 - ECC type.
- Flow control
 - Power option: estimates and minimize power dissipation
 - Crosstalk option: estimates and minimize crosstalk impact
 - SSO option: estimates and minimize SSO noise
- Reporting
 - Verbose mode: displays the whole flow, with intermediate results.
 - Silent mode: displays nothing, but allows the graphical mode.
 - Graphical mode: displays a small graphical report of the run.
 - Result mode: append SSO, Crosstalk and Power results to the specified result file.
 - Help option: displays the help file (picture below)

In the following picture you can see the command line construction rules:



```

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"

```

Fig. 1: B-RoCKIT command line

3.2 Outputs

Depending on the selected mode and the technology and bus specification, B-RoCKIT produces:

- Estimated power consumption cost considering ECC-protected systems. The estimation function covers the case when a bus partition is supplied.
- A Verilog HDL file describing the fault-tolerant circuitry to be “glued” to the bus. The HDL described is supplied for the user-specified bus configuration.
- Back-End information describing the layout of the current bus architecture.

B-RoCKIT also delivers a reporting (displayed on the standard output) that describes the tool execution steps and features of the resulting fault-tolerant solutions (bus partition, power estimation).



3.3 B-RoCKIT flow

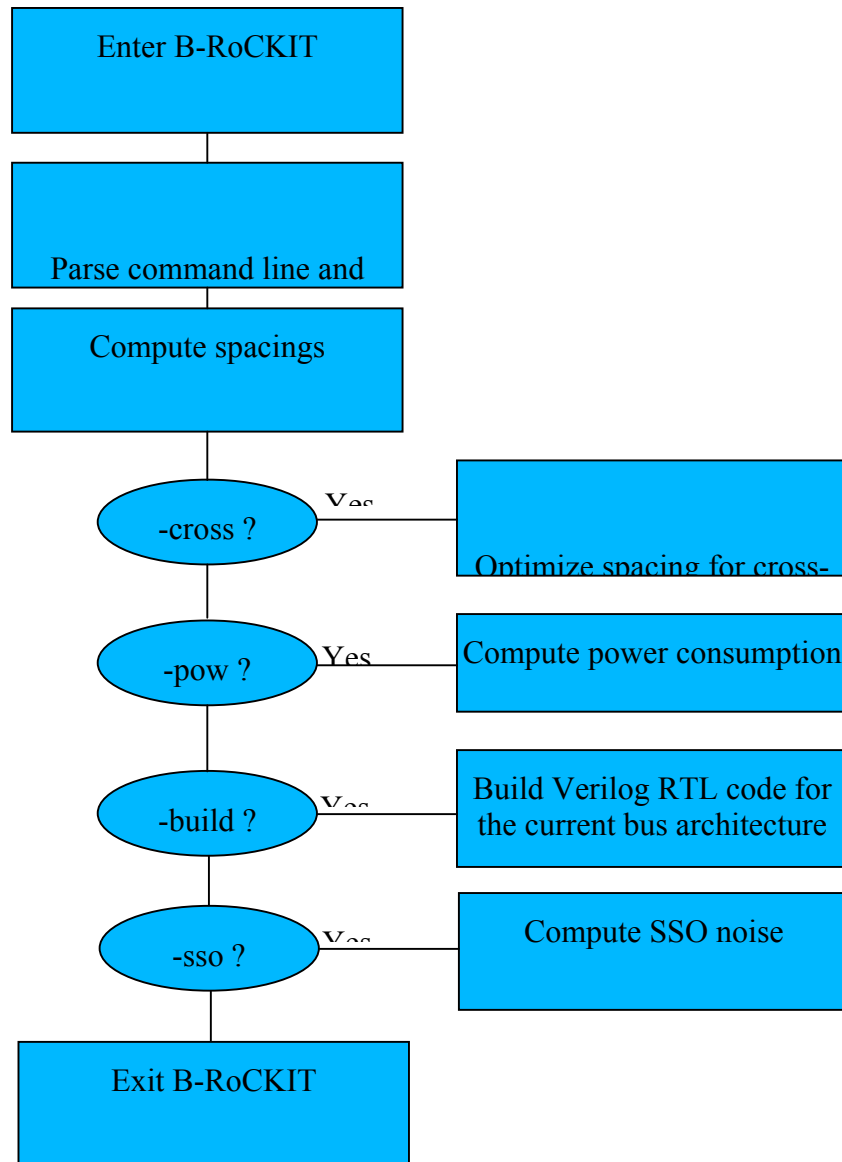


Fig. 2: B-RoCKIT flow



4. Unified electrical aspects

4.1 *The Trade-off issue*

In [D7], we have explained why there is no single ECC that minimizes all electrical aspects. The main issue is that power dissipation and crosstalk effect minimization are compatible, but this minimization (i.e. using dual-rail based ECCs) increases the SSO noise in the architecture. We can divide this noise by two by using the bus inversion approach. However, this is not the optimal solution, since the combination of Hamming code with the bus inversion approach results not only on lower SSO butr higher power consumption as well. Thus, there is no code that is optimal for all situations. As a matter of fact, the best solution depends on the context of a design, and the designer has to make the final choice corresponding to the best tradeoff with respect to the constraints of a given design. This requires exploring the space of possible solutions, in order to determine the information needed to make the final choice. To be practical, this exploration has to be done automatically. The goal of this task is the development of such module in the B-RoCKIT tool and its validation through benchmarking.

Design exploration, bus architecture answers to several constraints:

- Bus footprint
- Data width
- speed
- power dissipation
- signal integrity
- Silicon technology

Some constraints may vary in a design, but some parameters are fixed in the first specifications, such as data width, technology and eventually speed power budget, etc. When the designer knows his most stringent constraints, he can use them during the design space exploration in order to determine the most suitable architecture.

As an example, if you know you have a hard constraint on power consumption and your design will be pad limited, you can use more silicon to minimize power by increasing the bus footprint.

Then, he can use B-RoCKIT to find the best solution according to these constraints.

4.2 *Scripting B-RoCKIT*

B-RoCKIT uses very simple command line building rules. This allows the designer to write scripts that launch B-RoCKIT multiple times with different parameters. This section explains how to build these scripts and shows some examples.

4.2.1 **Result option**

B-RoCKIT was upgraded to enable the exploration of the design space. For that we have added the option `-result`, which writes all parameters of the input architecture in a file. The file is opened in append mode so the user can launch a whole exploration and have the results in a single file.

This file is written in CSV language that is easily understood by common spreadsheets.



```

1 ecc_type;footprint;power;crosstalk;SSO noise
2 mdrc;100.00;263.74; 6.60; 64.00
3 mdrc;101.00;262.76; 6.51; 64.00
4 mdrc;102.00;261.82; 6.42; 64.00
5 mdrc;103.00;260.92; 6.34; 64.00
6 mdrc;104.00;260.05; 6.26; 64.00
7 mdrc;105.00;259.22; 6.18; 64.00
8 mdrc;106.00;258.41; 6.11; 64.00
9 mdrc;107.00;257.64; 6.04; 64.00
10 mdrc;108.00;256.90; 5.97; 64.00
11 mdrc;109.00;256.18; 5.90; 64.00
12 mdrc;110.00;255.48; 5.84; 64.00
    
```

Fig. 3: B-RoCKIT result file

Figure 3 shows the results of a footprint exploration launched on a benchmark bus architecture. Here the designer uses the footprint as the exploration parameter, to compute the best footprint in order to minimize power. Only the dual rail code is explored in this example. The results are in Excel compatible format, from which the following picture is generated.

Power, crosstalk and SSO noise = f(footprint)

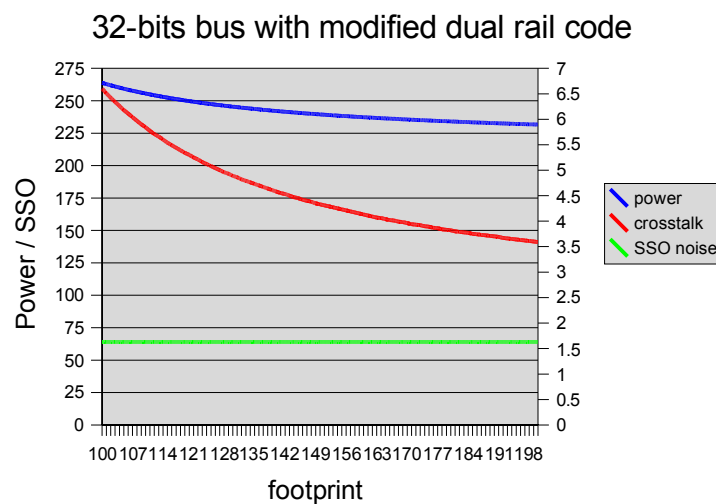


Fig. 4: Footprint exploration on ROC-S81 design

In this case, if the design constraints impose to dissipate less than 250 EGP while minimizing crosstalk impact, the designer will select a footprint of at least 120 units.

Since in the above example only the dual-rail code was explored, the designer does not dispose all the information needed to make the best option with respect to various electrical aspects. For these purposes, the designer can launch the exploration of all ECC architectures. The results of this exploration are shown in Figure 5. The different codes are reported on the x axis.



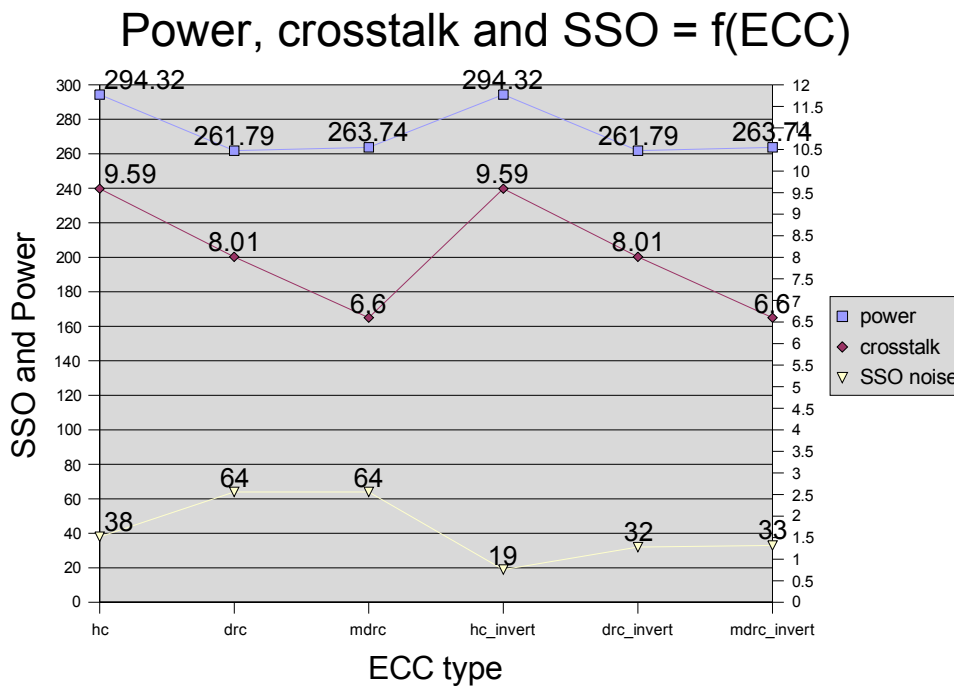


Fig. 5: ECC exploration on ROC-S81 design

In this figure, we can observe that no code is optimal for all electrical aspects. Thus, according to our constraints we will select one code or another. For instance, if our most stringent requirement is power, we will select the dual-rail code. If our most stringent requirement concerns crosstalk (e.g. if we need to increase speed), then we will select the modified dual-rail code.

If our most stringent requirement concerns SSO (e.g. for reliability reasons), then we will select the Hamming code with bus inversion. Finally, if our speed budget requires limiting cross talk to less than 7 and SSO noise to less than 35, then we will select the modified dual-rail code with bus inversion.

These examples illustrate the easiness of design space exploration and decision-making, when using the new options of the tool. These options unify the approaches developed in the project, by allowing the user to make the best tradeoffs with respect to his design constraints.

4.2.2 TCL examples

In this section we will present some TCL examples used for generating the pictures displayed above.

The first script launches B-RoCKIT for each ECC type available. The second script explores the footprint possibilities for a specific design.

```

proc explorer_ecc {} {
    set base_cmd_line "/users/dleroy/FTEA/dev/bin/brockit_local -s setup.txt -f 166 -foot 100 -map \[0
31\] -l 1000 -buf 0 -cross -power -sso -sil -result results.csv"

    set cmd_line $base_cmd_line
    lappend cmd_line -ecc hc
    if { [catch {eval exec $cmd_line} msg] } {
        error "Error in explorer execution:\n$msg"
    }
}
    
```



```

}

set cmd_line $base_cmd_line
lappend cmd_line -ecc drc
if { [catch {eval exec $cmd_line} msg] } {
    error "Error in explorer execution:\n$msg"
}

set cmd_line $base_cmd_line
lappend cmd_line -ecc mdrc
if { [catch {eval exec $cmd_line} msg] } {
    error "Error in explorer execution:\n$msg"
}

set cmd_line $base_cmd_line
lappend cmd_line -ecc hc -inv
if { [catch {eval exec $cmd_line} msg] } {
    error "Error in explorer execution:\n$msg"
}

set cmd_line $base_cmd_line
lappend cmd_line -ecc drc -inv
if { [catch {eval exec $cmd_line} msg] } {
    error "Error in explorer execution:\n$msg"
}

set cmd_line $base_cmd_line
lappend cmd_line -ecc mdrc -inv
if { [catch {eval exec $cmd_line} msg] } {
    error "Error in explorer execution:\n$msg"
}
}

```

Fig. 6: TCL script for ECC exploration

```

proc explorer_footprint {min max} {
    for {set i $min} {$i <= $max} {incr i} {

        set cmd_line "/users/dleroy/FTEA/dev/bin/brockit_local -s setup.txt -f 166 -foot 100 -map \[0
31\] -l 1000 -buf 0 -cross -power -sso -sil -result results.csv -ecc mdrc"

        lappend cmd_line -foot $i
        if { [catch {eval exec $cmd_line} msg] } {
            error "Error in explorer execution: $msg"
        }
    }
}

```

Fig. 7: TCL script for footprint exploration

These scripts call B-RoCKIT with new arguments each time. The results are saved in the "results.csv" file, along with the main parameters that led to these results. The user is then able to trace which parameters conduced to the desired results.



5. Running B-RoCKIT on RoC-S81

5.1 *The Trade-off issue*

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).

5.2 *Results*

The figures used in section 4.3.1 have been produced using RoC-S81 data bus.

In our design, we were Pad-limited and the chip was used in high-performance embedded architecture so power dissipation and crosstalk delays were our major concerns. We came to the conclusion that using a modified dual-rail code with intelligent spacing was the best solution (Fig. 5).

When considering Fig. 4, we saw that our footprint had to be larger than 150 to ensure our demanded level of power dissipation and crosstalk delays. Actually, the bigger the footprint, the lesser the power and crosstalk.



6. Conclusion

When you put enough constraints, the design space reduces itself. The designer has then to choose between several solutions. With B-RoCKIT, this choice guarantees the respect of specifications, and it may also improve the design by proposing a solution more efficient than what was expected.

Exhaustive exploration of design space is mandatory to make an efficient design, and B-RoCKIT is conceived to ease at most this task. TCL scripts described here can be added to any design flow to guide the designer through the difficult task to design a fault-tolerant bus with hard design constraints. More scripts can be conceived to explore the whole design space and take intelligent design decisions.



7. References

- D1 *“Optimal ECC for Power Consumption and Power Delay Product Reduction, and Encoding/Decoding Circuitry for the optimal ECC”*,
Deliverable D1, May/July 2001, FT-EA Project.
- D2 *“Prototyping for Fault Tolerance, Power Consumption and Power-Delay product reduction”*,
Deliverable D2, September 2003, FT-EA project
- D3 *“Optimal ECC for Cross-talk Induced Peak Voltage Reduction and Encoding/Decoding Circuitry for the optimal ECC”*,
Deliverable D3, August 2003, FT-EA project
- D4 *“Prototyping for Fault Tolerance and Crosstalk induced peak voltage reduction”*,
Deliverable D4, FT-EA project
- D5 *“Optimal ECC for Simultaneously Switching Outputs and Encoding/Decoding Circuitry for the optimal ECC”*,
Deliverable D5, March 2004, FT-EA project
- D6 *“Prototyping for Fault Tolerance and Simultaneously Switching Outputs”*,
Deliverable D6, May 2004, FT-EA project
- D7 *“ECC for Combined Electrical Aspects and Encoding/Decoding Circuitry for the Optimal ECC”*,
Deliverable D7, July 2004, FT-EA project



8. Appendix A : BRoCKIT user guide

8.1 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:

- 10 Computing power consumption of a specified bus for different (Hamming, Dual-Rail and modified Dual-Rail) code types,
- 11 Optimizing (limit) the cross talk impact on the design by intelligently arranging the bus wires,
- 12 Compute the Simultaneously Switching Outputs (SSO) noise in the bus, and propose a solution (Bus Inversion) to minimize it,
5. Implementing these codes in Verilog RTL.
6. Propose an exploration option that helps gather results from multiple B-RoCKIT launches on different parameters.

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.

8.2 Supported platforms

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

8.3 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
```



8.4 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: <code>brockit</code>
<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 (<code>brockit_setup.txt</code>) and typical examples of B-RoCKIT command line.
<code>./tcl</code>	Directory containing TCL scripts samples proposing some design exploration as footprint and ECC explorations.
<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.

```
iserver::~users/dleroy/FTEA/dev/test_dir > /users/dleroy/FTEA/dev/bin/brockit_local -h
___ brockit (iRoC Technologies) ___ ran by "dleroy"
___ Bus-RoCKIT Platform (Verilog RTL)

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]
[-result | --result_file result_output_file]

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
-result| --result_file append a summary of results to the specified output file
-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"
```



8.5 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 B-RoCKIT development goes on, the tool will be upgraded in order to widens the proposed solutions, such as enhancing the compatibility with back-end tools and proposing new design options.

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.

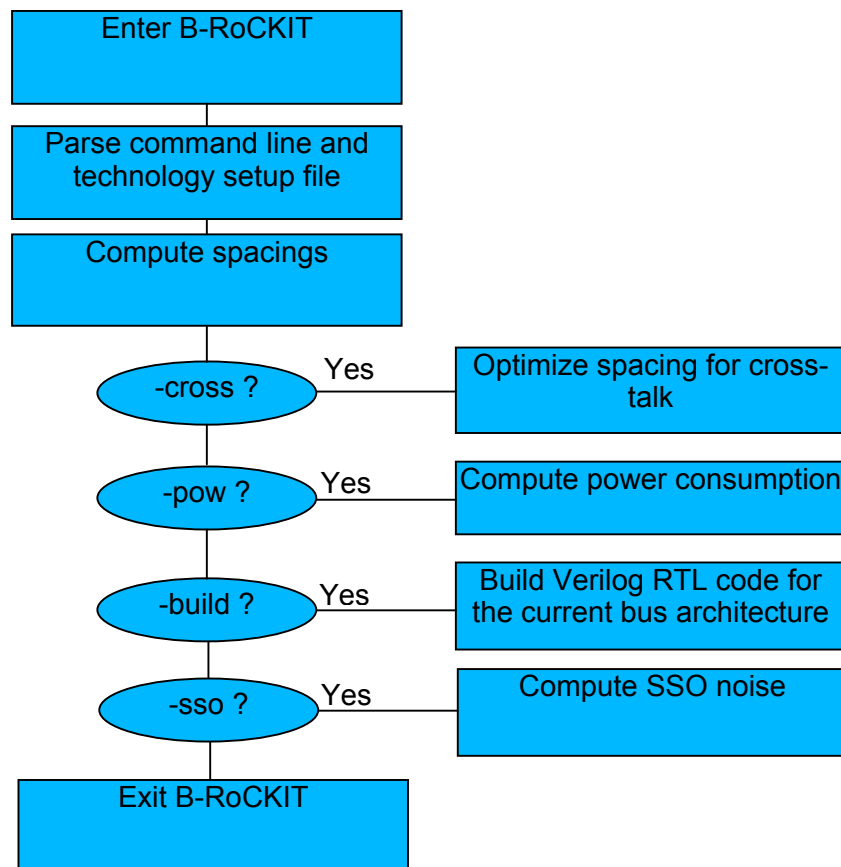
For the cross talk optimization, B-RoCKIT begins with computing power and crosstalk parameters for the current bus architecture, then it optimizes spacing between wires, and when the optimal solution has been defined, it returns the new parameters.

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:

9. 32 bits bus
10. 400 MHz frequency
11. 10 mm long
12. 4 repeaters per wire
13. 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.



8.6 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

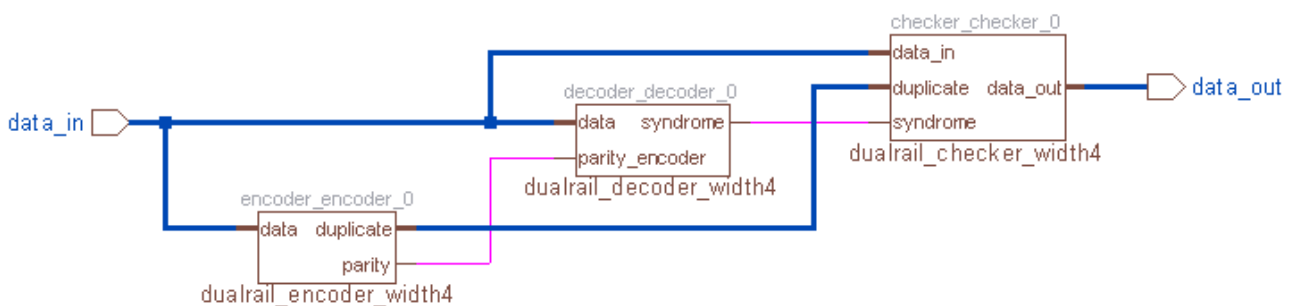
# value of Cebmax in fF/um
Cebmax = 0.0198

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

8.7 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.):



8.8 Design exploration

B-RoCKIT has been developed in order to help the designer explore design space to find the solution best suited to his needs. This task may be achieved by running B-RoCKIT multiple times with slightly modified parameters to meet design requirements at minimal cost.

As an example we see that enlarging bus footprint diminishes power dissipation and crosstalk delays, which are major concerns in high performance architectures. Running the footprint exploration TCL script sample present in this package will help find the threshold footprint over which the bus meets design constraints.

The first step is to run a TCL shell, with the following command:

```
> tclsh
```

then you have to source the explorer.tcl file:

```
% source explorer.tcl
```

and launch the explorer_footprint procedure with min and max footprints.

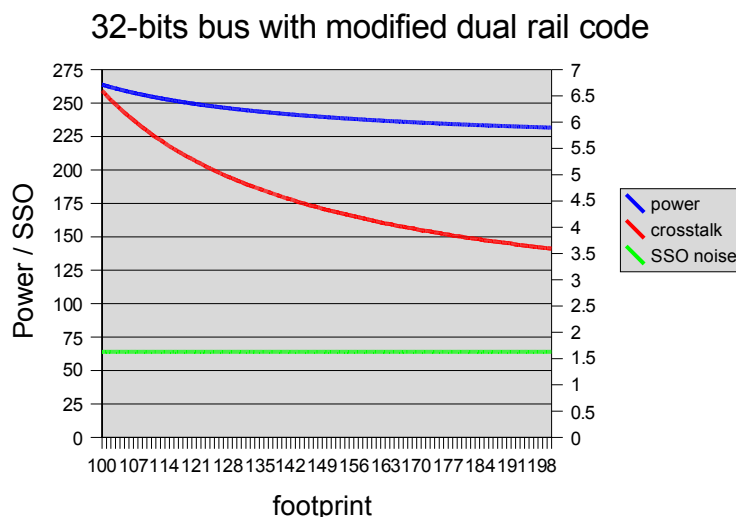
```
% explorer_footprint 100 200
```

After running this procedure, you may notice a new file in the current directory, results.csv:

```
1 ecc_type;footprint;power;crosstalk;SSO noise
2 mdrc;100.00;263.74; 6.60; 64.00
3 mdrc;101.00;262.76; 6.51; 64.00
4 mdrc;102.00;261.82; 6.42; 64.00
5 mdrc;103.00;260.92; 6.34; 64.00
6 mdrc;104.00;260.05; 6.26; 64.00
7 mdrc;105.00;259.22; 6.18; 64.00
8 mdrc;106.00;258.41; 6.11; 64.00
9 mdrc;107.00;257.64; 6.04; 64.00
10 mdrc;108.00;256.90; 5.97; 64.00
11 mdrc;109.00;256.18; 5.90; 64.00
12 mdrc;110.00;255.48; 5.84; 64.00
```

If you launch your spreadsheet and import the file under the tool format, you can easily build a

Power, crosstalk and SSO noise = f(footprint)



graphic and find the threshold footprint. Here is an example of the picture you can obtain:

If your maximal authorized power dissipation is 250 EGP at 166MHz and that your maximal crosstalk delay is 5.5 (no dimension), you find that the 120 footprint is the optimal one for modified dual-rail coding.

To be really accurate, this exploration has to be repeated for each code to find the best combination of every parameter.



8.9 Technical support and upgrades

Bugs should be reported to iRoC Technologies support team at support@iroctech.com.

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

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

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



9. 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).
- Release/tcl: contains a sample TCL script file, containing two design exploration functions ready to use in the test_dir directory. These functions explore ECC type and footprint.



10. Appendix C : BRoCKIT Software

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

