

DELIVERABLE  
IST-2001-38930  
FT-EA

Deliverable D2  
FTEA-Philips-29sept03-D2

# Prototyping for Fault Tolerance, Power Consumption and Power- Delay Product Reduction



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

|                                  |   |  |
|----------------------------------|---|--|
| <b>Internal Project Number</b>   | : | FTEA-Philips-29-sept03   |
| <b>Date</b>                      | : | September 29, 2003   |
| <b>Title</b>                     | : | Prototyping for Fault Tolerance, Power Consumption and Power-Delay Product Reduction |
| <b>Contributing Work Package</b> | : | WP..   |
| <b>Deliverable type</b>          | : | Report   |
| <b>Security</b>                  | : | Main report: Public / Appendix: Confidential   |
| <b>Author(s)</b>                 | : | D. Leroy, M. Cherif, M. Nicolaidis   |

**Abstract:**

This document is deliverable D2 of Work package 1 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 and the generation of the hardware description of the selected bus in a high-level hardware description language (HDL).

The confidential part of deliverable D2 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, HDL generation, bus, RTL description.



## Table of contents

|   |    |
|---|----|
| Summary .....   | 4  |
| 1 RoCKIT: Infrastructure IP Generation Platform ..... | 5  |
| 1.1 Introduction .....                                | 5  |
| 1.2 B-RoCKIT Overview .....                           | 5  |
| 2 B-RoCKIT System Specification & Architecture .....  | 6  |
| 2.1 Inputs .....                                      | 6  |
| 2.1.1 Setup File .....                                | 6  |
| 2.1.2 Command Line Options .....                      | 6  |
| 2.2 Outputs .....                                     | 6  |
| 2.3 Architecture Overview .....                       | 6  |
| 2.4 Main Components and Implemented Algorithms .....  | 7  |
| 2.4.1 Flow Manager .....                              | 7  |
| 2.4.2 Power Estimator .....                           | 7  |
| 2.4.3 Architecture Generator .....                    | 8  |
| 3 B-RoCKIT Benchmarks .....                           | 9  |
| 3.1 Validation .....                                  | 9  |
| 3.2 Benchmarking .....                                | 9  |
| 3.2.1 Benchmark presentation .....                    | 9  |
| 3.2.2 Benchmark results .....                         | 9  |
| 4 Conclusions .....                                   | 13 |
| 5 Appendix: Software Overview .....                   | 14 |
| 5.1 Introduction: A glance at B-RoCKIT .....          | 14 |
| 5.2 Supported platforms .....                         | 14 |
| 5.3 Package installation .....                        | 14 |
| 5.4 Installation directories .....                    | 15 |
| 5.5 Installation test .....                           | 15 |
| 5.6 Quick Start: B-RoCKIT flow .....                  | 16 |
| 5.7 An example .....                                  | 17 |
| 5.7.1 Setup file format .....                         | 17 |
| 5.8 Schematic view of resulting protected bus .....   | 18 |
| 5.9 Technical support and upgrades .....              | 18 |
| 6 Appendix: CDROM with sources (Confidential) .....   | 19 |



## Summary

This document contains is the second deliverable of Work package 1 of the FT-EA project, sponsored by the European Commission under IST-2001-38930. 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 research has led to publications on power consumption of fault tolerant busses. The findings described in these publications, a.o. Deliverable D1, are implemented in a prototype development tool which is described in this deliverable.

This deliverable describes the architecture of the tool as well as the way the designer interacts with it using command line options and configuration files. The tool evaluates the design choices the designer has to make, after which the selected configuration is generated in a synthesisable form using a Hardware description language. To illustrate the feedback the designer obtains from the tool, various results are presented for different configurations of fault tolerant busses. This part of the description is public.

The confidential software of the prototyping tool is presented in the appendix of this Deliverable in the form of a CD-ROM containing the sources, binaries and a description of the file structure.



# 1 RoCKIT: Infrastructure IP Generation Platform

## 1.1 Introduction

RoCKIT is a toolsuite for generating fault tolerant systems. It consists of three parts: one part is dedicated to logic (L-RoCKIT), one part to memories (M-RoCKIT). The last part, B-RoCKIT, is being developed within the FT-EA project and is focusing on generating fault tolerant interconnect. B-RoCKIT allows the designer to efficiently explore the design choices of fault tolerant busses. It compares different error correcting schemes, and presents the trade-off of power and area, taking into account speed constraints. Furthermore, it generates the high level description of the fault tolerant bus.

## 1.2 B-RoCKIT Overview

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 error-correcting code (ECC) is embedded. Actually, B-RoCKIT delivers estimation made with 2 different types of ECC: (1) Hamming code and (2) Dual-rail and parity combining code. 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. In a first approach, the user supplies this partition.
- Generating and connecting ECC logic to busses. For the best characterized ECC and bus partition solution, B-RoCKIT delivers a HDL output file that fully describes the ECC logic (mainly the encoders/decoders) at RT-level.

B-RoCKIT automates these two 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-tolerance flows.



## 2 B-RoCKIT System Specification & Architecture

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

### 2.1 Inputs

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

#### 2.1.1 Setup File

The setup file specifies features that are technology-dependent:

- Parameters of capacitance model, e.g., normalized capacitance parameters  $\lambda$ ,  $\gamma$ , and  $\delta$  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).
- 

#### 2.1.2 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 genuine 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.
  - Estimation of critical path delay expressed as the maximal value of ECC logic depth.

### 2.2 Outputs

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

- Estimated power consumption cost considering Hamming and Dual-rail systems. The estimation function covers the case when a bus partition is supplied.
- A HDL file describing the fault-tolerant circuitry to be "glued" to the bus. The HDL described is supplied for the user-specified bus configuration.

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).

### 2.3 Architecture Overview

An overview of B-RoCKIT system architecture is depicted in the below figure.



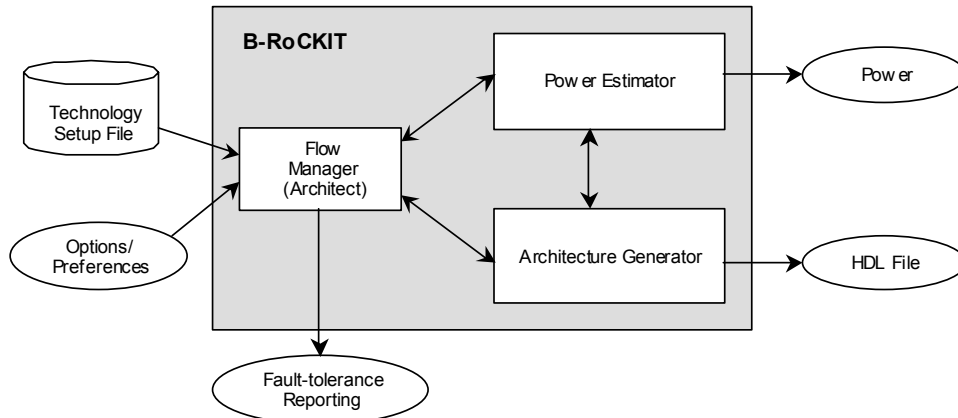


Figure 1: B-RoCKIT Flow Diagram

## 2.4 Main Components and Implemented Algorithms

B-RoCKIT is divided into 3 major components/functions that implement the different mentioned features.

### 2.4.1 Flow Manager

This implements the core algorithm of B-RoCKIT. It is in charge of reading the inputs (setup file and options) and launching the appropriate fault-tolerance function(s) selected by the user (estimation and generation). It manages the full B-RoCKIT flow.

### 2.4.2 Power Estimator

This implements the power dissipation cost using Hamming and Dual-rail systems. It handles the case when the genuine bus is broken down into sub-busses. For each ECC technique, the estimator computes and reports the values for: wire dissipation, buffer dissipation, ECC logic dissipation, and total dissipation.

The following procedure describes the pseudo-code of the algorithm implemented by the estimator.

**Procedure** ECC\_ESTIMATOR (ECC type, technology parameters, bus configuration)

**Begin**

$P_{tot} = 0$  /\* initial setting of total dissipation to 0 \*/

**Foreach** sub-bus  $B_i$  in specified bus partition **do**

**if** ECC is Dual-rail **then**

    Compute parameters specific to Dual-rail /\* as defined by Rossi et al. \*/

    Compute  $P_{tot}(B_i) = P_{wires}(B_i) + P_{buffers} + P_{logic}(B_i)$  /\* specific to Dual-rail \*/

**elseif** ECC is Hamming **then**

    Compute parameters specific to Hamming /\* as defined by Rossi et al. \*/

    Compute  $P_{tot}(B_i) = P_{wires}(B_i) + P_{buffers} + P_{logic}(B_i)$  /\* specific to Hamming \*/

**endif**

$P_{tot} = P_{tot} + P_{tot}(B_i)$  /\* add  $P_{tot}(B_i)$  to  $P_{tot}$  \*/

Report  $P_{tot}(B_i)$

**Endforeach**

Report  $P_{tot}$

**Return**  $P_{tot}$

**End**

**Procedure** ESTIMATOR (technology parameters, bus configuration, constraints)

**Begin**

$P_{tot}(\text{Hamming}) =$

    ECC\_Estimator (Hamming, technology parameters, bus configuration)

$P_{tot}(\text{Dual-rail}) =$

    ECC\_Estimator (Hamming code, technology parameters, bus configuration)

Report best ECC that fits footprint constraints

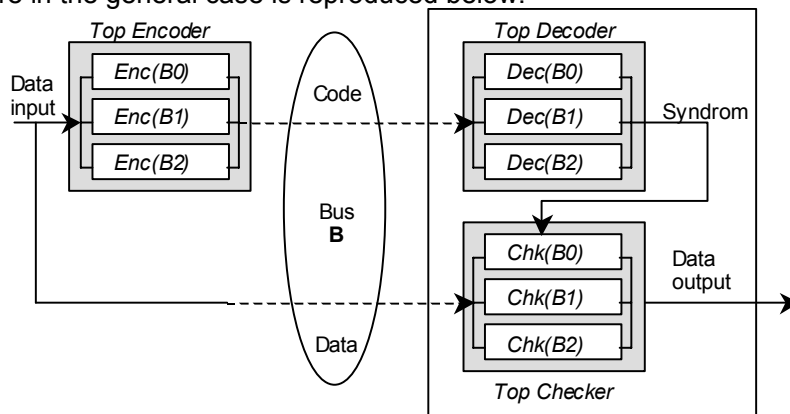
Report critical path for each ECC

**End**



### 2.4.3 Architecture Generator

This implements the ECC architecture for the genuine bus (and the genuine bus partition provided by the user) if the option has been selected. The generated architecture in the general case is reproduced below:



**Figure 2: Hierarchical scheme of the generated ECC architecture**

The following procedure describes the pseudo-code of the algorithm implemented by the architecture generator.

```

Procedure ARCH_GENERATOR (ECC type, bus partition { $B_i$ })
  Begin
    /* Phase 1: Architecture building */
    Create a top module ECC(top)
    Create 3 modules: Encoder, Decoder and Checker
    Foreach sub-bus  $B_i$  do
      /* Begin with Encoder */
      Create Enc( $B_i$ ) in Encoder
      Build Enc( $B_i$ )
      Propagate signals on the inputs/outputs of Encoder
      /* here Enc( $B_i$ ) is fully built, we can instantiate it within Encoder */
      Instantiate Enc( $B_i$ ) in Encoder
      /* Continue with Decoder */
      Create and build Dec( $B_i$ ), and instantiate it in Decoder
      Propagate signals on the inputs/outputs of Decoder
      /* Finish with Checker */
      Create and build Chk( $B_i$ ), and instantiate it in Checker
      Propagate signals on the inputs/outputs of Checker
    Endforeach
    Instantiate Encoder, Decoder and Checker in ECC(top)
    Build a unique Bus B of the correct width
    Interconnect Encoder, Decoder and Checker in ECC(top) with B

    /* Phase 2: HDL generation of blocks Enc( $B_i$ ), Dec( $B_i$ ), Chk( $B_i$ ) */
    Foreach block sub-bus  $B_i$  do
      Generate HDL description of Enc( $B_i$ ), Dec( $B_i$ ), and Chk( $B_i$ ) into output file
    Endforeach

    /* Phase 3: HDL generation of blocks Encoder, Decoder and Checker */
    Generate HDL description of Encoder, Decoder, and Checker into output file

    /* Phase 4: HDL generation of top module */
    Generate HDL description of ECC(top) and store resulting code in output file
  End

```



## 3 B-RoCKIT Benchmarks

### 3.1 Validation

B-RoCKIT has been validated with respect to the examples studied for Deliverable D1. The results of B-RoCKIT are within a 2% margin compared with the examples studied for D1. This is well within the accuracy of the approximations made for developing the theoretical model.

### 3.2 Benchmarking

#### 3.2.1 Benchmark presentation

Within the FT-EA project, it was decided to use an existing bus for benchmarking the tool. The specifications of the selected bus are listed below.

- 0.13  $\mu\text{m}$  CMOS technology,
- 1 millimeter long, unbuffered,
- 166 MHz operating frequency,
- 32 data bits.

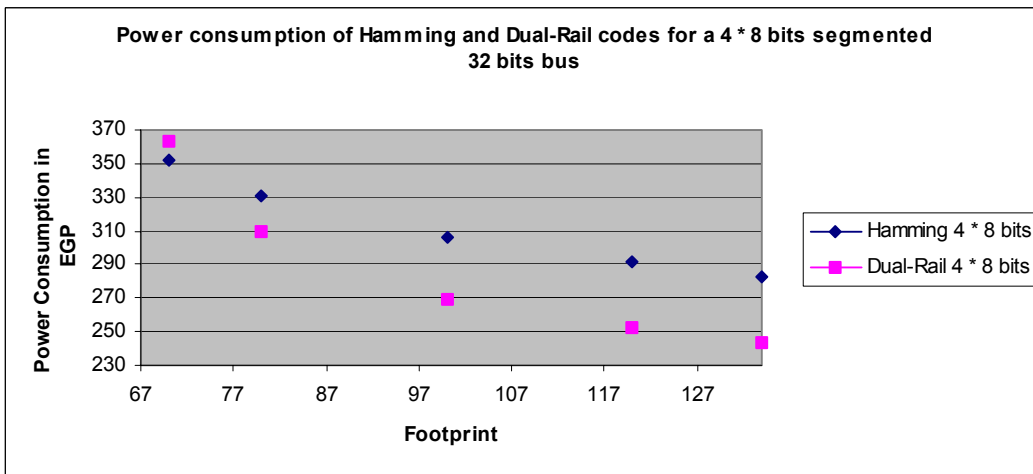
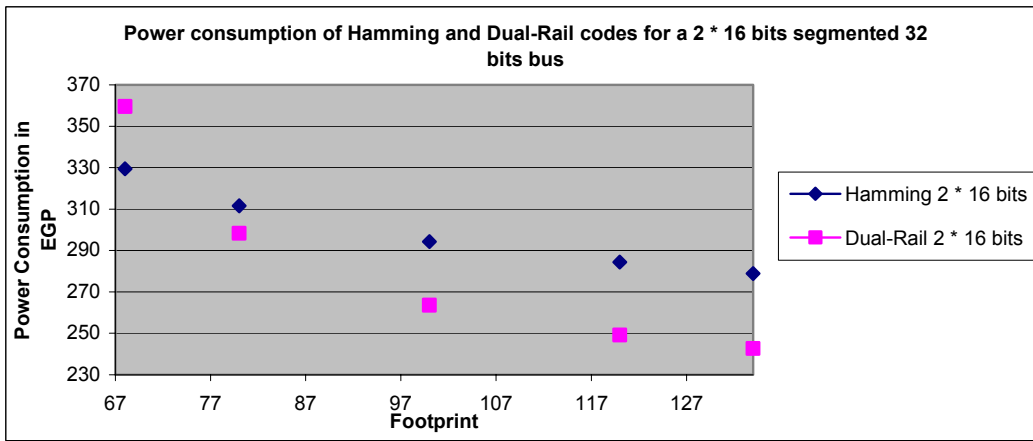
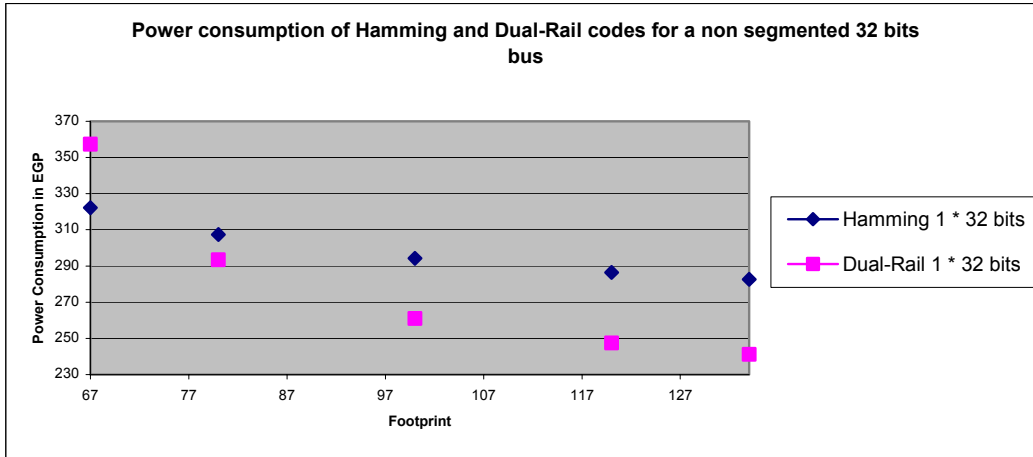
This bus is a 32 bits ARM bus, which connects a RISC fault-tolerant processor and a protected memory. The bus uses the AMBA AHB protocol (Advanced Microcontroller Bus Architecture, Advanced High-performance Bus).

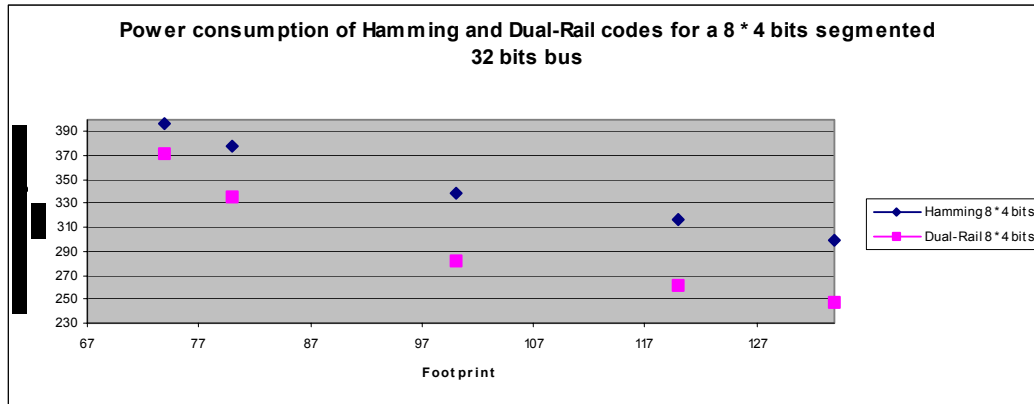
The non-protected bus has been implemented in a Fault-Tolerant SoC by iRoC, and protecting this bus is a further improvement of this design since it carries the fault-tolerant achievement between the CPU and the memory, which are both already designed to protect the integrity of the data they process. 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).

#### 3.2.2 Benchmark results

The benchmarked bus has been used in the environment of the 32 bits LEON RISC processor. The minimum footprint in which can be implemented the dual-rail code may be computed as two times the data width, plus the parity bit, plus both shields, which equals to 67. We have considered footprints between 67 and 134 for this implementation. Furthermore, we have looked into the effect of segmentation of busses. Segmentation can be done to reduce the delay of the encoder and decoder by decreasing logic depth of the parity trees in them. This can be especially beneficial for short busses, since the total bus delay is then dominated by the logic, and not yet by the propagation delay on the wires. Therefore, we have considered subdivisions of the bus in 2, 4 and 8 parts. The results for different number of subdivisions/segments are presented in Figure 3.







**Figure 3 : Power consumption of a 32 bits fault-tolerant bus for different ECCs and footprints.**

We have summed these results in a single chart below, with the relative percentage gains between the two Error Correcting Codes for different subdivisions of the 32 bits bus, for different Footprints.

**Figure 4: Compared relative power gains between DR and H codes.**

This figure shows that the more the bus is segmented, the more efficient Dual-Rail is compared to Hamming code. This comes from the fact that segmenting a bus will lower the number of data bits per code, thus lowering the efficiency of the Hamming code. The Dual-Rail code will be more efficient for low data widths than the Hamming code.

Note: In the figures above, the minimal footprint is accorded to the segmentation (e.g. for 4 \* 8 bits it is 70 instead of 67 for the original bus). We also have computed the power consumption of a bus of variable data width, which has lead to the picture below (Figure 5):

**Figure 5: Relative power gain between Hamming and Dual-Rail variable width busses.**

In this Figure 5, we can see that the Dual-Rail code is efficient as long as the footprint exceeds 2.5 times the data width. Otherwise there isn't enough space for the intelligent spacing to act. We can also see the points corresponding to the addition of a new check bit where the power gain decreases abruptly. In the end (for data exceeding 30 bits), the footprint becomes too small for the dual-rail code, and the power gain decreases consistently.

**Figure 6: Influence of bus length on the Power Consumption of a 32 bits buffered bus.**

We must also consider the case with repeaters that are mandatory in some applications. In the graph above, we can see that if the bus is buffered, there appears a heavy load on the Dual-Rail code:

In the Figure 6, we are able to see that as soon as bus length exceeds 4 millimeters (i.e. more than one repeater) the Dual-Rail code becomes inefficient. This comes from the larger number of wires that heightens the number of repeater comparing to Hamming implementation. This trend is better shown in the Figure 7 where we have fixed the bus length and varied the spacing between repeaters:

**Figure 7: Influence of repeater spacing on the power consumption of a 32 bits segmented bus.**



This picture shows that repeater spacing is one of the main factors influencing the benefits of the Dual-Rail implementation over the Hamming one. As soon as the bus is buffered, the Dual-Rail implementation becomes less interesting, and soon loses any advantage while the spacing between repeaters diminishes. The last value (spacing = 20 mm) equals to a non-buffered bus. In this example we can't see the inversion of the power gain because of the bus segmentation, but the trends stay the same.



## 4 Conclusions

The B-RoCKIT tool has been set-up and implements the theoretical results presented in Deliverable D1. The software architecture has been established, and will form the basis for implementing the results of the work packages to follow.

The results of the Benchmark show that there are 3 main factors influencing the choice of the ECC, which are the footprint/data width ratio, the repeater spacing, and finally the data width. The results show that the Dual-Rail code will be efficient for busses of small widths, and for rapid busses which require segmentation of the original bus. Furthermore, it has been shown by the tool that Dual Rail encoding is the more power efficient fault tolerant solution for short busses where no buffers are needed.

The tool presented provides a powerful tool for evaluation of the power consumption of the different fault tolerant bus implementations, and helps the designer to make the best choice.



## 5 Appendix: Software Overview

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

The goal of B-RoCKIT is to perform and manage error correcting code implementation in internal busses of integrated circuits. Roughly, B-RoCKIT version 1.0 operates at RT-level and is organized as a single stand-alone binary file performing the following tasks:

- Computing power consumption of a specified bus for different code types :
  - Hamming code
  - Dual-Rail code
- Implementing the supported codes in Verilog HDL at RT-level.

B-RoCKIT offers the following features:

- Processing bus structure according to many parameters
- Accounting for any technology
- Providing an easy/flexible monitoring of the software functions
- Offering an option set allowing a complete exploration of the design space
- User-friendly text command line interface that makes it easy for the user to script and incorporate B-RoCKIT in its design flow.

### 5.2 Supported platforms

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

### 5.3 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**



## 5.4 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:<br><b>brockit</b>   |
| <b>./doc</b>                | Documentation directory containing the user guide in PDF and the Bug Tracking Form in both PDF and MS doc formats..               |
| <b>./src (confidential)</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.  |

## 5.5 Installation test

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

```
> install_dir/brockit/bin/brockit
```

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

```
sun15::test_dir > brockit
brockit: options and arguments missing
usage: brockit [-s | --setup setupfilename] [-h | --help] [-verb | --verbose] [-sil | --silent]
[-f | --frequency frequency] [-foot | --footprint footprint]
[-depth | --logical_depth logicaldepth] [-buf | --buffer number_of_buffers_per_wire]
[-ecc | --ecc_type ecc_type] [-build | --builder_outfile outfile]
[-power | --power_estimation [ bus_partition ]] [-l | --length bus_length]

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
-depth | --logic_depth 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)
-build | --builder_out builder output file name
-power | --power_estim estimation of the power dissipated by the protected bus described here
bus_partition        [ boundary_1 boundary_2 boundary_3 ... final_boundary ]

sun15::test_dir > █
```

**Figure 8: B-RoCKIT on-line help**



## 5.6 Quick Start: B-RoCKIT flow

The ultimate goal of B-RoCKIT is to produce the “best” protected bus according to FT-EA based criterias. This goal is achieved by carefully choosing the Error Correcting Code (ECC). This choice allows to reduce the power consumption of 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 and route processes.

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

B-RoCKIT starts by reading its command line, then reading the specified 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.

If the build option has been selected, B-RoCKIT then dumps the Verilog RTL description of the chosen bus in a file specified in the command line.

If no ECC is specified, B-RoCKIT will compute power dissipation for both code and choose the best one for the implementation.

The overall flow of B-RoCKIT is depicted by the picture below (Fig. 9):

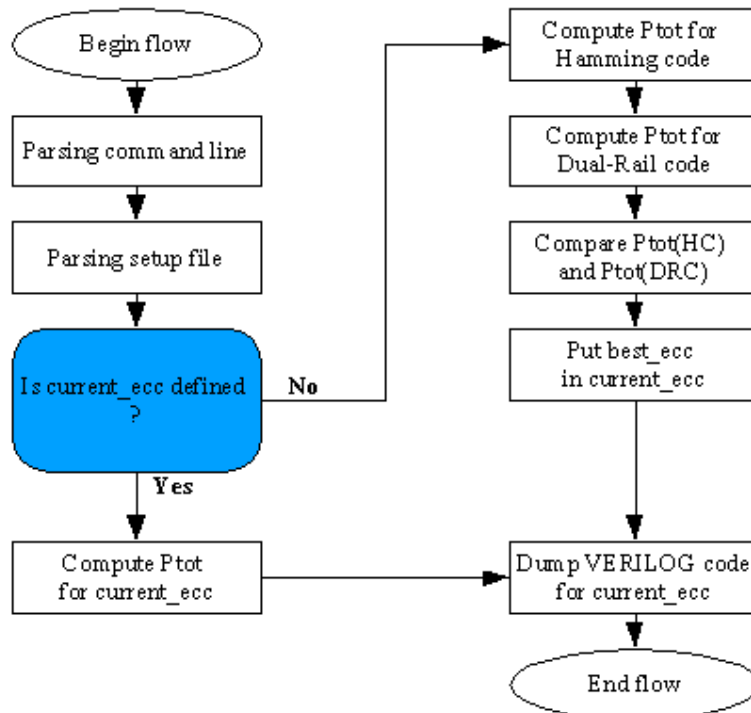


Figure 9: B-RoCKIT flow



## 5.7 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.

### 5.7.1 Setup file format

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.

The B-RoCKIT setup file is structured as:

```
# comments
name_of_value = value
```

Make sure to respect the unit given in the template, located in **install\_dir/brockit/templates/**

Below 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

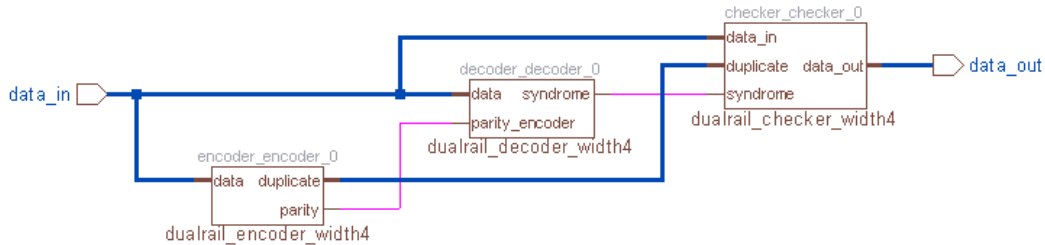
# value of Cebmin in fF/um
Cebmin = 0.00533
```



## 5.8 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.):



**Figure 10: Architecture of the generated bus**

## 5.9 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.



## 6 Appendix: CDROM with sources (Confidential)

