Deliverable – D22

Architecture level techniques for process variation tolerance in networks-on-chip

Abstract:

Manufacturing processes required to enable deep-nanometer technologies (65 nm and beyond) have to tackle significant challenges. Despite the best efforts of process engineers, large variations in device parameters will be unavoidable. In the extreme case, architectures of multi-core integrated systems will have to deal with manufacturing faults in order to sustain yield. Traditional synchronous design methodologies, that require the system to adhere to strict timing constraints over the entire chip area, suffer badly from wide ranging parameter variations. GALS systems may offer significant advantages with this respect, as they essentially divide a complex system into small, independent modules. Beyond containing the performance implications of process variations in local modules, designers now have to devote special care to the design of process-variation tolerant synchronization interfaces. This report documents the development effort of GALS design methods to improve the system performance and reliability under large process variations. Although not limited to on-chip networks, most design techniques have been specifically conceived for them in light of the key role they play for global system connectivity. Finally, the report moves from the assumption that mitigating process variations is not a task on burden of a single abstraction layer but requires a complementary course of action to be taken at each layer of the design hierarchy.

Keyword list: Process variations, logic based distributed routing, yield, routing skew, interconnect parameter mismatch, adaptive body biasing, adaptive voltage scaling
<table>
<thead>
<tr>
<th>Function</th>
<th>Responsibility</th>
<th>Date</th>
<th>Signature</th>
</tr>
</thead>
<tbody>
<tr>
<td>Written by:</td>
<td>Davide Bertozzi, Milos Stanisavljevic, Alessandro Strano</td>
<td>30.04.2010</td>
<td></td>
</tr>
<tr>
<td>Checked by:</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Approved by:</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Reserved to EC

| Approved by:   |            |            |           |
## CHANGE RECORDS

<table>
<thead>
<tr>
<th>ISSUE</th>
<th>DATE</th>
<th>§ : CHANGE RECORD</th>
<th>AUTHOR</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>30-Apr-010</td>
<td>First Draft</td>
<td>Davide Bertozzi, Milos Stanisavljevic, Alessandro Strano</td>
</tr>
</tbody>
</table>
Manufacturing processes required to enable deep-nanometer technologies (65 nm and beyond) have to tackle significant challenges. Despite the best efforts of process engineers, large variations in device parameters will be unavoidable. In the extreme case, architectures of multi-core integrated systems will have to deal with manufacturing faults in order to sustain yield. Traditional synchronous design methodologies, that require the system to adhere to strict timing constraints over the entire chip area, suffer badly from wide ranging parameter variations. GALS systems may offer significant advantages with this respect, as they essentially divide a complex system into small, loosely coupled (if not independent) modules. In this new context, beyond limiting the performance implications of process variations in local modules, designers now have to devote special care to the design of process-variation tolerant synchronization interfaces. This report documents the development effort of GALS design methods to improve the system performance and reliability under large process variations. Although not limited to on-chip networks, most design techniques have been specifically conceived for them in light of the key role they play for global system connectivity. Finally, the report moves from the assumption that mitigating process variations is not a task on burden of a single abstraction layer but requires a complementary course of action to be taken at each layer of the design hierarchy.
# TABLE OF CONTENTS

1 INTRODUCTION ............................................................................................................ 10

2 REFERENCES ............................................................................................................... 12
   2.1 ACRONYMS ......................................................................................................... 12
   2.2 REFERENCE DOCUMENTS .................................................................................. 12

3 VARIABILITY AWARE GALS LINK DESIGN............................................................... 16
   3.1 TARGET GALS PLATFORM ................................................................................ 20
   3.2 YIELD-ORIENTED EVALUATION METHODOLOGY OF NETWORK-ON-CHIP ROUTING IMPLEMENTATIONS.............................................................................................................. 21
      3.2.1 Evaluation methodology ................................................................................ 22
   3.3 ASSESSMENT OF LOGIC-BASED ROUTING IMPLEMENTATIONS WITH LAYOUT AWARENESS .................................................................................................................... 24
      3.3.1 First routing mechanisms: LBDR ................................................................. 26
      3.3.2 Second routing mechanisms: LBDindr ......................................................... 27
      3.3.3 Third routing mechanisms: Region based routing ....................................... 29
      3.3.4 Table-based vs Table-less routing: the physical perspective ..................... 32
      3.3.5 Evaluation of routing implementations ......................................................... 33
         3.3.5.1 Coverage results ................................................................................... 35
         3.3.5.2 Performance and Cost Results ............................................................... 36
      3.3.6 Implementation of logic based distributed routing inside the xpipesLite switch: architecture design techniques ........................................................................................................ 38
         3.3.6.1 Fixed Priorities, Smart Priorities and Allocators ..................................... 39
         3.3.6.2 Fixed Priorities ...................................................................................... 40
         3.3.6.3 Smart Priorities ..................................................................................... 41
         3.3.6.4 Extension To More Cores Per Switch .................................................... 42
      3.3.7 Summing up .................................................................................................... 43
   3.4 VARIABILITY DETECTOR FOR RANDOM PROCESS VARIATION-TOLERANT GALS LINK DESIGN .............................................................................................................. 45
      3.4.1 architecture of the variation detector ................................................................. 45
      3.4.2 Variation detector at work ............................................................................... 47
      3.4.3 Validation ........................................................................................................ 48
      3.4.4 Experimental results ...................................................................................... 51
         3.4.4.1 Process variations in the data link wires ...................................................... 51
         3.4.4.2 Process variations affecting the detector ..................................................... 53
      3.4.5 summing up ................................................................................................... 54
3.5 DEVELOPMENT OF A NEW PHYSICAL ROUTING APPROACH FOR ROBUST BUNDLED SIGNALLING ON NOC LINKS ................................................................. 55
   3.5.1 Motivating example .............................................................................. 55
   3.5.2 NoCs link routing flow ......................................................................... 57
      3.5.2.1 Pin placement and ordering ......................................................... 58
      3.5.2.2 Prioritizing links .......................................................................... 59
      3.5.2.3 Link routing ................................................................................. 59
   3.5.3 Crosstalk handling .............................................................................. 62
   3.5.4 Experimental results .......................................................................... 63
   3.5.5 Summing up ....................................................................................... 66
3.6 VARIABILITY COMPENSATION FOR FULL-SWING VS LOW-SWING ON-CHIP COMMUNICATION ................................................................. 69
   3.6.1 Communication channel design ......................................................... 71
   3.6.2 Inherent robustness to process variations ......................................... 72
   3.6.3 Post-silicon variability compensation .............................................. 75
      3.6.3.1 Experimental framework ............................................................. 76
      3.6.3.2 Compensation efficiency in full-swing links............................. 77
      3.6.3.3 Compensation efficiency in low-swing links............................ 78
      3.6.3.4 Role of random variations ......................................................... 79
   3.6.4 Variability Compensation with crosstalk ........................................ 79
      3.6.4.1 Signal integrity .......................................................................... 80
      3.6.4.2 Compensating crosstalk-affected links .................................... 82
      3.6.4.3 Summing up .............................................................................. 84
4 VARIABILITY AWARE TECHNIQUE FOR DIGITAL LOGIC ......................... 85
   4.1 MOTIVATION .......................................................................................... 85
   4.2 IMPACT OF LOCAL VARIATIONS ON THE MAXIMUM CRITICAL PATH DELAY DISTRIBUTION ............................................................. 86
   4.3 MINIMIZATION OF THE IMPACT OF LOCAL DELAY VARIATIONS ........ 87
   4.4 OPTIMIZATION RESULTS OF VARIATIONS MINIMIZATION .............. 89
5 CONCLUSIONS ............................................................................................ 91
LIST OF FIGURES

Figure 1. Target GALS Platform ........................................................................................................ 20
Figure 2. Mechanisms used in the yield-oriented evaluation methodology ........................................ 23
Figure 3. Example of the effect of variability on link frequencies ....................................................... 25
Figure 4. LBDR core routing logic .................................................................................................... 27
Figure 5. Topology not supported by LBDR ....................................................................................... 27
Figure 6. Deroute logic ...................................................................................................................... 28
Figure 7. (a) Two deroute options (b) Deroute solution (c) Topology not supported by LBDRrdr ......... 28
Figure 8. Example region definition .................................................................................................. 29
Figure 9. Hardware implementation of RbR ..................................................................................... 31
Figure 10. Area and routing delay analysis for 16 destinations ............................................................ 32
Figure 11. Scalability of LBDR and memory macro-based routing tables when increasing the amount of destinations ........................................................................................................ 32
Figure 12. Coverage results for chips using a 4x4 mesh. sigma=0.05. High spatial correlation: lambda=0.4 ........................................................................................................................................ 35
Figure 13. Coverage results for chips using a 4x4 mesh. sigma=0.05. Low spatial correlation: lambda=1.2 ........................................................................................................................................ 35
Figure 14. Maximum speed achievable by the xpipesLite switch with different routing mechanisms inside. Post-layout results ........................................................................................................ 36
Figure 15. Switch area. Area at 1 GHz is estimated accounting for the switch pipelining overhead (best and typical case area ratios) ....................................................................................... 37
Figure 16. Coverage/area. 4x4 mesh, sigma=5%, lambda=0.4 ................................................................ 37
Figure 17. Switch architecture with source-based routing .................................................................... 38
Figure 18. Switch architecture with LBDR .......................................................................................... 39
Figure 19. LBDR with static priorities ................................................................................................ 40
Figure 20. LBDR with smart priorities. (a) Filter and control signals layout (b) Details of a control signal ........................................................................................................................................ 41
Figure 21. LBDR oriented node labelling in concentrated k-ary n-mesh topologies ......................... 42
Figure 22. Scalability results of a switch with source routing vs one with LBDR for concentrated topologies. X-axys is the switch radix. (a) Timing results (b) area results .......................................................... 43
Figure 23. Architecture of the variation detector .................................................................................. 46
Figure 24. Variation Detector Reset phase .......................................................................................... 47
Figure 25. Timing Margin of the Mesochronous Synchronizer .............................................................. 49
Figure 26. Settling time of the incoming data ...................................................................................... 50
Figure 27. Timing Margin of a Mesochronous Synchronizer with a Variation Detector in front of it.

Figure 28. Timing Margin of the Baseline Mesochronous Synchronizer in the presence of delay variability in the data link wires.

Figure 29. Timing Margin of the Proposed Architecture in the presence of process variation in the data link wires.

Figure 30. Random Variations affecting the detector architecture.

Figure 31. Link route obtained with state-of-the-art P&R.

Figure 32. Link routing flow.

Figure 33. Link wires with the same trajectory and length.

Figure 34. Link bundle routing steps.

Figure 35. Bundled link routed layout example.

Figure 36. (a): Wire length variation (b): Number of vias.

Figure 37. a: resistance variation b: delay variation.

Figure 38. Average and Maximum of different characteristics of all 5 test cases.

Figure 39. Crosstalk delay (ns) comparison in shielding.

Figure 40. Crosstalk delay (ns) comparison in spacing.

Figure 41. Net switching power (μW) comparison for all test cases.

Figure 42. a) CMOS full-swing interconnect. b) Low-swing interconnect. c) PDIFF low-swing receiver from [ZHA00]. d) Optimized PDIFF low-swing receiver.

Figure 43. Sensitivity to systematic variations.

Figure 44. Sensitivity to random variations.

Figure 45. Working samples after compensation of full-swing channels. x-axis indicates the channel circuits to which compensation was applied.

Figure 46. Working samples after compensation of PDIFF low-swing channels. x-axis indicates the channel circuits to which compensation was applied.

Figure 47. Framework for assessing the effectiveness of variability compensation techniques.

Figure 48. 32-bit communication channel layout.

Figure 49. Breakdown of flit_28 capacitance.

Figure 50. Sampling failure due to wire coupling with clock.

Figure 51. Breakdown of the flit_7 capacitance.

Figure 52. Working samples after ASV compensation with random, systematic variations and crosstalk.

Figure 53. Working samples after p-mos n-mos ABB compensation with random, systematic variations and crosstalk.

Figure 54. WID maximum critical path delay distribution for different values of Ncp and D2D critical path delay distribution.

Figure 55. Fastest path evaluator and standard flip-flop.
Figure 56. Schematic of the reduced overhead FPE flip-flop

Figure 57. distribution for a normalized maximum critical path delay
1 INTRODUCTION

While research in the field of NoCs has been very extensive and has addressed many pressing issues pertaining to the interconnect (e.g., performance, area, power, and fault-tolerance), a new serious impediment is currently emerging as a force to be reckoned with, which has yet to be addressed in this context: process variations resulting from manufacturing imperfections. The whole ideology of aggressively scaling die sizes and pursuing heavy on-chip integration is overshadowed by the aggravation of manufacturing uncertainties. PV is observed due to random effects, like Random Dopant Fluctuations (RDF), and systematic spatially-correlated effects like dose, focus and overlay variations. These uncertainties impact various device characteristics, such as effective gate length, oxide thickness and transistor threshold voltages. Altered device characteristics may lead to significant variations in power consumption and to timing violations. The gap between the expected and manufactured characteristics widens further as feature sizes shrink, significantly impacting the chip yield, especially in the regime of stringent timing and power budgets.

Despite the best efforts of process engineers, large variations in device parameters will be unavoidable. In the extreme case, architectures of multi-core integrated systems will have to deal with manufacturing faults in order to sustain yield. In fact, the failure of a single component may result in the failure of the entire system. For this reason, the notion of fault-containment units in a multi-core architecture is gaining momentum, in order to avoid the propagation of faults to the entire system and to preserve yield. Although in principle different from manufacturing defects, process variations affecting the nominal performance of architecture building blocks may lead to the discarding of these latter whenever they cannot keep up with the target operating speed, thus calling for similar countermeasures as for manufacturing faults. However, a wider range of architecture design techniques is feasible in the presence of process variations. They can be essentially categorized into:

- Add on worst-case guard bands to critical paths;
- Use of statistical delay calculation technology for reduced design margins and improved design speed;
- Tolerate infrequent run-time timing violations, where delay failures are tolerated at the cost of performance
- Post-silicon detection and compensation, where the circuit is designed for the typical case and variability compensation is performed post-silicon at some cost (performance, power).

In all cases, there is no doubt on the fact that process variations instil a probabilistic flavour to circuit timing analysis. With this respect, traditional synchronous design methodologies, that require the system to adhere to strict timing constraints over the entire chip area, suffer badly from wide ranging parameter variations. GALS systems may offer significant advantages with this respect, as they essentially divide a complex system into small, loosely coupled (if not independent) modules. In this new context, beyond limiting the performance implications of process variations in local modules and addressing them with traditional design techniques, designers now have to devote special care to the design of process-variation tolerant synchronization interfaces. In fact, these latter become the true weak-point of the system that needs to be safeguarded against timing failures. As an example, it might take too much time to generate the strobe signal out of the source synchronous clock at the receiving end of the GALS link, and current data might be lost and the next one might be erroneously sampled. This scenario is by itself very sensitive to unforeseen link delays arising as an effect of process variations. Still, source synchronous links are generally designed under the assumption that there will no or very limited routing skew between data lines and the source synchronous
Process Variation Tolerance in GALS Systems
2 REFERENCES

2.1 ACRONYMS

<table>
<thead>
<tr>
<th>Acronym</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AES</td>
<td>Advanced Encryption Standard</td>
</tr>
<tr>
<td>CDF</td>
<td>Cumulative Distribution Function</td>
</tr>
<tr>
<td>D2D</td>
<td>Die-to-die</td>
</tr>
<tr>
<td>FPE</td>
<td>Fastest Path Evaluator</td>
</tr>
<tr>
<td>GALS</td>
<td>Globally Asynchronous Locally Synchronous</td>
</tr>
<tr>
<td>LSI</td>
<td>Local Synchronous Island</td>
</tr>
<tr>
<td>PDF</td>
<td>Probability Density Function</td>
</tr>
<tr>
<td>WID</td>
<td>Within-die</td>
</tr>
</tbody>
</table>

2.2 REFERENCE DOCUMENTS

<table>
<thead>
<tr>
<th>Ref.</th>
<th>Document Title</th>
</tr>
</thead>
</table>

[ANG08] I.Loi, F.Angiolini, L.Benini, "Developing Mesochronous Synchronizers to enable 3D NoCs", pp.1414-1419, DATE 2008


[BRA02] D.Braginsky and D.Estrin, "Rumor Routing Algorithm For Sensor Networks", WSNA 2002


Process Variation Tolerance in GALS Systems

---


[LUD09] Daniele Ludovici, Alessandro Strano, Davide Bertozzi, "Architecture Design Principles for the Integration of Synchronization Interfaces into Network-on-Chip Switches", NoCArc, 2009


3  VARIABILITY AWARE GALS LINK DESIGN

Variability sources can be divided into front-end and back-end ones. The front-end phase of the IC fabrication process is related with the steps concerning the creation of devices whereas the back-end stage comprises steps that deal with wiring definition. The main sources of front-end variation are $L_{\text{gate}}$ (transistor effective channel length) and threshold voltage variations. These variations are mainly a consequence of deviations introduced in the photolithographic process and due to random dopant fluctuations, respectively. On the other hand, back-end variation sources are capacitance and resistance variations, that arise as a consequence of variations in wire dimensions, defects introduced mainly by the chemical metal planarization (CMP) process.

Many recent works analyze the impact of process variations in NoCs. Most of them focus on the impact of within-die variation in devices without considering variations in the links [BON08,SAR08]. However, links in the NoC are also affected by variability, although they use semi-global metal layers, where metallizations are much wider than in lower layers, and therefore they should be much less affected by process variation than active devices located on the silicon surface. Unfortunately, the effect of variability on NoC links is not negligible. On one hand, although there are examples of repeaterless NoC links [JOS05], these latter typically undergo repeater insertion, thus they suffer from $L_{\text{gate}}$ variations and dopant fluctuations in the transistors building up a repeater stage. On the other hand, NoC links also suffer from the variability introduced by the CMP process that causes surface imperfections because of dishing and erosion. Actually, this process may be a very important source of variability according to [MON07].

Process variation in NoC links causes that links in the network present different delays, despite that they were initially designed to be identical. Thus, some links will not be able to switch at the intended frequency, reducing performance. This will be the case when NoC operating speed is set by link delay. With respect to this, recent implementation works on NoCs have proved that the critical path of the network is rapidly moving from switches to inter-switch links as technology scales below 65nm [BERMUC]. Moreover, the random component of delay variability impacts the different on-chip interconnects of a switch-to-switch link and their respective repeater stages differently, thus resulting in a misalignment of signal edges of the propagating parallel bit vector. This latter might cause the receiver stage to fail.

As previously stated, the chemical metal planarization process is one of the sources of timing variability in NoC links. The CMP process causes surface imperfections in the wires as a consequence of dishing and erosion. Wide wires, as the ones located in the semi-global layers, are strongly affected by dishing causing considerable changes in the interconnect resistance. Since the degree of dishing and erosion strongly depends on the pattern density of the metallization and NoC links are built in a regular layout, it is possible to assume that all wires in the link and all links in the NoC will be affected by the CMP process in a similar way. The performance constraint for link synthesis determines the sensitivity of the link to delay variability. It is showed in [HER10] that delay variation across a range of technology nodes remains below 0.1% of the mean delay. These results contradict those in [MON07] where authors measure a delay variation, as a consequence of dishing, of around 9% of the mean delay. In this latter case, however, links were synthesized for maximum performance. In the former case, links were synthesized for low power and link delay was dominated by repeater delay and consequently variations in wire resistance have a negligible contribution to the resulting link delay.
The main source of random variation in NoC links is threshold voltage variation due to Gaussian random dopant fluctuations (RDF). RDF will increasingly affect deep submicron technologies scaling from 45nm to 16nm. Table 1 shows the values of $\sigma_{Vth}$ and of $\sigma_{Lgate}$ for the technologies considered as provided by the ITRS report. The $\sigma_{Vth}$ values showed in the table represent the total threshold voltage variation. In [KEN08] it is showed that at least 50% of the variation is due to RDF for a 45 nm technology node. Note that these values represent the $\sigma_{Vth}$ of minimum size devices. Interconnect repeaters usually have larger gain in order to be able to drive high load values as a consequence of link length. It is showed in [HER10] that $\sigma_{Vth}$ can be minimized by increasing the width of repeaters, however this might conflict with low power budgets.

<table>
<thead>
<tr>
<th>Technology (nm)</th>
<th>45 nm</th>
<th>32 nm</th>
<th>22 nm</th>
<th>16 nm</th>
</tr>
</thead>
<tbody>
<tr>
<td>$3\sigma_{Vth}$</td>
<td>40%</td>
<td>58%</td>
<td>81%</td>
<td>112%</td>
</tr>
<tr>
<td>$3\sigma_{Lgate}$</td>
<td>12%</td>
<td>12%</td>
<td>12%</td>
<td>12%</td>
</tr>
</tbody>
</table>

**Table 1: $Vth$ and $Lgate$ variation according to ITRS**

As a consequence of random variations, not all wires in a given link will be able to operate at the same frequency. For example, for a 45nm technology all link wires will work at 2 GHz whereas only 50% of them will operate at 2.15 GHz. In this case there is no significant differences among wires for that link. On the contrary, when using a 16nm technology, all wires in the link will work at only 1.5GHz, while 50% of the wires will be able to operate at a frequency higher than 2.5GHz.

The systematic component of front-end variability is strongly related with the photolithographic process. Lens aberrations may lead to an important systematic spatial non-uniformity of $Lgate$ over the reticle field. According to [ITRS07], $3\sigma$ $Lgate$ variation can be as high as 12% for fabrication processes from 45nm to 16nm. However, it is not enough knowing the maximum percentage of variation in $Lgate$. It is required to know how variations in $Lgate$ impact variations in link delay and also how variations in $Lgate$ are spatially distributed in the exposure field. Thus, a systematic variability characterization is based on the modelling of the spatial distribution of the $Lgate$ variation, and assigning a value of $Lgate$ to each repeater according to both the link configuration and the spatial distribution, and on analyzing how the values of $Lgate$ assigned to repeaters affect link delay.

At a higher level of the design abstraction, the architecture designer has to deal with the random and the systematic component of link delay variability. This is especially true for GALS link design, in that such links are subject to a number of timing constraints for correct operation. With the term “GALS link” we denote a generic link connecting two different clock domains, which can be mesochronous or fully asynchronous with each other. Of course, this report follows all the work already performed in the Workpackage 6 of the Galaxy project, and therefore assumes the global picture and the architecture template of GALS NoCs developed in that workpackage.

An example of the timing analysis intricacy in the GALS NoC domain is showed in [ANG08], where timing constraints for the upstream link of a mesochronous synchronizer are given. They can be used as an input to devise new link and synchronization architecture design techniques, as done in [LUD10]. This latter paper embodies the results of Task 6.2 of the Galaxy project.
As an example, the tightly coupled mesochronous synchronizer proposed in deliverable D6 does not affect the critical path of NoC switches. However, in interconnect-dominated designs, the critical path moves to the switch-to-switch link, and the tightly coupled mesochronous synchronizer implies the delay for a double traversal of the upstream link to be included in the critical path. When this cannot be tolerated, the paper in [LUD10] shows a hybrid solution, which embeds the data link synchronizer in the downstream switch while the control wire synchronizer is left as an external block of the upstream switch. This hybrid architecture enables to reduce the link delay contribution to the critical path to a single link traversal. Similar considerations apply to the dual-clock FIFO architecture presented in deliverable D13.

Timing constraints for the GALS links developed in the Galaxy project have been enforced in met across a number of link inference experiments, thus proving solid timing margins and the efficiency of timing-efficient link design techniques (such as sampling of the input data signal in the synchronizer at the falling edge of the source-synchronous clock). However, in the presence of process variations, both random and systematic, such margins need to be re-analysed and eventually re-enforced. For this purpose, this section of the report illustrates GALS link design techniques at several layers of the design hierarchy aiming at robustness to process variations. During the work hereafter reported, we found that cross-layer design and optimization is really the key approach to effectively tackle process variations, and that techniques developed at a specific abstraction layer do not suffice. In particular, architecture level design techniques are the last-level solution to preserve yield (i.e., guarantee that the entire network can still work in spite of performance sub-optimal switches and links and/or in the presence of manufacturing faults). We find that electrical and circuit level design techniques can easily increase network robustness since they operate at an abstraction layer which is closer to the source of network non-ideality.

As an example, link delay variability might induce a sampling failure in the synchronizer of the downstream switch, which is an hard-to-handle event at the architecture layer. The cross-layer design and optimization policy might consist of adaptively delaying the sampling time of input data and eventually down-clocking the switch, which would end up incurring a longer critical path as effect of the late arrival time of input data.

Our work on process variation tolerance of GALS links has been structured as follows, moving from the top layers of the design hierarchy up to the bottom ones:

- **Development of an yield-oriented evaluation methodology of network-on-chip routing implementations.** In case a switch of the NoC is malfunctioning or cannot be operated at the target frequency of the network because of process variation-induced delay degradation, a design-time regular network may turn out to show a post-silicon irregular connectivity pattern. Topology agnostic routing algorithms have the potential to tolerate process variations without degrading performance. We propose a three step methodology for evaluating routing algorithms in their ability to deal with variability. Using yield enhancement and operation speed preservation as the criteria, we demonstrate how this methodology can be used to select the best design choice among several plausible combinations of routing algorithms and implementations.

- **Efficiency analysis of logic-based distributed routing implementations dealing with irregular topologies.** The support for unexpected and unpredictable topology irregularity should be there in the architecture for NoC building blocks. The routing framework is particularly affected by this new requirement, which poses a new task on burden of the NoC designer. On one hand, among all fault-tolerant routing algorithms that can still provide complete post-silicon connectivity (a vast literature covers this topic in different environments [GPS00,BRA02,MEJ06]), he has to select the most...
performance-efficient one with respect to application traffic patterns. On the other hand, he is faced by the choice of the routing implementation, which has deep implications on the final architecture and on overall performance and complexity figures. In fact, every fault-tolerant routing algorithm lends itself to a table-based implementation. Unfortunately, routing tables are expensive in terms of access time and resources, and feature poor scalability properties. This report makes use of the comparative analysis framework introduced in the item before to assess a couple of table-less routing implementations recently proposed in the open literature, which can be used to minimise the impact of variability on operating frequency and manufacturing yield. The relevant contribution of this report was to provide a physical-synthesis based assessment of the alternative approaches, thus bridging a gap of the results reported in the literature.

- **Development of a GALS link receiver augmenting the baseline synchronization function with robustness to random process variations.** In order to ensure a correct synchronization, source synchronous interfaces route the source clock along with the data at the receiving end and assume an ideal alignment between clock signal and data as well as no routing skew. Unfortunately, this assumption is not verified in presence of process variations. As a result, every routed signal experiences a different delay and the alignment property between clock signal and data wires will be lost at the receiver boundary. In this scenario, the reliability of a source synchronous design becomes unpredictable as it tightly depends on in-situ actual conditions. This document proposes a novel architecture circuit guaranteeing the reliability of the NoC source synchronous interface under high random process variations. The architecture is centered around a variation detector which senses the offset between the source clock signal and the data signals. Therefore, it feeds the downstream synchronizer with a delayed clock signal able to guarantee a safe data sampling. The detector is meant for use during a reset process where the link performs a basic self-testing mechanism based on the principle of self-calibration (i.e., run-time adaptation of the link parameters to in-situ actual operating conditions).

- **Development of a new physical routing approach for robust bundled signalling on NoC links.** Tightly matched signal propagation and strong crosstalk protection are key requirements for next-generation NoC links featuring GALS synchronization and low-swing signaling. In this report, we present a new methodology for NoC global link routing which addresses these challenges. Our approach creates bundled link routes with geometrically matched wires, thus leading to much reduced intra-link variations. Moreover, our link router supports high-regularity wire spacing and shielding strategies. Delay variation among different wires of a link is 25% to 70% lower than what can be achieved using a state-of-the-art timing-driven global routing flow. Additionally, crosstalk effects are reduced by more than 30%. While the approach of the item above deals with random process variations, this physical routing solution effectively deals with the systematic component of process variations, in that it forces all wires of a GALS link to undergo the same routing and layout conditions.

- **Variability compensation for full-swing vs low-swing on-chip communication.** Adaptive body bias (ABB) and adaptive supply voltage (ASV) are effective methods for post-silicon tuning to reduce variability on generic combinational circuits or microprocessor circuit sub-blocks. We focus in this report on global point-to-point interconnects, which are evolving into complex communication channels with drivers and receivers, in an attempt to mitigate the effects of reverse scaling and reduce power. The work in this item is at lower level of the design hierarchy, and deals with
the reliable design of the electrical link building up a GALS link, although it is not restricted to GALS at all. The characterization of the performance spread of these links and the exploration of effective and power-aware compensation techniques for them is becoming a key design issue. This report compares the effectiveness of ABB vs ASV when put at work on two on-chip point-to-point link architectures: a traditional full-swing and a low-swing signaling scheme for low-power communication. This work provides guidelines for the post-silicon variability compensation of these communication channels, while considering realistic layout effects. In particular, the implications of cross-coupling capacitance on the effectiveness of variability compensation are analysed.

All the sub-tasks illustrated in this report include an extensive validation framework giving practical relevance to the theoretical concepts.

### 3.1 TARGET GALS PLATFORM

In this sub-section, let us recall the target GALS NoC platform developed and refined in the context of Workpackage 6 of the Galaxy project. It consists of implementing the on-chip network as an independent clock domain, and therefore to place circuitry to reliably and efficiently move data across asynchronous clock boundaries between NoC switches and connected network interfaces. These latter are assumed to be part of the clock domain of the IP core that they serve. Dual-clock FIFOs are an effective solution to provide asynchronous boundary communication, especially in throughput-critical interfaces. However, many designers are skeptical about their utilization due to the relevant latency, area and power overhead they incur. Beyond urging research activities aiming at the optimization of dual-clock FIFO architectures, this fact emphasizes the need for their conscious use in GALS systems. Figure 1 depicts our GALS target platform.

![Figure 1. Target GALS Platform](image)

Aware of this, we try to minimize their usage as much as possible by instantiating them only at IP core boundaries, after their respective network interfaces. However, this choice moves many chip-wide timing concerns to the on-chip network. In fact, this latter ends up spanning the entire chip and might be difficult to clock due to the growing chip sizes, clock rates, wire delays and parameter variations. There is little doubt on the fact that a high-performance and cost-effective Network-on-Chip can only be designed in 45nm and beyond under a relaxed synchronization assumption.
We find that mesochronous synchronization can relieve the burden of chip-wide clock tree distribution while requiring simpler and more compact synchronization interfaces than dual-clock FIFOs. Hierarchical clock tree synthesis is an effective way of exploiting mesochronous links, as already experimented in [PAN08]. During the first step, a clock tree is synthesized for each network switch with a tightly controlled skew (e.g., 5\%). Consequently, each clock tree is characterized with its input delay, skew and input capacitance. This information is used by the clock tree synthesis (CTS) tool to infer a top clock tree balancing the leaves with a much looser skew constraint (e.g., 30-40\%). The ultimate result is a global clock tree which consumes less power then the traditional one generated by enforcing chip-wide skew constraints.

For future large multiprocessor systems-on-chip, the use of this methodology can be not just an issue of power efficiency but even of CTS feasibility. Of note, power savings with this methodology should not be taken for granted, since it involves some overheads: the transmission of the clock signal across mesochronous links, the mesochronous synchronizers themselves (implementing power-hungry buffering resources) and the increased number of buffer slots needed at link end-nodes to cover the larger round-trip time (associated with the synchronization latency) for correct flow control management.

Our design platform aims at minimizing such overheads through a novel mesochronous architecture taking advantage of the tight integration of the synchronizer into the NoC architecture. However, since these solutions give rise to timing constraints that might not be verified for specific layout conditions, we provide architecture variants for these cases as well, thus coming up with a flexible NoC suitable for many design instances. Further details can be found in [LUD10], condensing results of Galaxy Workpackage 6 and of its deliverables.

### 3.2 YIELD-ORIENTED EVALUATION METHODOLOGY OF NETWORK-ON-CHIP ROUTING IMPLEMENTATIONS

Not only fault-tolerance is becoming a critical requirement in designing modern chips, but the on-chip networking scenario is raising new challenges for fault-tolerance. Because of variability-induced performance asymmetry, the post-silicon NoC topology at the target nominal frequency may differ from the projected one at design time, and in particular it might turn out to be highly irregular in spite of the inherent regularity of the original design. All modules and links could be then slowed down to the frequency of the slowest element, but this is not acceptable for all application scenarios. This example points out the ultimate challenge: the support for unexpected and unpredictable topology irregularity should be there in the architecture for NoC building blocks. The routing framework is particularly affected by this new requirement, which poses a new task on burden of the NoC designer.

On one hand, among all fault-tolerant routing algorithms that can still provide complete post-silicon connectivity, he has to select the most performance-efficient one with respect to application traffic patterns. On the other hand, he is faced by the choice of the routing implementation, which has deep implications on the final architecture and on overall performance and complexity figures. In fact, every fault-tolerant routing algorithm lends itself to a table-based implementation. Unfortunately, routing tables are expensive in terms of access time and resources, and feature poor scalability properties. This report shifts its interest on logic based distributed routing as an alternative routing implementation mechanism for NoCs. The basic idea is to employ few configuration bits of the routing mechanism at each NoC switch, and to determine the next hop to take by means of simple combinational logic.
Unfortunately, different kinds of logic routing implementations feature a different coverage of topology irregularities. As an example, the LBDR mechanism illustrated in [FLI08] can implement many distributed routing algorithms even on irregular topologies, provided the communication between two end nodes in the post-silicon topology can still go through a minimal path of the nominal regular topology. When the irregularity pattern is such to violate this constraint, more complex logic and more configuration bits are required in the routing mechanism to be able to still route traffic successfully. In this case, the designer needs to know whether the increase in complexity of the routing implementation is adequate or disproportionate with respect to the achieved coverage of topology irregularities, similar to the KILL rule for multi-core design [AGA07].

To the best of our knowledge, there is no generally accepted methodology for quantitatively evaluating the trade-off between complexity (area, power, impact on critical path) of the routing implementation and coverage of irregularity patterns. This report moves a first step in this direction and proposes a yield-oriented methodology for the evaluation of NoC routing implementations other than look-up tables. The proposed framework also considers the common practice of decreasing the post-silicon operating frequency of the network to make variability-affected switches and links again operational. Any combination of variability model, NoC architecture, routing algorithm and routing implementation can then be evaluated with respect to yield enhancement and fabrication cost.

The methodology allows the joint evaluation of a routing algorithm together with its implementation mechanism (hereafter denoted as the routing framework) assuming the existence of a variability pattern in the chip manufacturing process. The methodology helps in assessing the effectiveness of the routing framework in sustaining yield when facing within-die process variations, while relating its benefits to its cost.

In this direction, the methodology estimates the percentage of cases alternative routing frameworks succeed in tolerating variability patterns and makes a relative comparison between such percentages and the resources used by each solution. Ultimately, the designer has a clear view of the complexity-coverage trade-off that the routing frameworks under test span. The methodology has gained scientific relevance with the publication in [SRO09].

### 3.2.1 Evaluation methodology

Figure 2 shows the three steps of the methodology. Each step performs a different task independently from the others. This property makes the methodology easily extensible to several types of variability models, NoC architectures and routing frameworks.

The design entry is the top-level layout of the system, comprising networked IP cores (functional units, memories,..) and the interconnect topology. The first step of the methodology is to build a statistically significant database of chip instances resulting from variability patterns, in a similar way to what is done in a Montecarlo analysis.

To do this, a process variation model needs to be fed to the methodology, which is then used to statistically inject deviations to nominal delay values of links and switches. The methodology can support first-order models as well as more accurate ones. For instance, abstract models projecting switch delay variations based on the number of critical paths and on the logic depth can be used. Alternatively, the true switch gate-level netlist can be used for the injection of Gaussian noise to the gate delays, and static timing analysis tools can then be used to measure the critical path variation (similar to [BON08]). As regards links, variability models can capture interconnect-related effects such as dishing, combined with the delay variability of drivers and repeaters. Our methodology does not pose any limitation to the kind of variability
model used, as long as it can result in delay variations of NoC switches and switch-to-switch links with respect to nominal values.

![Diagram](image)

**Figure 2. Mechanisms used in the yield-oriented evaluation methodology.**

Each chip instance features a topology with a given irregularity pattern. However, such pattern changes as a function of the target operating speed. As this latter is slowed down, progressively more switches and links become operational again and the critical path may move from links to switches or vice-versa. The lowest frequency for the analysis is the one that makes all switches and links fully operational. As the target frequency is raised, the networks becomes increasingly irregular until some designer defined boundary is achieved (e.g., a maximum number of disconnected nodes, full connectivity retained between predefined nodes). Each topology with its associated speed is denoted as a topology instance.

The second step deals with the definition of a suitable routing algorithm for every chip/topology instance. In this step care must be taken to guarantee the necessary conditions of a deadlock-free network and of a connected chip. Also, the resources devoted (at design time) to routing purposes (e.g. virtual channels) must be considered at this step. The methodology has been designed to accept any routing algorithm at this step, so the designer is free to test any routing algorithm from the wide myriad of options available. However, the designer must be aware of the irregularities the chip is facing (due to the variability). So, at this step, it seems more interesting to test either topology agnostic routing algorithms (able to work on any topology) or fault-tolerant routing algorithms. In both cases the implementation of the routing engine will need to be assessed, and this will be performed at the next step in the methodology.

In the third step, a range of alternative routing implementations is tested for every successful routing algorithm achieved in the previous step. Notice that fault-tolerant routing algorithms must be evaluated together with their particular implementations. Also, topology-agnostic routing algorithms can be evaluated with a wider set of implementations ranging from look-up tables to minimum specific logic-based implementations. In other words, not all routing algorithms can be implemented by a given mechanism, and therefore, not all possible combinations are feasible.

There are several ways of defining the test success at this stage. For instance, communication flows as extrapolated from an annotated task graph can be applied to the topology instance and the feasibility of those flows can be assessed. A large number of evaluation tools can be used at this stage, including custom home-made connectivity verification tools all the way to functional simulation tools.

The methodology will list the set of topology instances that succeeded with a particular routing algorithm and routing implementation. Therefore, percentages of successful chips will be exposed to the designer in order to assess the tolerance of a particular routing framework to the effects of variability.
We define a “coverage” metric as the percentage of chips that are usable for a given frequency and routing framework. However, the implementation cost of the routing mechanism that achieves a good coverage result might be disproportionate with respect to the coverage itself, or in contrast might be well justified. With the “coverage/area” metric we intend to capture whether the benefits are worth the increase in complexity, which can be understood by comparing this metric for competing solutions.

3.3 ASSESSMENT OF LOGIC-BASED ROUTING IMPLEMENTATIONS WITH LAYOUT AWARENESS

The comparative analysis framework illustrated above has been used in order to discriminate between alternative logic-based routing implementations. We believe such routing implementation is promising for future on-chip networks (and therefore also for our GALS NoC architecture template) in need of network-level fault tolerance mechanisms because of their better scalability with respect to table-based implementations (which will be hereafter proven). Beyond expressing the coverage/area metric, this report adds both architecture-level design considerations (illustrating how to integrate such routing mechanisms in the existing xpipesLite NoC architecture) and physical design considerations (illustrating the area and critical path of the switch after the distributed routing mechanism has been implemented). We would like to point out that the routing mechanisms themselves are NOT the contribution of this report and of the Galaxy WP6 activities, in that existing ones were assessed to be suitable for NoC needs. The contribution is rather in their layout-aware assessment, which was achieved by defining the above evaluation methodology and by implementing them in the xpipesLite NoC switch for physical synthesis. This contribution was completely missing in the results reported in the open literature, and therefore significantly advances state-of-the-art of GALS NoC design practice.

The complexity of routing mechanisms and the set of routing algorithms supported by each of them strongly depend on the topology they have to cope with. The most straightforward topology for NoCs is the 2D mesh structure as it offers regularity and simplicity for routing. All the links have the same length thus, exhibiting the same latency. Also, local traffic is well supported since latency is low. One of the inconveniences of the 2D mesh, however, is the relative higher hop count for messages travelling distant nodes. Fortunately, this impact is minimized with the use of wormhole and virtual cut-through switching (where hop count is additive to latency).

On the other hand, in CMP systems and high-end MPSoCs, tile-based design is gaining momentum. The tile is designed in isolation and, once finished, the chip is built by replicating tiles. By doing this, the design effort to build a chip is drastically reduced. Tiled designs also advocate for regular network structures like 2D meshes.

Due to the previous reasons, we advocate for 2D meshes in CMPs and high-end MPSoCs. However, even if the design of a CMP chip with a 2D mesh network is correct, the final on-chip network may face new raising challenges, leading to a non-regular network structure. Several challenges are being identified in the following years to come: manufacturing defects, effective chip utilization, voltage/frequency islands, and effective power saving techniques.
The challenging issue of defective components has already been discussed early in this section. We just recall here that the allocation of a defective tile, if not addressed, will ruin the 2D mesh structure of the network, leading to an irregular network not handled by the routing algorithm and thus making the chip unusable.

As an example of this problem, Figure 3a shows the network for a 16-core chip using a 4x4 mesh topology to interconnect the cores. In this system links have been initially designed to work at 1GHz. However, because of a 5% of variability (for current technologies), we will end up having links working at different frequencies. Figure 3a shows the frequency of each link after applying the variability model in [HER10].

In order to keep the chip working and not decreasing yield, the simplest thing we can do is to lower the operating frequency of the entire network to the lowest link frequency. In this case, our example chip would work at 872 MHz. This, in fact, could be the only feasible option if, for instance, we implement the DOR routing in the network. DOR is designed for 2D meshes and cannot work on a network with one or more missing links. However, as variability increases, reducing chip frequency to the lowest link frequency in the NoC may be unacceptable.

Notice that in the example chip, frequency is reduced down to 872 MHz just because one of the links is not able to switch at a faster rate. Actually, just by not considering two links in the network we could increase chip frequency up to 932 MHz.

Figure 3b shows the resulting topology for that frequency, where all the cores remain connected but two links are disabled (those not reaching that frequency). However, this would...
turn an initial regular topology into an irregular one and, therefore, the DOR routing cannot be used because there is a pair of unconnected nodes. If the routing layer does not take this into account, the chip would have to be discarded, reducing yield and increasing manufacturing costs (or would have to be operated at a lower frequency, if the design allows it, decreasing thus performance and benefit).

Another challenge is to get enough parallelism to efficiently use tens and hundreds of cores. This will result into a low utilization of the cores, or alternatively, into the need to partition the chip into multiple domains, each one running a different application. In this scenario, and to fit as many applications as possible, the partitions of the chip resources will become irregular. The way applications are mapped onto the chip belongs to the concept known as virtualization, where a real chip is divided into several smaller virtual chips.

Voltage/frequency islands are being identified as a need, where different regions of the chip will have different operating conditions. The GALS NoC design techniques developed in the Galaxy WP6 go exactly in the direction of isolating such regions.

Finally, another major challenge is the need of efficient power saving methods. As the number of cores increases, probably most of the cores will remain unpowered (in sleep mode) most of the time, thus enabling large savings in power consumption. The same should be applied to the on-chip network, which has been reported to consume 30% of the total chip power consumption. Powering off and on different routers will lead to irregular patterns over time.

All these previous challenges require some effort at the on-chip network level, specifically supporting irregular network topologies. But rather than addressing completely irregular topologies (more suitable for low-end MPSoC systems) the effort will be made hereafter to address irregular topologies derived from an original 2D mesh with the following properties: (1) a router is connected to at most four routers each one in a different direction and dimension, and (2) a hop along a valid direction and dimension will not cross more than one row and column.

In this report we comparatively analyse mechanisms able to cover an increasing amount of failure patterns derived from a 2D mesh, that is, with increasing support for any failure/virtualization/domain/region configuration.

### 3.3.1 First routing mechanisms: LBDR

The basic logic-based distributed routing mechanism denoted as LBDR was firstly proposed in [FLI08] and then extended in [ROD09] by Universidad Politecnica de Valencia. LBDR requires eight routing bits (R_ne, R_nw, R_en, R_es, R_wn, R_ws, R_se, R_sw) and four connectivity bits (C_n, C_e, C_w, C_s) per router to define the routing and connectivity pattern. Those bits are used by a combinational logic to decide the output port to use for every message.

The LBDR mechanism relies on the use of minimal paths for every source-destination pair. Indeed, LBDR uses two comparators to decide the directions to use based on the relative position of the current router and the destination router. The directions (labelled as N’, E’, W’, and S’ in Figure 4) are then checked against the routing/connectivity bits. Therefore, only minimal paths are allowed. This leads, as we will see in the evaluation, to a low percentage of topologies being supported. Indeed, in a 2D mesh it is easy to imagine sets of failed links/routers that require non-minimal paths for some source-destination pairs. A single hole in the center of the network is a clear example. Figure 5 shows a topology not supported by LBDR. The path from A to B is non-minimal. At router A, the possible directions to reach B are N and E, however, both links are missing, and therefore no possible way out.
3.3.2 Second routing mechanisms: LBDRdr

We need to provide non-minimal support in an efficient way, that is, with as few logic as possible. Figure 6 shows the additional logic proposed. In particular, for every input port of the router a deroute option is provided. A set of two bits encode the deroute option that can be N, E, W, or S. Whenever the previous LBDR logic is unable to provide a valid output port (NOR gate with four inputs) the deroute option is taken into account. The logic is replicated for every input port, therefore the deroute option used is the one associated with the input port the message arrived from.
Alternatively, the deroute option can be designed for the entire router, instead of having a deroute option per input port. However, this reduces flexibility and leads to non-supported topologies (this alternative will be analyzed later). Figure 7a shows an example, where two different deroute options are required for two different input ports at router A. If going N, and the message comes from input port S, then a deroute is set to W. On the other hand, if the message is coming from W, and the intention is to go E, then a deroute is set to S. It is worth mentioning that the deroute option needs to be computed in accordance to the routing algorithm, as it must not introduce cycles that could lead to deadlocks. In Figure 7a a deroute option at input port W at router B can not be set to S as it would let messages crossing a routing restriction.

The algorithm computes first the set of routing bits. This is done by taking into account the topology (including the failed/powered down routers and links) and the routing algorithm. The selected algorithm in this report is segment based routing SR [MEJ06] as it can be applied to any topology and does not require virtual channels. Alternatively, for a healthy chip the XY routing algorithm can be used. LBDR bits are computed by taking into account the location of the routing restrictions. Thus, computation is straightforward and the complexity is low (linear with the number of routing restrictions).

Once LBDR bits are computed the deroute options are searched. To do this, the algorithm looks for a valid path for every source-destination pair (the algorithm is computed offline before
any normal operation of the chip, thus computation complexity is not a major issue). As LBDR may allow multiple paths for a given source-destination, the algorithm deeply searches all the paths in a recursive way. Two end nodes are connected by LBDR if all the possible paths reach the destination. In the search of all paths, whenever it fails to provide a valid path, then, a misrouting action is needed. Figure 7b shows the case at router B for messages going from router A to router C. In this situation, the algorithm tries all the possible deroute options available, one per output port but avoiding U turns (so, west port is not considered). Options leading to crossing routing restrictions are also evicted. The algorithm starts with the first deroute option and keeps following the path, thus taking the deroute, checking if the path (and all their possible alternative paths) will reach the destination. In case of failure, the deroute option is set. In case of failure (destination is not reached) another deroute option is tried. In case all deroute options fail the topology is not covered by LBDR_dr. Notice that several deroute options may be required for a single path. Figure 7b shows the route along the path using the deroute bits at router B.

This mechanism will enhance greatly the coverage of LBDR. However, there are subtle cases that are still not covered by LBDR_dr. Figure 7c shows an example. The problem comes by the fact that for some destinations located at the same quadrant, at router B the routing engine should provide one port (N) for some destinations (router C) and another port (W) for other destinations (router A). As LBDR (or LBDR_dr) works in quadrants, there is no way to indicate the router which option should be given to the message. Addressing this issue with LBDR-style of routing implies implementing virtual cut-through instead of wormhole as the switching mechanism, and will be therefore ignored in this report.

![Figure 8. Example region definition](image)

3.3.3 Third routing mechanisms: Region based routing

The Region-based Routing (RbR) framework [FLI07] allows to route messages by using simple and efficient blocks of logic referred to as regions. Each region is a square box of destinations that is identified by the possible output ports, the top left most switch and the bottom right most switch of the region. A different set of regions is implemented in logic at
every input port of a given switch. When a message header arrives to the input port, it inspects all regions in parallel in order to find out which are the possible output ports it can take. The computed regions are used to program hardware registers located at the input port of every switch. These registers must be programmed before routing any packet at network boot time. A nice feature of RbR is that it allows the implementation of topology-agnostic routing algorithms providing appropriate support for routing under the presence of link and node failures.

An example of region definition is illustrated in Figure 8. some switches have been grouped into regions (r1, r2, r3). Regions are defined for the switch highlighted with a circle (this switch will be referred to as the reference switch). All the packets arriving to the reference switch (or being generated at its local port) addressed to any destination included in region r1 necessarily need to use the W output port at the reference switch. Notice that if they use the N port, they will need to cross a routing restriction which is forbidden or use a not minimal route which is inefficient. The same happens when using the E and S ports. Therefore, at the reference switch, region r1 includes destinations that can be reached only through the W port.

Also, there are other destinations that can be reached using only the W port (i.e. switches within region r3), however the mechanism relies on defining rectangular regions for the sake of implementation, thus, two regions (r1 and r3) are defined for the same output port.

An interesting property of the mechanism is that regions can be defined for more than one output port, thus allowing the adaptiveness inherent to the applied routing algorithm. For instance, region r2 is defined by those switches that can be reached either using the N or W ports at the reference switch.

The input port used by the packet to arrive to the reference switch must be also taken into account when defining regions. For instance, at first sight we can use ports W and S to reach region r4. However, W output port should be only used when the packet does not reach the reference switch through the N port, as it would cross a routing restriction at the reference switch and potentially would lead to deadlock. Therefore, regions must be defined based on the set of destinations, the output ports and the input ports. In the figure, region r4 is defined for packets using the input port E and I and region r5 is defined for packets using the input port N. Notice that the set of destinations of two regions can be overlapped, but they will differ in the set of input ports.

To summarize, a set of regions is computed at every switch. Each region is defined by the possible input ports used by the packets, a subset of the output ports that may be used and the potential destinations that can be reached. Additionally, as regions are defined by rectangular boxes of switches in a 2D mesh network, we can notate a region for switch x as Rx(iport list, {swref1, swref2}, oport list)

(swref1 is the top left most switch and swref2 is the bottom right most switch).

Therefore, whenever a packet arrives into a switch, all the regions must be inspected in order to detect which are the regions suitable for routing the packet. The control routing unit will, thus, select the set of output ports provided by the matched regions and deliver all the possible output ports to the arbiter in charge of selecting the most convenient one.

Let us now detail the hardware implementation of Region based Routing (see Figure 9). The input port for wormhole switches generally requires two different blocks, an input port controller (IPC) that manages the input buffer and transmits the status information to neighboring nodes, and a header decoder (HD) that decodes the header information of every packet. Among other information it contains the destination identifier of the packet. In RbR, the packet ID identifies the X and Y coordinates of the destination (the MSBs indicate the row RowDst and the LSBs indicate the column ColDst). Once decoded, the coordinates of the destination are compared against the coordinates of the current switch. If equal, the packet is
delivered to the internal port, otherwise, the coordinates of the destination are sent to the routing control unit.

Figure 9. Hardware implementation of RbR

The routing control (RC) unit is made of different logic modules, one for each possible region defined in the switch for the region-based routing. Each module has six registers, the input port IP register with one bit per input port (NEWSI; North, East, West, South, and Internal), the ROW1, COL1, ROW2, COL2 registers with $\log(N)$ bits each (containing the coordinates of the top leftmost switch and the bottom rightmost switch of the region), and the output port OP registers with one bit per output port (NEWS). These registers must be programmed before routing any packet at network boot time.

Once the packet header has been decoded it is passed to the RC unit together with the input port identifier where the packet arrived to the switch. At the RC unit this information is compared against the pre-defined regions. In order to save power, a pre-selection of regions is made according to the IP registers. That is, those regions whose input port registers do not match with the input port of the packet being routed are discarded. For this purpose, the region trigger unit RT matches the input port of the packet with the input port register of every region. The RT unit consists of five AND gates (one per input port) and a single OR gate. The output of the RT unit is a signal that triggers the rest of the logic for the region.

After checking the input port, and only on success, the rest of the logic for a region is activated. In particular, the logic determines if the destination is within the boundaries defined by the region. To achieve this, the row and column of the packet’s destination are compared with the contents of ROW1, COL1, ROW2 and COL2 registers. For this, four magnitude comparators are used at the region matcher unit RM. If all comparisons are true, the destination of the packet is within the region and therefore, the output ports defined for the region must be considered for routing purposes. Thus, the logic selects the output ports previously introduced in the OP register. Notice that the implementation allows different output ports to be selected from the same region. Once all regions have been evaluated, all the selected output ports from all the regions that matched the packet are ORed and passed to the arbiter. The arbiter may choose one output port based on different criteria. Then the packet will
be routed to the next switch until arriving to destination. As showed, the control unit of the region-based routing mechanism can be implemented using a very simple and fast combinational logic circuit as it is showed within the dotted square in Figure 9. Note that RT and RM units have been serialized. Although such a decision will increase latency, it will reduce energy consumption. Thus, there is a trade-off between delay and energy that must be evaluated when designing the NoC.

![Figure 10. Area and routing delay analysis for 16 destinations.](image)

![Figure 11. Scalability of LBDR and memory macro-based routing tables when increasing the amount of destinations.](image)

### 3.3.4 Table-based vs Table-less routing: the physical perspective

First of all, the objective of our work was to compare a table-based routing implementation with a table-less one, in order to confirm the goodness of our focus on this latter for nanoscale NoCs. In this direction, both LBDR and RbR were implemented inside the xpipesLite NoC switch. For the analysis that follows, only LBDR will be considered.

LBDR removes the need for routing tables at switches. In most designs of practical interest, forwarding tables are usually implemented by means of memory macros. This motivates a
comparison between LBDR logic implementation and memory macro-based routing tables in terms of area and routing delay.

For our experiments we used Memaker, a memory compiler from Faraday Technology, generating memory macros for a 90 nm UMC process technology. LBDR logic was synthesized for the new technology library (so to result in technology-homogeneous comparisons) and its delay contrasted with the access delay of a corresponding memory macro meeting the same routing requirements.

For the evaluation, this scenario was assumed: switches are placed in a 2D mesh topology, with one computation tile attached to each switch. Each tile consists of a processor core and a memory core, each one requiring a master and a slave xpipesLite network interface for accessing the network. Thus, switches in the worst case need 8 I/O ports. The total number of destinations is 16 end-nodes in the system at hand (a 4x4 2D mesh).

When routing tables are used for distributed routing, each switch input port has a memory module with a number of words equal to the amount of destinations. Every word is composed of 3 bits, matching the switch radix. Given a destination ID, the switch thus selects the target output port (from 0 to 7) based on table look-up.

The minimum size in words that Memaker, at the 90nm technology node, can generate is 256 words. For fewer words, Memaker can however infer a single-port register file. We also included this option in the comparison of Figure 10, targeting our system under test with 16 destinations. The figure shows that LBDR logic consumes significantly less area while matching the delay of the register-based routing table solution. The oversized memory macro is clearly non-competitive at this system scale.

Figure 11 illustrates area and delay scalability of LBDR and memory macro-based routing tables with an increasing number of network destinations. Clearly, while the memory macro suffers from increasing area and delay penalties, LBDR logic complexity does not depend on the number of destinations, hence stays constant. LBDR logic just grows with switch radix.

When the number of destinations is between 16 and 256. Figure 10 and Figure 11 suggest that LBDR will be by far the most area effective solution, while at least small routing delay improvements with respect to register-based routing tables can be conservatively expected.

### 3.3.5 Evaluation of routing implementations

The above routing routing implementations were assessed with the yield-oriented evaluation methodology. In order to probe the usability of the method with a real case study, we selected a variability model, a set of routing algorithms (derived from a routing methodology) and three different NoC routing implementations. Our aim is to compare the routing implementations under their achieved coverage (percentage of supported topologies/chips) once the variability model is applied. At each stage of the methodology we will customize the definition of successful tests for our needs.

Without lack of generality, we selected the variability model presented [HER10] as the starting point for generating the pool of chips and topologies. In particular, this model takes into consideration the effect of variability on switch-to-switch link delay. Although we are assuming here that link delays determine network operating speed (and therefore that switch critical path can be tuned with compensation techniques or is simply shorter than link delay), this is an interesting case study since recent implementation works on NoCs have proved that the
critical path of the network is rapidly moving from logic to inter-switch links as technology scales below 65nm [BER09,LUD09].

<table>
<thead>
<tr>
<th>Variability model</th>
<th>Number of topologies</th>
<th>Chip layouts</th>
<th>Routing instances</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\sigma 0.05 \lambda 0.4$</td>
<td>50</td>
<td>4x4 mesh</td>
<td>16</td>
</tr>
<tr>
<td>$\sigma 0.05 \lambda 1.2$</td>
<td>50</td>
<td>4x4 mesh</td>
<td>16</td>
</tr>
<tr>
<td>$\sigma 0.18 \lambda 0.4$</td>
<td>50</td>
<td>8x8 mesh</td>
<td>8</td>
</tr>
<tr>
<td>$\sigma 0.18 \lambda 1.2$</td>
<td>50</td>
<td>8x8 mesh</td>
<td>8</td>
</tr>
</tbody>
</table>

**Table 2 Parameters used in the evaluation**

Table 2 shows the $\sigma$ (link delay variability) and $\lambda$ (variability spatial correlation) parameters used by the variability model (as predicted by the ITRS roadmap).

After applying the variability model with the different parameters we obtained 50 different chip instances per configuration. In each case all the links are labelled with their post-silicon delay. The methodology obtains different topology instances from each chip (each instance corresponding to a different network speed) by setting the link delay threshold of the chip and enabling only those links below the threshold. The first topology for a chip is set by the maximum frequency the network can tolerate (the connectivity among all nodes is ensured but the topology is probably highly irregular) and the last topology is set by the minimum frequency the network can tolerate (maximum connectivity is achieved). In each case all the links will work at the same frequency and a varying number of topologies will be achieved from each chip.

We feed the second step with a topology-agnostic routing algorithm that can be used for irregular topologies, the Segment-based Routing (SR). SR allows multiple instances of the routing algorithm for the same topology. Thus, for each topology obtained in the previous step, we compute a set of 16 SR instances for the 4x4 layouts and a set of 8 SR instances for the 8x8 layouts (note that 8x8 topologies allow for 64 instances by starting the computation of the routing algorithm from a different node, but for the sake of reducing analyzed data, we only compute 8 of them by starting in the diagonal nodes). The set of routing instances will be used in the next step.

Finally for the third step we selected four different routing implementations: Logic-based Distributed Routing (LBD), LBD with de-routes (LBDRdr) (i.e., LBD with a de-route output port set on every switch where LBD fails) and Region-based Routing (RbR) (two different versions of RbR will be used, RbR_8r with eight regions per input port and RbR_12r with 12 regions per input port). These mechanisms have been selected since they represent the most efficient implementations of routing mechanisms for on-chip networks proposed so far **without the need for routing tables and virtual channels**. As pointed out later, these mechanisms feature different levels of complexity.

For every computed routing algorithm, we will test if every routing implementation works or not. It works if it is able to route packets from every source to every destination. If a routing implementation works for at least one of the routing algorithms for a topology with a given
operating speed, then, it is said that the routing implementation covers the current chip at that frequency. We will rank all the chips with the maximum supported link frequency for every routing implementation.

### Figure 12. Coverage results for chips using a 4x4 mesh. sigma=0.05. High spatial correlation: lambda=0.4

![Figure 12](image)

### Figure 13. Coverage results for chips using a 4x4 mesh. sigma=0.05. Low spatial correlation: lambda=1.2

![Figure 13](image)

### 3.3.5.1 Coverage results

In this first evaluation we analyze the coverage that can be achieved for each routing implementation. To do so, we classify the chips according to the frequency they can work at. Figure 12 and Figure 13 show the distribution along the link frequency of all the 4x4 chips (sigma=0.05) analyzed for two values of spatial correlation, lambda=0.4 and lambda=1.2, respectively. The figure includes results for LBDR, LBDR_dr, RbR_8r, RbR_12r, and MAX (routing tables). The MAX curve represents the percentage of chip instances that have a connected network (with respect to the total amount of instances). The same baseline frequency $f_0$ is assumed for all experiments, selected in the high-performance range of the network (e.g., 1GHz) where the reliability-performance trade-off appears more clearly.

As a first observation (see MAX curve) we can see how the spatial correlation of the variability affects the connectivity of the network. With a low spatial correlation (lambda=1.2) more chips can be operated at a higher frequency (the topology gets connected at higher frequencies). However, the most interesting results are related to LBDR and RbR coverage. We can see
that the best option for coverage purposes is RbR with 8 regions. It achieves 100% coverage for all connected chips (RbR_8r equals MAX) in all the frequency ranges. Also, LBDR achieves a decent although minor result for the coverage. It achieves, in most cases, 50% of coverage for the connected chips for lambda=0.4, however it achieves lower coverage for lambda=1.2. Differences are larger for 8x8 chips (not shown). It can be seen that LBDR_dr, however, increases LBDR coverage by a larger extend in the lambda=1.2 case, and to fully cover all the connected chips in the lambda=0.4 case. These results are achieved also for the 8x8 chip case (not shown).

The reason for this is that for low correlation values, the frequencies of the links of a given router spread out to a larger frequency range, and therefore, when a frequency threshold is applied, that router has more chances to get any of its links above the threshold. On the contrary, for high correlations, all the links of the router will work at similar frequencies, and thus, for a given frequency threshold, the probability of all of them having a lower (or higher) frequency than the threshold is higher. If they are below the threshold, the router will remain unconnected.

3.3.5.2 Performance and Cost Results

In the previous section we have analyzed the coverage each routing implementation provides for different values of sigma and lambda. The analysis has been performed for the same design frequency f_0. In this section we analyze the maximum operating frequency reachable by each switch with the different routing implementations and the area overhead for their logic and registers. The intention behind this is to characterize the reliability-performance/area trade-off, and to ultimately assess whether the implementation complexity of a routing mechanism largely exceeds its coverage improvements or whether these latter are worth the cost.

The routing mechanisms were implemented inside the xpipesLite switch and synthesized with an STMicroelectronics 65nm low-power technology library used with commercial backend synthesis tools.

Figure 14 shows the maximum achievable switch frequency when using each possible routing mechanism. Results are normalized to the fastest solution (LBDR). Logic depth for each scheme is important, since it resides on the switch critical path. Clearly, LBDR_dr incurs only a minor degradation of the maximum speed (which is slightly more than 1 GHz for baseline
LBDR). In contrast, RbR suffers from about 30\% frequency reduction, with a further degradation when moving from RbR_8 to RbR_12. The main cause for this large critical path lies in the larger number of cascaded logic stages (local-port matching, region matching, output port selection) and in the use of high-fanout nets at switch inputs. The low performance of RbR can be hidden by pipelining switch operation. To reach 1GHz switch frequency the single cycle switch architecture can be retained only for LBDR solutions. RbR requires 2 cycle switches to keep up with that frequency.

As regards area, we can see in Figure 15 the results. The left-hand plot shows post-layout area at maximum performance, while on the right-hand side an area estimation is reported for the case where all switches have to be operated at 1 GHz. Therefore, we end up having two mechanisms with opposite benefits, LBDR having low coverage but small area requirements and RbR having excellent coverage but large area requirements. In order to quantify this trade-off, we define the coverage/area metric, which expresses how effectively network area is used with respect to the reliability objective.

Coverage/area is measured by dividing the coverage by the chip area devoted to the routing mechanism (area of the mechanism multiplied by the number of routers). Results are shown in Figure 16. When considering real implementation costs we get opposite coverage results.
LBDR achieves a high coverage/area value since its implementation costs are very modest. Values higher than 300 are obtained (300 times more coverage than area required). However, RbR achieves a low coverage/area value because although it achieved an overall high coverage value, its area requirements are prohibitive. Even more interesting, we can see that the addition of a de-route port to LBDR is coverage-efficient (see previous section) and area-efficient (Figure 16). So, the designer is exposed to the real benefits of an incremental addition (LBDR_dr) to a previous routing implementation (LBDR).

Figure 17. Switch architecture with source-based routing

3.3.6 Implementation of logic based distributed routing inside the xpipesLite switch: architecture design techniques

The analysis performed so far has revealed that LBDR and its incremental enhancements are a promising way in order to get a scalable, high coverage and low footprint routing implementation for nanoscale on-chip interconnection networks. For this reason, in this section we illustrate the architecture-level design techniques to bring LBDR inside the xpipesLite switch and try to answer questions such as: is the modularity of the switch broken? Does the switch architecture need to be radically changed? How difficult is it to evolve a switch natively conceived for source based routing to LBDR?

The baseline xpipesLite architecture, reported for the sake of reference and understanding, is showed in Figure 17. The switching fabric implements a 2-cycle-latency (one for switch operation and one for traversing the output link), output-queued wormhole-switched router supporting round-robin arbitration on the output ports. The input ports are latched to break the timing path. Allocation of inputs towards specific output ports is handled by an allocator module for each output port (in practice, this is a two phase arbiter). Arbitration is subsequently performed upon receipt of a header flit and output ports are granted until a tail flit arrives. Since the switching fabric natively supports source-based static routing, the routing information is attached to the header flit by the network interface, which checks the transaction address against an LUT.

The length of the routing path field in the header depends on maximum switch radix and maximum hop count in the specific network instance at hand. The switch in Figure 17 is an interesting reference architecture to assess LBDR complexity, since most of the complexity of the routing architecture is on burden of the network interface in source-based routing and therefore the switch exhibits minimum complexity. It just has to read the target output port from
the packet head and to route the entire packet accordingly. As an optimization, the xpipesLite switch architecture rotates away the routing information pertaining the current switch in the head flit. This allows positioning of the per-hop routing bits at a fixed offset within the head flits of the packets, thus simplifying switch implementation.

![Switch architecture with LBDR](image)

**Figure 18. Switch architecture with LBDR**

The implementation of LBDR inside the xpipesLite switch architecture is illustrated Figure 18. In the original switch, the allocator checks the target output port from the head flit of the packet and compares it with its own output port ID, thus generating a *match* signal in case of correspondence. In the modified switch variant, this task is offloaded to the allocator since it is on burden of LBDR logic. This time the head flit contains destination switch coordinates and not routing bits any more. However, the packet length is not affected since in any case the reference architecture places this kind of information in the head flit of the packet and payload is never allowed to mix with the header. LBDR logic is a two stage logic, which is illustrated as a single box for each input port in Figure 18. It is interesting to observe that LBDR keeps the modular design style of the switch architecture. The output signals from the LBDR modules represent exactly the match signals that the allocator used to compute by itself before and which indicate that the packet from a given input port requires a specific output port. The allocator now just has to discriminate between competing requests. LBDR logic therefore fits nicely into the xpipesLite switch architecture. Moreover, the used two-phase allocator requires input LBDR modules to activate only one routing option each, therefore for certain routing algorithms the filtering logic described in the next Section would be needed. In particular, the fixed priority scheme described next is implemented.

### 3.3.6.1 Fixed Priorities, Smart Priorities and Allocators

LBDR provides in some cases more than one routing option depending on the routing bits. For a packet being routed north-east the logic may provide both output ports N and E as eligible for packet forwarding. It is the responsibility of the switch allocator to select one of the output ports. This leads to an increase in the routing flexibility and a potential increase in network performance.
However, in some NoC designs the switch allocator may be simple and does not allow multiple routing options from the same input port. This is the case of switches designed with a two-phase arbiter where allocators are implemented only at the output boundaries of the switch. In this section, we extend the LBDR mechanism in order to allow its use on such switch designs. We provide two basic modifications to the logic in order to provide only one routing option per input port. In the first one, referred to as fixed priorities, each input port will filter some routing options in order to provide only one. This will be done locally at every input port, thus no need for communication between different LBDR implementations at each input port of the switch. The second one, referred to as smart, will make more elaborated decisions when filtering the routing options. Please notice that fixed priorities have been implemented and evaluated in the previous switch design, whereas smart priorities are proposed for future implementation and cost analysis.

3.3.6.2 Fixed Priorities

LBDR logic provides the following sets of two routing options: NE, ES, SW, and WN. This is because all the provided paths are minimal within the topology and the packets are forwarded to one of the possible quadrants (NE, ES, SW, and WN quadrants). The idea behind fixed priorities is to filter such routing options but providing equal probabilities to every output port. This is achieved by providing higher priority to each output port in a different quadrant. Figure 19 shows the extended LBDR logic providing priorities to the N port for the NE quadrant, E port for the ES quadrant, S port for the SW quadrant, and W port for the WN quadrant.

In particular, for the N port, the new logic filters the port if the packet can be forwarded also through the W port (in that case the N port is not eligible for routing purposes). Notice that the N port is not filtered if the packet is going only through N direction or through the NE quadrant (the N port has priority in this quadrant). Similar deductions can be obtained for the remaining set of logic equations.

The main benefit of fixed priorities is the fact that LBDR logic is compact and still isolated at every input port. At the end LBDR provides only one routing option per input port.
3.3.6.3 Smart Priorities

The original LBDR logic providing more than one routing option can be easily coupled with a three-phase arbiter. At the first phase each input port provides requests to the output ports (more than one is accepted). At the second phase each output port independently selects just one routing option and notifies it back to the input port. At the third phase the input port selects one of the accepted routing options. The main drawback of the three-phase arbiter approach consists of the increased arbitration latency and thus increased packet latency. This is one of the reasons to prefer two-phase arbiters in NoCs.

However, in some cases there is still margin to implement some common logic sharing all the requests from input ports in order to make smarter arbitration decisions. It is a design that fits with the lightweight implementation of switches with two-phase arbiters.

Figure 20. LBDR with smart priorities. (a) Filter and control signals layout (b) Details of a control signal

Figure 20 shows a possible implementation of a smart priority mechanism coupled with LBDR. In this case, there is a common logic (Figure 20a) that reads all the routing options provided by every input port at each possible arbitration cycle. In particular, three control signals are activated from each input port, plus four control signals from the local port. Each control signal, labelled iXY, indicates if input port X is requesting output port Y. Possible ports are NEWS and L (local port). The filter logic provides four control signals to every output port of the switch, thus providing sixteen control signals. The output signal labelled oXY means input port X finally requests output port Y. Notice that the filter logic will assert only one control signal for every output port (remember a two-phase arbiter is assumed).

The filter logic is shown in Figure 20b. The goal of the filter logic is to remove routing options whenever the requesting input port is providing two routing options and at least there is another input port (or local port) requesting only that output port. In other words, priority is given to input ports requesting only one output port by filtering routing options whenever possible. The Figure shows a possible logic implementation for such mechanism. The first part of the logic computes five internal control signals (N1, E1, W1, S1, and L1). X1 means X port is requesting only one output port.

The second part of the logic filters routing options. Focusing on routing option iEN (input port E requests output port N) we can see that this routing option is filtered (output signal oEN' is
reset) by the smart filter if there is at least one input port requesting only that output port (control signal iWN is set and input port W is only requesting one output port, iSN is set and input port S is only requesting one output port, or iLN is set and input port L is only requesting one output port). The remaining routing options are filtered in the same way and in parallel. Also, notice that routing options are filtered again using a fixed priority module in order to provide one routing option per input port. In the provided example the final oEN routing option is filtered if the input port is also requesting output port W (for the quadrant NW west direction has priority).

![Figure 21. LBDR oriented node labelling in concentrated k-ary n-mesh topologies.](image)

3.3.6.4 **Extension To More Cores Per Switch**

LBDR natively supports a large range of routing algorithms for k-ary 2-mesh topologies. Although widely used, it is well known that this topology scales poorly with the number of nodes and that it incurs a large area and power overhead. Under certain operating conditions, concentrated topologies become attractive [BER09]. Basically, the idea consists of reducing the number of topology dimensions or (as in our case) of switches in each dimension of a k-ary n-mesh and to increase the number of cores attached to each switch. This way, bisection bandwidth is traded for low latency, area and power.

LBDR was extended to support multiple cores per switch in concentrated mesh topologies. For this purpose, a different labelling scheme for network nodes had to be devised, since LBDR originally required the switch coordinates within the 2D mesh. Now, multiple cores might be associated with the same switch coordinates. The basic idea is that one local core inherits the same coordinates of the switch, while the other ones have an incremental x coordinate. From a network viewpoint, x coordinates of the switches appear to increase at a coarse granularity, where the granularity is determined by the number of cores attached to each switch. The figure illustrates the case with 4 cores per switch.

This implementation has two advantages: (i) the packet head does not need to be changed since it still carries the destination switch coordinates; (ii) only the first logic stage of LBDR needs to be slightly extended, while the second one is unaltered.
Modification of the first logic stage of LBDR includes the return of a match signal whenever the target y switch coordinate matches that of the local switch AND the target x coordinate falls within the range of the x coordinates of the local connected cores. In this case, the packet is forwarded to the right local output port. Vice-versa, if the packet is headed to another switch, the native LBDR logic can handle this without any modification.

![Graph showing scalability results](image)

**Figure 22.** Scalability results of a switch with source routing vs one with LBDR for concentrated topologies. X-axys is the switch radix. (a) Timing results (b) area results

By implementing this extension, we were able to implement 6x6 and 8x8 switches for the support of 2 and 4 cores per switch respectively. Comparative timing and area scalability results are illustrated in Figure 22a and Figure 22b. While critical path delay in the LBDR switch has been scaled almost linearly by the synthesis tool, some unpredictable optimization has been performed on the xpipesLite 6x6 switch. However, the 8x8 switch implementation confirms that the timing gap between the two variants is kept limited within 6%. Again, area results are promising for the LBDR switch, which shows a good scalability of this metric even to 8x8 switches.

In all cases, the delay and area differences between the two switch variants do not follow a clear scalability trend but have to be assessed case by case, since they are tightly dependent on the optimizations the synthesis heuristic is able to perform on the specific design at hand. This further confirms that the overhead of LBDR is so small that it falls within the unpredictability margin of the synthesis tool behaviour.

### 3.3.7 Summing up

In this early part of the report we have assessed an alternative approach to routing implementation than table-based routing. In particular, logic-based distributed routing, combined with few configuration registers at each switch, allows for efficient implementation of most of the existing distributed routing algorithms in regular as well as many irregular NoC topologies derived from a 2D mesh. Therefore, logic-based distributed routing overcomes the flexibility limitations of historical algorithmic routing strategies, traditionally constrained to the specific topologies they are conceived for.

By comparing area and routing delay of distributed routing logic with those of routing tables, the superiority of LBDR has been showed. Even considering register and memory macro implementation options for routing tables (when applicable by the memory compiler), LBDR always proved the most area-saving solution and already performance-efficient in the worst case scenario. Above all, the constant area and delay of LBDR with an increasing number of destinations in the network makes the point for its better scalability.
The implementation of LBDR in a real-life network-on-chip switch architecture and a comparative analysis with a lightweight switch for source-based routing show a delay penalty of only 5% and even a small area saving at the 65nm technology node. In general, its overhead is so small that it falls within the behaviour unpredictability margin of synthesis tool heuristics. Moreover, LBDR was extended to support the connection of multiple cores to the same switch, and the nice comparative properties with respect to a switch for source-based routing have been proven to scale to higher values of the switch radix. Above all, the architecture design techniques have been documented in this report.

The true reason why logic-based distributed routing was analysed in this report was to assess its suitability for fault- and variability-tolerance in on-chip networks. For this purpose, we have developed an yield-oriented evaluation methodology for routing implementations. The methodology is able to determine the fault coverage the routing implementation is capable of in the presence of a network affected by a given failure pattern. The failure pattern is a function of the target frequency, since as an effect of link delay variability a decrease of the target frequency may render some links back operational. A few logic-based distributed routing mechanisms (and variants thereof based on their configuration parameters) were explored, and it was found that LBDR with some incremental modifications can achieve from 60% to 100% coverage depending on the parameters of the variability model. Alternative approaches can do better (e.g., RbR), but with a prohibitive area cost.

Finally, please notice that the evolution of the xpipesLite switch to virtual cut-through switching would enable the LBDR mechanism to be augmented for 100% coverage on strongly connected irregular topologies derived from a 2D mesh. It was not possible to explore this alternative given the effort planned for task 7.3 in the Galaxy project. However, we feel that this preliminary work sets a clear direction to take for future variability- and fault-tolerant networks-on-chip, and illustrates architecture design techniques for designers eager to go this way. Also, an efficient configuration methodology for connectivity and routing bits will have to be devised as a future work.
3.4 VARIABILITY DETECTOR FOR RANDOM PROCESS VARIATION-TOLERANT GALS LINK DESIGN

After viewing system level issues related with process variation and manufacturing defect--tolerant GALS NoC design, we now move to an abstraction layer closer to the actual architecture implementation. At this level, the specific architecture paradigm of the GALS NoC platform comes into play and is essential to devise suitable optimizations to enhance variability robustness. Our reference is to the target GALS NoC illustrated in 3.1 and developed throughout the activities of Workpackage 6 of the Galaxy project. In particular, we focus on the mesochronous links bridging switches with each other and implementing the tightly or loosely coupled mesochronous synchronizers illustrated in Deliverable D6 and in [LUD10] at the receiver end.

Such synchronization interfaces require the circuitry to be able to reliably and efficiently move data across clock domain boundaries. With the onset of significant process variation effects, the behaviour of source synchronous interfaces might not match the initial expectations any more, and the risk is to have carefully engineered local islands of synchronicity unable to work because of the failure of the synchronization interface providing processing data.

As recalled many times throughout this report, source synchronous interfaces should be able to route the source synchronous clock along with the data and to guarantee a good alignment between the clock signal and the data lines (i.e., to minimize the routing skew).

Unfortunately, even though the physical design techniques illustrated later on in this report are applied, the effect of random process variations might invalidate the assumption and cause the failure of the synchronization interface. In practice, every routed signal experiences a different link delay and the alignment property between clock signal and data or between data lines with each other will be loosened at the receiver boundary.

This document proposes a novel circuit guaranteeing the reliability of the NoC source synchronous interfaces under high process variation conditions. The circuit should be placed in front of the far-end synchronizer, and builds on the principle of self-calibration: the circuit in practice adapts to the in-situ actual operating conditions. This circuit has been validated through a number of experimental results, where process variations have been injected in the GALS link AND in the source synchronous interface as well. The outcome of the reliability tests have been compared with those of a baseline interface, thus proving the increased robustness.

3.4.1 architecture of the variation detector

This section presents a novel architecture circuit guaranteeing the reliability of the NoC source synchronous interfaces under high process variation conditions. The circuit should be placed in front of the far-end synchronizer, and builds on the principle of self-calibration: the circuit in practice adapts to the in-situ actual operating conditions. This circuit has been validated through a number of experimental results, where process variations have been injected in the GALS link AND in the source synchronous interface as well. The outcome of the reliability tests have been compared with those of a baseline interface, thus proving the increased robustness.
Under this condition, the arrival time of the data becomes unpredictable and the synchronism property between the source clock signal and the incoming data is lost. As a result, the stand-alone synchronizer (a mesochronous synchronizer in our case) is not able to work properly incurring in sampling violations and metastability issues. Process variation concerns can be tackled with the help of the architecture proposed in this document. It can be tightly integrated into the synchronizer at the receiver end (i.e., mesochronous synchronizer or dual clock FIFO synchronizer) restoring the required alignment between source clock signal and data.

In order to achieve this purpose, the variation detector architecture senses the offset between the source clock signal and the data signal. Therefore, it provides the receiving synchronizer with a clock signal version able to guarantee a safe data sampling. The clock signal is a delayed replica of the transmitted source synchronous clock. A schematic of the proposed architecture is illustrated in Figure 23.

It is composed of a parametric number of brute force synchronizers sampling the output of an AND block. The AND gate receives the incoming data as input generating a high value as soon as all the data input bits are high. The source synchronous clock signal (routed along with the data) crosses a set of delay chains introducing an incremental amount of delay with respect to the nominal source clock.

In order to preserve the synchronism property between reset and clock signal, the reset is bundled with the data as well and it crosses the delay chains together with the clock. The number of delay chains depends on the number of brute force synchronizers in order to guarantee to every synchronizer block a clock/reset with a different offset. Moreover, it depends on the amount of expected routing skew between data and clock and between data lines with themselves.

Figure 23. Architecture of the variation detector
Finally, the outputs of the brute force synchronizers select through the multiplexer the clock signal offset required to feed the receiving mesochronous synchronizer with a safe strobe signal. The selection of the safe clock signal takes place during the NoC reset phase and it represents a key step in the proposed architecture. It is detailed in the following chapters. However, it is worth recalling that once the strobe signal is selected during the reset process (for instance, at system bootstrap), then the selection stays the same throughout the entire use cycle of the system, thus not generating an overhead for correct system behaviour. Used over time, the proposed architecture allows the network to deal also with wear-out effects.

3.4.2 Variation detector at work

The architecture of the variation detector provides the safe clock signal before the tightly/loosely coupled synchronizer starts its operation. To achieve this result, the proposed architecture comes into play during the NoC reset phase.

Furthermore, to guarantee the synchronization between the incoming data and the source synchronous clock signal, the circuit must detect the exact arrival time of every data bit in the GALS link. This is possible by sensing the incoming bits when they are switching from a high logic value to a low logic value or vice versa (a stable bit does not provide any arrival time information). As a result, the proposed architecture can properly sense routing skews when it receives two consecutive data patterns so that every data bits of the first transaction is negated with respect to the data bits of the other one. These two transactions can be purposely generated from the out_buffer of the upstream switch during the reset process.

To meet this goal, we designed an out_buffer in the xpipesLite architecture capable of driving during the reset phase a sequence of low-logic value bits (00..0) followed by a sequence of high-logic value bits (11..1). It should be noticed that the proposed out_buffer required only an additional token ring counter to provide this behaviour.

Figure 24 reports the post-layout reset phase waveforms of a variation detector composed of 5 delay lines and 5 brute force synchronizers.

![Figure 24. Variation Detector Reset phase](image-url)
As represented in the figure, once the first sequence of zeros is driven at the variation detector input port, the output of the AND module is set to a low logic value ("and_out"). In the meantime, the source synchronous clock signal ("clk tx") is propagated through the delay lines and it generates 5 replicated clock signals with a different offset ("clk_A", "clk_B", "clk_C", "clk_D", "clk_E"). Until the incoming sequence has low value, the brute force synchronizers sample the low AND output and they set to zero the control signal driving the multiplexer (see "control[5]" in Figure 24). In this configuration the multiplexer output is permanently at a low logic value and the receiver mesochronous synchronizer does not receive any clock signal.

Then, the transmitter starts to drive the sequence of ones and the data bits ("flit tx") start to switch at the variation detector input port. Finally, when the incoming data is stable and every data bit has switched to the high logic value, the AND output assumes a high logic value.

Although the AND output feeds all the brute force synchronizers, only some of them are able to sample the high value at the AND output. In fact, the brute force synchronizers driven by clock signal with an early positive edge will not reveal the high value; on the contrary, the brute force synchronizers driven by the clock signal with a late positive edge will sample it and will set the multiplexer control signal. As result, the control signal collects all the information about the result of the AND output sampling and this latter information is exploited to distinguish between the safe clock signal and the potentially unsafe clock signals (early clock signals). In Figure 24, the control[3] and the control[4] signals are set to the high logic value and this means that the clk_D and clk_E are safe: the positive edges of these latter clock signals occur after each data bit have switched to the high logic value (i.e. the data is stable).

Since it is the first of the safe clock signals to be selected to cross the multiplexer, in our example, the clk_D is the clock signal driving the receiver synchronizer ("clk_synch"). It should be noted that a source synchronous synchronizer in a similar scenario could not properly work if driven by the nominal source clock signal (clk_tx). In fact, as showed in Figure 24, the positive edge of the source clock signal occurs when the incoming data are still switching.

The sequence of 1s is driven by the transmitter during the last clock cycle of the reset phase. As result, once the control signal in the variation detector is correctly set, the reset switch to the low value and the variation detector circuit interrupts its operation by freezing the safe clock signal at the multiplexer output. Since the detector circuit works only during the reset phase, the power consumption overhead is minimized.

### 3.4.3 Validation

To validate the proposed architecture a set of experiments was performed. Since the goal of our architecture is to support a source synchronous communication in a high variability environment, we instantiated a platform composed of a transmitter sending data along with the clock to a mesochronous synchronizer (the receiver) and we injected process variations in the link wires. The mesochronous synchronizer is tightly coupled into the switch architecture.

In particular, in order to study the robustness of the communication when the ideal alignment between the data and the source synchronous clock is not respected, we inject an increasing delay into the clock signal wire. Therefore, we compared the performance of a baseline source synchronous architecture with the one achieved by a source synchronous architecture augmented with the proposed variation detector.
Figure 25. Timing Margin of the Mesochronous Synchronizer

Figure 25 reports the timing margin derived from a post-layout analysis of a source synchronous mesochronous synchronizer synthesized at 1GHz WITHOUT the variation detector. Setup and hold times have been experimentally measured by driving the mesochronous synchronizer under test with a source synchronous clock affected by an increasing delay and by monitoring the relative waveforms. X-axis reports the amount of delay (expressed as percentage of the clock period) injected into the clock line with respect to an ideal clock with null routing skew. The figure also compares setup and hold times with the minimum values required by the technology library for correct sampling (denoted FF_Time).

First of all, we observe that the setup time increases and the hold time decreases linearly with the increase of the clock delay. When the clock delay reaches 65% of the clock period, the hold time violates the minimum timing margin and a failure is detected in the mesochronous synchronizer. Injecting a clock delay between 70% and 100% of the clock period, the source synchronous clock starts to sample the next incoming data and the behaviour of the system is unpredictable. As a conclusion, the baseline mesochronous synchronizer was able to support a clock skew with data between 0% and 65% of the clock period. 35% of the clock period is unsafe and a positive clock edge, loosely synchronous with the data, can only occur during the early 65% of the clock period.

This experiment was carried out considering an incoming data stable for 75% of the clock period at the mesochronous synchronizer input port (see Figure 26). This accounts for the delay variability between data lines and for the non-null settling time usually found in post-layout link switchings.
In order to perform a fair comparison, a similar experiment was performed for a mesochronous synchronizer WITH the variation detector connected in front of it. In this scenario, the transmitter clock crosses the variation detector before to reach the mesochronous synchronizer. The timing margin of this second system is showed in Figure 27.

We can notice a significant difference of results due to the variation detector integration. The setup margin is strictly symmetric with respect to the hold margin. Moreover, the timing margin is composed of the periodic repetitions of the same triangular-shaped trend.

As before, at the increase of the clock delay corresponds a decrease of the hold margin although, in this case, as soon as the hold time degradation becomes significant and dangerous for correct circuit operation the detector re-establishes safe margins. This effect is achieved with the help of the variation detector selecting a delayed clock with a different offset when the timing margin becomes low. As result, we have 5 periodic repetitions of a triangular-shaped trend corresponding to the output of the 5 delay lines sequentially selected.

In conclusion, the proposed architecture guarantees a safe communication in a mesochronous link in every clock delay scenario as opposed to a baseline source mesochronous architecture.
3.4.4 Experimental results

Experimental results are structured into two subsections. In the first one, the process variations are introduced in the data link wires and the post-place&route results are presented documenting robustness difference between the baseline source synchronous architecture and the proposed architecture (mesochronous synchronizer augmented with the variation detector). In the second one, the process variations are injected into the variation detector cells too in order to prove the robustness of the proposed architecture overall. All physical synthesis experiments have been performed with a 65nm STMicroelectronics technology library.

3.4.4.1 Process variations in the data link wires

Although it is essential to guarantee the synchronization property between the data signal and the source clock signal, the size of the timing window when the data is stable at the synchronizer input port is also an important parameter for a reliable communication.

In particular, we measured the performance of the baseline source synchronous architecture and the proposed architecture varying this latter parameter. Since the considered timing window is longer when the data bits arrive simultaneously, this parameter is tightly dependent on the amount of process variations affecting the upstream link. Therefore, we injected an increasing amount of delay variability in the wires carrying the data bits and we monitored the behaviour of the synchronizer under test.

Figure 28 reports the timing margin derived from a post-layout analysis of a mesochronous synchronizer synthesized at 1GHz. The timing window when the data is stable is identified by the two red lines crossing the y-axis in (0,1) and (0,-1). Y-axis reports the position of the data sampling inside the timing window: a sampling close to the upper bound can incur setup violations; on the contrary, a sampling close to the lower bound can incur hold violations. In a high variability environment, the data should be sampled as close as possible to the middle of the timing window to avoid timing violations. The position of the data sampling in the timing window is reported as a function of the delay between the data and the source synchronous clock signal. The figure also compares the results in the presence of a different amount of delay variability in the data link wires, i.e. different timing window sizes.

The analysed timing windows range from 95% to the 50% of the clock period size. As expected, with the increase of the clock delay, the data sampling time moves closer to the lower bound of the window until an hold violation occurs. Moreover, the higher the delay variability in the data wires the lower the probability of sampling in the safe timing window.
Figure 28. Timing Margin of the Baseline Mesochronous Synchronizer in the presence of delay variability in the data link wires.

A similar experiment was carried out by reporting the timing margin of a mesochronous synchronizer augmented with the variation detector (Figure 29). As result, the data sampling occurs inside the safe timing window in every experimented scenario.

We can notice how the sampling time moves closer to the unsafe lower/upper bound as the timing window size shrinks. In every case, the proposed architecture does not incur timing violations even when the data is stable for only 50% of the clock cycle.

Figure 29. Timing Margin of the Proposed Architecture in the presence of process variation in the data link wires.
3.4.4.2 Process variations affecting the detector

If the previous experimental results proved the capacity of the variation detector to absorb the variability injected into the link, this section shows the effect of process variations affecting the variation detector on the correct operation of this latter. To meet this goal, random delay variability was injected into each logic cell of the detector architecture. The cell delay variability was generated by following ITRS predictions for a 65nm technology library.

In Figure 30, we measured the timing margin of the proposed architecture in the presence of minimal, typical and maximal random process variation conditions. We considered a timing window, when the data is stable at the synchronizer input port, of 50% of the clock period size. The timing margins are derived from a post-layout analysis performed at 1GHz.

As reported in the figure, the variation detector proves to be robust to the random variability achieving similar performance in the different process variation scenarios. Similar tests have been performed by applying the variability model in [HER10] to the delay of the logic gates, and considering worst cases also for 45 nm technology variations. Again, the result proved the relative insensitivity of our detector to the amount of injected delay variability.

Therefore, we were able to avoid the risk of having a variability-sensitive detector of link delay variations, which would not have made any sense.

Figure 30. Random Variations affecting the detector architecture.
3.4.5 summing up
This section of the report has illustrated a variation detector to be used in source synchronous links in front of the receiver synchronizer. The detector was in particular conceived to deal with link delay variability, whose effects are twofold:

- It induces a routing skew between the source synchronous clock signal and data lines
- It induces a relative routing skew between the lines themselves, thus shrinking the stability timing window of data in front of the sampling element at the synchronizer.

The detector builds on the principle of self-calibration, i.e., adaptation of the receiving circuit to in-situ actual operating conditions. In practice, the detector selects a delayed variant of the source synchronous clock signal so to be able to safely latch the input data. In tandem with the physical routing technique that will illustrated in next section, these design techniques can significantly increase the robustness of a GALS link to systematic as well as random process variations. Above all, the variation detector has been proven to be robust to the delay variability of its logic cells.
3.5 Development of a New Physical Routing Approach for Robust Bundled Signalling on NoC Links.

With respect to previous sections, let us now move down the design hierarchy and delve into the physical synthesis process, and into the physical routing process in particular. Interconnects have moved to the forefront as the limiting factor in IC performance, and dealing with them during the routing process is mandatory. Nowadays, there are cases where interconnect delay, especially that of global interconnects, accounts for more than 75 percent of the total path delay. In addition, the increasing link delay variability translates into a mismatch of the wire delays building up a link. This is very critical for the reliability of NoC links, which make increasingly use of advanced synchronization and signaling schemes, to improve IP decoupling and energy efficiency [MED08].

In this work we focus on routing of NoC links, which often span a significant on-die distance. At a first glance, link physical routing can be carried out with a standard physical implementation flow, which routes each net independently, while trying to match design constraints on delay, area and power. However, routing algorithms are extremely complex and influenced by a number of factors (such as wire congestion). Hence, in practice it is extremely difficult to ensure that wires which take widely different routes maintain closely-matched delay and power consumption. In other words, when the logical atomicity of a link is broken at the physical level, we experience significant length, resistance and load deviations among different wires of a link. Reducing delay and power variation between wires of a link is a key requirement for advanced NoC implementation styles. First and foremost, robust Mesochronous [LUD09] and Globally-Synchronous, Locally-Asynchronous (GALS) NoCs [LUD09,MED08] rely on source-synchronous signaling, where the data and strobe signals have to arrive at the destination within a narrow timing window to achieve correct and high performance operation.

We propose a link routing approach which considers all nets of a link together and efficiently routes NoCs links as an atomic entity (a bundle). This routing step is performed early in the design implementation flow, at the global routing step, and it integrates seamlessly with the design implementation tools to ensure that link quality will be fully preserved during detailed routing. As a result, the variability in link net attributes such as length, resistance, load, and delay is much reduced. Moreover, enforcing the same “trajectory” on all nets of a link facilitates wire spacing and shielding. Finally, bundled routing improves the routability of the NoCs link, as sometimes even a powerful place and route tool struggles to perform detailed routing of a large number of links. This can significantly reduce routing run time as well as the time for parasitics extraction.

3.5.1 Motivating example

All large body of work in NoC designs [LUD09, MED08] has recently focused on GALS and mesochronous links, in some cases pushed by the activities of the Galaxy project. These approaches require a “strobe” signal (normally a copy of clock signal of the sender) to be provided with the data/flow-control signals to give a time reference for sampling data at the point of maximum eye opening. These approaches require tight matching of the propagation delay of the strobe signal with that of data and flow control wires. The maximum phase mismatch between strobe and data signals for safe input data sampling is a small fraction of the clock cycle, (1/10 or less); otherwise, the receiver latch enable signal might be activated too late or too early, with disastrous consequences on probability of metastability and error rates.
In practice, precise delay matching between link wires can be achieved in regular (tile-based) NoC designs with manually planned bundled link routes. However, manual link routing is difficult or even impractical in SoCs with irregular tiles and topologies, where the NoC is automatically instantiated and then handled at the RTL to the design implementation flow. Unfortunately, even by using the most recent place&route (PnR) tools, there is always an amount of variation between the propagation delay of data, flow-control and timing signals which increase the phase mismatch between data and strobe signals. Figure 31 shows a part of an NoC layout routed in the most recent version of Synopsys IC Compiler. Link wires have different routing trajectories, which leads to different lengths and delays. Although in some cases the nets have the same lengths, their delays are different due to vias. Note that this layout has been obtained with the automatic bus routing feature of the tool enabled. Even if it may theoretically be possible to achieve perfect delay matching for irregularly routed links, this would imply numerous design convergence iterations (which take days for an average size SoC).

Moreover, high-performance NoC implementation often relies on low-swing signaling to boost energy efficiency [BER05]. Signal-to-noise ratio is a serious concern in both low-swing and full-swing communication [BER05]. Shielding and wire pacing techniques are commonly used to control the crosstalk effects in global interconnects [RAV03]. Managing the routing path of the wires makes it easier and more efficient to apply these techniques on the global interconnects. These goals can be achieved by having tight control on the link trajectory and keeping all wires of a link together. This is helpful not only to have matched delays, but also to manage and reduce, through link shielding and spacing, crosstalk noise from bus wires as well as for other interconnects.

![Figure 31. Link route obtained with state-of-the-art P&R.](image-url)
3.5.2 NoCs link routing flow

Physical routing is divided into two main steps: global routing and detailed routing. In the global routing phase the path of any net is determined as the Manhattan distance of that net without considering the routing blockages and constraints. In a traditional flow, detailed routing considers nets independently, and after this phase each net of a link in NoC has different trajectory leading to high deviation in resistance, capacitance, length and delay among bits of a link. Note that, a link is a group of logically related interconnects. As outlined before we propose a methodology to manage the routing of NoC links in which we consider all bits of each link together. Figure 32 shows our routing methodology, consisting of four main steps: (i) pin placement and ordering (ii) prioritizing links (iii) creating virtual routing blockages and routing one net of each link as the link agent and (iv) applying the agent's trajectory to other nets of the current link.

![Diagram of Link Routing Flow]

Figure 32. Link routing flow
3.5.2.1 Pin placement and ordering

Before routing, we perform pin placement and ordering. In this phase for each link we perform two steps: (i) determining the side of pins around the blocks, and (ii) specifying the pins order (named pin placement and pin ordering, respectively). We consider all components in the floorplan as rectangular objects. Therefore, there are four possible sides (North-East-West-South) for the pin positions of each component, and, therefore, for each link we have 16 possible options for the pin placement (each link is connected to two blocks).

In the pin placement step, for each link we have four possible sides for both source and destination blocks. We empirically realized that to have the same length for all nets of a link it is better to place the input and output pins of a link on different sides. For example, if we place the output pins of a link on the East side of source block, we do not consider East side of destination block for the input pins; therefore, for each link we have 12 different solutions to place all input and output pins. Each solution is a pair of sides; for example Pair (West, East) shows that the output pins are placed on the West side of the source block and the input pins are place on the East side of the destination block. We start with those links whose source and destination blocks are farther from each other. From the 12 possible solutions for each link, we select a feasible case that leads to shorter Manhattan distance for each link. A solution is feasible if there are enough spaces on both sides of the pair to place all pins of the related link. However, if all pins of a link cannot be placed on a side because of already placed pins of other links or because of lacking space, the solution is not feasible and we have to select the next pair.

Table 3 Pin ordering for 6 pin placements

<table>
<thead>
<tr>
<th>Pin placement</th>
<th>(North,West)</th>
<th>(North,South)</th>
<th>(North,East)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pin Order</td>
<td>(LR,DU)</td>
<td>(LR,LR)</td>
<td>(LR,UD)</td>
</tr>
<tr>
<td>Pin placement</td>
<td>(West,North)</td>
<td>(West,South)</td>
<td>(West,East)</td>
</tr>
<tr>
<td>Pin Order</td>
<td>(DU,LR)</td>
<td>(UD,LR)</td>
<td>(UD,UD)</td>
</tr>
</tbody>
</table>

To have the same length for all nets of a link, we order pins after pin placement. In this step, we assign an order to pins of one side of each link and update the pins order of the other side based on that; order of pins can be from LSB to MSB or wise versa. To make the problem easier, for each pair of 12 possible placement options we define a pair of orders. Each item of the order pair shows the direction of LSB to MSB. For example the order Pair (LR, UD) means that the output pins at the source block should be placed in order of LSB to MSB from left (L) to right (R), and the input pins at the destination block should be placed in order of LSB to MSB from up (U) to down (D). Note that block position and routing trajectory of a link do not change the pin orders, and the only factor which affects the order of pins is the sides where pins are placed on source and destination blocks. Therefore, we have a fixed table to
determine the pin orders for each of 12 pin placement pairs. Table 3 shows pin orders of 6 different pin placement pairs.

3.5.2.2 Prioritizing links
After setting the pin positions and orderings for all links, we give a priority to each link to route the most critical links first. Priority is based on the Manhattan distance of each link, to increase the chance to meet timing constraints on more critical links which have to span a longer distance. In fact, the router will be faced with lower congestion when routing the high-priority long links. When routing resources become tight, routing short links that need less routing resources is much easier than routing long links.

3.5.2.3 Link routing
The basic idea is to first route one net of a link (called link agent), and then route the other nets along the same trajectory [MOF07]. Our algorithm entails the following steps: (i) extending the routing blockages to force the router to find a path where the whole link can be routed (ii) routing the middle net of the link (iii) applying the trajectory of the middle net to the other nets (iv) creating routing blockages around the routed link. In the first preparation step, overlapping with routing blockages is prevented for the current link by extending existing routing blockages on all sides by W/2 where W is the width of the link. As a result, we create virtual routing blockages and force the router to route the link agent at a distance of W/2 from the existing actual routing blockages. These virtual routing blockages will be removed after routing the middle net leading to a narrow channel where the entire link can be routed.

After the preparation step, we consider the middle net of the current link as the link agent, and ask the router to route that net. To do so, differently from [MOF07] we take advantage of a powerful timing-driven commercial router. We set timing as the objective of the router. If the routing succeeded to route the middle net it creates a routing trajectory and a channel for the link where the entire link can be routed. However, if the router fails, the nets of the link can not be routed together with the same trajectory with the current constraints and floorplan which means the link is not routable. In this situation we add the current link to the unroutable links list, report a warning message and will move to route the next link in the prioritized list. We will route all unroutable links using the normal flow of the tool as a fall-back option.

After routing the middle net, we first remove the virtual routing blockages, and then apply the trajectory of the middle net to other nets of the link. To do this, we use a two-step algorithm. In the first step we replicate the vias of the middle net for all other nets at the appropriate position determined by the pin placement and ordering of the related link. In the second step, for each net we connect the source pin to the first via and then connect the vias together sequentially, and finally connect the last via to the destination pin. Note that if there is no via for the middle net, it means the middle net is fully horizontal or vertical, and in this case we simply replicate the middle net trajectory for other nets.
For each net of the link, via positions are computed in such a way that the via-to-via length of all nets of a link is equal. Because of suitable pin ordering that we considered in the first step, the length difference between different nets of a link from source pin to the first via will be compensated by length difference from the last via to the destination pin. Figure 33 shows an example. As it can be seen in Figure 33, all net segments from via group A to via group B have the same lengths, this is also true for all segments from via Group B to via Group C. Note that, since the source pins are aligned, the segments from source pins to via Group A have different lengths, but this difference is compensated by length difference from via Group C to destination pins. Therefore, all nets of the link have the same lengths.

To replicate the vias of the middle net for the other nets of the link, we have to calculate the positions of vias for each net. To do so, we have two functions F and G to calculate horizontal and vertical positions of each via, respectively. F and G are functions of middle-net's via position, wire to wire distance, and pin placement pair of the related link; they are defined for all 12 possible options of pin placement and ordering. To clarify the algorithm for routing the middle net and applying the trajectory of middle net to other nets, we show it step-by-step on an example shown in Figure 34.
Process Variation Tolerance in GALS Systems

Figure 34. Link bundle routing steps

Assume a part of an NoC layout shown in Figure 34-a in which there are some existing routing blockages. We have already performed the pin placement and ordering phase on this layout, and we are going to route link L12 from switch S1 to switch S2. First we extend the existing routing blockages as W/2 where W is the width of Link L12 which leads to the layout shown in Figure 34-b. Then we ask the router to route the middle net of the link, the middle net trajectory is shown in Figure 34-c. As it can be seen in Figure 34-c the middle net has 3 vias. Assume that the space between two wires for Link L12 is P, the wire width is H, and the link has M bits. To replicate the vias for other nets of Link L12 we use the formula:

\[
X^i_j = F(X^M/2_i, j, P, H), \quad \text{where } X^i_j \text{ is the horizontal position of } i\text{-th via for the } j\text{-th bit of the related link. For Link L12 we have: } F = X^M/2_i + (j-M/2)(P+H) | j \in \{1,...,M\}, i \in \{1,2,3\}
\]

\[
Y^i_j = G(Y^M/2_i, j, P, H), \quad \text{where } Y^i_j \text{ is the vertical position of } i\text{-th via for the } j\text{-th bit of the related link. For Link L12 we have: } G = Y^M/2_i + (M/2-j)(P+H) | j \in \{1,...,M\}, i \in \{1,2,3\}
\]

After replicating all vias for all nets, we simply connect the vias of each net together and connect the source pin to the first via and the destination pin to the last via. Figure 34-d shows the routed link. We define a set of F and G functions for all 12 possible pin placements.
and orders. The distance between any two subsequent vias for all nets on a link is constant and does not depend on the bit number, thus the only factor that can change the wire length for different nets of a link is the distance from Source to the first Via (SV) and the distance from the last Via to the Destination (VD). By specifying the appropriate pin orders, and suitable F and G, the sum of SV and VD would be constant for all nets of a link and does not depend on the bit number, thus, leading to the same length and trajectory for all nets of that link.

Figure 35 shows the same layout of Figure 31 which is routed by our algorithm. As can be seen in this figure all wires of each link are routed together with the same trajectory (outlined in red).

![Figure 35. Bundled link routed layout example.](image)

### 3.5.3 Crosstalk handling

There are two different types of crosstalk affecting bits of a link: the crosstalk between nets of a link and other nets of the design (inter-link crosstalk) and the crosstalk among different nets of a link, themselves (intra-link crosstalk). We have solutions to control both of them.

To handle the first case, we create a routing blockage around each link. As we know the trajectory of the entire link, we can easily create a routing blockage with any size around the routed link. Modifying this blockage's size avoids aggressor signals or other links being routed too close to the current link. For the intra-link crosstalk, we use two techniques to decrease the crosstalk between nets of a link: spacing and shielding. In the spacing method we tune the space between different nets of a link to reduce the crosstalk delay. We start with the minimum spacing and check the crosstalk delay. If it was satisfying we keep the spacing; otherwise we gradually increase the spacing to reduce the crosstalk effect. We modify the virtual routing blockages and possibly change the trajectory of the link to apply spacing.
In the shielding technique, we use ground or power nets as a shield for the nets of a link. This converts the coupling effect between two nets to coupling effects between nets and ground/power nets and subsequently removes the crosstalk effect. As we route all nets of a link together, except the first and last nets, we can use one shielding net for two nets of a link. Therefore, in our routing methodology, for a link with $M$ number of bits, we need $M+1$ shielding nets to remove the crosstalk effect between different nets of that link. Also, one alternative solution is to mix shielding and spacing techniques to handle both inter and intra link crosstalk. This can be carried out by spacing wires in the link and decrease inter-link crosstalk with two shielding wires at the boundaries of the link to handle inter-link cross-coupling. Our tool flow supports all these options.

### 3.5.4 Experimental results

To evaluate our methodology, we applied it on a set of benchmarks which contain several macro blocks and different number of switches and links. Table 4 shows their characteristics. For each case, we performed bottom-up synthesis and placement and routing of the blocks. We used Synopsys IC Compiler and Design Compiler for PnR and logic synthesis, respectively. We mapped all designs on CMOS 45nm technology. For pin placement and ordering phase, we used IC compile commands to place pins around blocks. To implement our routing algorithm, we took advantage of routing guides and the proper script commands to create nets and vias in IC Compiler. Our algorithm and flow is implemented in TCL using IC Compiler scripting facility.

We compared the original flow of the tool with our link routing technique. Note that the tool has a feature for manually drawing wires together; however, we compared our algorithm with its alternative automatic flow. In the original automatic flow of the tool we used the bus routing feature of the tool which is a tool’s feature for automatic bus routing and compared the results of this flow with those of obtained by our algorithm. Results are compared in terms of wire length, resistance, load, and delay variations. We also compared the actual net delay of these two ways, to ensure that delay matching is not obtained at the price of a significant increase in average delay.

### Table 4 Characteristics of the benchmarks.

<table>
<thead>
<tr>
<th>Design</th>
<th># of Macros</th>
<th># of Switches</th>
<th># of links</th>
</tr>
</thead>
<tbody>
<tr>
<td>D1</td>
<td>6</td>
<td>10</td>
<td>10</td>
</tr>
<tr>
<td>D2</td>
<td>7</td>
<td>12</td>
<td>8</td>
</tr>
<tr>
<td>D3</td>
<td>8</td>
<td>13</td>
<td>10</td>
</tr>
<tr>
<td>D4</td>
<td>21</td>
<td>18</td>
<td>15</td>
</tr>
<tr>
<td>D5</td>
<td>31</td>
<td>20</td>
<td>18</td>
</tr>
</tbody>
</table>

Figure 36-a compares the wire length variation of different links in D3. As can be observed, our routing methodology has lower length variations than the original flow. Almost in all cases the variability of wire length in a link for our method is zero. In some rare cases it is not zero, as
our pin placement step could not find enough space around the related block and had to place pins on the same side of source and destination which leads to different length. However, this deviation is very small and comes from the length difference at the first and last vias and is at most 2*W where W is the link width.

![Graph A](image1)

**Figure 36. (a): Wire length variation (b): Number of vias**

Figure 36-b compares the number of vias per each link in D1. Generally, the number of vias in our routing is smaller than that of the normal flow. This is another factor which decreases variability and even improves delay.

Figure 37-a and Figure 37-b compare the wire resistance and delay variations respectively for D3. As seen, the delay variation between wires of links is decreased by an amount in the range of 30-70% in our routing methodology compared to that of the normal...
flow. Note that, the wire length deviation of, for example, Link1 in our flow is zero, but the delay and resistance deviation are non-zero. These deviations in delay and resistance are not related to the wire length, but are due to detailed deep-submicron manufacturing (DSM) effects that commercial tools consider to calculate the net characteristics. Synopsys IC compiler models the etching effects which will result in trapezoidal wire cross-sections for wires close to each other. This will effectively change the R and sidewall C, and therefore alter the wire delay. Figure 38 compares the maximum and average of link delay deviation, link load deviation and the worst case delay for all designs. As it can be seen, our method does not impose timing overhead on the links and in some cases the delay of links is 5 to 10 percent better than that of the state-of-the-art timing driven flow.

![Graph a: Resistance Deviation](image1)

![Graph b: Delay Deviation](image2)

Figure 37. a: resistance variation  b: delay variation.
As we explained before, from the signal integrity point of view our methodology gives better results since we have control on the entire link route. We applied spacing and shielding to reduce the crosstalk effect on links. In spacing we increased the space between different wires of each link and in shielding we shielded all nets of each link with a grounded net. Figure 39 compares the average of crosstalk delta delay for each link in D2 for a normal flow with that of our approach with/without shielding. Crosstalk delta delay is a portion of the net delay which is related only to the crosstalk effect on that net. Figure 40 compares the average of delta delay for each link in D2 for the normal flow with that of our approach with three different spaces between nets of links. As can be seen, although the crosstalk delta delay of normal flow is sometimes smaller than that of our flow without shielding and spacing, we are able to reduce it in our routing flow by shielding and spacing. By shielding we decreased the crosstalk delay by 90 to almost 100 percent, and by increasing the space between wires of links by factors of 1.5x and 2x, we reduced the x-talk delay by 30-50 and 90-100 percent, respectively. Crosstalk can also be reduced in the standard flow. However, this is achieved by changing the spacing and trajectory of the nets, which leads to significant net attributes variation. Moreover, as in the normal flow the trajectories of the nets of links are very diverse, controlling cross-talk is more difficult than in our flow.

To see the effect of our routing methodology on link power consumption, we also compared the total net switching power of our method with that of the normal flow. Figure 41 compares the total net switching for all benchmarks. As it can be seen, our approach does not impose power overhead on the design. On the contrary, in most cases switching power is about 5 to 10 percent less than that of the normal flow, due to lower via count and less wire jogs. We also compared area. In all the benchmarks, our method does not impose area overhead, as we never had to change the size of floorplan. Finally, there are benefits on routing run-time as well. In our comparison experiments, the normal routing flow took from 3 to 5 times more run time than that of our bundled routing approach.

### 3.5.5 Summing up

This report illustrates a new approach for bundled routing of NoC links. Our specialized routing flow produces highly regular link routes, thus leading to much reduced intra-link delay variations. Moreover, our flow supports spacing and shielding to ensure crosstalk immunity at the link level. Delay matching and low crosstalk noise are required features for advanced GALS synchronization and low-swing signaling, which are critical in NoC deployment for aggressively scaled CMOS.
Process Variation Tolerance in GALS Systems

Figure 38. Average and Maximum of different characteristics of all 5 test cases.
Figure 39. Crosstalk delay (ns) comparison in shielding.

Figure 40. Crosstalk delay (ns) comparison in spacing.

Figure 41. Net switching power (uW) comparison for all test cases.
3.6 Variability Compensation for Full-Swing vs Low-Swing On-Chip Communication.

Moving further down the design hierarchy, the need to control the delay of the wires building up a GALS NoC link arises. In fact, the physical routing process can balance wire delays by making them undergo the same layout conditions. However, process variations induce an uncertainty on actual circuit parameters that cannot be completely eliminated by careful physical routing. In particular, while bundled routing can make the impact of systematic variations on the delay of the wires of a link homogeneous, it cannot avoid delay differentiation as an effect of random process variations. To stay on the safe side, it would be desirable to set up a variability compensation framework during post-silicon testing, where actual delays can be measured and brought back within nominal bounds.

Post-silicon tuning allows the adjustment of device characteristics after a die has been manufactured to compensate for the specific deviations that occurred on that particular die [TSC02,TSC03]. One of the methods utilizes the transistor body effect to change transistor threshold voltage by applying an adaptive body bias (ABB) to modulate performance and power [WAN95,TSC02]. The other method for post-silicon tuning is to adjust the supply voltage (ASV) to trade performance with power, thus achieving a similar effect to ABB in spite of the different physical mechanism, implementation overhead and trade-off curves.

The effectiveness of ABB and ASV in reducing variability has been assessed and compared mainly on combinational logic circuits [CHE03], key elements of microprocessor critical paths [TSC03] and ring oscillators [MEI04], sometimes achieving counterintuitive and even conflicting conclusions.

The reason for this is that the effectiveness of ASV and ABB can not be generically assessed, but it has to be referred to the variance of a specific manufacturing process and to the performance and power tuning requirements of the design at hand. With the advent of multicore integrated systems, the assessment of post-silicon variability compensation techniques cannot be limited to the traditional testbenches of past research any more, such as combinational logic circuits or even microprocessor circuit sub-blocks. In fact, the new architecture trend requires long (global) interconnects for the connection of system-level blocks with each other. Unfortunately, physical properties of these on-chip interconnects are not scaling well with feature sizes, and they are becoming a key limiting factor for performance, reliability and timing closure of the whole system. A common practice is to overcome the effects of interconnect reverse scaling by means of circuit-level techniques, so that on-chip interconnects cannot be viewed as simple on-chip wires any more, but rather as communication channels including complex drivers and receivers [DOB08,JOO08]. Analyzing the impact of process parameter variations on the performance and reliability of these communication channels and exploring effective means for their compensation is a key design issue.

The relative effectiveness of ABB and ASV in this domain may greatly depend on the specific circuit implementation of the communication channels. A traditional design technique for long links consists of inserting equally spaced CMOS repeaters to deal with resistive loss along the wire. However, with the increase in number and density of the wires with each new technology, interconnect area and power are severely impacted [SYL99]. The most effective technique for global interconnects to achieve significant power savings and energy-delay efficiency is to reduce the voltage swing of the signal on the wire [ZHA00] and, possibly, to avoid the use of repeater stages, like in [KAN06].
On the other hand, low-swing signaling reduces noise immunity and poses non-trivial circuit design challenges. Many previous works in the open literature, like [BER02], compare power, area and delay of full-swing vs low-swing communication links. The novel contribution of this report is to compare the two signaling schemes from the viewpoint of their robustness to process variations. We distinguish between an inherent robustness, associated with the characteristics of the specific circuits building up the communication channels, and the robustness achieved as an effect of variability compensation.

The different circuit properties of full-swing and low-swing channels also determine an increased or decreased sensitivity of their performance to the different compensation mechanisms. Therefore, knowledge of the delay tuning range of ABB and ASV does not suffice to discriminate between them, since other effects need to be taken into account. First, for a given process variation scenario, the amount of induced delay variability is circuit-dependent, therefore making even the weakest (and typically most power saving) tuning mechanisms attractive for the most robust channels. Second, the sensitivity of channel performance to that of specific critical sub-blocks may be exploited to amplify the tuning capability of a variability compensation technique.

This work considers the compensation efficiency - cost trade-off by evaluating local circuit-level costs incurred by the compensation mechanisms. In practice, the power overhead of the compensated communication channels is considered, caused by the modified supply or body voltages. Other system-level costs, associated with the availability of multiple biasing voltages or their distribution across the chip, are not considered here and are left for future work. This report intends to identify the most promising compensation technique for each kind of communication channel and variability scenario, so to justify an effort for its system-level realization later on.

In our study, the effectiveness of variability compensation techniques when applied to on-chip links is assessed in two steps. At first, the inherent effectiveness of the compensation mechanisms for the channel at hand is investigated. Later, it will be analysed how such an effectiveness is impacted by layout effects in real life designs, especially crosstalk. Our objective here is to investigate the interaction between crosstalk effects and the behaviour of the compensation mechanisms. Our findings apply to generic point-to-point on-chip communication channels. Without lack of generality, given the emerging role of networks-on-chip (NoCs) as reference interconnect fabrics for MPSoC platforms [HAT07], we selected the links used for switch-to-switch connectivity in NoCs as a reference case study throughout this report. Wires of a GALS link are a subset of the considered case study, where the emphasis is on the electrical-level implementation of variability-robust links and not on the synchronization mechanism, which is a level of abstraction above. A parametric injection of process parameter variations was used to test the robustness of the signaling schemes/compensation mechanisms combinations to increasingly severe variability scenarios.
3.6.1 Communication channel design

We at first present the design of the point-to-point communication channels that will be assessed later on in terms of robustness to process variations and suitability for traditional post-silicon compensation techniques. Without lack of generality, we restrict our analysis to an intermediate layer wire with a length of 2mm, which is already the typical length of a switch-to-switch link in a regular network-on-chip architecture [HAT07].

Inserting repeaters to minimize delay of a wire is effective only when the wire is at least twice as long as the critical length of the technology and of the specific routing layer. In our target 65nm technology, a 2mm wire falls below this threshold and the choice is therefore for an unrepeated interconnect. Even for longer links, solid network-on-chip implementation works like [KAN06] suggest the use of unrepeated wires for the point-to-point communication links between switches, unlike other scenarios where high-fanout nets are required. To the limit, link pipelining can be used to break long timing paths. Following these indications, this report assumes the use of unrepeated wires for network-on-chip communication. We model the on-chip wire in HSPICE with a $\pi$3 distributed RC model. An STMicroelectronics technology library is used for the experimental framework.

At first, we assume that cross-coupling capacitance is tackled by means of physical-level techniques such as shielding or proper wire spacing, therefore no crosstalk effect is modeled at this time. The interaction between crosstalk and variability compensation will be studied later on in this report.

Figure 42. a) CMOS full-swing interconnect. b) Low-swing interconnect. c) PDIFF low-swing receiver from [ZHA00]. d) Optimized PDIFF low-swing receiver.
The reference link architecture uses a 1V full-swing signaling (Figure 42a). The driver consists of a (minimum sized) library flip-flop and a chain of buffers sized based on the exponential horn methodology for minimum delay. The receiver is yet another library flip-flop. The alternative communication scheme is the low-swing pseudo-differential (PDIFF) interconnect architecture reported in Figure 42b. The voltage swing is chosen to be 200mV. The basic circuit is taken from [ZHA00]. The driver is an NMOS-only push-pull driver which allows the use of very low power supplies and a quadratic energy reduction as a function of the voltage reference/swing $V_{ref}$. The receiver is still clocked but requires the voltage reference as an additional input. The original receiver circuit proposed in [ZHA00] is the clocked sense amplifier followed by a static latch illustrated in Figure 42c. This pseudo-differential scheme uses single wire per bit while still retaining most advantages of differential amplifiers such as low input offset and good sensitivity. The major reliability degradation may come from the local device mismatch between the double input transistor pairs and from the variation between distant references of the driver and the receiver. In contrast, receiver operation is largely insensitive to $V_{dd}$ supply noise, as opposed to other schemes. This was the basic motivation for selecting this scheme from [ZHA00].

However, we apply some improvements to this receiver, ending up with the circuit in Figure 42.d. First, PMOS transistor P6 in Figure 42.c has the task of equalizing the connected nodes, however it remains active even after the initialization, thus slowing down node transients. Moreover, it is not very conductive when the connected nodes reach an initialization value approaching its voltage threshold. In Figure 42.d it has been replaced by an NMOS transistor driven by the clock, thus achieving a better equalization and a faster node transition. Second, although the NOR static latch in Figure 42.c appears to be symmetric, it features uneven 0-to-1 and 1-to-0 switching times. Balancing rise and fall times makes the circuit actually asymmetric. The solution in Figure 42.d allows an easier balancing of these times while keeping the cross-coupled inverter pair fully symmetric: the outputs of the pseudo-differential receiver in fact directly drive the transistors (dis-)charging the flip-flop output capacitance, while the cross-coupled inverter pair keeps the sampled values. Output capacitance for the differential signal was tuned to be the same for POUT and POUTN signals. As a side effect, the flip-flop in Figure 42.d turns out to scale better from a performance viewpoint and enables higher operating frequencies for a comparable area than that of Figure 42.c.

Transistor sizing for the low-swing communication channel is done to keep the same (maximum) performance of the full-swing interconnect (1.68 GHz): driver sizing is used to achieve the same link propagation delay, while receiver and static latch sizing is used to enforce the same clock propagation delay, so that the next logic stage fed by the communication channel is impacted in the same way. In particular, the technology library constraints for such propagation time have been enforced. Our low-swing channel consumes almost 5x less power than the full-swing one, confirming its power efficiency. As regards area, the low-swing channel has a negligible 1% increase in area.

### 3.6.2 Inherent robustness to process variations

The first objective of this report is to compare the inherent robustness of full-swing and PDIFF low-swing signaling schemes to process variations, while compensation techniques will be addressed in the next sub-sections.

Our focus is on within-die variations, which happen at the length scale of a die, and that can be further divided into two contributors: systematic and random. Systematic variations can be predicted prior to fabrication and exhibit space locality. In contrast, random variations are due to the inherent unpredictability of the semiconductor technology itself. In our tests, we inject
effective gate length variations, which have implications on the threshold voltage as well, as computed by the HSPICE device models. We ignore variations in wires in this experiment.

Figure 43 shows the sensitivity of the signalling schemes to an increasing amount of systematic variations. The sensitivity is measured as the variation-induced deviation of the clock propagation time of the receiver from the nominal value. The propagation time goes from the clock sampling edge to the 50% voltage swing of the receiver output, and its nominal value is the same for both full- and low-swing channels, since they were designed to impact the next stage of the design in the same way. Systematic variations have been applied selectively to the transmitter, to the receiver and to the whole channel, so the bars in Figure 43 should be read pairwise.

It can be clearly observed that low-swing signaling proves a far more robust scheme to systematic variations. By restricting the analysis to the full-swing channel, its transmitter turns out to be the weak point of this scheme. The reason lies in the high sensitivity of the library flip-flop (i.e., the receiver) to the settling time of its input signal. This latter significantly deviates from nominal conditions when systematic variations affect the transmitter, and this explains the large degradation of the whole full-swing channel performance. In contrast, the receiver seems much more robust, and variations affecting the whole channel introduce only an incremental degradation with respect to the one caused by the transmitter.

The only exception occurs for channel-wide 5% systematic variations, where nominal delay is degraded by 90% (height of the last bar for full-swing is truncated to preserve the scale). This

Figure 43. Sensitivity to systematic variations.
is much more than one could expect by looking at the transmitter-degraded case, but this is due to the fact that we are working close to the point where full-swing channel operation fails: in this region, delay is highly sensitive to process parameter variations.

The opposite holds for the low-swing channel. The PDIFF receiver does a good job in providing a noise margin to the perturbations of its input signal induced by systematic variations in the transmitter. However, when variations affect directly the receiver, the PDIFF scheme suffers from increased switching delay. Clock propagation delay variations are much smaller for low-swing channels with respect to the full-swing ones anyway, and these latter might more easily induce the following stage in the design to fail, since it may be impossible to leave a 90% performance degradation margin for 5% systematic variations. We detected a failure of the full-swing channel when the transmitter is affected by 6/7% variations (tolerating a maximum propagation delay degradation by 90%), while the low-swing channel can keep working also under 70% systematic variations affecting both transmitter and receiver, after that the channel fails. At that time, however, propagation delay is degraded by 40%. The sensitivity of the channels under test to random variations ($3\sigma/\mu=15\%$) is illustrated in Figure 44.

![Figure 44. Sensitivity to random variations.](image)

Delay variability is similar in the two cases, with a slightly more tightened distribution for the low-swing channel. Again, we found the transmitter to be the most critical part of the full-swing channel, while the receiver is obviously the weak point of the low-swing channel. In fact, its pseudo-differential behaviour makes it very sensitive to random process variations, although we found only a negligible amount of malfunctioning channels with $3\sigma/\mu$ lower than 20%. This indicates that under such variations, the unbalancing of the differential branches remains within the noise margin of the receiver and correct 1/0 sampling takes place in due time. Delay variations pointed out in Figure 43 and Figure 44 indicate that compensation is apparently
more challenging in full-swing channels, though the effectiveness of compensation depends not only on the delay spread, but also on the sensitivity of such delay to the compensation mechanism and to the interaction between the sub-blocks of the communication channel, as illustrated hereafter.

### 3.6.3 Post-silicon variability compensation

Next, we explore the effectiveness of ABB (and forward body bias, FBB, in particular) vs ASV in bringing channel instances slowed down by process variations back within nominal performance. Compensation is applied to both the driver and the receiver for channel-wide tuning, but also selectively to individual sub-circuits to capture sensitivity of channel performance to that of these sub-circuits and eventually come up with a better trade-off between compensation efficiency and cost.

Figure 45. Working samples after compensation of full-swing channels. x-axis indicates the channel circuits to which compensation was applied.

Figure 46. Working samples after compensation of PDIFF low-swing channels. x-axis indicates the channel circuits to which compensation was applied.
3.6.3.1 Experimental framework

Since our target 65nm manufacturing process does not provide a triple well, we apply forward body biasing only to PMOS transistors. Our analysis aims to capture whether this lower cost solution suffices for compensation purposes in on-chip communication channels. In addition, it is not possible to selectively apply ASV only to the receiver of a full-swing channel, since this would require a voltage level shifter which is not there. In contrast, such level shifter comes for free in a low-swing channel, which therefore allows PDIFF receiver selective compensation with ASV. FBB does not have any kind of constraints in any signaling scheme.

Our experiments encompass the compensation of a representative subset of variation scenarios. Similarly to [BON08], worst-case systematic variations of +5% of parameter nominal value are assumed and superimposed to random variations. For these latter, the $3\sigma/\mu$ of channel length distribution is varied from 10, 15 to 20%, thus giving rise to three scenarios featuring the same amount of worst-case systematic variations and an increasing parameter spread associated with random variations.

Systematic variations were applied to the whole channel but also selectively to the receiver and to the transmitter to account for place&route effects. In fact, transmitter and receiver might be far apart from each other, thus suffering from systematic variations to a different extent, or they might be placed close to each other. In this latter case, physical parameters of the whole communication channel would be skewed by the same amount. For lack of space, we hereafter report only this latter case and the differences (if any) with the other variation scenarios are discussed in the text. We also recall that random variations were always applied to the circuits of the whole channel.

Figure 47. Framework for assessing the effectiveness of variability compensation techniques.
Recently, advanced modeling frameworks have been proposed to propagate variation information from the transistor compact model up to the system level, offering a correlated view on yield, timing, dynamic and static energy. They also improve the traditional Monte Carlo statistical static timing analysis techniques by accounting for rare events in variability distributions. Since this report focuses on a relatively small yet critical amount of logic, we developed an ad-hoc and simplified methodology based on Monte Carlo analysis to study the impact of systematic and random variability and how effectively it can be compensated. For each signaling scheme, variation scenario and compensation technique, we perform Monte Carlo simulations with a statistically significant sample set. Each Monte Carlo run (i.e., a channel instance with different random variability injections) goes through the compensation methodology illustrated in Figure 47.

At first, we check for nominal performance requirements. If met, a new instance is analyzed. If not, a compensation step is applied. In practice, if FBB is under test, an incremental reduction step of the body bias is applied so to improve performance. Similarly, the supply voltage is increased when ASV is assessed. Decrements/Increments are applied with steps of 100 mV both for ASV and FBB. This choice stems from the conclusion of previous works [TSC02] and from considering realistic resolutions of low-cost voltage regulators.

After the compensation step, performance is re-evaluated and eventually an additional compensation step is applied. The process completes when nominal performance is finally met OR when the voltage range limit is reached: 500 mV for forward body bias (to avoid turning on the source pn junction of transistors) and 200mV for ASV (for reliability and technology library constraints).

Effectiveness of a technique is expressed as the percentage of the sample set that can be brought back within nominal performance by the compensation technique under test. We denote those successfully compensated samples as working samples. Nominal performance means correct sampling at 1.68GHz, with clock propagation time constraints met at the output of the receivers. Moreover, the average power overhead for compensating channel instances with the highest power supply value (lowest PMOS body bias value) is measured, denoting power efficiency of the compensation techniques. For low-swing signaling, we also explore adaptive voltage swing as an additional and built-in compensation technique by raising the voltage swing in increments of 100mV.

In the first set of experiments $3\sigma/\mu$ is assumed to be 15%. When systematic and random variations are injected into the entire channel, we find almost no channel instances in the sample set working without compensation, both for full-swing and low-swing channels. So, in the experiments that follow, the entire sample set needs to be compensated.

3.6.3.2 Compensation efficiency in full-swing links

As can be observed from Figure 45, neither ASV nor FBB are able to restore functionality of all working samples by only acting upon the transmitter or (for FBB) the receiver. The compensation in this case would be totally ineffective. Variability can only be compensated by tuning all the circuits of the channel. In fact, performance of full-swing channels is highly sensitive to the interaction between the signal provided by the transmitter and the requirements imposed by the receiver on the timing and shape of this signal. Moreover, systematic variations (recall Figure 43) significantly impact both the transmitter and the receiver. As a consequence, an effective compensation can only be carried out by acting upon both modules at the same time.

However, while ASV requires a single voltage step to reach 100% working samples, FBB needs its entire voltage range to achieve the same objective. Anyway, the large variations...
taking place in full-swing channels can be successfully compensated by FBB in spite of its inherently weaker performance tuning capability. In practice, the sensitivity of channel performance to transmitter-receiver interaction was found to be an amplifying effect of FBB tuning capability.

The main difference between the compensation techniques lies however in their power efficiency. When ASV raises the supply voltage to 1.1V, the communication channel instances on average exhibit a 23% power overhead with respect to the variation-free scenario. In contrast, a 500mV forward body bias incurs only an average power overhead of 2.4%, almost negligible.

When we applied systematic variations only to the transmitter (flip-flop and driver), we observed that a selective tuning of the transmitter circuits only partially solved the problem. ASV could restore about 80% of the samples, while FBB about 60% by remaining in the voltage range limits. This indicates the impact of random variations, which require a tuning of the receiver as well to restore 100% working samples. The situation is even worse when only the receiver is affected by systematic variations: while no selective tuning of the flip-flop is feasible with ASV due to a lack of a voltage level shifter, only 20% of working samples were achieved by selective FBB. Again, the only option was to tune the entire channel, finding again the same power efficiency gap between FBB and ASV.

3.6.3.3 Compensation efficiency in low-swing links
Quite different considerations hold for variability compensation in low-swing channels. This time, ASV can be selectively applied to the receiver since the level shifter is built-in in the signaling scheme. Figure 46 clearly shows that a selective tuning of the receiver with both ASV and FBB reaches a high percentage of working samples. With just one voltage increment step applied to the output flip-flop, ASV can restore performance of the entire sample set. More interestingly, the average power overhead is limited to 8.5%, much lower than in a full-swing channel.

In low-swing channels, the transmitter is marginally impacted by systematic variations. At the same time, receiver performance is much less sensitive to the perturbations of its input signal than in full-swing channels. Therefore, acting upon the receiver proves an effective compensation method.

Unfortunately, FBB cannot reach 100% working samples with a selective compensation at the receiver, and neither a channel-wide compensation can (90% is the best result achieved with a 500mV FBB). This is essentially due to the weak performance tuning knob represented by FBB, which is not boosted by any circuit level property in this case (for instance, no high sensitivity of channel performance to transmitter-receiver interaction). The average power overhead incurred for the worst-case FBB compensation is around 6%, comparable with that of ASV.

Considering the cases where systematic process variations affect only the transmitter or the receiver, we found that FBB is not able again to reach 100% of working samples (best coverage is 90%). ASV instead works effectively. However, in all cases and for both ASV and FBB, selective compensation at the receiver turns out to be as effective as full channel compensation. Power overhead for ASV is around 7 and 8%, while for FBB is around 3%.

Figure 46 also shows the efficiency of an intuitive compensation technique which stems from the possibility to tune the voltage swing in the low-swing channel. Although intuitive, this technique proves highly ineffective to restore channel performance. By increasing the voltage swing from 200mV to 400mV, only 50% of the non-working samples can be saved.
Interestingly, further increasing the swing proves useless, and no further improvements can be achieved, thus spending power uselessly. This is due to the fact that compensating process variations is not just an issue of speeding up signal propagation across the link, but to restore functionality at the transmitter, at the receiver and their correct interaction. Only when the transmitter is impacted by systematic variations while the receiver is not, then speeding up the link with a swing of 400mV achieves 82% working samples. Compensating receiver variability proves more difficult (about 60% working samples). Another argument against reference voltage scaling is power. The measured average power overhead for the worst case compensations (those at 400mV) amounts to a significant 46%. This confirms the results of the work in [MED08], showing that using the voltage swing to speed up a low-swing link is highly power inefficient.

3.6.3.4 **Role of random variations**

When we repeated the experiments with a $3\sigma/\mu=10\%$ and below, the minor role played by random variations translated into a better compensation efficiency of FBB in low-swing channels, since working samples were always close to 100%. The lower delay spread makes the worst-case compensation scenario affordable also for the tuning capability of FBB, so that this latter can be considered also for low-swing signaling as the impact of random variations decreases. Finally, $3\sigma/\mu$ was set to 20%. In this case, even for full-swing channels FBB could not bring all samples within nominal performance bounds, although still achieving around 95% working samples. Interestingly, in low-swing channels the effectiveness of FBB was as low as 70% working samples.

3.6.4 **Variability Compensation with crosstalk**

Robustness to delay variability and its compensation have been evaluated for link models ignoring crosstalk effects so far. However, as technology scales down to the nanoscale regime, coupling capacitance plays a dominant role in determining signal integrity. Moreover, this work also points out the implications of crosstalk effects on the effectiveness of variability compensation with ASV and FBB.

In order to capture realistic layout effects of on-chip interconnects, we synthesized a 32-bit unrepeated link with Synopsys Physical Compiler on the target technology library. Placement and routing were performed by Cadence SoC Encounter. Transmitter and receiver were placed in two fences 2mm far apart. This work considers networks-on-chip as an experimental case study, therefore STALL and VALID flow control wires were routed together with the 32 bit flit. The importance of control wires for communication reliability is such that they might be operated at full swing even though the flit is inferred with low-swing links. In this case, capacitive coupling between full- and low-swing wires within the same channel might be a serious concern. We analysed both cases: a fully low-swing link and an hybrid one. Finally, the clock tree was synthesized. Again, its coupling with the flit lines needs to be carefully monitored.

We then extracted the parasitic resistance and capacitance with the STAR RCXT tool, enabling the extraction of coupling capacitance as well. The result was the generation of an HSPICE link netlist (modeling parasitics), which was connected with both the full-swing and low-swing transmitters and receivers designed in Section 3.4.1. The two new link models are equivalent to the one analysed so far, except for the inclusion of coupling capacitance and for the account of realistic routing constraints.
3.6.4.1 Signal integrity

Figure 48 reports the communication channel routed by the SoCEncounter tool. Clearly, the routing pattern does not consist of fully parallel wires as often assumed in abstract link analyses, but encompasses some wire crossings and metal layer switchings. This implies a non-trivial crosstalk interaction among the wires and a hardly controllable signal integrity.

In fact, if we look at the capacitance breakdown of the wire named flit_28 (28th wire of the 34 bit link), we can clearly observe that the cross-coupling capacitance with the clock signal is 30.5% of the whole flit_28 line capacitance (Figure 49). In these conditions, the low-swing receiver even fails to correctly sample flit_28 where the full-swing one instead succeeds, as Figure 50 illustrates. The clock signal samples input data (second row), which is then output by the driver (third row - full swing). Monitoring the corresponding input of the receiver (fourth row - full swing, third row low-swing) clearly indicates a relevant cross-coupling effect with the clock signal, resulting in the sampling failure (last row - all schemes) of the low-swing receiver. Although the full-swing channel still works, its maximum speed when comparing the clock enabled with the disabled case (the clock is in this case provided by means of a simulation trick, not via the synthesized clock wire) degrades by 16.5%. In order to characterize maximum link speed, we applied an input pattern to the links where every bit switches in the same direction (0/1 → 1/0) except one, which does the opposite transition (1/0 → 0/1) putting its driver in the worst (dis-)charging condition for the switched cross-coupled capacitance. The opposite transition pattern is selectively applied to every wire, thus looking for the worst case performance across the entire link.
More in general, whenever in the same communication channel full-swing wires interact with low-swing ones, the signal integrity concern for these latter arises. In order to further prove this, we designed another low-swing link where the two flow control wires (STALL/GO and VALID) were operated at full-swing. The results showed a loss in maximum performance of 31.7% with respect to a fully low-swing link. The critical path was located across the wire denoted as \textit{flit}_7, whose capacitance breakdown is illustrated in Figure 51. Clearly, the cross coupling capacitance with the STALL control wire accounts for 43.5% of total line capacitance, thus leading to a significant performance degradation of the communication channel.

The key take-away here is that in order to materialize the power efficiency of low-swing signaling, reliability concerns caused by coupling with the clock signal and/or with other full-swing control wires need to be tackled by enforcing new routing constraints (e.g., wire extra spacing or shielding). This consideration is of the utmost importance for source synchronous communication schemes, where the clock signal has to be trasmitted together with data signals while experiencing the same routing delay. This report has already illustrated the work performed in this domain.
3.6.4.2 Compensating crosstalk-affected links

We repeated the variability compensation tests of the previous section with the crosstalk-augmented wire models, so to assess how crosstalk interferes with the compensation tasks of ASV and FBB. Based on the results of the previous analysis, these mechanisms are applied to the whole channel in full-swing links, while in low-swing links ASV is selectively applied to the receiver and FBB again to the whole channel. Assuming our routing requirements derived in
previous section are met, we consider links clocked by a simulation clock (to avoid considering the destructive crosstalk induced by the routed clock signal) and, in the low-swing channel, the low-swing operation even for the flow control wires (corresponding, in real-life layouts, to an increased spacing for these lines or to a shield between them and low-swing ones).

The same variability injection is operated like in previous section. The only difference is that this time the entire 34-bit link is compensated, not just a single wire, since interaction between wires is of interest to this experiment.

Even in the presence of crosstalk, the capability of ASV to restore 100% of the sample set with only one voltage increment for the full-swing link remains unchanged, as suggested by Figure 52. The average power overhead for compensation is 22.4%, similar to the overhead required without crosstalk. Whereas the low-swing link requires two ASV increment steps to restore nominal performance in all cases, while requiring 19.2% power overhead (it was around 8% without crosstalk). These results clearly indicate that crosstalk effects make variability compensation in low-swing channels more expensive for ASV. However, the power overhead for ASV to compensate a low-swing channel is lower than a full-swing one, indicating that whenever ASV is the only available compensation mechanism, low-swing signaling is more amenable to it.

Thereafter we ran the tests with the ABB performance tuning technique. In this case the presence of crosstalk affected the compensation of both full- and low-swing links, in fact only a statistically irrelevant percentage of the sample set was brought back to the nominal performance in both cases. The limitation posed by crosstalk on the tuning capability of ABB is extremely severe. We found the restricted applicability of ABB to p-MOS transistors only (due to the single well technology) a very limiting factor for this scenario. In fact, by artificially

---

**Figure 53. Working samples after p-mos n-mos ABB compensation with random, systematic variations and crosstalk.**
extending compensation to n-MOS transistors as well, the results of Figure 53 were obtained. For the full-swing channel, the entire sample set can be successfully restored at minimum power overhead, while for the low-swing channel ABB proves an even more ineffective alternative than in the absence of crosstalk.

3.6.4.3 **Summing up**

This part of the report explores the effectiveness of ASV and FBB as post-silicon variability compensation techniques for on-chip point-to-point communication channels. Since the focus is on the interaction between the compensation mechanism and the signalling scheme, we opt for test case implementations on an industrial 65nm technology library instead of the commonly used predictive technology models for the sake of increased accuracy. Process parameter variations are then injected in a parametric way to capture robustness of signalling schemes/compensation mechanisms combinations to increasingly severe variability scenarios.

Our work shows that FBB is effective for tuning performance of full-swing channels with minimum power overhead. In contrast, when applied to low-swing channels, FBB proves not capable of compensating all variation patterns, since its limited performance tuning capability is not amplified by any circuit property. On the other hand, ASV can exploit the built-in voltage level shifter in low-swing channels and achieve an effective and low cost selective compensation.

Crosstalk effects do not change the best compensation technique for each scenario, but make compensation more expensive. For full-swing channels, ABB remains the technique of choice for its minimum power impact, but coupling capacitance makes the tuning of both p-MOS and n-MOS transistors necessary. For low-swing links, ASV selectively applied at the receiver seems the best trade-off between compensation power overhead and yield both with and without crosstalk.

The results of this report point out the superior robustness of low-swing channels to process variations. After considering a realistic range of systematic and random WID process variations, it is evident that low-swing channels

- can better cope with systematic variations (lower delay deviations and functional correctness guaranteed over a wider range of variations),
- feature a lower delay spread under random variations.

After exploring all the possible countermeasures based on FBB and ASV, it can be also derived that low-swing channels can be compensated with success against delay variability at a low power cost. These features add up to the reference characteristic of low-swing channels, which is their inherent low power consumption. The results of this report have immediate applicability to the physical wires of GALS links, although future work will have to develop an automated way of distributing different voltages across the NoC and of inserting cells for low swing signalling into an industrial technology library and synthesis toolflow. The objective of this study was to point out which direction to go in light of the process variation challenges posed by nanoscale technologies.
4 VARIABILITY AWARE TECHNIQUE FOR DIGITAL LOGIC

Although the main focus of this report and of deliverable D22 of the Galaxy project is on design techniques for variability robustness in GALS NoCs, the scope of task 7.3 was broader and aimed at the development of high-level methods in GALS system partitioning to improve the system performance and reliability under large process variations. In this direction, we devote the last section of this report to a variability – aware design technique for digital logic. Although applied to a GALSified functional block as an early validation test, the technique can be easily extended to the digital logic inside on-chip networks, such as the switching fabrics.

4.1 MOTIVATION

The scaling increases die-to-die (inter-die, global) and within-die (intra-die, local) parameter variations in the manufacturing process which are found to heavily impact the yield of low voltage digital circuits. Die-to-die variations act globally on the entire chip or on functional blocks, so that each device on one chip or in one block shows the same deviation. Inter-chip or inter-block variations can be caused by systematic effects like process gradients over the wafer [STO96] with typical distances in the range of functional block sizes or above. Variations of the gate oxide thickness can be regarded as global variations. Sets of worst-case and best-case parameters are used during design verifications to mitigate the impact of global variations. Effects of within-die variations are becoming more and more prominent with scaling and they have a direct influence on local gate delay variations. Numerous random factors such as statistical deviations of the doping concentration and imprecision of lithography, lead to more pronounced delay variations for minimum transistor sizes [EIS97, BOW00]. These factors are intrinsic since they cannot be eliminated by external control of conventional manufacturing. The increase of the path delay variations for smaller device dimensions and reduced supply voltages is becoming more prominent with scaling. Circuits with a large number of critical paths and low logic depth are most sensitive to uncorrelated gate delay variations [BOW00].

While the impact of global variations can be mitigated with techniques such as adaptive body biasing and dynamic voltage scaling [MAR02], these techniques cannot reduce the impact of local variations. In GALS designs minimization of local variations becomes an imperative, since local variations are heavily impacting the performance of local synchronous island (LSI). The operating frequency of each LSI is determined by the slowest critical path in that island. Moreover, the impact of local variations is increasing compared to global variations and the application of corner design rules might not be sufficient. In Table 5 the values for die-to-die (D2D) and within-die (WID) variation for state-of-the-art 65nm process are given.

| Table 5 Normalized standard deviation of gate and critical path for WID and D2D variations and different critical path lengths (n<sub>cp</sub>) |
|---|---|---|---|---|
| (%) | n<sub>cp</sub> =1 | n<sub>cp</sub> =6 | n<sub>cp</sub> =8 | n<sub>cp</sub> =10 |
| WID | 11.48 | 5.44 | 4.78 | 4.44 |
| D2D | 7.96 | 7.94 | 8.12 | 8.22 |
| n<sub>cp</sub> =12 | n<sub>cp</sub> =16 | n<sub>cp</sub> =20 | n<sub>cp</sub> =24 |
| WID | 4.08 | 3.62 | 3.32 | 2.98 |
| D2D | 8.29 | 8.38 | 8.43 | 8.46 |
An almost constant value for the standard deviation of D2D variations for different critical path lengths confirms that D2D variations can be assumed as purely systematic. On the other hand, WID variations are mostly random.

The mean delay has been taken as a nominal critical path delay $T_{cp,nom}$. The WID and D2D nominal critical path standard deviations are $\sigma_{T_{cp,WID}}$ and $\sigma_{T_{cp,D2D}}$ respectively. The critical path delay probability density functions (PDF) resulting from WID and D2D parameter variations are modelled as normal distributions [CAO07, BOW02, LE04] in (1) and (2), respectively.

$$f_{T_{cp,nom,WID}}(T_{max}) = N(T_{cp,nom}, \sigma_{T_{cp,WID}}^2)$$  \hspace{1cm} (1)

$$f_{T_{cp,nom,D2D}}(T_{max}) = N(T_{cp,nom}, \sigma_{T_{cp,D2D}}^2)$$  \hspace{1cm} (2)

### 4.2 Impact of Local Variations on the Maximum Critical Path Delay Distribution

A large LSI contains many critical paths, all of which must satisfy the worst-case delay constraint [EIS97, BOW00, BOW02]. Assuming a number $N_{cp}$ of independent critical paths for the LSI, the maximum critical path delay PDF is then calculated following [BOW02] as

$$f_{chip,WID}(T_{max}) = \frac{dF_{chip,WID}(T_{max})}{dT_{max}} = N_{cp} f_{T_{cp,nom,WID}}(T_{max}) \left( F_{T_{cp,nom,WID}}(T_{max}) \right)^{N_{cp} - 1}$$  \hspace{1cm} (3)

where $F_{T_{cp,nom,WID}}$ is the critical path delay cumulative distribution function (CDF).

The LSI maximum critical path delay PDF is illustrated in Figure 54.

![Figure 54. WID maximum critical path delay distribution for different values of N_{cp} and D2D critical path delay distribution.](image-url)
By increasing $N_{cp}$,

- the mean of $f_{chip,WID}$ increases, since the slowest critical path limits LSI overall performance and the probability of a longer cycle time increases,

- the standard deviation of $f_{chip,WID}$ decreases and becomes relatively small compared to standard deviation of $f_{Tcp,nom,WID}$, e.g. for $N_{cp}=10^4$, $\sigma_{chip,WID}=0.3\sigma_{Tcp,nom,WID}$, and $\sigma_{chip,WID}=0.12\sigma_{Tcp,nom,d2d}$,

- $f_{chip,WID}$ becomes less sensitive to further increase of $N_{cp}$; qualitatively, this means that increase of $N_{cp}$ from 1 to 10 has a greater effect on the mean and variance of the WID distribution than increase from $10^3$ to $10^4$.

As the number of transistors per chip increases and the average gate delay is reduced, $N_{cp}$ is expected to increase for each technology generation [BOW02], and further reduce the sensitivity of the maximum critical delay to $N_{cp}$.

4.3 MINIMIZATION OF THE IMPACT OF LOCAL DELAY VARIATIONS

To account for the impact of local variations in LSI, a novel technique that uses selective redundancy is developed. The general principle consists in replicating $R$ times a critical path, and evaluating critical path outputs using a function that senses the fastest out of $R$ replicated paths. In this way, not only the standard deviation but also the mean value of the critical path delay is reduced as demonstrated further in Table 6.

The proposed technique can be also used to support recovering correct operation disrupted by other sources of variations such as unexpected voltage drops in the power supply network, local temperature fluctuations, crosscoupling noise, etc.

The circuit that performs the evaluation of the fastest input path (fastest path evaluator - FPE) is shown in Figure 55. It senses the path where the first signal change occurs. Depending on the previous signal state input signals are ANDed or ORed sensing the first falling or raising edge respectively. Since the information about the previous signal state is necessary in the fastest path evaluation, it is a natural idea to integrate this circuit into existing flip-flops. A key requirement for the integrated fastest path evaluator (FPE) flip-flop is that the delay overhead compared to the standard flip-flop should be minimal.

![Figure 55. Fastest path evaluator and standard flip-flop.](image)

Several methods are applied to reduce the delay overhead of the FPE flip-flop. The master latch is duplicated and its input inverter is replaced with AND and OR gates. The multiplexer at
the input of the flip-flop causes a significant delay overhead, and is therefore moved to replace
the output inverter of the master latch, as shown Figure 56. The FPE flip-flop is designed in
the state-of-the-art 65nm technology and the delay overhead compared to the equivalent type
of standard library flip-flop is less than 7%.

Figure 56. Schematic of the reduced overhead FPE flip-flop.

The improvement of the relative mean value \( \frac{\mu_{FPE}}{\mu_{\text{cp}}} \) and relative standard deviation \( \frac{\sigma_{FPE}}{\mu_{\text{cp}}} \) of the
critical path delay is shown in Table 6 for different critical path lengths \( n_{cp} \) and different
redundancy factors (\( R \)). The values are acquired from Monte Carlo SPICE simulations. The
additional delay introduced with the FPE flip-flop is also accounted. The reduction in the mean
value of delay ranges from 0.85%-2.38% and for standard deviation ranges from 17%-25% for
\( R=2 \) and \( R=3 \), respectively. The variance reduction from \( R=2 \) to \( R=3 \) is not as significant as
the reduction between no redundancy and \( R=2 \).

<table>
<thead>
<tr>
<th>( n_{cp} )</th>
<th>( % )</th>
<th>( R=2 )</th>
<th>( R=3 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>6</td>
<td>5.44</td>
<td>0.85</td>
<td>4.50</td>
</tr>
<tr>
<td>8</td>
<td>4.78</td>
<td>0.88</td>
<td>3.96</td>
</tr>
<tr>
<td>10</td>
<td>4.44</td>
<td>0.97</td>
<td>3.67</td>
</tr>
<tr>
<td>12</td>
<td>4.08</td>
<td>0.97</td>
<td>3.37</td>
</tr>
<tr>
<td>16</td>
<td>3.62</td>
<td>0.99</td>
<td>2.99</td>
</tr>
<tr>
<td>20</td>
<td>3.32</td>
<td>1.00</td>
<td>2.74</td>
</tr>
<tr>
<td>24</td>
<td>2.98</td>
<td>0.94</td>
<td>2.46</td>
</tr>
</tbody>
</table>

Table 6 Relative mean value and standard deviation of the critical path delay.
4.4 Optimization Results of Variations Minimization

The effect of reducing the mean delay and standard deviation of critical paths by replicating a critical path and inserting an evaluation block is exploitable only when the delay reduction is large enough to compensate for the area/power overhead introduced by the critical path replication.

An architecture that is optimized in the following in terms of delay variations is an AES encryption core [AES01]. The design is partitioned into three local synchronous islands: encryption module, key generation module and control module. Each of these three islands has been synthesized for the maximum operating frequency. Encryption module as the critical one in terms of speed has been additionally optimized for variations minimization using the presented technique of fastest path evaluation.

The encryption module has $N_{cp}=200$ independent critical paths (with slack smaller than 1% of the maximum operating frequency), with typical critical path length of $n_{cp}=16$. The distribution for a normalized maximum critical path delay is shown in Figure 57, considering cases with and without optimization. The optimization is performed with redundancy factors of 2 and 3.

![Figure 57. Distribution for a normalized maximum critical path delay](image)

There is an important take-away from Figure 57. It clearly shows reduction in the mean value of the maximum critical path delay which mostly benefits from the reduced mean value of each critical path. Moreover, a reduced standard deviation of each critical path also reduces the mean value of the maximum critical path delay. The improvement in the mean value of the maximum critical path delay is:

- 3.3% and 5.4% compared to the default case (without optimization) for $R=2$ and $R=3$, respectively.

There is a noticeable reduction in the standard deviation of the maximum critical path delay:

- 25.5% and 35.8% compared to the default case (without optimization) for $R=2$ and $R=3$, respectively.
WID variations determine the mean of the maximum critical path delay distribution while D2D variations determine the variance. With respect to this observation, any improvement in the mean value of the maximum critical path delay WID variations enables a direct improvement in the maximum critical path delay of any fabricated chip. A trade-off can be observed between the redundancy level involved, representing additional area/power, and the reduction of the maximum critical path delay which means increase in operating frequency. All the paths that have the delay within 1% of the slowest path are considered as critical. The total number of equivalent gates that are part of critical paths is 600 and that the whole encryption module has 23000 equivalent gates, replicating each critical path 2 and 3 times results in an overhead of 2.75% and 5.5% respectively. Replicated critical paths should be laid independently as separate paths during place&route. The technique should be applied only to time critical LSI’s in GALSified design.

The benefit of the selective redundancy-based technique to limit the impact of local delay variations has been demonstrated. Only critical segments (critical paths) are replicated and the viable improvement is achieved with the redundancy factor as low as $R=2$. This way, the proposed technique can optimally be used in the design of large synchronous digital systems. The inherent property of the technique is that the larger the variations are, the greater is the reduction in the maximum critical path delay, and therefore, the higher is the improvement in the maximum frequency the circuit can run.

The studies have shown that the technique can already be successfully applied for 65nm CMOS technology process where an optimal point has been shown to exist in the speed vs. area/power tradeoff. Even more advantage is expected for future nanoscale CMOS technologies such as 45nm and 32nm nodes. The technique can be especially beneficial in the low-power domain where reduced supply voltages lead to significantly increased local variation.
5 CONCLUSIONS

Manufacturing processes required to enable deep-nanometer technologies (65 nm and beyond) have to tackle significant challenges. Despite the best efforts of process engineers, large variations in device parameters will be unavoidable. In the extreme case, architectures of multi-core integrated systems will have to deal with manufacturing faults in order to sustain yield. Traditional synchronous design methodologies, that require the system to adhere to strict timing constraints over the entire chip area, suffer badly from wide ranging parameter variations. GALS systems may offer significant advantages with this respect, as they essentially divide a complex system into small, loosely coupled (if not independent) modules. In this new context, beyond limiting the performance implications of process variations in local modules, designers now have to devote special care to the design of process-variation tolerant synchronization interfaces. This report documents the development effort of GALS design methods to improve the system performance and reliability under large process variations. Although not limited to on-chip networks, most design techniques have been specifically conceived for them in light of the key role they play for global system connectivity.

This report moves from the assumption that mitigating process variations is not a task on burden of a single abstraction layer but requires a complementary course of action to be taken at each layer of the design hierarchy. In particular, this report delivers:

- **Variability-aware design techniques for the asynchronous links of a GALS NoC.** following the architecture template developed throughout the Galaxy project (and Workpackage 6 in particular). Such design techniques span several abstraction layers, namely:
  - **System-level techniques** to guarantee network survivability in the presence of link delay faults. For this purpose, it was showed that logic based distributed routing features better delay and area scalability of routing tables and features promising degrees of coverage of failure patterns in the network. Out of a comparison between logic based distributed routing mechanisms, LBDR with enhancements turns out to be the most promising approach in terms of coverage/area.
  - **A system-level methodology** to assess yield in the presence of process variations and thus testing the capability of routing mechanism/algorithm combinations to live with a wide range of failure patterns in the network.
  - **Architecture-level techniques** for GALS link and synchronization interface design. Essentially, a variability detector was illustrated and proved to be able to counter random process variations inducing routing skew between data and clock and shrinking the timing stability window of data in a source synchronous link. The detector was proved to be robust to the delay variability of its logic cells.
  - **Physical routing level techniques** for bundled routing implemented by means of scripts on top of the latest industrial routing tools. Our specialized routing flow produces highly regular link routes, thus leading to much reduced intra-link delay variations. Moreover, our flow supports spacing and shielding to ensure crosstalk immunity at the link level. Delay matching and low crosstalk noise are required features for advanced GALS synchronization and low-swing signaling, which are critical in NoC deployment for aggressively scaled CMOS.
  - **Circuit-level variability compensation techniques** for link delay variability. Our work shows that FBB is effective for tuning performance of full-swing
channels with minimum power overhead. In contrast, when applied to low-swing channels, FBB proves not capable of compensating all variation patterns, since its limited performance tuning capability is not amplified by any circuit property. On the other hand, ASV can exploit the built-in voltage level shifter in low-swing channels and achieve an effective and low cost selective compensation. Moreover, low swing channels have been proven to be inherently more robust to systematic variations and to feature a lower delay spread in the presence of random variations, although inherently more sensitive to crosstalk.

- **Variability-aware design techniques for the local islands of synchronicity of GALS systems.** In particular, a novel technique that uses selective redundancy was illustrated and validated, when put at work to reduce not only the standard deviation but also the mean value of critical path delays. Only critical segments (critical paths) are replicated and the viable improvement is achieved with the redundancy factor as low as \( R=2 \). This way, the proposed technique can optimally be used in the design of large synchronous digital systems. The inherent property of the technique is that the larger the variations are, the greater is the reduction in the maximum critical path delay, and therefore, the higher is the improvement in the maximum frequency the circuit can run. The technique can be especially beneficial in the low-power domain where reduced supply voltages lead to significantly increased local variation.