Radiation Hard ASICs and Opto-Electronics for the ATLAS Experiment

DISSERTATION

Presented in Partial Fulfillment of the Requirements for the Degree Doctor of Philosophy in the Graduate School of The Ohio State University

By

Dale Shane Smith, B.S., M.S.
Graduate Program in Electrical and Computer Engineering

The Ohio State University

2013

Dissertation Committee:
Dr. Steven Bibyk, Advisor
Dr. Waleed Khalil
Dr. Richard Kass
Abstract

Due to high radiation environments, commercially available electronics are often unsuitable for use in the instrumentation of high energy physics (HEP) experiments. Additionally, due to mechanical and space constraints, the size and materials used in these experiments often eliminate commercial devices from consideration. Given the facts above, physicists themselves lead the design and construction of most subsystems within their experiments. Because of the large scale of today’s HEP detectors, their construction would not be possible without the collaborative spirit of thousands of scientists and engineers. Included in the engineering required is the design of application specific integrated circuit (ASIC) chips. In attempts to conserve effort, designs of circuit blocks are frequently shared among collaborating institutes.

In this dissertation we present the design, verification, fabrication, testing, and qualification of three separate radiation hard ASIC chips as well as a novel fiber-optic transceiver board. Each of these components was developed to be used as custom high energy physics instrumentation. Two of the ASICs, the DORIC12 and VDC12, were designed to form the radiation hard chipset for the fiber-optic transceiver board. The other ASIC, the Hitbus chip, was developed for the ATLAS Diamond Beam Monitor (DBM) to provide the capability for the DBM detector to trigger its own readout. All
three ASICs require the use of radiation hardening by design techniques and a mixed-signal design flow. Intermixed in the development of the ASICs is the incorporation of shared intellectual property (IP) blocks. We report on the design and testing of the three ASICs and include insights gained through the process of reusing collaboratively shared designs. We present guidelines for effectively sharing, integrating, and verifying shared block designs. Moreover, we introduce a concise template to assist in the IP sharing process.
To Rachael and Dale
Acknowledgments

First and foremost, I want to thank my wife Rachael. Rachael, I could not have completed this work without your encouragement and support. I will never forget the sacrifices you made along the way. You are an incredible wife and mother (and editor)!

To my parents, Mike and Nancy, it would be impossible for me to thank you enough for all you have done for me. You encouraged my diverse interests and gave me a great childhood. To my sister, Michelle, though you are far away, you are always in my heart. To my extended family, I am particularly grateful for all the love and support I have received from you all.

I also would like to thank my employers in the Ohio State University Physics Department: K.K. Gan, Klaus Honscheid, Harris Kagan, and Richard Kass. Through your mentoring, guidance, and support I have learned more than I could have ever dreamed about electronics and how research is done. You were always patient when I made mistakes and always allowed me the chance to correct them. I am incredibly grateful for the time I have spent and the experience I have gained working for you.

Along these lines, thank you to all of the students, post-doctoral researchers, and staff in O.S.U. Physics that have accompanied me through this journey. I have been enriched by your diversity, intelligence, and camaraderie.

I want to recognize my friend and colleague Michael Ziolkowski. Through working with you I learned much about designing and debugging instrumentation. I have greatly benefited from witnessing your perseverance and willingness to miss sleep, two
of the main requirements for convincing an experiment to operate and take data. Additionally, I will never forget the kind compliments you have regularly given me.

My acknowledgments would not be complete without affirming gratitude for my advisor Dr. Steven Bibyk. I became fascinated with designing chips as an undergraduate through taking your courses. I strove to learn the subject and you were always complimentary of my work. You gave me the necessary confidence to pursue higher learning in the field and as a result I was able to formulate and complete this dissertation. I could not have reached this goal without your guidance, patience, and support.

I would like to express my appreciation to Dr. Waleed Khalil and Dr. Richard Kass for your service on both my dissertation and candidacy exam committees. Your suggestions and comments have been invaluable additions to this work and I thank you for kindly sacrificing your time to enhance it.

Finally, I want to give a big thank you to my daughter Dale. You amaze me every day. Your constant flow of reckless actions, brilliant statements, and loving hugs and kisses is simply incredible! You have introduced me to new and unexpected joys in life and reminded me of many that I had forgotten long ago. C'est tout!
Vita

June 1999 .......................................................B.S. Electronics Engineering Technology, Devry University, Columbus, Ohio
June 1999-present ..........................................Research Specialist, The Ohio State University
March 2005 ....................................................B.S. Electrical and Computer Engineering, The Ohio State University
August 2012 ...................................................M.S. Electrical and Computer Engineering, The Ohio State University

Publications

Research Publications


D. S. Smith et al., "Development of the Hitbus Chip Platform for the ATLAS DBM Detector at CERN," in 56th International Midwest Symposium on Circuits and systems, Columbus, Manuscript submitted for publication.

viii
A. Studen et al., "A laboratory PET scanner with silicon detectors segmented to 1 mm³ detection cells," in *2012 IEEE Nuclear Science Symposium and Medical Imaging Conference Record (NSS/MIC)*, Anaheim, 2012.


**Fields of Study**

Major Field: Electrical and Computer Engineering
Table of Contents

Abstract ......................................................................................................................................................... ii

Acknowledgments ......................................................................................................................................... v

Vita .................................................................................................................................................................... vii

List of Tables .................................................................................................................................................. xvii

List of Figures ................................................................................................................................................ xviii

Chapter 1: Introduction ................................................................................................................................. 1

1.1 Motivation .................................................................................................................................................. 1

1.2 Problem Statement .................................................................................................................................... 2

1.3 Organization ............................................................................................................................................. 3

Chapter 2: HEP, the ATLAS Pixel Detector, the ATLAS DBM Detector, Radiation Hard ASIC Design, and IC Design and Reuse ............................................................... 5

2.1 High Energy Physics (HEP) ..................................................................................................................... 5

2.2 The Standard Model ............................................................................................................................... 6

2.3 The Large Hadron Collider (LHC) ........................................................................................................ 6

2.4 The ATLAS Detector .............................................................................................................................. 8
Chapter 3: The VDC12, DORIC12, and Associated Optical Array Based Fiber-optic Transceiver

3.1 The ATLAS Pixel Detector Optical Links

xi
3.1.1 The VDC ................................................................. 35
3.1.2 The DORIC .............................................................. 36
3.2 Operational Experience with the Opto-board ......................... 37
3.3 Improved Opto-board Concept .............................................. 39
3.4 Precursors to the VDC12 and DORIC12 ............................... 41
3.5 The DORIC12 ................................................................ 42
3.6 The VDC12 ................................................................... 45
3.6.1 VDC12 at 5 Gb/s ...................................................... 47
3.7 Voltage Regulator ............................................................ 52
3.8 Improved Optical Packaging .............................................. 53
3.9 The Complete Opto-board Prototype ..................................... 58

Chapter 4: The Hitbus Chip for the ATLAS DBM Detector ................ 64
4.1 Hitbus Chip Rationale ...................................................... 64
4.2 DBM Triggering and the Origins of the Hitbus Chip ................. 65
4.3 Overview of the Hitbus Chip ............................................. 66
4.4 Hitbus Chip Functionality ................................................ 70
4.5 Clock Configuration ........................................................ 71
4.6 Design of the Hitbus Chip ................................................ 72
4.7 The Programmable Delay ............................................... 73
<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.8</td>
<td>The HitOr Control Block</td>
<td>75</td>
</tr>
<tr>
<td>4.9</td>
<td>The SEU Hard PLL</td>
<td>76</td>
</tr>
<tr>
<td>4.10</td>
<td>Hitbus Chip Quality Assurance and Testing</td>
<td>80</td>
</tr>
<tr>
<td>4.11</td>
<td>Hitbus Chip Irradiation Results</td>
<td>88</td>
</tr>
</tbody>
</table>

**Chapter 5: Reused Design Integration, IP Sharing Guidelines, and a Concise IP Sharing Template**

<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>5.1</td>
<td>Design Reuse: Technology Migration</td>
<td>91</td>
</tr>
<tr>
<td>5.1.1</td>
<td>Technology Migration Flow</td>
<td>93</td>
</tr>
<tr>
<td>5.1.1.1</td>
<td>Device Scaling</td>
<td>94</td>
</tr>
<tr>
<td>5.1.1.2</td>
<td>Scaled Design Verification</td>
<td>98</td>
</tr>
<tr>
<td>5.2</td>
<td>Experiences with IP Reuse in the VDC12, DORIC12, and Hitbus Chip</td>
<td>99</td>
</tr>
<tr>
<td>5.2.1</td>
<td>8-Bit DAC</td>
<td>100</td>
</tr>
<tr>
<td>5.2.2</td>
<td>SEU Hard DICE Latch</td>
<td>103</td>
</tr>
<tr>
<td>5.2.3</td>
<td>I/O Pads</td>
<td>104</td>
</tr>
<tr>
<td>5.2.4</td>
<td>PLL</td>
<td>105</td>
</tr>
<tr>
<td>5.2.5</td>
<td>FE-I4 Command Decoder</td>
<td>106</td>
</tr>
<tr>
<td>5.3</td>
<td>Guidelines for Reuse of Collaboratively Shared IP</td>
<td>109</td>
</tr>
<tr>
<td>5.3.1</td>
<td>Guidelines for IP Reuse</td>
<td>110</td>
</tr>
<tr>
<td>5.3.2</td>
<td>Guidelines for IP Sharing</td>
<td>113</td>
</tr>
</tbody>
</table>
B.3: Cadence Encounter Place and Route Script ........................................... 161

Appendix C: Circuit Schematics and Layouts .................................................. 166

C.1:DORIC12 Amplifier Top Level Schematic .............................................. 167
C.2:DORIC12 Amplifier Top Level Layout .................................................... 168
C.3:DORIC12 TIA Top Level Schematic ....................................................... 169
C.4:DORIC12 Analog Buffer Amplifier Schematic ........................................ 170
C.5:DORIC12 Limiting Amplifier Top Level Schematic ................................ 171
C.6:DORIC12 Limiting Amplifier Layout .................................................... 172
C.7:DORIC12 Resistively Loaded Differential Pair Amplifier Schematic .... 173
C.8:DORIC12 DLL Top Level Schematic ...................................................... 174
C.9:DORIC12 DLL Top Level Layout .......................................................... 175
C.10:VDC12 Single Channel Top Level Schematic ...................................... 176
C.11: VDC12 Single Channel Top Level Layout .......................................... 177
C.12: Hitbus Chip PLL Top Level Schematic .............................................. 178
C.13: Hitbus Chip PLL Top Level Layout .................................................... 179
C.14: Hitbus Chip PLL Schematic ............................................................. 180
C.15: Hitbus Chip PLL Charge Pump Schematic ......................................... 181
C.16: Hitbus Chip PLL Ring Oscillator Schematic ...................................... 182
C.17: Hitbus Chip PLL Current Starved Inverter ........................................ 183
C.18: LDO Top Level Schematic ......................................................... 184

C.19: LDO_AMP Schematic ............................................................... 185

C.20: LDO Complete Layout .............................................................. 186

Bibliography ...................................................................................... 187
List of Tables

Table 4.1: Measured supply currents of the production Hitbus chips. ......................... 85
Table 5.1: Simulated performances of the legacy and scaled unity gain voltage buffer. 98
Table 5.2: Transistor dimensions of the legacy and scaled unity gain voltage buffer..... 98
List of Figures

Figure 2.1: Diagram showing the LHC and the position of the principal detector experiments. Image provided by CERN. ................................................................. 7

Figure 2.2: Diagram showing the ATLAS detector highlighting the sub-detectors. Image provided by CERN. .............................................................................. 9

Figure 2.3: A pixel module, 2 staves of 14 modules, and a cutaway view of the completed detector. Images provided by CERN. .............................................................. 10

Figure 2.4: Three dimensional drawing of one half of the ATLAS DBM detector illustrating its position within the data signal and powering services for the ATLAS pixel detector. Image provided by CERN. ......................................................... 11

Figure 2.5: Three dimensional drawing showing one half of the DBM detector. The two PP0 boards and the four 3 module telescopes are highlighted. Image provided by the University of Toronto. ............................................................................. 11

Figure 2.6: Diagram of an n-channel MOSFET (NFET) illustrating the gate oxide charging caused by TID. The NFET before irradiation is shown in (a) and after irradiation is shown in (b) [12]. ........................................................................................................... 14

Figure 2.7: Diagram of the top view of a standard open-layout FET compared with an ELT FET. The leakage paths caused by TID are indicated. ............................ 16

Figure 2.8: Diagram of the analog IC design flow. .................................................. 24
Figure 2.9: Diagram of the digital IC design flow ........................................................... 26
Figure 2.10: Diagram of the mixed-signal IC design flow ............................................. 29
Figure 3.1: Photographs of the top and bottom sides of an opto-board ....................... 35
Figure 3.2: Block diagram of a single module of the pixel system ................................. 35
Figure 3.3: Block diagram of the VDC circuit ............................................................. 36
Figure 3.4: Block diagram of the DORIC circuit ......................................................... 37
Figure 3.5: Block diagram of the improved opto-board prototype ............................... 40
Figure 3.6: Photograph of the DORIC12 die .............................................................. 43
Figure 3.7: Layout of the DORIC12 ......................................................................... 43
Figure 3.8: Block diagram of the DORIC12 circuit ..................................................... 44
Figure 3.9: Photograph of the VDC12 die ................................................................. 46
Figure 3.10: Layout of the VDC12 ............................................................................ 46
Figure 3.11: Block diagram of the VDC12 circuit ...................................................... 47
Figure 3.12: Photographs of the VDC12 test PCB designed for operation at 5 Gb/s ...... 48
Figure 3.13: Optical eye diagram of a representative VDC12 channel coupled to VCSEL. The eyes shown compare the VDC12 performance with only one channel active and with all channels active ............................................................................................................ 50
Figure 3.14: Optical eye diagram of a representative VDC12 channel coupled to VCSEL compared to the optical output performance of a commercial SFP+......................... 51
Figure 3.15: Performance of two LDO voltage regulators during irradiation. Plot provided by the University of Siegen ................................................................. 52
Figure 3.16: Yields of the improved VCSEL (a) and PIN (b) array opto-packs ............. 54
Figure 3.17: Fiber coupled optical power from all VCSEL (a) and responsivity of all PIN (b) channels after mounting in the improved opto-packs. .............................................. 54
Figure 3.18: Drawing of the opto-shield mounted on an opto-pack, (a) front view and (b) cross sectional view revealing the half etch. ......................................................... 57
Figure 3.19: Photographs of the opto-shield mounted to an opto-pack: (a) VCSEL / PIN and wire bond cavity, (b) MT ferrule in place sealing the front of the cavity, (c) gap in opto-shield to prevent channel shorting to be filled with encapsulant for cavity sealing. 57
Figure 3.20: Photograph of the completed opto-board prototype front (a) and back (b). 59
Figure 3.21: Post-irradiation optical eye from the VDC12 spare channel at 160 Mb/s. 63
Figure 4.1: Block diagram of the ATLAS DBM detector. ............................................. 65
Figure 4.2: Block diagram of the signal connectivity for one quarter of the ATLAS DBM detector. The Hitbus chip is situated on PP0. .................................................. 67
Figure 4.3: Block diagram of the Hitbus chip. ............................................................... 69
Figure 4.4: Block diagram of the Hitbus control block. Cells acquired from collaborators are highlighted in yellow. ................................................................. 69
Figure 4.5: Photograph of the Hitbus chip die and an image from the design environment used to highlight the main components within the chip. ................................. 73
Figure 4.6: Block diagram and layout of the TTC delay cell. ...................................... 75
Figure 4.7: Block diagram of the PLL implemented in the Hitbus chip. ...................... 77
Figure 4.8: Photograph of the Hitbus chip test system. ............................................. 82
Figure 4.9: Photograph of a packaged Hitbus chip mounted to a PP0 board. .......... 83
Figure 4.10: Measured LVDS driver differential output amplitudes of the production Hitbus chips. ................................................................. 85
Figure 4.11: Measured LVDS driver common mode levels of the production Hitbus chips. ......................................................................................... 86
Figure 4.12: Measured LVDS driver rise times of the production Hitbus chips. ........... 86
Figure 4.13: Measured LVDS driver fall times of the production Hitbus chips.............. 87
Figure 4.14: Measured difference between the delay of the XCK and TTC for every matched delay pair in the production Hitbus chips......................................................... 87
Figure 4.15: Measured maximum possible delay for all XCK and TTC in the production Hitbus chips. ................................................................................. 88
Figure 4.16: Observed SEU induced function register reconfiguration events in both irradiated Hitbus chips. ................................................................. 90
Figure 4.17: Observed supply current behavior of both Hitbus chips during the irradiation............................................................................................................. 90
Figure 5.1: Diagram illustrating the technology migration flow. .................................. 93
Figure 5.2: Framework for the technology migration flow. ........................................... 94
Figure 5.3: Schematic of a legacy differential-input single-ended output amplifier configured as a unity gain buffer. ................................................................. 95
Figure 5.4: AC response of the legacy and scaled unity gain voltage buffer. .............. 97
Figure 5.5: Linearity of the legacy and scaled unity gain voltage buffer. ................. 97
Figure 5.6: Illustration showing the shorted via in the Hitbus chip and an SEM image of the via after FIB removal of the short......................................................... 109
Figure 5.7: Top level structure of the IP sharing template. ............................................. 116

Figure 5.8: Example contents of the readme.txt file...................................................... 117

Figure 5.9: Summary tab of the vitals spreadsheet......................................................... 119

Figure 5.10: Port list tab from the vitals spreadsheet..................................................... 120
Chapter 1: Introduction

1.1 Motivation

For more than a decade, billions of Euros have been spent by governments around the world to construct the Large Hadron Collider (LHC) and its associated high energy physics (HEP) experiments, A Toroidal LHC Apparatus (ATLAS) and the Compact Muon Solenoid (CMS), at the European Organization for Nuclear Research (CERN). In the past year, data from ATLAS and CMS have successfully detected a Higgs boson at 126 GeV [1, 2]. This particle was one of the main reasons for constructing the LHC and is one of the most exciting scientific discoveries in decades. While finding a Higgs boson has partially validated the tireless efforts of thousands of scientists and engineers collaborating to construct, commission, and operate the LHC, ATLAS, and CMS, the field remains in a constant race to enhance its instrumentation. By improving detector sensitivities, packing more readout channels into smaller areas, and increasing data collection efficiency, physicists aspire to enhance the quality, granularity, and sheer amount of data they can collect for making discoveries. The core of this dissertation is based on the development of small but critical components made to fuel this HEP upgrade race.
1.2 Problem Statement

Due to high radiation environments, commercially available electronics are often unsuitable for use in the instrumentation of HEP experiments. Additionally, due to mechanical and space constraints, the size and materials used in their construction often eliminate commercial devices from consideration. Furthermore, the market size and growth potential of components needed by HEP are normally below the levels that qualified companies require to take on a development project. Given these facts, physicists themselves lead the design and construction of most subsystems within their experiments. Because of the large scale of today’s HEP detectors, their construction would not be possible without the collaborative spirit of thousands of scientists and engineers. Included in the required engineering is the design of application specific integrated circuit (ASIC) chips. In attempts to save effort, designs of blocks are frequently shared among collaborating institutes. However, this intellectual property (IP) transfer is often complicated by institutes using diverse tool flows or providing caveats such as: “Note that we will be able only to give very limited support to you. The designer of this block has rather unexpectedly left us, and we have a lot to do right now to cope with his absence.”

This dissertation describes the design, verification, fabrication, testing, and qualification of three radiation hard ASIC chips developed within the HEP environment discussed above. Two of the ASICs were designed to form a radiation hard chipset for a novel optical array based fiber-optic transceiver board enabling active redundancy. The other ASIC was developed for the ATLAS diamond beam monitor (DBM) to provide the
capability for the DBM detector to trigger its own readout. These three ASICs, designed in a 130 nm complementary metal oxide semiconductor (CMOS) process, are mixed-signal in nature and contain fully customized analog and digital blocks. Additional digital blocks were designed using very-high-speed integrated circuits hardware description language (VHDL) that were synthesized and automatically placed and routed using standard cell libraries. In addition to these new designs, several shared IP blocks were incorporated into all three ASICs. In this dissertation we report on the design and testing of these ASICs including what has been learned through the process of collaboration and design reuse. We present guidelines for effectively sharing, integrating, and verifying shared block designs. Moreover, we propose a concise template for IP sharing.

1.3 Organization

The remainder of this dissertation is structured as follows:

Chapter 2 provides background information on HEP, the ATLAS pixel detector, the ATLAS DBM detector, radiation effects and radiation hardening of CMOS, IC design, and design reuse.

Chapter 3 details the development of two radiation hard ASIC chips, the VDC12 and DORIC12, for our novel optical array based fiber-optic transceiver board enabling active redundancy. We discuss the development of the two ASICs, their operational features, the design of the fiber-optic transceiver board, the design of custom optical packages for the fiber-optic transceiver board, and the results of the performance of the
fabricated ASICs, optical packages, and transceiver boards including an irradiation with 24 GeV protons to $1.5 \times 10^{15} \text{ p / cm}^2$.

In Chapter 4 we describe the design and results from the Hitbus chip, a radiation hard ASIC designed for the ATLAS DBM detector. The circuit design and verification of an SEU hard PLL, SEU hard control block, and SEU hard programmable delay will be discussed. We present results from bench tests and an irradiation with 24 GeV protons to $4.33 \times 10^{15} \text{ p / cm}^2$.

In Chapter 5 we review our experiences in reusing designs from both our own libraries and collaboratively shared IP within the ASIC chips designed in Chapter 3 and 4. Insights and methods for designing and verifying mixed signal ASICs including reused designs are discussed.

Chapter 6 provides concluding remarks and recommendations on the future direction of this work.

Appendix A includes VHDL code for the synthesized, placed, and routed digital blocks within the Hitbus chip.

Appendix B lists examples of the scripts used to perform synthesis, placement, and routing of the Hitbus chip control block.

Appendix C contains circuit schematics and layouts from the ASICs.
In Chapter 1, we introduced the foundation of this dissertation, the design of three ASICs for HEP. In this chapter, we detail the critical concepts driving the design of these ASICs. First, we provide an overview of HEP and the detectors in which our ASICs are used: the ATLAS Pixel detector and the ATLAS DBM detector. Next, we describe radiation-induced effects in ASICs and the process of radiation hardening by design. Concluding Chapter 2, we examine IC design and design reuse.

2.1 High Energy Physics (HEP)

High energy physics, also referred to as particle physics, is the study of the existence and interactions of particles that are the constituents of matter. In order to probe this realm, massive particle accelerators, collider, and detector systems are used to image the aftermath of collisions between particles traveling close to the speed of light prior to impact. By investigating an extremely large quantity of these particle collisions it becomes possible for physicists to observe a sufficient number of rare events that they become statistically significant. Only after sizable data sets have been collected can physicists seriously examine the quality of previous measurements or make new research studies. Given this requirement, the sensors, data acquisition hardware, and computing
infrastructure must collect data with maximum efficiency and accuracy in order to speed the time it takes for discoveries to be made.

2.2 The Standard Model

The Standard Model of particle physics is the model developed to understand the basic structure of the universe. To date, HEP experiments have found the Standard Model to match well with observations. However, a major component of the Standard Model called the Higgs boson had not been observed experimentally until very recently [1, 2]. With this discovery, the Standard Model is likely to become even more robust. The Standard Model describes all of the known fundamental particles and the interactions between these particles. The interactions include the strong nuclear force, the weak nuclear force, and the electromagnetic force. It is important to note that particles also interact via the gravitational force which is not explained by the Standard Model. The fundamental particles that exist in the Standard Model, classified as leptons, include the electron, the muon, the tau, and three neutrinos. The Standard model also includes quarks, and the interaction carriers called the photon, gluon, and the W and Z bosons.

2.3 The Large Hadron Collider (LHC)

To probe and verify the Standard Model and search for phenomena outside the existing theory, the LHC [3] was constructed at CERN in Geneva, Switzerland. The LHC is the largest and highest energy proton-proton collider in the world with a circumference of 27 km and center of mass collision energy of 8 TeV. By 2018, the center of mass
collision energy will be increased to 14 TeV through repairs and upgrades. The LHC is designed to accelerate and collide protons and lead nuclei. Presently, there are four principal detectors (Figure 2.1) collecting data from the LHC: A Large Ion Collider Experiment (ALICE) [4], A Toroidal LHC Apparatus (ATLAS) [5], Compact Muon Solenoid (CMS) [6], and Large Hadron Collider beauty (LHCb) [7]. Each detector is positioned at separate collision site locations around the LHC. Collision sites are locations where the LHC creates an intersection of its high energy particle beams causing collisions that the detectors in turn measure.

Figure 2.1: Diagram showing the LHC and the position of the principal detector experiments. Image provided by CERN.
2.4 The ATLAS Detector

When the proton beams from the LHC intersect, a wide spectrum of particles are produced from the proton collisions. The ATLAS detector, shown in Figure 2.2, is designed to measure this sea of particles and is the largest and the most complex high energy physics detector ever constructed. The innermost sub-detectors within ATLAS are used for finding the trajectories of the resultant particles as they travel from the collision point. These inner detectors, referred to as tracking detectors, are constructed using semiconductor pixel, semiconductor strip, and straw-tube detectors with each detector providing precise position resolution. The next layer of detectors beyond the tracking detectors are the electromagnetic and hadronic calorimeters. The electromagnetic and hadronic calorimeters are designed to measure the energy and position of electromagnetic particles and hadrons. The final sub-detector of ATLAS is the muon spectrometer. The muon spectrometer lies outside all of the other detectors because most other particles produced in the collisions are absorbed by the calorimeters. In addition to the detector systems, ATLAS also incorporates a powerful magnet system consisting of a solenoid surrounding the inner detector and a toroid consisting of eight air-core barrel loops and two end-cap magnets positioned outside the calorimeters but within the muon spectrometer. The strong fields produced by these magnets bend the post-collision particle trajectories allowing their momentum to be measured.
2.5 The ATLAS Pixel Detector

The ATLAS pixel detector, shown in Figure 2.3, is a major subsystem of the ATLAS detector and is the tracking device closest to the particle collision point. The ATLAS pixel detector consists of three barrel layers and three forward and backward disks on each end-cap. The pixel detectors themselves are reversed biased silicon diodes, with each pixel measuring 50 x 400 µm². The detector is segmented into modules with each module containing 47,268 pixel elements. In the entire detector, there are a total of 1,744 modules corresponding to ~80 million individual pixels [5].
2.6 The ATLAS Diamond Beam Monitor

The LHC is presently in a two year shutdown for upgrades and repairs. Among the upgrades include the insertion of a new detector subsystem within the ATLAS experiment called the Diamond Beam Monitor (DBM). The DBM detector, shown in Figure 2.4 and Figure 2.5, is comprised of four 3-layer telescopes placed on each side of the ATLAS interaction point a few meters forward and behind the pixel detector. Each telescope layer consists of one front end amplifier ASIC called the FE-I4 [8] that is bump bonded to a spatial sensitive pixel detector made with chemical vapor deposition (CVD) diamond and wire-bonded to a flexible circuit. Given this arrangement, the DBM consists of eight telescopes with 24 total modules. Each module connects to 20 mm x 16.8 mm of active detector area broken into 26,880 pixels [9]. To distribute the readout signals and power supply services to the DBM, two printed circuit (PCB) cards, named Patch Panel Zero (PP0) boards, are placed near the DBM modules.
Figure 2.4: Three dimensional drawing of one half of the ATLAS DBM detector illustrating its position within the data signal and powering services for the ATLAS pixel detector. Image provided by CERN.

Figure 2.5: Three dimensional drawing showing one half of the DBM detector. The two PP0 boards and the four 3 module telescopes are highlighted. Image provided by the University of Toronto.
2.7 Radiation Effects in CMOS and Radiation Hardening

Since the pixel and DBM detectors are very close to the particle collision point in ATLAS, they are inaccessible for years at a time, are exposed to a heavy dose of radiation, and all installed components are required to be radiation hard. The level of radiation hardness required varies for each component within ATLAS. The expected radiation dose is based on the location relative to the particle collision point. For the CMOS ASIC chips discussed in this dissertation, special techniques were used in order to meet the radiation hardness requirements. In the following sections we discuss the effects of radiation in CMOS devices and what steps can be taken to mitigate the unfavorable impacts.

2.7.1 Radiation Damage

Energy deposited by transiting radiation damages the behavior of and the materials used to form the devices present in CMOS processes. This damage has two causes, displacement and ionization [10]. Displacement damage is caused when the incident particles move atoms from their original locations. The impact of displacement depends on exactly where the displaced atoms come to rest. If they are moved into an interstitial site this will create a vacancy adding a new energy level to the bandgap. The performance of the device will then be altered. Ionization damage is caused by the generation of free electron-hole pairs when radiation passes through or stops in a material. The quantity of the free carriers generated is proportional to the dose or
radiation absorbed. These free carriers may or may not affect the device depending on where they travel.

### 2.7.2 Total Ionizing Dose

The total amount of energy absorbed in a material from radiation induced ionization is referred to as the total ionizing dose (TID). TID is specified in units of rad or grey (Gy). One rad is equal to 0.01 Gy. Both units provide a measure of the amount of energy absorbed in a material where one rad is the dose required to deposit $10^{-5}$ joules of energy in one gram of the material under question. The energy deposited in the target material is dependent on the properties of the target material itself and the energy, mass, and charge of the incoming particle.

For CMOS, TID is a major concern. The principle negative impact occurs in the oxide layer, made out of SiO$_2$, used to separate the metal-oxide-semiconductor field-effect-transistor (MOSFET) gate from the channel. Due to ionization, electron-hole pairs are created throughout the MOSFET. In the Si of the transistor bulk or the polysilicon forming the gate, the mobilities of the electrons and holes are similar and thus recombine quickly. However, in the oxide, holes have a much lower mobility. Because of this, recombination in the SiO$_2$ is not efficient and the holes that do not recombine travel to and become trapped at the Si-SiO$_2$ interface. In turn, the trapped charge changes the threshold voltage and mobility of the MOSFETs. Obviously, a change in the MOSFET threshold voltages can have a dramatic impact on the circuit’s behavior by modifying the bias conditions. Furthermore, a change in the MOSFET mobility reduces the
transconductance thus reducing the current flow in the device [11]. A diagram illustrating this effect is shown in Figure 2.6.

2.7.3 Mitigating Radiation Damage Effects

For modern CMOS processes with low operating voltages, displacement is not a significant concern unless the dose is very high, beyond $10^{16}$ 1 MeV equivalent neutrons
of total flux. The main reason for this is the fact that displacement primarily impacts the behavior of minority carrier devices and MOSFETs are majority carrier devices [13, 14]. However, the damage caused by TID in CMOS can be severe and designs must be hardened if they are destined for use in radioactive environments. Fortunately, to harden CMOS circuits against TID, a few approaches are available.

The first approach requires the application of special fabrication techniques referred to as process hardening. References describing various process hardening techniques may be found in the literature [15, 16, 17, 18, 19, 20]. As was mentioned in Chapter 1, the market for using a radiation hard process is small and thus the processes have low volume. Because of this, manufacturers do not devote as many resources to optimizing the processes, typically resulting in low yields. Given this, the costs of fabricating a design in a hardened process can be quite high and are thus not attractive to the high energy physics community. The second and simplest approach to achieve TID hardening is to take advantage of the fact that as process geometries shrink the associated oxide thicknesses shrink. This is very beneficial because it has been shown that MOSFETs with thinner oxides are more tolerant to TID than those having thicker oxides [21, 22, 23]. Therefore, if a process is chosen with sufficiently thin oxides for the expected TID, an ASIC designed using its thin oxide transistor variant will survive. Unfortunately, in many cases, it is not possible to use processes or devices with sufficiently thin oxides due to budgetary concerns or I/O voltage requirements. In this case, the final approach to harden CMOS circuits is to employ special layout techniques. Several layout methodologies have been developed and are discussed in [24, 25] with the
most effective and widely used in HEP being the use of enclosed layout transistors (ELTs) as described in [26]. With ELT devices, shown in Figure 2.7, the leakage paths between the source and drain caused by TID are eliminated. While ELTs cure the threshold shifts due to TID, there are some drawbacks to using this geometry in a design. First, the width to length ratio (W/L) is limited to values of ~7 and higher depending on the technology node process design rules. This is due to the fact that the gate must surround the inner diffusion making the minimum width quite large. Second, because of these large minimum widths, the parasitics of the ELT are higher than what can be achieved with a standard open-layout FET. Finally, it is difficult to model the W/L ratio for design and simulation purposes because ELT layouts are fully customized by the layout engineer and modeling is done at each process node through empirical methods without help from the foundries.

Figure 2.7: Diagram of the top view of a standard open-layout FET compared with an ELT FET. The leakage paths caused by TID are indicated.
In addition to the problems discussed earlier with the gate oxides, the field oxides in CMOS processes are also subject to damage from TID. For further protection against TID, it is also beneficial to place p-type guard rings between p-channel MOSFETs (PFET) and NFETs to reduce the leakage between the different devices [26]. Unfortunately, as with ELTs, the addition of the p-type guard rings increases the interconnect parasitics and die area used to form a circuit hardened against TID.

2.7.4 Single Event Effects

When particles having sufficient energy traverse sensitive areas of a CMOS circuit, single event effects (SEE) may occur. Like TID, the ionizing particles generate free electrons and holes in the CMOS structures. It is the collection of these carriers in sensitive nodes of the circuit that produce SEE. SEE can cause temporary changes in the logical states, interruptions to the normal operating behavior, or in the extreme case, permanent damage. SEE in microelectronics are widely studied. A historical perspective is collected in [27]. SEEs fall under three categories: soft SEE, hard SEE, and destructive SEE. Soft SEE are temporary effects where recovery and correction are possible. Soft SEE can result in bit flips called single event upsets (SEU), transients in analog circuitry called single event transients (SET), or single event functional interrupts (SEFI) that place circuit into a non-desired mode causing a temporary loss of functionality. Hard SEE occur when enough energy is deposited from a particle ionization in the gate of a MOSFET that the threshold voltage of the device permanently altered. Destructive SEE
can be manifested as single event latchup (SEL), single event burnout (SEB), single event
snapback (SESB) or single event gate rupture (SEGR) [28].

2.7.5 Mitigating Single Event Effects

Mitigation of SEE is possible at each level of the CMOS design process. At the
process technology level, modifying device structures and employing special process
hardening techniques can enhance SEE tolerance. However, as was mentioned earlier, the
cost of fabricating a design in a hardened process can be quite high. At the system level,
error detection, error correction, software algorithms, and redundant blocks may be
included to handle SEE. Unfortunately, these techniques can introduce significant
complexity and overhead to a system. At the circuit design level, modifications to the
circuit topology and layout can be made to reduce SEE sensitivity. Circuit level
implementations are the most efficient methods to harden against SEE and are of critical
importance to the ASICs we have developed.

SEU, SET, and SEL are the main effects that can be mitigated in CMOS circuits.
Conveniently, SEL susceptibility can be minimized by following the same techniques
outlined above, inserting p-type guard rings, for reducing the leakage between PFETs and
NFETs [29]. The devices most susceptible to SEU and SET are register and memory
cells. Fortunately, through layout modifications and special circuit topologies it is
possible to make hardened registers and memories [30, 31, 32, 33]. As process feature
sizes trend smaller, the probability of SEU and SET go up [34]. This is due to the fact
that reducing the capacitance of the sensitive nodes in turn lowers the charge required by
an ionizing particle to significantly impact the voltage at the sensitive nodes. However, by simply increasing the time constants of the sensitive nodes, this can be partially overcome. Unfortunately, if high speeds are required, the transistor drive strengths must be increased to compensate for the larger parasitics and thus the circuit will suffer an increase in current consumption. In addition, the shrinking of CMOS circuitry makes the likelihood that a single strike will introduce SEU and SET in neighboring devices. A method to combat this is to include redundant copies of the same cell distributed by some distance within the die. The spacing between the redundant cells is determined by tradeoffs between the speed of the circuit, the die area usage, and the level of confidence that the separation is sufficient to be immune to SEU / SET from the same particle. The result of the logic function of redundant cells is then passed through a voting circuit with the final outcome determined by a majority vote. By using votes requiring 2 out of 3 or votes of 4 out of 5, the probability that an SEU or SET will propagate can be greatly reduced. Therefore, triplicated registers, combinatorial logic, or memory elements can be made that are sufficiently SEU / SET hard for most any application. In the ASICs discussed in this dissertation, the dual-interlocked cell (DICE) topology [35] is used to implement all static configuration registers.

2.8 IC Design and Reuse

The first integrated circuit was conceived and constructed by Jack Kilby and Robert Noyce in 1958. It is remarkable that in the short time since, CMOS ICs are now possessed by almost every human being, and have dramatically impacted the way
humans live. Our information society would not exist without the developments made by semiconductor manufacturers at the wafer level that continue to keep pace with Moore’s law [36]. Presently, the numbers of functions that can be included per square millimeter of silicon demand that IC designers become not only more productive but more creative. It also demands that they work at full capacity for longer stretches to fill the increasing die area. Engineering time is already quite expensive and increasing individual productivity is challenging. As such, techniques are being developed to more efficiently fill die area. In the realm of the relatively low transistor count ASICs discussed in this dissertation, these techniques have been partially adopted to minimize the length of design cycles given the limited manpower available. In the next sections, we introduce the IC design process including methods employed to efficiently assemble, verify, and fabricate designs that include IP.

2.8.1 IC Design Flows

Before advancements to computing in the 1970’s and 1980’s, IC layouts were crafted by skilled technicians who adhered opaque tape to films to be used as photomasks. Today, IC layout and design is accomplished through using electronic design automation (EDA) software suites. The industry leaders in EDA include Cadence Design Systems [37], Synopsys [38], and Mentor Graphics [39]. The available EDA tools include schematic capture, manual layout composition, design rule checkers (DRC), layout versus schematic (LVS) verification, and automated place and route (PNR). The final output of these powerful computer aided design (CAD) tools is photo-mask artwork
not significantly different than what used to be generated by black tape and transparencies.

The design flows used to reach production of final mask sets for fabrication vary depending on the type of IC that is being produced. There are three main categories of IC design: digital, analog, and mixed-signal. Traditionally analog and digital designs were kept on separate die and mixing the two, mixed-signal, was done through the interconnection of the individual die on a printed circuit card or hybrid board. As process technologies and markets evolved, the first mixed-signal ICs were developed. Currently, most ICs produced, including the ASICs discussed in this dissertation, are mixed-signal in nature. The IC design process, whether digital, analog, or mixed-signal, begins with a system or block specification and then the acquisition of a process design kit (PDK) from the fabrication vendor in the specific process node that the design is targeted for. These design kits contain detailed documentation on the fabrication process, and a set of files containing the same information in syntax to be understood by the EDA tools. Once a PDK is in hand, the path to producing an IC diverges for a purely analog or digital design. In the case of a mixed-signal design, the separate analog and digital flows merge at the simulation and verification steps.

2.8.1.1 Analog IC Design Flow

Despite advancements in digital signal processing (DSP) and trends to perform traditionally analog functions in software, analog circuits will always be necessary to provide an interface to the natural world [40, 41]. Given the variety of circuit topologies,
sensitivity to process and environmental variations, and the nearly limitless tradeoff space between performance, size, and other requirements, analog IC design requires skill and knowledge that can only be collected through experience. While promising work continues on small cells [42, 43, 44], the automatic generation of analog designs based on a general set of specifications remains a goal that has not been fully realized.

The analog design flow, shown in Figure 2.8, begins with the composition of a schematic by pulling devices from the design kit libraries, placing them, connecting them, and sizing them. At this stage, the designer performs simulations to examine the functionality of the circuit. These simulations can be done within the schematic entry environment or through scripts. By using a script based simulation approach, it is simple to modify parameters and stimuli programmatically thus allowing automatic exploration of a circuit. Using scripts also enables direct preservation of documentation on the simulations performed and outcomes achieved. Through these initial simulations, the circuit and devices are modified and adapted until the desired specifications are met. Optimization tools are available in the EDA suites, and can be used to speed this process as long as the circuit topology is fixed. Once satisfied with the initial simulations, the designer then verifies the robustness of the circuit under temperature, supply voltage, and process parameter extremes. These process parameter extremes are known as corners. If time permits, a more detailed exploration of the behavior of the circuit over a wider range of process parameter values, called a Monte Carlo analysis, can be performed. Since Monte Carlo analysis is statistical in nature, running more simulation trials yields more convincing results. However, these trials can add significant simulation time. Once
complete with the schematic verification, the designer either begins work on the physical layout or goes back to edit the circuit schematic to address flaws discovered in the corner or Monte Carlo simulations.

At the physical layout stage of the analog design flow, a designer reads the fully verified schematic and places layout views of the components listed in the schematic into a layout editor. Composing a layout that does not significantly diminish performance requires a very thorough understanding of the process node and also detailed knowledge of the circuit’s behavior. Effectively balancing speed, noise, power distribution, and device matching in an IC layout is an art form. As the layout is composed, the design should be passed through DRC many times. It is important to perform DRC often because repairing errors can significantly alter the design. Also, as a layout grows, some DRC errors might become difficult to repair without drastically modifying the rest of the design. Once the layout is DRC clean and complete it is passed through LVS. Since the process of analog IC layout is manual, it can be prone to error. Therefore, repairing mistakes to arrive at an LVS clean layout often takes several iterations. Once the design is both DRC and LVS clean, its netlist is extracted to include the parasitic resistances, capacitances, and inductances that arise from the layout. At this point, the extracted layout (including parasitics) is verified using the same simulations from the schematic design phase to check the robustness of the layout under extremes in temperature, supply voltage, and the process corners. Usually, problems will be discovered and corrected by optimizing the layout or modifying the schematic. Once fully verified at the layout level, the design can be submitted for fabrication.
2.8.1.2 Digital IC Design Flow

With the exception of special circumstances the creation of digital ICs are completed through automatic means. Therefore, the digital design flow, illustrated in Figure 2.9, can be much quicker. The first step in digital design is to compose the hardware description language (HDL) description for meeting the system or block specifications. The HDL is then simulated and verified until the designer is satisfied that the logic is correct. With the completed HDL in hand, scripts must now be composed to direct the EDA tool responsible for performing logic synthesis. Logic synthesis is the conversion from the abstract HDL to a logic gate-level netlist built out of parts from a standard cell library designed for the particular process node. While the logic synthesis
tools from the EDA vendors allow interactive operation, using scripts to direct the synthesis is much more robust and guarantees that the same steps and settings are used each time a design synthesized. The synthesis scripts include design decisions such as timing constraints and controls over logic usage and optimization. Once a synthesized netlist comprised of standard cells is available, the next step is to compose scripts to direct the EDA tool responsible for performing the placement of the layout cells and connection of the netlist. This process is called place and route (PNR). The scripts for PNR also include design decisions like what metal layers to use, the allowed size of the design, the size and spacing of the power grid, and the timing constraints. Additionally, the DRC and LVS are completed during the PNR. This comes about as long as the PNR scripts are written using a correct by construction approach and reference the appropriate technology rules in the PDK and the standard cell library. As in analog design, once a design has finished PNR its netlist is extracted to include the parasitic resistances, capacitances, and inductances that arise from the layout. Due to the complexity of digital blocks and their large number of nodes, the extraction must often be performed using techniques to reduce the number of parasitic devices that are added to the netlist. This is due to the fact that simulations become more intractable as the number of elements increase. However, given the nature of digital design, it is common to optimize post-layout simulations by replacing the physical logic cell layouts with functional models provided by the standard cell vendor. In addition, the digital verification complexity is reduced by estimating the routing net parasitics and then inserting appropriate delays into the interconnections of the functional models. These digital post-layout modeling
techniques are very mature, and the performance predicted by the principle EDA vendors are quite representative of actual layouts. As with the analog flow, the layout including parasitics is verified using the same simulations from HDL design phase. This verification includes checking the robustness of the design under extremes in temperature, the supply voltage, and the process corners. At this stage, problems will be discovered that are then corrected by modifying the synthesis scripts, PNR scripts, or the HDL.

Figure 2.9: Diagram of the digital IC design flow.
2.8.1.3 Mixed-Signal IC Design

To simplify packaging, minimize size, and maximize performance, analog and digital designs are frequently combined into single die. In mixed-signal design, the digital and analog flows described earlier remain separate at the block level and unite at the verification steps. A block diagram of the mixed-signal design flow is shown in Figure 2.10. In addition to the design challenges introduced by the operation of noisy digital circuits in close proximity to analog circuits, the vastly different analog and digital verification processes introduce further complications. Unfortunately, the verification and simulation of the behavior of the interconnected analog and digital blocks is only possible once the full layout is complete. Depending on the size of the chip and the precision of the parasitic extraction, a simulation of the full chip including all blocks represented at the transistor level may not be possible in a reasonable time frame. To overcome this, investments in computing power or reductions in the precision of the parasitic extraction are possible solutions. However, despite these efforts, whole chip simulation runs can still take weeks or months. Since verification of the behavior of the fully interconnected chip is required in order to submit confidently for fabrication, these delays are unacceptable for almost any IC design. As was mentioned in Section 2.8.1.2, post-layout simulation for digital blocks, using functional models and delays provided by the standard cell vendor and estimated by the EDA tools, is a very mature and accurate process. These estimation techniques greatly increase the speed of the digital verification. Unfortunately, creating similarly accurate and reliable models for analog blocks is very difficult. The process of creating analog behavioral models, called macromodeling, is an
active research topic. In practice, the most widespread approach towards macromodel creation is manual abstraction [45]. During manual abstraction, macromodels are created by the block designer through mining simulation results and fitting them to functions. This is the only feasible approach for some blocks and unfortunately has deficiencies. Manual abstraction’s primary deficiency is its tendency to miss errors in the design that the verification was meant to discover in the first place. What if the designer performing the abstraction did not conceive of the deficiency? Could it then be modeled? Since accurate manual macromodeling is experience based, time consuming, and requires highly detailed knowledge of the block under consideration, the time required to develop a macromodel may negate the time saved later in the verification activities. Given the above, it is of great interest to develop and utilize automated techniques for creating macromodels [45]. Ultimately, if there is sufficient time for performing transistor level simulations on the full mixed-signal chip they should be done even if proven macromodels are available. This is simply due to the fact that the accurate device models and parasitic extraction rules available in modern EDA tools and PDKs will provide a very close approximation to the behavior of the fabricated design.
2.8.2 Design Reuse in ICs

As a mixed-signal IC grows in complexity, it is commonly referred to as a system on a chip (SOC). SOCs include combinations of processors, memories, data converters, power regulators, radio receivers, and radio transmitters. The combination of such devices on a common substrate is challenging and can compromise the performance of
the analog elements. Additionally, the inherent complexity makes verifying SOC designs tedious and in some cases impossible. However, designers are eager to confront these challenges in order to benefit from reductions in manufacturing and packaging expenses. Also attractive are the associated reduced power budgets and speed improvements made possible by on die routing of the interface signals between the SOC components. A great example of a modern SOC is Broadcom’s BCM2835 [46] that is behind the features that have made the Raspberry Pi [47] a cultural phenomenon. Over the past 15 years, SOC designs have become more complex and their rapid development has been made possible through the practice of reusing block designs. Design reuse enables an accelerated and less expensive IC development cycle due to the fact that the reused blocks are already designed and verified. These reusable block designs are often referred to as IP cores [48]. The practice of IP reuse is so commonplace that today there are scores of IP vendors and websites where the exchange of IP is managed [49, 50, 51]. Unfortunately for designs that are required to be radiation hard, such an IP marketplace does not exist.

2.8.2.1 IP Classifications

The methodology to design or utilize a reusable core depends on the function of the core and its classification [52, 53]. There are three classifications of IP cores: soft IP, firm IP, and hard IP.

Soft IP cores come in the form of RTL or a high level description such as VHDL or Verilog. Soft cores leave synthesis and PNR to the end user and are thus process node independent. This freedom leaves much verification responsibility to the end user since
the performance of a digital block is highly dependent on the decisions made by the synthesis scripts and in the PNR process. Fortunately, synthesis and PNR scripts are often provided with soft IP. Given the immaturity of the synthesis of general analog circuitry, soft IP cores are used primarily for digital blocks [54].

Both analog and digital designs are commonly distributed as hard IP. Hard IP cores are process node dependent and include the layout level description of a design. Due to this, hard IP blocks, especially those which have been fabricated and tested, can be confidently expected to perform up to the specifications. The drawback to hard IP is that it forces use in a single process node. This severely limits the portability of the hard IP. Furthermore, the long term value of hard IP is limited as fabrication DRC rules are subject to change. This could render the IP obsolete since a design with DRC errors is usually rejected for fabrication.

Firm IP contains a mixture of hard and soft IP. Firm IP is tunable by the end user and is made available with some physical design content to enable fabrication of a design based on the parameters selected by the end user. Given this, firm IP is process node dependent and thus hard IP drawbacks apply.

2.8.2.2 IP Core Design Challenges

The central concept behind creating an IP core is that it be designed with reuse in mind [48, 52]. This is a new approach for the majority of designers and the process is more difficult and time consuming than developing a standard block. One reason for this difficulty is that for a design to be reusable it must be compatible with simulation,
synthesis, and verification in a variety of EDA tool and standard cell library environments. If this is not the case, a design is not truly portable. Under this directive, the IP core designer must take care that the design does not require the usage of special cells that may not be available in all standard cell libraries. Furthermore, the designer must be familiar with many EDA tools and check the design under these tools. Designing for reuse is difficult too in that it is often required that a design be compatible with many process nodes. This is problematic in that for hard IP, an efficient strategy to translate the design into different process nodes is required. For soft IP, the process is more straightforward, but of course the large range of performances possible from one process node to another must be considered when specifying a design. Additionally, designing for reuse is complicated because a designer must be very diligent in the documentation of designs. This documentation extends to rigorous and well-structured commenting in PNR, synthesis, simulation, and verification scripts.
Chapter 3: The VDC12, DORIC12, and Associated Optical Array Based Fiber-optic Transceiver

As introduced in Chapter 2, the ATLAS pixel detector is a major subsystem of the ATLAS experiment. In order to send and receive data from this ~80 million channel detector, it was decided to use fiber optic links. In this chapter we detail the development of two radiation hard ASIC chips designed for use in upgrades to the optical links of the ATLAS pixel detector, the VDC12 and DORIC12, along with a novel optical array based fiber-optic transceiver board enabling active redundancy to house the ASICs. We provide an overview of the optical link system and discuss the development, optimization, and verification of the two ASICs, their operational features, the design of the fiber-optic transceiver board, the design of custom optical packages for the fiber-optic transceiver board, and the results of the performance of the fabricated ASICs, optical packages, and transceiver boards including an irradiation with 24 GeV protons to $1.5 \times 10^{15}$ p / cm².

3.1 The ATLAS Pixel Detector Optical Links

When a charged particle passes through one of the individual silicon pixels in the pixel detector, charge carriers are freed within the silicon and are in turn sensed by a low noise amplifier. This signal is then digitized and serially transmitted on 1 m of twisted-
pair cable, called Type0 cable, from the pixel front end electronics to an optical module. The optical modules, called opto-boards, are mounted on a patch panel called PP0. On the opto-board, this differential signal is converted by a vertical cavity surface emitting laser (VCSEL) driver ASIC (VDC) into a single-ended signal appropriate to drive a VCSEL [55]. It is then transmitted over ~100 m of optical fiber to the readout system in the detector control center. This data output (DTO) signal connection, from the detector to the control room, is called the uplink. The downlink, or the signal going from the control room to the detector, carries timing, trigger, and control (TTC) signals encoded with a 40-MHz clock which are used to control the pixel detector electronics. The downlink signal is also transmitted via a fiber and is detected by a PIN [56] photo-diode on the opto-board. The signal from the PIN photo-diode is amplified and the clock and data are decoded using the digital opto-receiver ASIC (DORIC). One per-module pair of uplink and downlink signals is called an opto-link, and on each opto-board, there are up to seven such links. To minimize the size of the opto-board, the ASICs are unpackaged and mounted directly to the board. On the opto-board, these ASICs are connected to miniaturized custom 8-channel VCSEL and PIN array optical packages called opto-packs. All told, the ATLAS pixel optical link system contains 1088 VDCs and 544 DORICs with each chip containing four channels, coupling to 544 VCSEL and 272 PIN opto-packs. Thus, in total, 272 opto-boards are present for the complete readout of the detector. A photograph of an opto-board is shown in Figure 3.1 and a block diagram of the connectivity of a single module of the pixel system is shown in Figure 3.2. More detailed descriptions of the pixel read out electronics can be found in [57] and [58].
Figure 3.1: Photographs of the top and bottom sides of an opto-board.

Figure 3.2: Block diagram of a single module of the pixel system.

3.1.1 The VDC

The VDC ASIC [58], shown in Figure 3.3, is used to convert an LVDS input signal into a single-ended signal appropriate to drive a VCSEL anode in a common cathode array. The output modulation current of the VDC is variable up to 20 mA and is set through an external control current, called Iset. The VCSEL bias current is set to a
fixed 1 mA to improve the switching speed of the VCSEL but is also adjustable externally if required. The electrical output is designed to have rise and fall times (20–80%) less than 1.0 ns. In order to minimize the power supply noise on the opto-board, the VDC is designed to consume constant current even if there is significant DC content in the data. To accomplish this, the VDC contains a dummy driver and load circuit which draws an identical amount of current from the positive power supply when the VCSEL is off as is flowing through the VCSEL when it is on.

![Figure 3.3: Block diagram of the VDC circuit.]

3.1.2 The DORIC

The function of the DORIC ASIC [58], shown in Figure 3.4, is to decode bi-phase mark (BPM) encoded data received by a PIN diode. For the pixel detector front end electronics, a 40 MHz clock is required along with configuration data having a bandwidth of 20 Mb/s. Therefore, the DORIC contains a clock and data recovery (CDR) circuit, based on a delay locked loop (DLL), with LVDS outputs for connection to the front end chips. The 40 MHz clock recovered by the DORIC is required to have a duty cycle of 50 ±4 % with a positive width jitter of less than 1 ns.
3.2 Operational Experience with the Opto-board

After five years of operation, the opto-boards generally proved to be reliable and performed well. Despite this, over time, opto-board related failures rendered nearly 4% of the detector impossible to read out. In order to avoid similar problems in future upgrades to the pixel detector, or in possible new deployments, we have developed an improved opto-board concept.

The initial design, production, and assembly of the opto-boards and the off-detector opto-electronics were completed between 2003 and 2007. Since 2008, the pixel detector and the opto-boards have been successfully taking physics data, and during the last three years data from the pixel detector used in the discovery of a Higgs-like Boson at 126 GeV [1] were transmitted though opto-boards and the off-detector opto-electronics. Unfortunately, this decade of operational experience with the optical links was not free from troubles and presented us with obstacles to overcome.
The most significant problems encountered were large numbers of premature VCSEL deaths on the off detector TTC transmitter (TX) modules for both the pixel and semiconductor tracker (SCT) detectors. After a year of investigation it was determined that the lifetimes of the particular VCSELs used in all of the ATLAS pixel and SCT opto-links were sensitive to humidity [59]. In response, all of the TX modules were replaced using VCSEL arrays from a commercial vendor with devices designed and qualified to be insensitive to humidity exposure. Thankfully, the humidity inside the detector where the opto-boards reside is nearly zero and to date, only one similar VCSEL death on opto-boards has occurred.

Unfortunately, the single failed VCSEL mentioned above is not the only failure that occurred with the opto-boards. At the end of data taking in 2013, three complete opto-boards, 10 VCSELs, and two PINs were non-functional. This corresponds to 31 modules or approximately 1.8% of the pixel detector which cannot collect physics data because of an opto-board related failure. Through a postmortem study performed during the ongoing LHC shutdown, it has been determined that all of the VCSEL and PIN failures are due to cold solder joints to the opto-packs. Among the failures, the most serious was the loss of the three complete opto-boards. In all three cases, the failures were caused by a broken or intermittent connection in the Iset line. The postmortem mentioned above discovered that the opto-boards themselves were fully functional and the break in the Iset line was located elsewhere in the service chain.

Motivated by these experiences, we have developed an improved opto-board concept. The first problem addressed in this new opto-board is to add redundancy to the
opto-board. With redundancy on the opto-board, and improved detector control software, a system of active redundancy could be implemented. With an active redundancy system in place, in the event a link failed, an automated failure detection and reconfiguration of the opto-board could occur. The second problem addressed on our opto-board concept is that there was no control of the individual VCSEL modulation currents. Having individual controls for the VCSEL modulation current reduces the likelihood of a single point failure rendering all links on an opto-board useless. The final problem addressed was to use a VCSEL and packaging that does not have an inherently short lifetime in the deployment environment.

3.3 Improved Opto-board Concept

To produce an improved opto-board enabling active redundancy and individual control over the VCSEL modulation currents, it was necessary to develop ASICs to replace the existing VDC and DORIC. Due to the compact nature of the pixel detector and the requirement that it be comprised of as little material as possible for physics reasons, it is simply not feasible to install a dedicated electrical or optical control interface to configure opto-boards. However, to implement our improvements, the opto-board requires such an interface. To accomplish this, our new DORIC, called the DORIC12, has been modified to not only recover and pass along the data and clock from the incoming TTC signals but to also process the information contained within the data. Once a valid opto-board specific command is detected, the DORIC12 then reconfigures itself or sends signals over a serial interface to configure the VDC12s on the same opto-
board. To permit individual control over the VCSEL modulation currents, 8-bit digital to analog converters (DAC) have been installed in each VCSEL driver channel. These DACs are controlled via the serial interface provided by the DORIC12.

![Block diagram of the improved opto-board prototype.](image)

Our prototype opto-board, shown in Figure 3.5, is designed to be used in a system where eight channels of each optical array are needed as dedicated links. Because commercially available optical arrays are 12 channel devices, this allows four PIN and four VCSEL channels per 12 channel array to be reserved as spares. Cross-point signal networks are contained within the new ASICs to allow switching from a default pass though between the hardwired electrical connections to the detector and the optical arrays to a state where a broken VCSEL or PIN channel can be bypassed and the hardwired electrical signal is
sent to one of the spare optical channels. These cross point networks are configurable over the remote interface scheme discussed above. With this configuration, in the unlikely event that half of the default optical channels failed, it would be possible for the opto-board to remain fully functional.

3.4 Precursors to the VDC12 and DORIC12

The original VDC and DORIC, developed in the early 2000’s, were designed using a standard commercial 250 nm CMOS process. When we initiated the development of the chipset for the new opto-board, the CERN microelectronics group and HEP community had selected a standard 130 nm process as the mainstream technology for use in LHC detector upgrades. Since we chose the same 130 nm process that was adopted by the CERN community its universal use allowed for us to gather and use blocks developed by our collaborators. Additionally, the microelectronics group at CERN had already performed radiation studies and provided us with guidelines on radiation tolerant design in the 130 nm process [60, 61]. Given these facts, the 130 nm process was the clear choice for us to use for our new chipset.

To develop 12 channel ASICs containing the complete functionality for demonstrating our improved opto-board concept we produced two rounds of test chips which contained the basic building blocks of the VDC and DORIC. In the first test chip, we performed a vertical porting [62] of many blocks from the existing 250 nm designs into the 130 nm process. In Chapter 5 we discuss the porting process and its benefits in detail. This first test chip was submitted in March 2008 and contained two versions of the
VDC and a single version of the DORIC chip capable of operation at 40, 160, or 320 Mb/s. The chip was tested in our labs and then irradiated to 70 Mrad (Si) using 24 GeV protons. This test chip was mostly successful with the results discussed in [63] and [64]. In our second prototype chip we expanded the design to test the redundancy scheme by including four VDC channels and four DORIC channels with one channel of each designated as a spare. With this configuration, and the inclusion of a remote control interface, the test chip allowed redundancy in both directions by enabling a signal to be re-routed from a bad VCSEL or PIN channel. Also included in this prototype were several IP blocks provided by collaborators. We acquired a DAC, a band gap reference, a single event upset (SEU) tolerant dice latch [65], and a command decoder. In Chapter 5 we detail the process of incorporating these IP blocks into the ASICs. In 2010, the fabricated prototype chips were tested in our labs and then irradiated to 46 Mrad (Si) using 24 GeV protons. Results from the test chip and irradiation, discussed in [66], led us to proceed in producing the 12- channel chipset.

3.5 The DORIC12

The 12-channel DORIC12, shown in Figure 3.6, Figure 3.7, and Figure 3.8, enables active redundancy for the improved opto-board concept along with individual control over the VCSEL modulation currents of the VDC12. The design of each of the channels in the DORIC12 is based on lessons learned from the 130 nm prototype fabrications discussed earlier. All 12 channels contain a transimpedance and limiting amplifier connected to a corresponding channel on a PIN array. However, only the inner
eight channels contain clock and data recovery circuits for decoding the 40 Mb/s BPM
TTC signals. The recovered clock and data signals are transmitted from the DORIC12 via
LVDS drivers. Selected circuit schematics and layouts for blocks within the DORIC12
can be found in Appendix C.

Figure 3.6: Photograph of the DORIC12 die.

Figure 3.7: Layout of the DORIC12.
In order to allow for active redundancy, 5-to-1 multiplexers have been inserted in the post amplification paths of each of the inner eight channels. If a PIN diode in one of the inner eight channels becomes non-functional, the 5-to-1 multiplexer for that channel can be remotely set to pass a signal from any of the four spare PIN channels to its clock and data recovery circuitry. This allows the TTC signal to be sent to a spare PIN and then be decoded and transmitted to the detector thus bypassing the broken PIN. To allow remote control over the channel multiplexers and the setting of the VDC modulation currents, we included the same command decoder used in the 2010 prototype chip in each of the inner eight channels. For processing the information from these command decoders we designed a logic block that was synthesized from VHDL and routed using a standard
cell library. This block is configured to allow full control of the chip even if only one PIN channel is still alive.

3.6 The VDC12

The 12-channel VDC12, shown in Figure 3.9, Figure 3.10, and Figure 3.11, enables active redundancy among the VCSELs on an opto-board and individual remote control over the VCSEL modulation currents. In addition, a global remote control over the VCSEL bias current is provided. The fabricated chip is fully functional and all 12 channels are capable of operation from DC to 5 Gb/s. Each channel has an LVDS receiver, an 8-bit DAC, and a VCSEL driver circuit capable of delivering 12 mA of modulation and 4 mA of bias current. The outer four channels are designated as spares. Multiplexer networks are connected to the LVDS receiver output of each of the inner eight channels. These networks allow the signals from any of the eight inner channels to be re-routed to the VCSEL driver of any of the spare channels. Each channel contains an 8-bit DAC for setting the VCSEL modulation current. In addition, a single 8-bit DAC is available for setting a common bias current for all of the VCSEL driver channels. The configuration of the VDC12 chip is controlled via a unidirectional three-wire serial interface provided by the DORIC12. To enable operation in case of a failure in the communication link to the DORIC12, a power-on reset circuit is included that sets the VCSEL modulation current to 10 mA upon power up. Selected circuit schematics and layouts for blocks within the VDC12 can be found in Appendix C.
Figure 3.9: Photograph of the VDC12 die.

Figure 3.10: Layout of the VDC12
As we mentioned earlier, the VDC12 is capable of operation from DC to 5 Gb/s. However, in order to achieve successful operation at this high bit rate, optical packaging to minimize parasitics, controlled impedance transmission lines feeding the VDC12, and careful PCB layout techniques were required. Photographs of the test card enabling 5 Gb/s operation are shown in Figure 3.12. The VDC12 is wire-bonded directly to a VCSEL array fabricated by ULM Photonics with a specified bandwidth of 10 Gb/s [67]. In order to improve the switching speed of the VDC12, a dedicated power and ground pad for the output of each channel is provided. By keeping the wirebonds to the power plane on the PCB, ground plane on the PCB, and the VCSEL channels themselves as...
short as possible, the wire-bond parasitics are minimized. The VCSEL itself is placed between two stainless steel guide pins that conform to the MTP® brand multi-fiber connector from US Conec Ltd. [68]. This is a challenging operation as the VCSEL must be positioned between the stainless steel pins with accuracy better than ~15 µm in order to efficiently couple the VCSEL optical power into the fiber. The 5 Gb/s LVDS input is brought to the VDC12 and the PCB through U.FL series coax connectors. The short traces on the PCB leading to the VDC12 have controlled impedances to match the 50 Ω input lines.

Figure 3.12: Photographs of the VDC12 test PCB designed for operation at 5 Gb/s.
The optical eye diagram of a representative VCSEL channel being driven by the VDC12 is shown in Figure 3.13. To introduce cross talk, all other channels in the VDC12 were active with the same pseudo-random input signal. To de-synchronize these signals, delays were introduced by varying the length of the cabling and buffer chips distributing the signals into each VCSEL driver channel. It is important to note that the bit error rate for the VDC12 is less than $5 \times 10^{-13}$ with all channels active at 5 Gb/s. From Figure 3.13 it is evident that the eye is open with all channels active or with only one channel active. Furthermore, in both cases, the VDC12 passes the optical mask test modified from 10 Gb/s to 5 Gb/s from the Institute of Electrical and Electronics Engineers (IEEE) 802.3ae standard. To provide a comparison with the performance of a superior commercial device, we tested a small form factor (SFP+) transceiver from Finisar Corporation [69]. This SFP+ transceiver [70] is specified to operate at 10 Gb/s and thus was expected to outperform the VDC12. Figure 3.14 shows a comparison of the VDC12 operating at 5 Gb/s and the SFP+ operating at the same rate with the same input source. From these measurements it is clear that the VDC12 performs worse from a jitter perspective.
Figure 3.13: Optical eye diagram of a representative VDC12 channel coupled to VCSEL. The eyes shown compare the VDC12 performance with only one channel active and with all channels active.
Figure 3.14: Optical eye diagram of a representative VDC12 channel coupled to VCSEL compared to the optical output performance of a commercial SFP+.
3.7 Voltage Regulator

The DORIC12 chip and the DAC and LVDS receivers within the VDC12 chip are designed to operate with a 1.5 V supply. However, the VCSEL driver stage within the VDC12 is designed to operate with a 2.5 V power supply. A higher supply voltage is required by the output stage to drive VCSELs which normally have threshold voltages of about 2 V. In order to allow maximum flexibility for deploying our improved opto-board concept, we have developed a PMOS low dropout (LDO) regulator to allow the opto-board to be powered by a single DC power supply service line. Our LDO’s output is adjustable via an external resistor network but has been optimized to generate 1.5 V from a 2.5 V rail. The LDO is capable of delivering 1 A at 1.5 V. The schematic and layout of the LDO can be found in Appendix C.

Figure 3.15: Performance of two LDO voltage regulators during irradiation. Plot provided by the University of Siegen.
After successful testing in our labs, we completed a dedicated irradiation of the LDO with 24 GeV protons to $1.5 \times 10^{15}$ p/cm$^2$. This irradiation was performed with alternating periods of the beam being switched on and off. During the first beam on period of 20 hours, there was a decrease of the output voltage magnitude. The observed decrease was in part compensated, as shown in Figure 3.15 for a sample of two regulators, by increasing the input voltage from 2.50 V to 2.54 V. The output voltage level changed accordingly from 1.50 V to 1.46 V. The remaining four beam-on periods were between five and seven hours of duration and the corresponding beam-off periods were 20 hours long each. The result of this irradiation test shows a steady output voltage level which did not degrade any further than the initial 40 mV drop.

3.8 Improved Optical Packaging

Of the individual failed channels in the optical links of the ATLAS pixel detector, all but one was caused by cold solder joints between the opto-board and the VCSEL and PIN opto-packs. We therefore developed a new optical package that is wire bonded rather than soldered to the opto-board [71]. In the complete development history, we have produced 321 VCSEL and 161 PIN opto-packs. As shown in Figure 3.16, the yield has been high, nearly 90 % successes for the VCSEL and 95 % for the PIN opto-packs. Furthermore, we are consistently successful in achieving an acceptable fiber-coupled VCSEL optical power of 2.5 mW on average with a standard deviation of 0.6 mW, and a PIN responsivity of 0.69 A/W on average with a standard deviation of 0.06 A/W, shown in Figure 3.17.
Figure 3.16: Yields of the improved VCSEL (a) and PIN (b) array opto-packs.

Figure 3.17: Fiber coupled optical power from all VCSEL (a) and responsivity of all PIN (b) channels after mounting in the improved opto-packs.
Over the course of many years we learned the best way to handle and package VCSELs in order to maximize their lifetime. In this process, we realized that the original VCSELs and packaging had major deficiencies. As mentioned earlier, our first discovery was the fact that the original VCSELs themselves were sensitive to humidity. This problem was resolved by selecting modern VCSEL products from a different vendor that have been specifically designed to be humidity resistant. The second problem was discovered after premature failures started to occur upon testing the humidity tolerant VCSELs in accelerated lifetime studies of our improved optical package. The VCSEL vendor provided a qualification report showing no observed failures in arrays from many wafers tested in a damp heat environment of 85°C and 85% relative humidity for up to 1,000 hours. Thus in the accelerated lifetime experiments of our improved optical package we attempted to show that our packaging did not cause the VCSELs to fail before reaching 1,000 hours. In the damp heat test, all VCSELs were powered continuously with a DC bias of 10 mA and the tests were performed with spring loaded MTP connectors in place. By using the MTP connectors, a continuous force was applied to the optical packages identical to what would occur during a deployment of the optical package.

As a result of these tests we learned that covering the VCSEL arrays [72] with optical epoxy [73] adversely affects the lifetime of the arrays. Out of 20 opto-packs tested having optical epoxy in place, 14 failed after only 250 hours in our environmental chamber. Upon running a sample of 20 VCSEL arrays not covered in optical epoxy, we found that 236 out of the 240 individual channels survived beyond 2000 hours. One of
the failed channels had clear mechanical damage which should have been discovered in a visual inspection prior to starting the test and the other failed channels were caused by problems with our test setup. By comparing these results to the qualification report provided by the VCSEL vendor we demonstrated that our improved optical package with no optical epoxy maintains the inherent lifetime of the VCSELs.

The assembly of opto-boards using opto-packs with no optical epoxy covering the VCSEL and PIN arrays have two complications. First, the opto-packs need to be handled with more care as there is no epoxy to protect the wire bonds. Second, there is a risk that dust might fall directly on the light emitting aperture of the VCSELs (~15 µm) resulting in much reduced coupled optical power. To prevent dust collection we developed a shielding cover (opto-shield) shown in Figure 3.18 and Figure 3.19. The design of the opto-shield encloses the VCSEL or PIN in a cavity that is sealed when a fiber connector is inserted. The opto-shield, made of type 316 stainless steel, is fabricated using a photochemical etching process [74]. We glue the opto-shield to the opto-pack once a VCSEL or PIN array has been mounted and wire bonded. Through the use of half etching techniques, we leave a gap of 125 µm between the overhang of the opto-shield and the opto-pack to allow the wirebonds to pass under the MT-ferrule sealed package face. Additionally, the half etched design forms a gap to keep the traces on the opto-packs from being shorted out along the corner of the opto-pack. Furthermore, the opto-shield defines a minimum gap between the optical array surface and the MT-ferrule face. We have successfully constructed several samples of VCSEL and PIN opto-packs built using
the opto-shield and have seen no impact on the fiber coupled optical power or responsivity.

Figure 3.18: Drawing of the opto-shield mounted on an opto-pack, (a) front view and (b) cross sectional view revealing the half etch.

Figure 3.19: Photographs of the opto-shield mounted to an opto-pack: (a) VCSEL / PIN and wire bond cavity, (b) MT ferrule in place sealing the front of the cavity, (c) gap in opto-shield to prevent channel shorting to be filled with encapsulant for cavity sealing.
3.9 The Complete Opto-board Prototype

With the successful implementation of the constituent parts described above, we proceeded to design and construct complete prototype opto-boards to demonstrate the possibility of active redundancy and individual control over the VCSEL modulation currents. As discussed in Section 3.3, our prototype opto-board is designed to be used in a system where eight channels of each optical array are needed as dedicated links thus allowing four PIN and four VCSEL channels per 12-channel array to be reserved as spares. On our prototype opto-boards, 16 DTO lines and 8 TTC lines are available, identical to what is shown in Figure 3.5. To implement this scheme each opto-board must contain two VDC12 chips, two VCSEL opto-packs, one DORIC12 chip, and one PIN opto-pack. Additionally, we included our voltage regulator, configured to generate 1.5 V from 2.5 V, to allow operation with a single input power supply. The opto-board itself is a six-layer board designed using standard PCB technology which has been adhered to a 1 mm thick copper backing plate for thermal management. A photograph of a completed opto-board is shown in Figure 3.20.
We have produced five complete and fully functional opto-board prototypes that enable active redundancy and programmatic setting of the individual VCSEL bias and modulation currents. With the VCSEL modulation and bias currents set to 12 mA and 4 mA respectively and an input supply voltage of 2.5 V, the total current consumption for each assembled board is ~950 mA. We are able to successfully configure all DACs within the VDC12s remotely by communicating optically with the command decoders in
the DORIC12. Additionally, all possible combinations of rerouting to and from the spare channels have been studied in both the DORIC12 and VDC12 at the nominal bit rates of 40 Mb/s TTC and 160 Mb/s DTO. We have seen no degradation in the bit error rate (BER) performance of the channels following reconfiguration. The test was running error free for at least one hour under each rerouted configuration with $2^{23}$ pseudo-random (PRBS) data, yielding a BER of less than $6.9 \times 10^{-12}$ for the TTC links and $1.7 \times 10^{-12}$ for the DTO links. Furthermore, these tests confirm that the use of our voltage regulator ASIC under the nominal load of a fully populated board does not diminish the performance of the DORIC12 or VDC12. The most sensitive figure of merit related to power supply noise, the lower PIN current threshold for no bit errors in the DORIC12, is less than 75 µA peak to peak (pk-pk) on the completed boards with all other links running error free at the specified bit rates. This is well below our typical system level specification for the PIN amplifier sensitivity of 100 µA.

To fully qualify our opto-board prototype, we irradiated three complete opto-boards with 24 GeV protons at CERN to $1.5 \times 10^{15}$ p/cm$^2$. Because we have already performed studies on the radiation hardness of the VCSEL [72] and PIN [75] arrays chosen for use on the opto-boards [76], our test was focused mainly to study the behavior of the ASIC chips. To study the ASIC performance and SEU tolerance, we sent BPM encoded signals over 30 m of optical fiber from the control room to the beam area where the opto-boards were installed. These fibers were attached to the PIN arrays on the opto-boards and the signals were in turn decoded by the DORIC12 chips. Through wiring on a host PCB, the LVDS outputs of the DORIC12 were routed back to the VDC12 chips on
the opto-boards. With this configuration, the recovered clock and data were sent by the VDC12 and VCSELs over 30 m of optical fiber back to our test electronics in the control room for comparison against the data that was generated.

Upon setting up this test system and placing the opto-boards in the beam, we discovered a serious flaw in the control logic of the DORIC12. For a valid command to be received in the command decoders found in each channel and interpreted in our control logic, a pre-defined sequence of 23 header bits in the incoming data must be present. We observed that the probability for the noise to generate a valid command was much higher than expected. In the case of a non-illuminated PIN channel, spurious commands were observed a few times per minute. As we increased the number of non-illuminated channels, the rate of erroneous commands increased. This effect was due to the fact that without an input, the DORIC12 simply amplifies noise and sends it to the clock and data recovery circuitry. We knew of this effect before designing our command acceptance logic but greatly underestimated the likelihood of the occurrence of the 23 header bits in the amplified noise. In retrospect, this should have been anticipated. The DORIC12 can potentially generate $2^{23}$ random bits in 0.21 s assuming each bit has the proper signal shape for a 40 MHz clock. It is therefore quite feasible to generate several valid signals per minute. Given this rate of erroneous commands re-configuring the DORIC12 and VDC12s on the opto-boards, it was nearly impossible to perform a quantitative survey of their SEU tolerance. Based on this experience, we have worked to design improved command acceptance logic that will only respond if identical commands are received by at least three channels within a limited time window.
Despite the flawed command acceptance logic, we completed the irradiation of the three boards to check for other failure modes. The boards survived the irradiation and after a cool down period were shipped back to our labs for further tests. Upon receiving the boards we found that the lower PIN current threshold for no bit errors in the DORIC12 was maintained at less than 75 µA pk-pk with all other links running error free at the specified bit rates. Furthermore, all possible combinations of rerouting to and from the spare channels in both the DORIC12 and VDC12 at the nominal bit rates, 40 Mb/s TTC and 160 Mb/s DTO respectively, had undiminished BER performance. Finally, we investigated the quality of the optical eye-diagrams from all VDC12 channels with all possible rerouting combinations. The worst case eye-diagram, shown in Figure 3.21, occurring on a spare channel being re-routed over the full length of the chip with all channels active at 160 Mb/s is still perfectly acceptable after irradiation.
Figure 3.21: Post-irradiation optical eye from the VDC12 spare channel at 160 Mb/s.
Chapter 4: The Hitbus Chip for the ATLAS DBM Detector

As introduced in Chapter 2, the DBM detector is a new subsystem of the ATLAS experiment. In order to maximize the functionality of the DBM, and to utilize all of the available services, a scheme was developed to insert a radiation hard ASIC, the Hitbus chip, within the DBM to collect trigger information from the detector modules, process this information, and provide the capability for the DBM detector to trigger its own readout without communicating with the ATLAS trigger system. In this chapter we will detail the design, results from bench tests, and an irradiation with 24 GeV protons to $4.33 \times 10^{15}$ p/cm$^2$.

4.1 Hitbus Chip Rationale

Since the DBM is based on using the same front end chips as the IBL upgrade of the ATLAS pixel detector, the FE-I4, it was decided to use the same power supply and readout services that had been developed for the IBL. To distribute the readout signals and power supply services to the DBM, two printed circuit (PCB) cards named Patch Panel Zero (PP0) boards are placed near the DBM modules. Due to the arrangement and quantity of available services, there are two PP0 boards on each side of the ATLAS interaction point and four in total. A block diagram of the ATLAS DBM detector is shown in Figure 4.1. Once all services and signals are delivered to and from the DBM
modules, four spare data output (DTO) signal links and two spare timing, trigger, and control (TTC) signal links are available on each side of the interaction point. Given the available spare services, a solution has been implemented to provide a mechanism for the DBM to trigger its own readout.

4.2 DBM Triggering and the Origins of the Hitbus Chip

During normal acquisition, the FE-I4 chips in the DBM are only read out when the ATLAS Central Trigger Processor (CTP) issues a Level-1 Accept (L1A) [77]. However, since the DBM aims to provide online bunch-by-bunch relative luminosity and beam spot information, the relatively slow maximum L1A rate, 75 kHz, severely limits the sampling rate of the detector. While software algorithms would allow a faster readout of the DBM by partially circumventing the CTP, the most optimal solution to readout the DBM is to allow it to trigger itself. Conveniently, a non-return-to-zero (NRZ) single ended output is
provided by the FE-I4 chip called the HitOr signal. This HitOr signal is a wired OR of the
trigger outputs of all the enabled pixels in the FE-I4. By selecting a region of interest in a
module the HitOr can be used to detect if a particle was incident anywhere within that
region. By performing logic on the three HitOr signals from a telescope it is then possible
to discern quickly whether a track passed through a defined acceptance region in the three
modules to form a trigger. To realize a self-triggering DBM, the spare DTO lines in the
DBM services are used to send the HitOr signals from each module to the read out data
(ROD) acquisition system. The quantity of HitOr signals to be sent, 12 per side, and the
single-ended nature of the signals make it impossible for the FE-I4 to directly deliver the
HitOr over the ~80 m distance to the ROD. Therefore, the use of an ASIC, the Hitbus
chip, is required to process and buffer the HitOr signals. Due to the TID expected at the
DBM’s location, 2x10¹⁵ 1 MeV nₑₑ/㎝², including a safety factor of two, it is necessary
for the Hitbus ASIC to be radiation hard.

4.3 Overview of the Hitbus Chip

The connectivity for one quarter of the ATLAS DBM detector is shown in Figure
4.2. The Hitbus chip, situated on PP0, provides two main functions. The first function is
collecting trigger information from the six HitOr outputs of the two DBM telescopes, A
and B, for delivery to the off-detector electronics. After receiving the 1.5 V CMOS HitOr
signals, the Hitbus chip contains a control block to perform logical operations, serialize,
and transmit the hit information. The HitOr processor block, within the control block, is
designed using VHDL and a standard cell library for synthesis and automated placement
and routing of the layout. The HitOr processor block contains duplicate logic to serve each telescope separately.

Figure 4.2: Block diagram of the signal connectivity for one quarter of the ATLAS DBM detector. The Hitbus chip is situated on PP0.

The result of the selected logical operation per-telescope is sent at up to 160 Mb/s to the off detector electronics on two separate outputs named DTOTA and DTOTB for telescopes A and B respectively. These outputs are sent via low-voltage differential signaling (LVDS) over ~6m of twisted-pair cable to an opto-board which then repeats the signals over ~80 m of optical fiber. The second function of the Hitbus chip is to buffer
and distribute the 40 Mb/s TTC signals used for configuration of the DBM modules.
Each Hitbus chip receives four such TTC lines which contain a clock (XCK) and data
(DCI) signal pair. These clock and data pairs are transmitted to the Hitbus chip via LVDS
over ~6 m of twisted pair cable from a fiber-optic transceiver which receives the TTC
signals via a ~80 m optical fiber from the off-detector DBM electronics. One TTC pair is
used for remote configuration of the Hitbus chip. To allow configuration by the same off-
detector electronics as the DBM modules we use a command decoder identical to the one
in the FE-I4. The three remaining TTC pairs are reserved to configure the DBM modules.
The Hitbus chip actively buffers and distributes these three TTC signal line pairs to the
six DBM modules it services. A programmable matched delay is inserted within each
buffered TTC to permit precise control over the time that configuration data is sent to the
DBM modules. A block diagram of the entire Hitbus chip is shown in Figure 4.3 and a
block diagram of the control block is shown in Figure 4.4.
Figure 4.3: Block diagram of the Hitbus chip.

Figure 4.4: Block diagram of the Hitbus control block. Cells acquired from collaborators are highlighted in yellow.
4.4 Hitbus Chip Functionality

The Hitbus chip can be configured to provide several logical combinations of the HitOr inputs, or it can optionally provide a sampled version of their raw time sequence. The Hitbus operations (per telescope) are listed below:

- pass only the HitOr signal from module 1, 2, or 3
- AND the HitOr of all modules
- OR the HitOr of all modules
- majority vote on the HitOr of all modules
- select which HitOr modules to ignore in an AND
- select which HitOr modules to ignore in an OR
- serialize and transmit the three HitOr signals
- serialize and transmit the three HitOr signals with the addition of a selectable marker bit
- serialize and transmit the three HitOr signals with the addition of two selectable marker bits
- serialize and transmit the function register of the other telescope (for debugging)

The result of these logical operations is a non-return to zero (NRZ) signal. Depending on the configuration, the Hitbus chip NRZ data is almost entirely DC which cannot be correctly received by the backend optical receiver electronics that are AC coupled.
Therefore, we have added an option to apply Bi-phase mark (BPM) encoding to the outputs of the Hitbus chip.

4.5 Clock Configuration

An important feature of the Hitbus chip is the inclusion of a phase-locked loop (PLL) with frequency dividers for generating higher speed clocks from the incoming 40 MHz TTC clock. The purpose of the higher speed clocks is to serialize and transmit the HitOr signals fast enough to provide a reconstruction of the time history of the received HitOr signals for each proton bunch crossing in ATLAS. These bunch crossings take place every 25 ns. With a raw stream of the three HitOr signals alone it would be impossible to determine which signal is which. Therefore, we insert a marker bit before the first HitOr signal in the sequence to allow alignment of the receiver. Given this configuration, sending a marker bit and then the three HitOr signals, transmitting at 160 Mb/s yields one sample of all HitOr signals per 25 ns. To send BPM at 160 Mb/s requires a 320 MHz clock. Our PLL is designed to produce an output of 640 MHz with divider taps inserted at 320 MHz, 160 MHz, and 80 MHz. In order to be compatible with unforeseen scenarios, we have included the possibility to remotely select the clock speed of the Hitbus chip processor and BPM encoder. The available combinations allow a maximum bit rate of 160 Mb/s BPM down to a 20 Mb/s BPM rate which operates on the TTC clock alone in case of a failure in the PLL.
4.6 Design of the Hitbus Chip

Many of the circuit blocks contained in the Hitbus chip were collected from collaborators in order to accelerate the time during which the design could be completed. In Chapter 5 of this dissertation we will discuss in detail the experiences, lessons learned, and techniques developed for incorporating such IP blocks. However, in this chapter we focus on the new custom blocks designed for the Hitbus chip, a programmable delay, the HitOr control block, and an SEU hard PLL. Figure 4.5 shows a photograph of the Hitbus chip die and an image from the design environment highlighting the main components within the chip.
4.7 The Programmable Delay

As mentioned above, a programmable delay is inserted within both the TTC clock and data paths to permit precise control over the time that configuration data arrives at the DBM modules. The programmable delays have a resolution of seven bits and are required to provide a maximum delay of at least 5 ns with the Hitbus chip powered at its nominal supply voltage of 1.5 V and operating at 23° C. The designs of the delays themselves were completed using VHDL, synthesis, and PNR. The VHDL code can be found in Appendix A. We used structural VHDL to build the delays which contain strings of inverters with binary weighted lengths. By adding a multiplexer network, the strings of inverters can be switched in and out to achieve the desired delay setting. In order for the
inverter strings to exist post-synthesis, the synthesis tool had to be forced to not optimize the logic. Due to the layout, supply voltage, and temperature dependent nature of this design, to guarantee the specified 5 ns maximum delay required simulation of a fully extracted layout with parasitics to be remotely correct. In order to achieve the 5 ns under the normal operating conditions, the synthesis, PNR, parasitic extraction, and simulation was run many times using various inverter and buffer strengths available in the standard cell library. It was discovered that all of the buffer varieties were too fast and power hungry, so we settled on using one of the weaker inverters available in the library. We achieved a simulated delay of ~9 ns under the nominal power and temperature conditions. Following this, corner simulations were run using varying power levels to ensure that the delay would meet the specification in extreme cases. We were satisfied when we achieved 5 ns and 1.6 V at the fast-fast corner and 9 ns and 1.2 V at the fast-fast corner. As will be discussed in the quality assurance testing section below, the delays of the fabricated chips met the specification.

In order to ensure that the delays remain static in a high radiation environment, a seven bit register based on the SEU hard latch described in Chapter 3 was used to store the delay setting. Additionally, in case of a problem with the delay, we incorporated a bypass path to send the TTC signals directly around the delay. A block diagram of the delay cell and an image of the layout are shown in Figure 4.6.
4.8 The HitOr Control Block

The HitOr control block is responsible for performing logical operations on the HitOr signals it services, processing commands received by the FE-I4 command decoder, and to send the result of the logical operation using BPM line code to the off-detector electronics. The control block contains a logic core designed using VHDL, synthesis, and PNR. The VHDL code for the control block can be found in Appendix A and the synthesis and PNR scripts can be found in Appendix B. It is important to note that the foundation of these synthesis and PNR scripts was developed by the CERN Microelectronics group. The logic core contains BPM encoders, serializers, and counters. Thus, after synthesis, it contains sequential logic. To enhance the control block’s hardness against SEU, all sequential logic blocks synthesized from standard cell libraries.
were triplicated with majority voting. Like in our delay cell discussed previously, we modified the synthesis scripts so that the redundant logic was not removed during optimization steps. Because configuration from the off-detector electronics only occurs every few hours, the chosen configuration including the selected logic function, delay settings, BPM encoding enable, and clock rate must be stored in memory. These memories are designed using the DICE latches we acquired from a collaborator [65].

The maximum clock speed for the control block is 320 MHz. Mixed signal simulations of the block using functional representations of the synthesized logic worked well. However, simulations of the extracted layout with the block connected to its peripherals did not operate properly. The failures were caused by timing violations. To remedy this, timing constraints were applied to the block in both synthesis and PNR. Additionally, due to the parasitic loads on the input and output lines from the control block, strong buffers were placed on the timing critical pins during synthesis and PNR. After these actions were taken, simulations of the extracted control block connected to its peripherals were successful.

4.9 The SEU Hard PLL

As mentioned earlier, the Hitbus chip contains a PLL for frequency multiplication of the TTC input clock which operates at 40 MHz. The PLL provides a maximum multiplication of 16 times the input clock frequency yielding an output of 640 MHz. Figure 4.7 shows the block diagram of the PLL implemented in the Hitbus chip. In the quality assurance testing section of this chapter, we will discuss the satisfactory
performance of the PLL in the fabricated chips. Circuit schematics and layouts for the PLL can be found in Appendix C.

Figure 4.7: Block diagram of the PLL implemented in the Hitbus chip.

Using Figure 4.7, the open loop transfer function of the Hitbus chip PLL is defined by:

\[ G(S) = K_p K_f K_{VCO} \]  \hspace{1cm} (4.1)

Where \( K_p \), \( K_f \), and \( K_{VCO} \) represent the gain constants of the phase frequency detector (PFD), loop filter, and the voltage controlled oscillator (VCO) respectively. The PFD circuit, based on concepts from [78], is implemented with full custom sequential logic. The PFD is used for detecting the frequency and phase differences of the input and output clocks. The PFD is realized using two edge-triggered resettable D-type flip flops. The 40MHz TTC clock and the reference clock, the 16 times divided output of the voltage
controlled oscillator (VCO), are fed to the clock inputs of the two D flip flops. An AND gate connected to the outputs of the two flip flops is used to reset the flip flops when a phase difference is detected. The closed loop transfer function of the PLL is:

\[ T(s) = \frac{G(s)}{1 + G(s)H(s)} \]  \hspace{1cm} (4.2)

Where \( H(s) \) is the selected dividing factor within the clock divider block:

\[ H(s) = K_n = \frac{1}{N} \]  \hspace{1cm} (4.3)

The loop filter is a second order passive RC filter with an impedance of:

\[ K_s = \frac{1}{C_2 \left( s + \frac{1}{RC_1} \right)} \frac{s^2 + \frac{s(C_1+C_2)}{RC_1C_2}}{s^2 + \frac{s(C_1+C_2)}{RC_1C_2}} \]  \hspace{1cm} (4.4)

The resistor is included in order to provide stability to the overall feedback system, the complete PLL, by introducing a zero in the loop gain. The introduction of this resistor in the loop filter reduces the phase shift to less than 180° thus stabilizing the PLL. A second capacitor is inserted in the loop filter to reduce the ripple present in the control voltages fed to the VCO. These ripples disrupt the function of the VCO and consequently introduce errors in the output frequency of the PLL. The chosen value of the resistor and capacitors for the second order RC loop filter are as follows: \( R = 1009 \, \Omega; \, C_1 = 5.36 \, \text{pF}; \, C_2 = 0.625 \, \text{pF} \). The value of \( C_2 \) is selected to be approximately one-tenth of \( C_1 \) to smooth
the VCO control voltage ripple while ensuring that the PLL remains stable. For the charge pump and loop filter combination, the gain is defined as:

\[
K_p K_f = I_p \left[ (R + \frac{1}{sC_1}) || \frac{1}{sC_2} \right]
\]  

(4.5)

Where \(I_p\) is the current provided by the charge pump. The charge pump, designed using approaches described in [78], is located between the PFD and the loop filter. It consists of two switched current sources that pump charge into or out of the loop filter based upon the logical outputs received from the D flip flops. The switches are located at the drains of complimentary current mirror transistors. When both the outputs of the D flip flops are low (00) no current is provided by either of the current sources to the charge pump and thus the output voltage of the charge pump remains constant. The capacitor of the loop filter is charged and discharged through the charge pump for a sequence corresponding to (10) and (01) of the outputs of the two D flip flops respectively. \(I_p\) was designed to be \(~50 \mu A\).

The VCO used in the design, similar to [79], is a ring oscillator constructed with current starved inverters. The VCO output frequency is proportional to the amount of current flowing through each of the inverter stages. The VCO configuration selected requires two control voltages, one for the NMOS transistor and the other for the PMOS transistor in the current starved inverter. For this reason, duplicate charge pumps and loop filters were inserted in the PLL which receive complimentary versions of the PFD output to in turn produce the complimentary bias voltages necessary. \(K_{vco}\) is the
relationship between the output frequency of the PLL and the input voltage provided by the charge pump. Simulations of the extracted layout at the nominal process corner, temperature, and supply voltage yield a \( K_{vco} = 2.24 \times 10^4 \) Radians/s/V.

In order to allow variable frequency outputs, a switching network connecting to taps in the frequency divider network in the PLL feedback loop was included. For SEU hardness, all flip-flops included in the frequency divider network were triplicated and their results compared in a majority vote for a final output. To speed the design, the frequency divider network and output switching network were implemented using VHDL, synthesis, and PNR. The VHDL code can be found in Appendix A.

### 4.10 Hitbus Chip Quality Assurance and Testing

The entire DBM detector requires only four Hitbus chips. Due to their buffering of the TTC signals to configure the modules, the detector could not operate without these four chips. Given their importance to the DBM system, the Hitbus chips and packaging destined for use in the detector were subjected to a full quality assurance (QA) regimen. Following the wire bonding and packaging of the Hitbus die the chips underwent a quick test to verify complete functionality. The test system for the Hitbus chips, shown in Figure 4.8, was designed to be used both in lab testing and at the irradiation facility [80]. To simplify the design and construction, the base system was designed to support only one chip. However, as the need was foreseen to scale to multiple chip tests, the system was designed so that copies of the base system could be fabricated and then read out by a single data acquisition PC via the Universal Serial Bus (USB). The base system consists
of two standard technology 4-layer PCBs. The first board, called the motherboard, houses a USB-FPGA Module 1.11 from Ztex GmbH [81] for generating, receiving, and checking the HitOR, TTC, and DTO signals. The Module 1.11 contains a Xilinx Spartan 6 LX25 FPGA and a USB 2.0 microcontroller for interfacing to a PC. We chose to use this module to reduce the development and construction time for our system. Since the Module 1.11 was pre-built and tested it enabled us to simply plug the module in to our motherboard and start debugging our VHDL firmware. This provided a considerable time savings since the effort to incorporate a complicated Spartan-6 LX25 on our motherboard would not have been insignificant. Furthermore, since the interface chip for USB 2.0 communication was already integrated on the Module 1.11 and sample firmware for performing USB transfers was made available by Ztex, developing a working communication link between our data acquisition PC, running LabVIEW, and the Spartan-6 was straightforward. Additional reasons we opted to use the Ztex module over modules available from other vendors were its compactness, price, and the number of differential pairs that were brought out to its general purpose I/O connectors. In addition to the Ztex module, the motherboard contained two ribbon cable connectors for connection to the second PCB in the test system and a power supply consisting of several voltage regulators for powering the Hitbus chip, Module 1.11, LVDS receivers, and LVDS drivers. Finally, the motherboard provided an interface to the USB-6216 data acquisition card from National Instruments which we used to acquire analog information about the Hitbus chip such as the supply voltage and current. The second PCB, called the daughterboard, houses the Hitbus chip, LVDS termination resistors, power supply
decoupling capacitors, and ribbon cable connectors for connection to the motherboard. The daughterboard was designed to be compatible with a directly soldered Hitbus chip or with a zero insertion force burn in socket. With a zero insertion force (ZIF) socket, many chips can be tested and retested using the same board. Additionally, installation and removal of the Hitbus chips is simple and requires only a screwdriver. This is especially beneficial if the chip under test is destined to be re-used for another purpose.

Figure 4.8: Photograph of the Hitbus chip test system.

After the basic functionality test, the Hitbus chips were burned in at their nominal supply voltage of 1.5 V at 50° C for 72 hours while operating under a full load with a test
system sending pseudo-TTC information. During the burn in, the supply current was monitored and a continuous bit error test on the sent and received data was performed. Following burn in, the chips were thermally cycled in an environmental chamber. The thermal cycle consisted of ten 30 minute ramps between -25° C and +50° C with a one hour pause at each -25° C point. Once the burn in was completed, the chips underwent detailed QA measurements to verify their complete functionality prior to delivery to CERN. For the DBM installation a total of eight chips were subjected to this battery of tests and all passed. A photograph of one of the delivered Hitbus chips soldered to a PP0 board at CERN is shown in Figure 4.9.

![Hitbus Chip](image)

Figure 4.9: Photograph of a packaged Hitbus chip mounted to a PP0 board.

The first measurement performed as a part of the QA was a measurement of the power supply current with the chip completely operational at the highest clock rate of 320 MHz. The results from the eight production chips are listed in Table 4.1. Next, the quality of the LVDS driver output were checked. The quantities of interest are the differential
amplitude, common mode level, rise, and fall times. Histograms of these quantities are shown in Figure 4.10, Figure 4.11, Figure 4.12, and Figure 4.13. The measured results of the LVDS driver quality were sufficient for the DBM system. Subsequently, the operation of the PLL was verified. Because the direct output of the PLL was not brought out to pads on the chip, the measurement was done indirectly. By switching on the BPM encoder and sending all ‘1’ to its input, the output would be half the rate of the logic core clock. For example, in the case that the logic core is running at 320MHz, the BPM all ‘1’ condition is a 160MHz square wave. Besides performing the simple checks that the output frequency was correct, the peak-to-peak jitter of both the output period and positive width were checked. The worst case peak-to-peak period jitter observed at 160 MHz is only ~3 %. The worst case peak-to-peak positive width jitter measured at 160 MHz is only ~5 %. Due to the nature of the BPM encoder design, the positive width exhibited a worse performance. However, both of these measurements were perfectly acceptable for the DBM. Another measurement performed on each chip is to test the functionality of the TTC delays, their matching, and to check the maximum value at the nominal supply voltage of 1.5 V. The delay functionality test was performed by simply verifying that the delay can be set to arbitrarily chosen values. To check the matching, the delays were set to their minimum and then maximum values, recorded in both cases, and then the TTC and clock delays were compared. As a result of this measurement we also checked that the maximum delay was sufficient for each channel. Histograms of the delay matching of TTC pairs and the maximum delay of the TTC outputs from the eight production Hitbus chips are shown in Figure 4.14 and Figure 4.15 respectively. These
results were quite satisfactory. The final measurement performed on each chip was to check that all of the possible HitOr functions work as expected. Since there were many possible combinations of HitOr signal input vs. selected logic operation, a LabVIEW routine was used to automatically scan through the parameter space. All chips tested passed this functionality test.

<table>
<thead>
<tr>
<th>Chip #</th>
<th>2211</th>
<th>2214</th>
<th>2216</th>
<th>2217</th>
<th>2218</th>
<th>2219</th>
<th>2221</th>
<th>2223</th>
</tr>
</thead>
<tbody>
<tr>
<td>IVDD (mA)</td>
<td>73.2</td>
<td>74.3</td>
<td>79.5</td>
<td>72.3</td>
<td>71.7</td>
<td>71.3</td>
<td>75.0</td>
<td>77.0</td>
</tr>
</tbody>
</table>

Table 4.1: Measured supply currents of the production Hitbus chips.

![Figure 4.10: Measured LVDS driver differential output amplitudes of the production Hitbus chips.](image)
Figure 4.11: Measured LVDS driver common mode levels of the production Hitbus chips.

Figure 4.12: Measured LVDS driver rise times of the production Hitbus chips.
Figure 4.13: Measured LVDS driver fall times of the production Hitbus chips.

Figure 4.14: Measured difference between the delay of the XCK and TTC for every matched delay pair in the production Hitbus chips.
4.11 Hitbus Chip Irradiation Results

A critical test to certify the Hitbus chip for its installation within the intense radiation environment of the ATLAS detector was to perform an irradiation. Given the chip’s distance from the interaction point of ATLAS, we irradiated two chips to $4.3 \times 10^{15} \text{ p/cm}^2$ or 115 Mrad (Si) with 24 GeV protons at the CERN PS irradiation facility [80]. To characterize the chips during the irradiation, they were completely active with all TTC channels receiving 40 MHz clock and $2^{23}$ pseudo-random (PRBS) data at 40 Mb/s. The irradiation zone is far removed from the control room of the irradiation facility, described in Chapter 2, for safety reasons so we sent and received all LVDS signals from the Hitbus chips on ~25 meters of twisted pair cable. This cable length limited the operating speed
of the HitOr outputs to 40 Mb/s. During the irradiation we monitored the buffered TTC outputs checking for any data or clock errors. We also sent HitOr signals to the chip in order to check proper functionality as the irradiation progressed. The supply current consumption of both chips was recorded throughout the irradiation. Both chips experienced a slight but acceptable increase and can be seen in Figure 4.17. Additionally, we studied SEU induced upsets in the HitOr control block. The results observed from the irradiation are encouraging. Over 20 days of running, only 427 and 236 upsets requiring reconfiguration of the HitOr control block occurred in chip 1 and chip 2 respectively. For chip 1, this corresponds to an SEU cross section of $\sim 4 \times 10^{-15}$ cm$^2$. Knowing that the peak particle flux expected at PP0 is $\sim 1 \times 10^8$ particles/cm$^2$/sec. [65] yields a maximum rate of only $\sim 12$ SEU induced reconfigurations per year. Figure 4.16 shows the number of reconfigurations required per chip as the dose increased. The stair step appearance was caused by the fact that there were periods of time when the chips were held slightly out of the beam to receive a slower dose and periods where the chips were completely removed from the beam.

The two Hitbus chips survived the irradiation and were completely operational upon removal from the beam. Furthermore, we verified that the error performance was better than $1.5 \times 10^{-13}$ for sending and receiving TTC data at 40 Mb/s on all channels. These checks were completed before dismantling the setup at the irradiation facility. We are awaiting delivery of the irradiated chips after a cool down period in order to measure the post-irradiation behavior at higher bit-rates along with the quality of the LVDS drivers and receivers.
Figure 4.16: Observed SEU induced function register reconfiguration events in both irradiated Hitbus chips.

Figure 4.17: Observed supply current behavior of both Hitbus chips during the irradiation.
Chapter 5: Reused Design Integration, IP Sharing Guidelines, and a Concise IP

Sharing Template

As described in Chapter 3 and Chapter 4, we have successfully designed and implemented new custom analog and digital CMOS circuit blocks for use in radiation hard ASICs for the ATLAS experiment. Alongside the new blocks, the ASICs contain existing blocks from our libraries that were converted from a different process node to our target process node. Additionally, the ASICs include IP blocks. As was stated in Chapter 2, an IP marketplace for radiation hard designs does not exist and thus the IP blocks we used were shared with us by collaborators. In this chapter we discuss migrating legacy blocks from one process node to another and detail our experiences including collaboratively shared IP blocks in our ASICs. Also, we recommend guidelines for integrating and verifying mixed signal ASICs which include reused designs. Furthermore, we propose a template for IP sharing.

5.1 Design Reuse: Technology Migration

The operation of reusing IP targeted for one process node and implementing it in another process node is referred to as technology migration or porting. By porting a circuit, the design cycle in the new process node can be significantly shortened.
Furthermore, porting enables earlier fabrication and provides more time to focus on the design and verification of new blocks. There are two distinct directions possible in the porting of a design: horizontal and vertical. In horizontal porting, a design is converted to a process with the same minimum transistor feature sizes but from a different foundry. In vertical porting, a design is converted to a process with smaller or larger minimum transistor feature sizes [62]. Since the design libraries of the original DORIC and VDC that were fabricated for the first generation of the ATLAS pixel detector readout were still available, we chose to port many cells for use in the DORIC12 and VDC12. These legacy designs were completed in a 0.25 µm process and the original designers were unfortunately not present to assist in the porting process. Through employing the flow described in the next section, and illustrated in Figure 5.1, we were able to successfully reuse many legacy cells by performing a vertical port from the 0.25 µm to the 0.13 µm process. Due to the limited personnel resources available and the differences between the target and source process node DRC sets, porting was only used for the schematic level. The legacy layout topologies were studied but no attempt to directly scale the existing layout was attempted.
5.1.1 Technology Migration Flow

Foremost, to arrive at a functional design in the target process requires a framework of design content from the legacy design, the legacy process, and the target process. The most critical part of this framework, illustrated in Figure 5.2, is knowledge of the topology of the legacy design. Included in this topology is the architecture of the transistor level schematics. Without these schematics, the porting exercise cannot proceed. Given the legacy design schematic, it is most efficient to employ the same architecture as the baseline in the ported circuit. This is due to the simple fact that if a different architecture is used, devices will be added or removed which might significantly...
change the behavior of the circuit. The next step in the technology migration flow is to scale the device sizes.

![Figure 5.2: Framework for the technology migration flow.]

5.1.1.1 Device Scaling

Many approaches to device scaling during technology migration can be found in the literature [82, 83, 84, 85, 86]. The scaling approach selected depends on whether the circuit is digital or analog in nature and on the specific analog functionality. To illustrate the usefulness of following one of these approaches, we provide an example and results
of the initial scaling exercise performed for a legacy unity gain voltage buffer block, shown in Figure 5.3, ported for use in the DORIC12.

![Schematic Diagram]

**Figure 5.3:** Schematic of a legacy differential-input single-ended output amplifier configured as a unity gain buffer.

In order to preserve the original 3 dB bandwidth and reduce the total current consumption we employed the channel-length scaling technique detailed in [87]. Given the device sizes in the 0.25 µm process and physical parameters from both the 0.25 µm process and the 0.13 µm process we completed the scaling exercise as follows. First, the scaling factors for voltage ($K_V$), channel length ($K_L$), and oxide thickness ($K_{OX}$) were calculated:
\[ K_V = \frac{\text{Legacy Process VDD}}{\text{Target Process VDD}} = \frac{2.5}{1.5} = 1.67 \quad (5.1) \]

\[ K_L = \frac{\text{Legacy Process Minimum Length}}{\text{Target Process Minimum Length}} = \frac{250 \text{ nm}}{130 \text{ nm}} = 1.92 \quad (5.2) \]

\[ K_{OX} = \frac{\text{Legacy Process Gate Oxide Thickness}}{\text{Target Process Gate Oxide Thickness}} = \frac{62 \text{ nm}}{22 \text{ nm}} = 2.82 \quad (5.3) \]

Next the FET lengths and widths in the 0.13 µm process were found by:

\[ L_{0.13} = \frac{L_{0.25}}{K_L} \quad (5.4) \]

\[ W_{0.13} = \frac{W_{0.25} K_V^2 K_L}{K_{OX}} \quad (5.5) \]

Finally, the new drain currents, \( I_D \), were determined:

\[ I_{D(0.13)} = I_{D(0.25)} K_V^2 K_L^2 \quad (5.4) \]

By applying these rules, the initial scaled design of the buffer amplifier agreed quite well in comparison to the original design. In Figure 5.4, the AC gain and phase of the both the legacy 0.25 µm design and the scaled 0.13 µm design are shown. With a 40 Mb/s input signal, the nominal rate of the DORIC, centered at mid-rail, the dynamic range of the legacy design and the scaled design are compared in Figure 5.5. Table 5.1 and Table 5.2 summarize the results of the scaling exercise. From these simulations, it is clear that the initial scaling exercise yielded excellent results. Furthermore, this activity reveals that the application of an appropriate scaling approach can be quite useful and can save considerable design time and effort.
Figure 5.4: AC response of the legacy and scaled unity gain voltage buffer.

Figure 5.5: Linearity of the legacy and scaled unity gain voltage buffer.
### Table 5.1: Simulated performances of the legacy and scaled unity gain voltage buffer.

<table>
<thead>
<tr>
<th></th>
<th>Power Consumption (mW)</th>
<th>- 3 dB Frequency (GHz)</th>
<th>Phase Margin (-3 dB)</th>
<th>Dynamic Range (% of Rail)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Legacy Design (0.25 μm)</td>
<td>2.26</td>
<td>2.191</td>
<td>56.88°</td>
<td>72.0</td>
</tr>
<tr>
<td>Scaled Design (0.13 μm)</td>
<td>0.95</td>
<td>2.188</td>
<td>101.99°</td>
<td>73.3</td>
</tr>
</tbody>
</table>

### Table 5.2: Transistor dimensions of the legacy and scaled unity gain voltage buffer.

<table>
<thead>
<tr>
<th></th>
<th>M0</th>
<th>M1</th>
<th>M2</th>
<th>M3</th>
<th>M4</th>
</tr>
</thead>
<tbody>
<tr>
<td>Legacy W (μm)</td>
<td>150.000</td>
<td>30.000</td>
<td>150.000</td>
<td>24.000</td>
<td>24.000</td>
</tr>
<tr>
<td>Legacy L (μm)</td>
<td>0.300</td>
<td>1.400</td>
<td>0.300</td>
<td>0.300</td>
<td>0.300</td>
</tr>
<tr>
<td>Scaled W (μm)</td>
<td>284.326</td>
<td>56.865</td>
<td>284.326</td>
<td>45.492</td>
<td>45.492</td>
</tr>
<tr>
<td>Scaled L (μm)</td>
<td>0.156</td>
<td>0.728</td>
<td>0.156</td>
<td>0.156</td>
<td>0.156</td>
</tr>
</tbody>
</table>

#### 5.1.1.2 Scaled Design Verification

Once the scaling step, similar to what we described in the previous section, has been completed, the new design must be simulated and the performance checked against the required specifications. These simulations will likely show that the circuit is not functioning up to the required specification or could benefit from optimization. At this stage the designer begins the process of optimizing the new device sizes to achieve enhanced performance. The fact that the topology has been preserved from the legacy design, and that a reasonable scaling approach has been employed, will place the initial scaled circuit close to a working state. Therefore, the optimization and re-sizing effort...
should be not as challenging as if no previous information about the legacy transistor dimensions were known. As described in the analog design IC flow, Chapter 2, Section 2.8.1.1, once the designer is satisfied with these initial simulations, a verification of the robustness of the circuit under temperature, supply voltage, and process parameter extremes is performed. After the ported schematic verification is complete, the designer can begin work on the physical layout.

### 5.2 Experiences with IP Reuse in the VDC12, DORIC12, and Hitbus Chip

When we initiated development of the VDC12 and DORIC12, the CERN microelectronic group and HEP community had already selected a commercial 0.13 µm process as the mainstream process node for use in ASIC developments for the LHC detector upgrades. As a part of making this selection, the CERN microelectronics group performed extensive radiation studies on test cells in the 0.13 µm process and provided guidelines to meet radiation hardness requirements [60, 61]. Additionally, several collaborating groups within ATLAS were developing ASICs in this process that would in the future communicate through our opto-board and its associated chipset. Given the positive results from the radiation hardness studies and the growing experience among our collaborative support structure, the 0.13 µm process was the clear choice for the VDC12 and DORIC12 development effort.

The concept to design and include the Hitbus chip in the ATLAS DBM detector was conceived very late in the DBM design cycle and with respect to other simultaneous upgrades to ATLAS. Because of the location of the DBM within ATLAS, it was required
to be installed during an early phase of the upgrade process. Furthermore, if the DBM’s schedule were to slip, the rest of the ATLAS upgrade plans would be delayed. Since the inclusion of the Hitbus chip in the DBM was to provide additional but not absolutely required functionality, its development was not permitted to make any impact on the DBM installation schedule. Given this, the design and submission for fabrication was required to be completed in only two months. Based on the installation schedule, only the first article chips would be ready in time for the DBM. Therefore the first article Hitbus chips were required to work perfectly or else the DBM would be constructed without them. To meet the tight fabrication deadline and increase the probability of receiving a completely functional ASIC on the first fabrication run, the Hitbus chip was designed using the same 0.13 µm mainstream process mentioned above.

Because of the universal use of the 0.13 µm process we were able to gather several IP blocks from collaborating institutes within ATLAS and combine them with our own blocks to create the DORIC12, VDC12, and the Hitbus chip. In the following sections we discuss the IP blocks that were incorporated into our ASICs and discuss notable experiences that were encountered during their integration into our design environment.

5.2.1 8-Bit DAC

Our first experience in reusing an IP block from a collaborator, designed in the same 0.13 µm process node we were targeting, was with an 8-bit DAC. Included in the DAC design were a bandgap reference and a bias generator. We were provided the
complete Cadence libraries of these cells, and our hope was to use the designs with no modifications. At the schematic level, the DAC performed quite well and was perfectly suited for our application to allow remote control over the VCSEL modulation currents in the VDC12. Unfortunately, upon investigation of the layout, we discovered that for two reasons, the design required modifications. One, the DAC layout was developed in an earlier version of the Cadence IC design environment. Two, the design was based on the use of a different metal stack-up variant than what the VDC12 was targeted for.

The first step in reusing the DAC layout was to convert the design from Cadence’s IC front-end to back-end (ICFB) version 5.14 to ICFB / Virtuoso version 6.1. The process involved converting the DAC libraries from Cadence's common database format (CDB) to the Open Access (OA) format. This conversion caused many problems with the vias as the via definitions in the updated version of Virtuoso were no longer based on parameterized cells from the design kit. To correct this problem, we imported the vias from the old design kit library into the new OA format and manually changed the old and incorrect via instances to point to the new OA library. Unfortunately, this task took almost a week. The main reason for this slow repair was that some of the vias were instantiated as arrays. At the time, the find and replace tools in Virtuoso could not find cells that were part of an array. Eventually we located a SKILL script [88] on Cadence’s online support site which greatly sped up the via substitution. The difficulty of the via redefinition and layout repair process was further complicated by the differences between the metal stack-ups used in the DAC and VDC12 since the DAC included routing on the fourth metal layer (M4) which was not available in the metal stack used in the VDC12.
Because of this, the routing on M4 needed to be moved to a different layer. To get from M3 to the next higher metal layer in the stack-up used for the VDC12, MQ, the original vias from M3 to M4 needed to be changed to M3 to MQ vias. Because the via spacing and sizing DRC rules differed when going from M3 to M4 versus going from M3 to MQ, almost every instance of the new M3 to MQ via required modifications to the M3 routing in order to keep the layout DRC clean. To complete this process required considerable effort.

An additional complication was discovered in preparing the DAC IP for use in the VDC12. In the final stages of the DRC process, checks are performed to test the layout for susceptibility to the antenna effect, or plasma induced gate oxide damage [89]. Surprisingly, the DAC had several such antenna rule violations that required correction. To complete these corrections, the original layout had to be further modified in order to allow room for the addition of n-well and gate tie down diodes.

The considerable modifications described above were significantly more than we had hoped to perform when we acquired the DAC IP. Furthermore, the scope of these modifications placed risk on the preservation of the IP’s original functionality. Fortunately, following the post layout extracted simulations and verification, the DAC still performed up to the required specifications for inclusion in the VDC12. Because no time was spent on the schematic level conception, sizing, simulation, and verification, the acquisition and use of the DAC IP still saved us an enormous amount of time. Most importantly, as was discussed in Chapter 3 the DAC IP performed well in the fabricated VDC12 chip.
5.2.2 SEU Hard DICE Latch

The SEU hard DICE latch IP was used in all three of our ASICs. Fortunately, we received the IP after going through the repair process on the DAC IP. Since the SEU hard DICE latch IP was also developed in the older version of the Cadence IC design environment, it had similar problems with the vias. However, because we were experienced in making the via replacements, the process was not as difficult as what we experienced with the DAC IP. The DICE latch IP was designed using the same metal-stack variant as the VDC12, DORIC12, and Hitbus chip. Given this, following the replacement of the vias, the design was free of DRC errors. The post layout extracted simulations and verification showed that the IP worked well. All totaled, the process of importing and verifying the SEU hard DICE latch IP only took a few days. Additionally, as was discussed in Chapter 3 and Chapter 4, the SEU hard DICE latch IP performed well in the fabricated DORIC12, VDC12, and Hitbus chip.

Like the 8-bit DAC, the acquisition of the SEU hard DICE latch IP afforded us a great time savings. While the schematic level design of the DICE latch IP could have appeared simple, properly qualifying the latch required a few SEU and irradiation studies by the original designers because the layout topology had to be considered very carefully in order to improve the tolerance to SEU. As was mentioned in Chapter 3, the separation of the storage node pairs in the DICE latch had to be studied, adjusted, and re-organized to obtain a satisfactory level of SEU hardness. These studies took the original designers a few years to complete. If we were to have developed such a latch on our own, a similar development schedule would have been required. The time and effort savings was
enormous when comparing the years of development versus the few days it took for us to incorporate and verify the IP in our environment.

5.2.3 I/O Pads

For the DORIC12 and VDC12 we developed our own I/O pads. However, because of the compressed schedule of the Hitbus chip and the fact that we did not possess compact LVDS receiver, LVDS driver, or CMOS receiver pads, we acquired a set of I/O pad IP from a group of collaborators. Like the other IP we gathered, this provided us with a significant time savings. Included in the I/O pad IP were an LVDS driver, an LVDS receiver, a CMOS receiver, power pads, and ground pads. In this case, the IP was designed in the same version of the Cadence software we were using and employed the same metal stack variant of the 0.13 um process we were targeting. Unfortunately, we still ran into significant difficulty in verifying the IP in our design environment because the designer of the I/O pads used different physical verification software than we did. While the DRC checks were mostly passing, we found many problems when performing the LVS checks. The process to adjust the DRC violations was quick and straightforward but repairing the LVS in our environment was time consuming. We were certain that the layout of the IP was correct because we performed die probing and testing of 2268 chips containing the same I/O pad IP for the collaborator who designed the IP. These chips had a yield better than 95%. Given this knowledge, we decided to only make modifications at the schematic level to achieve an LVS clean result. The adjustments included changes to the NFET substrate connectivity and in a few cases
the sizing of dummy transistors included by the original designer to improve matching. After slightly more than a week of effort, the I/O pads were DRC and LVS clean in our environment and the post layout extracted simulations and verification revealed that the massaged IP was working well. Additionally, as was shown in Chapter 4, the I/O pads performed well in the fabricated Hitbus chip.

5.2.4 PLL

The Hitbus chip required a PLL with frequency dividers to generate higher speed clocks from the incoming 40 MHz TTC clock. Fortunately, we were able to obtain a PLL block from a collaborator whose IP would generate the frequencies required for the Hitbus chip application. This PLL IP was designed in the same version of the Cadence software we used, and was designed in the same metal stack variant of the 0.13 um process we had selected. Surprisingly, based on our prior experiences, the PLL block was DRC and LVS clean upon arrival. Unfortunately, upon insertion into the complete Hitbus chip, we found that the PLL caused DRC errors. This result was completely unexpected because the DRC on PLL alone passed. The nature of the DRC error was potentially very serious because it suggested a possible power to ground short. Additionally, the fabrication vendor would not accept a design with this particular DRC error. We consulted with the institute that provided us with the PLL IP and they revealed that the original designer was no longer an employee of the institute. In a generous attempt to help, the institute’s remaining designers ran checks on the PLL. They found that the PLL had an LVS mismatch when tested alone, but which disappeared once the PLL was
inserted into a higher level of the hierarchy. This institute used different physical verification software than we did, so the results did not give us much confidence. We spent some time trying to correct the DRC error but it was not possible given the time available. For safety, and to make the fabrication deadline, we submitted the Hitbus chip with our own PLL design we had verified in simulation but had not yet proven in silicon. Fortunately, as was described in Chapter 4, our PLL design worked well in the fabricated Hitbus chip.

5.2.5 FE-I4 Command Decoder

As discussed in Chapters 3 and 4, we used the command decoder IP from the FE-I4 to provide a communication interface to configure the DORIC12 and Hitbus chips. As with the other IP we reused, the block was generously shared by a collaborator. This introduced complications. The command decoder IP was designed using a standard cell library, and access to the back end of line views was not legally accessible to a university group such as us. As such, the multi-project wafer vendor we used for fabrication instantiated the standard cell layouts after we submitted the design. This made it impossible for us to perform DRC or LVS on either chip with the command decoder inserted. An added complication was that without an LVS, it was not possible to perform the extraction of the chip for a true full chip simulation. Since the command decoder was hard IP and had been proven to work in other ASICs, we took the risk to include it in both chips.
Our first experience with using the command decoder IP was in the design of a prototype of the DORIC12. Because we had the schematic view of the command decoder, we were able to employ mixed signal simulation tools to perform simulations using functional models of the command decoder logic within the extracted simulation of the full chip. As it turned out, this gave us false confidence that our interfaces to the command decoder were correct. We shared our schematic, simulation results, and plans for interfacing to the command decoder with the collaborator who designed the block. He looked over our work and found no problems. The chip was fabricated and upon receipt, we could not communicate with the command decoder. After another review by the original designer it was discovered that we connected the scan chain incorrectly. Because we left the scan chain enable pin to the command decoder IP floating, it was possible for the scan chain to start operating unexpectedly, thus corrupting the normal behavior of the command decoder. This error was not found during the functional simulation because the simulator set the initial condition of the scan enable low, disabling the scan chain. Luckily, we had foreseen the possibility of making an error in using the command decoder IP and had included a three line serial test port to allow us to operate the rest of the chip excluding the command decoder. This fallback solution worked. Upon verifying the rest of the DORIC12 prototype excluding the command decoder, we fabricated the full DORIC12 with a properly connected command decoder IP. It functioned as expected upon arrival.

Our second experience with the command decoder IP came after inserting it into the Hitbus chip. Since we had already used the block successfully in the DORIC12, we
had confidence in the management of the connectivity and in how to interface signals from the Hitbus chip to the command decoder. We fabricated the Hitbus chip with the command decoder and once again, upon arrival, we could not communicate with it. Upon pouring over the design and layout we found that the problem was caused by a via stack that mistakenly contained too many layers, shorting the input serial data line to the power supply rail. An image showing the shorted via is shown in Figure 5.6. This problem was not found during LVS of the final chip because, as was mentioned previously, we could not perform LVS once the command decoder was inserted. Despite our great care and previous experience in making connections to the IP, we still made an error. Thankfully, through the wonders of focused ion beam (FIB) circuit editing [90] the short could be removed. However, due to power and ground routing above the via, the process required the FIB house to remove large areas of metal on three layers in the metal stack above the offending via. After testing a few methods to accomplish this prospecting, the FIB repair was successful and became reproducible. As was detailed in Chapter 4, the edited Hitbus chips proved to work well.
Figure 5.6: Illustration showing the shorted via in the Hitbus chip and an SEM image of the via after FIB removal of the short.

5.3 Guidelines for Reuse of Collaboratively Shared IP

The aforementioned examples clearly reveal that incorporating collaboratively shared IP is not a trivial pursuit. This might seem surprising considering the fact that in our case the collected IP blocks were all hard IP designed in our target process node and previously proven in silicon. However, despite the difficulties, the realization of our ASICs would have been significantly more arduous and time consuming without the availability of the collaboratively shared IP. The challenges we faced highlight the importance of designing with reuse in mind, especially if IP sharing is planned. Unfortunately, designing for reuse requires much additional time, effort, and expense and
is thus a difficult path to rationalize for the designer who is usually required to produce blocks as quickly as possible. Armed with the knowledge gained through our IP reuse experiences, we have developed a set of guidelines to assist in efficiently integrating and verifying shared IP blocks. These guidelines are especially useful for incorporating blocks that were not originally designed with reuse in mind. We have also collected a similar set of guidelines for a designer of an IP block who plans to share it. These guidelines are of particular use to an IP designer who has little to no time to support the end user of the IP.

5.3.1 Guidelines for IP Reuse

1) Ask the designer if the IP has been proven in silicon. If the answer is yes, the IP block may be used with more confidence. However, if test results or performance reports are not available, maintain vigilance in your verification of the block. If the IP has yet to be fabricated by the original designer, increased emphasis must be placed on the post layout simulation and verification of the IP in order for it to be trusted.

2) If available, collect documentation, simulation setups, simulation results, and test benches from the IP designer. This will guide the verification and speed up integration of the IP.

3) Get a list of the process options used for the IP. Included in the list of options should be what metals were used, the metal stack variant, and a list of what devices were used. If options, devices, or metals were used that are not available in the options of your target, plan to spend a considerable amount of time converting the design to be
compatible. In a case of critical design functionality being provided by special
devices or process options, depending on the criticality of the IP, it may be necessary
to change the target process options to match the options used in the IP.

4) Find out what physical verification software was used by the IP’s designer. If
possible, use this software for physical verification. If this is not possible, expect
errors during the physical verification of the IP using your software. Plan accordingly
and allow time in your schedule to resolve these errors. To assist in an event of
disparate physical verification tools, ask the original designer for the configuration
files of the DRC and LVS runs in her software. Minor tweaks to DRC and LVS
configuration options can make significant differences in the outcomes of these
checks. The original designer’s configuration files should be used as a guide to
configuration of the DRC and LVS runs on the IP in your software environment.

5) Check the version of the design software that was used to design the block. If
possible, use this version. If this is not possible, expect that time will be required to
convert the design to the design software version you are using. If you work in a
newer version of the software the IP database will likely be convertible. However, if
your software is out of date, importing IP created in newer software may not be
possible. If many IP blocks are being incorporated, upgrade all design software to the
latest version. Once all of the IP blocks have been incorporated freeze the design
software versions until fabrication unless instability or bugs in the latest design
software are found.
6) Determine which version of the process design kit was used to design the IP. If the IP was designed in an older version of the design kit, expect that errors may be discovered. Also, ask the designer if any custom changes were made to their design kit. Customizations such as the addition of layers or changes to the extraction rules will likely render the IP difficult to verify unless a version of the modified design kit is also shared. However, know the risks involved and use extreme caution in trusting a modified design kit for verification.

7) Ask the designer what level of DRC correctness was achieved. For example, were antenna rules and pattern density checked? If the answer is no, plan for the additional time required to modify the design in order to pass these checks.

8) If many changes have been made to the layout or schematic to make an IP block LVS or DRC clean in your environment, a considerable effort must be taken to verify that the IP’s performance is still satisfactory. This is especially the case for analog blocks in which track resistances, via resistances, and other parasitics are critical to performance.

9) Ask the designer if the IP must be inserted into a higher hierarchical level to pass LVS or DRC checks. If this is the case, ask the designer specifically what should be done at the higher level to make the schematic and layout match or make the DRC pass. Designs constructed in this way can be very difficult to sort out unless some knowledge of the higher level fixes are known.

10) Never trust your eye to perform LVS. In other words, due to the metal layer count and the complexity of most designs, it is impossible to expect a designer to make even
small changes to a layout without risk that an error was inserted. Regardless of the skill and experience of the designer, if a chip cannot be checked with LVS, it will have errors. This guideline is especially critical in the case of a shared IP block that you cannot LVS due to legal reasons, expense, or lack access to the libraries used. If you cannot perform an LVS, simply refuse to use the layout of the block altogether. Access to schematics or the RTL of the IP would provide a significant time savings because time would only be spent creating the layout in your environment. On the other hand, if the IP designer is willing to share verification responsibilities and run the LVS and DRC on the full chip once her IP is included success might be possible. However, remember that she will have the same difficulties as you in making all of the other blocks clean in her environment. The bottom line: do not agree to fabricate an ASIC including IP that cannot be fully verified by yourself or a trusted collaborator.

5.3.2 Guidelines for IP Sharing

1) When sharing IP, make the design DRC and LVS clean under as many physical verification software packages as possible. Notify the end user of all verification software suites that were used. Include documentation on the configuration settings used and process switches selected.

2) Try to keep metal routing limited to the lowest metal layers and use as few layers as possible. Furthermore, attempt to keep your routing on layers that are available in all variants of the metal stack options for a given process node. In the documentation
packet to be given to the end user, include which metals were used in the IP and the metal stack variant selected.

3) Try to use devices such as resistors, capacitors, and FETs that are available in the standard variant of the process node. Using special options will make the design less portable. Keeping the design within the standard options will enable easier use by the end user. Furthermore, the end designer will not have to pay for extra options they might not have access to. In the IP documentation include a list of all of the devices used.

4) Verify the design with the newest available version of the process design kit. The newest design kit will be the most mature and will thus include fixes to errors or oversights in previous versions of the kit. Do not customize the process design kit. While this may be useful to simplify aspects of your design cycle, it will make trusting the IP more difficult and will require the end user to use the modified design kit.

5) In the IP documentation packet given to the end user, include the versions of the design software used, the design kit version, and the physical verification software versions used. If modification of the PDK was absolutely necessary, document any customizations that were made. If legally acceptable, include the files you modified in the PDK.

6) Place easily visible labels on the ports/pins of the layout. This will allow a designer who is not familiar with the block to easily find the ports/pins for connection to the other blocks in her design.
7) Document the schematics by inserting comments. Provide text describing critical node voltages, bias currents, and other critical parameters. Include simulation results and test benches for the IP.

8) Provide the end user with any documentation written about the IP. Include articles, datasheets, and verification reports.

9) Make the IP block LVS and DRC clean without requiring insertion in a higher level of the hierarchy. If the IP requires such special treatment, document what needs to be done at a higher level.

10) As with any design, the verification of the IP to be shared should not be performed solely by the IP designer. The verification task should be given to another designer to maximize the probability of finding faults not anticipated or conceived by the original designer.

5.4 Concise IP Sharing Template

Based on our experiences reusing IP and the guidelines provided above, we have constructed a concise template for IP sharing. Included in our template are a base document and a directory structure for organizing the IP. It is our intention that the template be assembled and constructed by the IP designer before the design is shared. If the IP designer does not have knowledge of the template, a request should be made that they construct it before sharing the design libraries. We believe that receiving organized and documented IP to match this concise template will greatly speed the incorporation of the IP and reduce the amount of support required from the original designer.
5.4.1 IP Sharing Template Structure

Once assembled, an IP design packaged using the template should be shared in the form of a tar archive with a top level structure as shown in Figure 5.7. The /designfiles/ directory contains the design libraries for the block in their native format. The /doc/ directory holds any useful documentation the IP designer chooses to include in the IP template. Examples of useful documentation include a datasheet like description of the block, actual articles written about the block, or test results. The /FE_verification/ folder contains simulation results, simulation setups, and any other content developed by the original designer to perform the functional, also known as front end, verification of the IP. Ultimately, the /FE_verification/ directory should contain a full “Characterization Environment” as described in [91]. The /BE_verification/ directory should hold the setups or scripts used to run the physical, also known as back end, verification of the block. If the IP contains digital content, the /PNR/ directory is the location to store any HDL source code, test benches, synthesis scripts, place and route setups, or timing information.

Figure 5.7: Top level structure of the IP sharing template.
5.4.1.1 readme.txt

The readme.txt file placed in the root folder of the archive, shown in Figure 5.8, provides a short description of the IP template and a mapping of its contents. Any file contained within the template, excluding the design files themselves, should be listed in the readme.txt file. Additionally, a short description of each file contained in the archive should be written next to the path to the file. Also, the readme file should contain the contact information for the IP designer(s).

Figure 5.8: Example contents of the readme.txt file.
5.4.1.2 “Vitals” Spreadsheet

The most critical part of the template, aside from the design library itself, is a spreadsheet, IP_BLOCK_NAME_vitals.ods, placed in the root folder of the archive. Containing two tabs, this spreadsheet holds the vital information for the block. The first tab, shown in Figure 5.9, contains the critical process node and physical verification information about the block. The second tab, shown in Figure 5.10, contains the list of the ports, a short description on the port properties, and their location in the layout. A close inspection of both tabs reveals that they summarize the information deemed critical by our IP reuse and IP sharing guidelines.
Figure 5.9: Summary tab of the vitals spreadsheet.

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
</tr>
</thead>
<tbody>
<tr>
<td>Designer</td>
<td>Jane Designer</td>
<td>614-555-5555</td>
<td><a href="mailto:Jane1234@asu.edu">Jane1234@asu.edu</a></td>
</tr>
<tr>
<td>Backup</td>
<td>Joe Vermeer</td>
<td>614-555-5554</td>
<td><a href="mailto:Joe123@asu.edu">Joe123@asu.edu</a></td>
</tr>
<tr>
<td>IP Name</td>
<td>IP_BLOCK_NAME</td>
<td></td>
<td></td>
</tr>
<tr>
<td>IP Description</td>
<td>8 bit DAC with 2us settling time</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Top Level Cell</td>
<td>DAC00 TOP</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Proven in Silicon?</td>
<td>Yes</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Process Noise</td>
<td>330 nm</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Design Size</td>
<td>300 x 300</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Foundry</td>
<td>IBM</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Process Variant</td>
<td>DM</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Design Kit Version</td>
<td>V1 8.0 4DM</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Design Kit Customized</td>
<td>No</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Metal Stack Variant</td>
<td>3.2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Metal Used</td>
<td>MI-ML3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Devices Used</td>
<td>mmi, mcap, dgt, ohf, ohf, dcap, ohf, ohf, dcap, ohf</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Standard Cell Library Name</td>
<td>NA</td>
<td></td>
<td></td>
</tr>
<tr>
<td>LVS Clean</td>
<td>Yes</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ERC Clean</td>
<td>Yes</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Antenna Rules Clean</td>
<td>Yes</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Pattern Density Clean</td>
<td>Yes</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Local Density Clean</td>
<td>Yes</td>
<td></td>
<td></td>
</tr>
<tr>
<td>June Mode &amp; Orthogonality Clean</td>
<td>Yes</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ESD Clean</td>
<td>No</td>
<td></td>
<td></td>
</tr>
<tr>
<td>All clean without hierarchy?</td>
<td>Yes</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Design Software</td>
<td>Cadence Virtuoso</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Version</td>
<td>HOT x IC0614S13 414_Inc86</td>
<td></td>
<td></td>
</tr>
<tr>
<td>LVS Software</td>
<td>Assura</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Version</td>
<td>HOT x ASSURA04 12.010-834_Inc86</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ERC Software</td>
<td>Assura</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Version</td>
<td>HOT x ASSURA04 12.010-834_Inc86</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Sheet 1/2 mp1 STD Sum=0 Average=
**Figure 5.10:** Port list tab from the vitals spreadsheet

<table>
<thead>
<tr>
<th>Port Name</th>
<th>Description</th>
<th>Location X</th>
<th>Location Y</th>
<th>Metal Layer</th>
</tr>
</thead>
<tbody>
<tr>
<td>VDD</td>
<td>positive power supply, +1.5 V</td>
<td>200</td>
<td>300</td>
<td>M3</td>
</tr>
<tr>
<td>GND</td>
<td>power supply return</td>
<td>200</td>
<td>100</td>
<td>M3</td>
</tr>
<tr>
<td>VREF</td>
<td>DAC reference voltage</td>
<td>100</td>
<td>100</td>
<td>M2</td>
</tr>
<tr>
<td>VOUT</td>
<td>Output voltage</td>
<td>300</td>
<td>200</td>
<td>M2</td>
</tr>
<tr>
<td>B0</td>
<td>DAC value bit 0</td>
<td>100</td>
<td>120</td>
<td>M2</td>
</tr>
<tr>
<td>B1</td>
<td>DAC value bit 1</td>
<td>100</td>
<td>140</td>
<td>M2</td>
</tr>
<tr>
<td>B2</td>
<td>DAC value bit 2</td>
<td>100</td>
<td>160</td>
<td>M2</td>
</tr>
<tr>
<td>B3</td>
<td>DAC value bit 3</td>
<td>100</td>
<td>180</td>
<td>M2</td>
</tr>
<tr>
<td>B4</td>
<td>DAC value bit 4</td>
<td>100</td>
<td>200</td>
<td>M2</td>
</tr>
<tr>
<td>B5</td>
<td>DAC value bit 5</td>
<td>100</td>
<td>220</td>
<td>M2</td>
</tr>
<tr>
<td>B6</td>
<td>DAC value bit 6</td>
<td>100</td>
<td>240</td>
<td>M2</td>
</tr>
<tr>
<td>B7</td>
<td>DAC value bit 7</td>
<td>100</td>
<td>260</td>
<td>M2</td>
</tr>
<tr>
<td>WR</td>
<td>write signal (on rising edge)</td>
<td>100</td>
<td>280</td>
<td>M2</td>
</tr>
</tbody>
</table>
Chapter 6: Conclusion

The purpose of the research described in this dissertation was to design, verify, fabricate, test, and qualify three separate radiation hard ASIC chips and a novel fiber-optic transceiver board all developed to be used as custom HEP instrumentation. Two of the ASICs, the DORIC12 and VDC12, were designed to form the radiation hard chipset for the fiber-optic transceiver board and the other ASIC, the Hitbus chip, was developed for the ATLAS DBM to provide the capability for the DBM detector to trigger its own readout. All three ASICs required the use of radiation hardening by design techniques and a mixed-signal design flow. Intermixed in the research effort was the incorporation of shared IP blocks. In this dissertation we reported on the design and testing of the three ASICs and included insights learned through the process of collaboratively shared design reuse. Moreover, we presented guidelines and a concise template for effectively sharing, integrating, and verifying shared block designs with an emphasis on IP blocks not originally designed with reuse in mind.

6.1 VDC12, DORIC12, and Improved Opto-board Results and Future Work

We successfully developed prototypes of an opto-board and a 12-channel optical array based ASIC chipset, consisting of the DORIC12 and VDC12, which are capable of
allowing active redundancy and individual control over the VCSEL bias and modulation currents. In addition, we developed an optical package that does not adversely impact the lifetime of VCSELs once they have been mounted. We have constructed a significant sample of optical packages with good yield providing efficient coupling of light between the VCSEL and PIN arrays and fibers. Irradiation results show that our improved opto-board and its essential components, the DORIC12, VDC12, VCSELs, PINs, and optical packaging are radiation hard to at least 9.0x10^{14} 1-MeV neq/cm^2 (Si) or 7.7x10^{15} 1-MeV neq/cm^2, (GaAs), allowing PIN receive data rates of 40 Mb/s and VSCEL transmit rates of 160 Mb/s after irradiation. Furthermore, operation of the VDC12 at 5 Gb/s was demonstrated. The optical eye diagram satisfies the IEEE 802.3ae specification with a good margin. The VDC12 is the first array driver ASIC available to the HEP community that can provide this much data bandwidth.

Given the problem discovered with the rate of erroneous commands re-configuring the DORIC12 on the opto-boards during the irradiation, it is critical that the DORIC12 be re-fabricated with modified command processor logic. The improved command acceptance logic should be designed to respond only if identical commands are received by at least three channels and within a limited time window. With this algorithm the spurious command rate will be dramatically reduced. Additionally, the acceptance time window should be programmable either by wire-bonding or external biasing. Because the foreseen upgrade scenarios, occurring in the early 2020’s, do not require higher downlink rates, an improved version of the DORIC12 would be suitable for operation in ATLAS for more than 10 years.
Considering the same upgrade scenarios planned by ATLAS, 10 Gb/s uplink signals will be required. Fueled by the successful operation of the VDC12 at 5 Gb/s, we are pursuing a design of the VDC12 capable of 10 Gb/s. In addition to creating an updated VDC12 ASIC, it will also be necessary to significantly modify the design of the opto-board and the array optical packages in order to operate at 10 Gb/s. While 10 Gb/s optical links are mature in industry, as yet there are none that have sufficient radiation hardness for the most challenging HEP deployments.

6.2 Hitbus Chip Results and Future Work

As was presented in Chapter 4, we successfully developed, qualified, and delivered the production quantity of Hitbus chips to CERN to be installed in the ATLAS DBM detector. As a part of this process we developed new custom circuitry and incorporated IP that was provided to us by collaborators. Two samples of the Hitbus chip survived an irradiation to $4.3 \times 10^{15}$ p/cm$^2$ maintaining complete operational functionality and error free performance at 40 Mb/s. Furthermore, the SEU hardness of the Hitbus chip was tested and found to have a predicted rate of only ~12 SEU induced reconfigurations per year. The first article Hitbus chip design proved to be fully operational and was delivered in time.

Once the radioactive activation has decayed enough for shipment to our labs, we plan to fully characterize the irradiated Hitbus chips. However, since we closely monitored the chips both during and after the irradiation we do not expect to find any problems. Other future endeavors include supporting the team at CERN responsible for
developing the data acquisition software. Furthermore, other groups have requested use of the chip for varying purposes and we plan to deliver chips and provide support to their utilization.

6.3 Design Reuse Results and Future Work

We hope that this dissertation makes clear that collaboration and design reuse among University or scientific ASIC design teams, though difficult, can be extremely beneficial in allowing newly conceived ideas to be implemented and better instrumentation constructed. By reusing designs from our own libraries we demonstrated the effectiveness of employing methods to guide process migration and to reuse such designs. Through our experiences incorporating IP blocks not originally designed with reuse in mind, we developed a set of guidelines for both reusing IP and designing IP for reuse. In addition to these guidelines, we developed an IP sharing template which collects our guidelines and promotes a smoother IP transfer process. In summary, we would like to highlight the three most critical guidelines for reusing IP and designing IP for reuse:

IP Reuse

1) Find out what physical verification software was used by the IP’s designer. If possible, use this software for physical verification. If this is not possible, expect errors during the physical verification of the IP using your software. Plan accordingly and allow time in your schedule to resolve these errors.
2) If many changes have been made to the layout or schematic to make an IP block LVS or DRC clean in your environment, considerable efforts must be taken to verify that the IP’s performance is still satisfactory. This is especially the case for analog blocks where track resistances, via resistances, and other parasitics are critical to performance.

3) Never trust your eye to perform LVS. Regardless of the skill and experience of the designer, if a chip cannot be checked with LVS, it will have errors.

**IP Sharing**

1) When sharing IP, make the design DRC and LVS clean under as many physical verification software packages as possible. Notify the end user of the verification software suites that were used.

2) Make the IP block LVS and DRC clean by itself without requiring insertion in a higher level of the hierarchy. If the IP requires such special treatment, document what needs to be done at the higher level.

3) Like any design, the verification of the IP to be shared should not be performed solely by the IP designer. The verification task should be given to another designer to maximize the probability of finding faults not anticipated or conceived by the original designer.

The work on collaborative IP reuse presented in this dissertation can be extended to improve the way designs are completed for the next generation HEP experiments. Because of the collaborative nature of HEP, the practice of IP sharing will not stop.
CERN is currently working to develop support for a 65 nm process using an approach similar to that utilized during the development of the 130 nm process support structure. With this in mind, it would be very useful to include the design with reuse in mind approach at an early stage of the development of the 65 nm support structure. Included in this approach should be the adherence to packaging IP as described by our concise IP sharing template. To test the viability of our template, we first plan to prepare and package the IP for a single VCSEL driver channel. Once the packaging is complete, we will then ask a volunteer institute to receive and attempt to incorporate the VDC IP. We foresee that this process will reveal flaws and omissions in the assembly of the template and in our IP. In performing these exercises we expect to enhance our IP and make the template more robust for use by others.
Appendix A: VHDL Code
A.1: Hitbus Chip Programmable Delay: delay_module.vhd

--Program: delay_module.vhd
--Description: Delays signal by programmable amount
library IEEE, fvhdl;
use IEEE.STD_LOGIC_1164.all, fvhdl.all;

entity delay_module is
  port ( 
      buf_in : in std_logic_vector (7 downto 0); -- buffer the delay bus for
      -- connection to the next delay
      buf_out : out std_logic_vector (7 downto 0); 
      del_in : in std_logic_vector (6 downto 0); 
      m_data_out : out std_logic; 
      m_data_in : in std_logic; 
      bypass : in std_logic); 
end delay_module;

architecture structural of delay_module is

component BUFFER_I -- buffer from the standard cell library
  port ( 
      Z  : out std_ulogic; 
      A  : in std_ulogic); 
end component;

component one_delay
  port ( 
      one_data_out : out std_logic; 
      one_data_in : in std_logic); 
end component;

component two_delay
  port ( 
      two_data_out : out std_logic; 
      two_data_in : in std_logic); 
end component;

component four_delay
  port ( 
      four_data_out : out std_logic; 
      four_data_in : in std_logic); 
end component;

component delay8
  port ( 
      data_out : out std_logic; 
      data_in : in std_logic); 
end component;

component delay16
  port ( 
      data_out : out std_logic; 
      data_in : in std_logic); 
end component;

component delay32
  port ( 
      data_out : out std_logic; 
      data_in : in std_logic); 
end component;

component delay64
  port ( 
      data_out : out std_logic; 
      data_in : in std_logic); 
end component;
end component;

signal d8d7, d7d6, d6d5, d5d4, d4d3, d3d2, d2d1 : std_logic;
signal d8o, d7o, d6o, d5o, d4o, d3o, d2o, d1o : std_logic;
signal data_delayed : std_logic;
signal delay_len : std_logic_vector (6 downto 0);

begin -- structural

buff_0 : BUFFER_I port map (
  Z => buf_out(0),
  A => buf_in(0));

buff_1 : BUFFER_I port map (
  Z => buf_out(1),
  A => buf_in(1));

buff_2 : BUFFER_I port map (
  Z => buf_out(2),
  A => buf_in(2));

buff_3 : BUFFER_I port map (
  Z => buf_out(3),
  A => buf_in(3));

buff_4 : BUFFER_I port map (
  Z => buf_out(4),
  A => buf_in(4));

buff_5 : BUFFER_I port map (
  Z => buf_out(5),
  A => buf_in(5));

buff_6 : BUFFER_I port map (
  Z => buf_out(6),
  A => buf_in(6));

buff_7 : BUFFER_I port map (
  Z => buf_out(7),
  A => buf_in(7));

buff_del_0 : BUFFER_I port map (
  Z => delay_len(0),
  A => del_in(0));

buff_del_1 : BUFFER_I port map (
  Z => delay_len(1),
  A => del_in(1));

buff_del_2 : BUFFER_I port map (
  Z => delay_len(2),
  A => del_in(2));

buff_del_3 : BUFFER_I port map (
  Z => delay_len(3),
  A => del_in(3));

buff_del_4 : BUFFER_I port map (
  Z => delay_len(4),
  A => del_in(4));

buff_del_5 : BUFFER_I port map (
  Z => delay_len(5),
  A => del_in(5));

buff_del_6 : BUFFER_I port map (
  Z => delay_len(6),
  A => del_in(6));
A => del_in(6));

d1 : one_delay port map (  
  one_data_out => d1o,  
  one_data_in => d2d1);  

d2 : two_delay port map (  
  two_data_out => d2o,  
  two_data_in => d3d2);  

d3 : four_delay port map ( -- note to reader, the larger delays follow the same format as  
  -- 1delay.vhd and 2delay.vhd listed below  
  four_data_out => d3o,  
  four_data_in => d4d3);  

d4 : delay8 port map (  
  data_out => d4o,  
  data_in => d5d4);  

d5 : delay16 port map (  
  data_out => d5o,  
  data_in => d6d5);  

d6 : delay32 port map (  
  data_out => d6o,  
  data_in => d7d6);  

d7 : delay64 port map (  
  data_out => d7o,  
  data_in => m_data_in);  

m_data_out <= m_data_in when bypass = '1' else data_delayed;  
data_delayed <= d1o when delay_len(0) = '1' else d2d1;  
d2d1 <= d2o when delay_len(1) = '1' else d3d2;  
d3d2 <= d3o when delay_len(2) = '1' else d4d3;  
d4d3 <= d4o when delay_len(3) = '1' else d5d4;  
d5d4 <= d5o when delay_len(4) = '1' else d6d5;  
d6d5 <= d6o when delay_len(5) = '1' else d7d6;  
d7d6 <= d7o when delay_len(6) = '1' else m_data_in;  

end structural;
A.2: Hitbus Chip Single Delay Element: 1delay.vhd

library IEEE, fvhdl;
use IEEE.std_logic_1164.all, fvhdl.all;

entity one_delay is
  port ( 
    one_data_out : out std_logic;
    one_data_in : in std_logic);
end one_delay;

architecture structural of one_delay is
  component INVERT_A --"slow" inverter from the standard cell library
    port ( 
      Z : out std_ulogic;
      A : in std_ulogic);
  end component;

  signal internal : std_logic;

  begin
    d1 : INVERT_A port map ( 
      A => one_data_in,
      Z => internal);

    d2 : INVERT_A port map ( 
      A => internal,
      Z => one_data_out);
  end structural;
A.3: Hitbus Chip Twin Delay Element: 2delay.vhd

----------------------------------------------------------------------------------
--Program: 2delay.vhd
--Description: two delay elements in series
----------------------------------------------------------------------------------
library IEEE, vhdl;
use IEEE.std_logic_1164.all, vhdl.all;

entity two_delay is
  port (
    two_data_out : out std_logic;
    two_data_in : in std_logic);
end two_delay;

architecture structural of two_delay is
  component one_delay
    port (
      one_data_out : out std_logic;
      one_data_in : in std_logic);
  end component;

  signal internal_delay : std_logic;

begin
  d1 : one_delay port map (
    one_data_out => internal_delay,
    one_data_in => two_data_in);
  d2 : one_delay port map (
    one_data_out => two_data_out,
    one_data_in => internal_delay);
end structural;
A.4: Hitbus Chip HitOr Control Block: hitor_processor.vhd

library IEEE, fvhdl;
use IEEE.STD_LOGIC_1164.all, fvhdl.all;
use IEEE.VITAL_Timing.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;
use IEEE.std_logic_textio.all;

entity hitor_processor is
  port (por_in : in std_logic;
        h1a : in std_logic;
        h2a : in std_logic;
        h3a : in std_logic;
        h1b : in std_logic;
        h2b : in std_logic;
        h3b : in std_logic;
        d_in : in std_logic_vector (15 downto 0);
        valid_in : in std_logic;
        clk1x : in std_logic;
        clk2x : in std_logic;
        fa_in : in std_logic_vector (7 downto 0);
        fb_in : in std_logic_vector (7 downto 0);
        encode_in : in std_logic;
        delbus : out std_logic_vector (7 downto 0);
        fa_out : out std_logic_vector (7 downto 0);
        fb_out : out std_logic_vector (7 downto 0);
        clk_set_out : out std_logic_vector (7 downto 0);
        bypass : out std_logic;
        bypass_load : out std_logic;
        dload1 : out std_logic;
        dload2 : out std_logic;
        dload3 : out std_logic;
        dload4 : out std_logic;
        dload5 : out std_logic;
        dload6 : out std_logic;
        data_outA : out std_logic;
        data_outB : out std_logic;
        clr_def : out std_logic;
        encode_out : out std_logic;
        encode_load : out std_logic;
        fa_load : out std_logic;
        fb_load : out std_logic;
        clk_load : out std_logic;
        reset_out : out std_logic)
);

architecture Behavioral of hitor_processor is
  signal encvoteA1 : std_logic;
  signal encvoteA2 : std_logic;
  signal encvoteA3 : std_logic;
  signal encvoteB1 : std_logic;
  signal encvoteB2 : std_logic;
  signal encvoteB3 : std_logic;
  signal A_processed : std_logic;
  signal B_processed : std_logic;
  signal internal_reset : std_logic;
  signal rem_rst : std_logic;
  signal h1a_sampa : std_logic;
  signal h2a_sampa : std_logic;
  signal h3a_sampa : std_logic;
signal h1b_sampa : std_logic;
signal h2b_sampa : std_logic;
signal h3b_sampa : std_logic;
signal h1a_sampb : std_logic;
signal h2a_sampb : std_logic;
signal h3a_sampb : std_logic;
signal h1b_sampb : std_logic;
signal h2b_sampb : std_logic;
signal h3b_sampb : std_logic;
signal h1a_sampc : std_logic;
signal h2a_sampc : std_logic;
signal h3a_sampc : std_logic;
signal h1b_sampc : std_logic;
signal h2b_sampc : std_logic;
signal h3b_sampc : std_logic;
signal h1a_vote : std_logic;
signal h2a_vote : std_logic;
signal h3a_vote : std_logic;
signal h1b_vote : std_logic;
signal h2b_vote : std_logic;
signal h3b_vote : std_logic;
signal hastream : std_logic_vector(2 downto 0);
signal hbstream : std_logic_vector(2 downto 0);
signal haserial1 : std_logic;
signal hbserial1 : std_logic;
signal haserial2 : std_logic;
signal hbserial2 : std_logic;
signal haserial3 : std_logic;
signal hbserial3 : std_logic;
signal haserial_vote : std_logic;
signal hbserial_vote : std_logic;
signal hamarkstream : std_logic_vector(4 downto 0);
signal hbmarkstream : std_logic_vector(4 downto 0);
signal hamarkserial1 : std_logic;
signal hbmarkserial1 : std_logic;
signal hamarkserial2 : std_logic;
signal hbmarkserial2 : std_logic;
signal hamarkserial3 : std_logic;
signal hbmarkserial3 : std_logic;
signal hamarkserial_vote : std_logic;
signal hbmarkserial_vote : std_logic;
signal load3a : std_logic;
signal cnt3a : std_logic_vector (1 downto 0);
signal load3b : std_logic;
signal cnt3b : std_logic_vector (1 downto 0);
signal load3c : std_logic;
signal cnt3c : std_logic_vector (1 downto 0);
signal load3_vote : std_logic;
signal load3a : std_logic;
signal load3b : std_logic;
signal load3c : std_logic;
signal load3_vote : std_logic;
signal valid_del : std_logic;
signal del1a : std_logic;
signal del2a : std_logic;
signal del1b : std_logic;
signal del2b : std_logic;
signal del1c : std_logic;
signal del2c : std_logic;
signal del3a : std_logic;
signal del3b : std_logic;
signal del3c : std_logic;
signal load4a : std_logic;
signal load4_vote : std_logic;
signal cnt4a : std_logic_vector(2 downto 0);
signal haonemark_vote : std_logic;
signal hbonemark_vote : std_logic;
signal haonestream : std_logic_vector(3 downto 0);
signal hbonestream : std_logic_vector(3 downto 0);
signal haon SERIAL1 : std_logic;
signal haon SERIAL2 : std_logic;
signal haon SERIAL3 : std_logic;
signal hbon SERIAL1 : std_logic;
signal hbon SERIAL2 : std_logic;
signal hbon SERIAL3 : std_logic;
signal load4b : std_logic;
signal load4c : std_logic;
signal cnt4b : std_logic_vector(2 downto 0);
signal cnt4c : std_logic_vector(2 downto 0);
signal load8a : std_logic;
signal load8_vote : std_logic;
signal cnt8a : std_logic_vector(2 downto 0);
signal hbfxread_vote : std_logic;
signal hafxread_vote : std_logic;
signal hbfxstream : std_logic_vector(7 downto 0);
signal hafxstream : std_logic_vector(7 downto 0);
signal hbfxread1 : std_logic;
signal hafxread1 : std_logic;
signal load8b : std_logic;
signal load8c : std_logic;
signal cnt8b : std_logic_vector(2 downto 0);
signal cnt8c : std_logic_vector(2 downto 0);
signal hbfxread2 : std_logic;
signal hafxread2 : std_logic;
signal hbfxread3 : std_logic;
signal hafxread3 : std_logic;

component BPM_encoder
port {
    dd_clk : in std_logic;
    reset : in std_logic;
    data : in std_logic;
    encode : in std_logic;
    encoded_out : out std_logic);
end component;

component piso3g
port{
    DIN : in std_logic_vector(2 downto 0);
    CLK, rst, load : in std_logic;
    SO : out std_logic);
end component;

component piso4g
port{
    DIN : in std_logic_vector(3 downto 0);
    CLK, rst, load : in std_logic;
    SO : out std_logic);
end component;

component piso5g
port{
    DIN : in std_logic_vector(4 downto 0);
    CLK, rst, load : in std_logic;
    SO : out std_logic);
end component;

component piso8g
port{
    DIN : in std_logic_vector(7 downto 0);
component dff_asynch -- d flip flop from standard cell libraries
  port(
    D, Clk, rst : in std_logic;
    Q : out std_logic);
end component;

begin
  -- delay valid in for setup of data
  dff_1A : dff_asynch
  port map (valid_in, clk2x, internal_reset, del1a);
  dff_2A : dff_asynch
  port map (del1a, clk2x, internal_reset, del2a);
  dff_3A : dff_asynch
  port map (del2a, clk2x, internal_reset, del3a);
  dff_1B : dff_asynch
  port map (valid_in, clk2x, internal_reset, del1b);
  dff_2B : dff_asynch
  port map (del1b, clk2x, internal_reset, del2b);
  dff_3B : dff_asynch
  port map (del2b, clk2x, internal_reset, del3b);
  dff_1C : dff_asynch
  port map (valid_in, clk2x, internal_reset, del1c);
  dff_2C : dff_asynch
  port map (del1c, clk2x, internal_reset, del2c);
  dff_3C : dff_asynch
  port map (del2c, clk2x, internal_reset, del3c);
valid_del <= valid_in and ((del2a and del2c) or (del2b and del2c) or (del2a and del2b));
  -- write to clk select register = 0x15xx
  clk_load <= not por_in and valid_del and not d_in(15) and not d_in(14) and not d_in(13)
  and d_in(12) and not d_in(11) and d_in(10) and not d_in(9) and d_in(8);
clk_set_out <= d_in (7 downto 0);
  -- triply redundant BPM encoders for output A and B
ENC1 : BPM_encoder
  port map(clk2x, internal_reset, A_processed, encode_in, encvoteA1);
ENC2 : BPM_encoder
  port map(clk2x, internal_reset, A_processed, encode_in, encvoteA2);
ENC3 : BPM_encoder
  port map(clk2x, internal_reset, A_processed, encode_in, encvoteA3);
data_outA <= (encvoteA1 and encvoteA2) or (encvoteA2 and encvoteA3) or (encvoteA1 and
  encvoteA3);
ENC4 : BPM_encoder
  port map(clk2x, internal_reset, B_processed, encode_in, encvoteB1);
ENC5 : BPM_encoder
ENC6 : BPM encoder
port map(clk2x, internal_reset, B_processed, encode_in, encvoteB2);

data_outB <= (encvoteB1 and encvoteB2) or (encvoteB2 and encvoteB3) or (encvoteB1 and encvoteB3);

--reset command
rem_rst <= not por_in and not d_in(15) and not d_in(14) and not d_in(13) and not d_in(12) and not d_in(11) and not d_in(10) and not d_in(9) and not d_in(8) and not d_in(7) and not d_in(6) and not d_in(5) and not d_in(4) and not d_in(3) and not d_in(2) and not d_in(1) and d_in(0);

internal_reset <= rem_rst or por_in;
reset_out <= internal_reset;

-- clear POR default signal = 0x94CD - pulses POR default CLR for duration of CMD WR
clr_def <= not por_in and valid_del and d_in(15) and not d_in(14) and not d_in(13) and d_in(12) and not d_in(11) and not d_in(10) and not d_in(9) and d_in(8) and not d_in(7) and d_in(6) and not d_in(5) and not d_in(4) and d_in(3) and not d_in(2) and not d_in(1) and d_in(0);

-- set encode register = 0x02xx -LSB determines encoding state
encode_out <= not por_in and not d_in(15) and not d_in(14) and not d_in(13) and not d_in(12) and not d_in(11) and d_in(10) and not d_in(9) and not d_in(8) and not d_in(7) and d_in(6) and not d_in(5) and not d_in(4) and d_in(3) and not d_in(2) and not d_in(1) and d_in(0);

encode_load <= not por_in and valid_del and not d_in(15) and not d_in(14) and not d_in(13) and not d_in(12) and not d_in(11) and not d_in(10) and d_in(9) and not d_in(8) and not d_in(7) and d_in(6) and not d_in(5) and not d_in(4) and d_in(3) and not d_in(2) and not d_in(1) and d_in(0);

-- write delays to SEU registers = 0x05xx to 0x10xx
dload1 <= not por_in and valid_del and not d_in(15) and not d_in(14) and not d_in(13) and not d_in(12) and not d_in(11) and not d_in(10) and not d_in(9) and not d_in(8) and not d_in(7) and d_in(6) and not d_in(5) and not d_in(4) and d_in(3) and not d_in(2) and not d_in(1) and d_in(0);

dload2 <= not por_in and valid_del and not d_in(15) and not d_in(14) and not d_in(13) and not d_in(12) and not d_in(11) and not d_in(10) and not d_in(9) and not d_in(8) and not d_in(7) and d_in(6) and not d_in(5) and not d_in(4) and d_in(3) and not d_in(2) and not d_in(1) and d_in(0);

dload3 <= not por_in and valid_del and not d_in(15) and not d_in(14) and not d_in(13) and not d_in(12) and not d_in(11) and not d_in(10) and not d_in(9) and not d_in(8) and not d_in(7) and d_in(6) and not d_in(5) and not d_in(4) and d_in(3) and not d_in(2) and not d_in(1) and d_in(0);

dload4 <= not por_in and valid_del and not d_in(15) and not d_in(14) and not d_in(13) and not d_in(12) and not d_in(11) and not d_in(10) and not d_in(9) and not d_in(8) and not d_in(7) and d_in(6) and not d_in(5) and not d_in(4) and d_in(3) and not d_in(2) and not d_in(1) and d_in(0);

dload5 <= not por_in and valid_del and not d_in(15) and not d_in(14) and not d_in(13) and not d_in(12) and not d_in(11) and not d_in(10) and not d_in(9) and not d_in(8) and not d_in(7) and d_in(6) and not d_in(5) and not d_in(4) and d_in(3) and not d_in(2) and not d_in(1) and d_in(0);

dload6 <= not por_in and valid_del and not d_in(15) and not d_in(14) and not d_in(13) and not d_in(12) and not d_in(11) and not d_in(10) and not d_in(9) and not d_in(8) and not d_in(7) and d_in(6) and not d_in(5) and not d_in(4) and d_in(3) and not d_in(2) and not d_in(1) and d_in(0);

delbus <= d_in (7 downto 0);

-- choose to bypass delays = 0x11xx -LSB determines encoding state
bypass <= not por_in and not d_in(15) and not d_in(14) and not d_in(13) and not d_in(12) and not d_in(11) and not d_in(10) and not d_in(9) and d_in(8) and not d_in(7) and d_in(6) and not d_in(5) and not d_in(4) and d_in(3) and not d_in(2) and not d_in(1) and d_in(0);

bypass_load <= not por_in and valid_del and not d_in(15) and not d_in(14) and not d_in(13) and not d_in(12) and not d_in(11) and not d_in(10) and not d_in(9) and not d_in(8) and not d_in(7) and d_in(6) and not d_in(5) and not d_in(4) and d_in(3) and not d_in(2) and not d_in(1) and d_in(0);

-- write to FA register = 0x12xx
fa_load <= not por_in and valid_del and not d_in(15) and not d_in(14) and not d_in(13) and not d_in(12) and not d_in(11) and not d_in(10) and not d_in(9) and d_in(8) and not d_in(7) and d_in(6) and not d_in(5) and not d_in(4) and d_in(3) and not d_in(2) and not d_in(1) and d_in(0);

fa_out <= d_in (7 downto 0);

-- write to FB register = 0x13xx

137
fb_load <= not por_in and valid_del and not d_in(15) and not d_in(14) and not d_in(13)
and d_in(12) and not d_in(11) and not d_in(10) and d_in(9) and d_in(8);

fb_out <= d_in (7 downto 0);

-- triple redundant sample at 160M, on falling edge of clk 160
sampler : process(clk1x, internal_reset)
begin
if internal_reset = '1' then
  h1a_sampa <= '0';
  h1a_sampb <= '0';
  h1a_sampc <= '0';
  h2a_sampa <= '0';
  h2a_sampb <= '0';
  h2a_sampc <= '0';
  h3a_sampa <= '0';
  h3a_sampb <= '0';
  h3a_sampc <= '0';
  h1b_sampa <= '0';
  h1b_sampb <= '0';
  h1b_sampc <= '0';
  h2b_sampa <= '0';
  h2b_sampb <= '0';
  h2b_sampc <= '0';
  h3b_sampa <= '0';
  h3b_sampb <= '0';
  h3b_sampc <= '0';
elsif (clk1x = '0' and clk1x'event) then
  h1a_sampa <= h1a;
  h1a_sampb <= h1a;
  h1a_sampc <= h1a;
  h2a_sampa <= h2a;
  h2a_sampb <= h2a;
  h2a_sampc <= h2a;
  h3a_sampa <= h3a;
  h3a_sampb <= h3a;
  h3a_sampc <= h3a;
  h1b_sampa <= h1b;
  h1b_sampb <= h1b;
  h1b_sampc <= h1b;
  h2b_sampa <= h2b;
  h2b_sampb <= h2b;
  h2b_sampc <= h2b;
  h3b_sampa <= h3b;
  h3b_sampb <= h3b;
  h3b_sampc <= h3b;
end if;
end process;

h1a_vote <= (h1a_sampa and h1a_sampb) or (h1a_sampb and h1a_sampc) or (h1a_sampa
and h1a_sampc);

h2a_vote <= (h2a_sampa and h2a_sampb) or (h2a_sampb and h2a_sampc) or (h2a_sampa
and h2a_sampc);

h3a_vote <= (h3a_sampa and h3a_sampb) or (h3a_sampb and h3a_sampc) or (h3a_sampa
and h3a_sampc);

h1b_vote <= (h1b_sampa and h1b_sampb) or (h1b_sampb and h1b_sampc) or (h1b_sampa
and h1b_sampc);

h2b_vote <= (h2b_sampa and h2b_sampb) or (h2b_sampb and h2b_sampc) or (h2b_sampa
and h2b_sampc);

h3b_vote <= (h3b_sampa and h3b_sampb) or (h3b_sampb and h3b_sampc) or (h3b_sampa
and h3b_sampc);
-- determine how hit signals are processed

muxA : process(fa_in, h1a_vote, h2a_vote, h3a_vote, haserial_vote, hamarkserial_vote, haonemark_vote, hbfxread_vote)
begins
  case(fa_in) is
    when "00001001" => A_processed <= h1a_vote;
    when "00010010" => A_processed <= h2a_vote;
    when "00011100" => A_processed <= h3a_vote;
    when "00111111" => A_processed <= h1a_vote and h2a_vote and h3a_vote;
    when "01001111" => A_processed <= h1a_vote or h2a_vote or h3a_vote;
    when "01011111" => A_processed <= (h1a_vote and h2a_vote) or (h2a_vote and h3a_vote) or (h1a_vote and h3a_vote);
    when "01101000" => A_processed <= (h1a_vote or fa_in(0)) and (h2a_vote or fa_in(1)) and (h3a_vote or fa_in(2));
    when "01101001" => A_processed <= (h1a_vote or fa_in(0)) and (h2a_vote or fa_in(1)) and (h3a_vote or fa_in(2));
    when "01101010" => A_processed <= (h1a_vote or fa_in(0)) and (h2a_vote or fa_in(1)) and (h3a_vote or fa_in(2));
    when "01101011" => A_processed <= (h1a_vote or fa_in(0)) and (h2a_vote or fa_in(1)) and (h3a_vote or fa_in(2));
    when "01101100" => A_processed <= (h1a_vote or fa_in(0)) and (h2a_vote or fa_in(1)) and (h3a_vote or fa_in(2));
    when "01101101" => A_processed <= (h1a_vote or fa_in(0)) and (h2a_vote or fa_in(1)) and (h3a_vote or fa_in(2));
    when "01101110" => A_processed <= (h1a_vote or fa_in(0)) and (h2a_vote or fa_in(1)) and (h3a_vote or fa_in(2));
    when "01101111" => A_processed <= (h1a_vote or fa_in(0)) and (h2a_vote or fa_in(1)) and (h3a_vote or fa_in(2));
    when "01111000" => A_processed <= (h1a_vote and fa_in(0)) or (h2a_vote and fa_in(1)) or (h3a_vote and fa_in(2));
    when "01111001" => A_processed <= (h1a_vote and fa_in(0)) or (h2a_vote and fa_in(1)) or (h3a_vote and fa_in(2));
    when "01111010" => A_processed <= (h1a_vote and fa_in(0)) or (h2a_vote and fa_in(1)) or (h3a_vote and fa_in(2));
    when "01111011" => A_processed <= (h1a_vote and fa_in(0)) or (h2a_vote and fa_in(1)) or (h3a_vote and fa_in(2));
    when "01111100" => A_processed <= (h1a_vote and fa_in(0)) or (h2a_vote and fa_in(1)) or (h3a_vote and fa_in(2));
    when "01111101" => A_processed <= (h1a_vote and fa_in(0)) or (h2a_vote and fa_in(1)) or (h3a_vote and fa_in(2));
    when "01111110" => A_processed <= (h1a_vote and fa_in(0)) or (h2a_vote and fa_in(1)) or (h3a_vote and fa_in(2));
    when "01111111" => A_processed <= (h1a_vote and fa_in(0)) or (h2a_vote and fa_in(1)) or (h3a_vote and fa_in(2));
    when "10001111" => A_processed <= haserial_vote;
    when "10011000" => A_processed <= hamarkserial_vote;
    when "10011010" => A_processed <= hamarkserial_vote;
    when "10011011" => A_processed <= hamarkserial_vote;
    when "10101000" => A_processed <= haonemark_vote; --send onemark
    when "10101001" => A_processed <= haonemark_vote; --send onemark
    when "11001111" => A_processed <= hbfxread_vote; --readout b register
    when others => A_processed <= h1a_vote or h2a_vote or h3a_vote;
  end case;
end process;

--B_processed

muxB : process(fb_in, h1b_vote, h2b_vote, h3b_vote, hsserial_vote, hbmarkserial_vote, hbonemark_vote, hbfxread_vote)
begins
  case(fb_in) is
    when "00100001" => B_processed <= h1b_vote;
    when "00101010" => B_processed <= h2b_vote;
    when "00111010" => B_processed <= h3b_vote;
    when "01000111" => B_processed <= h1b_vote and h2b_vote and h3b_vote;
  end case;
end process;
when "01010111" => B_processed <= h1b_vote or h2b_vote or h3b_vote;
when "01100111" => B_processed <= (h1b_vote and h2b_vote) or (h2b_vote and h3b_vote) or (h1b_vote and h3b_vote);
when "01110000" => B_processed <= (h1b_vote or fb_in(0)) and (h2b_vote or fb_in(1)) and (h3b_vote or fb_in(2));
when "01110001" => B_processed <= (h1b_vote or fb_in(0)) and (h2b_vote or fb_in(1)) and (h3b_vote or fb_in(2));
when "01110010" => B_processed <= (h1b_vote or fb_in(0)) and (h2b_vote or fb_in(1)) and (h3b_vote or fb_in(2));
when "01110011" => B_processed <= (h1b_vote or fb_in(0)) and (h2b_vote or fb_in(1)) and (h3b_vote or fb_in(2));
when "01110100" => B_processed <= (h1b_vote and fb_in(0)) or (h2b_vote and fb_in(1)) and (h3b_vote or fb_in(2));
when "01110101" => B_processed <= (h1b_vote and fb_in(0)) or (h2b_vote and fb_in(1)) and (h3b_vote or fb_in(2));
when "01110110" => B_processed <= (h1b_vote and fb_in(0)) or (h2b_vote and fb_in(1)) and (h3b_vote or fb_in(2));
when "01110111" => B_processed <= (h1b_vote or fb_in(0)) and (h2b_vote or fb_in(1)) and (h3b_vote or fb_in(2));
when "10000000" => B_processed <= (h1b_vote and fb_in(0)) or (h2b_vote and fb_in(1)) or (h3b_vote and fb_in(2));
when "10000001" => B_processed <= (h1b_vote and fb_in(0)) or (h2b_vote and fb_in(1)) or (h3b_vote and fb_in(2));
when "10000010" => B_processed <= (h1b_vote and fb_in(0)) or (h2b_vote and fb_in(1)) or (h3b_vote and fb_in(2));
when "10000011" => B_processed <= (h1b_vote and fb_in(0)) or (h2b_vote and fb_in(1)) or (h3b_vote and fb_in(2));
when "10000100" => B_processed <= (h1b_vote and fb_in(0)) or (h2b_vote and fb_in(1)) or (h3b_vote and fb_in(2));
when "10000101" => B_processed <= (h1b_vote and fb_in(0)) or (h2b_vote and fb_in(1)) or (h3b_vote and fb_in(2));
when "10000110" => B_processed <= (h1b_vote and fb_in(0)) or (h2b_vote and fb_in(1)) or (h3b_vote and fb_in(2));
when "10000111" => B_processed <= (h1b_vote and fb_in(0)) or (h2b_vote and fb_in(1)) or (h3b_vote and fb_in(2));
when "10100000" => B_processed <= hbserial_vote;
when "10100001" => B_processed <= hmarkserial_vote;
when "10100010" => B_processed <= hmarkserial_vote;
when "10100011" => B_processed <= hmarkserial_vote;
when "10110000" => B_processed <= hbonemark_vote; --send onemark
when "10110001" => B_processed <= hbonemark_vote; --send onemark
when "11010000" => B_processed <= hafxread_vote; --readout a register
when others => B_processed <= h1b_vote or h2b_vote or h3b_vote;
end case;
end process;

-- triply redundant shift register for hit signals, no marker bits LSB comes out first!
haserial(0) <= h1a_vote; hastream(1) <= h2a_vote; hastream(2) <= h3a_vote;
hbstream(0) <= h1b_vote; hbstream(1) <= h2b_vote; hbstream(2) <= h3b_vote;

piso3a1 : piso3g
port map (haserial, clk1x, internal_reset, load3_vote, haserial11);
piso3a2 : piso3g
port map (haserial, clk1x, internal_reset, load3_vote, haserial2);
piso3a3 : piso3g
port map (haserial, clk1x, internal_reset, load3_vote, haserial3);
haserial_vote <= (haserial1 and haserial2) or (haserial2 and haserial3) or (haserial1 and haserial3);
piso3b1 : piso3g
port map (hbstream, clk1x, internal_reset, load3_vote, hserial11);
piso3b2 : piso3g
port map (hbstream, clk1x, internal_reset, load3_vote, hbserial2);

piso3b3 : piso3g
port map (hbstream, clk1x, internal_reset, load3_vote, hbserial3);

hbserial_vote <= (hbserial1 and hbserial2) or (hbserial2 and hbserial3) or (hbserial1 and hbserial3);

-- triply redundant shift register for hit signals, with marker bits LSB comes out first!
hamarkstream(0) <= h1a_vote; hamarkstream(1) <= h2a_vote; hamarkstream(2) <= h3a_vote;
hamarkstream(3) <= fa_in(0); hamarkstream(4) <= fa_in(1);
hbmarkstream(0) <= h1b_vote; hbmarkstream(1) <= h2b_vote; hbmarkstream(2) <= h3b_vote;
hbmarkstream(3) <= fb_in(0); hbmarkstream(4) <= fb_in(1);

piso5a1 : piso5g
port map (hamarkstream, clk1x, internal_reset, load5_vote, hamarkserial1);

piso5a2 : piso5g
port map (hamarkstream, clk1x, internal_reset, load5_vote, hamarkserial2);

piso5a3 : piso5g
port map (hamarkstream, clk1x, internal_reset, load5_vote, hamarkserial3);

hamarkserial_vote <= (hamarkserial1 and hamarkserial2) or (hamarkserial2 and hamarkserial3) or (hamarkserial1 and hamarkserial3);

piso5b1 : piso5g
port map (hbmarkstream, clk1x, internal_reset, load5_vote, hbmarkserial1);

piso5b2 : piso5g
port map (hbmarkstream, clk1x, internal_reset, load5_vote, hbmarkserial2);

piso5b3 : piso5g
port map (hbmarkstream, clk1x, internal_reset, load5_vote, hbmarkserial3);

hbmarkserial_vote <= (hbmarkserial1 and hbmarkserial2) or (hbmarkserial2 and hbmarkserial3) or (hbmarkserial1 and hbmarkserial3);

haonestream(0) <= h1a_vote; haonestream(1) <= h2a_vote; haonestream(2) <= h3a_vote;
haonestream(3) <= fa_in(0);
hbonestream(0) <= h1b_vote; hbonestream(1) <= h2b_vote; hbonestream(2) <= h3b_vote;
hbonestream(3) <= fb_in(0);

piso4a1 : piso4g
port map (haonestream, clk1x, internal_reset, load4_vote, haoneserial1);

piso4a2 : piso4g
port map (haonestream, clk1x, internal_reset, load4_vote, haoneserial2);

piso4a3 : piso4g
port map (haonestream, clk1x, internal_reset, load4_vote, haoneserial3);

haonestream_vote <= (haoneserial1 and haoneserial2) or (haoneserial2 and haoneserial3) or (haoneserial1 and haoneserial3);

piso4b1 : piso4g
port map (hbonestream, clk1x, internal_reset, load4_vote, hboneserial1);

piso4b2 : piso4g
port map (hbonestream, clk1x, internal_reset, load4_vote, hboneserial2);

piso4b3 : piso4g
port map (hbonestream, clk1x, internal_reset, load4_vote, hboneserial3);

hbonestream_vote <= (hboneserial1 and hboneserial2) or (hboneserial2 and hboneserial3) or (hboneserial1 and hboneserial3);
hafxstream(0) <= fa_in(0);
hafxstream(1) <= fa_in(1);
hafxstream(2) <= fa_in(2);
hafxstream(3) <= fa_in(3);
hafxstream(4) <= fa_in(4);
hafxstream(5) <= fa_in(5);
hafxstream(6) <= fa_in(6);
hafxstream(7) <= fa_in(7);

piso8a1 : piso8g
  port map (hafxstream, clk1x, internal_reset, load8_vote, hafxread1);
piso8a2 : piso8g
  port map (hafxstream, clk1x, internal_reset, load8_vote, hafxread2);
piso8a3 : piso8g
  port map (hafxstream, clk1x, internal_reset, load8_vote, hafxread3);

hafxread_vote <= (hafxread1 and hafxread2) or (hafxread2 and hafxread3) or (hafxread1 and hafxread3);

hbfxstream(0) <= fb_in(0);
hbfxstream(1) <= fb_in(1);
hbfxstream(2) <= fb_in(2);
hbfxstream(3) <= fb_in(3);
hbfxstream(4) <= fb_in(4);
hbfxstream(5) <= fb_in(5);
hbfxstream(6) <= fb_in(6);
hbfxstream(7) <= fb_in(7);

piso8b1 : piso8g
  port map (hbfxstream, clk1x, internal_reset, load8_vote, hbfxread1);
piso8b2 : piso8g
  port map (hbfxstream, clk1x, internal_reset, load8_vote, hbfxread2);
piso8b3 : piso8g
  port map (hbfxstream, clk1x, internal_reset, load8_vote, hbfxread3);

hbfxread_vote <= (hbfxread1 and hbfxread2) or (hbfxread2 and hbfxread3) or (hbfxread1 and hbfxread3);

counter3a_int : process (clk1x, internal_reset)
begin
  if internal_reset = '1' then
    cnt3a <= "00";
    load3a <= '0';
  elsif (clk1x'event and clk1x = '1') then
    case (cnt3a) is
      when "00" => cnt3a <= "01";
      load3a <= '0';
      when "01" => cnt3a <= "10";
      when "10" => cnt3a <= "00";
      load3a <= '1';
      when others => cnt3a <= "00";
    end case;
  end if;
end process;

counter3b_int : process (clk1x, internal_reset)
begin
  if internal_reset = '1' then
    cnt3b <= "00";
    load3b <= '0';
  elsif (clk1x'event and clk1x = '1') then
    case (cnt3b) is
      when "00" => cnt3b <= "01";
      when "01" => cnt3b <= "10";
      when "10" => cnt3b <= "00";
      load3b <= '1';
    end case;
  end if;
end process;
load3b <= '0';
when "01" => cnt3b <= "10";
when "10" => cnt3b <= "00";
load3b <= '1';
when others => cnt3b <= "00";
end case;
end if;
end process;

counter3c_int : process (clk1x, internal_reset)
begin
if internal_reset = '1' then
cnt3c <= "00";
load3c <= '0';
elsif (clk1x'event and clk1x = '1') then
  case (cnt3c) is
    when "00" => cnt3c <= "01";
    load3c <= '0';
    when "01" => cnt3c <= "10";
    load3c <= '0';
    when "10" => cnt3c <= "00";
    load3c <= '1';
    when others => cnt3c <= "00";
  end case;
end if;
end process;

load3_vote <= (load3a and load3b) or (load3b and load3c) or (load3a and load3c);

counter4a_int : process (clk1x, internal_reset)
begin
if internal_reset = '1' then
cnt4a <= "000";
load4a <= '0';
elsif (clk1x'event and clk1x = '1') then
  case (cnt4a) is
    when "000" => cnt4a <= "001";
    load4a <= '0';
    when "001" => cnt4a <= "010";
    when "010" => cnt4a <= "011";
    when "011" => cnt4a <= "000";
    load4a <= '1';
    when others => cnt4a <= "000";
  end case;
end if;
end process;

counter4b_int : process (clk1x, internal_reset)
begin
if internal_reset = '1' then
cnt4b <= "000";
load4b <= '0';
elsif (clk1x'event and clk1x = '1') then
  case (cnt4b) is
    when "000" => cnt4b <= "001";
    load4b <= '0';
    when "001" => cnt4b <= "010";
    when "010" => cnt4b <= "011";
    when "011" => cnt4b <= "000";
    load4b <= '1';
    when others => cnt4b <= "000";
  end case;
end if;
end process;

counter4c_int : process (clk1x, internal_reset)
begin
if internal_reset = '1' then
cnt4c <= "000";
load4c <= '0';
elsif (clk1x'event and clk1x = '1') then
  case (cnt4c) is
  when "000" => cnt4c <= "001";
    load4c <= '0';
  when "001" => cnt4c <= "010";
  when "010" => cnt4c <= "011";
  when "011" => cnt4c <= "000";
    load4c <= '1';
  when others => cnt4c <= "000";
  end case;
end if;
end process;

load4_vote <= (load4a and load4b) or (load4b and load4c) or (load4a and load4c);
counter5a_int : process (clk1x, internal_reset)
begin
  if internal_reset = '1' then
    cnt5a <= "000";
    load5a <= '0';
  elsif (clk1x'event and clk1x = '1') then
    case (cnt5a) is
    when "000" => cnt5a <= "001";
      load5a <= '0';
    when "001" => cnt5a <= "010";
    when "010" => cnt5a <= "011";
    when "011" => cnt5a <= "100";
    when "100" => cnt5a <= "000";
      load5a <= '1';
    when others => cnt5a <= "000";
    end case;
  end if;
end process;
counter5b_int : process (clk1x, internal_reset)
begin
  if internal_reset = '1' then
    cnt5b <= "000";
    load5b <= '0';
  elsif (clk1x'event and clk1x = '1') then
    case (cnt5b) is
    when "000" => cnt5b <= "001";
      load5b <= '0';
    when "001" => cnt5b <= "010";
    when "010" => cnt5b <= "011";
    when "011" => cnt5b <= "100";
    when "100" => cnt5b <= "000";
      load5b <= '1';
    when others => cnt5b <= "000";
    end case;
  end if;
end process;
counter5c_int : process (clk1x, internal_reset)
begin
  if internal_reset = '1' then
    cnt5c <= "000";
    load5c <= '0';
  elsif (clk1x'event and clk1x = '1') then
    case (cnt5c) is
    when "000" => cnt5c <= "001";
      load5c <= '0';
    when "001" => cnt5c <= "010";
    when "010" => cnt5c <= "011";
    when "011" => cnt5c <= "100";
    when others => cnt5c <= "000";
    end case;
end if;
end process;
when "100" => cnt5c <= "000";
load5c <= '1';
when others => cnt5c <= "000";
end case;
end if;
end process;
load5_vote <= (load5a and load5b) or (load5b and load5c) or (load5a and load5c);
counter8a_int : process (clk1x, internal_reset)
begin
if internal_reset = '1' then
  cnt8a <= "000";
  load8a <= '0';
elseif (clk1x'event and clk1x = '1') then
  case (cnt8a) is
  when "000" => cnt8a <= "001";
    load8a <= '0';
  when "001" => cnt8a <= "010";
  when "010" => cnt8a <= "011";
  when "011" => cnt8a <= "100";
  when "100" => cnt8a <= "101";
  when "101" => cnt8a <= "110";
  when "110" => cnt8a <= "111";
  when "111" => cnt8a <= "000";
    load8a <= '1';
  when others => cnt8a <= "000";
  end case;
end if;
end process;
counter8b_int : process (clk1x, internal_reset)
begin
if internal_reset = '1' then
  cnt8b <= "000";
  load8b <= '0';
elseif (clk1x'event and clk1x = '1') then
  case (cnt8b) is
  when "000" => cnt8b <= "001";
    load8b <= '0';
  when "001" => cnt8b <= "010";
  when "010" => cnt8b <= "011";
  when "011" => cnt8b <= "100";
  when "100" => cnt8b <= "101";
  when "101" => cnt8b <= "110";
  when "110" => cnt8b <= "111";
  when "111" => cnt8b <= "000";
    load8b <= '1';
  when others => cnt8b <= "000";
  end case;
end if;
end process;
counter8c_int : process (clk1x, internal_reset)
begin
if internal_reset = '1' then
  cnt8c <= "000";
  load8c <= '0';
elseif (clk1x'event and clk1x = '1') then
  case (cnt8c) is
  when "000" => cnt8c <= "001";
    load8c <= '0';
  when "001" => cnt8c <= "010";
  when "010" => cnt8c <= "011";
  when "011" => cnt8c <= "100";
  when "100" => cnt8c <= "101";
  when "101" => cnt8c <= "110";
  when others => cnt8c <= "000";
end case;
end if;
end process;
when "110" => cnt8c <= "111";
when "111" => cnt8c <= "000";
    load8c <= '1';
when others => cnt8c <= "000";
end case;
end if;
end process;

load8_vote <= (load8a and load8b) or (load8b and load8c) or (load8a and load8c);
end Behavioral;
A.5: Hitbus Chip BPM Encoder: BPM_encoder.vhd

library IEEE;
use IEEE.STD_LOGIC_1164.all;

design Name:BPM_encoder.vhd
description: a Bi-phase mark encoder

entity BPM_encoder is
port (
    dd_clk : in std_logic;
    reset : in std_logic;
    data : in std_logic;
    encode : in std_logic;
    encoded_out : out std_logic);
end BPM_encoder;

architecture behavioral of BPM_encoder is
begin
    process(dd_clk, data, reset)
    begin
        if reset = '1' then
            last_state <= '0';
            state <= "00";
        elsif dd_clk = '1' and dd_clk'event then
            case state is
                when "00" =>
                    if last_state = '1' then
                        last_state <= '0';
                    end if;
                    if data = '1' then
                        state <= "01";
                    else
                        state <= "10";
                    end if;
                when "10" =>
                    state <= "00";
                when "01" =>
                    last_state <= '1';
                    if data = '1' then
                        state <= "11";
                    else
                        state <= "10";
                        last_state <= '1';
                    end if;
                when "11" =>
                    last_state <= '0';
                    if data = '1' then
                        state <= "11";
                    else
                        state <= "10";
                        last_state <= '1';
                    end if;
                when others => state <= "00";
            end case;
        end if;
    end process;

    enc_s : process(dd_clk, encode, last_state, data)
    begin
        if dd_clk = '1' and dd_clk'event then
            if encode = '1' then
                encoded_out <= last_state;
            else
                encoded_out <= '0';
            end if;
        end if;
    end process;
end behavioral;
end if;
end if;
end process;
end behavioral;
library IEEE, fvhdl;
use IEEE.STD_LOGIC_1164.all, fvhdl.all;
use IEEE.VITAL_Timing.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;
use IEEE.std_logic_textio.all;

entity pll_wrap is
port (
    fast_clk : in std_logic;
    processor_reset : in std_logic;
    clksel : in std_logic_vector (5 downto 0);
    div_clk : out std_logic;
    clk_0 : out std_logic;
    clk_1 : out std_logic
);
end pll_wrap;

architecture Behavioral of pll_wrap is
    signal clk320 : std_logic;
    signal clk160 : std_logic;
    signal clk80 : std_logic;
    signal clk40 : std_logic;

    component div2
    port(
        clk : in std_logic;
        reset : in std_logic;
        outp : out std_logic);
    end component;

begin
    div_clk <= clk40;
    feq_div2 : div2
    port map (fast_clk, processor_reset, clk320);
    feq_div4 : div2
    port map (clk320, processor_reset, clk160);
    feq_div8 : div2
    port map (clk160, processor_reset, clk80);
    feq_div16 : div2
    port map (clk80, processor_reset, clk40);
    clkmux1 : process(clksel, clk320, clk160, clk80, clk40)
    begin
        case (clksel(5 downto 3)) is
        when "000" => clk_1 <= clk320;
        when "001" => clk_1 <= clk160;
        when "010" => clk_1 <= clk320;
        when "011" => clk_1 <= clk40;
        when "100" => clk_1 <= clk320;
        when "101" => clk_1 <= clk80;
        when "110" => clk_1 <= clk320;
        when "111" => clk_1 <= clk320;
        when others => clk_1 <= clk320;
        end case;
    end process;
end behavioral;
clkmux2 : process(clksel, clk320, clk160, clk80, clk40)
begin
  case(clksel(2 downto 0)) is
    when "000" => clk_0 <= clk320;
    when "001" => clk_0 <= clk160;
    when "010" => clk_0 <= clk320;
    when "011" => clk_0 <= clk40;
    when "100" => clk_0 <= clk320;
    when "101" => clk_0 <= clk80;
    when "110" => clk_0 <= clk320;
    when "111" => clk_0 <= clk320;
    when others => clk_0 <= clk320;
  end case;
end process;
end Behavioral;
A.7: Hitbus Chip Triply Redundant Frequency Divide by Two: div2.vhd

-- Design Name: div2.vhd
-- Description: frequency divide by 2 triply redundant

library IEEE, fvhdl;
use IEEE.STD_LOGIC_1164.all, fvhdl.all;
use IEEE.VITAL_Timing.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;
use IEEE.std_logic_textio.all;

entity div2 is
  port ( 
    clk : in std_logic;
    reset : in std_logic;
    outp : out std_logic 
  );
end div2;

architecture Behavioral of div2 is

  signal vote : std_logic;
  signal qbar1 : std_logic;
  signal qbar2 : std_logic;
  signal qbar3 : std_logic;
  signal q1 : std_logic;
  signal q2 : std_logic;
  signal q3 : std_logic;

  component dff2
    port(D, clk, rst : in std_logic;
         q , qbar : out std_logic)
  end component;

begin

dff_A : dff2
  port map (vote, clk, reset, q1, qbar1);

dff_B : dff2
  port map (vote, clk, reset, q2, qbar2);

dff_CA : dff2
  port map (vote, clk, reset, q3, qbar3);

  vote <= (qbar1 and qbar2) or (qbar2 and qbar3) or (qbar1 and qbar3);
  outp <= (q1 and q2) or (q2 and q3) or (q1 and q3);

end Behavioral;
Appendix B: Synthesis, Placement, and Routing Scripts
B.1: Software / Computing Used

The PNR scripts to follow were tested under Red Hat Enterprise Linux 4 and Red Hat Enterprise Linux 5. The EDA software tools called were limited to what was available under the Cadence University Software Program license. The versions of the Cadence tools tested were:

**Encounter Digital Implementation System:**
Hotfix_EDI09.14.000_lnx86

**Cadence Incisive Verification Kit:**
Update_INCISIV09.20.038_lnx86

**Cadence RTL Compiler:**
Update_RC09.10.204_lnx86
B.2: Scripts to Perform Logic Synthesis

“run_rc”

rc -gui -files ../scripts/rc.tcl
# add the power and gnd pins for schematic
perl ../scripts/addpower.pl ../output/r2g.v ../output/r2g_w_pgs.v

“rc.tcl”

if {{catch {

    source ../scripts/init.tcl

    # Write the netlist
    write -m > $ec::outDir/r2g.v

    # Write SDC file
    write_sdc > $ec::outDir/r2g.sdc

    # Write RC script file
    write_script > $ec::outDir/r2g.g

    # Write LEC file
    write_do_lec -no_exit -revised_design $ec::outDir/r2g.v
    >../../lec/scripts/rtl2map.tcl

    report timing -full

    # end timer
    puts "\nEC INFO: End at: [clock format [clock seconds] -format {%-x %X}"
    set ec::end [clock seconds]
    set ec::seconds [expr $ec::end - $ec::start]
    puts "\nEC INFO: Elapsed-time: $ec::seconds seconds\n"

    # done
    #exit

}} msg} {

    puts "\nEC ERROR: RC could not finish successfully. Force an exit now. ($msg)\n"
    #exit -822

}
namespace eval ec {}
# start timer
puts "Start at: [clock format [clock seconds] -format {%x %X}]"
set ec::start [clock seconds]

# Setup file, directories, and variables
set ec::inDir           ../vhdl
set ec::outDir          ../output
set ec::reportDir       ../report
set ec::SYN_EFFORT      high
set ec::MAP_EFFORT      high
set ec::INCR_EFFORT     high
set PROCESS_PDK $env(PROCESS_PDK)
set PDK_OPT $env(PDK_OPT)
set CORE_CHIP  CORE
set DFT     OFF
set ec::RTL_PATH        ../vhdl/
set ec::LIB_PATH        "$PROCESS_PDK"
set ec::LIBRARY
"$PROCESS_PDK/PROCESS_node/std_cell/rel$PDK_OPT/synopsys/slow_v140_t125
/PROCESS_NODE_NODE_SC_SLOW_V140_T125.lib \
$PROCESS_PDK/PROCESS_node/short_io/rel$PDK_OPT/synopsys/slow_v135_t125
_pv162/PROCESS_NODE_BASE_SHORT_IO_SLOW_V135_T125_PV162.lib "

set ec::VERILOG_LIST    {  }
set ec::VERILOG_VERSION 2001
set ec::VHDL LIST       { BPM_encoder.vhd hitor_processor.vhd
piso3g.vhd piso5g.vhd piso4g.vhd piso8g.vhd dff.vhd }
set ec::VHDL_VERSION    1993
if {$PDK_OPT == "DM"} {
    set ec::LEFLIB
"$PROCESS_PDK/PROCESS_node/std_cell/rel$PDK_OPT/lef/PROCESS8ma_mg.lef \
$PROCESS_PDK/PROCESS_node/short_io/rel$PDK_OPT/lef/PROCESS_NODE_N
ODE_SC.lef "

$PROCESS_PDK/PROCESS_node/short_io/rel$PDK_OPT/lef/process_short_
io_5V_8MA.lef "
    set ec::CAPTABLE
"$PROCESS_PDK/PROCESS_node/std_cell/V2.0/captable/nodeprocess_8MA_32_nm .CapTbl"
}
if {$PDK_OPT == "LM"} {
```tcl
set ec::LEFLIB
"$PROCESS_PDK/PROCESS_node/std_cell/rel$PDK_OPT/lef/tech8rf8l8m_lm.lef \\
$PROCESS_PDK/PROCESS_node/std_cell/rel$PDK_OPT/lef/PROCESS_NODE_NODE_SC.
.lef \\
$PROCESS_PDK/PROCESS_node/short_io/rel$PDK_OPT/lef/process_short_io_5V_8LM.lef"

set ec::CAPTABLE
"$PROCESS_PDK/PROCESS_node/std_cell/V2.0/captable/node_8LM_62_nm.CapTbl"
}
set ec::SDC       ../sdc/constraint.sdc
#set ec::SDC     /dev/null
#        ../sdc/constraint.sdc
set ec::SUPPRESS_MSG   {LBR-30 LBR-31 VLOGPT-35}
#include needed script
# include needed script
#include load_etc.tcl
#####################################################################
# Preset global variables and attributes
#####################################################################
# define diagnostic variables
set iopt_stats 1
set global_map_report 1
set map_fancy_names 1
set path_disable_priority 0
# set report_unfolding 1
# set cost_grp_details_in_iopt 1
# define diagnostic variables
set iopt_stats 1
set global_map_report 1
set map_fancy_names 1
set path_disable_priority 0
# set report_unfolding 1
# set cost_grp_details_in_iopt 1
# define QoR related variables
# set global_area 2 ; # valid range: 0 through 6
# set dont_downsize_components 1
# set map_slackq 0
# set final_remaps 0
# set initial_target 0
# set crr 1
set crr_max_tries 300 ; # higher the number, more iterations: not much runtime penalty
# define tool setup and compatibility
set_attribute information_level 9 / ; # valid range: 1 (least verbose) through 9 (most verbose)
set_attribute hdl_max_loop_limit 1024 /
set_attribute hdl_reg_naming_style %s_reg%s /
set_attribute gen_module_prefix G2C_DP_ /
# set_attribute endpoint_slack_opto 1 /
#set_attribute optimize_constant_0_flops false /
#set_attribute optimize_constant_1_flops false /
set_attribute inputPragma_keyword {cadence synopsys get2chip g2c} /
set_attribute synthesis_off_command translate_off /
set_attribute synthesis_on_command translate_on /
set_attribute input_case_coverPragma {full_case} /
set_attribute input_case_decodePragma {parallel_case} /
set_attribute input_synchro_resetPragma sync_set_reset /
```
set_attribute input_synchro_reset_blk_pragma sync_set_reset_local /
set_attribute input_asynchro_reset_pragma async_set_reset /
set_attribute input_asynchro_reset_blk_pragma async_set_reset_local /
set_attribute delayed_pragma_commands_interpreter dc /
set_attribute script_begin dc_script_begin /
set_attribute script_end dc_script_end /
set_attribute iopt_force_constant_removal true /
# suppress messages
suppress_messages $ec::SUPPRESS_MSG
# setup shrink_factor attribute
set_attribute shrink_factor 1.0 /

#########################################################################
# RTL and libraries setup
#########################################################################
# search paths
set_attribute hdl_search_path $ec::RTL_PATH /
set_attribute lib_search_path $ec::LIB_PATH /
# timing libraries
set_attribute library $ec::LIBRARY
# lef & captbl
set_attribute lef_library $ec::LEFLIB /
set_attribute cap_table_file $ec::CAPTABLE /
set_attribute interconnect_mode ple /
# report time and memory
puts "\nEC INFO: Total cpu-time and memory after SETUP: [get_attr runtime /] sec., [get_attr memory_usage /] MBytes.\n"
### Power root attributes
set_attribute lp_insert_clock_gating false /
set_attribute lp_clock_gating_prefix lpg /
set_attribute lp_insert_operand_isolation true /
set_attribute hdl_track_filename_row_col true /

## Power root attributes -NEW
#set_attribute lp_insert_clock_gating true /
#set_attribute lp_clock_gating_prefix <string> /
#set_attribute lp_insert_operand_isolation true /
#set_attribute lp_operand_isolation_prefix <string> /
#set_attribute lp_power_analysis_effort <high> /
#set_attribute lp_power_unit mW /
#set_attribute lp_toggle_rate_unit /ns /
#set_attribute hdl_track_filename_row_col true /
#set_attribute hdl_language sv /
# Load RTL
#########################################################################
#set_attribute hdl_language vhd1 /
#set_attribute hdl_infer_unresolved_from_logic_abstract true
set_attribute hdl_infer_unresolved_from_logic_abstract false
read_hdl $ec::VHDL_LIST

set_attribute optimize_merge_flops false /
set_attribute optimize_merge_latches false /
# report time and memory
puts "\nEC INFO: Total cpu-time and memory after LOAD: [get_attr runtime /] sec., [get_attr memory_usage /] MBytes."

#####################################################################
# Elaborate
#####################################################################
elaborate

# report time and memory
puts "\nEC INFO: Total cpu-time and memory after ELAB: [get_attr runtime /] sec., [get_attr memory_usage /] MBytes."

#####################################################################
# Constraint setup
#####################################################################
# read sdc constraint
foreach ec::FILE_NAME $ec::SDC {
  read_sdc $ec::FILE_NAME
}
###add buffers to every pin
insert_io_buffers -isolate_top inouts

# report time and memory
puts "\nEC INFO: Total cpu-time and memory after CONSTRAINT: [get_attr runtime /] sec., [get_attr memory_usage /] MBytes."

#####################################################################
# Define cost groups (clock-clock, clock-output, input-clock, input-output)
#####################################################################
#define_cost_group -name I2C
#define_cost_group -name C2O
#define_cost_group -name I2O
#define_cost_group -name C2C
#path_group -from [all des seqs] -to [all des seqs] -group C2C -name C2C
#path_group -from [all des seqs] -to [all des outs] -group C2O -name C2O
#path_group -from [all des inps] -to [all des seqs] -group I2C -name I2C
#path_group -from [all des inps] -to [all des outs] -group I2O -name I2O

#####################################################################
# Initial reports
#####################################################################
# print out the exceptions
set ec::XCEP [find /designs* -exception *]
puts "\nEC INFO: Total numbers of exceptions: [llength $ec::XCEP]\n" catch {open $ec::reportDir/exception.all "w"} ec::FXCEP

158
puts $ec::FXCEP "Total numbers of exceptions: [llength $ec::XCEP]\n"
foreach ec::X $ec::XCEP {
    puts $ec::FXCEP $ec::X
}
close $ec::FXCEP

# report time and memory
puts "\nEC INFO: Total cpu-time and memory after POST-SDC: [get_attr runtime /] sec., [get_attr memory_usage /] MBytes.\n"

# report initial design
report design > $ec::reportDir/init.design

# report initial gates
report gates > $ec::reportDir/init.gate

# report initial area
report area > $ec::reportDir/init.area

# report initial timing
report timing -full > $ec::reportDir/init.timing

# report initial timing groups
report timing -end -slack 0 > $ec::reportDir/init.timing.ep
report timing -from [dc::all_inputs] > $ec::reportDir/init.timing.in
report timing -to [dc::all_outputs] > $ec::reportDir/init.timing.out
set ec::CNT 1
foreach ec::CLK [find /designs* -clock *] {
    exec echo "####################" >
    $ec::reportDir/init.timing.clk$ec::CNT
    exec echo "# from clock: $ec::CLK" >>
    $ec::reportDir/init.timing.clk$ec::CNT
    exec echo "# to clock: $ec::CLK" >>
    $ec::reportDir/init.timing.clk$ec::CNT
    exec echo "####################" >>
    $ec::reportDir/init.timing.clk$ec::CNT
    report timing -from $ec::CLK -to $ec::CLK >>
    $ec::reportDir/init.timing.clk$ec::CNT
    incr ec::CNT
}

# report initial summary
puts "\nEC INFO: Reporting Initial QoR below...\n"
redirect -tee $ec::reportDir/init.qor {report qor}
puts "\nEC INFO: Reporting Initial Summary below...\n"
redirect -tee $ec::reportDir/init.summary {report summary}

report timing -lint
#########################################################################
# DFT
#########################################################################
set ec::DESIGN [find * -design *]
“addpower.pl”

#!/usr/bin/perl
if(@ARGV<2) {
  die("Please enter 2 inputs: E.g. addpower.pl input.v output.v\n"
} #if
if(!open(inPort, @ARGV[0])) {
  die("Cannot open file @ARGV[0]\n"
} #if
if(!open(outPort, "@ARGV[1]")) {
  die("Cannot create file @ARGV[1]\n"
} #if
$pin1="VDD";
$pin2="GND";
$pin3="NW";
$pin4="SX";
$first_input = 0;
while( chomp($inLine=<inPort>) ) {
  $inLine2 = $inLine;
  $inLine2 =~ s/^ *///;
  @myList=split / /, $inLine2;
  if( @myList[0] =~ /module/ ) {
    $first_input = 0;
    $inLine =~ s/\(/($pin1, $pin2, $pin3, $pin4, /;
  } #if
  if( @myList[0] =~ /input/ && $first_input == 0 ) {
    #   if( @myList[0] =~ /input/ ) {
    $first_input = 1;
    $inLine =~ s/input/inout $pin1, $pin2, $pin3, $pin4;\n  input/;
  } #if
  if( $inLine =~ /^\[ \t\]*[A-Z0-9a-z].*(..*\(.*\).*/ ) {
    if( $inLine !~ /^module / ) {
      # print "No module:$inLine\n";
      $inLine =~ s/\(/\(\n    .$pin1\($pin1\),\n    .\$pin2\($pin2\),\n    .\$pin3\($pin3\),\n    .\$pin4\($pin4\),\n    /; 
    } #else {
    #print "Found module:$inLine\n";
  } #if
  print outPort $inLine, "\n";
} #while
### B.3: Cadence Encounter Place and Route Script

**“encounter.cmd”**

```plaintext
loadConfig hitbus_block.conf 0
commitConfig
setDrawView fplan
addRing -spacing_bottom 5 -width_left 10 -width_bottom 10 -width_top 10
-spacing_top 5 -layer_bottom MG -stacked_via_top_layer MA -width_right
10 -around core -jog_distance 0.2 -offset_bottom 6 -layer_top MG -
threshold 0.2 -offset_left 6 -spacing_right 5 -spacing_left 5 -
offset_right 6 -offset_top 6 -layer_right MQ -nets {GND VDD } -
stacked_via_bottom_layer M1 -layer_left MQ
addStripe -block_ring_top_layer_limit M3 -max_same_layer_jog_length 0 -
padcore_ring_bottom_layer_limit M1 -set_to_set_distance 60 -
stacked_via_top_layer MA -padcore_ring_top_layer_limit M3 -spacing 5 -
xleft_offset 40 -merge_stripes_value 0.2 -layer MQ -
block_ring_bottom_layer_limit M1 -width 5 -nets {GND VDD } -
stacked_via_bottom_layer M1
addWellTap -cell NWSX -cellInterval 14 -skipRow 0 -prefix WELLTAP
sroute -connect { blockPin padPin padRing corePin floatingStripe } -
layerChangeRange { M1 MA } -blockPinTarget { nearestRingStripe
nearestTarget } -padPinPortConnect { allPort oneGeom } -
checkAlignedSecondaryPin 1 -blockPin useLef -allowJogging 1 -
crossoverViaBottomLayer M1 -allowLayerChange 1 -targetViaTopLayer MA -
crossoverViaTopLayer MA -targetViaBottomLayer M1 -nets { GND VDD }
setNanoRouteMode -quiet -routeTopRoutingLayer default
setPlaceMode -reset
setPlaceMode -congEffort medium -timingDriven 1 -modulePlan 1 -
doCongOpt 0 -clkGateAware 0 -powerDriven 0 -ignoreScan 1 -reorderScan 1
-ignoreSpare 1 -placeIOPins 1 -moduleAwareSpare 0 -
checkPinLayerForAccess { 1 } -preserveRouting 0 -rmAffectedRouting 0 -
checkRoute 0 -swapEEQ 0
setTrialRouteMode -highEffort true -floorPlanMode false -detour true -
maxRouteLayer 3 -handlePreroute true -autoSkipTracks false -
handlePartition true -handlePartitionComplex false -useM1 false -
keepExistingRoutes false -ignoreAbutted2TermNet false -pinGuide true -
honorPin false -selNet {} -selNetOnly {} -selMarkedNet false -
setMarkedNetOnly false -ignoreObstruct false -PKS false -
updateRemainTrks false -ignoreDEFFtrack false -printWiresOnRTBlk false -
usePagedArray false -routeObs true -routeGuide {} -blockageCostMultiple
1 -maxDetourRatio 0
scanTrace
setDrawView place
placeDesign -inPlaceOpt -prePlaceOpt
setNanoRouteMode -drouteStartIteration default
setNanoRouteMode -routeInsertAntennaDiode false
setNanoRouteMode -routeInsertDiodeForClockNets false
setNanoRouteMode -routeAntennaCellName {}
setNanoRouteMode -routeFixAntenna true
setNanoRouteMode -routeBottomRoutingLayer 0
setNanoRouteMode -routeTopRoutingLayer 3
setNanoRouteMode -drouteEndIteration default
```

161
setNanoRouteMode -droutePostRouteWidenWireRule NA
setNanoRouteMode -routeWithTimingDriven false
setNanoRouteMode -routeWithSiDriven false
setNanoRouteMode -drouteOnGridOnly via
setCTSMode -traceDPinAsLeaf false -traceIoPinAsLeaf false -routeClkNet true -routeGuide true -topPreferredLayer 3 -bottomPreferredLayer 2 -routeNonDefaultRule {} -useLefACLimit false -routePreferredExtraSpace 1 -opt true -optAddBuffer true -moveGate true -useHVRC true -fixLeafInst true -fixNonLeafInst true -verbose false -reportHTML false -addClockRootProp false -nameSingleDelim false -honorFence false -useLPROCESSaxFanout false -useLPROCESSaxCap false
setOptMode -effort high -leakagePowerEffort none -yieldEffort none -reclaimArea true -simplifyNetlist true -setupTargetSlack 0.5 -holdTargetSlack 0.5 -maxDensity 0.95 -drcMargin 0 -usefulSkew false
createClockTreeSpec -output Clock.ctstch -bufferList BUFFER_C BUFFER_D BUFFER_E BUFFER_P BUFFER_H BUFFER_I BUFFER_K BUFFER_L BUFFER_M BUFFER_N BUFFER_O CLKI_I CLKI_K CLKI_M CLK_I CLK_K CLK_M CLK_O INVERTBAL_E INVERTBAL_J INVERT_A INVERT_C INVERT_D INVERT_E INVERT_F INVERT_H INVERT_I INVERT_K INVERT_L INVERT_M INVERT_N INVERT_O
optDesign -preCTS -drv
createClockTreeSpec -output Clock.ctstch -bufferList BUFFER_C BUFFER_D BUFFER_E BUFFER_P BUFFER_H BUFFER_I BUFFER_K BUFFER_L BUFFER_M BUFFER_N BUFFER_O CLKI_I CLKI_K CLKI_M CLK_I CLK_K CLK_M CLK_O INVERTBAL_E INVERTBAL_J INVERT_A INVERT_C INVERT_D INVERT_E INVERT_F INVERT_H INVERT_I INVERT_K INVERT_L INVERT_M INVERT_N INVERT_O
clockDesign -specFile Clock.ctstch -outDir ../report/clock_report -fixedInstBeforeCTS
optDesign -postCTS -drv
setNanoRouteMode -routeTopRoutingLayer 3
routeDesign -globalDetail
timeDesign -postRoute -pathReports -drvReports -slackReports -numPaths 50 -prefix i2c_slave_postRoute -outDir ../report/timingReports
setOptMode -effort high -leakagePowerEffort none -yieldEffort none -reclaimArea true -simplifyNetlist true -setupTargetSlack 0.5 -holdTargetSlack 0.5 -maxDensity 0.95 -drcMargin 0 -usefulSkew false
optDesign -postRoute -drv
optDesign -postRoute -si -hold
clockDesign -specFile Clock.ctstch -outDir ../report/clock_report -fixedInstBeforeCTS
optDesign -postRoute -si -hold
setNanoRouteMode -routeInsertAntennaDiode true
setNanoRouteMode -routeInsertDiodeForClockNets true
setNanoRouteMode -routeAntennaCellName FTIE_G_A
routeDesign -globalDetail
setNanoRouteMode -droutePostRouteSwapVia multiCut
setNanoRouteMode -routeWithEco true
routeDesign -globalDetail -viaOpt
setNanoRouteMode -droutePostRouteSwapVia none
setFillerMode -reset
setFillerMode -corePrefix FILLER -createRows 1 -doDRC 1 -deleteFixed 1 -ecoMode 0
globalNetConnect GND -type pgpin -pin LT -inst DECAP_C
addFiller -cell FILL2 FILL1 NWSX -prefix FILLER
saveOaDesign hitor_processor hitor_processor
saveDesign routed.enc
saveNetlist ../output/post_route.v
saveNetlist ../output/lvs.v -excludeLeafCell fit
global rda_Input
set cwd
/n/15/ssmith/cadence/DM_2011/HITOR_LOGIC/hitor_processor_4bitmark/pnr/work
set rda_Input(import_mode) {-treatUndefinedCellAsBbox 0 -
  keepEmptyModule 1 }
set rda_Input(ui_netlist) "../../syn/output/r2g.v"
set rda_Input(ui_netlisttype) {Verilog}
set rda_Input(ui_rtllist) ""
set rda_Input(ui_ilmkdir) ""
set rda_Input(ui_ilm主持召开) ""
set rda_Input(ui_ilmpek) ""
set rda_Input(ui_settop) {1}
set rda_Input(ui_topcell) {hitor_processor}
set rda_Input(ui_celllib) ""
set rda_Input(ui_iolib) ""
set rda_Input(ui_areaiolib) ""
set rda_Input(ui_bkilib) ""
set rda_Input(ui_kboxlib) ""
set rda_Input(ui_gds_file) ""
set rda_Input(ui_oa_oa2lefversion) {}
set rda_Input(ui_core_height) {207.0}
set rda_Input(ui_core_width) {209.4}
set rda_Input(ui_core_to_left) {40.0}
set rda_Input(ui_core_to_right) {40.0}
set rda_Input(ui_core_to_top) {40.0}
set rda_Input(ui_core_to_bottom) {40.0}
set rda_Input(ui_max_io_height) {0}
set rda_Input(ui_row_height) {4.8}
set rda_Input(ui_isHorTrackHalfPitch) {0}
set rda_Input(ui_isVerTrackHalfPitch) {1}
set rda_Input(ui_ioOri) {R0}
set rda_Input(ui_isOrigCenter) {0}
set rda_Input(ui_isVerticalRow) {0}
set rda_Input(ui_exc_net) ""
set rda_Input(ui_delay_limit) {1000}
set rda_Input(ui_net_delay) {1000.0ps}
set rda_Input(ui_net_load) {0.5pf}
set rda_Input(ui_in_tran_delay) {0.1ps}
set rda_Input(ui_captbl_file) "-typical /data/cadence_installs/PROCESS_node/std_cell/rel/captable/node_8MA_32_n m.CapTbl -best /data/cadence_installs/ PROCESS_node/std_cell/rel/captable/node_8MA_32_SigCmin.CapTbl -worst /data/cadence_installs/PROCESS_node/std_cell/rel/captable/node_8MA_32_SigCmax.CapTbl"
set rda_Input(ui_preRoute_cap) {1}
set rda_Input(ui_postRoute_cap) {1}
set rda_Input(ui_postRoute_xcap) {1}
set rda_Input(ui_preRoute_res) {1}
set rda_Input(ui_postRoute_res) {1}
set rda_Input(ui_shr_scale) {1.0}
set rda_Input(ui_rel_c_thresh) {0.03}
set rda_Input(ui_tot_c_thresh) {5.0}
set rda_Input(ui_cpl_c_thresh) {3.0}
set rda_Input(ui_time_unit) {none}
set rda_Input(ui_cap_unit) {}
set rda_Input(ui_oa_reflib) {node short_io}
set rda_Input(ui_oa_abstractname) {abstract}
set rda_Input(ui_oa_layoutname) {layout}
set rda_Input(ui_sigstormlib) ""
set rda_Input(ui_cdb_file,min) ""
set rda_Input(ui_cdb_file,max) ""
set rda_Input(ui_cdb_file) ""
set rda_Input(ui_echo_file,min) ""
set rda_Input(ui_echo_file,max) ""
set rda_Input(ui_echo_file) ""
set rda_Input(ui_xtwf_file) ""
set rda_Input(ui_qxtech_file) ""/data/cadence_installs/ PROCESS_node/std_cell/rel/tch/node_8MA_32_nm.tch"
set rda_Input(ui_qxlayermap_file) ""
set rda_Input(ui_qxlib_file) ""
set rda_Input(ui_qxconf_file) ""
set rda_Input(ui_pwrnet) {VDD}
set rda_Input(ui_gndnet) {GND}
set rda_Input(flip_first) {1}
set rda_Input(double_back) {1}
set rda_Input(assign_buffer) {1}
set rda_Input(use_io_row_flow) {0}
set rda_Input(ui_pg_connections) ""
set rda_Input(ui_gen_footprint) {0}
Appendix C: Circuit Schematics and Layouts
C.1:DORIC12 Amplifier Top Level Schematic
C.2: DORIC12 Amplifier Top Level Layout
C.4:DORIC12 Analog Buffer Amplifier Schematic
C.5: DORIC12 Limiting Amplifier Top Level Schematic
C.6: DORIC12 Limiting Amplifier Layout
C.7: DORIC12 Resistively Loaded Differential Pair Amplifier Schematic
C8:DORIC12 DLL Top Level Schematic
C.9:DORIC12 DLL Top Level Layout
C.10: VDC12 Single Channel Top Level Schematic
C.11: VDC12 Single Channel Top Level Layout
C.12: Hitbus Chip PLL Top Level Schematic
C.13: Hitbus Chip PLL Top Level Layout
C.14: Hitbus Chip PLL Schematic
C.15: Hitbus Chip PLL Charge Pump Schematic
C.16: Hitbus Chip PLL Ring Oscillator Schematic
C.17: Hitbus Chip PLL Current Starved Inverter
C.18: LDO Top Level Schematic
C.19: LDO_AMP Schematic
Bibliography


188


[46] "BCM2835 High Definition 1080p Embedded Multimedia Applications Processor,"
Broadcom Corporation, [Online]. Available: 


Grecu and A. Ivanov, "System-on-Chip: Reuse and Integration," Proceedings of the 


IEEE Journal of Selected Topics in Quantum Electronics, vol. 6, no. 6, pp. 1201-
1215, 2000.

[56] Y. Zebda and S. Abu-Helweh, "Bandwidth improvement of a homojunction p-i-n 
photodiode," IEEE Journal of Quantum Electronics, vol. 33, no. 8, pp. 1333-1337, 
1997.

[57] G. Aad et al., "ATLAS pixel detector electronics and sensors," Journal of 

[58] K. E. Arms et al., "ATLAS pixel opto-electronics," Nuclear Instruments and 

[59] M. S. Cooke, "Studies of VCSEL Failures in the Optical Readout Systems of the 
ATLAS Silicon Trackers and Liquid Argon Calorimeters," in Proceedings of the


[67] The VCSEL array used is an ULM850-10-TN-N0112U, fabricated by ULM Photonics.


[70] The SFP+ Transceiver used is an FTLX8571D3BCL, fabricated by Finisar Corporation.

[72] The VCSEL array used is V850-2093-001, fabricated by Finisar Corporation.

[73] The epoxy used is Epotek 353ND, manufactured by Epoxy Technology Inc.


[75] The PIN array used is an ULMPIN-04-TN-U0112U, fabricated by ULM Photonics.


