Readout Driver Firmware Development for the ATLAS Insertable B-Layer

Shaw-Pin Chen

A thesis
submitted in partial fulfillment of the
requirements for the degree of

Master of Science in Electrical Engineering

University of Washington
2014

Committee:

Scott Hauck
Shih-Chieh Hsu

Program Authorized to Offer Degree:

Electrical Engineering

University of Washington

During the Large Hadron Collider shutdown from 2013 to 2014 a fourth silicon layer, called the Insertable-B Layer (IBL), was inserted inside the existing ATLAS Pixel Detector. The IBL uses the state-of-the-art FE-I4 front-end readout ASICs for enhanced detector readout efficiency during upcoming LHC runs at higher energy and luminosity. The control and data acquisition (DAQ) of the IBL requires the commissioning of new off-detector readout electronics, mainly consisting of Field-Programmable Gate Array (FPGA)-based Readout Driver (ROD) and Back-of-Crate (BOC) Cards. This thesis focuses on the architecture, implementation, simulation, and hardware test results of the new IBL ROD datapath firmware. Characterization of the IBL detector front-end and an overview of ATLAS Trigger DAQ (TDAQ) system are provided in the first chapters of the thesis. IBL ROD datapath firmware was designed and simulated in a ModelSim testbench with a realistic HDL FE-I4 model as source of data. The hardware tests using both real and emulated inputs have been performed for data taking. As this is an on-going development effort for the IBL collaboration, the outlook and near-term development goals for IBL ROD firmware are specified in this report.

Readout Driver Firmware Development for the ATLAS Insertable B-Layer

Shaw-Pin Chen

Chair of the Supervisory Committee:
Professor Scott Hauck
Electrical Engineering

Contents

Chapter 1

Introduction

The ATLAS experiment at the Large Hadron Collider (LHC) at CERN is aimed at discovering new subatomic particles that contribute to the fundamental building blocks of the Universe. During LHC operations, protons are accelerated to near light speed in opposing directions along a 27 km long underground circular track. The ATLAS detector, shown in figure 1, is located at the interaction point where opposing beams are guided to collide by superconducting magnets. Sub-detectors of ATLAS register various particle signatures produced by proton-proton collisions and the resulting data is used to reconstruct LHC events for physics analysis. The ATLAS data acquisition system is based on a sophisticated three-level trigger system, called the Trigger-DAQ (TDAQ) [1]. Triggers are rapid decision making mechanisms used to quickly filter out uninterested detector events in order to keep only interesting data for analysis. While the lowest level triggers retrieve data from the sub-detectors, high level triggers (HLT) are used to build and filter ATLAS-wide events for hard disk storage. After this trigger system, only a few hundred ATLAS events out of billions every second will remain for permanent storage [2]. These results are downloaded for offline analysis by scientists worldwide.

The silicon sensor-based Pixel Detector is the innermost sub-detector of ATLAS. This three-layer state-of-the art particle detector is responsible for the monumental discovery of the Higgs Boson on July 4th, 2012. The discovery provided much insight into the untested theories of the Standard Model in modern particle physics.

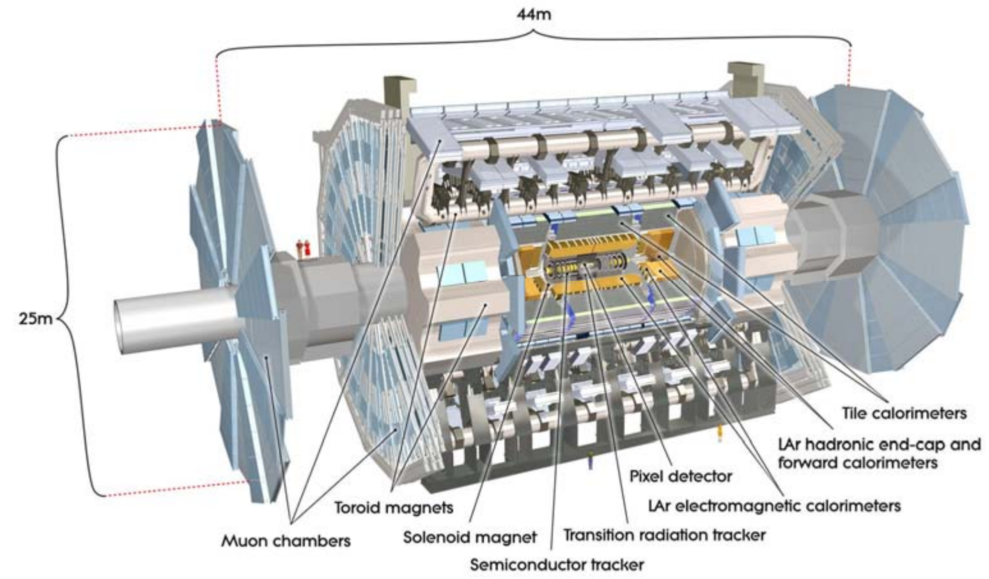

44m
25m
Tile calorimeters
LAr hadronic end-cap and forward calorimeters
Pixel detector
LAr electromagnetic calorimeters
Toroid magnets
Transition radiation tracker
Muon chambers
Solenoid magnet
Semiconductor tracker

Figure 1: Layout of the ATLAS Detector at the Large Hadron Collider, CERN, Geneva [3].

The LHC is currently experiencing a long shutdown to accommodate its Phase 0 (2013-2014) upgrade schedule. This is the first phase of the three-stage upgrade program, aimed to reach higher energy and luminosity to further efforts in characterizing the Higgs Boson and discovering new elementary particles. It is during this phase a fourth silicon layer called the Insertable-B Layer (IBL) was commissioned and successfully installed as the new innermost detector layer. This new detector layer uses a new generation of silicon sensors and front-end readout ASIC technologies. The motivation behind the IBL is to compensate for radiation damage to the existing layers as well as the readout inefficiency experienced by the other Pixel layers under increasing luminosity. The IBL offers a higher output data rate as well as radiation tolerance compared to other detector layers. In order to fully read out the IBL, a new off-detector readout system was designed for higher bandwidth and new DAQ electronics were commissioned. By adopting up-to-date processors and a novel calibration methodology, the readout electronics for IBL have become highly integrated and promise enhanced performance compared to their predecessors used for the Pixel DAQ.

The IBL Readout Driver (ROD), the main off-detector readout and control processor card, plays an essential role in the IBL DAQ system. The IBL ROD remains, in ways, similar to the Pixel ROD used for the reading out the existing Pixel Detector layers. It uses several powerful Field-Programmable Gate Array (FPGA) devices in order to perform real-time data acquisition and calibration for the front-end. The new ROD hardware cards were produced by INFN Bologna for the IBL collaboration. This thesis work focuses on particular aspects contributed by the University of Washington in close collaboration with the other IBL institutes responsible for firmware development.

Brief introductions to the IBL, the ATLAS Trigger DAQ system, and the silicon pixel detector are presented in chapter 2. This chapter discusses the general layout of the IBL, the DAQ components used for the IBL off-detector, and working theories of the hybrid pixel detector.

The firmware architecture of the IBL ROD, its front-end readout data processing, detector control, and detector calibration procedures are provided in chapter 3. This chapter discusses detailed VHDL ROD firmware theory and design, with strong emphasis on the data-taking path.

In order to verify ROD firmware functionality in both front-end readout and calibration, simulation and hardware tests were performed. In chapter 4 details and results of a simulation testbench implementing a realistic front-end HDL model, as well as ROD hardware trigger and readout tests have been provided. As the IBL ROD firmware/software development and field testing are ongoing, chapter 5 provides a summary of this thesis work as well as a list of near future firmware development tasks.

This thesis report has three appendices. Appendix A defines a list of currently (at the publication of this thesis work) implemented ROD slave FPGA control and monitoring registers. Appendix B summarizes all data formats involved in the ROD data processing. Appendix C collects acronyms used by the IBL collaboration.

Chapter 2

The IBL and DAQ System

The DAQ system for the IBL consists of the on-detector readout electronics, off-detector readout system, off-detector event computing, and communications subsystems between them. There are 14 IBL carbon fiber support structures, called staves, arranged in a barrel structure surrounding the beam pipe, each hosting 32 detector modules (silicon pixel sensors and front-end readout ASICs). The ATLAS experiment uses a sophisticated trigger DAQ system to filter data produced by its detectors and reconstruct physics events for use by particle physicists worldwide.

2.1. The Insertable-B Layer

The IBL sub-detector is installed between the existing Pixel Detector and the new ATLAS beam pipe. 16 dual-FE-I4 modules, to which 32 silicon pixel sensors are bump-bonded, are attached to a carbon fiber stave body called an "IBL stave." One stave implements twelve large planar sensors (each bonded to a dual FE-I4 module) and eight 3D silicon sensors in a symmetrical layout with the 3D sensors located on the extremities. The lightweight carbon fiber material fulfilled the low material budget for IBL as well as providing strong mechanical support for this new layer. At the center of each IBL stave body, a titanium pipe allows carbon dioxide-based cooling during detector operations to suppress thermal noise generation in front-end electronics. Very thin flexible PCBs (as shown in figure 2.1) are glued onto the stave body and sensors in which FE-I4 modules are wire-bonded in order to distribute power and communicate off-stave. All signal transmissions to and from the staves are converted from electrical to optical for minimal propagation delays.

A total of 14 staves are arranged in a barrel structure, as shown in figure 2.2, at a radius of 3.3 cm away from the center of the new beam pipe, to form the entire IBL. 448 FE-I4s are used to readout the IBL, providing over 12 million readout channels.

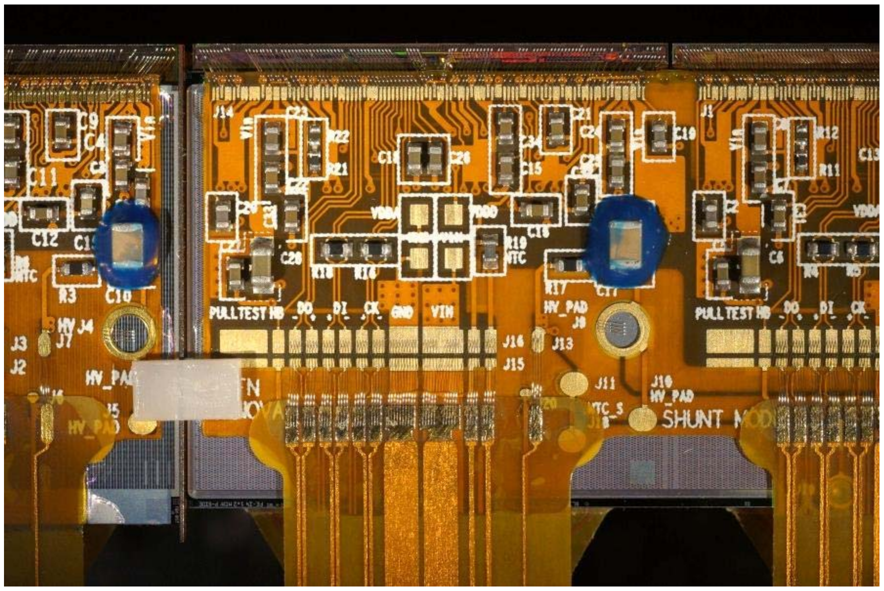


Figure 2.1: Flex PCB and wire-bonding connection to the FE-I4 modules on an IBL stave. The sensor chip on the left is a 3D sensor. The shorter dual sensor module on the right hosts planar sensors.

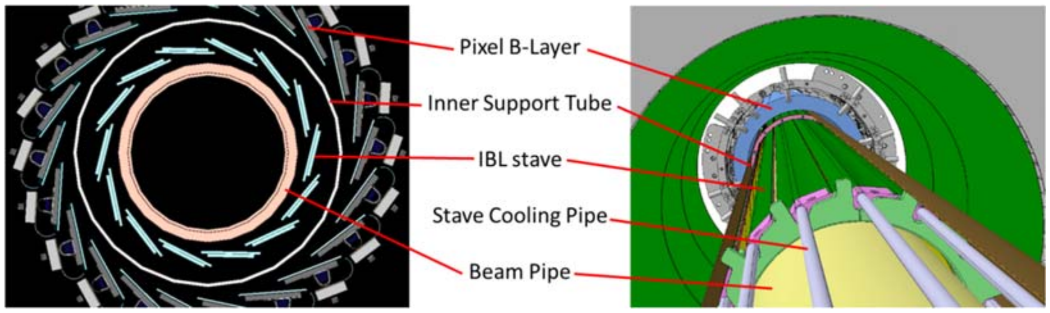


Figure 2.2: (left) Cross section view and (right) 3D view of the existing Pixel B-Layer and IBL [4]. The Inner Support Tube[1] is a 7m-long carbon fiber tube that was inserted into ATLAS detector prior to IBL installation in order to support IBL and alleviate constraints imposed by tight mechanical clearances between B-Layer and IBL.

2.2. The Trigger and DAQ System

The IBL is managed by the same TDAQ structure as Pixel for its data acquisition. Triggers are mechanisms used to quickly filter interesting events based on simple criteria when data throughput is large, especially at the LHC scale. They are essential for high rate processing and their design affects detector efficiency. The functional block diagram for TDAQ concerning the IBL is illustrated in figure 2.3. The VMEx64-based off-detector readout and trigger distribution forms the core of Level-1 TDAQ. Brief introduction of ATLAS Trigger System and core components of the Level-1 TDAQ are introduced in the following passages.

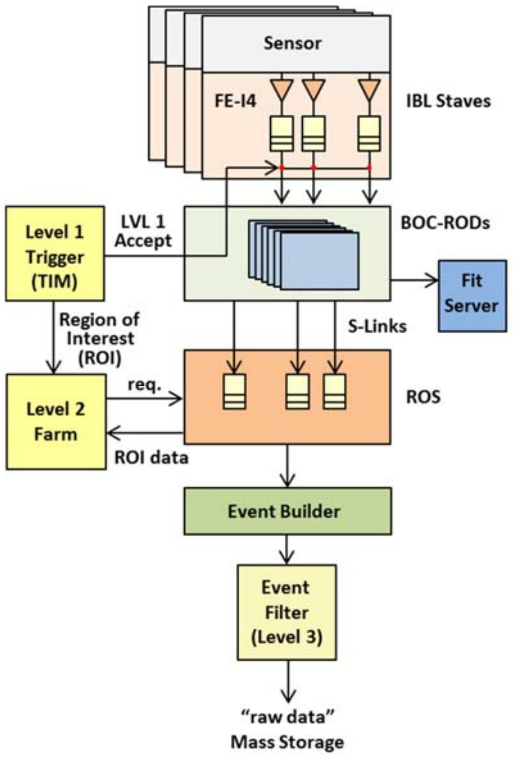


Figure 2.3: ATLAS Trigger DAQ system showing only the IBL section.

[1] The Inner Support Tube was designed and fabricated by the University of Washington, Seattle.

ATLAS Trigger System

Each trigger level is used to filter and keep desired data based on specified criteria. As the level of trigger increases, the computation of triggering criteria becomes more complex and trigger rates becomes orders of magnitude lower. Level-1 triggers are implemented in special front-end hardware and have the highest rate of firing. The High Level Trigger (Level-2 and 3) uses software triggers implemented in large computing clusters nearby the detector. In IBL, Level-1 triggers are sent to the FE-I4s to retrieve events at a rate of 100 KHz, and hit data is transmitted from the FE-I4s to the off-detector readout electronics. These triggers are distributed by the TTC Interface Module (TIM). After front-end events are gathered and processed, the data is sent to and stored in the Readout Subsystem's (ROS) readout buffers awaiting Level-2 triggers for further analysis. The desired data for analysis at this level is selected based on regions of interest (ROI), defined by Level-1 triggers, such as track paths or clustered hits. Level-2 triggers request ROI data for all ATLAS detectors to be retrieved from the ROS buffers and send data to a higher level event building serve farm. After ATLAS-wide events have been built, they are selected by Level-3 triggers for the final "raw data" storage.

TTC Interface Module (TIM)

TIM is the VME card that interfaces with the LHC-wide Timing Trigger and Control System (TTC) for the Level-1 TDAQ [5]. It distributes Level-1 triggers to the off-detector readout electronics. TIM distributes a global, 40 MHz clock to Back-Of-Crate (BOC) cards to synchronize both on-detector and off-detector readout electronics. For every trigger TIM produced, unique identifications are sent to the Readout Drivers (ROD) for data synchronization checks. These IDs include trigger-accept counter, proton crossing counter, and their reset counter values. Detailed definitions and ID-checking processes are described in section 3.2.5 under "Error Detector". TIM handles "busy" requests from RODs during data acquisition. In an event during which ROS is unavailable or off-detector readout data threatens to overflow, TIM halts triggering the detector to avoid potential loss of data. Section 3.2.8 is dedicated to describing ROD slave busy mechanisms that have been implemented.

Back-Of-Crate Card (BOC)

BOC is the VME opto-electrical conversion board that interfaces with one IBL stave. It includes three FPGA devices—one master controller and two slave data links. Each BOC slave receives serial data from 16 FE-I4s at 160 Mbit/s and is tasked with clock-data recovery and 8b/10b-decoding of these signals. A BOC sends decoded front-end data at 80 Mbit/s to ROD for front-end event-building and receives back processed data at the same rate. Data from every four FE-I4s are multiplexed and sent to the ROD on 12-bit buses. There are four S-Link[2] modules on the BOC, forwarding ROD data to ROS at a rate of 5.12 Gbit/s. High-level interactions between the detector, the BOC, and the ROD are shown in figure 2.4.

Readout Driver Card (ROD)

ROD is the VME data processor board responsible for front-end event-building and detector calibration. It includes four FPGA devices—one high-level master controller, one low-level board controller, and two slave datapaths. In data-taking mode, clock and trigger distribution is provided by the TIM, and the ROD data processing is event-driven [6]. In calibration mode, ROD is tasked with initializing detector modules, trigger generation, data gathering and histogramming FE-I4s. Each ROD slave exchange TX/RX data with one BOC slave. The new IBL calibration analysis only requires the ROD to produce simple histograms. Complete histograms are transferred to a server farm, called the FitServer, for fitting analysis. Detail architectures of IBL ROD and its datapath are discussed in chapter 3.

[2] S-Link: [7] CERN's custom data transmission protocol based on a FIFO-like user interface

Fitting Server Farm (FitServer)

The IBL FitServer is a high-performance, off-the-shelf computer. In Pixel ROD, occupancy histograms are generated by several slave DSP units and transferred to the VME host computer. In IBL, histograms are quickly transferred from ROD to FitServer via Gbit Ethernet. By moving time-consuming fitting analysis off-ROD and exploit commodity hardware, this setup not only overcomes VME bandwidth limitations but also achieves better performance compared to the existing Pixel ROD [6].

Readout Subsystem (ROS)

The ROS is the computing farm that buffers event data generated by RODs until Level-2 trigger reaches decisions for further readout. The Readout Buffer Input Devices (ROBIN) [8] are the primary ROD-ROS interface hardware data buffer for receiving front-end event data.

As there are 14 staves in the IBL, 14 new RODs and BOCs will form the complete IBL off-detector readout system. An additional ROD-BOC will be used for Diamond Beam Monitor (DBM). They will be hosted in one VME crate, along with one TIM card and a VME Single Board Computer[3] (SBC).

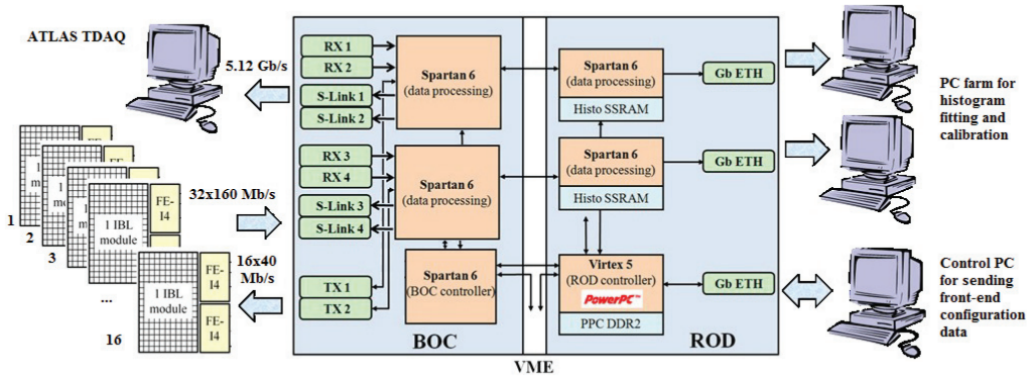

ATLAS TDAQ
5.12 Gb/s
32x160 Mb/s
16x40 Mb/s
1 IBL module
FE-I4
1
2
3
...
16
RX 1
RX 2
S-Link 1
S-Link 2
RX 3
RX 4
S-Link 3
S-Link 4
TX 1
TX 2
Spartan 6 (data processing)
Spartan 6 (data processing)
Spartan 6 (BOC controller)
BOC
Spartan 6 (data processing)
Histo SSRAM
Spartan 6 (data processing)
Histo SSRAM
Virtex 5 (ROD controller)
PowerPC
PPC DDR2
ROD
VME
Gb ETH
Gb ETH
Gb ETH
PC farm for histogram fitting and calibration
Control PC for sending front-end configuration data

Figure 2.4: IBL readout electronics in the ATLAS TDAQ environment [6].

Motivations for New Off-Detector Readout Electronics

The IBL is positioned closer to the interaction point relative to the existing Pixel layers. With an even higher luminosity for the LHC during 2015, the IBL aims to fully restore the function of the Pixel B-Layer and test the new sensor and IC developments. Occupancy, the number of hits detected during a given collision event, is expected to increase. The FE-I4s in IBL have addressed the expected issue of lowered detector efficiency (dead time and event pile-up) for Pixel with their increased granularity, larger active areas, more efficient readout architecture, and higher data bandwidth. However, a couple of reasons had motivated the development and commissioning of new off-detector electronics for the IBL readout [6]:

- **Readout FE-I4:** the new BOCs must handle four times the readout bandwidth than the old ones at 160 Mbit/s for each link, as well as handling the 8b/10 decoding.
- **Limited spares of Pixel readout electronics.**
- **VME bus bandwidth limitation:** higher speed is required for on-line monitoring as well as detector calibration procedures. It was decided that time consuming fitting analysis is moved off-ROD to commodity server machines using Gigabit Ethernet at a speed in excess of 1 Gbit/s per link. Compare to the Pixel ROD, which transfers the locally processed fitting data

[3] SBC: a commercial VME crate controller by Concurrent Technologies, used for calibration analysis by Pixel ROD. Not needed for the current IBL calibration.

to the VME crate controller at 7 MByte/s per link [9], the new IBL calibration architecture promises enhanced calibration speed.

- **Old and new technologies:** old data processing hardware components are obsolete and newer technologies have become available.

By keeping the same operating concepts of Pixel detector, the new readout system can integrate Pixel readout into more powerful devices and ease the short-term development efforts. The new electronics are required to handle the new front-end control and data processing, which remain similar to the Pixel detector. High speed Ethernet transmission (as an alternative to the VME) has been employed to enhance control and calibration of IBL.

2.3. Silicon Pixel sensor and FE-I4 readout ASIC

The front-end modules of IBL, like the ATLAS Pixel layers, consist of silicon-based particle detectors. Over the years, improvements in IC foundry processes and design techniques allow silicon sensors and readout ASICs to operate near highly irradiated interaction points (collision sites) and fulfill the need of higher detector performance. The two-layer approach of attaching a sensor to a readout chip is called "hybrid pixel detector" in which the sensor chip detects particles and the readout chip processes sensor signals. In both the Pixel and IBL detectors, readout chips are attached to sensor chips via bump-bonding technologies [4]. Each pixel unit on the sensor chip is bonded directly to a single readout channel that consists of analog and digital circuit blocks. One advantage of this hybrid approach is that a generic readout ASIC, in this case the FE-I4, can facilitate different types of sensors chips. Recent sensor technology candidates include diamond, gas, 3D, and planar in a variety of materials. The new generation of planar n-in-n and 3D n-in-p silicon pixel sensors with large active sensor areas were chosen for IBL.

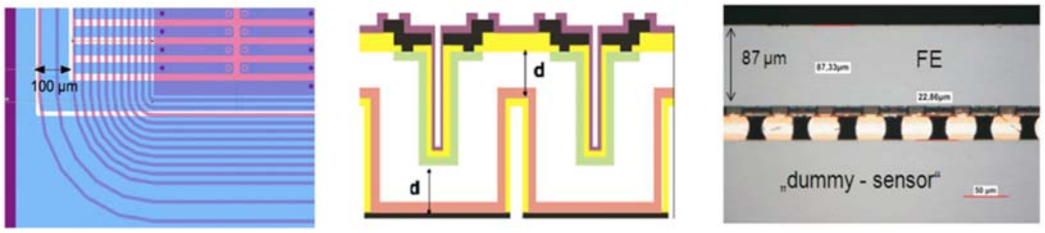


Figure 2.5: (left) slim-edge planar n-in-n silicon pixel sensor, (middle) cross section of 3D n-in-p silicon pixel sensor, (right) cross section view of a front-end readout ASIC bump-bonded to a "dummy sensor" [4].

During proton-proton collisions, ionizing particles travel through the pixel sensor layer and transfer energy to silicon atoms within the lattice. A sufficient amount of energy creates an electron-hole pair through material band gap, generating a small current signature for the penetrating particle. A silicon sensor utilizes high DC voltage (HV) to reverse-biased its p-n junction to create a depletion region in order to prevent immediate recombination of electron-hole pair and clear the sensor bulk of free charge carriers. During physics experiments, silicon sensors and readout chips are constantly under high energy radiation. Over time, displacements of silicon atoms create point defects would cause an increase in leakage current and reduces detectable current levels. Radiation damage is irreversible to silicon pixels, and detector modules generally have useful lifetimes of 10 years before rework becomes mandatory. Leakage currents in silicon pixels are sources of noise and power consumption. They increase rapidly with rising bulk temperature. To avoid high operating temperatures in the detector during experimentation, forced cooling mechanisms are built into the bodies of detector modules, as well as in the support structure.

When a penetrating particle is detected, the analog readout cell coupled to the pixel amplifies the generated current using a two-stage programmable current integrator (a.k.a. charge sensitive amplifier).

The first-stage preamplifier shapes the current pulse into a wider pulse shape, while the second stage provides additional voltage gain to the pulse [10]. The resulting signal is fed through a discriminator, compared to a programmable threshold, and becomes a digitized square pulse. The length during which this signal is digitally high is sampled with a counter and converted to Time-over-Threshold (TOT) that represents the number of clock periods the signal amplitude exceeds the threshold value. This digitized signal is stored in a local memory buffer waiting for further processing during triggered readout. A simplified FE-I4 analog pixel readout chain is illustrated in figure 2.6.

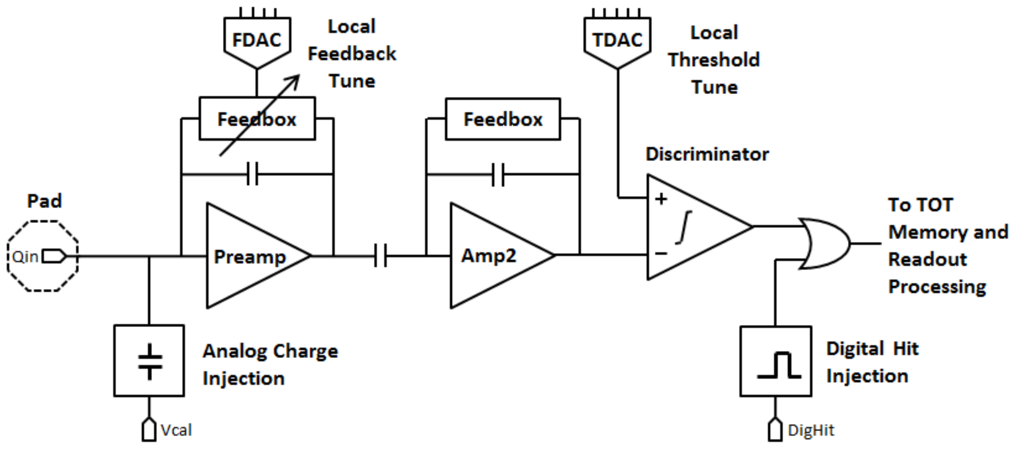


Figure 2.6: The FE-I4 analog readout cell structure with two-amplifier stage, simplified from [10].

Each analog readout cell allows fine tuning of the first stage preamplifier feedback value (for pulse shaping) and the discriminator threshold value through local DACs. Coarse tuning of these values is achieved via global tuning mechanisms. The cell also implements analog charge injection and digital hit injection tests; both are important for production tests and whole-chip calibration. Analog injection can be achieved through an internal pulser DAC (for selected pixels) or by injecting charges through an external bond pad. Calibrations of the detector are mandatory prior to any experiments due to variations in pixel characteristics across different process corners and accumulated radiation damage through aging. The FE-I4s are more robust against radiation damage due to much higher dopant densities in active devices with respect to silicon sensors [11], as well as their extensive use of redundancy digital logic design techniques.

The functional block diagram of the FE-I4 is illustrated in figure 2.7. This readout chip accommodates a pixel address space of 40 digital double columns (DDC) and 336 rows, totaling 26880 readout channels. When the FE-I4 receives a readout trigger, each DDC collects the hit TOT stored in its local digital cell and repacks the data. Neighboring hits in the same column are repacked into the same data word, called a "double-hit", under dynamic-φ pairing for better transmission efficiency. The end of digital double columns logic (EODCL) then passes a read token through the DDCs in ascending column order to readout the events. As the event is read out, the end of chip logic (EOCHL) further processes the event with frame markers and chip records. When ready-to-go data arrives at the data output block, it is 8b/10b encoded and serialized for high speed transmission. The 8b/10b encoding is an encoding scheme used to ensure DC balancing of signals sent through fiber optical channels. An on-chip integer-N phase-locked loop allows accurate locking of the input reference clock (40 MHz nominal) and can provide 1×, 2×, 4×, or 8× multiplied clocks to operate FE-I4. Clock and data ports of the FE-I4 use low-voltage differential signal (LVDS) protocol to communicate off chip to enhance noise performance.

A new ATLAS Pixel detector implementing next generation pixels has been foreseen for the LHC Phase-II upgrade beyond 2022. A 65-nm front-end readout chip is currently under planning and development (compared to the 130-nm FE-I4). HV CMOS with capacitive coupling transmission capabilities has been demonstrated. Additionally, monolithic pixel detector prototypes (MAPS) with substrate-integrated readout electronics have also been demonstrated as promising candidates.

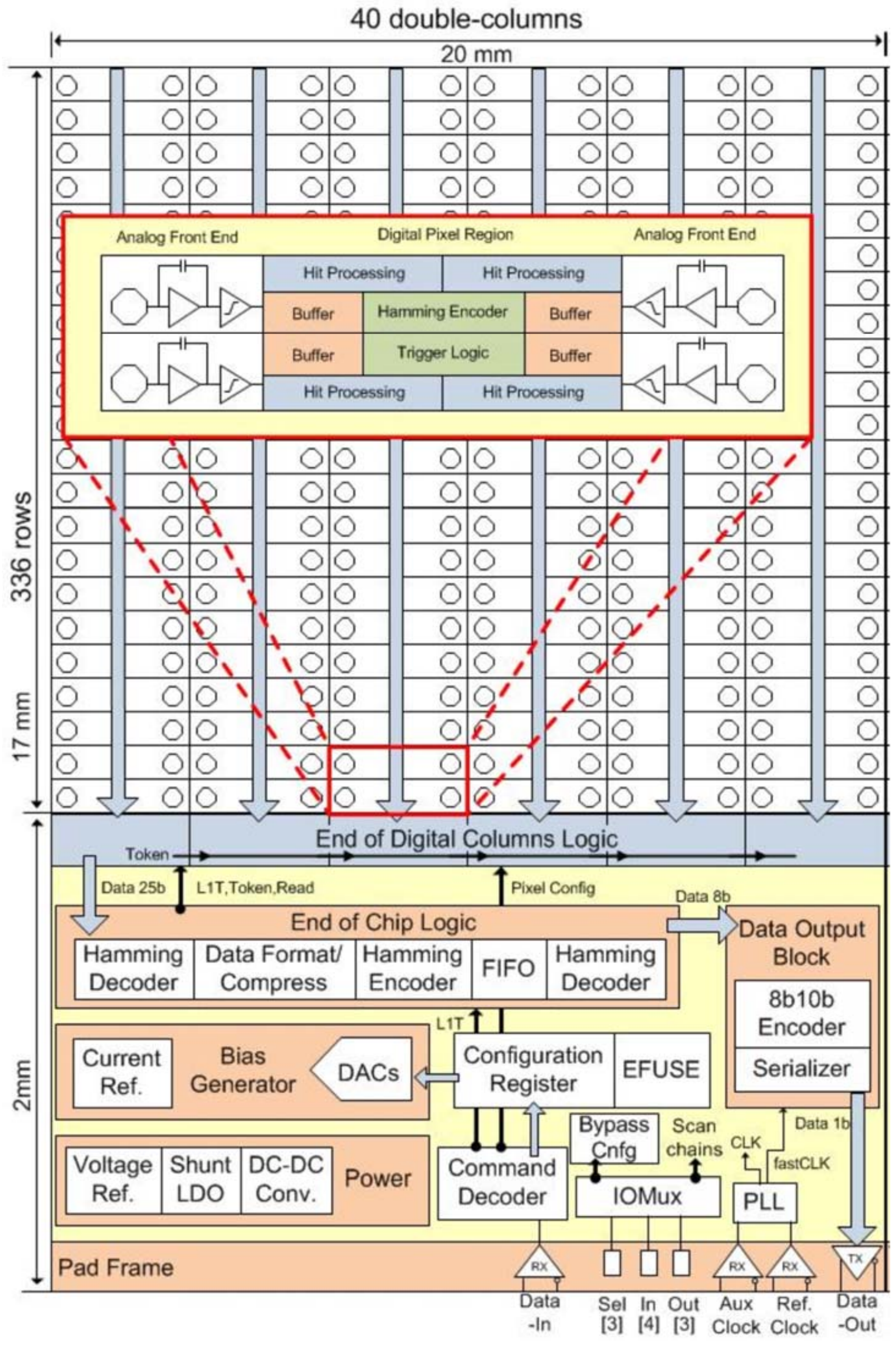


Figure 2.7: FE-I4 block diagram [10].

Chapter 3

ROD Architecture

The hardware and firmware implementations of IBL ROD are improvements on the Pixel ROD, with differences in front-end requirements (FE-I4 vs. FE-I3-MCC), level of integration, and calibration procedures. By utilizing more-to-date FPGAs, an IBL ROD processes data at higher rates with fewer components than a Pixel ROD. The IBL front-end readout structure remains similar to the Pixel readout. Simplified ROD control and datapaths are illustrated in figure 3.1. The control path interfaces to higher level systems, processes TTC triggers, configures the front-end, monitors the datapath, and executes calibration scans for the detector. The datapath processes front-end data for the ROS, or generates simple histograms for fitting analysis during calibration. An additional utility path is used to handle low-level coordination, reset, and programming of ROD.

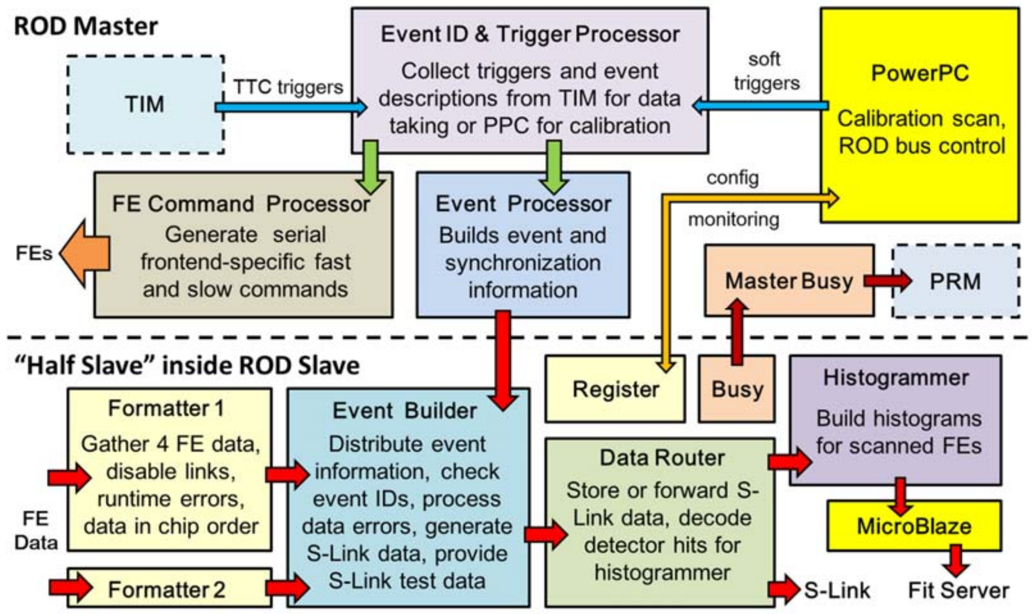

ROD Master
TIM
TTC triggers
Event ID & Trigger Processor
Collect triggers and event descriptions from TIM for data taking or PPC for calibration
soft triggers
PowerPC
Calibration scan, ROD bus control
FE Command Processor
Generate serial frontend-specific fast and slow commands
FEs
Event Processor
Builds event and synchronization information
config
monitoring
Master Busy
PRM
"Half Slave" inside ROD Slave
Register
Busy
Histogrammer
Build histograms for scanned FEs
Formatter 1
Gather 4 FE data, disable links, runtime errors, data in chip order
FE Data
Formatter 2
Event Builder
Distribute event information, check event IDs, process data errors, generate S-Link data, provide S-Link test data
Data Router
Store or forward S-Link data, decode detector hits for histogrammer
MicroBlaze
S-Link
Fit Server

Figure 3.1: ROD firmware control and data processing flows.

The control path on ROD master is split into real-time and non-real-time controls. All real-time functions, including trigger and event information processing, are run by firmware VHDL codes. During data-taking runs, triggers from TIM are only processed in real-time by the master firmware to provide datapath event information. Section 3.1 briefly describes major master firmware blocks currently implemented in IBL ROD. Non-real-time controls in the master are implemented with the FPGA's embedded microprocessor, called PowerPC (PPC). PPC configures FE-I4 through firmware with data received from an external or VME host PC. They can communicate using high-speed Gigabit Ethernet, or through the alternative VME bus interface. In addition to detector configuration, PPC provides software triggers, executes calibration scans, and drives the setup buses to access internal control/monitoring registers on ROD Slave and BOC card. ROD Master manages the two datapath FPGAs, communicates to the BOC card, and receives TTC commands from the TIM.

The firmware VHDL datapath implemented on the two ROD slaves processes data coming from up to 32 FE-I4s. Each slave receives four 12-bit buses from BOC. Each bus represents byte-level-multiplexed data from four FEs after the 8b/10b decoding (details in section 3.2.1). The details of the slave firmware design can be found in section 3.2. Data-taking and calibration in the slaves share the same data processing. For every triggered event processed, the slave assembles an

S-Link data packet for groups of eight FEs and sends it back to BOC for ROS-forwarding or deconstructed to extract hit information. New calibration methods require the ROD to build simple FE-I4 histograms. Each slave FPGA is equipped with two 5 MB SSRAMs and a DDR for fast histogramming updates and transferring. The embedded MicroBlaze processor on the slave FPGA is used to transfer completed histograms off to FitServer.

There are four FPGAs implementing the required ROD functionalities:

- **Xilinx Virtex-5 FX70T**: "Master" with embedded Power-PC processor that implements the control path and calibration.
- Two **Xilinx Spartan-6 LX150**: "Slave" with embedded MicroBlaze processor that implements datapath. -3 Speed grade FPGAs for fast histogramming.
- **Xilinx Spartan-6 LX45**: Program Reset Manager (PRM) handles low-level controls and VME interface with ROD Master.

While the ROD is designed to support both front-panel JTAG and VME firmware programming, program loading is achieved only through VME during full commissioning of the IBL. The IBL ROD PRM is implemented with most of its architecture inherited from the Pixel ROD PRM. Its main functionality includes rebooting the Master and Slave FPGAs during hard resets (or when the VME crate is powered up) and reinitializes data interfaces between ROD and its VME host. During configuration and reset, the PRM asserts a busy signal to TIM in order to halt issuance of triggers. In addition to the FPGAs, a Texas Instruments DSP chip is used on ROD as a backward compatible microprocessor used to run Pixel ROD control software. A revision-C IBL ROD is shown in figure 3.2.

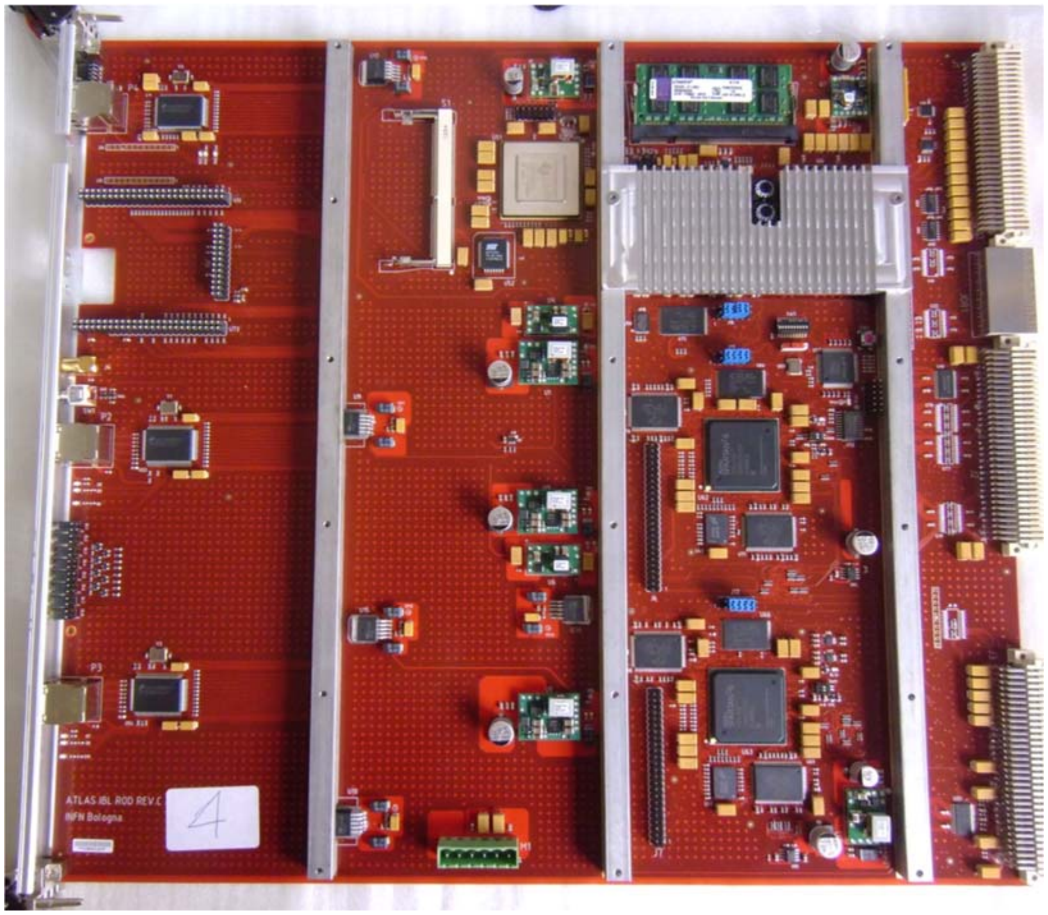


Figure 3.2: A revision-C IBL ROD VME card.

ROD Bus Interface

The ROD bus is an asynchronous signal bus that allows the ROD master PPC software to access slave internal registers and configure BOC master FPGA control registers. Most signal lines are shared between FPGAs as shown in figure 3.3, with the exception of the *chip enable* signals. Through the ROD bus, the PPC can monitor internal states of ROD slaves during operations and apply appropriate control mechanisms to the datapath. In addition, the PPC also configures histogrammer settings in the beginning of each calibration scan via the ROD bus. The master address decoder bus bridge VHDL block handles the interface between the PPC and the ROD bus by converting register address range to different target FPGAs. Inside ROD slave a register block handles all ROD bus interface signals, as detailed in section 3.2.2. All ROD master-slave bus signal definitions can be found in appendix B.

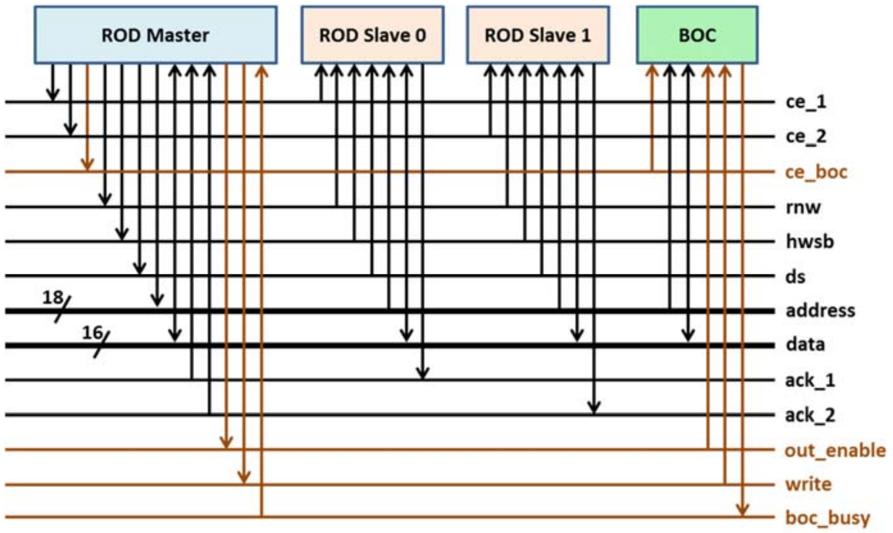


Figure 3.3: ROD bus interface signals flow diagram

3.1. Master Firmware

The IBL ROD master firmware inherits most of its features from Pixel ROD. The main functionality of ROD Master is divided into five blocks, shown in figure 3.4: Event ID and Trigger Processor, Front-end Command Processor, Event Processor, Internal Scan Processor, and the Diagnostics Generator. Modifications to the Pixel ROD firmware are done mainly in the Front-end Command Processor (FE-I4-specific), Event Processor, and register interface to ROD slaves. Very brief descriptions of main ROD master firmware blocks (except for the Internal Scan Processor and Diagnostic Generator), their interactions with the TTC and ROD slaves, and data formats with respect to IBL are provided in this section. Up-to-date materials presented in this section can be found in [12].

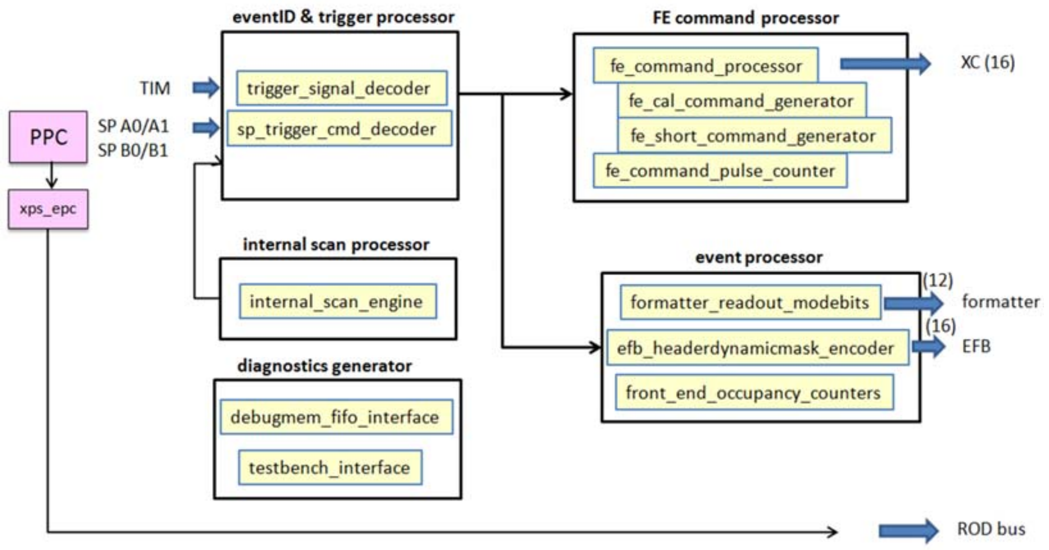


Figure 3.4: ROD master firmware block diagram [12].

3.1.1. Event ID and Trigger Processor

The Event ID and Trigger Processor collects triggers and event description data from different ROD trigger sources. Based on the type of trigger or command, this block generates corresponding pulses to the Front-end Command Processor, and forwards event description data to the Event Processor. Trigger sources for the ROD include PPC, TIM/TTC, the Internal Scan Processor, and the Diagnostics Generator. Currently, triggers from Internal Scan Processor and Diagnostics Generator are not available. The primary functions of this block are realized by VHDL-based Trigger Signal Decoder and Serial Port Trigger Command Decoder. The Trigger Signal Decoder performs shift-registered serial decoding for trigger types and event information from the TIM serial input lines. The Serial Port Trigger Command Decoder, originally connected to the Internal Scan Processor and Master DSP in Pixel ROD, is now receiving software calibration triggers from the PPC. For calibration, this decoder counts the number of Level-1 Accept (L1A) triggers issued by the PPC, as well as keeping free running bunch crossing value (BCID) counters. For every L1A trigger received, the decoded or calculated L1ID and BCID are sent to the Event Processor, along with the single-clock trigger/reset pulse sent to the Front-end Command Processor. Detailed definitions for these event IDs are provided in appendix B.

3.1.2. Front-end Command Processor

The Front-end Command Processor generates four types of commands to the detector that are requested by either the TIM or PPC:

- **L1A:** "11110", the Level-1 front-end detector trigger is the only short command, specific for FE-I4.
- **BCR:** "101100001", Bunch Counter Reset is a medium-length command that resets the BCID on FE-I4, same as the FE-I3.
- **ECR:** "101100010", Event Counter Reset is a medium-length command that resets the event counts on FE-I4, same as the FE-I3.
- **CAL:** "101100100", calibration command followed by a L1A packet, same as the FE-I3.

The processor glues the short and medium length commands together and applies front-end command masks, specified by the PPC through the register block, and generates proper front-end command streams. These commands are serialized and masked as the 8-bit "xc_a" and "xc_b" outputs at the Master top level. Each bit represents a command line for one double-chip FE-I4 module. They are routed to the BOC for front-end interfacing via ROD slave FPGAs. A pulse counter module is implemented in the processor to keep track of L1As issued.

3.1.3. Event Processor

The Even Processor builds event information for each trigger, and distributes the data to slave datapaths for event building. The Formatter Readout Mode Bits Encoder inside the processor sends the appropriate Mode Bit mask to the Formatters when the ROD detects a L1 trigger from any source. These Mode Bits are stored in a Look-Up-Table (LUT) with one default set and one corrective set. The default mode bits are sent to the Formatters for all "normal" (no correction required) triggers received from the TIM. If the ROD receives triggers via PPC serial ports, the default Mode Bits are sent when a trigger is detected on serial port A and the corrective Mode Bits are sent when a trigger is detected on serial port B. Mode Bits are currently not used in IBL event building. For detailed definitions of Formatter Readout Mode Bits, refer to [13].

The EFB Header ID and Dynamic Mask Encoder in the Event Processor is used to transmit the trigger information that is sent from the TTC system or generated internally on the ROD to the EFB prior to the arrival of event data from the FE modules. The EFB Header/Dynamic Mask Encoder is implemented in the Virtex-5 and consists of 5 FIFOs, 2 LUTs and an FSM to control the event header building task. The primary function is to collect all of the trigger description data, format and send the event ID header and the default or corrective Dynamic Mask information to the EFB. The format of generated data by this block can be found in appendix B.

If the TIM Trigger Signal Decoder is enabled, the L1ID, BCID and Atlas and TIM Trigger Types will be sent to the ROD on the TIM TTC Bus inputs and, when received, loaded to the Header/Dynamic Mask Output FIFO. An L1A from the TIM will load the ROD Trigger Type from the internal block RAM to the Header/Dynamic Mask Output FIFO. If the ROD is using the PPC Serial Port Outputs (SP_A & SP_B) for triggers, the Header ID information is generated using internal counters and registers in the Virtex-5 for each input.

When the ROD receives a "normal" L1A with no correction requirements, the default Dynamic Mask Registers are sent as the Dynamic Mask. If the ROD is using the Serial Port Inputs (SP_A & SP_B), the Default Dynamic Mask Bits are sent when a trigger is detected on SP_A and the Corrective Dynamic Mask Bits are sent when a trigger is detected on SP_B.

3.2. Slave Firmware

The main tasks of the ROD slave firmware are to provide robust front-end data readout processing, and fast histogramming for the IBL stave calibration. During datapath testing, BOC-emulated data or software can provide alternative front-end test data in the absence of real front-end electronics. The ROD slave firmware top level diagram is shown in figure 3.5. A “core” ROD datapath unit receives data from 8 FE-I4s and is organized into a “half slave” module, hosting two Formatters, one Event Fragment Builder (EFB), and one Router block. The two half slaves implements the slave FPGA datapath requirements, each forwarding data to its assigned histogrammer and S-Link output DDR blocks. S-Link output and histogrammer input data share the same datapath up until the Router output in the half-slave. One half-slave, one histogrammer, the embedded MicroBlaze processor, along with the external memories together makes up the ROD slave portion of front-end calibration loop.

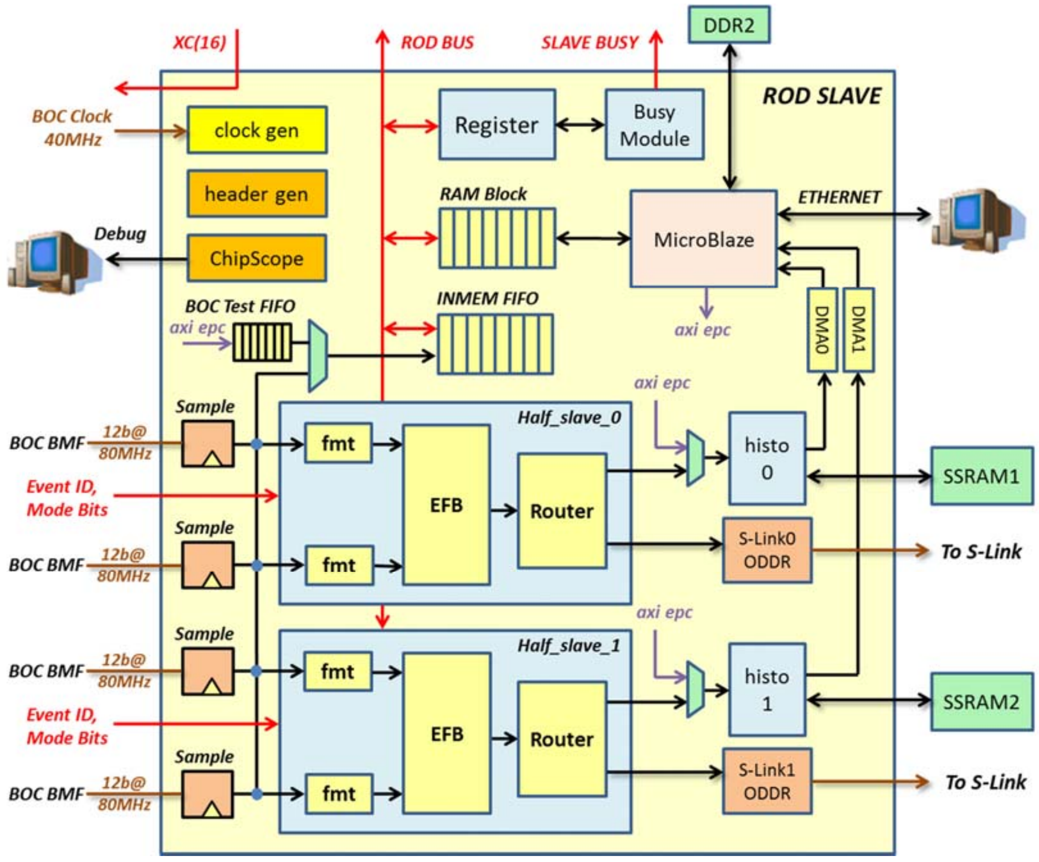


Figure 3.5: ROD slave firmware block diagram.

Since ROD slave interacts with external components using different clock speeds, the system clocks are generated and synchronized by utilizing an on-chip PLL as well as an external one. In order to de-randomize incoming data and operate across different clock domains, customized dual port RAMs as well as FIFOs with asymmetrical read/write features are heavily utilized in this firmware design. While the slaves communicate with the master and S-Links using a 40 MHz clock frequency, the majority of the data is processed at 80 MHz. The MicroBlaze and histograms operate on a 100 MHz clock. All system clocks can be monitored via buffered FPGA ports.

The data processing functionalities of Formatter, EFB, Router, and calibration are integrated into a single slave firmware, where in Pixel ROD they were implemented on separate FPGAs and DSP devices. The control and monitoring for current firmware blocks are now managed by a single slave register block that handles the ROD bus interfacing with the ROD master. Aside from a new calibration methodology, the current slave firmware handles FE-I4-specific data formats as well as service records. The new slave busy mechanism is used to halt TTC triggers from the TIM when buffer overflows or backpressure from the ROS S-Links occurs. The block also performs histogramming for different sources internal to the slave.

This firmware design takes advantage of the memory and slice resources provided on the Xilinx Spartan-6 LX150 FPGA. There are two direct memory access controllers inside the MicroBlaze, each used to speed up complete histograms from the external SSRAMs to DDR2-RAM. In addition, the firmware includes a ChipScope core – the Xilinx integrated logic analyzer suitable for spying die-level signals during hardware debugging.

3.2.1. Input sampling and data format

Each IBL ROD slave receives BOC BMF (BOC slave FPGA) data on a 48-bit bus. The choice to sample BOC data via double-data-rate input registers (IDDR) or phase-delayed multi-stage registering can be specified via a slave top level Boolean constant called *b2r_serdes*. When the latter is chosen, data is sampled at 80 MHz with a 240-degree phase off-set in the first three fabrics. This phase angle was found to be optimal for data RX during the ROD production tests. Aside from BOC input data, the slave also receives trigger-specific event information, link dynamic mask bits, and synchronization mode bits from the ROD master. The 48-bit BOC data bus can be divided into four 12-bit bus groups since front-end data are multiplexed in groups of four FE-I4s. The incoming data from ROD Master does not require specific input sampling as it comes from the same PCB with adequate setup and hold time margins. The slave final output data formats to the S-Links are also defined in appendix B.

BOC input Links

Each BOC BMF uses CDR and 8b/10b to decode and de-serialize differential data output lines from FE-I4s. The recovered 8-bit data frames are stored in FIFOs and read out by MUX controllers, along with chip identification, control word signal and a handshake valid signal. The four MUX units inside the BMF can handle data from up to 16 FE-I4s (from a half stave). The MUX outputs a data word from each FIFO (one per front-end link) in rotation to avoid data pile ups in any single FIFO. The order of data words, therefore, is not necessarily in ascending order of chip address. One 12-bit BOC MUX output bus maps directly to the inputs of one Formatter in the following format:

Bit [0:7] data: data fragment or a frame-alignment word.

Bit [8:9] address: indicates from which front-end chip the data originates.

Bit [10] write enable: data valid handshake, active low.

Bit [11] control word: indicates non-idle 8-bit data fragment or trailer word.

Every front-end event decoded by the BMF keeps the FE-I4 data protocol that utilizes unique control words in the 8b/10b system as special frame markers. Every event packet begins with the start-of-frame (SOF) word 0xfc at the beginning, followed by 24-bit data words (grouped in threes), and ends with the end-of-frame (EOF) word 0xbc. When there is no active data transmission by the FE-I4s, the 8-bit data portion of the bus transmits the DC-balanced idle state marker 0x3c. The Formatter gathers and latches complete data words (three 8-bit words) after a SOF is found, and writes a trailer after an EOF is detected. The FE-I4 output data format is provided in appendix B.

3.2.2. ROD bus communications in slave

The ROD bus can interact with either the slave register block or the RAM block. The slave register block is used to configure different run modes, provide information for data processing, and monitor slave internal events during ROD operation. This block provides the ROD bus interface for datapath firmware blocks inside half-slaves, as well as slave busy. The RAM block supports the ROD master PPC access with the slave MicroBlaze processor through the same bus, depending on the selected address range. The RAM Block implements a dual port block RAM in order to resolve communications in different clock domains between the ROD bus and MicroBlaze. Interactions between the ROD bus, the slave register block, the RAM block, and the MicroBlaze are shown in figure 3.6.

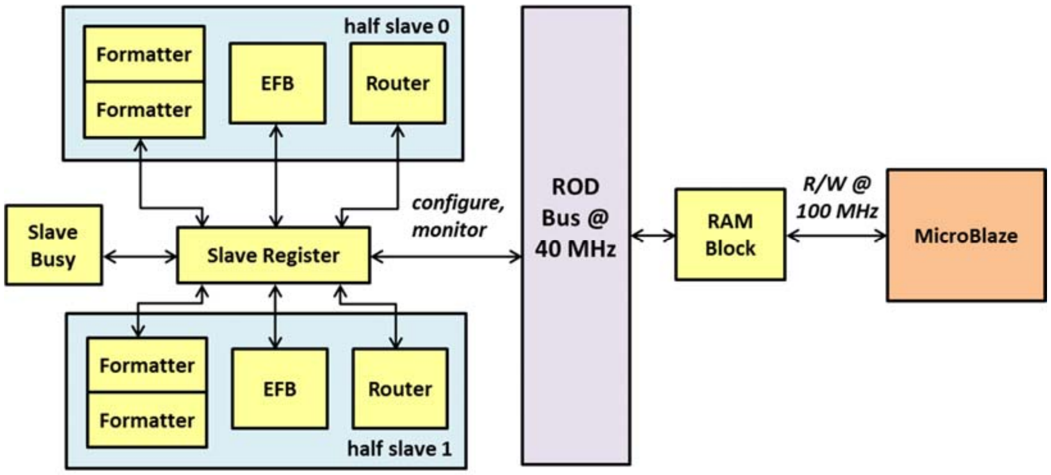


Figure 3.6: ROD bus interaction inside ROD slave firmware.

The slave register addresses are grouped mainly according to firmware block associations, as summarized in table 3.1. A list of all currently available slave register names and definitions is provided in appendix A. As firmware development continues in the near future, increasing Slave functionality would require further populating the current address space.

Register Group	Functionality
Global	Run mode configurations, link masks
INMEM FIFO	INMEM channel select, reset, readout
Formatter	Formatter configurations and monitoring
Link Error Masks	Set individual link's error mask
Event Fragment Builder	EFB configurations and monitoring
Slave Busy	Busy configurations and readout

Table 3.1: ROD slave register group and functionality.

All register addresses are defined in separate VDHL packages in order to ensure consistency across multiple firmware and software versions. At slave firmware top level there are two register packages used to generate two C++ header files: *rodSlaveRegPack.vhd* and *rodHistoPack.vhd*. These files are used by the PPC software to run calibration and provide firmware control/monitoring access to high level users. Since none of the register addresses were hard-coded, correct register interfacing is required between firmware and software.

As register tests are customary in system checks prior to operations, writing to and reading several link error masks of this firmware are recommended as they require accesses to internal memory blocks across different clock domains. When the Slave firmware is first initialized or reset, the register block automatically clears out the link error mask values. Prior to any ROD operation involving the datapath, the user needs to specify the slave ID for each ROD slave. Only the slave handling links 16 to 31 would need to have the ID set to the non-default value of "1."

3.2.3. INMEM FIFO

INMEM FIFO is a 1K word deep storage queue used to sample BOC input link data. The selection of BOC-to-ROD channel and readout of the data is done by setting the 2-bit channel selection register in the slave register block. To avoid populating the FIFO with idle data, a registered process is used to filter out the idle data when 0x3c is identified as the 8 LSBs on the 12-bit input bus. The incoming data for INMEM FIFO could originate from the FE-I4s, BOC-emulated data, or the BOC test FIFO in the slave firmware. The BOC test FIFO receives data from MicroBlaze and provide data to the INMEM FIFO based on a selection bit. The contents of INMEM FIFO are accessible to the PPC via slave register and ROD bus. Prior to its reading out, the INMEM FIFO should be emptied by pulsing the reset register inside the slave register block. The operational procedure of the INMEM FIFO is illustrated in figure 3.7.

Aside from sampling incoming data during ROD operations, INMEM FIFO can also be used for register read/write tests and data frame alignment checks. INMEM FIFO read/write using software injected data is a typical software test procedure during development. Incorrect data readout when sampling BOC data may signify erroneous register operation, incorrect sampling phase, BOC errors, or front-end errors.

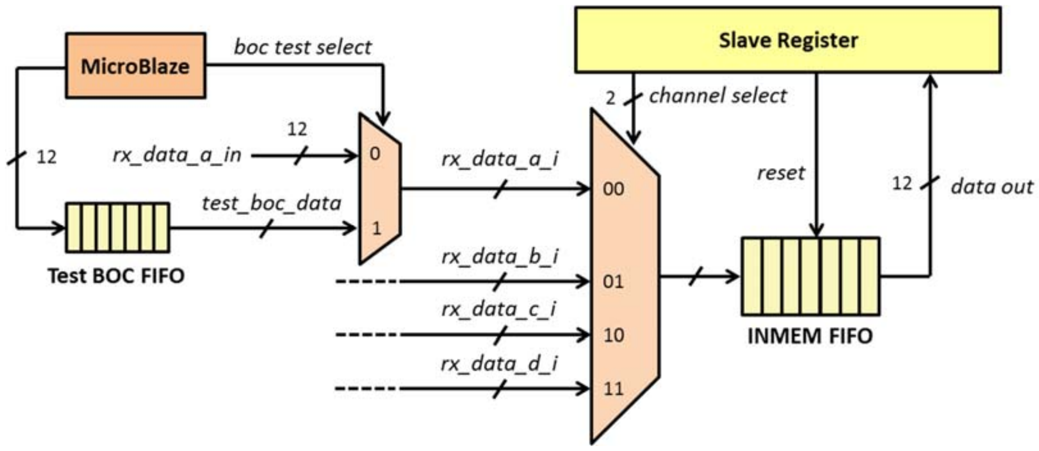


Figure 3.7: INMEM FIFO data select and readout diagram.

3.2.4. Formatter

The IBL ROD Formatter receives the multiplexed 12-bit BOC data and distributes decoded data into four storage FIFOs based on the corresponding link address. A few runtime errors such as timeout are checked and flagged in this block. During a complete event send off to the EFB, the Formatter marks the beginning of event (*boe_word*) and the end of event (*eoe_word*) for the EFB for processing. Due to inherent latency of event ID decoding in ROD Master after triggering, Formatter should hold data before the arrival of event ID data to the EFB. This is currently not implemented. Instead, shift registers with programmable length are used to delay triggers sent to the front-end. At the Formatter top level, front-end links can be disabled on an individual-link basis by setting the Formatter "link enable" register in the slave register block. The FIFO readout controller checks and flags any runtime errors for the enabled links in the data stream or data headers. Each Formatter is assigned to a four-bit portion of the 16-bit link enable signal that can be set by user via the Slave Register block. This signal bus allows the Formatter to cut off data stream coming from the undesirable front-end module(s). The top level block diagram of the IBL ROD Formatter is shown in figure 3.8. Formatter data format definitions can be found in appendix B.

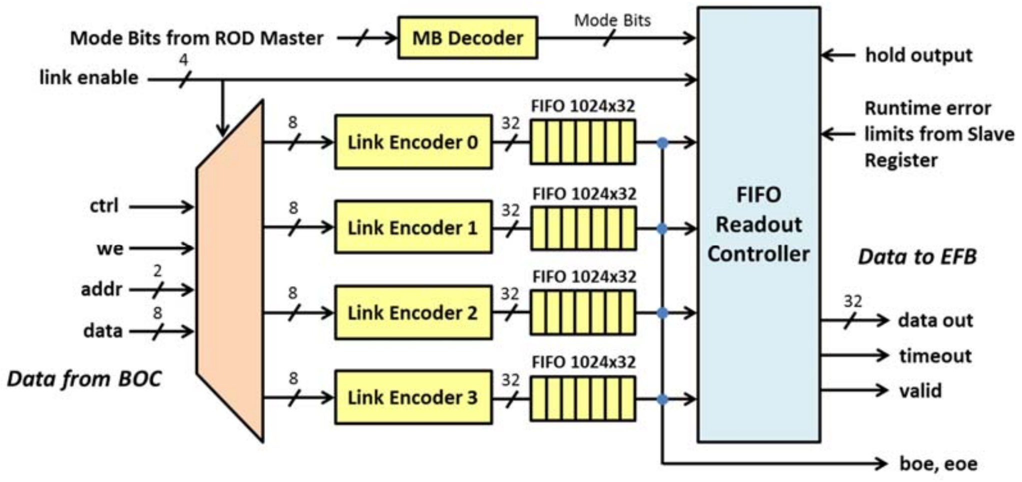


Figure 3.8: Formatter block diagram.

Link encoder

The link encoder begins latching incoming valid 8-bit words in group-of-three upon detection of a data header record. The 24-bit word is then rearranged into a 32-bit data format, required by the ROS, and pushed into the link FIFO. The link encoder ignores address records and value records from the front-end and accepts only the data headers, service records, and data records. A 32-bit trailer word including the link number would be written to the link FIFO upon detection of the FE-I4 EOF word. Skipped trigger counter, preamble error flag, and front-end error flag are reserved in the header word and are yet to be implemented.

For every 32-bit trailer written, a 12-bit combination of 3-bit LSB of L1ID and the BCID is inserted in the word. This feature is useful during debugging for checking whether the header and trailer written belong to the same event data packet.

Mode Bit decoder

The Mode Bits (MB) are not used in the current IBL ROD. The main code body of the decoder is inherited from Pixel ROD and act like a placeholder in IBL ROD. This module receives MBs from ROD Master and stores them in a dual-clock FIFO for use by the FIFO readout controller block. The main purpose of the MBs is to resynchronize the links with front-end events. Each link receives a 2-bit MB. "00" is the default setting of MB where no action is done to link events. Depending on MB value, the readout control could skip link readout or dump stored link data. Further usage information of the Mode Bits operations can be found in [13].

FIFO readout controller

The Formatter FIFO readout controller organizes the readout of the enabled link FIFO as well as handling a few runtime error scenarios. As the readout Mode Bits are not currently used for event synchronization, the Mode Bit readout value in for the controller is set to a default value for normal readout operations at this stage. The FIFO readout controller implements the FSM shown in figure 3.9. The readout order for the Formatter links and events when all links have been enabled should be in the ascending order of link numbers and events.

The FIFO readout controller implements a timeout counter and a data overflow counter. When the Formatter is set to read a new link FIFO, a timeout counter will begin counting clock periods. If the link FIFO remains empty when the count reaches the user defined limit (FIFO timeout limit), the timeout header and trailer will be generated for the link before the controller moves on to read the next link. The FE-I4 outputs a data header with event IDs and sometimes service records upon every trigger. A data output register stage counts the data elements of a given event. If the event exceeds the user defined limit (data overflow limit), the data overflow trailer will be generated. The data formats of timeout and data overflow is defined in appendix B.

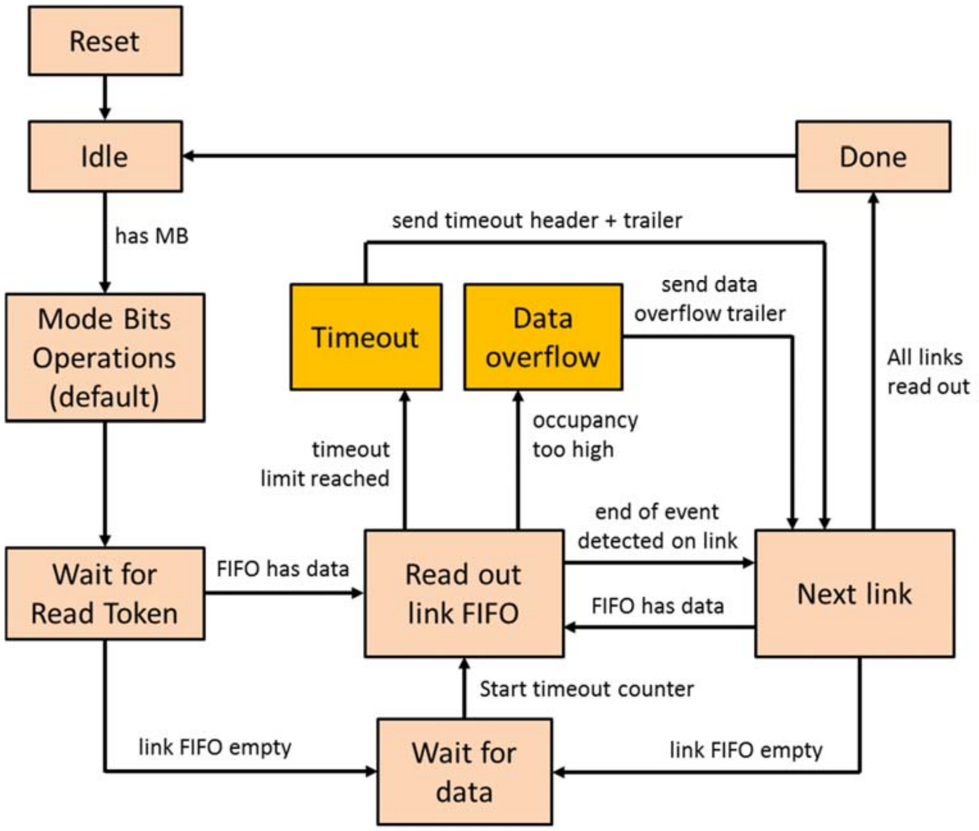


Figure 3.9: Formatter FIFO readout controller FSM.

Runtime errors checked in Formatter

Most of the runtime error check mechanisms in Pixel ROD Formatter are not implemented in the current IBL ROD Formatter, except for the timeout error flag. Typically, each runtime error is measured by a dedicated counter with programmable threshold values via the slave register. Each error can be flagged in the data trailer or data packet overhead. Runtime error checks applicable to the IBL ROD Formatter are:

Timeout error: after a specific trigger is sent for an enabled link and the Formatter does not receive any data, the Formatter will generate timeout-specific header and trailer as well as flagging timeout error bit in the data overhead to the EFB. The limit of timeout is programmable.

Header error: when the 8-bit MSB of header is received and is not a hexadecimal "E9", header error is flagged in either the header or trailer. This could signify sampling errors.

Trailer error: when the hexadecimal trailer "BC" is never received by the link encoder, trailer error is flagged in a generated trailer after a certain amount of time. This could signify sampling errors.

ROD Busy limit reached: when any given data storage FIFO occupancy exceeds a programmed number, the Formatter will assert a "busy" signal to ROD Master to halt triggering in order to avoid fatal data loss. This runtime flag does not appear in the data stream.

Synchronization error: this means a misalignment or errors in frames. This error may be detected by checking for invalid frame sequence. Currently this runtime error is checked inside IBL BOC slave FPGAs but is not processed by ROD.

Skipped events: when the 16-word deep buffer of pending triggers inside FE-I4 is full, any additional trigger (due to a new L1A command or to multiple triggers from a multiplier greater than 1) will result in skipped triggers. These triggers are ignored, but the FE-I4B chip keeps track of how many have been ignored with the 10-bit skipped trigger counter. This counter is read out with Service Record 15, which comes out automatically if there are skipped triggers (unless explicitly disabled). The formatter extracts and saves the number of lost events from the appropriate header and inserts the required number of empty events into the data flow. The inserted empty events will contain the correct L1ID and sequential BCID and will contain a flag in the header that will indicate the number of events that were skipped for the current L1 trigger. This process performs the required function of keeping events in synchronization and informs the offline system that an error has occurred on a module.

3.2.5. Event Fragment Builder

The EFB handles the majority of the data checking and event fragment generation in IBL ROD Slave. It receives event description data from ROD Master and data coming from two Formatters. The EFB has an error detect block that marks errors from event-trigger mismatch, data corruption, and FE-I4 service records. The fragment generator generates S-Link format front-end event data once all the data of an event had been received and processed by the EFB. Many FIFOs were used for de-randomizing incoming event data as well as to provide temporary data storage in case the readout chain comes to a halt. Some smaller FIFOs or RAMs, up to depths of 1K, were implemented using distributed elements inside the FPGA in order to conserve Spartan-6 block memory resources for higher priority functions. Memory elements with asymmetric read/write widths can only be implemented via block memory due to FPGA design constraints. The EFB top level block diagram is shown in figure 3.10. Format definitions for EFB-relevant data can be found in Appendix B.

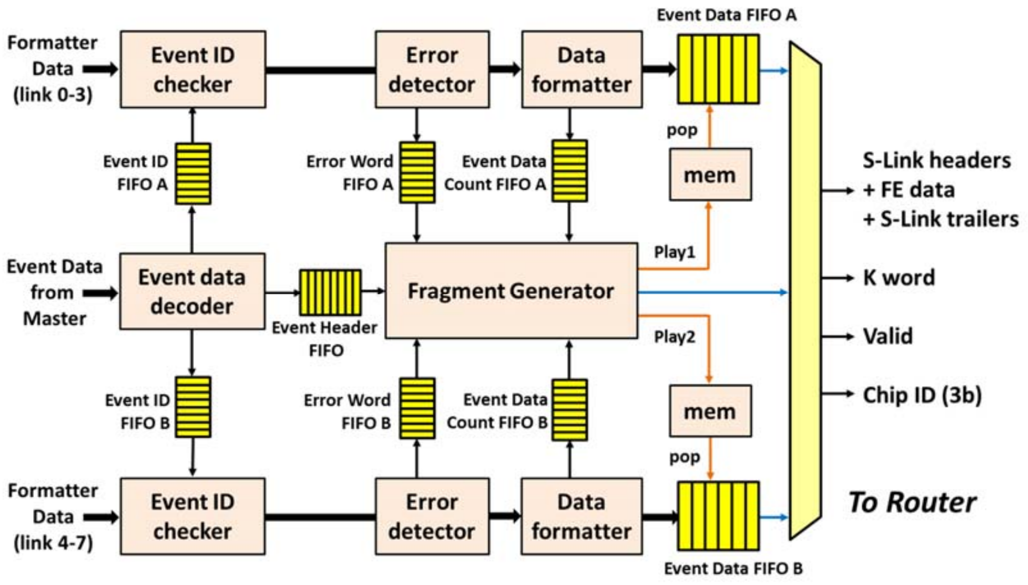


Figure 3.10: Event Fragment Builder firmware block diagram.

The majority of digital logic and memory blocks inside the EFB operate on the 80 MHz system clock. FIFO buffers receiving data from ROD master and the error mask LUTs used by the error detectors are implemented with dual-clock memory elements that can operate between 40 and 80 MHz clock domains.

At the final output of the EFB to the router, multiplexers are used to select outputs between fragment generator blocks, event data FIFO A, and event data FIFO B. An additional 3-bit chip ID and 1 bit control word are sent to the Router. The chip ID is retrieved from the data output and sent to the Router block to support hits-extraction to the histogrammer.

Event ID definitions

The event IDs generated by the TTC and detector front-end electronics are used to cross-check synchronization between a trigger request and the returned event. Event IDs inside FE-I4 are used to correctly read out requested hit events.

- **Bunch crossing ID:** the LHC circulates the accelerated proton beams in opposing directions and crosses them in the ATLAS detector every 25 ns. The TTC keeps track of this bunch crossing value. The FE increments its internal bunch crossing counter at 40 MHz and assigns the counter value to a recorded event. This 10-bit counter can be reset through a BCR command.

- **L1ID:** the 23-bit Level-1 ID counter inside the FE is incremented every time it receives a L1A trigger request. The FE only provides the 5 LSBs of this counter value in the output data header. If any of the higher MSBs are incremented a service record will be generated with the MSB count value. The L1 counter inside the FE can be reset through an ECR command.
- **BCRID:** the bunch crossing reset counter increments every time a BCR command is received by the FE, which also resets the bunch crossing counter.
- **ECRID:** the event reset counter value is incremented every time the FE receives a ECR command, which also resets the L1 counter. The ECR may be issued to resynchronize the trigger and readout event.

Event data decoder

The event data decoder receives eight 16-bit event ID data words from the ROD master and assigns corresponding event ID and link dynamic masked bits to the event ID checkers. It allows BCID offset calculation with a rollover option and generates 64-bit event header words for the fragment generator block to generate S-Link headers at the output of the EFB. The high level block diagram for the event data decoder is illustrated in figure 3.11. The received event ID words are first stored in a 1K-depth FIFO that provides a word count. When the FIFO holds eight or more words, a FSM pops the FIFO to shift the output through the register stages. When a complete event ID packet of eight words has shifted out, the words are then latched for event ID and dynamic link mask retrieval. Word 0 to Word 3 are concatenated into a single 64-bit word and latched into the event header word FIFO at the EFB top level. A 2-bit mask selection that comes from the EFB engine ID and Slave ID is used to determine which dynamic mask bits would be assigned to the link group. The decoder generates two decoded 27-bit words and stores the words in the event ID FIFOs for further processing by the event ID checkers. The input and output formats of the decoder are provided in appendix B.

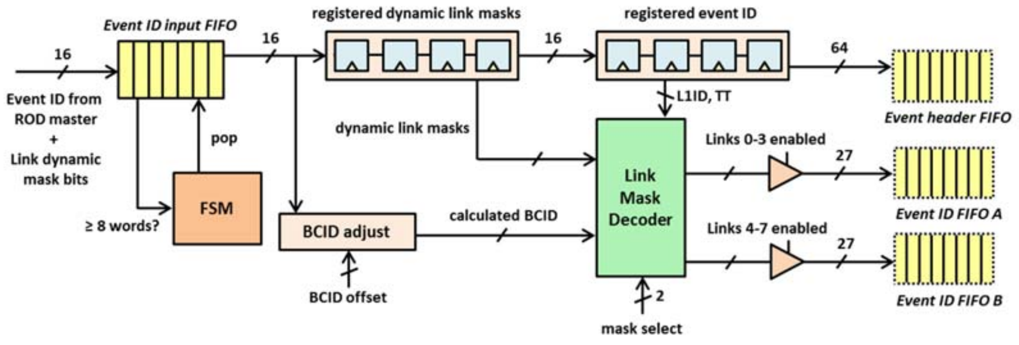


Figure 3.11: Event data decoder firmware block diagram.

Event ID checker

This block receives data from the Formatter, decodes the link number, calculates the L1ID offset, resynchronizes the four FE data links (individually) with dynamic masked bits, checks errors (BCID, L1ID, non-sequential FE order), provides an event group counter, flags the end of an event, and allows single event ID trapping. The block diagram for the checker is illustrated in figure 3.12. A gatherer ID (to distinguish the paired Formatter) and the EFB Engine ID is assigned to this block to determine the dynamic link mask assignment to the data. The Formatter beginning-of-event word input is used to pop the event ID FIFO and retrieve ID and link masks. If the first word arrives at the block and the check is unable to retrieve any data from the event ID FIFO, an event syncrhonization error flag will go high, and reset of the ROD is required. The checker uses the dynamic link masks to calculate the L1ID offset to each link and either add or substract one from the L1ID extracted from the event ID data. The newly computed "expected" L1ID and extracted BCID values are checked against the IDs found in the header words of incoming Formatter data and marked for any errors in the output data overhead. An additional non-sequential chip error flag in the overhead checks for whether the front-end links are coming in non-descending chip order. These overhead bits are defined in appendix B.

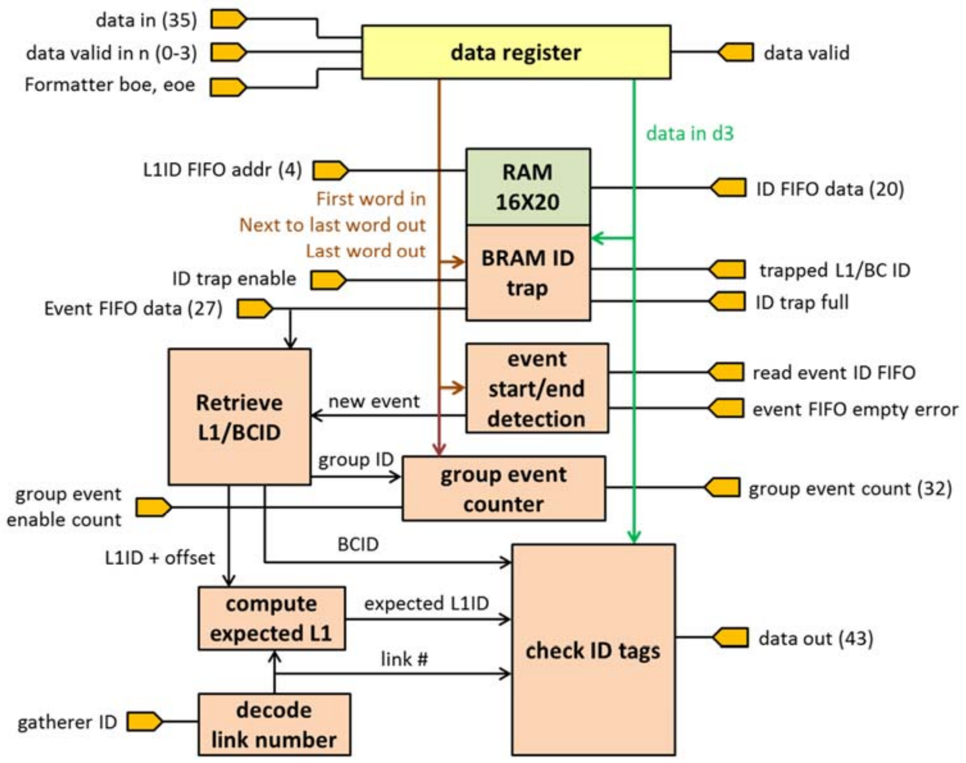


Figure 3.12: EFB event ID checker block diagram.

The ID trap in this block allows the user to take a snapshot of the event IDs in the data stream. When the trap is enabled throught the slave register, a set BCID and L1ID from the first incoming data header is latched into a small distributed RAM for Register readout. This is a one-shot setup, so the trap must be enabled again via the slave register if another set of IDs are to be trapped. A set of group event counters are provided in this block to keep track of the number of events arrived and processed on each link. The 4-bit counter of each link is incremented every time the corresponding bit to the link in end-of-event word is pulsed. The event counter values for all eight links can be read out under a single Slave Register address.

Error detector

The error detector is responsible for recording all the marked errors in the previous datapath, check for errors reported by front-end service records, and generate error flag words as well as the error count that occurred during an event for event fragment generation. The block diagram for the EFB error detector is shown in figure 3.13, and the pseudo code for the error processing is given in figure 3.14. It consists of two distributed-RAM blocks, a register data block, a decode error block, and a process error block. In order to account for the runtime errors found previously in the Formatter, now embedded in the data trailer word and data overhead, the *trailer error count* was used alongside the *count error* signal. While the trailer error count determines and reports the number of flagged errors in the trailer to the *process errors* block, *count error* gets flagged when other types of errors are detected. There are two main classes of errors – generic and front-end-specific errors. Generic errors are operational errors such as event-trigger mismatch and runtime errors while front-end specific errors are decoded from service records. Each error detector has a dual-clock dual-port RAM storing a 32-bit error mask word for each of the four links. The error mask words can be written in or read out via slave register. Upon every system reset, the register clears out the error mask words in the RAM so no errors are initially masked. The definitions of the error word and error type can be found in appendix B.

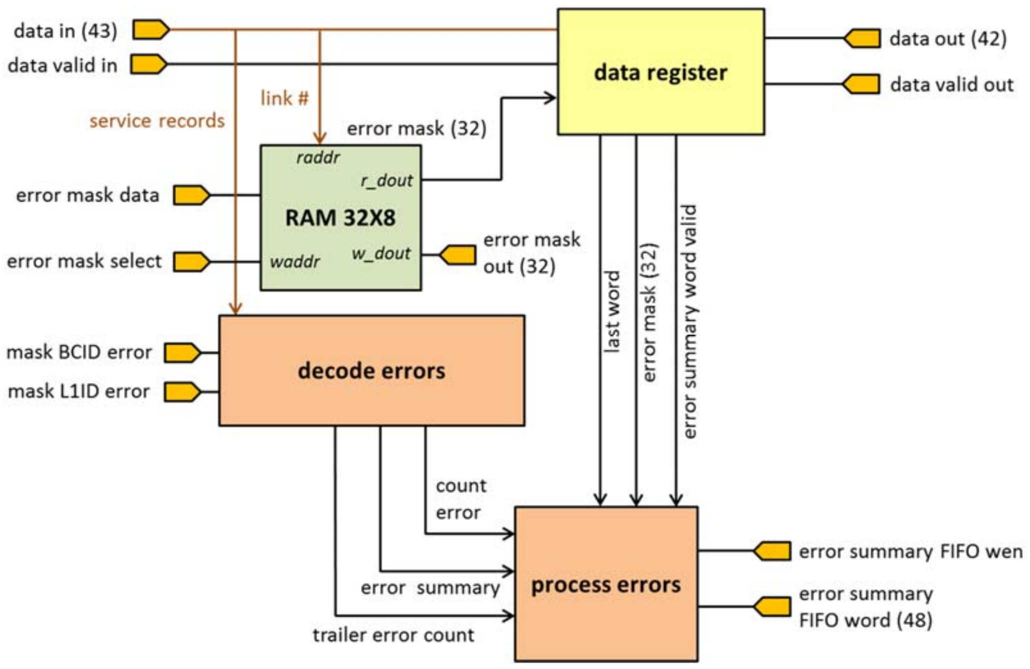


Figure 3.13: EFB error detector block diagram.

```
Module error detector (
   I/O
Begin RTL
Dual port RAM{
   w/r error mask words (32 links)
}
Register Data{}
Decode Errors{
   -- checks flagged errors in data header
   if (error found)
         flag error summary word (i) and set count0
   -- check trailer flagged errors
   if (error present)
         flag error summary word(i) and calculate trailer error count
   -- check SR code data words
   if (SR counter != 0)
         flag error summary word(i) and set count0
}
Process Errors{
   summary flag (i) = summary word (i) AND NOT error mask(i)
   error counts = error counts + count0 + trailer error count
}
Error FIFO data = summary flag & '0' & error count
Writes data to error word FIFO at end of event "last word"
)
```

Figure 3.14: EFB error checking pseudo code

The new FE error flags for the IBL ROD are determined by checking service records in the data stream against FE-I4 service record codes and the payload counter values. Errors that are of similar category or require the same actions are grouped together. For example, there triple redundant mismatch errors are grouped as one error; all service records that suggest a L1 counter reset in the FE-I4 are grouped as another single error. Since the same error summary format is desired and data width is limited, several errors are combined into one. However, the error count for each of the combined error element is counted individually; that is, if errors in the same group are detected, error counter will increment by number of those error, while these errors are represented by a single bit in the error summary word.

It is important to note that in the final error flag format (see table B.7 in appendix B), Bit 3 and Bit 4 have different meanings and formatting compared to the generic field errors. Bit 3 compares all the data flags in Bit 14-31 and becomes set only if there are non-masked and detected errors found. It is a single summary flag bit that indicates any of the multiple errors occurred in the data words themselves. Bit 4 compares the error flags in Bit 14 and 15 and gives a summary flag that indicates whether any internal buffers had overflown. The error checker increments a counter every time a non-masked error is found. At the end of each event, the error summary flags are concatenated to the error counter value and written to the error summary FIFO at the EFB top level. Whenever an error summary FIFO data word becomes available, it is written into a FIFO that will eventually be read out by the event fragment generator and formatted as two separate output error trailers to the S-Link. This error summary word is made up of error summary flags and error counts.

In order to verify the error handling of the FE-I4 service records, informational service records with frequent occurrences can be temporarily "tagged" as error records in the VHDL code during testbench simulation. The "error" would then become observable in the S-Link trailer at the EFB output.

Data formatter

The EFB data formatter functions as a simple data word counter for event data that passes through. This block receives the 42-bit output from the error detector, and writes the lower 32 bits of data to the connecting event data output FIFO for readout by the fragment generator. For every data word that has been pushed into the FIFO, a counter increments the current word count until the end of event marker in the input data overhead (bit 41) becomes high. This marker is used as a signal to latch and write the word count to the data count FIFO and reset the counter. The maximum allowable word count for any event in the current implementation is 4K words, which is the depth of the event data output FIFO. The block, as shown in figure 3.15, is currently using one extra register stage which could be omitted.

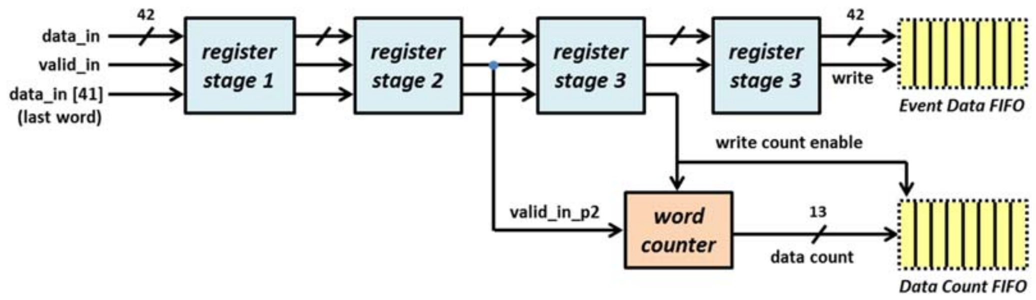


Figure 3.15: EFB data formatter block diagram.

Output memory controller

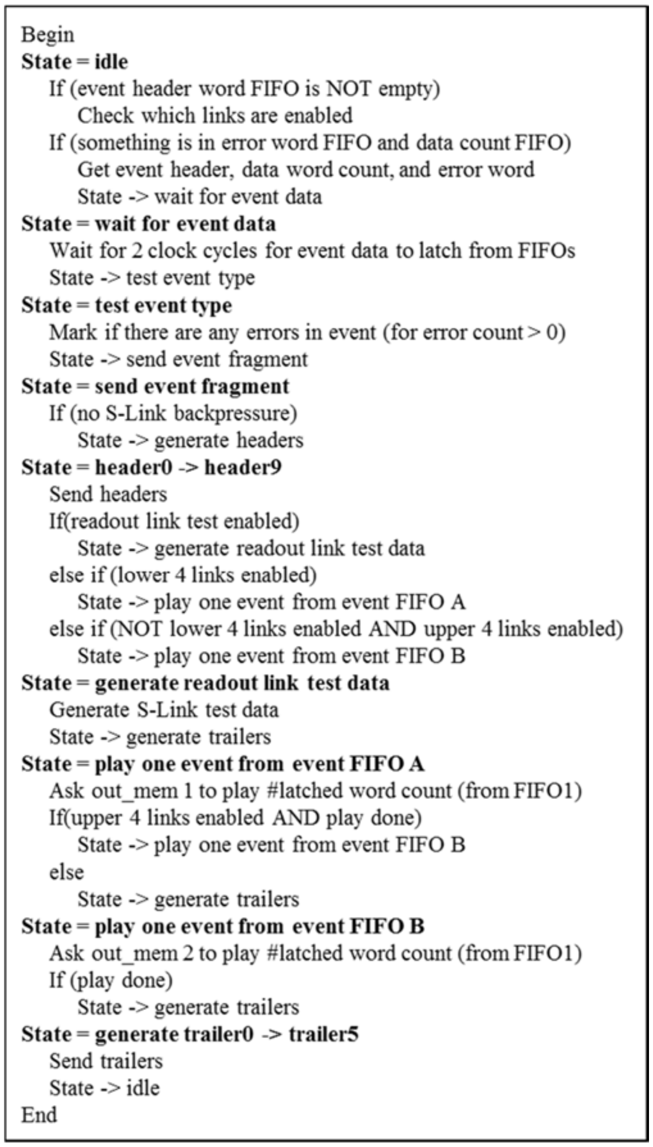
This is a FIFO readout controller with delicate timing mechanisms. It receives the number of words to play from the fragment generator during S-Link data generation, and pops the event data FIFO the same number of times. When the complete event is played, the block signals the fragment generator to either move on to play the other event data FIFO or to generate the trailer words. Each output memory controller handles stored events from four Formatter front-end links. If all four links are disabled, this block will never receive any play commands from the fragment generator.

Fragment generator

The EFB fragment generator takes in the ATLAS event information (trigger type, event type, run number, detector module ID), as well as the collected data words from the front-end chips, and organizes them into a 33 bit S-Link data packet (control “K” word + 32-bit data word). This block coordinates and interfaces with the FIFOs holding processed event data, event ID, event word count, error summary, and the output memory controllers to piece together S-Link data in the following sequence:

- **headers** (beginning-of-frame word and ATLAS event information)
- **data from Event Data FIFO A** (front-end data provided by Formatter 0)
- **data from Event Data FIFO B** (front-end data provided by Formatter 1)
- **trailers** (error summary, size of event, end-of-frame word)

The fragment generator implements a FSM to generate data. The pseudo-code of this FSM is shown in figure 3.16. The FSM will halt output data when S-Link asserts backpressure and resumes data generation when backpressure is alleviated. The fragment generator also provides readout link test data. The complete definitions for output data format in this block are found in appendix B.



```
Begin
State = idle
    If (event header word FIFO is NOT empty)
        Check which links are enabled
    If (something is in error word FIFO and data count FIFO)
        Get event header, data word count, and error word
        State -> wait for event data
State = wait for event data
    Wait for 2 clock cycles for event data to latch from FIFOs
    State -> test event type
State = test event type
    Mark if there are any errors in event (for error count > 0)
    State -> send event fragment
State = send event fragment
    If (no S-Link backpressure)
        State -> generate headers
State = header0 -> header9
    Send headers
    If(readout link test enabled)
        State -> generate readout link test data
    else if (lower 4 links enabled)
        State -> play one event from event FIFO A
    else if (NOT lower 4 links enabled AND upper 4 links enabled)
        State -> play one event from event FIFO B
State = generate readout link test data
    Generate S-Link test data
    State -> generate trailers
State = play one event from event FIFO A
    Ask out_mem 1 to play #latched word count (from FIFO1)
    If(upper 4 links enabled AND play done)
        State -> play one event from event FIFO B
    else
        State -> generate trailers
State = play one event from event FIFO B
    Ask out_mem 2 to play #latched word count (from FIFO1)
    If (play done)
        State -> generate trailers
State = generate trailer0 -> trailer5
    Send trailers
    State -> idle
End
```

Figure 3.16: FSM pseudo code for EFB event fragment generator

3.2.6. Router

Once events have been processed by the Formatters and EFB, the Router either forwards the generated contents to ROS through S-Link or extracts hits for the histogrammer. The block diagram for the Router is illustrated in figure 3.17 (S-Link interface signals excluded).

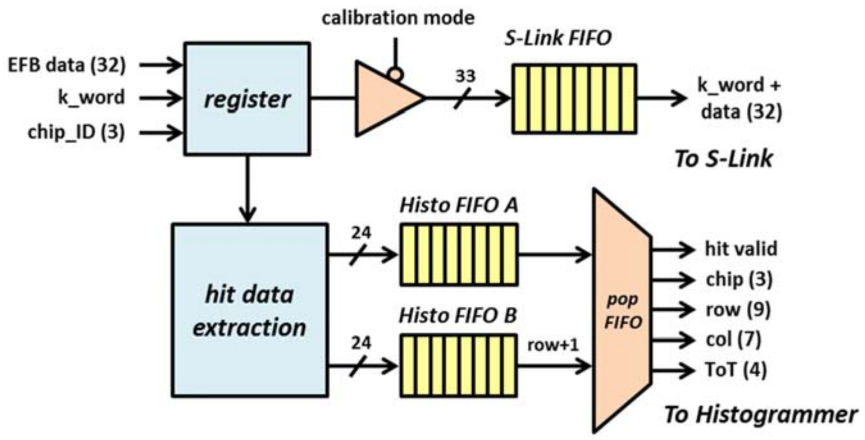


Figure 3.17: Router block diagram.

S-Link data forwarding

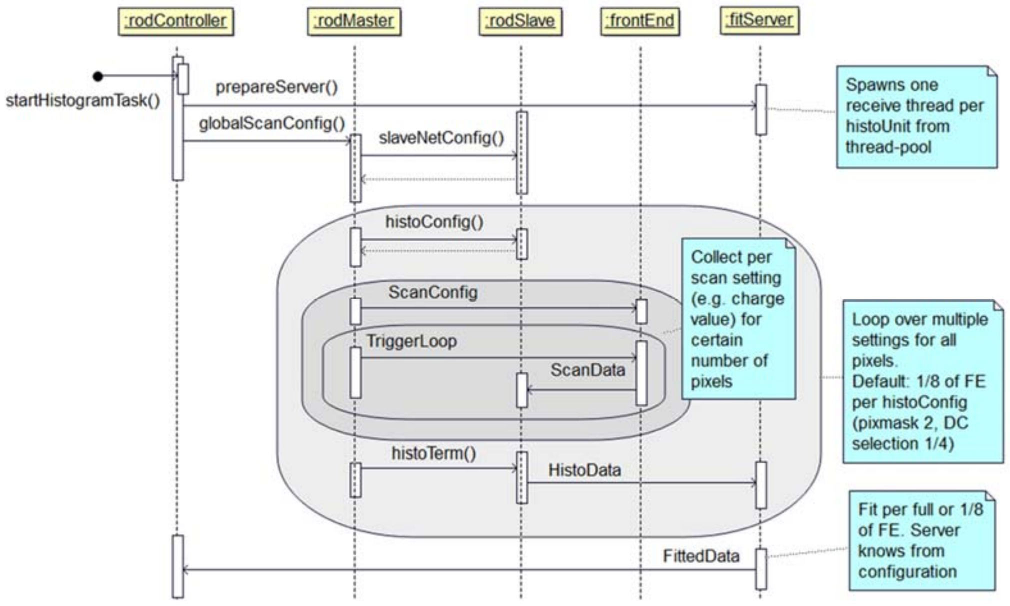
Once the S-Link-compliant data from the EFB arrives at the Router, it is pushed into a 33-bit wide 1K-word deep dual-clock forwarding FIFO prior to transmission off to the BOC. The input *k_word* concatenates to the 32-bit EFB data as the MSB to form the 33-bit FIFO input word. When the FIFO pops, this MSB is used to determine the S-Link interface control signal *slink_uctrl*. With an 80 MHz write-in speed and a 40 MHz readout speed, segmented S-Link data received from the EFB is sent to the output DDR block (at the slave top level) as a continuous S-Link packet. The S-Link FIFO was implemented as First-Word-Fall-Through to meet BOC receiver specifications. It should be noted that all interfacing signals with the external S-Link are active low. When the forwarding FIFO is not empty it simply pops its content, unless input *slink_ld* (link down) or *slink_lff* (link full flag) signals become active. These two flags are the so-called “S-Link backpressure” that could assert the ROD busy status during ROD data taking. When the Router is configured to be in calibration mode via slave register, EFB output data becomes cut off from the S-Link forwarding FIFO.

Hits extraction to histogrammer

The Router is responsible for sending event hit information to the histogrammer during detector calibration. The hit information includes 3-bit chip ID, 9-bit row, 7-bit column, and the 4-bit TOT value. A data extraction block is used to filter out non-hit data words and hits with incorrect pixel addresses (out of range row and/or column). If a hit has an invalid pixel address, the error checker inside the EFB would have flagged the error and the data is still forwarded to S-Link for analysis. It is necessary for the data extractor to decode hit data words that contain “double hits” repacked under dynamic φ-pairing mechanisms inside the FE-I4’s EODCL. All decoded single hits are pushed into FIFO A and forwarded out. Double hits data is identified by checking the TOT_2 field of an FE-I4 data record. If the 4-bit field is not 0xf, the record contains information for two neighboring hits. When a double hit data record is found, the first hit is pushed into FIFO A and the second hit is pushed into FIFO B with identical pixel addresses from each other. As a hit pops out of FIFO B, its row value is incremented by 1. The readout order of the dual FIFOs is priority-based to ensure immediate readout for double hits. Both hit storage FIFOs are dual-clocked to interface the 80 MHz EFB input data rate to the 100 MHz histogrammer operating speed. The current depth of each forwarding FIFO is 64 words, which is sufficient for current calibration requirements. In the case of whole-chip calibration (every data word contains double hits), the dual FIFOs are likely to overflow and must be increased in depth accordingly. The *histo_rfd* ready for data signal is the backpressure from the histogrammer.

3.2.7. Histogrammer and calibration

IBL calibration is perhaps the most differentiating firmware feature compared to the Pixel ROD firmware. Each slave FPGA implements two histogrammer blocks, each assigned to the output of a half-slave Router. This block is used to generate per-pixel histograms of occupancy, the ∑ToT, and ∑ToT^2 values necessary for detector calibration, as well as online monitoring. Each histogrammer handles up to eight FE-I4s with a total of 215,040 pixels, stores and updates the computed histograms in the external SSRAM. Right before each calibration run, the histogramming unit is configured to the desired run mode by the PPC through the ROD Bus. When a histogram built is completed, the slave embedded MicroBlaze transfers the results through direct memory access (DMA) to the external DDR2-RAM and further transferred off to FitServer via TCP/IP. The following descriptions of the ROD histogrammer operation are partially extracted and from the *IBL ROD-BOC Manual Draft* [12]. The UML description of the IBL calibration scan process is illustrated in figure 3.18.



:rodController
:rodMaster
:rodSlave
:frontEnd
:fitServer
startHistogramTask()
prepareServer()
globalScanConfig()
slaveNetConfig()
Spawns one receive thread per histoUnit from thread-pool
histoConfig()
ScanConfig
TriggerLoop
ScanData
Collect per scan setting (e.g. charge value) for certain number of pixels
Loop over multiple settings for all pixels. Default: 1/8 of FE per histoConfig (pixmask 2, DC selection 1/4)
histoTerm()
HistoData
FittedData
Fit per full or 1/8 of FE. Server knows from configuration

Figure 3.18: UML diagram for IBL ROD calibration [14].

Histogrammer Operation

The histogram collection for threshold and ToT scans requires simple arithmetic operations which can easily be executed in the ROD slave FPGA devices. For the threshold scan a 7-bit adder is needed to accumulate the pixel response during 100 iterations per charge step. The FPGA can perform histogram updates at a speed of 100MHz, using the logic shown in figure 3.19. Each FE-I4 can generate hits at a maximum rate of 11MHz. Serializing data from 8 FE-I4 chips results in a maximum hit rate of 85MHz. The chip number, together with row and column number from the data stream, are translated into an absolute address for the internal histogrammer memory. For every hit the corresponding value is read from the memory, incremented and written back to the memory in a pipelined fashion, such that one input hit can be processes in every clock cycle. In case of the ToT scan, 2 additional operations are needed: summing the 4-bit ToT values, and summing of the squared ToT values.

Mask stepping

During calibration there exists the possibility to sample data only from a fraction of the connected front-end chips: One can select to either sample histograms from one or from all available FE-I4s. Additionally one can choose to sample every row or only every 2nd, 4th, or 8th row in order to take the limitations of the FE-I4 into account and allow for a parallelization of the histogramming and fitting tasks.

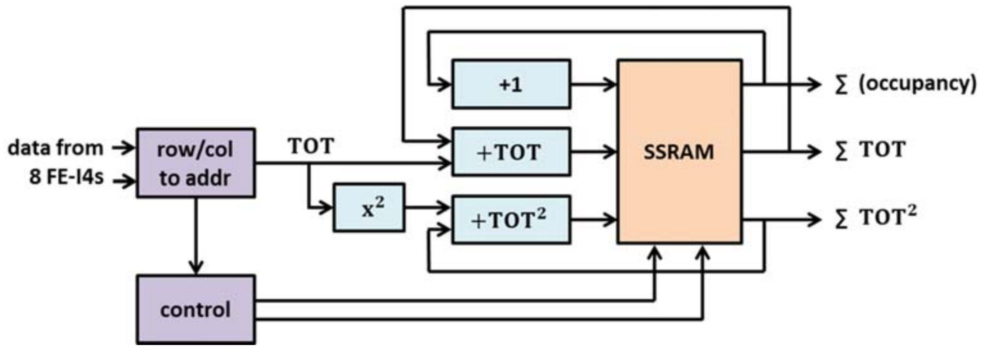


Figure 3.19: Simplified histogramming diagram [12].

Readout

There are currently two available modes for reading out the external SSRAM: one mode transfers all the values that were gathered for each individual pixel, whereas the second mode combines occupancy values of four pixels into one data word, thus reducing the overhead of transferring ∑ToT and $\sum\text{ToT}^2$ values when they are not needed (during a threshold scan for example). The readout uses a DMA transfer from the external SSRAM to the DDR2-RAM that is accessible by the MicroBlaze CPU.

Network Transfer

As soon as the histograms are relocated to the DDR2-RAM the MicroBlaze performs their off-ROD transferring using the Gigabit Ethernet interfaces. It uses an embedded TCP/IP stack to send the data to the FitServer for further processing. Studies with a comparable setup (CERN-THESIS-2012-067) indicate transfer rates of 140 Mbit/s, while the foreseen use of Jumbo Ethernet frames might push these to well above 200 Mbit/s, allowing for a safety factor of at least 2 compared to the bandwidth of the incoming data from the front-ends.

Scan Modes

There will be four readout modes which should cover all use cases of the histogrammer but at the same time allow for an efficient transfer of the relevant data:

- **LONG_TOT:** Reads out the complete 36 bits/pixel thus generating two data words per pixel.
- **SHORT_TOT:** Computes the difference between expected hits and actual hits that a pixel receives and packs this value together with ∑ToT and $\sum\text{ToT}^2$ into one 32-bit word – to be used for scans that are well above threshold.
- **ONLINE_OCCUPANCY:** Samples and reads out 24-bit wide occupancy values only - to be used during data-taking for on-line monitoring.
- **OFFLINE_OCCUPANCY:** Combines the 8-bit occupancy values of 4 pixels into one 32-bit word – to be used for threshold scans.

Work in Progress

With a maximum hit rate per histogrammer of 85 MHz, access to the SSRAM needs to be operated twice as fast at 170 MHz as each hit results in a read and write operation. Work on the SSRAM is ongoing. At the moment each histogrammer operates at 100 MHz clock and updates the SSRAM at 50 MHz.

3.2.8. Slave busy module

Every ROD slave provides the ROD master with an active-low busy signal. The Master processes slave busy signals with its busy logic and notifies the TTC to stop sending triggers to the front-end. During system reset or configuration, the entire ROD is defaulted to the busy status. The slave busy module counts busy occurrences of all busy sources and produces histograms which are accessible through slave register readout. Specified busy sources can be forced active or masked during testing and code development. A source with high busy occurrences may require resizing of its buffering elements. The high-level implementation of the slave busy module is illustrated in figure 3.20.

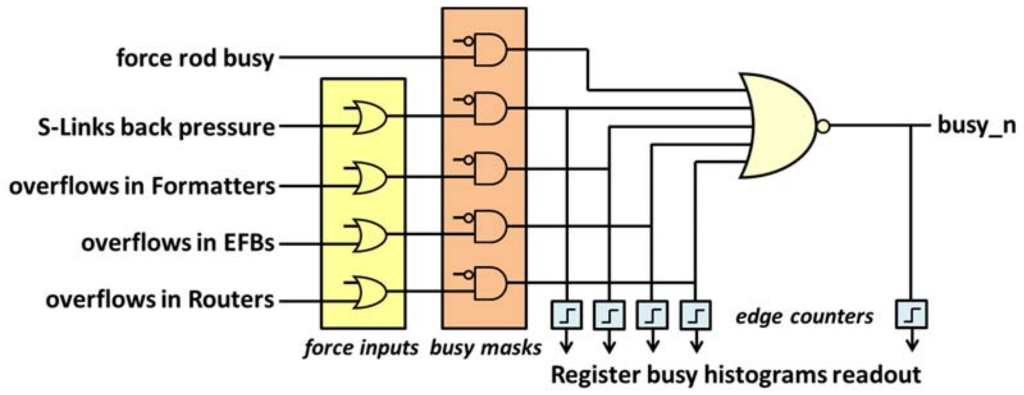


Figure 3.20: ROD slave busy logic diagram.

The sources contributing to IBL ROD slave busy include:

- **S-Link backpressure:** The ROD slave receives active-low "link full" and "link down" flags from S-Link 0 and S-Link 1. When ROS buffers are full or the links are not functional, Slave asserts a busy to avoid data loss. These backpressure signals should be the dominant busy sources during ROD data taking since events are not expected to pile up anywhere from Formatter to Router outputs.

- **Formatter link FIFOs:** The Pixel ROD Formatter implements a ROD busy limit for its link FIFOs. When one or more of the FIFOs reaches an occupancy level greater than or equal to the user-defined rod busy limit, the ROD asserts the busy signal as there is no more event storage space. This busy source has yet to be implemented in the busy module.

- **EFB internal FIFOs:** If any of the EFB output event FIFOs, error summary word FIFOs, or data word count FIFOs become almost full or full, a busy signal is asserted. Event ID FIFOs are not currently accounted for by this busy source, but should be implemented in future iterations.

- **Router internal FIFOs:** If the dual-clock FIFOs to S-Link, or any of the hits decoder FIFOs are almost full or full, a slave busy should be asserted. The S-Link FIFOs are less likely to become full since they forward data to the S-Link at a higher rate than the FIFO write-in speed. While hits decoder FIFOs simply act as information forwarding FIFOs to the histogrammers, they may become full quickly when large amount of pixels are injected with calibration charges (whole-chip calibrations). In that case, the hits decoder FIFO depths should be increased to accommodate the hits pile up.

3.2.9. Datapath FIFOs sizing

Considerations should be taken in sizing customized FIFOs used in ROD data processing. Figure 3.21 illustrates the relationships between the de-randomizing FIFOs used for the ROD datapath. The quarter version of a complete ROD slave datapath shown in this figure is adequate to help visualize the sizing strategy discussed in this section. FIFO-sizing in ROD slave firmware should mainly consider the following factors:

- **Expected event sizes:** The number of hits per FE during an event. Expected event size may increase for detectors operating at higher luminosity. The event size also depends on requirements for calibration (i.e. number of injected hits during a scan).
- **Expected event pile-up:** The number of events allowed for temporary storage.
- **FIFO asymmetry:** Different read and write speeds or asymmetric read/write data widths.
- **Application:** Whether the FIFO is used for de-randomizing storage or simple forwarding.

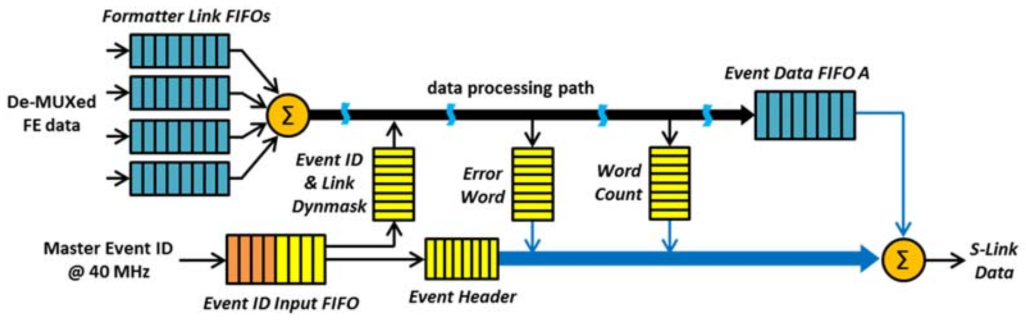


Figure 3.21: De-randomizing FIFOS in ROD slave datapath.

The blue FIFOs in figure 3.20 should primarily consider the expected event sizes during sizing. Since the event data FIFO must hold all data processed from the four Formatter link FIFOs, its depth is sized four times the depth of a single Formatter link FIFO. Currently, Formatter link FIFOs are 1K-deep each and the event data FIFO has a depth of 4K.

The event ID input FIFO implemented inside the EFB event decoder receives event description data from the ROD master at 40 MHz and pushes decoded data out at 80 MHz. Since most of the event description FIFOs inside the EFB (FIFOs in yellow) operates on 80 MHz, the Event ID input FIFO depth is allowed to be sized half as deep. One decoded event ID word requires a set of 8 input event ID words. In order to maintain the same event ID buffering capacity, the event ID input FIFO must be eight times deeper than the others. After combing both factors above, the event ID input FIFO should be sized four times deeper than the event description FIFOs. Currently all event description FIFOs have depths of 256 words, and the event ID input FIFO has a depth of 1K.

Data forwarding FIFOs such as the ones in the Router are sized in a different manner from FIFOs used for de-randomizing. Since these FIFOs simply push out data whenever they are not empty in the absence of data backpressure, read/write asymmetries and the readout mechanisms are the primary concerns for their sizing. All Router FIFOs are implemented as dual-clock FIFOs with asymmetric write and read speeds. The FIFOs could only experience overflow when the read clock is slower and the forwarding event is too large. The Router dual hits decoder FIFOs may experience overflows if the majority of the data records contain “double hits.” These FIFOs would overflow if the histogrammer is slow (speed upgrade in progress). Without considering other TDAQ components, sizing of data forwarding FIFOs inside ROD slave sets another upper limit to the detector trigger rate.

The currently implemented FIFOs provide adequate buffering for current testing and development. The depths of all ROD datapath FIFOs should be revised as the IBL operational requirements become clearer. If block memory usage becomes high in the firmware design, the use of LUTs as distributed FIFOs is recommended.

Chapter 4

Datapath Simulation and Hardware Test Results

During ROD datapath firmware development, testbenches were constructed to verify the functionality of each VHDL code block. Initially, small testbenches for individual small code blocks were tested in Xilinx ISim with forced inputs. Upon overall integration of the datapath firmware, the need for realistic data output from the FE-I4 became apparent, as simple FE-I4 emulators cannot generate the service records required to test datapath error handling. The ModelSim implementation of the realistic FE-I4 HDL model in a full datapath testbench has contributed to a better understanding of the firmware requirements, as well as lowered debugging efforts. The descriptions of this model, as well as its simulation results with off-detector readout firmware, are presented in section 4.1 of this chapter. Beside simulations, section 4.2 describes the hardware tests that were conducted to verify ROD datapath firmware. Methodologies and the results of hardware testing are described in the following section.

4.1. Testbench simulation using realistic FE-I4 model

The motivation behind testbench simulation is to verify the readout functionality of the IBL ROD according to specifications. Utilization of a realistic FE-I4 model in a complete off-detector FPGA simulation environment can speed up the debug processes and help firmware developers to better understand the readout chain. The ability for the simulator to spy on the firmware's internal signals and data flow greatly reduces the development cycle that could otherwise be experienced by hardware testing alone. Since real FE-I4s and readout hardware are scarce among developers, a ModelSim testbench implementing a single realistic FE-I4 model and the off-detector readout firmware has been realized.

4.1.1. FE-I4 HDL model

The realistic FE-I4 model (FE-I4B design version) was synthesized from the HDL designs provided by the FE-I4 design team. A behavioral model is used to represent the analog regions of the chip, while register-transfer level (RTL) models describe the digital logic blocks. A complete FE-I4 model is a collection of Verilog or System Verilog descriptions of the analog column, DDC, EODCL, EOCHL, command decoder, configuration register, and the data output block. Circuit blocks that are not essential for control and data generation (i.e. PLL, LVDS drivers, etc.) are excluded from the model. In order to protect the intellectual property of the chip designers, the model was packaged and encrypted into a "black box" model displaying only the external pins.

The creator of this FE-I4 model excluded power pins, as well as many other I/Os that are not used by the IBL. A block diagram of this model is illustrated in figure 4.1. For the convenience of development, only positive ports of differential inputs are active. Compare to the simple FE-emulator used during the early IBL ROD prototype developments, this powerful model supports:

- Same configurations to FE-I4B.
- Incrementing of L1ID and BCID counters.
- Several operating modes (different triggering modes, calibration charge injection, etc.).
- Direct TOT injections into the desired pixel arrays
- Front-end records generation.

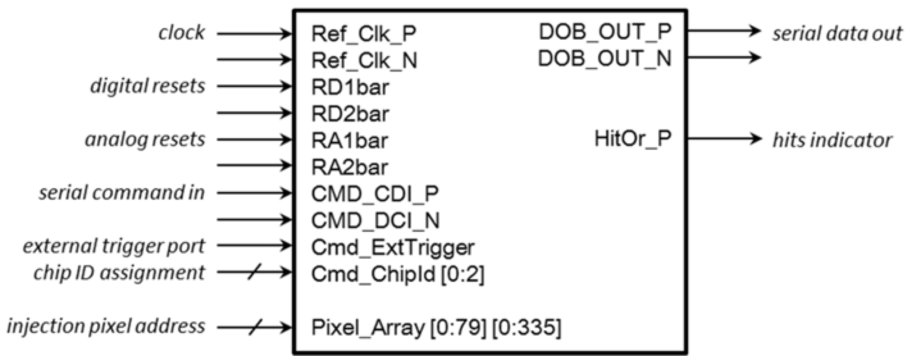


Figure 4.1: Realistic FE-I4 HDL model top-level diagram.

4.1.2. Testbench Setup

The simulation testbench setup in this work concerns only the off-detector readout datapath without the TIM and ROD master. All FPGA embedded software components were excluded in order to reduce simulation time. A ModelSim Tcl script for ModelSim was created to generate libraries, compile source code, and specify parameters for the simulation environment. A few additional *.wave* files were created for automatic waveform generation. Much of the waveform is color-coded with appropriate display radices for ease of reading. An illustration of the testbench environment is shown in figure 4.2. Due to the limited memory of the machine used for simulation, only one realistic FE-I4 model was used. Its serial data output was replicated 16 times by the simplified BOC firmware to provide full data inputs to a ROD slave. The CDR-8b/10b decoder and the BOC2ROD multiplexer block provide sufficient BOC functionality for the off-detector readout datapath without data monitoring. In addition to datapath firmware, two VHDL packages were created to accommodate the ROD control path in the absence of the ROD master firmware and the PPC. The *fei4_pack.vhd* file provides procedures to interface the FE-I4 model by generating serialized commands. The *rodslv_pack.vhd* provides the PPC's ROD bus driver to access the ROD slave registers as well as sending user-defined event ID data to the slave EFB block. While the ROD slave histogrammer blocks and the SSRAM HDL models were included in this testbench, their functionalities have not been verified in this work.

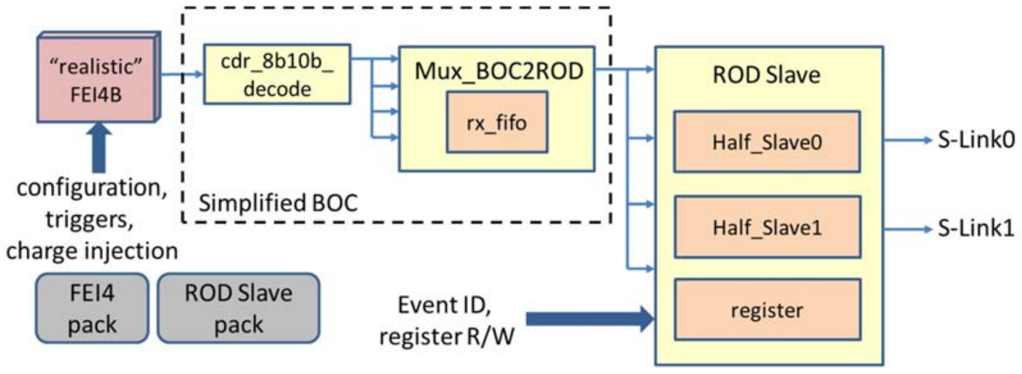


Figure 4.2: Datapath testbench setup implementing one realistic FE-I4 model.

The VHDL testbench process begins with resets for all components. An immediate slave register read/write test sequence is initiated in parallel to configure the FE-I4 model. The failure to write and read back certain registers indicates the possibility of incorrect source code compilation. When the FE-I4 model has been configured and put into "run mode," the differential data output pair is initialized. It then produces a continuous stream of serialized 8b/10b data. Data generated by the FE-I4 right after its initialization is meaningless and should be discarded by flushing all internal datapath FIFOs via the ROD slave register. Once the ROD slave is ready for data-taking, the testbench user can begin sending

L1A triggers to the front-end model or digitally inject hits into the desired FE-I4 pixel arrays. The TOT value of an injected hit is specified by the forced-to-high duration of the specified pixel address. The code example for injecting a single TOT=2 hit into a pixel located in the first row and first column would be:

```
Pixel_Array_TB_inv(0) <= '1';
wait for 55 ns; -- more than two 40 MHz clock periods
Pixel_Array_TB_inv(0) <= '0';
```

The observed TOT decimal value for this particular hit becomes “1” at the output of the ROD slave Router as an encoded TOT by the FE-I4. The TOT encoder table [10] is provided in appendix B. The *HitOr_P* signal at the FE-I4 output is an useful indicator for quickly identify non-empty events. If the testbench user sends L1A triggers to the FE-I4 model during simulation, the model will generate data headers and service records without any data records.

For every event that is either triggered or injected, its corresponding event description must also be sent to the slave EFB for event processing. The event ID, as well as the link dynamic masks, are user-defined and sent to the slave firmware with procedures defined in the *rodslv_pack.vhd*.

4.1.3. Simulation Results

The simulation results presented in this section illustrates FE-I4 and ROD datapath functionality in an illustrative fashion. The presented waveforms begin with the FE-I4 configuration to the output of the S-Link ODDR. Waveforms internal to the ROD slave are also shown in this section.

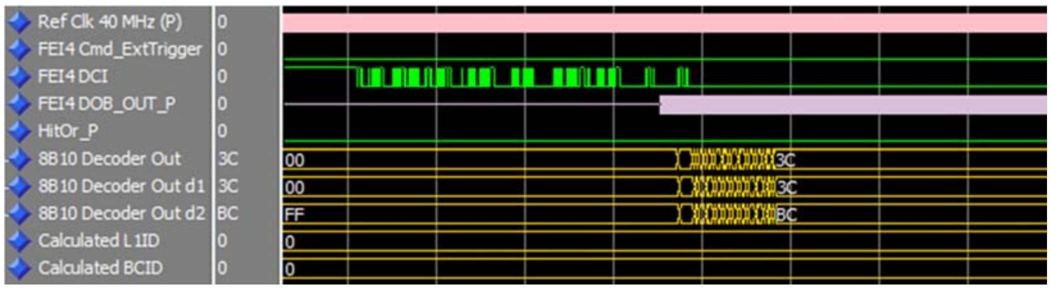


Figure 4.3: Prior to sending triggers or injecting hits into the FE-I4, the front-end model must be configured and put into run mode. The correct set up of the testbench is indicated by a continuous output stream on the serial *DOB_OUT_P* signal. After this signal is decoded by the BOC CDR and 8b/10b decoder, the signal would stabilize at the DC-balanced hexadecimal “3c.” Some junk data is generated by the FE-I4 right after its initialization which must be discarded prior to any data-taking tests.

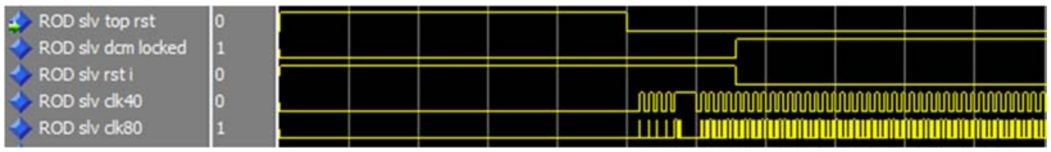


Figure 4.4: Soon after the simulation begins, the ROD slave digital clock manager (DCM) generates a 40 MHz and an 80 MHz clock signals from the BOC clock input. The ROD slave global reset signal is based on the combination of external reset and whether DCM is locked or not. The DCM must be locked to its source clock in order for the ROD slave to operate.

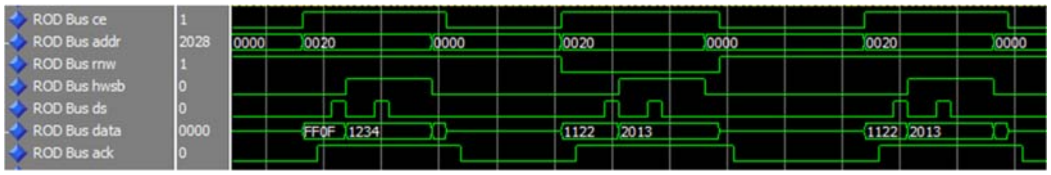


Figure 4.5: Prior to testing the ROD datapath, several slave registers must be configured accordingly. It is also important to test the read and write functionality of the registers before other tests. In this figure, the Formatter control register at the address 0x0020 is read, written over, and read back correctly.

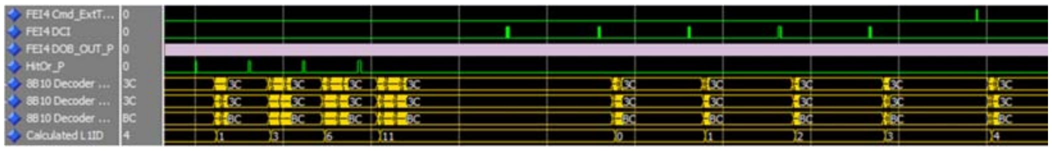


Figure 4.6: Three types of FE-I4 triggering modes are illustrated in this waveform. The first four triggers are self-triggers. They are achieved by injecting TOT hits into the FE-I4. When the FE-I4 detects hits, the *HitOr_P* signal is pulsed. This pulse is re-routed back to the command decoder block to produce a self-trigger. Note that the L1 ID is not incrementing in any meaningful way by design during self-trigger mode. The FE-I4's internal event counter is reset via an ECR after the self-triggering sequence. The second type of triggering demonstrated here is the L1A command trigger via the serial *DCI* port. The last trigger in this waveform is an external trigger.

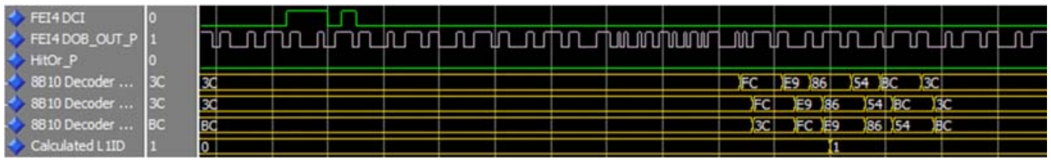


Figure 4.7: The FE-I4 model is triggered via the L1A request on the FE-I4 *DCI* line. No service record is generated. This is a second L1A-triggered event and has an L1ID of 1.

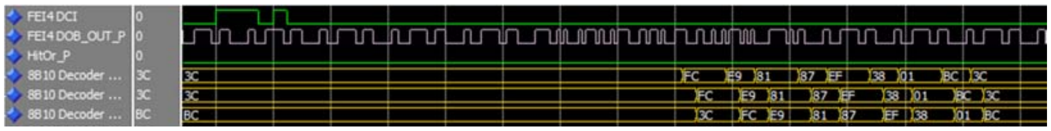


Figure 4.8: The FE-I4 model is triggered via the L1A request on the FE-I4 *DCI* line. The L1A trigger is a short command with a bit sequence of "11101." Beside the BOF frame, data header, and the EOF frame, a service record is generated. This service record (SR 14) is and informational record generated due to an increment on the 3 MSBs of the FE-I4's BC counter (from 0 to 1).

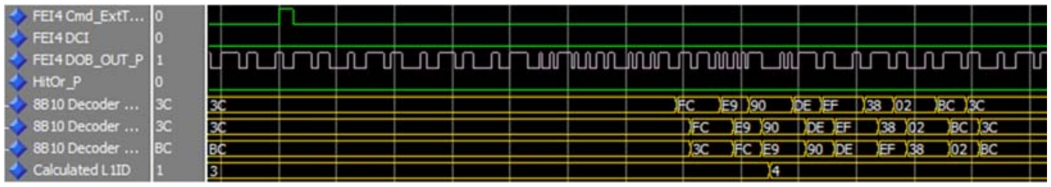


Figure 4.9: The FE-I4 model is triggered via an external pad. The external trigger is pulsed with a duration of 25 ns, which covers one 40 MHz clock period. Beside the BOF frame, data header, and the EOF frame, a service record is generated. This service record (SR 14) is and informational record generated due to an increment on the 3 MSBs of the FE-I4's BC counter (from 1 to 2).

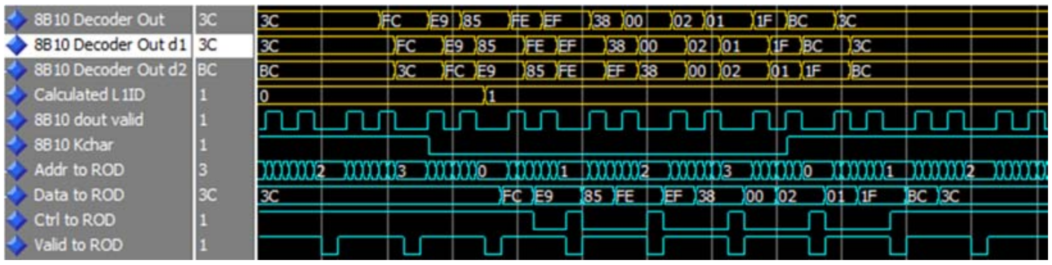


Figure 4.10: A hit with TOT = 2 is injected into the first pixel with the address {row : column} = {1 : 1}. The FE-I4 *DOB_OUT_P* is replicated four times at the inputs of the *MUX_BOC2ROD* block. Link 0 to Link 3 receive the exact same FE-I4 data.

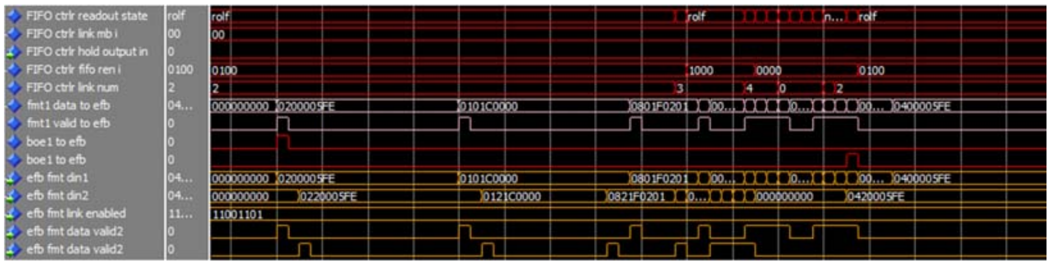


Figure 4.11: A Formatter gathers FE data from the BOC with chip addresses based on link masking. The Formatter produces a *boe* pulse at the beginning of event and an *eoe* pulse at the end of an event to the EFB.

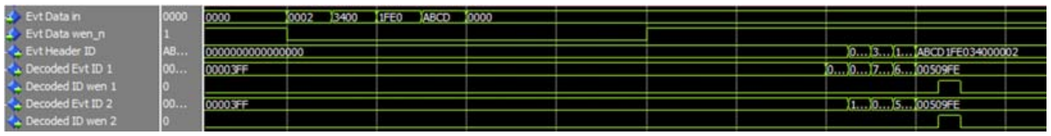


Figure 4.12: A packet of “dummy” event ID is sent to the ROD slave event data decoder block to mimic the event IDs and EFB dynamic mask bits sent out by the ROD master during the actual triggering mode.

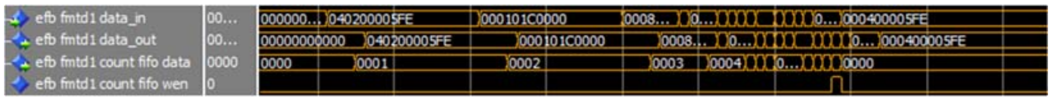


Figure 4.13: After the EFB event ID checker and the error detector, the EFB data formatter count the number of data words in an event writes the count to the corresponding event data count FIFO at the end of this event.

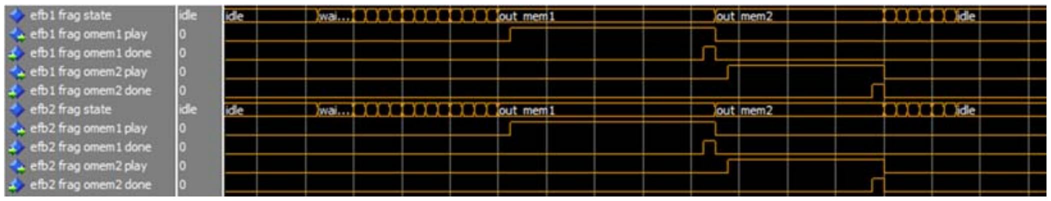


Figure 4.14: Once the event has been processed and event data becomes available in the de-randomizing FIFOs inside the EFB, a S-Link fragment will be generated. The EFB fragment generator generates the S-Link header words, plays the EFB event data FIFOs, and lastly the S-Link trailer words.

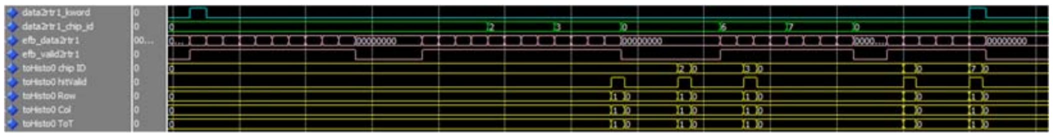


Figure 4.15: The Router block outputs the fragmented S-Link data as well as extracting hit information to the histogrammer. The first and last data elements of the S-Link event are accompanied by the 1-bit control K words. While FE-I4 data is replicated at each link, only links 0, 2, 3, 6, and 7 are enabled in the Formatters through the ROD slave register. The injected hit is correctly extracted with the “encoded” TOT of 1 (true TOT of 2 in the default setting of hit discrimination).

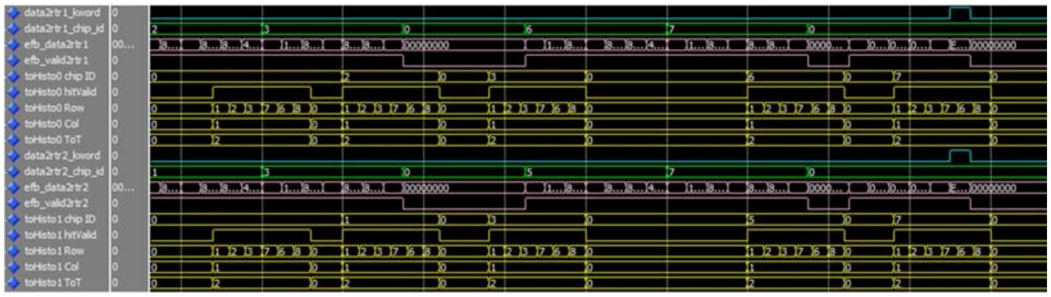


Figure 4.16: Correct S-Link data and hits extraction for hits injected to neighboring pixel addresses are shown with links 0, 2, 3, 6, 7, 8, 9, 11, 13, and 15 enabled in the Formatters. Due to the priority FIFO read out mechanism used during hits extraction, decoded hits are not always forwarded to the histogrammer in strict ascending address order. This phenomenon does not affect detector calibration operations.

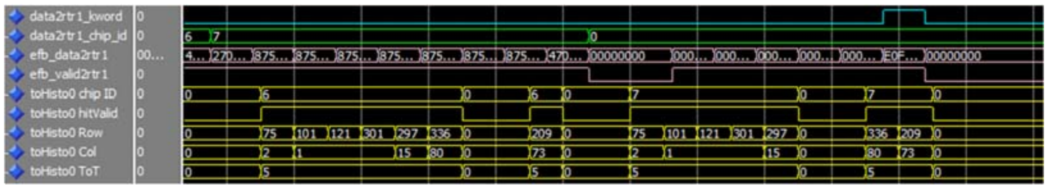


Figure 4.17: Due to asymmetric read and write speeds of the dual hits decoder FIFO inside the Router, the extracted hits information is not always forwarded to the histogrammer in a continuous stream. This behavior does not affect detector calibration provided that sufficient FIFO depths are used for buffering during high occupancy.

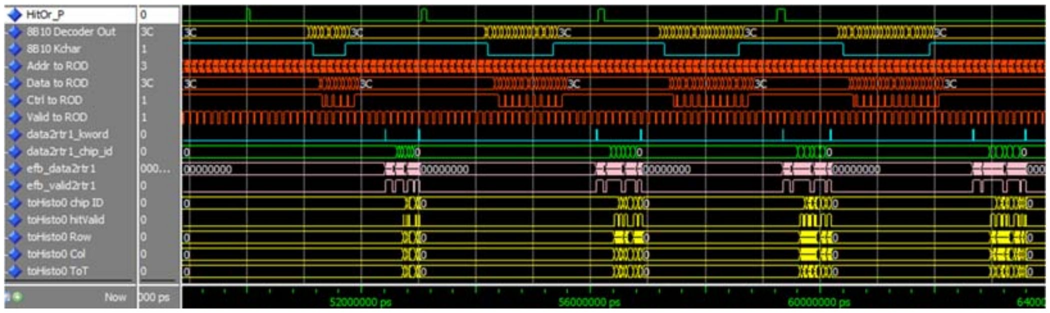


Figure 4.18: The FE-I4 model receives a sequence four hit injections. The ROD slave datapath is shown to have generated correct S-Link data and extracted hits for the histogrammer. The self-triggering rate in this test is set at roughly 330 KHz. While a triggering rate of 100 KHz is expected for IBL, higher triggering rate is allowed during testing where test event sizes are small.

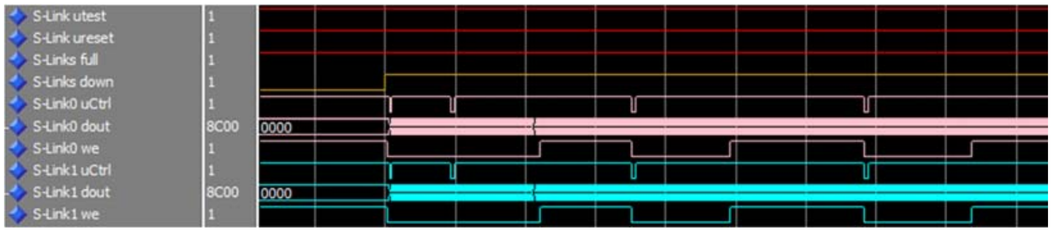


Figure 4.19: The outputs of the S-Link ODDRs at the ROD slave top level. This waveform shows the Router holding the S-Link data in the S-Link forwarding FIFO until the S-Link backpressure *S-Link down* (active low) is relieved.

There are two scaling limitations in simulation. First, a realistic FE-I4 model would consume up to 2G of computer memory, which is not ideal for scaling the simulation on machines with limited memory resources. It is ideal to use 8 discrete FE models to feed the minimal ROD datapath in the testbench so each model receives unique hit injection patterns for datapath test. The current approach duplicates the output from one FE model and provides identical data at the Formatter inputs. While identical input data is ideal for verifying the datapath processing homogeneity, unknown failure patterns may not be observable unless different hits were injected across multiple FEs. The second limitation to this simulation approach is the time-scaling for realistic trigger tests. In IBL, FE-I4s will typically be triggered at 100 KHz. Such a time scale becomes difficult to implement in the simulation environment, especially when millions of triggers are required for realistically testing event synchronization. While running the testbench simulation at the normal IBL trigger rate for multiple events can be time consuming, one could inject smaller events or send empty triggers to the FE model at higher trigger rates for synchronization tests. Hardware testing using ChipScope data sampling is more efficient for longer test durations. The setup and results for ROD slave firmware are discussed in the next section.

With the simulation time of a few hundred microseconds or less, most of the ROD slave functionality can be tested for correctness. This simulation environment provides convenient spying on the unit-under-test's internal processes. The high observability of data flow inside the firmware allowed debugging to be much more efficient. By utilizing a powerful FE-I4 model, along with essential datapath firmware components, much of the ROD slave functionalities have been verified. Core datapath requirements that were verified include FE link masking, event ID decoding, register control, error handling, S-Link data generation, and histogrammer hits decoding.

4.2. Hardware Testing

Hardware testing is the final and most crucial part of the firmware development cycle before deployment. All hardware tests described in this section were conducted at CERN, Geneva. The tests performed include trigger tests using TIM and readout tests with the ROS. Calibration results are also provided in this section.

4.2.1. TIM trigger test

Trigger tests using the TIM trigger source were the first hardware tests conducted for this firmware design. The setup of the test began by programming all FPGA devices on the BOC and ROD cards. When the readout electronics became ready, a dual-FE-I4 module was initiated (using the *miniDCSclient* GUI). Next, a PPC code was used to set up the slave register as well as clearing any internal buffers in the datapath. Lastly, the TIM was programmed with the desired trigger type, triggering rate, and intervals for event counter reset (ECR).

In order to debug the firmware and check data correctness at the ROD slave level, a Xilinx ChipScope embedded core was inserted into the slave firmware. ChipScope is an integrated logic analyzer used to directly probe signals at die level. By setting the appropriate sampling criteria (also called "trigger"), desired data can be stored in and readout from a 1K-deep sampling buffer. In order to better observe event synchronization between the TIM trigger source and the FE-I4 event data, an event counter was added to the ROD slave. During successful tests, events were synchronized between all devices at trigger rates up to 400 kHz. Figure 4.20 shows the ChipScope sampled data during a TIM trigger test at a triggering rate of 200 kHz. Since there were no charges injected to the FE-I4s, the front-end sends back only the data header and a service record after receive each trigger. During one test run, triggers and events were observed to be in sync over a period of more than 10 hours at 200 kHz.

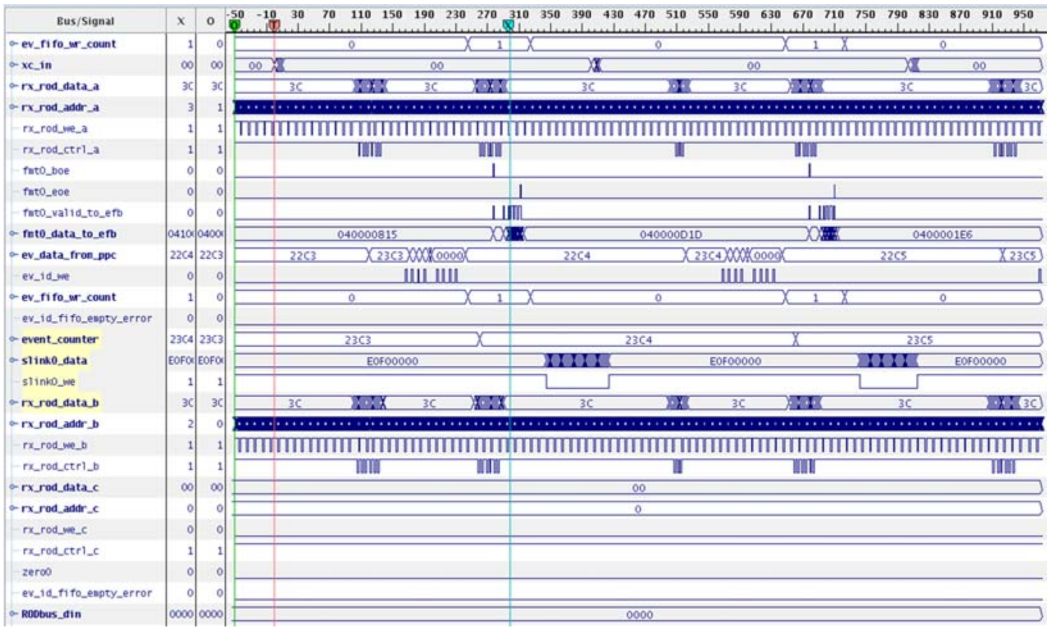


Figure 4.20: ChipScope waveform capture of TIM trigger test using a dual FE-I4 module.

4.2.2. ROS readout test

Hardware testing including the ROS input buffer hardware ROBIN was also performed for the datapath firmware at CERN. Two different data sources were used during the ROS readout test: RoL test data generated by the ROD slave EFB, and the FE data emulators located on the BOC BMF. The readout chain of this test involves the TIM, BOC, ROD, S-Link, and ROBIN. In the case of readout tests using emulated FE data, TIM triggers were sent to all 32 simple FE-I4 emulator inside the BOC.

While the readout electronics can process triggers at a rate greater than 200 kHz during the test, event arrivals to ROBIN at such a rate quickly resulted in buffer overflows. Eventually, a successful ROS readout test for 16 million emulated events at a triggering rate of 50 kHz was conducted. No data corruption was observed by the ROS.

Since the IBL off-detector electronics must produce S-Link data in the same way as the Pixel readout, the IBL ROS readout tests could ensure the correct IBL integration into the ATLAS Experiment.

Chapter 5

Conclusion and Outlook

The off-detector readout electronic system is the critical path between the IBL detector front-end and the backend systems during ATLAS physical experiments. By adopting powerful FPGAs as front-end data processors, the IBL ROD was able to achieve higher levels of integration compare to the Pixel ROD. The presented work focuses on the development and characterization of the ATLAS IBL ROD datapath firmware. The integration and adaptation of the Pixel ROD firmware for IBL, the creation of an abstracted off-detector readout testbench implementing a realistic front-end model in simulation environment, and testing hardware functionality of the off-detector DAQ system, are achieved in the framework of this thesis. At the present moment, ROD development and testing are ongoing across the IBL collaboration.

While a suite of ROD firmware has been developed to fulfill basic IBL operational requirements, advanced features to ensure optimal performance must be developed and tested prior to the full commissioning of IBL TDAQ by the end of the ATLAS Phase 0 Upgrade in 2015. Foreseeable IBL ROD firmware development tasks for the near future are summarized in the following:

Robust busy recovery mechanism: Readout electronics must be capable of swift and automatic recovery from all possible busy scenarios to ensure uninterrupted LHC experiments. Therefore it is unacceptable for the IBL ROD to have frequent hiccups and halt TTC triggering for all other ATLAS detectors. While the current ROD busy implementation allows masking of problematic busy signals, the firmware should provide low level self-recovery mechanism. For example, the current firmware allows disconnection of any faulty Formatter data links through the slave register control. Special care should be taken to ensure event and trigger re-synchronization in the case of event loss during ROD recovery.

Development of ROD built-in test firmware: A ROD built-in test suit should be implemented to perform system diagnosis. The test should focus on both the control and datapath features. It would be ideal to implement standalone ROD data emulation. Presently, the existence of FE-I4 emulators inside the BOC BMF may provide sufficient features. The primary goal of the built-in test is to emulate a potential crisis for the IBL and its TDAQ to verify ROD performance under such scenarios. The secondary goal of the built-in test is to inject timeout scenarios or incorrect data fragments and to verify if the ROD datapath can handle faulty data accordingly. To do so, an FE-I4 data frame synchronization checker should be implemented in the ROD slaves. A few mechanisms for ROD-level error handling exist but were not extensively tested in this thesis work. Another built-in test for the ROD is the automatic data processing check. In this case the testing blocks should generate event ID information and produce triggers in a realistic way that accounts for signal delays. Several diagnostic schemes for the Pixel ROD described in [13] should be studied and ported to the IBL ROD.

Integration of Pixel FE-I3 readout into IBL ROD: Identical IBL ROD cards to those described in this thesis will be commissioned as the readout drivers for Layers 1 and 2 of the Pixel Detector in 2015 to upgrade their readout bandwidth under higher occupancies. Therefore it is necessary to implement the FE-I3-MCC readout firmware into the IBL ROD. Given the resources available in the Spartan 6 FPGAs used as ROD slave data processor, a single version of readout firmware for both Pixel and IBL should be viable.

Complete off-detector readout simulation environment: A more powerful off-detector DAQ readout simulation environment should be created to help verify the full control and readout operations. While a readout simulator implementing a realistic FE-I4 model has been created in this thesis work, the testbench did not incorporate the ROD master, ROD PRM, BOC BCF (master), and the full BOC BMF firmware. An off-detector readout environment incorporating all, or the necessary subset, could be even more valuable to IBL off-detector developers.

Bibliography

[1] The ATLAS Collaboration, “ATLAS High-Level Trigger, Data Acquisition and Controls”, CERN-LHCC-2003-022.

[2] ATLAS Experiment © 2014 CERN. “ATLAS Fact Sheet”, http://www.atlas.ch/pdf/ATLAS_fact_sheets.pdf

[3] ATLAS Experiment © 2014 CERN.

[4] The ATLAS Collaboration, “ATLAS Insertable B-Layer Technical Design Report”, CERN-LHCC-2010-013.

[5] Butter,worth, J; Lane, J.B; Postranecky, M; Warren, M.R.M, “TTC Interface Module for ATLAS Read-Out Electronic: Final production version based on Xilinx FPGA devices”, ATL-ELEC-PUB-2007-004, September 2004.

[6] Polini, A. et al. “Design of the ATLAS IBL Readout System”, Physics Procedia, Volume 37, pp 1948-1955, 2012.

[7] Van der Bij, H. C.; McLaren, R.A.; Boyle, O.; Rubin, G., "S-LINK, a data link interface specification for the LHC era," *Nuclear Science, IEEE Transactions on* , vol.44, no.3, pp.398,402, Jun 1997

[8] Kugel, A. et al. “The final design of the ATLAS Trigger/DAQ Readout-Buffer Input (ROBIN) Device”, Proceedings of 10th Workshop on Electronics for LHC Experiments and Future Experiments, Boston, 2004.

[9] J. Dopke, D. Falchieri, T. Flick, et al. “Study of FPGA and GPU Based Pixel Calibration for ATLAS IBL,” May 2010.

[10] The FE-I4Collaboration, “The FE-I4B Integrated Circuit Guide”.

[11] Backhause, M. “High bandwidth pixel detector modules for the ATLAS Insertable B-Layer”, Dissertation, Universitat Bonn, January 2014.

[12] The IBL Collaboration, “IBL ROD-BOC Manual” (development draft version)

[13] Joseph, J. et al, “ATLAS Silicon ReadOut Driver (ROD) Users Manual”

[14] Originally by Andreas Kugel, Institut fur Technische Informatik als zentrale Einrichtung der Universitat Heidleberg

Acknowledgements

In the brief period of one year I have successfully migrated and upgraded the Pixel ROD firmware to IBL ROD firmware (both master and slave). The enhanced full datapath can now process the required IBL readout capacity of 32 FE-I4s. I am glad to have spent energy on the full datapath simulator using the realistic FE-I4 model because it gave me (and hopefully others) a very powerful tool to debug code quickly. During my trip to CERN this March I was able to verify the firmware functionality in hardware tests involving other VME DAQ components as well as FE-I4s. I remember the moment I have successfully integrated all ROD firmware versions under one (data-taking and histogramming). It seemed almost unreal, as the code version chaos ended.

All that being said, it would not have been possible without the patience and supports from my colleagues.

I want to thank the people who gave me the opportunity to work on this very special project. This is in particular to Prof. Dr. Shih-Chieh Hsu and Pro. Dr. Scott Hauck with whom I have enjoyed working with and learning a lot from. Through this project, they have equipped me with even more skillsets and “triggered” my fascinations with the Large Hadron Collider and FPGAs. In all, my involvement with the IBL collaboration has helped me build up undeniable street cred.

During this past one year, I enjoyed working alongside my colleagues from INFN and the University of Bologna. I appreciate the warm hospitality from Dottor Alessandro Gabrielli and his team during my visits to Bologna. I would like to extend a special thanks to Dottor Davide Falcheri, whom I have worked closely with in putting together the IBL ROD firmware. Not only have I benefitted a lot from our discussions, I have also learned much about the wonderful Italian way of life. Of course I appreciate my discussions with Dottor Riccardo Travaglini, Gabriele Balbi, and Luca Lama.

I would like to thank Ales Svetek, with the Jozef Stefan Institute, for putting together the realistic FE-I4 HDL model for the firmware developers. His consideration for protecting the FE-I4 designer’s IP is well noted. Because of this model, I was able to accelerate my development efforts for the IBL ROD datapath firmware. Without this model in my testbench simulation, I would have been throwing a rock at the apple instead of shooting at it with a laser. I would also like to acknowledge the FE-I4 design team for sharing the original files.

I would also like to thank Dr. Marilyn Marx for her support during my time at CERN as well as joining our remote discussions. Even though I have only the honor for brief face-times with some colleagues, I have found these discussions very valuable in helping me understand different components the project. Those thanks go to Dr. Karolos Potamianos, Dottor Marcello Bindi, Dr. Laura Jeanty, Doktor Malte Backhaus, Dr. Yosuke Takubo, and many others. I shall not forget the efforts Mortiz Kretz and others put into the histogrammer/calibration development. I would also like to express my gratitude for the work done by the BOC firmware developers, namely Marius Wensing.

I would like to thank Dr. John Joseph and the original Pixel ROD team. I have never met Dr. Joseph in person but his VHDL codes have no doubt provided me with a huge amount of learning. I would also like to thank the IBL and Pixel management for their support of this project.

A very special thanks goes to Doktor Andreas Kugel who, in too many instances, lent a helping hand and gave me guidance. I have learned quite a few useful design tricks from him.

Lastly, my sincere and particular thanks goes to Senteara Orwig for accompanying me through all the hard times during my entire graduate degree. Without her, I doubt the outcome would be any more ideal.

Appendix A: Slave Registers

The following register addresses are current as of June 12th, 2014. For the latest version of register mapping, refer to *rodSlaveRegPack.vhd* file in the IBL ROD Slave source code repository. As functionality of ROD Slave increase during on-going code development, register address space will also expand.

Description	Address	Access	Width
Formatter Link Enable	0x0000	RW	16
ROD Slave ID	0x0005	RW	1
Calibration Mode	0x0006	RW	1
Formatter Readout Timeout Limit	0x0010	RW	32
Formatter Data Overflow Limit	0x0014	RW	16
Formatter Header Trailer Limit	0x0018	RW	32
Formatter ROD Busy Limit	0x001C	RW	32
Formatter Control Register Bits definition reserved	0x0020	RW	32
Formatter0 Status Register Bits definition reserved	0x0022	R	32
Formatter1 Status Register Bits definition reserved	0x0023	R	32
Formatter2 Status Register Bits definition reserved	0x0024	R	32
Formatter3 Status Register Bits definition reserved	0x0025	R	32
Mode Bits Control Omitted in IBL ROD			
INMEM FIFO Channel Select Select BOC to ROD channels as data source	0x0040	RW	2
INMEM FIFO Reset	0x0042	W	1
INMEM FIFO Data	0x0044	R	12
RAM Block Select	0x1---	RW	
Formatter Readout Timeout Error	0x0070	R	16
Formatter Data Overflow Error	0x0074	R	16
Formatter Header Trailer Error	0x0078	R	16
Formatter ROD Busy Error	0x007A	R	16
Slave Control Register Formerly part of Formatter registers (not used)	0x0100	RW	32
Slave Status Register Formerly part of Formatter registers (not used)	0x0104	R	32
Slave Busy Control Register Forced busy inputs and busy masks	0x0108	W	32
EFB Format Version 0x0301UUUU U = user defined, top half fixed	0x2200	R RW	32 16
EFB Source ID 0xUU1SMMMM S = Sub Detector ID, M = Module ID	0x2204	R RW	32 23
EFB Run Number	0x2208	RW	32
EFB1 Command Register Bit[0]: 0 (send empty events) Bit[1]: Mask BCID Error Bit[2]: Group Event Counter Enable Bit[3]: Mask L1ID Error Bit[4:7]: 0 (link input test select) Bit[8]: Mask TIM Clock Error Bit[9]: Mask BOC Clock Error Bit[10]: 0 (mask sweeper error) Bit[11]: Enable L1 and BC ID Trap Bit[12]: TIM BCID Readout Link Enable	0x2210	RW	32

Bit[13]: BCID Rollover Select Bit[14:15]: 0 Bit[16:25]: BCID Offset Bit[31:26]: 0			
EFB2 Command Register See above for description	0x2212	RW	32
EFB1 Run-Time Status Register Bit[0]: FIFO1 "almost_full_n" Status Flag Bit[1]: err_sumry_fifo1_almost_full Bit[2]: ev_id_fifo_empty_error1 Bit[3]: fifo 1 pause to Formatter Bit[4]: FIFO2 "almost_full_n" Status Flag Bit[5]: err_sumry_fifo2_almost_full Bit[6]: ev_id_fifo_empty_error2 Bit[7]: fifo 2 pause to Formatter Bit[8]: halt output from router Bit[9]: rod_event_type_empty Bit[10]: rod_event_type_afull Bit[11]: rod_event_type_full	0x2218	R	12
EFB2 Run-Time Status Register See above for description	0x221A	R	32
ROD Code Version and Board Version Bit[0:7]: HDL Code Version Bit[8]: 0 Bit[9:15]: 00 & Board Revision Number	0x221C	R	16
EFB1 Event Header Data	0x2220	R	16
EFB2 Event Header Data	0x2222	R	16
EFB1 Group Event Count	0x2230	R	32
EFB2 Group Event Count	0x2232	R	32
EFB1 Out FIFOs Reset	0x2248	W	1
EFB2 Out FIFOs Reset	0x224A	W	1
EFB1 Out FIFO Status Flags Bit[0]: FIFO1 NOT "empty_n" Status Flag Bit[1]: FIFO1 NOT "almost_empty_n" Status Flag Bit[2]: FIFO1 NOT "full_n" Status Flag Bit[3]: FIFO1 NOT "almost_full_n" Status Flag Bit[4]: FIFO2 NOT "empty_n" Status Flag Bit[5]: FIFO2 NOT "almost_empty_n" Status Flag Bit[6]: FIFO2 NOT "full_n" Status Flag Bit[7]: FIFO2 NOT "almost_full_n" Status Flag Bit[8]: 0 (HTFIFO "empty" Status Flag) Bit[9]: 0 (HTFIFO "full" Status Flag) Bit[10]: ev_id_empty1 Bit[11]: ev_id_almost_full1 Bit[12]: ev_id_empty2 Bit[13]: ev_id_almost_full2 Bit[14]: header_ev_id_empty Bit[15]: header_ev_id_almost_full Bit[16]: header_ev_id_full Bit[17]: ev_data_empty Bit[18]: ev_id_full1 Bit[19]: ev_id_full2 Bit[20]: count_fifo1_full Bit[21]: count_fifo2_full Bit[22]: err_sumry_fifo1_full Bit[24]: err_sumry_fifo2_full	0x224C	R	24
EFB2 Out FIFO Status Flags See above for description	0x224E	R	32
EFB1 L1/BCID Trapped Values Bit[15:0]: latched_bcid0 & 0 & latched_l1id0 Bit[31:16]: latched_bcid1 & 0 & latched_l1id1	0x227C	R	32
EFB2 L1/BCID Trapped Values See above for description	0x227D	R	32
EFB1 Miscellaneous Status Register Bit[0]: err_sumry_fifo1_almost_full	0x227E	R	32

Bit[1]: err_sumry_fifo2_almost_full Bit[2]: ev_id_fifo_empty_error1 Bit[3]: ev_id_fifo_empty_error2 Bit[4]: ev_id_empty1 Bit[5]: ev_id_empty2 Bit[6]: header_ev_id_empty Bit[7]: fifo1_pause_o Bit[8]: fifo2_pause_o Bit[9]: gatherer_halt_output Bit[10:17]: 0 Bit[18]: header_ev_id_data_out(35) Bit[19]: ev_id_almost_full1 Bit[20]: ev_id_almost_full2 Bit[21]: header_ev_id_almost_full Bit[22]: ev_data_almost_full_n_o Bit[23]: header_ev_id_full Bit[24]: ev_id_full1 Bit[25]: ev_id_full2 Bit[26]: ev_data_empty_o Bit[27]: err_sumry_fifo1_full Bit[28]: err_sumry_fifo2_full Bit[29]: count_fifo1_full Bit[30]: count_fifo2_full			
EFB2 Miscellaneous Status Register See above for description	0x227F	R	32
Half-Slave 0 Error Masks Link 0 Link 1 Link 2 Link 3 Link 4 Link 5 Link 6 Link 7	 0x3000 0x3004 0x3008 0x300C 0x3020 0x3024 0x3028 0x302C	RW	32
Half-Slave 1 Error Masks Link 8 Link 9 Link 10 Link 11 Link 12 Link 13 Link 14 Link 15	 0x3040 0x3044 0x3048 0x304C 0x3060 0x3064 0x3068 0x306C	RW	32
Simulation Config1	0xEEEE	RW	32
Simulation Config2	0xFFFF	RW	32

Appendix B: Data Formats

B.1. ROD slave top level signals

Signal	Notes
rod_bus_ce	Target device select signal from ROD Master
rod_bus_rnw	Read-not-write
rod_bus_hswb	Half-word select bit, used for distinguish half words of 32-bit data
rod_bus_ds	Single clock cycle data strobe for sampling data
rod_bus_addr [17:0]	ROD Slaves currently use 16 bits, BOC uses 8 bits
rod_bus_data [15:0]	Bi-directional bus for read and write data
rod_bus_ack_out	ROD Slave to Master acknowledgement

Table B.1: ROD Bus interfacing signals between ROD master PPC and ROD slave register block.

B.2. Formatter data formats

Data Type	Bits [31:0]
header	001pxxnn0000MMMMFLLLLLBBBBBBBBB
hit(s)	100xxxnnTTTTttttCCCCCCCRRRRRRRR
service record	0001xxnnxSSSSSSxxxxxxxDDDDDDDDD
trailer	010xxxnnEMPxMMMMFLLLLLBBBBBBBBB

B = BCID	P = PPC link mask (unused)
C = FE column	p = preamble error (unused)
D = service record counter	R = FE row
E = timeout flag (unused)	S = service record code
F = FE flag	T = ToT(1)
L = L1ID	t = ToT(2) double hit
M = # of skipped triggers	x = don't care
n = link number	

Bit [32]	Timeout error flag (all words)
Bit [33]	Condensed mode flag (unused)
Bit [34]	Link masked by PPC flag (unused)

Table B.2: Formatter output data bits [31:0] and overhead bits definitions. This table is subject to change in the near future. Check [12] for latest format.

timeout header	0x2000dead
timeout trailer	0x4000dead
data overflow trailer	0x44000000

Table B.3: Formatter generated header and trailer during timeout and data overflow.

B.3. EFB data formats

Word 0	L1ID [15:0]
Word 1	ECRID [7:0] & L1ID [23:16]
Word 2	BCID [11:0] & x & RoL & BOC OK & TIM OK
Word 3	TT – ROD [5:0] & TIM [1:0] & ATLAS [7:0]
Word 4	Dynamic Mask for Links [7:0]
Word 5	Dynamic Mask for Links [15:8]
Word 6	Dynamic Mask for Links [23:16]
Word 7	Dynamic Mask for Links [31:24]

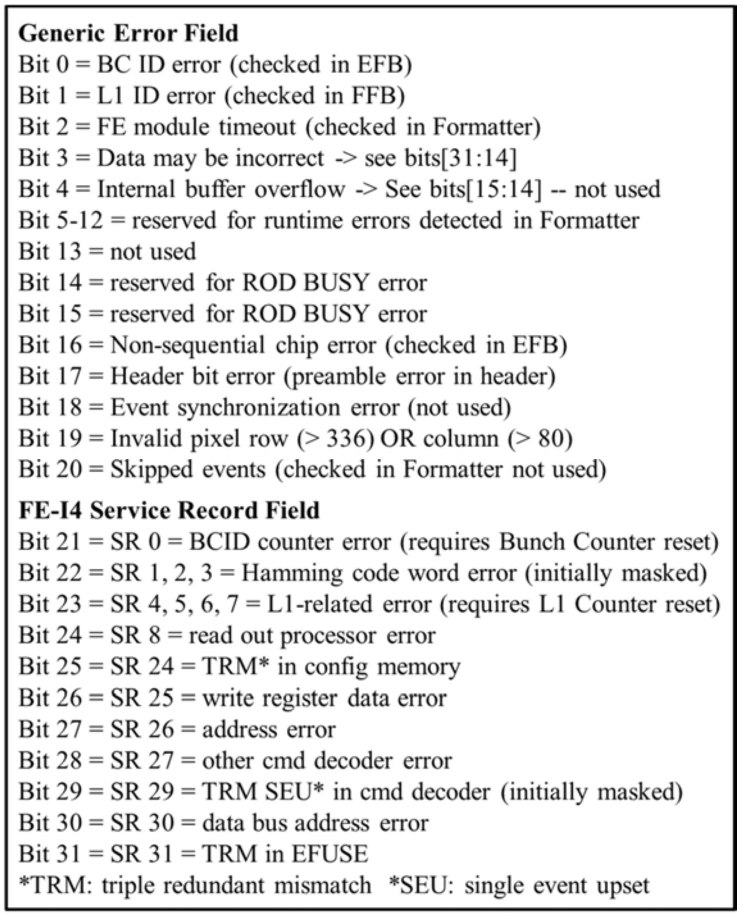
Table B.4: Event description data from ROD master event processor to slave event data decoder.

Bits	Event ID Data 1	Event ID Data 2
[9:0]	BCID	BCID
[14:10]	L1ID	L1ID
[18:15]	ROD Trig Type (4 LSBs)	ROD Trig Type (4 LSBs)
[20:19]	Dynamic mask for link 0	Dynamic mask for link 4
[22:21]	Dynamic mask for link 1	Dynamic mask for link 5
[24:23]	Dynamic mask for link 2	Dynamic mask for link 6
[26:25]	Dynamic mask for link 3	Dynamic mask for link 7

Table B.5: Decoded event ID words from the event data decoder to the event ID FIFOs.

Bit [31:0]	Data words
Bit [35:32]	EFB ID & Gatherer ID & link number (2)
Bit [36]	Timeout error flag
Bit [37]	Not used
Bit [38]	L1ID error flag (header only)
Bit [39]	BCID error flag (header only)
Bit [40]	Not used
Bit [41]	End of event fragment (offset to next header)
Bit [42]	Non-sequential chip error flag

Table B.6: Output data format of the event ID checker.


Generic Error Field
Bit 0 = BC ID error (checked in EFB)
Bit 1 = L1 ID error (checked in FFB)
Bit 2 = FE module timeout (checked in Formatter)
Bit 3 = Data may be incorrect -> see bits[31:14]
Bit 4 = Internal buffer overflow -> See bits[15:14] -- not used
Bit 5-12 = reserved for runtime errors detected in Formatter
Bit 13 = not used
Bit 14 = reserved for ROD BUSY error
Bit 15 = reserved for ROD BUSY error
Bit 16 = Non-sequential chip error (checked in EFB)
Bit 17 = Header bit error (preamble error in header)
Bit 18 = Event synchronization error (not used)
Bit 19 = Invalid pixel row (> 336) OR column (> 80)
Bit 20 = Skipped events (checked in Formatter not used)

FE-I4 Service Record Field
Bit 21 = SR 0 = BCID counter error (requires Bunch Counter reset)
Bit 22 = SR 1, 2, 3 = Hamming code word error (initially masked)
Bit 23 = SR 4, 5, 6, 7 = L1-related error (requires L1 Counter reset)
Bit 24 = SR 8 = read out processor error
Bit 25 = SR 24 = TRM* in config memory
Bit 26 = SR 25 = write register data error
Bit 27 = SR 26 = address error
Bit 28 = SR 27 = other cmd decoder error
Bit 29 = SR 29 = TRM SEU* in cmd decoder (initially masked)
Bit 30 = SR 30 = data bus address error
Bit 31 = SR 31 = TRM in EFUSE
*TRM: triple redundant mismatch *SEU: single event upset

Table B.7: Error detector 32-bit error and field definitions.

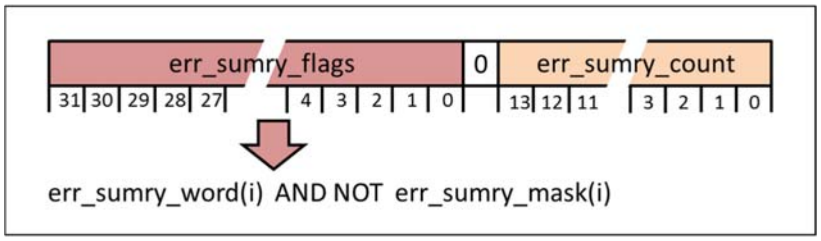
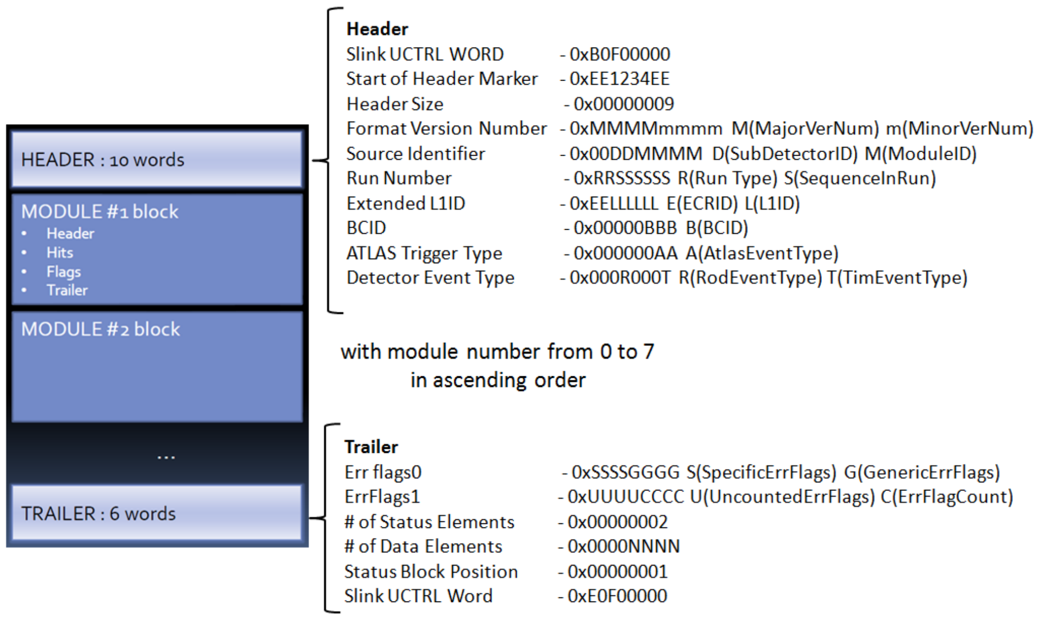


Table B.8: Format of the concatenated error summary written to the error word FIFOs.



Header

Slink UCTRL WORD	- 0xB0F00000
Start of Header Marker	- 0xEE1234EE
Header Size	- 0x00000009
Format Version Number	- 0xMMMMmmmm M(MajorVerNum) m(MinorVerNum)
Source Identifier	- 0x00DDMMMM D(SubDetectorID) M(ModuleID)
Run Number	- 0xRRSSSSSS R(Run Type) S(SequenceInRun)
Extended L1ID	- 0xEELLLLLL E(ECRID) L(L1ID)
BCID	- 0x00000BBB B(BCID)
ATLAS Trigger Type	- 0x000000AA A(AtlasEventType)
Detector Event Type	- 0x000R000T R(RodEventType) T(TimEventType)

Trailer

Err flags0	- 0xSSSSGGGG S(SpecificErrFlags) G(GenericErrFlags)
ErrFlags1	- 0xUUUUCCCC U(UncountedErrFlags) C(ErrFlagCount)
# of Status Elements	- 0x00000002
# of Data Elements	- 0x0000NNNN
Status Block Position	- 0x00000001
Slink UCTRL Word	- 0xE0F00000

Table B.9: S-Link data format (output at the EFB and Router).

B.4. FE-I4 model data formats

24-bit Record Word	Field 1	Field 2	Field 3	Field 4	Field 5
Data Header DH	11101	001	Flag	LV1ID [4:0]	bcID [9:0]
Data Record DR	Column [6:0]	Row [8:0]	ToT(1) [3:0]	TOT(2) [3:0]	
Address Record AR	11101	010	Type	Address [14:0]	
Value Record VR	11101	100	Value [15:0]		
Service Record SR	11101	111	Code [5:0]	Number [9:0]	
Empty Record ER	abcdefgh	abcdefgh	abcdefgh		

Table B.10: The six 24-bit record words output by the FE-I4 [10].

“True” ToT	**HitDiscCnfg**			
(clocks)	00	01	10	11
Below tresh	F	F	F	x
1	0	E	E	x
2	1	0	E	x
3	2	1	0	x
4	3	2	1	x
5	4	3	2	x
6	5	4	3	x
7	6	5	4	x
8	7	6	5	x
9	8	7	6	x
10	9	8	7	x
11	A	9	8	x
12	B	A	9	x
13	C	B	A	x
14	D	C	B	x
15	D	D	C	x
$\geq$16	D	D	D	x

Table B.11: ToT 4-bit code values (hex) for actual time over threshold values (left column) and hit discrimination configuration settings. Setting “11” is not valid and will result in hit output being completely disabled. The values above the dividing line in the central 3 columns will only be read out if they are next to a “big hit” – all values below the line are “big hits” [10].

Appendix C: Acronyms

ADC Analogue-to-Digital Converter
ASIC Application-Specific Integrated Circuit
ATCA Advanced Telecommunications Computing Architecture
ATLAS A Toroidal LHC ApparatuS
BC Bunch Crossing
BCR Bunch Counter Reset
BOC Back Of Crate
CERN European Organization for Nuclear Research
CMOS Complimentary Metal-Oxide Semiconductor
CTE Coefficient of Thermal Expansion
DAC Digital-to-Analogue Converter
DAQ Data acquisition system
DC Direct Current
DCS Detector Control System
DDC Double Digital Column
DDR Double-Data-Rate
DMA DirectMemory Access
DPRAM Dual-port RAM
DSP Digital Signal Processors
ECR Event Counter Reset
EFB Event Fragment Builder
EOCHL End of Chip Logic
EoS End of Stave
FE Front-End
FIFO First-in-first-out
FPGA Field-Programmable Gate Array
FSM Finite State Machine
HV High Voltage
IBL Insertable B-Layer
INFN Instituto Nazionale di Fisica Nucleare
IST IBL Support Tube
JTAG Joint Task Action Group
L1A Level-1 Trigger
LHC Large Hadron Collider
LVDS Low-Voltage Differential Signalling
LV Low Voltage
MAPS Monolithic Active Pixel Sensor
MCC Module Control Chip
PCB Printed Circuit Board
PRM Program Reset Manager
PST Pixel Support Tube
RAM Random Access Memory
RCE Reconfigurable Cluster Element
ROBIN ReadOut Buffer INterface
ROD ReadOut Driver
ROS ReadOut Sub-system
RX Receiver
SBC Single Board Computer
SCT SemiConductor Tracker
SerDes Serializer/Deserializer
SEU Single Event Upset
SSRAM Synchronous Static Random Access Memory
TDAQ Trigger and Data AcQuision

TIM	TTC Interface Module
TOT	Time Over Threshold
TT	Trigger Type
TTC	Timing, Trigger, and Control
TX	Transceiver
VHDL	Very High Speed Integrated Circuits Hardware Description Language
VME	Versa Module Eurocard