Oct 2011

Department of Physics
and Technology
University of Bergen
Norway

Arun Kumar Sharma

MASTER THESIS

DESIGNING AND DEVELOPMENT OF READOUT AND CONFIGURATION PROGRAM FOR THE MXGS–INSTRUMENT

Preface

I worked on this thesis in the Microelectronics Research Group in the Department of Physics and Technology at University of Bergen, Norway between August 2010 and October 2011.

First of all, I would like to thank my supervisor Prof. Kjetil Ullaland for his motivation, encouragement, guidance and support throughout the work on this thesis as well as my Master degree courses and to give me an opportunity to join "NI Days 2011-The LabVIEW conference" at Lillestrøm, Norway.

A lot of credits go to Georgi Genov and C. Linga Reddy for their discussions and explanations. Furthermore, I wish to thank Christian Erstad for creating the proper groundwork for my assignment. Arne Olav Solberg deserves credits for sharing his neighborhood, wisdom and knowledge with me. Thanks as well to rest of the ASIM team.

In addition, big thanks go to my fellow students Magnus Lode Roscoe and Arild Velure for their company and assistance during my master degree.

A special thank goes to my daughter Angelina Sharma, my wife, my sister-in-law and her family for encouraging and supporting me during my whole study period. I really appreciate my wife, Sunita Rani Sharma for her support and her patience. Special thanks to my sister-in-law Babita Roopan for her assistance during this period.

Finally, I wish to thank my parents along with other family members and my friends in INDIA and in NORWAY for their encouragement and support.

Bergen, September 2011.

Arun Kumar Sharma

Abstract

The Atmosphere-Space Interactive Monitor (ASIM) is a project running under the guideline and financial support of the European Space Agency (ESA). The ASIM instrument consists of several high resolution, optical cameras and an X-ray and Gamma-ray detector. The ASIM payload is to be installed on the Columbus module on International Space Station (ISS) which will orbit the Earth at an altitude of 300-400 km, and will observe the Earth's atmosphere for a time span of two years after installation. A new H-2B heavy lift rocket carrying the H-2 Transfer Vehicle (HTV) is expected to install ASIM onto the Columbus module in 2014, which is developed by the Japan Aerospace Exploration Agency (JAXA). The main goals of the ASIM project are to study the Transient Luminous Events (TLEs) and Terrestrial Gamma Flashes (TGFs) with their effects on the atmosphere and space, and to collect the data related to the Earth's atmosphere, including the climate changes. The ASIM payload consists of a Modular X-ray and Gamma-ray Sensor (MXGS), six high resolution optical cameras, six photometers and other supporting electronics. Further the MXGS instrument consists of two detector modules named the CZT and the BGO having high imaging capability and are sensitive to Photons having energy range of 10 keV to 20 MeV. The University of Bergen is working together with the Danish National Space Center (DNSC), University of Valencia, the Polish Academy of Sciences Warszawa and many other research groups on this project under the supervision of the European Space Agency (ESA).

The University of Bergen is responsible for the development of the MXGS-detector plane, Detector array and its Front End Electronics (DFEE). At the time of writing, the project is in phase C/D where a full-fledged ASIM payload engineering model for Flight Version is under development. The phase C/D will end with the installation of final Flight Version of ASIM onto the ISS. For testing the MXGS instrument in the lab at UiB, a readout and configuration program is developed in LabVIEW programming environment.

This thesis describes the designing, and development of the readout and configuration program developed for the MXGS instrument in LabVIEW environment. A LabVIEW program is developed and used to communicate with different detector modules in MXGS instrument. The user can use this program from the PC to send commands to the detector (RCU-Readout Control Unit), to configure the ASICs in the detector modules (to activate

them for event signals) and for acquisition and analysis the measurement data in the form of real-time plots visualization on the PC monitor.

Each detector assembly unit consists of a RCU (Readout Control unit). The RCU hosts a FPGA (Field Programmable Gate Array). There are number of Control and Status Registers defined in the various modules of the RCU firmware. The developed program creates an interface between the user and these CRs/ SRs to trigger the various functions in the detector. Whenever any photon event occurs on the detector, the science data packets (SCDPs) contain the photon energy and time information is created at the RCU. These SCDPs are made available for analysis on the user's PC in the form of real-time visual plots by using the LabVIEW program. A Control and Status Command Interface (CSCI) is developed, where it is much easier to send commands to the RCU and to verify the results. A separate graphical user interface is developed for ASICs configuration. Also program is made compatible with both the CZT and the BGO detectors.

Contents

Table of Figures

Chapter 1
Introduction

For almost 125 years, unexplained luminous phenomena above thunderstorms have been reported in the literature, beginning with MacKenzie and Toynbee (1886), who described what today might be regarded as a giant jet. In this long time span, many scientific missions regarding these phenomena has been launched but still there is no clear scientific evidence of their effects on the atmosphere and climate. To study such unexplained lighting phenomena, the European Space Agency (ESA) has initiated a project, named Atmosphere Space Interaction Monitor (ASIM). A detailed description including scientific goals of the ASIM project is given in this chapter. The ASIM is expected to be launched onto the International Space Station (ISS) in 2014 from where it will observe the earth's atmosphere for the next two years.

Above the altitudes of normal lightning, various types of electrical-breakdown phenomena have been observed. These upper-atmospheric lightning are termed as transient luminous events (TLEs) which refer to different types of electrical-discharging phenomena named red sprites, blue jets, gigantic jets, and elves. In addition, 1-5 ms flashes of gamma-rays were observed by the Burst and Transient Source Experiment (BATSE) instrument in 1994 when the Compton Gamma Ray Observatory (CGRO) spacecraft was above a large thunderstorm system, and these flashes are termed as Terrestrial Gamma-ray Flashes (TGFs) [2]. The more description about TLEs and TGFs are provided in next sections. This chapter is based on the information in references [1 to 25].

1.1 Transient Luminous Events (TLEs)

The Transient Luminous Events (TLEs) are high-altitude lightning flashes rising above thunderstorm clouds up to ionosphere (~ 100 km). These are short-lived lightning phenomena which generally last from less than a millisecond to more than two seconds and are not visible to the naked eye. Their characteristics are different from the characteristics of tropospheric lightning in many ways. Figure 1.1 provides an idea about the various electrical-discharging phenomena in Earth's atmosphere.

Red Sprites: Discharging of positive lightning between thunderclouds and the ground produces red sprites. The brightest region of the Red Sprites lies in the altitude range 65-75 km, but a faint red glow structure can extend to about 90 km above the Earth's surface. The Red Sprites can be reddish-orange or greenish-blue in color.

Blue Jets: Blue jets project from the top clouds above a thunderstorm, in a narrow cone to the lowest level of the ionosphere. Blue jets do not get triggered directly by lightning and are brighter than sprites and are blue in color. Blue jets occur less frequently than sprites.

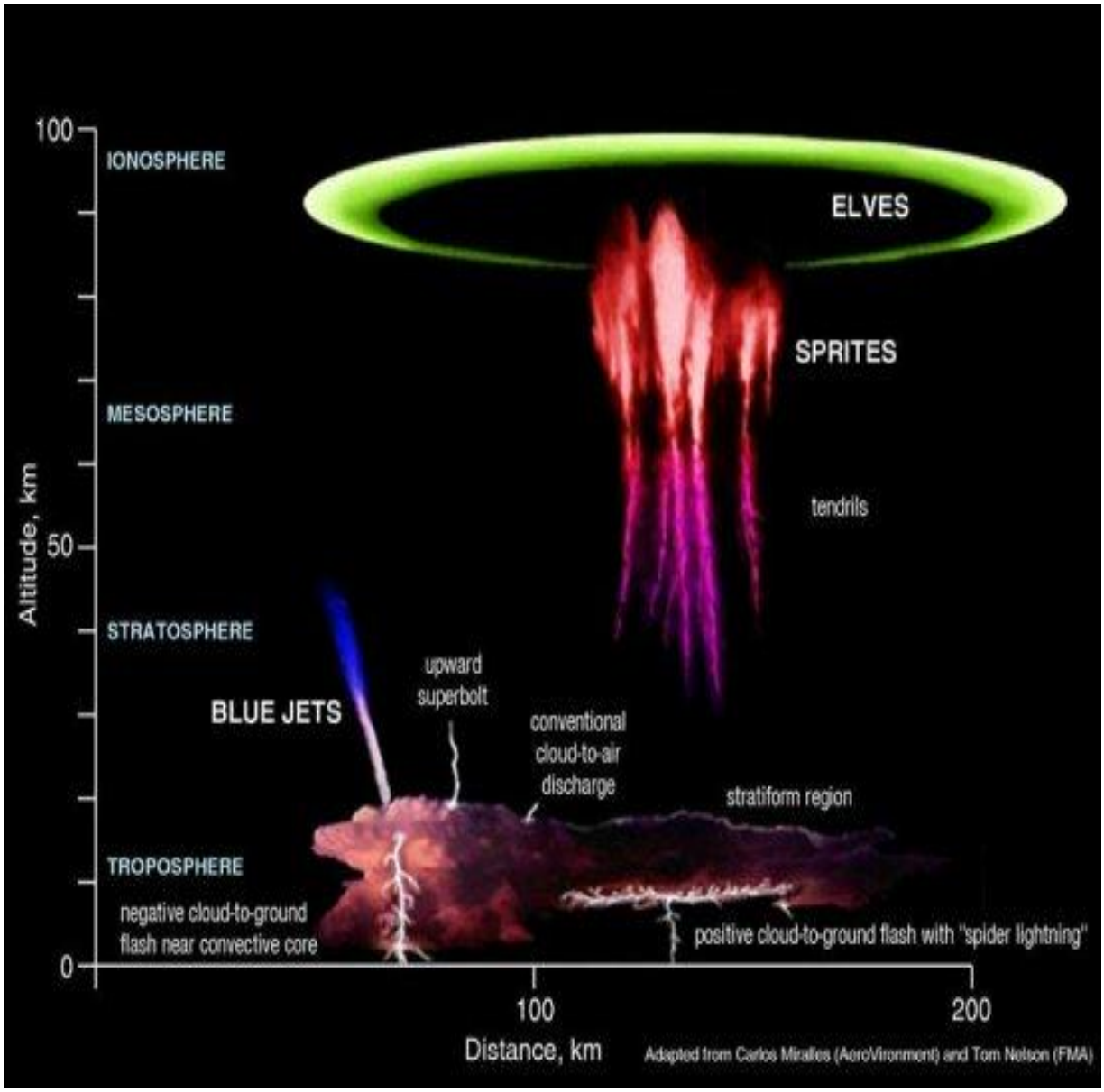


Figure 1.1: Representation of different electrical-discharging phenomena in Earth's atmosphere [3].

Gigantic jets: These jets occur at around 70 km in the atmosphere, located above a thunderstorm over an ocean, with shapes similar to giant trees and carrots and lasts for less than a second. In 2002, five gigantic jets between 60 to 70 km lengths were observed over South China Sea from Taiwan [4]. Also on July 2007, twenty blue gigantic jets were observed over a frontal system in China [5].

Elves: Elves appear as a dim, flattened glow about 400 km in diameter and lasts for one millisecond in the ionosphere above thunderstorms. The color resembles to red hue. Elves were recorded first time at *French Guiana* in 1990 [6].

1.2 Terrestrial Gamma-ray Flashes (TGFs)

The Terrestrial Gamma-ray Flashes were first discovered in 1994 by the Burst and Transient Source Experiment (BATSE) on Compton Gamma-Ray Observatory, a NASA spacecraft. These are flashes of gamma-rays in the Earth's atmosphere above the top of thunderstorms which last from 0.2 to 3.5 milliseconds having energy up to 20 MeV. The TGFs are assumed to occur when electrons which are traveling at a speed close to the speed of light collides with the nuclei of atoms in the air and release their energy in the form of gamma-rays (Bremsstrahlung) [7]. The BATSE instrument detected 75 occasions of TGFs in over eight years on its life span [2].

The newer Reuven Ramaty High Energy Solar Spectroscopic Imager (RHESSI) satellite has observed more energetic TGFs compared to the TGFs recorded by BATSE. The RHESSI was designed to provide high resolution spectroscopy of energetic photons from soft x-rays (~3 keV) to gamma-rays (up to ~20 MeV) in solar flares [8]. The RHESSI can measure TGFs accurately but still data provided by it is not enough for complete understanding of the phenomena of the TGFs and a detailed study is required in this regard.

1.3 Next Step in the same direction - The ASIM

To study TLEs and TGFs more thoroughly, the European Space Agency (ESA) has initiated a project along with the Danish National Space Center (DNSC), University of Bergen (UiB), University of Valencia, the Polish Academy of Sciences, Warszawa and some other research groups. The Danish Technical University is providing the scientific leadership and the Danish

company Terma is providing the technical leadership. The project is named as Atmosphere-Space Interaction Monitor (ASIM). The ASIM instrument consists of many high resolution optical cameras and an X-ray & Gamma-ray detector. The ASIM payload will be installed on the Columbus External Payload Facility (CEPF) in 2014 and become a part of ISS (International Space Station) [9].

A new H-2B heavy lift rocket carrying the H-2 Transfer Vehicle (HTV) developed by the Japan Aerospace Exploration Agency (JAXA) is expected to install ASIM onto the Columbus Module [9]. Figure 1.2 shows Columbus's position on the ISS.

The Columbus is a science laboratory on the ISS and is controlled by the Columbus Control Centre (CCC) in German Space Operations Centre (GSOC). The ISS orbits the Earth at an altitude of about 278-460 km and complete 15.7 orbits per day. The various sections of ISS are operated by their builders, the American National Aeronautics and Space Administration (NASA), the European Space Agency (ESA), the Russian Federal Space Agency (RKA), the Japan Aerospace Exploration Agency (JAXA) and the Canadian Space Agency (CSA) [10].

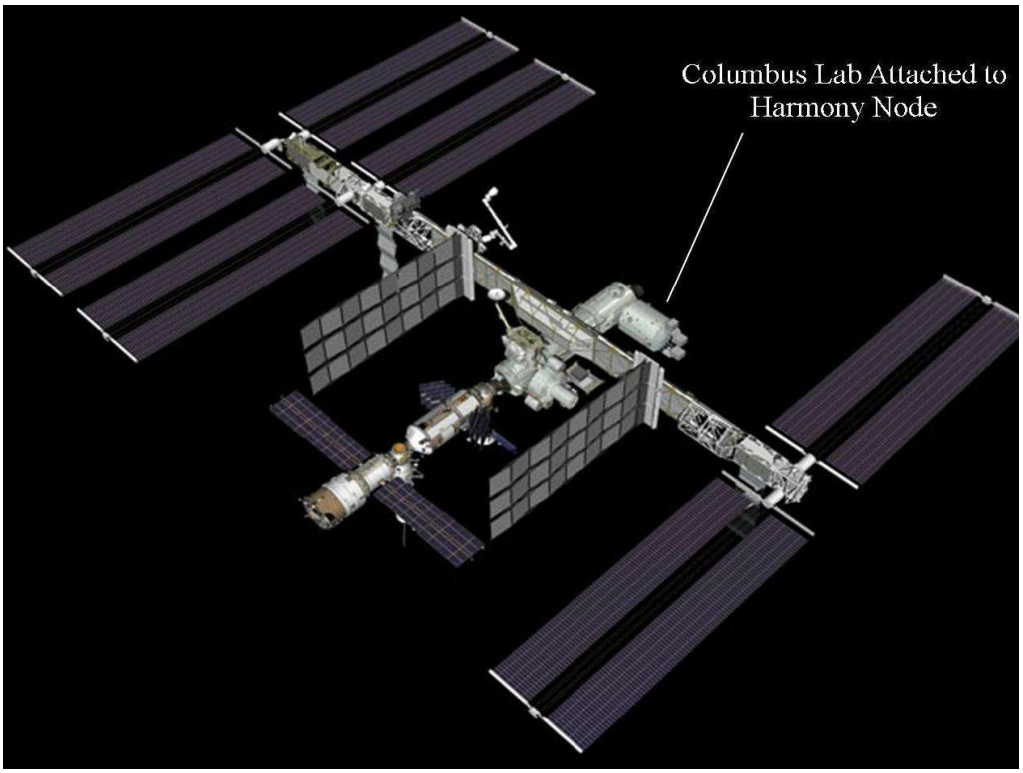


Figure 1.2: The Columbus's position on the ISS [NASA]

Figure 1.3 shows the platform where the ASIM instrument will be mounted as external payload of the ISS-Columbus Module directed towards the Earth's atmosphere.

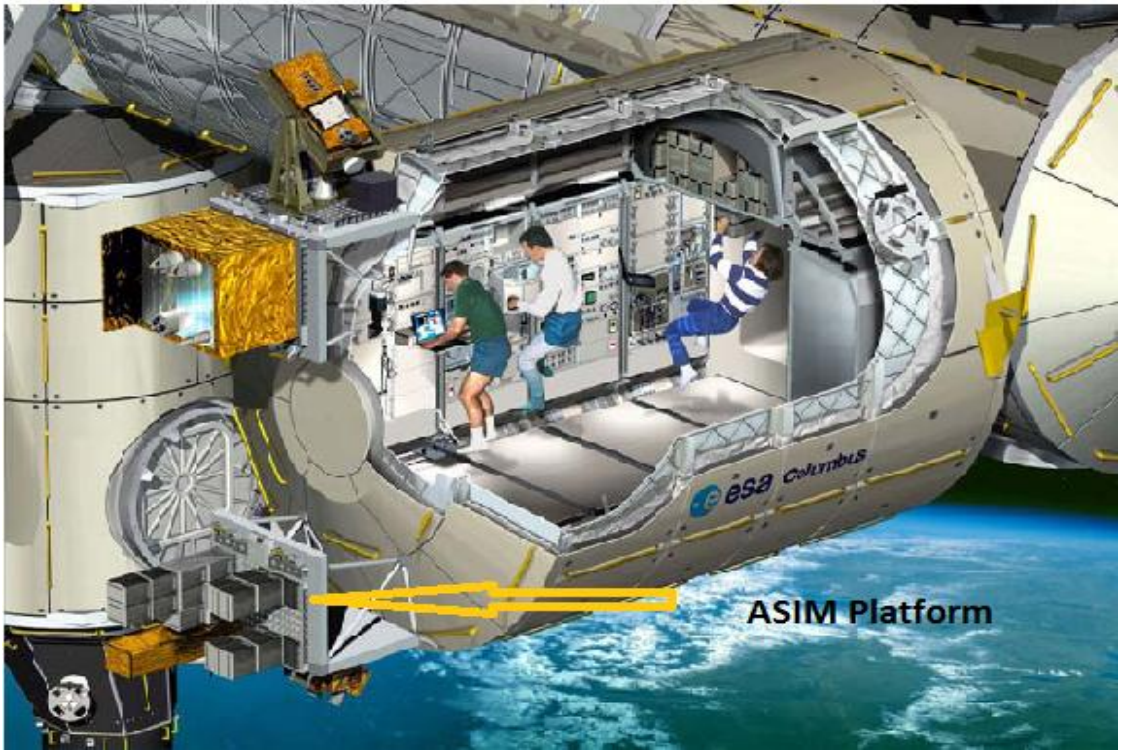


Figure 1.3: ASIM platform on the Columbus external pallet of ISS [9].

1.3.1 ASIM's goals

ASIM has several important scientific and technological goals, and some of them are:

- To understand the electrical discharging processes involved in thunderstorms and to understand the nature and origin of TLEs and TGFs. The ASIM is expected to provide more accurate data than any previous mission to understand the nature of these electrical phenomena.
- To understand the effect of these processes on the earth's atmosphere and its relation with climate changes. As these discharges inject water vapours, NOx and other greenhouse gases into the stratosphere, the goal is to study their impact on the climate. Along with these effects, ASIM will study the electrical influence on the ionosphere.
- To study more accurately and precisely the Aurora spectroscopy.
- To monitor the space processes and their impact on atmosphere such as meteors ablation, NO from solar radiations etc. especially in mesosphere and thermosphere.

1.3.2 Project's Plan-Phases

The ASIM project is divided into four phases. The first two phases has been successfully completed and approved by the ESA. Now (in 2011) the project is in the preliminary stage of third/fourth phase and currently under the development of the payload Engineering Model for Flight-Version. Phase-A included all the theoretical planning about the designing and development, which was completed during 2004-05. In Phase-B, a breadboard payload design was developed and checked and completed during 2007-09. In 2009, the Program Board for Human Spaceflight and Exploration (PB-HME) has approved the ASIM design. Phases-C/D was started in 2010. The developing stage of phase-C includes the development of electrical engineering related to ASIM instrument and in phase-D, the final Flight Module is going to be built. The Phase C/D will be finished with the installation of ASIM instrument onto ISS and is expected to be launched by the end of 2013 or in the beginning of 2014.

1.4 The ASIM Instrument

The technical information about the ASIM instrument is taken from sources [13, 15, 16, 17, 22, 23, 24 and 25].

The ASIM scientific instrument includes an X-ray and Gamma-ray detector, six cameras and six photometers with other supporting electronics. The instrument's observatory is divided into two parts. The first one towards horizon direction to the Earth called Limb viewing assembly and the second is directed down toward the Earth called Nadir viewing assembly. Figure 1.4 gives an idea about Limb and Nadir position to the Earth.

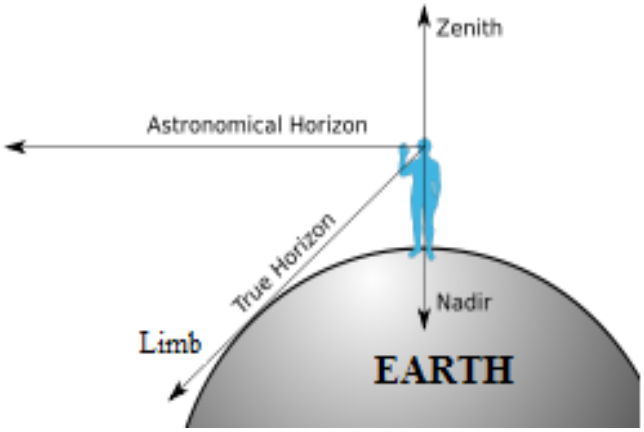


Figure 1.4: Representation of Limb and Nadir position [11].

Furthermore, the instrument is divided into two parts i.e. the Modular Multispectral Imaging Array (MMIA) and the Modular X-ray and Gamma-ray Sensor (MXGS). An overview of the ASIM instrument is shown in the figure 1.5.

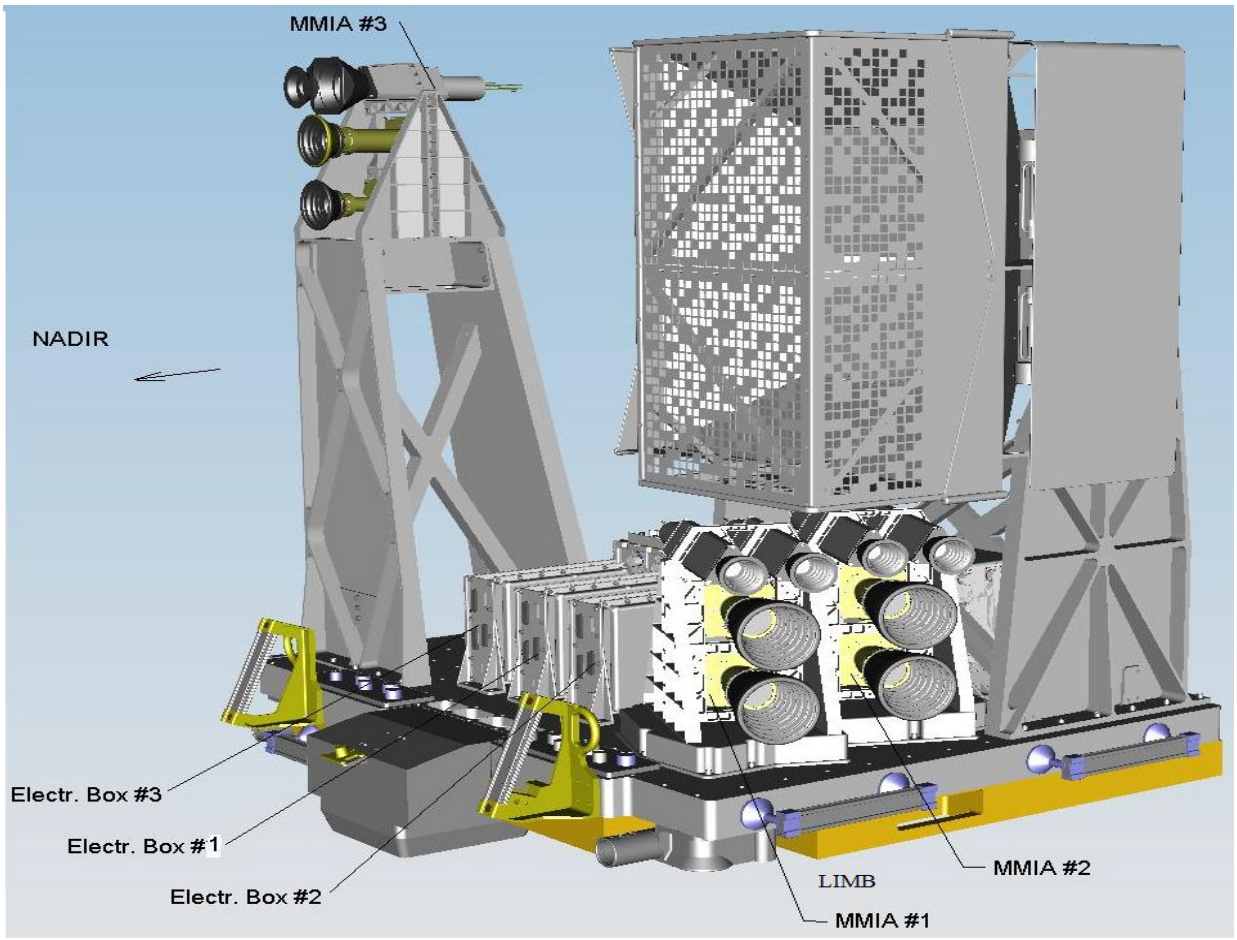


Figure 1.5: The position and direction of the MMIA units on CEPA plate [12].

1.4.1 The Modular Multispectral Imaging Array (MMIA)

The two optical cameras and two photometers constitute one MMIA module. There are three MMIA-modules available on the ASIM instrument, two pointing toward *Limb* or horizon direction and one pointing downward toward Earth or *Nadir*.

All the MMIA-modules can observe in different optical spectral bands. The *limb* viewing cameras can observe in the spectral band of 337 nm, 391.4 nm, 650-800 nm and 762 nm, whereas *nadir* viewing cameras have spectral bands 337 nm and 650-800 nm. All the cameras have resolution of 1024x1024 pixels. The cameras are light sensitive, so they can only observe in the night time or at sunset or sunrise. The cameras are made with new CCD on-

chip amplification without the use of intensifiers. This means that CCDs will not be damaged even the cameras view the direct sun or the moon by mistake. The photometer records the light intensity levels and transmits them to the DPU for further processing. The rapid time variation is measured by Photometers. The optical photometers view the exact same region as optical cameras but have high time resolution (~10 microseconds). Figure 1.6 shows one MMIA module.

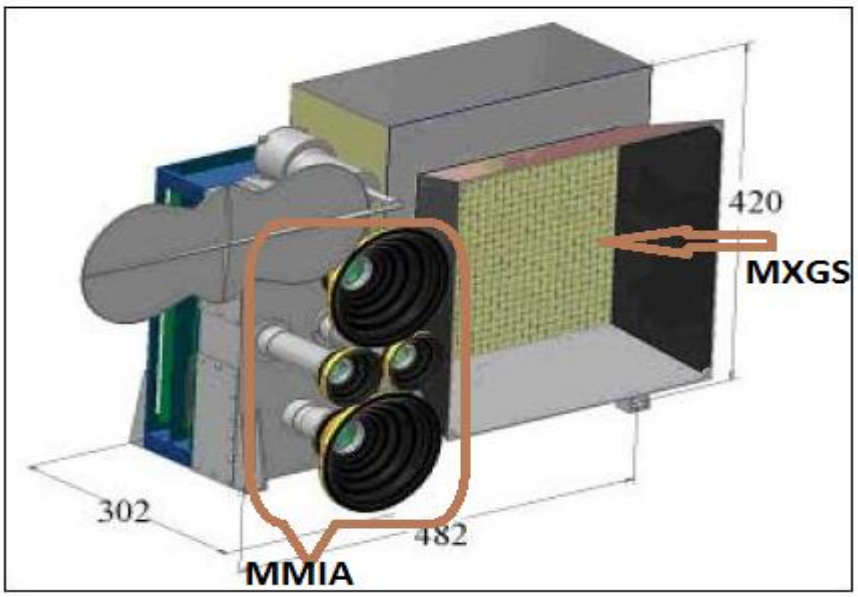


Figure 1.6: The nadir-viewing assembly with one MMIA and MXGS detector [13].

1.4.2 The Modular X-ray and Gamma-ray Sensor (MXGS)

The X-ray and Gamma-ray detector is called Modular X-ray and Gamma ray Sensor (MXGS). The MXGS is designed to detect "*Bremsstrahlung*" (the electromagnetic radiation produced by the deceleration of the charged particle like an electron when deflected by another charged particle such as an atomic nucleus) produced from TGFs and lightning. The detector instrument is built from two detector layer made of Cadmium Zinc Telluride (CZT) semiconductor and the Bismuth Germanate ($Bi_4Ge_3O_{12}$) crystal. The two detector layers of the MXGS covering a large energy range of 10 keV to 20 MeV. The MXGS detector measures each photon and determines its energy and time of arrival. The position of the MXGS-instrument on the ASIM payload is toward nadir.

- **The CZT Detector**: The first layer in the MXGS instrument is 0.5 cm thick and 1048 cm^2 in area made of CZT semiconductor, which can detect the X-rays and gamma-rays in the energy range of 10 keV to 500 keV. The CZT detector is transparent to radiations with greater energy range. The CZT detector is made of four Detector Assembly Units (DAUs). Each DAU consists of 16 Detector Modules (DM), and each DM is built with two XA-ASICs (Application Specific Integrated Circuits) with corresponding detector modules mounted on them. The flight version of CZT detector will contain 64 DMs having 128 XA-ASICs with altogether 16384 pixels. The detailed information about the DAU, DM and XA-ASICs is provided in chapter two.
- **The BGO Detector**: The second layer in the MXGS instrument is 3.2 cm thick and 900 cm^2 in area made of BGO crystal, which can detect the X-rays and gamma-rays from 200 keV to 20 MeV energy range. The BGO layer is consisted of four assembly units each with three BGO crystals with corresponding photomultiplier tubes (PMTs) and readout electronics. The BGO layer with 12 BGO-PMTs will have their own mechanical support, communicational link and supply inputs, which are placed under the CZT layer.

A hopper shaped collimator with 80^o x 80^o field of view, shields the detector plane against the Cosmic X-ray background radiations and reduce the noise signal in the detector. The Detector Front End Electronics (DFEE) is mounted under the detector. The BGO detector layer is placed under the CZT detector layer with BGO-DFEE.

The other electronics also contains the Data Processing Unit (DPU), the High-Voltage Power Supply (HVPS) and Low Voltage Power Supply (LVPS). The CZT detector is powered with 600 V negative bias voltage, and the BGO photomultiplier tubes (PMTs) are driven at approximately 1000 V.

The TGFs have a typical life expectancy of 1-5 ms. With the BGO extension in the MXGS, the sensitivity of the instrument is highly increased and it is expected to see about 1000 TGFs with each TGF have in average 300 counts. The expected observed TGFs will be 10 times more than RHESSI and 100 times more than BATSE detection. Figure 1.7 gives an overview of the MXGS assembly exploration.

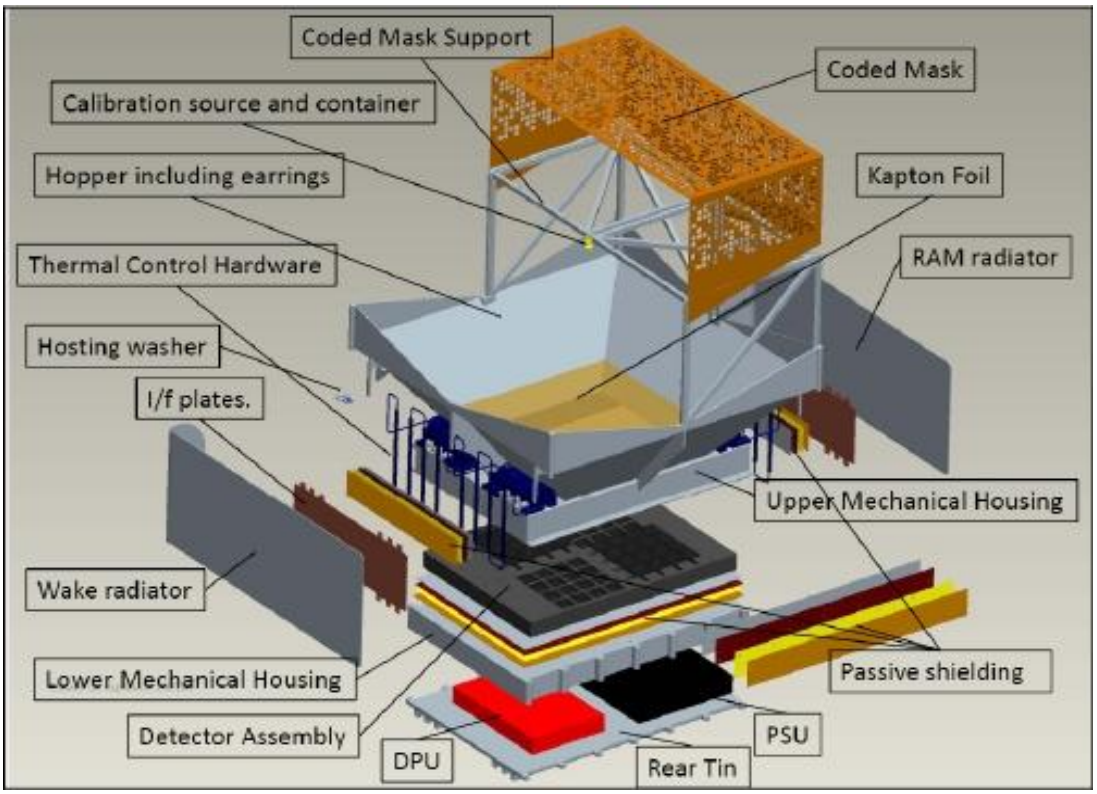


Figure 1.7: The MXGS full assembly exploded view [14].

1.5 About this work

At UiB, it is the Space Physics group working together with Microelectronics group who develops the ASIM-MXGS detector plane and its read-out electronics. The project is in phase-C/D where a fully integrated, qualified and accepted ASIM payload Flight Model together with all supporting hardware and software is going to develop. The research work on development of electrical engineering modules, preparation for flight version model of the MXGS-CZT and extension of the MXGS detector with BGO is under development. I continued the work of master student Christan Erstad, who made a program to configure the electronics and gather measurement data from the MXGS detector in LabVIEW programming environment. The observation and configuration LabVIEW program is developed for testing the MXGS instrument in Lab at the University of Bergen.

My assignment was related with the LabVIEW program developed for readout and configuration of the MXGS instrument. The first task was to provide a full description of the developed LabVIEW program, as there is no document available so far about the existing LabVIEW program. The second task was to improve the existing program mainly by

designing a new, more user-friendly Control and Status Command Interface (CSCI) and also make it compatible with the second detector, the BGO extension.

The purpose of the CSCI is to access the Control Registers (CRs) and Status Registers (SRs) in the RCU firmware. These registers are used to trigger firmware functions, define operation modes and control the functioning of the detector. So the aim was to design a CSCI interface, from where the user can access these CRs and SRs just by clicking some buttons in en easier and effective way.

To fulfill the requirements, I started with learning the LabVIEW programming tools. Then I started learning about the ASIM project and the architectural implementations and functioning of previously developed LabVIEW program. The next step was to create documentations of the previously developed LabVIEW program and in the final step, improvements are implemented in the existing LabVIEW program in the form of a new, more user friendly Control and Status Command Interface (CSCI), which also provide some new features and functions to the user. The new developed program is made compatible with both the CZT and the BGO detectors in the MXGS instrument.

1.6 Contents

This thesis describes the development of the tools for the observation and configuration of the MXGS detector and its readout electronics in LabVIEW programming environment. It describes the construction, functioning, improvements and their implementations and verifications of the LabVIEW program.

Chapter 2 provides an overall system overview of the MXGS-CZT detector and its components.

Chapter 3 deals with the DPU and the RCU firmware and functioning of the firmware. The detail about the CZT-RCU firmware and the BGO-RCU firmware is provided in this chapter and in Appendix-B.

Chapter 4 provides an introduction with the graphical programming language LabVIEW which is selected and used to develop a program for controlling and observing the functionality of the MXGS-instrument in the lab at the University of Bergen.

Chapter 5 describes the tasks and full architectural & functional development of the developed LabVIEW program which communicates and controls the MXGS-instrument.

Chapter 6 describes the new improvements made in the LabVIEW program which include the functioning and a full architectural description of the improvements. The compatibility of the LabVIEW program with the BGO detector and the new CSCI interface with its new features and function are described in this chapter.

Chapter 7 describes the functional verifications of the developed LabVIEW program. It describes the testing of the program with the CZT-DAU (Detector Assembly Unit).

Chapter 8 describes the conclusion and outlook of this thesis.

Appendix-A describes the command information related with different control buttons used on both the CSCI interfaces.

Appendix-B contains the list of CZT-RCU and BGO-RCU firmware modules.

Appendix-C shows the LabVIEW program-code for different loops in the main, developed observation and configuration LabVIEW program.

Appendix-D describes the manual of improved observation and configuration LabVIEW program.

Acronyms provide the list of used acronyms in this thesis.

Bibliography contains the list of references used in this thesis.

Chapter 2

System Overview

This chapter provides a brief introduction to the MXGS (Modular X-ray and Gamma-ray Sensor) and its components. The MXGS is a high-efficiency X-ray and Gamma-ray detector. It can detect radiation in the energy range of 10 keV to 20 MeV. The MXGS detector has two detector layers. The first layer is made up of CZT (Cadmium Zinc Telluride) semiconductor with a thickness of 0.5 cm and an area of 32x32 cm^2. It can detect radiation in the energy range of 10 keV to 500 keV. The second layer on the MXGS detector is made up of BGO (Bismuth Germanate) crystal with a thickness of 3.2 cm and an area of 900 cm^2. The BGO can detect radiation in the energy range of 200 keV to 20 MeV [22].

2.1 MXGS-DFEE

According to the references [14, 16, 17, 19, and 22], the MXGS-Detector Front End Electronics (DFEE) design is made of several modules.

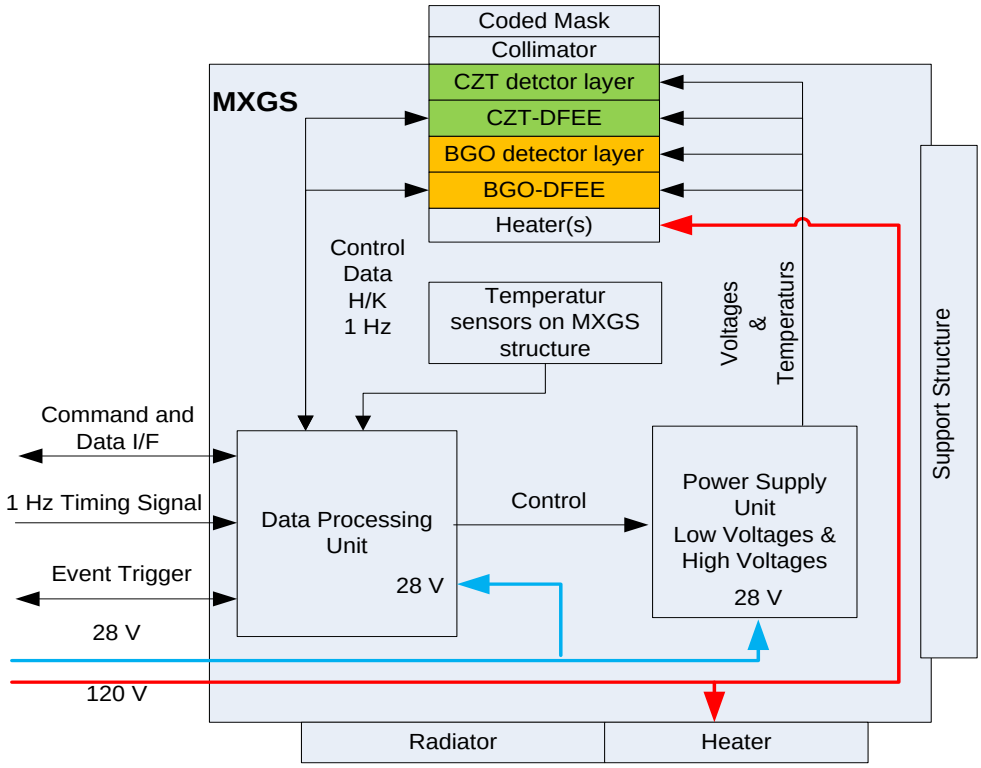


Figure 2.1: The MXGS system overview [16].

The brief introduction of these modules is given in this section. Figure 2.1 shows an overview of the MXGS system. The MXGS-DFEE is made of 4 Detector Assembly Unit (DAU), with each DAU have 16 Detector Modules (DM) and one Detector Assembly Board (DAB). The DAB holds the read-out electronics and the Readout Control Unit (RCU). The detailed information about these components is given in the sub-sections.

The 64 Detector Modules (DM) are combined together to build ***Detector Array*** of MXGS. These 64 DMs are fixed on 4 Detector Assembly Boards (DAB). A DAB holds the read-out electronics, Readout Control Unit (RCU) and connector for external interface to DPU (Data Processing Unit). The 16 DMs together with a combined thermal and mechanical structure and one DAB constituted a DAU as shown in figure 2.2. The main function of the DAU is to read out the events and transfer the data to the DPU. The mechanical support structure for the CZT module supports the CZT substrate and also working as a common cooling plate for the ASIC modules. Each ASIC module is thermally coupled to the structural and thermal plate so the heat from the thermal plate is transfer to the MXGS main structure.

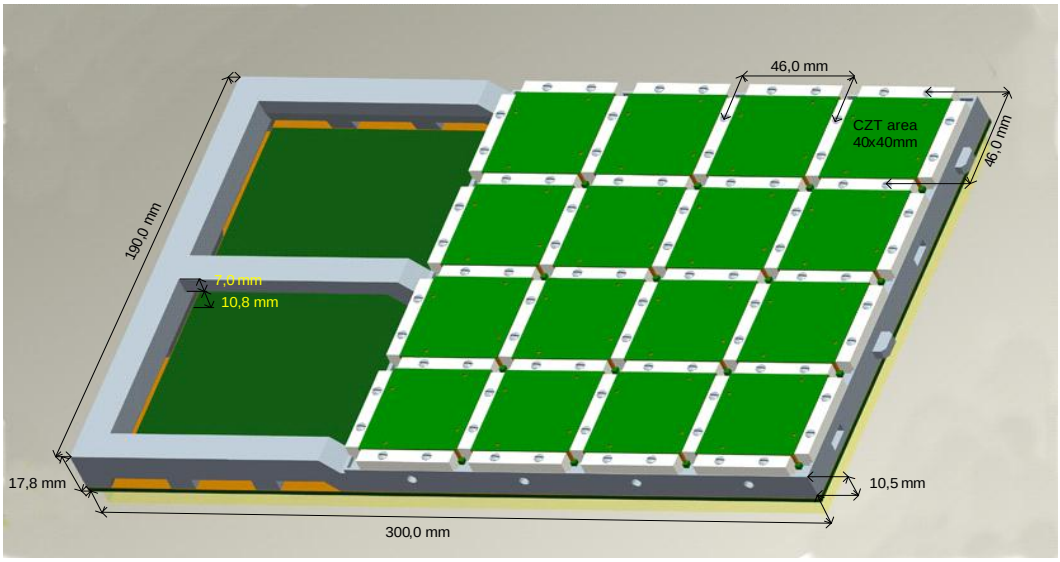


Figure 2.2: Structural view of single DAU with 16 Detector modules (DM) [14].

The RCU is interconnected with DPU and provide measurement data observed by MXGS detector to the user. All the four DAUs are framed in a single frame called the Detector Assembly Frame (DAF). Figure 2.3 shows how the 4 DAUs are assembled to construct a DAF. Each DAU gives out four read-out chains each with four DMs.

The final Flight-Version of MXGS–CZT instrument will contain 128 XA-ASICs with 16384 pixels altogether.

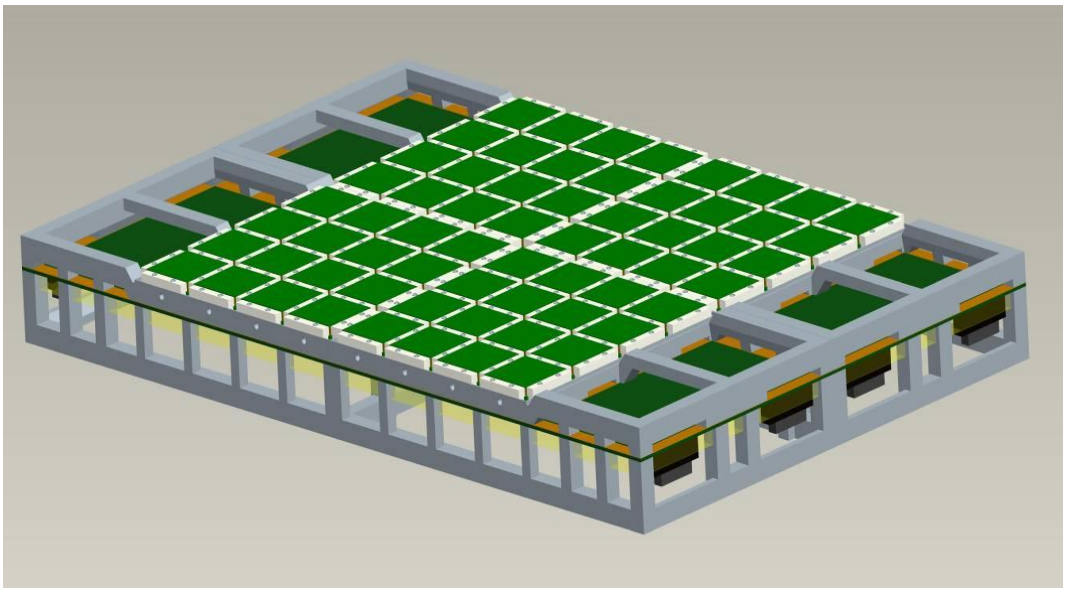


Figure 2.3: Four DAUs assembled on the Detector Assembly Frame (DAF) [14].

2.2 Detector Assembly Unit (DAU)

As mentioned above, a DAU is consisting of 16 DMs which create four DM readout chains and one Detector Assembly Board (DAB) having one RCU and its readout electronics. Each DM readout chain provides three output signals to the RCU as shown in figure 2.4.

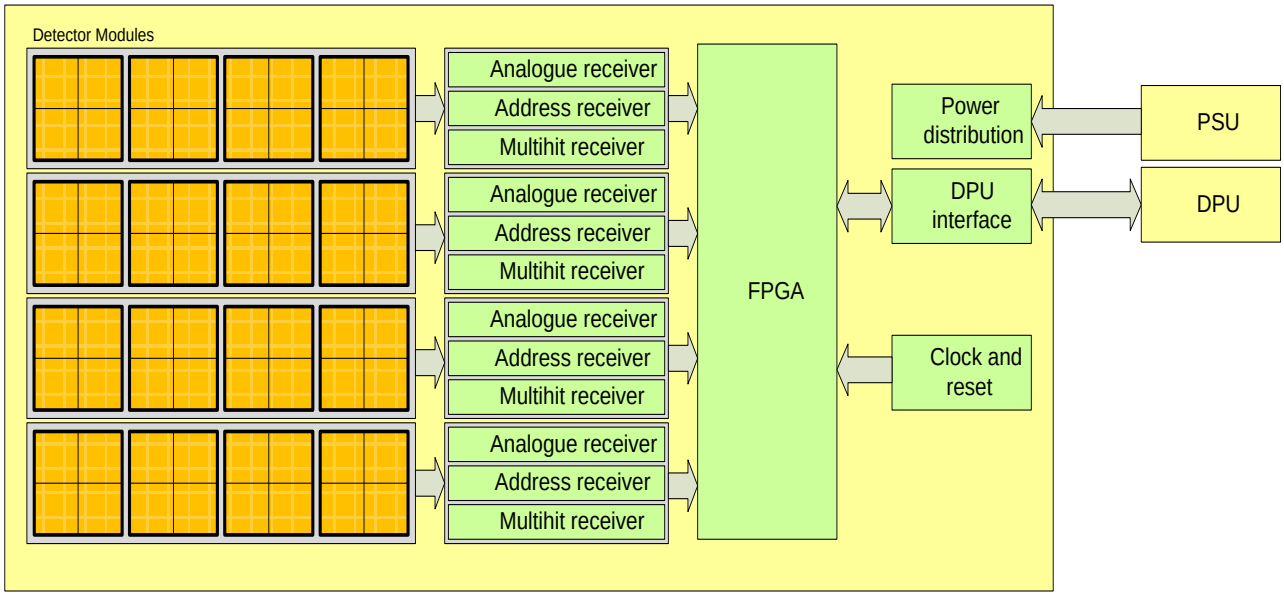


Figure 2.4: An overview of the detector assembly unit (DAU) [16].

These three signals consist of information about energy of photon, pixel address and multihit event data respectively. The event data packages are created in the FPGA and are further sent to the DPU through the RCU-DPU interface. The MXGS detector has four DAUs each having their own RCU and 16 DMs. Each DAU works in a similar way. The figure 2.4 gives an idea about the functioning of components on a DAU.

2.2.1 Detector Module (DM)

A Detector Module is made up from two components. The first component is the ASIC Module which is made of two ASIC chips, and the second component is the CZT module made of the Cadmium Zink Telluride-CdZnTe (CZT) Crystal. When any photon hit the CZT module, it converts the photon energy into electrical signals, and the ASIC Module amplifies shapes and stretches the analog signal and sends the result to the RCU for further processing. Furthermore, the ASIC Module generates the pixel value and the address of that event. A trigger signal is generated if the amplitude of the observed analog signal reaches the threshold value which indicates the detection of an event. The amplitude of the generated analog signal is proportional to energy of the photons. Further, the analog signal is converted into digital signal by ADCs (Analog to Digital Converter) and sends to the RCU. The XA 1.82 ASICs designed by Gamma Medica-Ideas, are used in MXGS instrument. The XA 1.82 ASIC chips are used before in medical radiation-cameras and now also using in space applications [26].

The CZT (Cadmium Zink Telluride) module is a semiconductor type of detector, which can operate at room temperature unlike some other materials who require liquid nitrogen cooling and works like a diode with a rectifying junction. This detector is transparent for all the radiations with energy range greater than 500 keV. To deal with radiations having energy range greater than 500 keV, a Bismuth-Germanate (BGO) detector, which is a scintillation detector, is mounted under CZT detector as an extension in the MXGS instrument.

2.2.2 XA-ASIC

The XA 1.82 ASIC (Application-Specific Integrated Circuit) manufactured by Gamma Medica-Ideas is selected for the MXGS instrument. The XA 1.82 ASIC is a fully data-driven signal acquisition chip which means no external trigger or reset is required during a data acquisition cycle [27]. Each XA-ASIC chip delivers energy and pixel address data of 128 channels. In the MXGS instrument, 128 ASIC chips are going to use with 128 channels each, which are 16348 channels or16348 pixels in total.

Each channel has a pre-amplifier circuit followed by a pulse shaping circuit, followed by peak hold circuit and followed by the trigger generation process as shown in figure 2.5. Whenever any event is observed at CZT, a pulse is collected at a channel input of the ASIC. The ASIC channel input is filtered by preamplifier & shaper and then send to the peak-holder. The peak-hold holds the analog output until the channel trigger fires which reduce the data rate and power consumption. The discriminator circuit generates a controlled trigger signal. Each ASIC channel produce one analog output and one trigger signal for further processing.

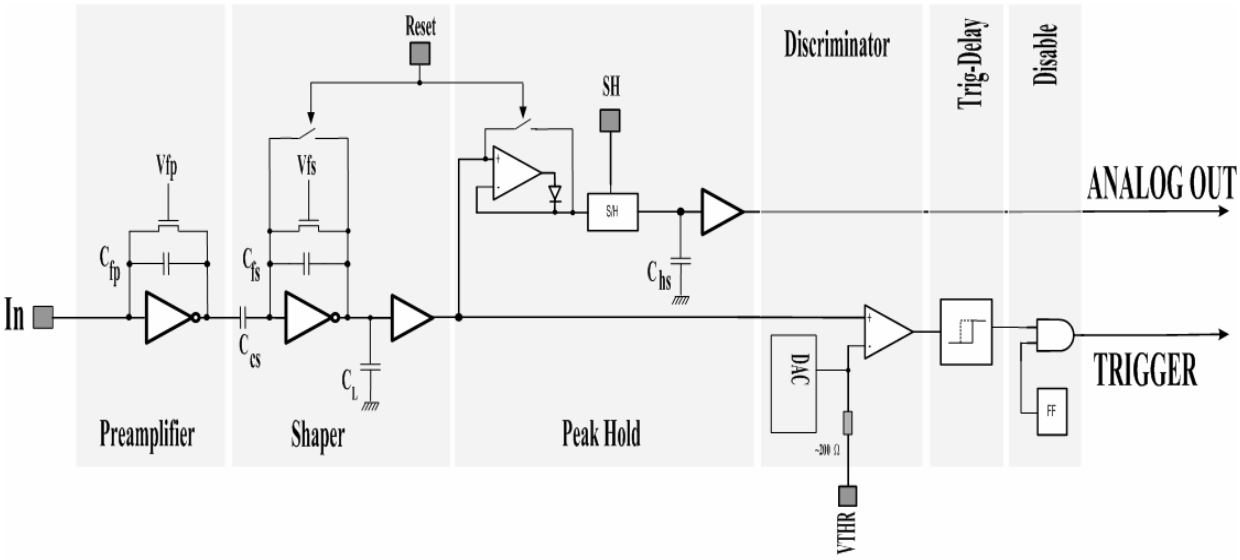


Figure 2.5: The internal architecture of a channel in the XA-ASIC [16].

There are also some control register containing bits for calibration and configuration of the ASIC chips. The configuration of ASICs is necessary to activate them for processing. The Register file must be download each time after the system is powered up because of volatile nature of the configuration registers.

2.2.3 Readout Control Unit (RCU)

The main function of the Readout Control Unit (RCU) is to provide a communication interface between the detector and the user or in more specific terms, between the DMs and the DPU. Each RCU hold one FPGA which is a RTAX2000 for the flight-version design made by the Actel Corporation. This FPGA contains 2-million system gates and has 288 kbits of embedded SRAM. The FPGA performs functions like monitoring the detector module for events, control the ADCs, create Science Data Packets whenever any event is observed,

configuration of XA-ASICs via serial links and support the memory read, and memory write command to provide status information to the DPU/User. The RCU handles all the outputs from 32 ASICs having event data such as photon energy, pixel address, time tag and multihit flag as well as collection of house-keeping data. The RCU internal memory is consisted of several modules, which are discussed in chapter 3. The entire internal memory of RCU is readable and writable by using Memory Read and Memory Write commands.

2.3 The DPU Emulator

The main function of the DPU is to create an interface between the user & the detector and provide collected observational data to the user. This is possible by sending commands to the RCU to configure the ASICs and other modules on the detector. The DPU provides overall control of the MXGS to the user, which involves acquisition and processing of data and monitoring functions of the MXGS. The User controlling the MXGS from his PC will send commands to the DPU for configuration and synchronization of detector modules and in return, he gets Science Data Packets (SCDP) along with the housekeeping data. The DPU sends information in the form of Science Data Packets on demand.

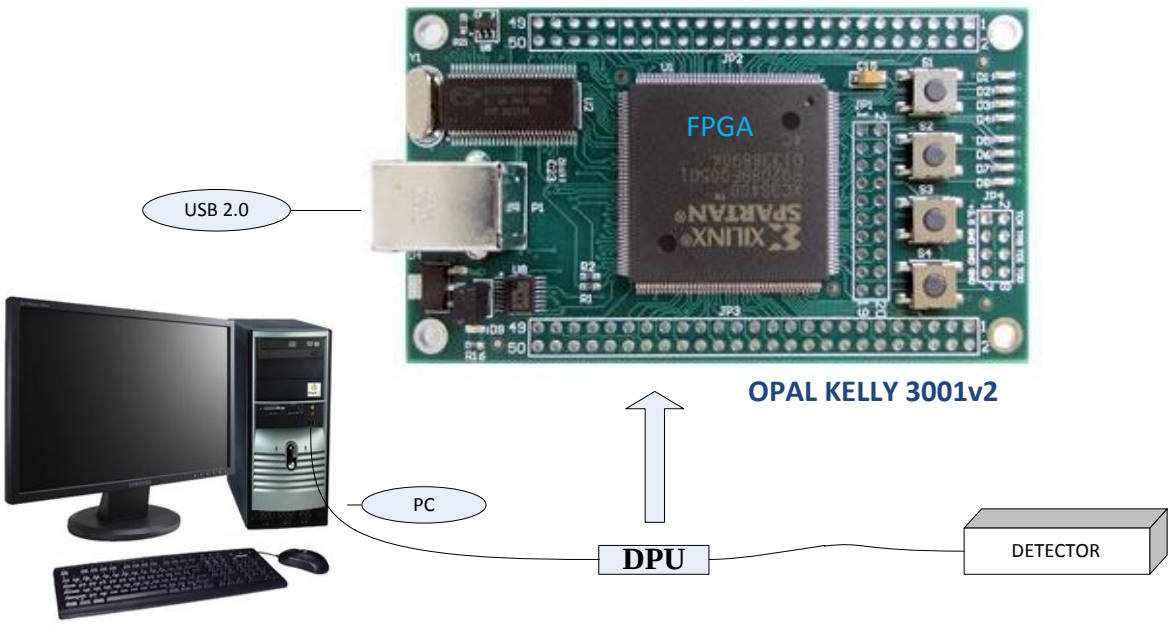


Figure 2.6: A block diagram overview of the DPU emulator [Opal Kelly] (edited).

For testing purpose at UiB, an Opal Kelly XEM3001v2 card having a FPGA and an USB 2.0 interface as shown in figure 2.6, is selected and used as a DPU Emulator. The XEM card is connected to a computer running the DPU interface emulator software one side and with the

detector on another side. The firmware on the XEM card is written in VHDL, which consists of three parts. The parts are the transmitter module, the receiver module and the FIFO.

1. The ***Transmitter Module*** consists of a 48 bit shift register controlled by a finite state machine (FSM). The main task of this module is to send commands to the RCU.

2. The ***Receiver Module*** is also consisting of a 48 bit shift register controlled by a FSM. This module handles the readout data from the RCU.

3. The asynchronous ***FIFO*** bundles the outgoing data from the receiver module for the better transmission to the computer. It can hold 2048 data packets. When the FIFO became full, a trigger signal is released, and the data is transferred to the computer disk and again FIFO become ready for next cycle.

There is a two-way link between the DPU and the RCU which means users can send commands by PC-DPU-RCU chain and receives SCDP and housekeeping data via RCU-DPU-PC chain. There are following four types of data packets sent between the DPU and the RCU. The data packets always have two flag bits in the data to identify the type of data [16].

1. *Science Data Packets* (SCDP) is the 48-bit data packet sent from the RCU to the DPU whenever an event is observed by the detector.
2. *Memory Dump Data Packets* (MDDP) is the 24-bit data packets sent from the RCU to the DPU as a response to memory read command.
3. *Memory Read Command (*MRC) is sent from the DPU to the RCU as a request to download memory data from a specific location. The MRC consists of 24-bit.
4. *Memory Write Command* (MWC) is used to control the RCU by writing data at any location in the RCU memory. The MWC is also the 24-bit package. The MWC triggers functions in the RCU to fulfill any requirement as well as used for debugging purpose.

For the MXGS instrument, the University of Valencia is going to develop a fully functional DPU. However, for testing the detector electronics at University of Bergen, the XEM3001v2 card with FPGA developed by OPEL KELLY is used as DPU emulator, which is assumed to be behaving in same way as that of real DPU.

2.4 The Power Supply Unit (PSU)

For the functioning of the MXGS instrument, two types of power supplies are required and used. A High-Voltage Power Supply (HVPS) for the CZT modules is needed and low voltage is supplied to all the electronic components on the MXGS instrument by a Low Voltage Power Supply (LVPS). A set of LV and HV converters with related housekeeping and controlling electronics makes a PSU. The PSU is commanded by the DPU. The PSU delivers the power to four DAUs and to the DPU.

The Low Voltage Power Supply is used for the functioning of ASICs, ADCs, FPGAs, memories, connecting interfaces and in DPU. The LVPS consists of the input EMC filter and Slow Start System, which protect the primary satellite power lines. The LVPS also includes the interface to DPU and common (for LVPS and HVPS) HK system, both controlled by "state machine" implemented in FPGA. The HK system monitors the main voltages, currents and temperatures within the PSU. The LVPS separates the supply of different DAU modules and provides a protection to the MXGS against any failure in detector modules.

The result shows that the XA-ASIC power consumption is approximately 1.14 mW per channel. The estimated total DFEE power consumption is 33W with XA 1.81 ASIC [19].

The CZT detector needs to be biased by a programmable negative voltage of -600 V. The negative biased voltage is generated by using a negative high-voltage converter. On the other hand, the BGO detector is going to provide with approximately 1000 V voltage for its proper functioning. The power supplies are controlled by the DPU.

Chapter 3

Firmware

This chapter provides a brief introduction to the DPU (Data Processing Unit) and the RCU (Readout Control Unit) firmware. The firmware is implemented onto the FPGA (Field-Programmable Gate Array) by programming small static RAM cells inside the FPGA to customize the routing and logic functions. The operational functions of the FPGA are written in HDL (Hardware Description Language). The FPGA then takes on the behavior of the firmware and stay in the same way until either the new firmware is loaded or the power is lost (SRAM based FPGAs). The information about the data type and data transmission between the RCU to the DPU and from the DPU to the RCU is provided in the chapter. This chapter is based on the references [16, 17, 19, 28 and 29].

3.1 Interface toward computer-Front Panel

The information in this section is mostly based on [28 and 29] and is essential to understand the operations of XEM card which is used as the DPU in the design.

For communication between the MXGS-DFEE (MXGS-Detector Front End Electronics) and the PC, a FPGA card XEM3001v2 which is made by Opal Kelly is used as a Data Processing Unit (DPU) which makes an interface between the detector and the user. The XEM3001 is a small FPGA board with Spartan-3 FPGA developed by Xilinx. The card can communicate by USB 2.0 with the help of Cypress CY68013 FX2 USB microcontroller mounted on the card. By USB peripheral it became a plug and play device which is easy to use. The FPGA on the card makes data transfer much faster than the parallel port interfaces. The XEM3001v2 is also equipped with a small serial EEPROM (electrically erasable programmable read-only memory) attached with the USB microcontroller, Cypress CY22150 PLL (Phase-locked loop) and some LEDs and pushbuttons.

The EEPROM is used to store boot code for microcontroller as well as PLL configuration data and device identifier string. The Cypress CY22150 PLL is a multi-output PLL that can provide up to five clocks, three to FPGA and another two to expansion connectors. The LEDs and push buttons are used for debugging inputs and outputs.

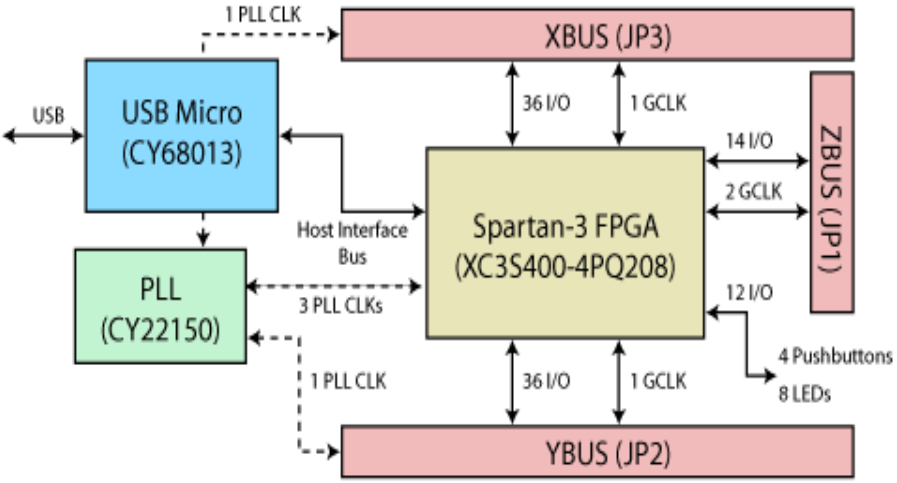


Figure 3.1: Functional block diagram of XEM3001v2 [29].

To make FPGA integration easier, productive, powerful and more configurable, the XEM3001 card is provided with a software platform named **FrontPanel**. The Front-Panel provides the basic functionality to configure and to interface with hardware including FPGA. The main purpose of the Front-Panel is to move data between PC and the FPGA in a convenient and effective way.

The USB interface in the XEM3001 can set and get signals via entities in the design. These entities are called ***Endpoints*** and are described below**.**

3.1.1 Endpoints

An endpoint is a bundle of interconnection internal to the design that transports data to or from the PC. There are three types of endpoints, i.e. Wire, Trigger and Pipe. These are directed in or out of the design. Each endpoint has an associated address so that it can be accessed easily and independent of other endpoints.

- **Wires: -** A wire is an asynchronous connection between the PC and a HDL endpoint. These are designed to fill the position of the devices such as LEDs, hexadecimal displays, and pushbuttons and so on. These are asynchronous with design and only send the current state of some internal signals. The Wires can update periodically. All wires are captured and updated simultaneously in order to avoid multiple transfers over the bus. The Wires are 16-bits wide on USB devices.
- **Triggers:** - The Triggers are synchronous connection between the PC and an HDL endpoint. Triggers are used to initiate a single event like start or end of state machine.

A Trigger In i.e. input to HDL, creates a signal that is asserted for a single clock cycle. A Trigger Out i.e. output from HDL, triggers the PC when rising edge of any signal is detected. Triggers are 16-bits wide on USB devices.

- **Pipes:** - Pipes are synchronous connection between the Front-Panel and the HDL endpoint. They can transmit bulk signals i.e. a series of bytes and mostly used to download or upload memory contents or stream data to and from the device. Pipes are the fastest way to transmit or receive bulk data. It is suggested to use FIFO if pipe transaction is performing at clock rate over 48MHz in case of USB. Pipe data transfer on FPGA side (HDL) is 16-bit word width but on PC side it has 8-bit word width.

Figure 3.2 shows the communicational functionality provided by the Front-Panel HDL modules which is implemented on the FPGA. In FPGA towards the design, there are many different endpoints are used to connect Front-Panel components with the signals.

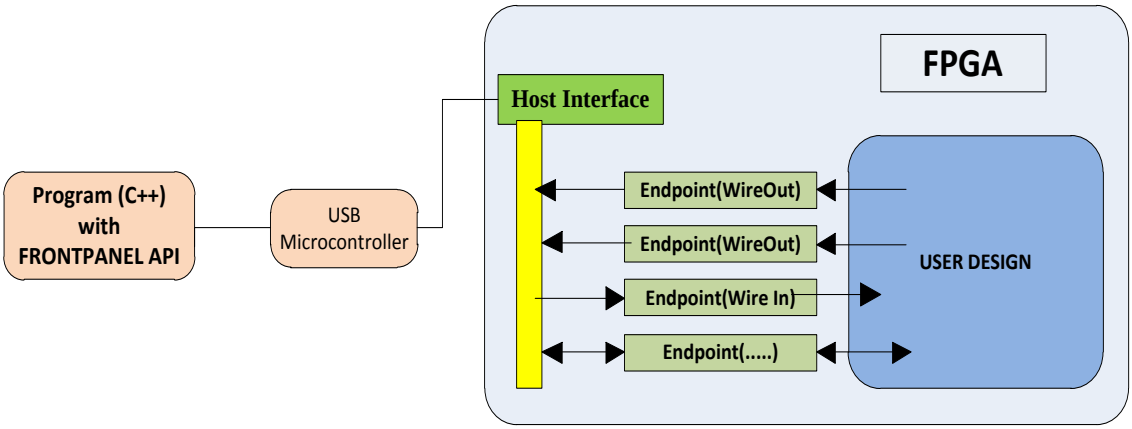


Figure 3.2: FPGA communication protocol with Front-Panel Host Interface [28] (edited).

3.2. Firmware for DPU Interface Emulator

The DPU-IF Emulator is implemented in a Xilinx FPGA on a XEM3001v2 card. This module has a USB interface from where it can get signals via endpoints in the design. The emulator is controlled by a LabVIEW program via Opal Kelly's Front-Panel API on the PC side. The figure 3.3 shows the DPU IF Emulator design.

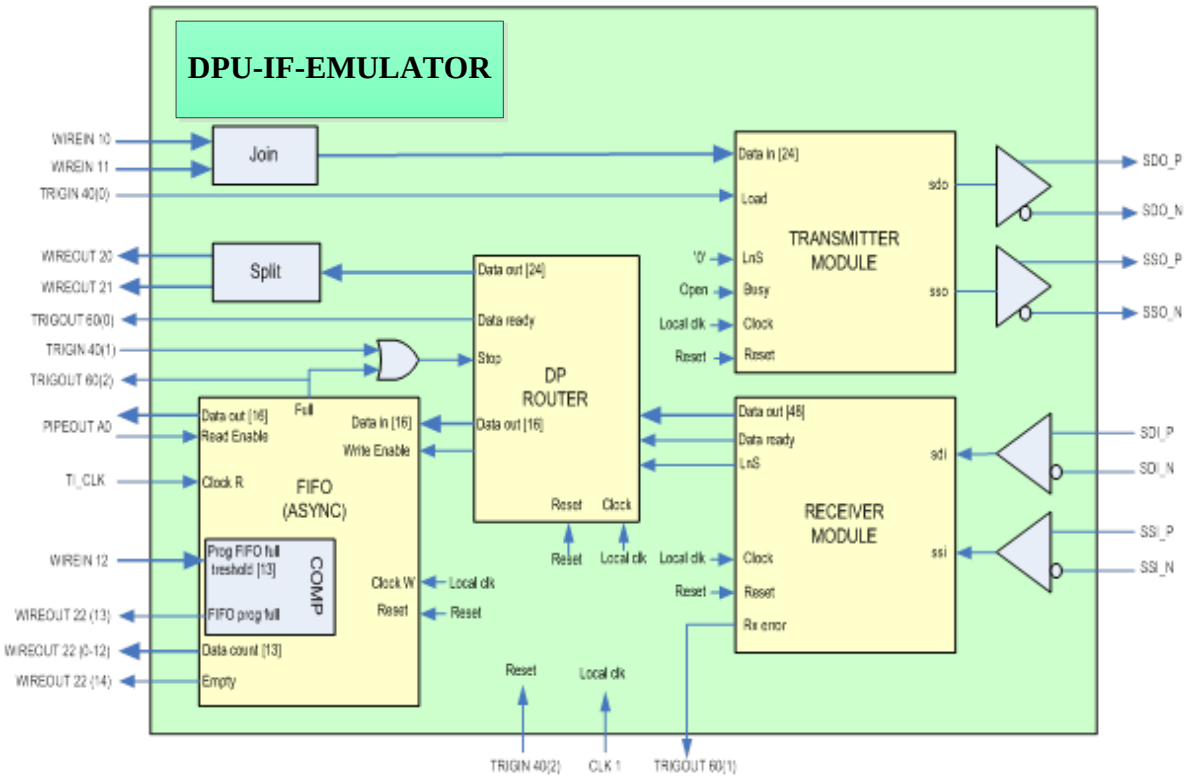


Figure 3.3: Block diagram of DPU IF Emulator design.

3.2.1 Sending of Data from the DPU to the RCU

The user sends the data to the RCU firmware from PC via DPU firmware. The data is sent to the RCU in 24-bit packet's format, which are applied on Wire Ins with the endpoint address 0x10 and 0x11 in this design as shown in figure 3.3. The placement of the bits has been changed in the 2.draft compareed to first draft to make the LabVIEW implementation easier. The Wire In 0x10 now applies the 2 flag bits and the 14 bits RCU memory address, and Wire In 0x11 applies the 8 bits of data in its least significant bits. The data is joined together and sent to the RCU by the Xlink transmitter module. The sending is triggered by Trigger In 0x40 bit 0.

3.2.2 Transferring of Data from RCU to DPU

In the response to the data received by the RCU, the RCU firmware sends the data packets to the user via DPU firmware. The transferred data packets from the RCU to the DPU received by the Xlink receiver module are of 24 and 48 bits. The error output of the receiver is put on Trigger Out 0x60 bit 1.

The 24-bit memory dump packets are sent to the PC by Wire Outs 0x20 and 0x21 in the same bit order as they described in the DPU to the RCU. The Trigger Out 0x60 bit 0 signals a data ready for the PC.

The 48 bit packets containing the measurement data are transferred at a high rate. To be able to receive all the measurements on the PC a Pipe Out is used for the transfer in the 2.draft whereas the Wire Outs are used in the first draft instead of Pipe Outs.

3.3. Data Packet Router

The main function of the data packet router is to route / split the different data packets on different endpoints according to the size. It routes 24 bit packets to the Wire Outs and the 48 bit packets on to the FIFO. This is done by a state machine which sets registers that control the routing. A statement diagram for the state machine for data packet router is shown in figure 3.4.

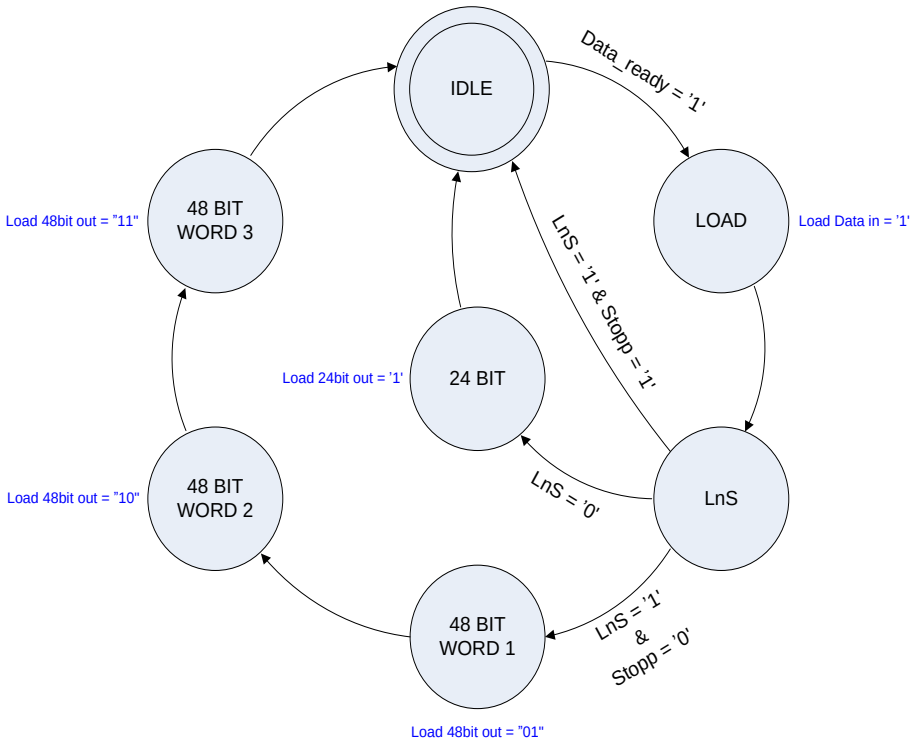


Figure 3.4: Diagram showing working of Data Packet Router [16].

3.3.1 State Machine

When data is ready at the input, the state changes to Load, and the data is loaded into a register. Then the state changes to LnS, and the LnS input is read. The LnS input comes from the receiver and tells if the data packet is 24 or 48 bits. If the data packet is 24 bits, it is put on the 24 bit output which is connected to the Wire Outs, and Trigger for the data ready is set.

If the packet is 48 bits and the stop signal is low, the data is put consecutively on a 16 bit output with the most significant word first. This output is connected to the input of a FIFO memory. The 48 bit *data ready* is held high throughout the words changing.

If the data is 48 bits and the stop signal is high, the state machine goes back to idle and wait for new data. The stop bit became high either the FIFO is full, or it is set high by Trigger In 0x40 bit 1.

3.3.2 FIFO

FIFO (First In First Out) is commonly used to buffer data between two asynchronous streams. FIFO queues allow data to be read and written at different rates. It works on the principle of what comes in first is handled first, what comes in next waits until the first is finished.

As mentioned earlier, a *Pipe Out* is used to send the measurement data to the PC at a higher speed than Wire Out (endpoint). The Pipe Out transfers data at different speeds depending on the size of the block of data transferred on each call. A FIFO memory with 6144 word depth has been implemented in the design to store the 48 bit measurement data packets. This means it can hold a total of 2048 data packets. The FIFO used in design was made by LogiCORE FIFO Generator in Xilinx ISE studio.

The FIFO is asynchronous as different clocks for reading and writing are needed. It has a programmable *FIFO FULL*, where the threshold is connected to the 13 least significant bits on Wire In 0x12. When the FIFO data count reaches this threshold, the signal connected to Wire Out 0x22 bit 13 goes high. This is used in the LabVIEW program to check if there is enough data to fill a Pipe Out block.

For testing purposes the Wire Out 0x22 bit 14 is connected to the FIFO empty signal, and the current data count is available on the 13 least significant bits of Wire Out 0x22.

The FIFO FULL signal sets Trigger Out 0x60 bit 2 high if it rises, so that the LabVIEW program knows if data may be lost due to a full FIFO. If FIFO is full and empty at the same time that means the FIFO is in reset, and the trigger will not be set.

3.4. The CZT-RCU Firmware

The CZT-RCU Firmware provides an interface between the XA-ASICs and the DPU for processing data received from analog readout electronics of the MXGS instrument. The main purpose of the CZT-RCU firmware is to access the detector modules, send control commands and get their status information. The following modules are defined in the CZT-RCU firmware.

- DM_IF modules.
- XA-CFG modules.
- RCU Master module.
- Bin Control module.

All the modules are defined with some control and status registers. Each register in the firmware modules has a unique address. The Modules are commanded by sending MWC (Memory write command) to the control registers and/or can read from the status registers by sending MRC (Memory Read Command) commands. These modules interface with the DPU through a data bus having one master module and number of slave modules where *DPU_IF* acts as a bus master as shown in figure 3.5. The *Memory data out* port is the only output port of all the slaves connected to an *addrdec* (address decoder). The *addrdec* selects the correct output to a specific address. By sending *memory data in* signal, bus master controls the read or write operation on the slave modules. There are four control registers (CR0-3) and four status registers (SR0-3) defined in the modules. The control registers can both be written and read by the bus interface whereas the status register can only be read.

The Science Data Packets are generated in the *DM_IF* modules and buffered into the FIFO. A *SCDP_ch_mux* (SCDP chain multiplexer) provides the write access for all four *DM_IF* modules.

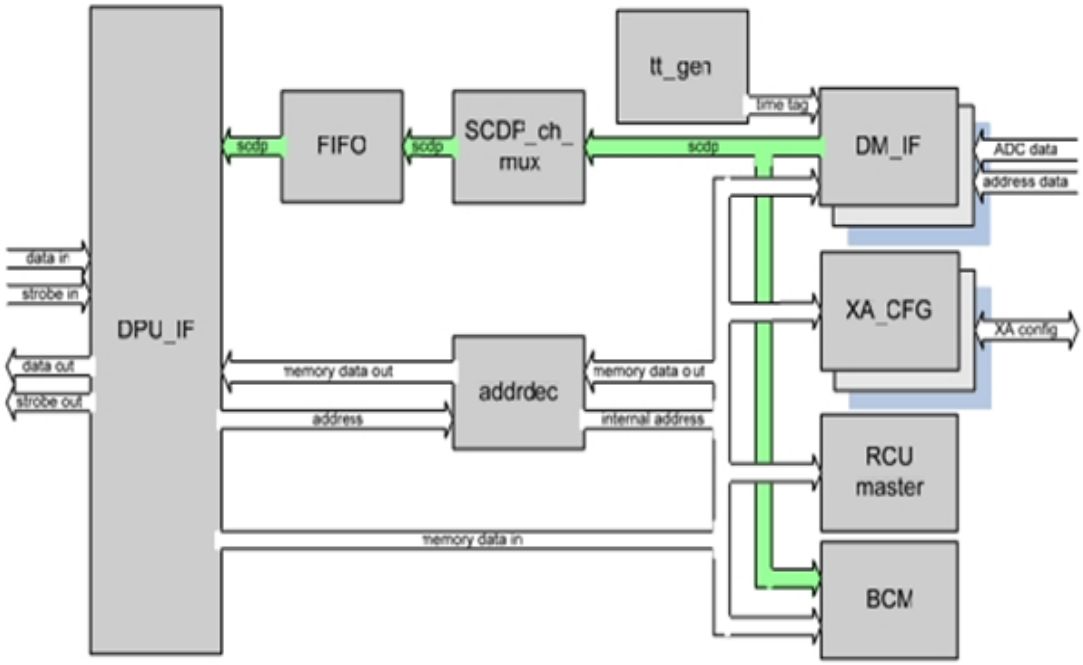


Figure 3.5: Block diagram of the CZT- RCU firmware modules [16].

3.4.1 DM_IF Module

The DM_IF firmware module collects and processes the measurement data received from the ADCs. It generates a SCDP for each photon hit with the address data and ADC output data. The DM_IF module samples the ADC output with frequency of 18 MHz continuously. When ADC data is sampled, it uses together with a 20 bit time-tag and constructs a 48-bit science data packet. The control registers CR0, CR1 and CR2 are defined inside each DM_IF to trigger any process and status registers SR0, SR1 and SR2 are defined to read the status of control registers.

3.4.2 XA _CFG Module

The XA_CFG modules are the entrance to the DM interfaces. There are four XA_CFG modules defined, one for each readout chain. The XA_ASICS in the DMs are configured by sending a serial bit stream with a clock signal through the XA_CFG Modules. The XA_ASICs are always in uninitialized and un-configured form when power is switched ON. Each XA_ASIC version 1.82 have shift register consisting of 858 bits for configuration, which is filled with clock signal and data provided by the serial data stream. The XA_CFG Module contains an embedded 4 kB RAM use for configuration data for all 32 ASICs in one DAU. The memory write command is used to write configuration data. The configuration data always sent twice. The data sent at second time is for reading purpose and is compared

with each bit, applied to start the configuration chain. The only control register CR0 is defined in the firmware so far to start the configuration / reset configuration status flags. The only status register SR0 is defined to read the status of the control register as register verification passed / failed.

3.4.3 RCU master

The RCU-master module in the firmware is defined for testing and housekeeping the entire RCU which receives the commands via internal bus interface. Eight LEDs was mounted on the DAB board in the BB and EBB model of MXGS for firmware and electronic testing but the LEDs will be omitted in flight version. Three control registers CR0-CR2 are defined to communicate and instruct the RCU-master module.

3.4.4 Bin Control Module (BCM)

The BCM module provides an alternate way of the data storage system in case of high event rate and saves them as SCDPs. The BCM module processes the science data packets generated by DM_IFs modules by binning into spectral bins. It creates a two-dimensional table of temporal and spectral bins, which is stored in a memory and can be accessible on the internal data bus. This module took the values of time tag and energy from a SCDP and determines which temporal and spectral bin the data belongs to. To communicate with the BCM, address 0x1028 is defined in the firmware which contains different control registers to trigger any function.

3.5. The BGO-RCU Firmware

The BGO-RCU Firmware provides an interface between the BGO detector and the user. The main purpose of the BGO-RCU firmware is to access the BGO detector modules, send control commands and get their status information. The following modules are defined in the BGO-RCU firmware.

- PMT_IF modules.
- TMON module.
- RCU-Master module.
- Bin Control module.

3.5.1. PMT_IF modules

The PMT_IF modules collect, read and process the data coming from ADCs and searches for gamma-ray events in the BGO detector. When it observes any event, it creates a SCDP packet with energy value and time of the event. There are three PMT tubes used on each DAU in the BGO detector. Therefore, there are three PM_IF modules defined in the firmware, one for each read out chain.

3.5.2. TMON module

The TMON module gathers and analyses the temperature values from the PM tubes and then uses that data primarily for temperature compensation in the PMT_IF modules and secondarily for housekeeping purpose. It always acquires ADC samples for the currently selected channel and store in internal register. The value of the register can be accessed from the DPU.

3.5.3. RCU-Master module

The RCU-Master module is the main controlling module for the entire RCU. This module is used for debugging and testing purpose. Many functions and control parameters are implemented in this module to check the instrument behavior. For example, the Master Reset command defined in its control register initializes all state machines and registers to their default state.

3.5.4. Bin Control module

In case of high event rate, the bin control module functions as a data-storage system and save the SCDPs. The Bin Control module processes the energy values and time-tags of the science data generated by the ADCs and create a two-dimensional table of temporal and spectral bins. The bin tables are stored in the memory and are accessible through the internal memory bus.

3.6. Data Packets

The data link between the RCU and the DPU is full duplex so the communication goes both ways simultaneously. It means commands from the DPU to the RCU and science data packets from the RCU to the DPU can sent independently. There are four types of data packets

transmitted between the RCU and the DPU. To minimize the traffic and to avoid the non-science data on the link, the data packets are made of different sizes. The data packets came in 24 and 48 bit sizes. The first two bit in each data packet are flag bits which identify the type of data packet. The flag bits indicate the direction and type of data.

3.6.1. Data packets from the RCU to the DPU

There are two types of data packets transmit from the RCU to the DPU. As mentioned above, the first flag bit "0" specifies that data has sent from the RCU. The second flag bit "0", notifying the receiver that data packet is a 48-bit SCDP-packet and if the second flag bit is "1", data packet is 24-bit MDDP packet. The data packets are:

Science Data Packet (SCDP): The Science data packets are transmitted from the RCU to the DPU whenever any event is observed. It has high priority on the bus. The SCDP is 48 bit data, which contain information about the ASIC channel address, the energy of the event, a multihit indicator and time tag bit. The two flag bit "00" indicates that data is sent from the RCU and is a 48-bit packet.

SCDP (48-BIT)

2 bits	7 bits	7 bits	10 bits	2 bits	20 bits
FLAG BITS	ASIC ADDRESS	CHANNEL ADDRESS	ENERGY	MULTI HITS	TIME-TAG
00	XX	XX	XX	XX	XX

Memory Dump Data Packet (MDDP): The memory dump packet is the 24-bit data packet and is the response of a MRC. It is used to read the status information as well as used for debugging purposes. The MDDP contains a dump of 1 byte of RCU memory. First two flag bits identify its type.

MDDP (24-BITS)

2 bits	14 bits	8 bits
FLAG BITS	ADDRESS	MEMORY DUMP DATA
01	XX	XX

3.6.2. Data packets from the DPU to the RCU

The data packets from the DPU to the RCU are of 24-bit which contains 14-bit of register address and 8-bit of command data along with two flag bits. First flag bit “1” specifies that data is coming from the DPU. The second flag bit “0”, indicate about 24-bit MRC data packet and if the second flag bit is “1”, data packet is 24-bit MWC data packet.

Memory Read Command (MRC): The MRC command instructs the RCU to download 8 bits of memory starting from the address specified in the address field. These packages are sending from the DPU to the RCU and contain 14 bit memory address from where the requested data should be downloaded.

MRC (24-BITS)

2 bits	14 bits	8 bits
FLAG BITS	ADDRESS	DON’T CARE
10	XX	XX

Memory Write Command (MWC): The MWC command is a 24 bit data package contains the register address and the data that has to be written on the RCU memory address. It is use to send commands to the RCU and to transport the configuration bits for XA configuration. This command is also used for debugging purpose. Special control registers are used to command the behavior of the RCU.

MWC (24-BITS)

2 bits	14 bits	8 bits
FLAG BITS	ADDRESS	DATA
11	XX	XX

3.6.3. System Response

The system always gives back a response when data is sent from the DPU to the RCU. The RCU generates a MDDP as a response of an MRC whereas it generates and sends the same data package back as the user sent to it, in response of the MWC.

Chapter 4

LABVIEW

To make a communication link between the MXGS detector and users, to control the detector and its various components, a software program is needed. A software is required that can make a bridge between the detector and users so that users can send commands to configure the RCU registers or to configure ASICs registers or to receive the data from the detector on his PC. For this purpose LabVIEW, a graphical programming software language is decided to use at UiB. This chapter provided a brief introduction with LabVIEW. The LabVIEW is the first priority of many scientists and researcher. *This chapter is based on the references [33, 34, 35 and 36]*.

4.1 LabVIEW-A Graphical Programming Language

LabVIEW is short for *Laboratory Virtual Instrument Engineering Workbench*. It is a powerful and flexible graphical development environment developed by National Instruments. Engineers and scientists in research, development, production, testing and service industries like automotive, semiconductor, aerospace, electronics, chemical and telecommunication have used and continue to use LabVIEW to support their works.

Figure 4.1: Example showing a LabVIEW program controlling and displaying the oscilloscope's functions on a PC [NI].

LabVIEW is a major player in the area of testing and measurement, industrial automation and data analysis. Figure 4.1 give an idea how users can control any instrument by running a LabVIEW program on their PC.

LabVIEW programs are called *Virtual Instruments* or ***VI*** because they often have the look and feel of physical instrument or system. LabVIEW uses a graphical programming language known as G programming language to create programs based on graphical symbols to describe programming actions. LabVIEW uses a terminology familiar to the scientists or the engineers and graphical icons used in G (Graphical programming language) can easily be identified by visual inspection. LabVIEW provides an extensive library of virtual instruments and functions to help the user in programming.

Each **VI** (a LabVIEW program) has two main parts, i.e. *front panel* and *block diagram.* Whenever we open any VI, two windows appear. The one is called front panel window and other is called block diagram window.

- *The **Block diagram** window* contains source program code that exists in a graphical form which consists of executable icons called nodes and wires that carry data between the nodes. Front panel objects appear as terminals on the block diagram. Block diagram is a pictorial representation of a program code or algorithm. The Block diagram contains terminals corresponding to front panel's controls and indicators as well as constants, functions, subVIs, structures and wires that carry data from one object to another object. The block diagram is built by wiring together objects that perform specific functions.
- *The **Front panel** window* is the interface to VI code. It is modeled from the front panel of physical instruments and is composed of switches, slides, meters, graphs, charts, gauges, LEDs and other various types of controls (inputs) and indicators (outputs). Controls simulate the types of input devices you might find on a conventional instrument such as knobs and switches and provide a mechanism to move input from the front panel to the underlying block diagram whereas Indicators provides a mechanism to display data originating in the block diagram back on the front panel. Indicators include various kinds of graphs and charts as well as numeric, Boolean, and string indicators. The icons and connectors specify the pathways for data to flow into and out of VIs. Users can control the program, change inputs and see data updated in real time.

4.1.1 The Control palette

The control palette is used to place controls and indicators on the front panel. It consists of top level icons representing sub-palette that contain a full range of available objects that user can use on the Front Panels. The Controls simulate input devices and provide a pipeline to move data to the block diagram. The Controls include knobs, push button and dials, etc. Almost every control on the control palette has a sub-palette which can be accessed by the user by clicking on the desired top-level icon. For example, clicking on sub-palette *Numeric* reveals various numeric controls (input) and indicators (output) that can be utilized on the front panel in a way to move data into and out of the program code.

The *control* palette is available only on the front panel window, when the front panel window is active.

4.1.2 The Functions Palette

The function palette is used to build the block diagram. It consists of many top-level icons representing sub-palettes, which contain a full range of available objects/functions that the user can use on the block diagram. The Function palette contains functions (vital elements of VIs) and many VIs that can be used to create new VI and they come with LabVIEW software. The sub-palettes on the function palette can be accessed by clicking on the top-level icon. For example, the sub-palette "*Structure*" contains For Loop, While Loop and Formula Nodes, etc. all of which are common elements of VIs.

The *Functions Palette* is available only on the block diagram, when the block diagram is active.

4.2 Sub-VIs

A sub-VI is a stand-alone VI that is called by other VIs, i.e. a sub-VI is used on the block diagram of a top-level VI. To create a sub-VI, it is required to build a connector pane and create an icon of any VI. The controls and indicators of a sub-VI receive data from and return data to the block diagram of a calling VI. There is no limit of the number of sub-VIs that a user can use in a calling VI. Using sub-VI in any program is an efficient programming technique which allows the user to reuse the same code in different situations. The user can call and run a sub-VI from inside a sub-VI as well. The user can create sub-VIs from VIs or can select a VI as a sub-VI. When creating new sub-VI from existing VI, it is important to

define the inputs and outputs of the sub-VI. This allows the calling VIs to send data to the sub-VI and to receive data from sub-VI. Furthermore, if an existing complex block diagram has a large number of icons, the user can choose any group related functions and icons into a lower-level VI to maintain simplicity of the block diagram.

4.3 Case Structures

The case structures are conditional branching control structure that executes only one of its sub-diagrams based on the specified input. It is similar to If-Then-Else command and the case statements in conventional programming languages. A case structure can have multiple sub-diagrams. The sub-diagrams are configured like a deck of cards from which only one card is visible at a time. The selector label is present at the top of the case structure border. The diagram identifier can be a numeric, string, Boolean or enumerated type control.

The selector label displays the values that cause the corresponding sub-diagrams to execute. The user can add a maximum of 2,147,483,648 (2^31) cases. An *Enumerated* type control is used to self-document each case of the Case Structure and generally is the best choice as case-selecting control. *Enum* (enumerated type) is a ring type control which associates a unique integer value with each item in a list of text descriptors. When *Enum* is wired to a selector terminal its text descriptors appear in the case selector label.

4.4 Local and Global Variables

In G-language, a variable is the wire connecting two objects on a diagram. If it exists only in one diagram or in one VI, it is a *local variable* and if it passes data among several VIs that run simultaneously, then it is a *global variable*. This can be possible by using shift register. By using shift register in a sub-VI, two or more calling VIs can share information by reading from and writing into the shift register. Such kind of variable is called Global variable. By using global variables user can access and pass data among several VIs.

A local variable allows the user to read data from or write data into, from controls and indicators without directly and actually wiring to the usual control or indicator terminal. This mean users have unlimited read/write access from multiple locations on the

diagram. Global variables can be read or written any time, any place and all callers can access the same data. LabVIEW has some built-in global variables as well. They are faster and more efficient than shift-register based global variables.

4.5 Loops

The loops allow a program to repeat commands over and over either forever or for a finite time period. There are two types of loop used in LabVIEW. The first type is **For loop,** which repeats a command to a predetermined number of times or until control by some instruction to stop the iteration and the second type is **While loop,** which repeats a command until a certain condition is met.

- ***The For Loop*** repeats the Block Diagram code to a predetermined number of times. When the number of iterations equals the predetermined counts, the loop stops. The For Loop works like the following code.

  For i = 0 to N-1

  Execute diagram inside the loop

  The For loop has two terminals, the *count terminal* (input terminal) and the *iteration terminal* (output terminal). The user can set the count by wiring a value from outside the loop to count terminal. The iteration terminal contains the current number of completed loop iterations; where 0 represent the first iteration and N-1 represent the Nth iteration.

- ***The While Loop*** is a structure that repeats a section of code until a certain condition is met. The While Loop is equivalent to the following code.

  Do

  Execute diagram inside the loop

  While the condition is TRUE/FALSE

  The While loop has also two terminals, the *conditional terminal* (input terminal) and the *iteration terminal* (output terminal). The input to the conditional terminal is always a Boolean variable, the True or False. The While Loop executes until the Boolean value wired to the conditional terminal become True or False. The iteration terminal has numeric output, which is equal to the number of times the loop has executed. It is zero for first and N-1 for the Nth iteration.

4.6 Error handling in LabVIEW

The Error handling is an essential part of creating a robust program. The computer programs are always subject to errors, either because of programming mistakes, invalid runtime inputs, or unavailability of required resources. The programs with no error handling will crash or invisibly perform incorrectly when an error occurs. The programs with basic error handling often display puzzled error messages and then exit. A program with sophisticated error handling will continue to run, automatically fix problems if possible, and if not, switch to a safe state appropriate to the error and inform the user.

By default, LabVIEW automatically handles errors when a VI run by suspending execution, highlighting the function or sub-VI where the error occurred and displaying an error dialog box. LabVIEW provides error handling VIs and functions on the *Function palette* to manage the errors. These VIs and functions return the errors either with numeric error-codes or with an error-cluster.

The error handling in LabVIEW follows the dataflow model which means the error information flow as the data flow through a VI. By connecting the error information from the beginning of a VI to the end and add an **error handler** VI at the end determines if the VI running without errors or not. To pass the error information through the VI, users must use the error-in and error-out clusters in each sub-VIs and VIs. As the VI runs, LabVIEW tests for errors at each execution node and if it detects an error, the node passes the error to the next node and to next without executing. In the end, it passes error to *Simple Error Handler* VI, which reports the error with some nice error message.

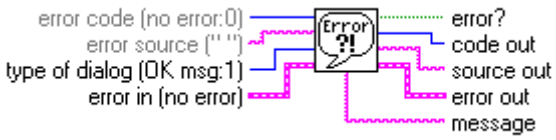


Figure 4.2: Snapshot of Simple Error Handler VI [LabVIEW 2010].

4.7 Instrument Drivers

An instrument driver in LabVIEW is a set of VIs that communicates and controls a particular programmable instrument. Each driver VI corresponds to a programmatic operation, such as configuring, reading from, write to, and triggering an instrument. They just simplify the

instrument control and make easy for the user by eliminating the need to learn the complex low-level programming command for each instrument. The user can use instrument drivers from LabVIEW library or can create their own drivers by using LabVIEW program. LabVIEW provides many supporting VIs that can be used in the construction of instrument drivers.

The LabVIEW instrument library contains instrument drivers for a variety of programmable instruments including GPIB (General Purpose Interface Bus), USB (Universal Serial Bus), TCP/IP (Transmission Control Protocol/Internet Protocol), VXI (VME eXtension for Instrumentation) and PXI (PCI eXtensions for Instrumentation) instruments. Instrument drivers contain high level VIs by which the user can quickly test and verify the capability of their instruments without the knowledge of device-specific syntax. The user can easily create instrument control applications by programmatically linking the instrument driver VIs in his program. To create an instrument's driver is same as creating any LabVIEW project.

Chapter 5
Software program for observation and configuration of the MXGS instrument

For testing the MXGS instrument in the lab at UiB, a program is developed in LabVIEW, which can communicate, control and trigger the various functions of the detector. The user runs the LabVIEW program on his PC, sends commands to the RCU, handle the configuration of the ASICs in DMs, collect the measurement data and analyze that data on his PC. This chapter introduces the tasks and provides a full architectural description of the LabVIEW program created for observation and configuration of the MGXS detector instrument.

5.1 Tasks of readout and configuration software

The program made for readout and configuration of MXGS instrument is created in LabVIEW development environment. The program is created for testing purpose in the lab and developed by focusing on the following tasks.

- Sending commands to the RCU.
- Configuration of ASICs in DMs.
- Collection and analysis of measurement data.

5.1.1. Send commands to the RCU

The first aim of the LabVIEW program is to communicate with the detectors. A program interface was needed where the user can send control commands to the MXGS instrument to trigger some functions in its Readout Control Unit (RCU) firmware. By using this program, the user can communicate with the various modules of the detector.

The RCU hosts an FPGA which is capable of manage the processes on the detector assembly unit (DAU). The RCU firmware consists of many modules, and each module is defined with some control registers (CRs) and status registers (SRs). The RCU creates and manages the science data packets (SCDP) for all photon events. The user can access the control and status registers using this LabVIEW program by inserting the memory write commands and memory read commands. The 14-bit address data describes the firmware module and register position

which the user wants to access. The command data is transmitted as 8-bit data to the specified address location as discussed in section 3.6, chapter 3.

In order to access CR1 (control register 1) in DM_IF 1 module in the CZT instrument, the address 1000 + 0001 = 1001 (in hex) has to be used Where 1000 (hex) is the address of DM_IF 1 module and 0001 (hex) is the address of CR1 control register in the DM_IF 1 module defined in the RCU firmware (see Appendix-B).

5.1.2. Configuration of ASICs

The ASIC configuration is an important function of this LabVIEW program. It activates the ASIC modules and prepares them for processing the event signals. Every time when the power is made on, all the ASICs are in uninitialized state and can't be used before their configuration. Each ASIC has 858 bits used for their configuration in shift registers. The ASICs are configured by sending a serial bit stream and a clock from an embedded RAM.

To configure the ASICs, the LabVIEW program has its own configuration interface (GUI) as shown in figure 5.1, which is discussed in detailed in section 5.3.1. With this configuration interface the user can adjust or change the parameters by clicking on different buttons and then save the ASICs configuration data into a text file. The configuration data file is saved as *.xar* file which refers to XA ASIC register data. The *.xar* files are text files, and it allows the user to change any data manually by using any text editor program. It is also possible to create, save and restore different configuration files for ASICs instead of using the default configuration file.

When the user configures the ASICs, the configuration file data stored in a cluster array is converted into a bit stream. These bits are sent to the RCU with Memory Write Command (MWC) in 8-bit data packets. The configuration bits always sent with "Start Configuration" command. All the ASICs are connected in series forming a configuration chain and then connected back to the feedback input of the FPGA. The bit stream data is always sent twice started with the most significant bit to the least significant bit. Each bit applied in the beginning of the configuration is compared with the bit at the end of the configuration chain. In case of any difference, the RCU sends an error message which indicates the improper functioning of any ASIC in the chain or the data configuration. In case all bits are identical, the ASICs are configured successfully and are ready to process the charged pulses coming from the CZT module.

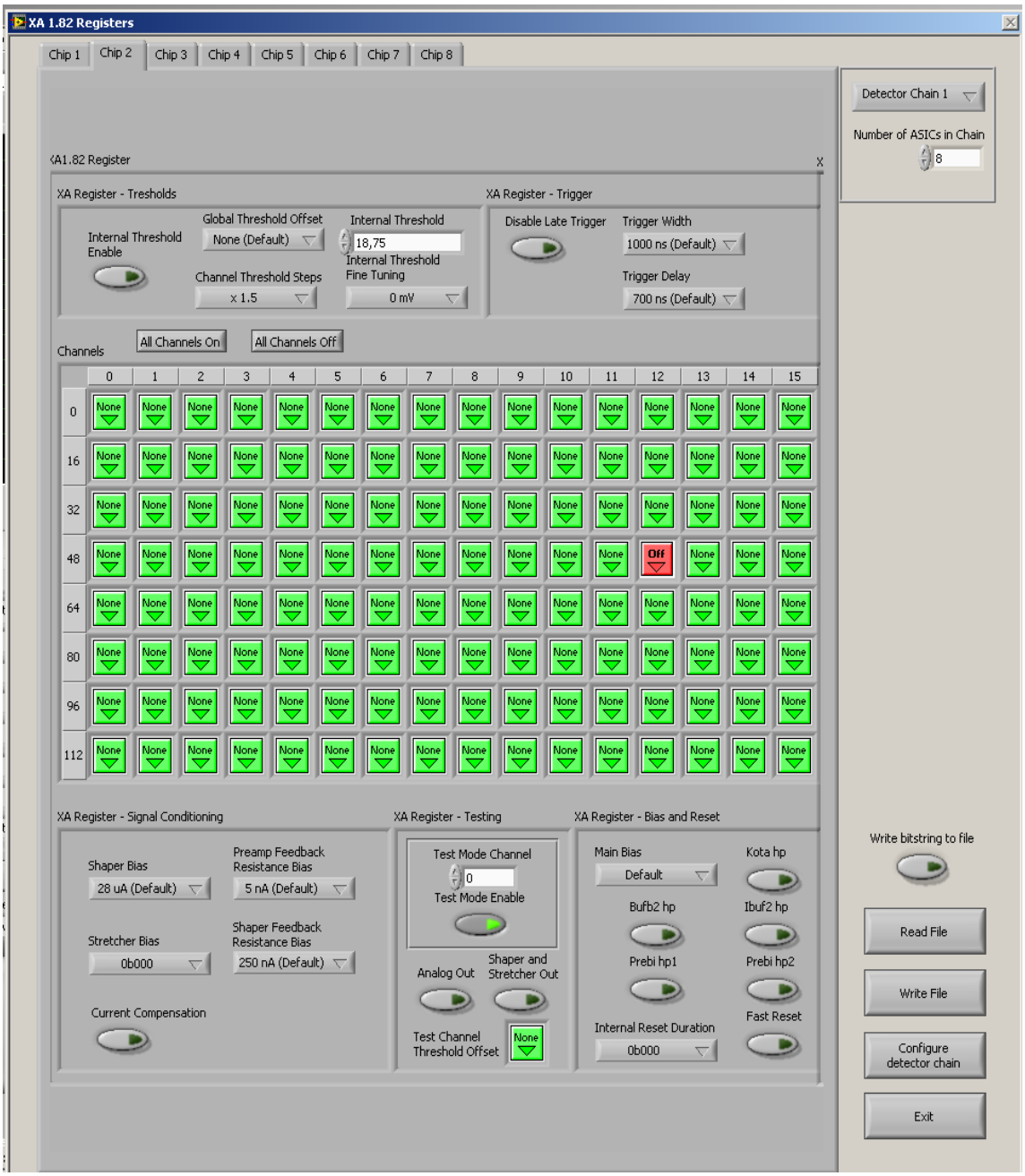


Figure 5.1: The XA-1.82 register's configuration interface [18].

5.1.3. Collection and analysis of measurement data

Whenever any photon event occurs at the detector, the science data packets (SCDP) containing information about photon energy, timing and pixel position/address are created and send. These data packets are recorded and saved in binary files. So another important task is the acquisition of science data packets, stores them on to the disk and later on analyzes the

measurement data by displaying the recorded data in various histograms and plots for further analysis.

The LabVIEW program is developed to fulfill all these requirements. The SCDP released from the RCU are buffered in the XEM-FIFO to increase the system performance. When the capacity of the FIFO reached to certain limit (2048 SCDPs), the XEM-FIFO signalizes a FIFO full and all the 2048 SCDPs are collected at once on to the disk and FIFO become ready for next cycle. It has been observed that computer become very slow to write the SCDPs on the disk in case of high data rate, i.e. when the science data package comes continuously; therefore, the SCDP packages are first stored in FIFO.

It is discussed in detail later in section 5.3, that the developed LabVIEW program is made of three parallel loops and the data acquisition is done in the producer loop, so the producer loop has highest process priority on the computer when LabVIEW program runs.

The consumer loop processes this data further. The data is temporarily buffered in the producer loop and then written on the disk in the consumer loop. The histogram arrays are generated in the consumer loop and then histogram data is displayed in the display loop. The display loop provides the feedback of all acquired data to the user. The data acquisition ends when the predefined number of SCDPs received (size target) or time target matched with the specified limit. The analysis of the captured data begins with conversation of 48-bit SCDP into Matlab matrices by C-based programing. The data package containing time tag information and multihit events data is ignored so far in this program. So the channel addresses vs. energy bin matrix, which contains the data about the number of counts detected at a particular point is shown as a result on the plots. The histogram data is sorted and displayed on the front panel of the LabVIEW program in four different plots. All the plots are discussed in section 5.3.3.

5.2 The Front Panel/Front End interface of MXGS instrument in LabView program

As discussed in chapter 4, a LabView program has always two faces (the block diagram and the front panel) so the developed LabView program to control the MXGS instrument has also two faces. The architectural description of the Block Diagram is fully discussed in section 5.3, but the Front Panel view of the developed LabView program is shown in figure 5.2.

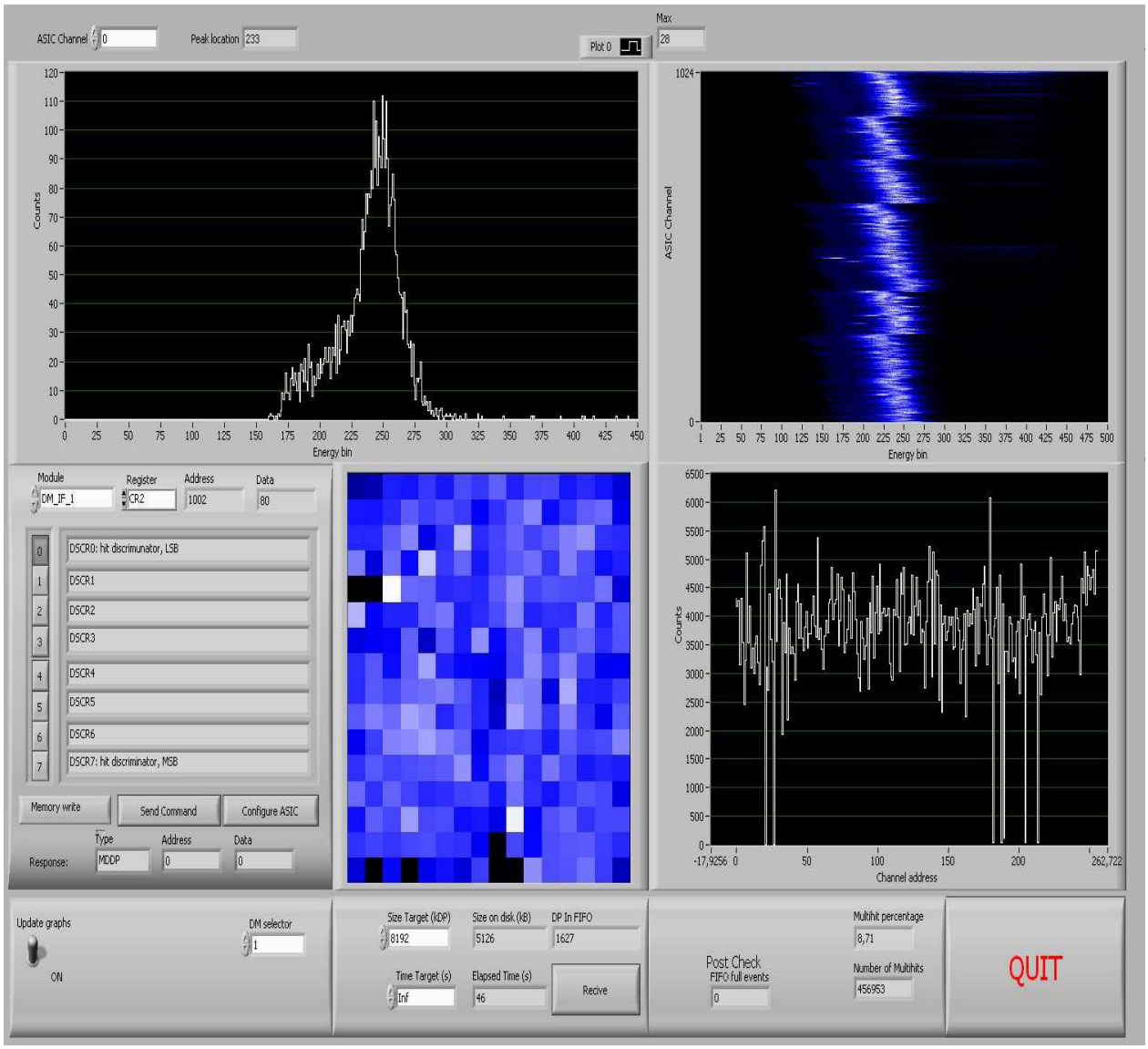


Figure 5.2: The Front Panel view of LabVIEW program created for the MXGS-CZT instrument [18].

The Front Panel of this developed program is a Graphical User Interface (GUI) made of Controls (Inputs) and Indicators (Outputs). As discussed in chapter 4, the controls can be knobs, switches, push buttons or dials, which represent the input of the device, whereas indicators can be graphs, charts, histograms or status message, which represent the output of the device. A GUI makes much easier and intimate interaction between the user and computer (instrument). The user can give some instructions to the instrument from the GUI and get feedback. Figure 5.1 showing the GUI interface developed for the ASICs configuration and figure 5.2 showing the main GUI interface for the MXGS instrument. The user can run the LabVIEW program on his PC, trigger some functions in the detector by sending command to

RCU from the GUI interface and observe the results on his PC on same front panel window. The main function of this window interface is to create easy communication with the detector and get feedback to the user by analyzing and displaying the collected measurement data on the PC. The XA-ASICs in the different detector modules are also configured from same front panel interface shown in figure 5.2. The above shown (figure 5.2) Front Panel Window/the GUI interface is developed on the basis of the schematic shown in figure 5.3 below.

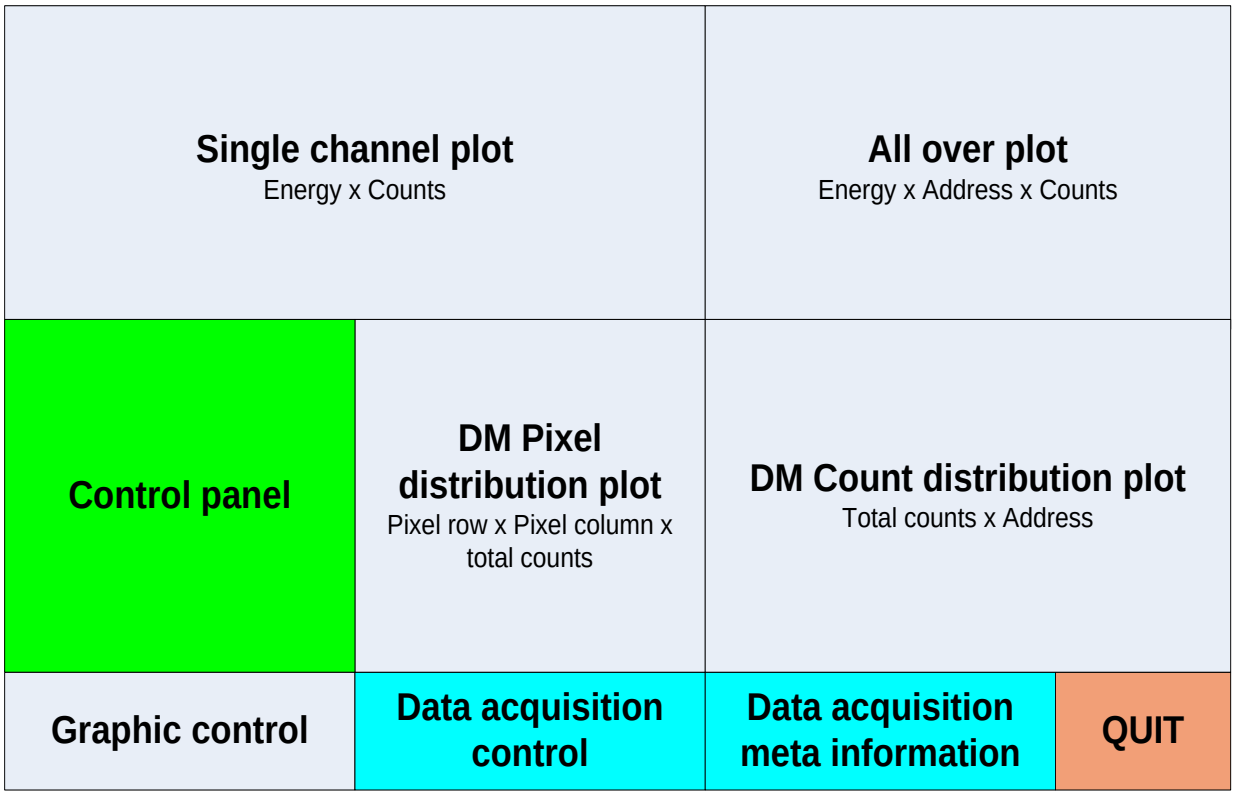


Figure 5.3: Schematic of EGSE front end [18].

5.3 Full description of readout & configuration software program

The readout and configuration software program for the MXGS instrument is developed in LabVIEW environment. All the technical, functional and architectural details of the developed readout and configuration software program are discussed detailed in this section.

The main program consists of three parallel timed loops. These loops perform different functions and have different performance priorities in the system. The three loops are:

- ***The Producer Loop*** hosts the main function states. This loop is developed to communicate with the detector, produce and collect the measurement data. Sending commands to the RCU registers, configuration of ASICs and collection of Science data packets all is performed in this loop. This loop has high priority during the data acquisition.
- ***The Consumer Loop*** handles the data packets which are collected and buffered by the producer loop. The histogram array is created from the data packets in this loop and data is prepared to display in the display loop.

- ***The Display Loop*** displays the collected data on the Front panel in the form of different plots and charts. This loop has lowest performance priority.

These loops also have different priorities in the processor queue to ensure that no data will be lost. As the program is created in LabVIEW software so the user can instruct/control the instrument and monitor results on the front panel after running the program and can modify or change the program codes on the block diagram.

5.3.1. The Producer loop

The producer loop performs the main functions in the program. The production and acquisition of data from the Opal Kelly XEM module is the most important task of this program and is performed in this loop. Users can't get any data to store or analysis if this loop stops functioning. This loop has the highest priority. Besides the acquisition of data, the loop also acts as the main loop of the program and controls the sending of data to the XEM module.

The producer loop is made as the state machine with the following states; Initialize, Idle, Configure XA Register, Send data, Initialize Get Science Data, Get Science Data, Exit Get Science Data and Shutdown as shown in figure 5.4. State machine is placed inside *Timed Loop.* The *Timed loop* executes one or more sub-diagrams sequentially, each iteration of the loop at the specified period. The user can use the time loop when he wants to develop VIs with multiple timing capabilities, precise timing and feedback on loop execution.

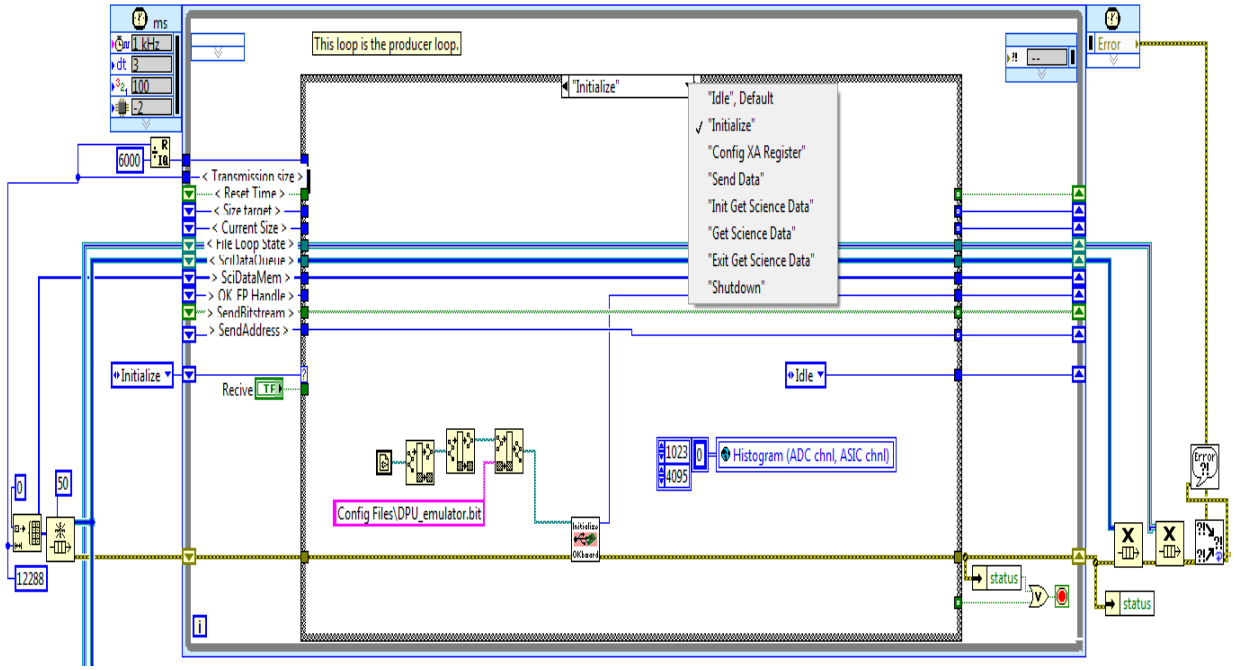


Figure 5.4: The block diagram view of the producer loop.

In FPGA based VI, the loop executes one sub-diagram at the same period as FPGA clock. The user can configure the timed loop by double clicking on the input-node or the right click on the structure. The values entered by the user in the dialog box will appear next to the input terminals on the input-node. The different states of the FSM in the producer loop are described blow.

1. Initialize:

This state runs as the first state when the program is started. It Initializes the Opal Kelly XEM module with the firmware found in the *DPU_emulator.bit* file stored in the configuration files folder of the LabVIEW project.

In "initialize" state on the block diagram, the file path of *DPU_emulator.bit* file is provided to a subVI named *Initialize DPU Emulation Board .vi.* The *Initialize DPU Emulation Board .vi* constructs a Front Panel object, opens a connection with the XEM board, loads the PLL from ROM and programs the FPGA from a Xilinx bit file.

"*Histogram"*, a global variable which contains the data of ADC channels and ASIC channels, which is a two dimensional (1024x4096) array is also initialize to zero in this state.

The *Initialize DPU Emulation Board .vi* further contain sub-VIs named the *construct.vi,* the *Open By Serial.vi,* the *Load PLL and get Frequence.vi,* the *configure FPGA.vi*, and the *Is Front Panel Enabled.vi.*

- The *construct.vi:* This VI is the constructor for the okUsbFrontpanel class. It returns a Handle for the okUsbFrontpanel object.
- The *Open By Serial.vi:* Before making any communication with the device, the device is to be opened by using [1]*okUSBFrontPanel.lvlib: Open By Serial.vi.* As the device is opened, this VI matches the given serial number string. If no serial number (or an empty string) is provided, then the first appropriate device is opened.
- The *Load PLL and get Frequence.vi:* This VI loads the PLL (Phase Locked Loop) and provides the current output frequency for a particular PLL output. It further contains subVIs named the *Load Default PLL Configuration.vi,* the *okPLLConstruct.vi,* the *Get EEPROM PLL22150 Configuration.vi,* the *Get Output Frequency.vi* and the *okPLLDestruct.vi* which are described below.
  - ✓ The *Load Default PLL Configuration.vi:* This VI configures the on-board PLL using the default configuration setup in EEPROM. If the specific device does not support this, the method returns error code unsupported feature.
  - ✓ The *okPLLConstruct.vi:* This VI is constructor for okCPLL22150 container class. This VI returns a Handle for okCPLL2215 object.
  - ✓ The *Get EEPROM PLL22150 Configuration.vi:* This VI retrieves the current PLL22150 configuration from the on-board XEM EEPROM. Then the PLL objects are initialized with this configuration.
  - ✓ The *Get Output Frequency.vi*: This VI provides the current output frequency for input PLL.
  - ✓ The *okPLLDestruct.vi*: This VI is the destructor for okCPLL22150 object.
  - ✓ The *Configure FPGA.vi*: This VI downloads the configuration file to the FPGA.
  - ✓ The *Is Front Panel Enabled.vi:* This VI checks if the Front-panel Host Interface has been introduced in the FPGA design. If it is detected, Front-panel support became enabled and endpoint's functionality became available.

[1] All VIs started with *okUSBFrontPanel.lvlib* are the drivers for Opal Kelly XEM3001 card in LabVIEW programming as the XEM card is provided with a software plateform named **FrontPanel** as mentioned in chapter 3.

The next state is the Idle state.

2. Idle

This is the default state of the state machine, where the program waits if nothing else happens. It checks if the Send Command, Config ASIC, Receive or Quit buttons have been pressed, and selects the corresponding state as the next state if regarding button is pressed. This is made by using the function named "Select". When the user press any of given button, the select function makes the FSM to run the related state.

This state consists of a VI named *rcu_registers.vi* which is used to send command instructions to the detector module and its registers. After communicating the detector module and register, related Data and Address are sent to *send data to RCU.vi* in "*send data*" state of the state machine. The command data values sent to the RCU must be in the binary form, therefore are converted into the binary data by using "*number to Boolean array*" function.

- The *Rcu_registers.vi*: This VI consists of two controls named *Module* and *Register*, two indicators named *Address* and *Data*, and two Clusters. All of these are visible on the Front Panel. First cluster is made of eight *OK* buttons and is used to select or change the data bit. Second cluster is made of eight *strings* and display the corresponding command description (according to the firmware) which will act after selecting the data-bit. This VI is based on the following example: In order to access CR1 (control register 1) in DM_IF 1 module in the CZT instrument, the address 1000+0001=1001 (in hex) has to be used. Where 1000 (hex) is address of DM_IF 1 module and 0001(hex) is address of CR1 control register in the DM_IF 1 module defined in the RCU firmware.

  By clicking on the '*Module'* control, user can select any module out of 14 given modules defined in the RCU firmware. Similarly, by clicking on the *'Register'* control, the user can choose any register out of four defined options. For each module, four control register are defined in this program and some of them are empty.

3. Configuration of XA-ASIC Registers

The ASICs configuration is one main function of this LabVIEW program. The ASICs configuration activates the ASICs and makes them ready for processing of the event signals. The ASICs have no memory to store the contents of configuration registers therefore it is

necessary to configure ASICs before each restart. The configuration register consists of 858 bits and includes 38 configuration parameters [16].

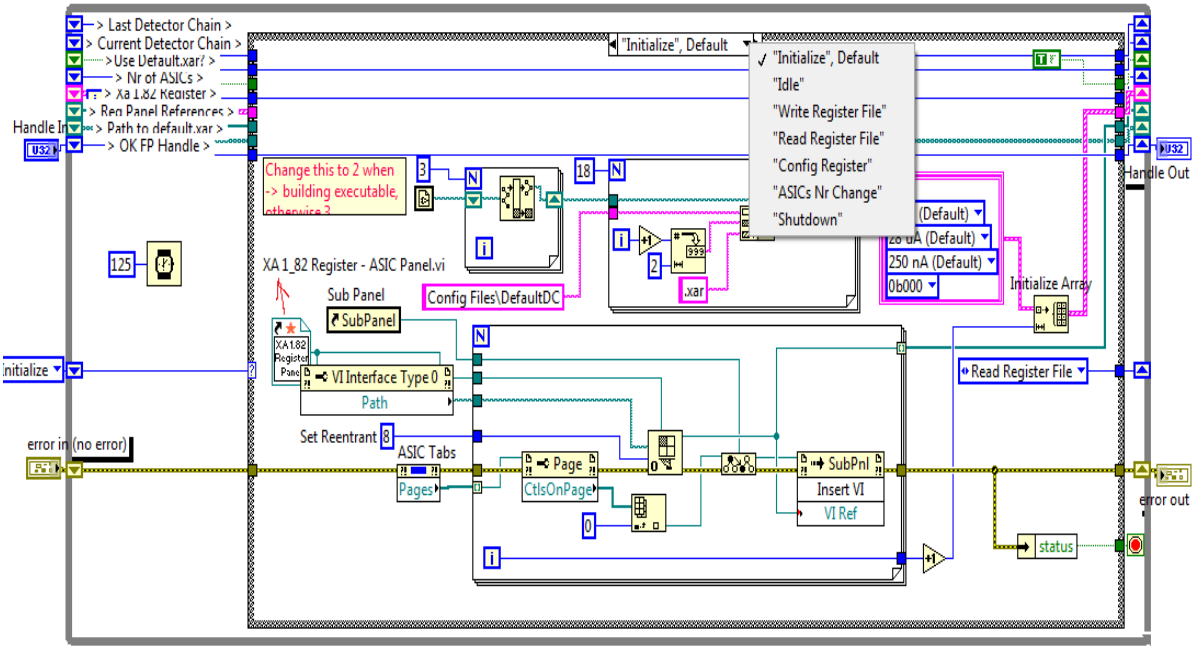


Figure 5.5: The block diagram view of Config XA Registers loop's states.

To configure the ASICs, this LabVIEW program has its own configuration interface (GUI) created in this state as shown in figure 5.1, which appears by pressing the '***Configure ASIC'*** button on the main front panel when the main program is running. The XA-ASICs became ready for processing of the science data after successful configuration of its registers. Next state is always Idle.

The ASICs configuration interface is also designed as a state machine with following states: Initialize (Default), Idle, Write Register File, Read Register File, Configure Register, ASICs Nr Change and Shutdown as shown in figure 5.5. The various states are described below.

3.1 Initialize (Default)

This state runs as the first state when the *"configure ASIC"* button on the Front Panel of the main program is pressed on. It initializes the XA1_82 registers data format which is created in the form of cluster in this loop.

On the block diagram of "initialize" state, the file path of *Default DC.xar* file is created and provided for further processing. The *Static VI Reference*: *XA 1_82 Register - ASIC Panel.vi* which is a strictly typed VI Reference is inserted into a

subpanel which makes it always appear a new window/subpanel on pressing the '***Configure ASIC***' button.

The 'Static VI Reference' function is used on XA*_1.82 Register - ASIC Panel.vi* which maintains the static reference to a VI and then *Strictly Typed VI Reference* is selected. Whenever a strictly typed VI reference is created, users don't have access to wire the VI reference to the 'RUN VI' method. Due to this reason, there is not any 'RUN' button on the *Config XA 1_82 Register GUI.vi.*

The next state is the *Read Register File* State.

3.2 Idle

This is the default state of the state machine in *XA-ASIC configuration* GUI program, where the program waits if nothing else happens. It checks if the *Write File, Read File, Config DC* or *Exit* buttons have been pressed, and selects the corresponding state as the next state. The **Select** function is used and it makes the FSM to reach the corresponding state. Three other controls named ASIC Tabs, Number of ASICs in Chain, and Detector Chain are used, which appear as *chip1...8, Number of ASIC in the Chain* and *Detector Chain* on the front panel.

In Idle state all the controls i.e. *Write File, Read File, Config DC, Exit, chip1...8, Number of ASIC in the Chain* and *Detector Chain* are waiting for the user to select any of them.

3.3 Write Register File

In this state, a referenced configuration file opens for which path is created in *Initialize (Default)* state and store the values in the XA 1.82 Register cluster with the help of *Write XA Register To File.vi* when the user uses the *Default.xar* file. If the user is not using the *default.xar* file, a file dialog box pop-ups where the user can save a new *.xar* file which is created by writing in with *Write XA Register To File.vi.*

The *Write XA Register To File.vi*: This VI opens a referenced configuration file and store the values in the XA 1.82 Register cluster. All the values are stored in sections named after their container bundle and chip n (where n is the chip address). The information about XA Register's Bias and Reset, Testing, Trigger, Signal Conditioning, Thresholds and about Channels is stored in the file. All the values are stored in sections called *XA Register – Bias and Reset/Testing/Trigger/Signal Conditioning/Thresholds/Channels – Chip1/2/3/.....8.*

This data is stored in the text file on the PC for housekeeping purpose in *.xar* format.

When the user using *Default.xar* file, and if the *current detector chain* is equal to *Last Detector Chain*, the S*hutdown* will be the next state otherwise next state will be the *Read Register File.*

When the user using other than the *Default.xar* file, next state will be the Idle state.

3.4 Read Register File

This state reads the configuration register file for current detector chain, which is stored on the PC for housekeeping in "*Write register file"* state. The reading process is done by *Read Register from File.vi.* When the *Default.xar* register configuration file is used, the registers from this file will be read by the program. In case the user wants to use any other configuration file, a file dialog box pop-ups and the user has to select any desired file. The contents of the new *XA register file.xar* file will be loaded into the program.

The *Read Register From File.vi*: This VI is almost same as that of *Write XA Register to File.vi* which stores the information about XA Register's Bias and Reset, Testing, Trigger, Signal Conditioning, Thresholds and about Channels in a file. The difference is that this VI bundled all the stored information into a cluster which is used for further processing.

If the user uses the *Default.xar* file or cancel the file dialog box, then the next state will be Idle. In opposite case, the next state is ASICs Nr Change.

3.5 Configure Register

In this state, the XA-1.82 Register format data is converted into a bit-stream and then send to the RCU by using the Memory write command. The conversion of configuration data into Bit-stream is done by *XA 1_82 Register To Bitstream.vi.*

The *XA-1.82 Register To Bitstream.vi*: This VI unbundles the XA 1.82 Register cluster into XA Register – Thresholds, signal conditioning, testing, trigger, and bias and reset and sorts the five resulting bundles to five VIs that converts the values to bit values and stores them into the XA Register Bit-stream. These five VIs are *Channels and Thresholds To Bitstream.vi, Signal Conditioning To Bitstream.vi, Trigger To Bitstream.vi, Testing to Bitstream.vi,* and *Bias and Reset To Bitstream.vi.*

- The *Channels and Thresholds To Bitstream.vi:* This VI unbundles the Channel and Threshold bundle and converts the values into bit arrays which is then replaced into the XA Register Bit-stream bit array.
- The *Signal Conditioning To Bitstream.vi:* This VI unbundles the Signal Conditioning bundle and converts the values into bit arrays and deletes the unused bits. The arrays are then replaced into the XA Register Bit-stream bit array. The Current Compensation bit is inverted according to the XA specifications.
- The *Trigger To Bitstream.vi:* This VI unbundles the Trigger bundle and converts the values into bit arrays and deletes the unused bits. The arrays are then replaced into the XA Register Bit-stream bit array.
- The *Testing to Bitstream.vi*: This VI unbundles the Testing bundle and converts the values into bit arrays and deletes the unused bits. The arrays are then replaced into the XA Register Bit-stream bit array.
- The *Bias and Reset To Bitstream.vi:* This VI unbundles the Bias and Reset bundle and converts the values into bit arrays and deletes the unused bits. The array is then replaced into the XA Register Bit-stream bit array.
- The *Channels setting to Bitstream.vi:* This VI unbundles the Channel Disable and Thresholds bundles in the Channel Settings array and converts the values into bit arrays, which is then replaced into the XA Register Bit-stream bit array. The Channel Enabled bits are inverted to correspond to the channel disable bits of the register.

  The Bit-stream is send to the RCU firmware with the Memory write command and starting address. The *Send Data to RCU.vi* is used to write the registers on the RCU with memory write command.

  Next state is Idle.

3.6 ASICs Number Change

This state provides reference to the ASIC pages, controlled/defined in by ASIC Tabs, which means clicking on any of chip 1.....8 buttons on the front panel opens that particular chip tab. Next state is *Idle.*

3.7 Shutdown

In this state *refnum* associated with an opened VI is closed and the FSM shutdowns. The XA-ASIC configuration GUI stops and quits.

4. *Send Data*

This state uses the *Send data to RCU.vi* to send the data from *Send-Bit-stream* registers to the addresses in the RCU by inserting *Memory Write*/*Memory Read* command from the *Send-Address* register.

The 'Send data' state in the block diagram consists of two sub-VIs, one control and three indicators. The *Send data to RCU.vi* and the *read from RCU.vi* are used to write the data and to get back a response from the RCU firmware respectively. The "command type" is a control to insert the *Memory Write* or *Memory Read* command.

The "*Data*", "*Address*" and "*Type*" are three indicators in this state, which are also visible on the Front panel of the main program. These three indicators provide system response data on the front panel when the *Memory Write or the Memory Read* command is sent to the system.

- The *Send data to RCU.vi*: This VI takes 8-bit data from the bit-stream in each iteration of the 'For loop' and sends them together with the RCU register address (14-bits) to the DPU emulation board on *Wire_In* 16 and 17 and then updates the *Wire_Ins*. The first two bits are always flag bits that send as "11" to signify the memory write operation. The *Activate Trigger in.vi* starts functioning on *Trigger_In* 64 and sends the data and address to the RCU. The *Send data to RCU.vi* further contains four sub-VIs which make together proper functioning of this VI.

  The two *Set Wire In Value.vi* VIs with different endpoint values, the *Update Wire Ins.vi* and the *Activate Trigger In.vi* are used as sub-VIs to create the *Send data to RCU.vi.*

  - ✓ The *Set Wire In Value.vi*: The Endpoint Wire_In's values are stored internally and updated when necessary by calling "Update Wire Ins.vi". The values are updated on a per-endpoint basis by calling this method. The user can set endpoint value according to his need when he uses this VI in his program. In addition, specific bits may be updated independent of other bits within an endpoint by using the optional mask input. Two *Set Wire In Value.vi* with endpoint value 16 and 17 are used.
  - ✓ The *Update Wire Ins.vi*: This VI is called after all Wire_Ins values have been updated using "*Set Wire In Value.vi".* This VI only updates the values held within a data structure inside the class and make the changes to the XEM simultaneously so that all the wires will be updated at the same time.

- ✓ The *Activate Trigger In.vi:* This VI activates the specified *Trigger_In* on the XEM card. The 64 is activated as *Trigger_In* here.
- ✓ The *Read from RCU.vi:* This VI provides the data gathered by the RCU, register address where the data is coming and the Type of data. Three VIs: *Update Wire Outs.vi*, and two *Get Wire Out Value.vi* are used as sub-VIs in this VI. Type of data can be SCDP, MDDP, MRC and MWC. This VI has two inputs, i.e. *Handle In* and *Error In* and five outputs i.e. *Type* of Data, *Handle Out, Data, Address* and *Error Out*. The three outputs: *Type, Data* and *Address* are visible on the Front Panel whereas *Handle Out* and *Error Out* are used for internal functioning.
- ✓ The *Update Wire Outs.vi:* This VI is called to request the current state of all *Wire_Out* values from the XEM. All *Wire_Outs* are captured and read at the same time. The "*Get Wire Out Value.vi*" is used to access the values captured after this call.
- ✓ The *Get Wire Out Value.vi:* This VI provides the Wire Out value at the specified address. The wire must first have been updated using "*Update Wire Outs.vi*".

The two *Get Wire Out Value.vi* with endpoint address 32 and 33 are used to make *"read from RCU.vi"* functioning. So the output is available on *Wire_Out* at endpoint address 32 and 33. The *wire_out* value at endpoint 32 is then converted into bit array and examined. The bits value of bit number 0 to 14 give us *ADDRESS* of the data. The next two bit is checked and provides the *type of data*. If these two bits value after bit 14 is 0, data is SCDP type, if the value is 1 data is MDDP type, if the value is 2 data is MRC type and if the value is 3 data is MWC type.

The *wire_out* value at endpoint 33 is also converted into a bit array and examined. The bits value from 0 to 8 bit gives us *DATA* value read from the RCU.

The *Data, Address* and *Type* indicators are available on the main Front Panel in front of *"Response".* The next state is Idle.

5. Initialize Get Science Data

This state initializes the values and the variables that are necessary for the acquisition of data from the XEM module. A file dialogue box with SCD file and *.scd* format appears where the user can select the file name and can select a folder where the acquired science data is to be

store and save. The file name is stored in a global variable so that file can be read/write from other states as well.

If the file dialogue box is canceled, an error is set and removed, the *Receive* button is reset and the next state will be *Idle*. If the file dialogue is not canceled, an *Open* state is put in the queue for the *Consumer* loop and the process priority, and the thread priority is set to maximum by using "*Set Process and Thread Priority.vi*" and the loading of science data into the FIFO on the XEM-FPGA module is started by setting bit 1 on *trigger_on* address 40 (0x64) by using "*Activate Trigger In.vi*" in the XEM module.

The *Size target* and read from the control are loaded in register as multiples of the transmission size. The values for *rest time* and *target size* are also loaded to the registers.

If the file dialogue box is canceled, next state will be *Idle* otherwise next state will be *Get Science Data.*

6. Get Science Data

This state updates the *wire_outs* from the XEM card and gets the FIFO Data count from *wire_out* 34 (0x22). It compares the data count to the transmission size. If there is enough data in the FIFO and condition is fulfilled then it reads the data from *pipe_out* 160 (0xA0). If there is not enough data in FIFO, it will show data packet count value in the '*DP in FIFO'* indicator, on the Front Panel. The data is further send to the file writing loop via Science Data Queue and a *Write state* is put in the file loop state queue.

If the *Receive* button is pressed, or the Elapsed time or size target's (size target <= current size) conditions are fulfilled, the 'Exit Get Science Data' will be the next state.

The "Time Target" control and two indicators "Size of disk (kB)" and "Elapsed Time" which are visible on the Front Panel are used in this state.

The *Update Wire Outs.vi, Get Wire Out Value.vi* and *Read From Pipe Out Fast.vi* are used as sub-VIs in this state.

- *The Update Wire Outs.vi*: This VI is called to request the current state of all *Wire_Out* values from the XEM card. All *Wire_Outs* are captured and read at the same time. The "*Get Wire Out Value.vi*" is used to access the captured values after this call.

- *The Get Wire Out Value.vi*: This VI provides the Wire Out value at the specified address but the wire must be updated before using "*Update Wire Outs.vi*". The endpoint address 34 is used to get Wire_Out value.
- The *Read From Pipe Out Fast.vi*: This VI transfers all the data from a specified *pipe_out* endpoint to the Data Out array which is further send to the file writing loop.

If the "*Receive* "button is pressed, or the *Elapsed time* is over or the *size target*'s (size target <= current size) conditions is fulfilled, the next state will be *Exit Get Science Data* state.

7. Exit Get Science Data

This state stops the loading of science data into the FIFO on the XEM module and resets the FIFO by *trigger_in* 40 (0x64). This state sets a "Close state" in the file loop queue, resets the *Receive* button if needed and resets the process priority and thread priority. The *Activate Trigger In.vi* is used twice with trigger bit 1 and 4. The Size Target is reset to 0.

- The *Activate Trigger In.vi* : This VI activates the specified *Trigger_In* on the XEM module.

The next state is *Idle.*

8. Shutdown

This state shuts down the connection to the XEM card, sets a "Shutdown state" in the *file loop queue*, and sets the global *Quit* variable to true. This ends the state machine loop. The *okUSBFrontPanel.lvlib:Destruct.vi* is used in this state as sub VI.

- *The Destruct.vi*: This VI is the destructor for okUSBFrontPanel object.

5.3.2 The Consumer loop

The Consumer loop handles the data packages which were collected and buffered by the producer loop. This loop queues the state machine for writing science data into a file and sorting it into a histogram array which is further used to display in the display loop. The Consumer loop is also made as FSM with following states: Open, Write, Close and Shutdown as shown in figure 5.6.

1. *Open*

This state reads the file path from the global variable, and opens the file. It initializes the Histogram global variable data and the value of last flush counter register.

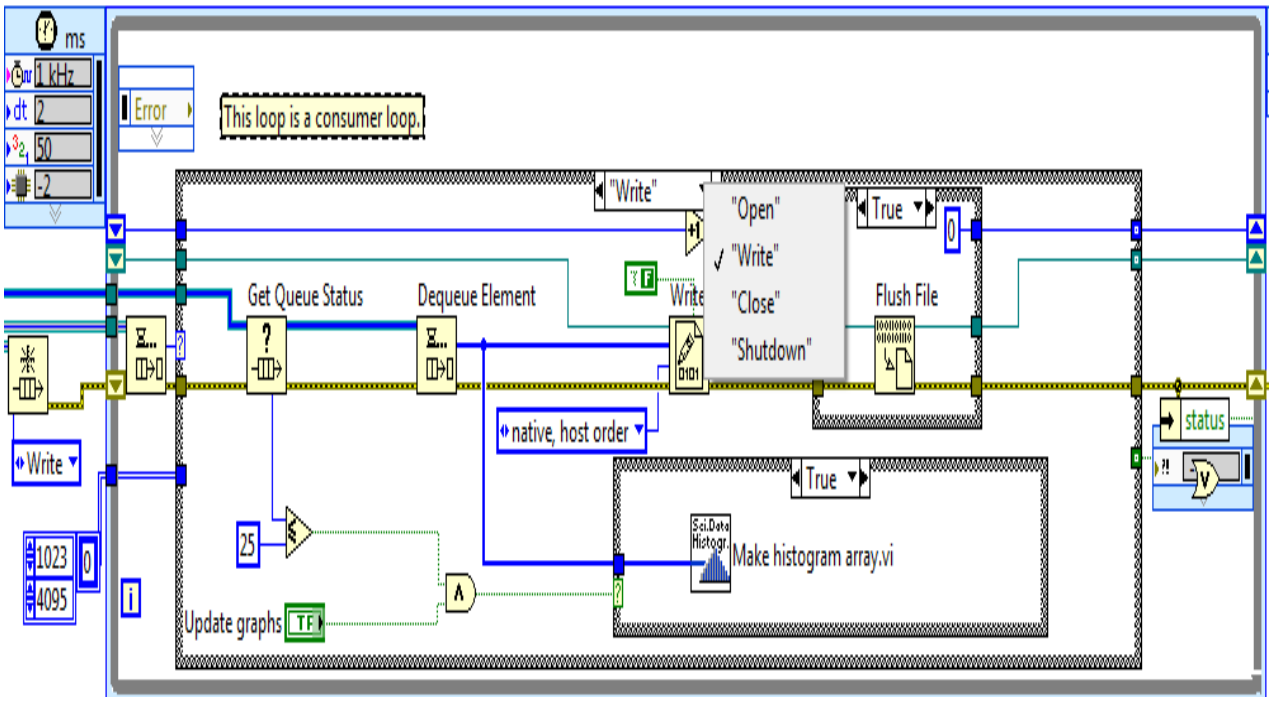


Figure 5.6: The block diagram view of The Consumer loop in the main program.

2. *Write*

As shown in the figure 5.6, this state checks the size or number of elements in the science data queue and read one array of data at a time from it. The data is written/store in a file or replaced in the existing file in a binary form, and that file is flushed (writes all buffers of the file to disk and updates the directory entry) every 10th time the loop runs.

If there are less than 25 arrays or elements in the Queue and *Update Graphs* switch is on, the science data is sorted in a histogram in a global variable. This histogram global variable is read by the Display loop.

3. *Close*

This state runs a post check on the science data that has been stored in the file, and closes the file. The *Post Check.vi* is used here which provides Multihit percentage,

FIFO full events and Number of Multihits. These indicators are available on the main Front Panel.

4. Shutdown

This state exits the consumer loop.

5.3.3. The Display Loop

The Display Loop makes it possible to display the collected measurement data on the front panel of the main program. Four different plots are created on the front panel with the collected science data. The Display loop updates its data when "*Update Graphs*" is ON or TRUE. The different plots on the Front Panel are created with the data collected in Histogram, which is a global variable and found in different states in the Producer Loop, the Consumer Loop and the Display Loop. Figure 5.7 shows the program code made to develop the different plots on Front Panel in the *Display Loop*.

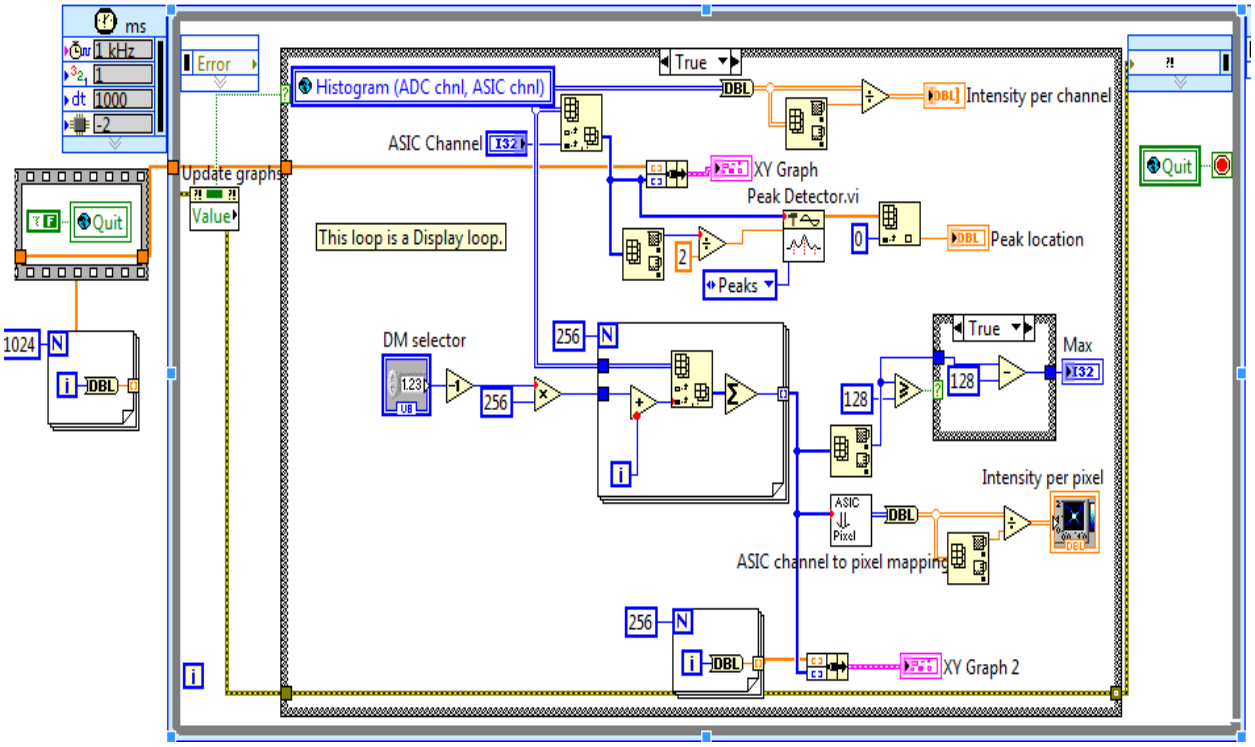


Figure 5.7: The block diagram view of The Display Loop in the main program.

The following plots are available on the Front Panel-User Interface created with the collected measurement data:

- The first plot on the Front panel is "**Intensity per channel"** graph formed by two-dimensional data collected in the histogram-global variable. This plot is created between the numbers of count per energy versus the channel address.
- The second plot is the *number of counts per energy for a particular single channel* which also provides the peak *location*. The channel can be select by the user. To get the peak location, *peak dectector.vi* is used. The *Peak dectector.vi* finds the location, amplitude and second derivative of peaks of the input signal.
- The third plot on the Front Panel is "**Intensity per pixel"** plot for selected DM which is created with the help of *ASIC channel to pixel mapping.vi*. This plot shows the number of counts on a pixel to their physical position in the DM. The user has a choice to select the desired DM and can observe the hits on that particular DM.

ASIC channels																
ASIC 1									ASIC 2							
034	036	030	032	028	020	018	016		036	034	030	032	028	016	018	020
038	040	026	024	022	012	014	002		038	040	024	026	022	002	012	014
041	045	043	031	029	006	004	008		041	045	031	029	027	008	006	004
042	047	037	027	025	010	001	003		042	047	043	025	021	003	001	010
039	035	044	021	033	005	007	009		039	035	037	046	048	033	005	007
049	051	053	050	048	046	011	015		049	051	044	052	054	050	009	011
055	057	059	056	054	052	013	017		055	057	053	058	060	056	013	015
061	063	64	062	060	058	019	023		061	063	059	064	062	023	017	019
068	066	070	065	067	108	110	112		068	066	065	067	069	071	108	110
074	072	081	071	069	073	114	116		074	072	070	073	075	077	112	114
085	083	090	077	075	079	118	120		085	083	081	079	106	104	116	118
094	092	096	104	106	102	122	124		094	092	090	100	102	120	122	124
078	080	076	098	100	127	128	126		078	080	096	107	098	126	128	127
084	082	103	105	107	115	119	123		084	082	076	103	105	123	119	115
087	086	097	099	101	121	125	111		087	086	099	093	101	111	125	121
089	088	091	093	095	109	113	117		089	088	097	091	095	117	113	109
Pixel numbers																
16	32	48	64	80	96	112	128		144	160	176	192	208	224	240	256
15	31	47	63	79	95	111	127		143	159	175	191	207	223	239	255
14	30	46	62	78	94	110	126		142	158	174	190	206	222	238	254
13	29	45	61	77	93	109	125		141	157	173	189	205	221	237	253
12	28	44	60	76	92	108	124		140	156	172	188	204	220	236	252
11	27	43	59	75	91	107	123		139	155	171	187	203	219	235	251
10	26	42	58	74	90	106	122		138	154	170	186	202	218	234	250
9	25	41	57	73	89	105	121		137	153	169	185	201	217	233	249
8	24	40	56	72	88	104	120		136	152	168	184	200	216	232	248
7	23	39	55	71	87	103	119		135	151	167	183	199	215	231	247
6	22	38	54	70	86	102	118		134	150	166	182	198	214	230	246
5	21	37	53	69	85	101	117		133	149	165	181	197	213	229	245
4	20	36	52	68	84	100	116		132	148	164	180	196	212	228	244
3	19	35	51	67	83	99	115		131	147	163	179	195	211	227	243
2	18	34	50	66	82	98	114		130	146	162	178	194	210	226	242
1	17	33	49	65	81	97	113		129	145	161	177	193	209	225	241

Figure 5.8: The Pixel number versus ASIC channel numbers for one DM results [16].

- ✓ The *ASIC channel to pixel mapping.vi*: The actual numbering/position of pixels in ASICs is not continuous and not same as shown in the "*XA 1.82 Registers.vi*" (VI used to configure the ASICs). To make it similar, mapping of the ASIC channels with pixels have been done in this VI. This channel to pixel mapping is done on the basis of results shown in figure 5.8.

- The fourth plot on the Front Panel is the total number of counts of all channels in a particular DM. The plot is between the counts verses the channel address. Users have a choice to select the desired DM and can observe the hitting counts on that particular DM.

With the first two plots, users can observe the energy distribution of measurement data whereas the last two plots deals with quantity of counts so the user can also have an idea about **Noise** detection. A channel having Noise shows more counts compared to its surrounding channels. Furthermore, the corner pixels in the DMs are suspected to have slightly different behavior than the central pixels.

The global variable 'Quit' is also present in this loop, which means that the user can quit the main program just by clicking on the ***Quit*** button on the front panel. The Display loop has the lowest performance priority.

5.3.4 Error handling

As discussed in chapter 4, error handling in LabVIEW is a very important task to make the program robust, to avoid crashing and incorrect functioning of the program. Error handling in LabVIEW follows the dataflow model. The error information flows through a VI just as data values flow. The LabVIEW program is developed by keeping these facts in mind. The error-information is wired from the beginning to the end through each loop in every VI and sub-VI. The *error-in* and *error-out* clusters are used in each VI for checking the error-information in each loop. In case LabVIEW detects en error, the node passes the error to the next node without executing that part of program code. Same does the next node with sending error information further and so on. In the end, the *error out* clusters from the producer loop, the consumer loop and the display loop are added together by *Merge Multiple Errors.vi* and connected with *Simple Error Handler.vi*. The *Simple Error Handler.vi* handles the errors and provides a description of the error in a dialog box without crashing the program.

Error Converter.vi: The *Error Converter.vi* is used in many sub-VIs. This VI converts Error Codes from *okFrontPanel.dll* calls into LabVIEW errors. The file "*okFrontPanel-errors.txt*" must be present in the folder *user.lib/errors* which is located in the LabVIEW folder. If the *Error Code* input is positive, it will set in a Positive Code Output which is used during reading and writing to pipes.

Merge Multiple Errors.vi: This VI merges errors I/O clusters from multiple parallel error chains by checking each input terminal for error status = TRUE and error code = nonzero. The terminals are searched in order of increasing terminal number. The first error found is returned to *error out* and the remaining terminals are ignored. If no errors are found then the first warning is returned, if present, or no error. This VI can be found on the *User Libraries* menu on the **Function palette**. This VI is used in many VIs as sub-VI and also on the main Block Diagram of MXGS DPU Emulator program.

Simple Error Handler.vi: This VI indicates the presence of any error. If an error occurred, this VI returns a description of the error and optionally displays a dialog box. The dialog box message describes the error code and a description of the source of the error. This VI came with the base package of the LabVIEW software and can find on the function palette.

5.4. How to operate the observation and configuration program of the MXGS instrument - A simple tutorial

Before running the program, make sure that detector is connected with the PSU and the DPU. Also check that the DPU is connected with the PC on which above mentioned LabVIEW program is to be run. Ensure that the PSU power is on. Now open the LabVIEW program and run the VI named "DPU Emulator 1_82 Top Level VI" in the project. To configure the ASICs, click on "*Configure ASIC*" button on the control panel. A window/ interface will be pop-up with a name "XA 1.82 Registers". On this window user can configure any desired ASIC or all the ASICs in a particular DM chain. Users can also choose the old stored configuration setting by pressing "*Read File*" button on the right-bottom corner or can configure different ASIC chips by changing the pixels value by 'click and choose' method on

different green-buttons. By pressing the "*Configure detector chain*" button first and "*Exit*" button in last, this user interface will be close.

To send the instructions/commands to the RCU, user has to select any module from the RCU firmware and control register in that module. To do this click and choose the control named "*Module*" and "*Register*" from the control panel. By choosing module and register, user has selected/chosen an address of the RCU firmware where some command has to be send. Also user can see some instructions/command descriptions in front of the numbered button ("0"-"7"). Users have to press one or more numbered buttons according to his requirement after reading the instructions/command descriptions in front of each numbered button and then have to choose "Memory write" (default value) or "Memory Read" by clicking on the button on the control panel. By pressing the Numbered buttons, user has sent data in a binary form to the RCU by changing the bit value on the selected address. This has been done with the help of Memory Read or Memory Write command depends what the user has selected. The Data and Address value will reach inside the RCU firmware after pressing "*Send Command*" button on the control panel. After sending command, the user can also check the response from the RCU on the control panel in front of "Response".

After setting the "*time target*" and "*size target*" or keep the default values, press "*Receive*" button. A file dialog box will pop-up to ask the user to choose the file or directory path to save the Science Data Packets receive from the detector. After selecting a new file or an existing file, the file dialog box will disappear. The SCDP will start collecting and storing in the selected file and start displaying on the four different plots on the Front Panel. Make sure that the "*update graphs*" button is ON

Just one click on the **"***QUIT***"** button at the right-bottom corner of the Front Panel will stop running the program and exit the main program.

Chapter 6
Improvements in the observation & configuration LabVIEW program

This chapter describes the improvements made in the observation and configuration LabVIEW program. A new, more user-friendly CSCI (control and status command interface) graphical interface has been developed for both detectors where the user has many new options to send commands to the MXGS instrument and verifies them by comparing with the RCU response. The other improvement is that the LabVIEW program is now capable of configure, observe and analyze both the CZT and the BGO detectors. The full description about implementation of these improvements and architectural description of the modified LabVIEW program is described in this chapter.

6.1. User friendly control and status command interface

The purpose of the control and status command interface (CSCI) in LabVIEW program is to access the embedded control registers (CRs) and status registers (SRs) in the RCU firmware. These registers are used to trigger firmware functions, define the operation modes and provide the housekeeping information.

As per the requirements, a new easier and user friendly CSCI interface is developed where the user can access the control and status registers just by clicking some buttons thereby trigger various firmware functions. Sending commands to the RCU is made very easy as each button shows what command it asserts. Depending on the selected detector mode, a new GUI interface will appear when the user press "**Send Command**" button on the front panel of the main program. The new GUI interfaces in case of the CZT detector and the BGO detector are shown in figure 6.1 and figure 6.2 respectively.

In case of the CZT detector, a 'Tab Control' function in LabVIEW with five different tabs is used for its construction. The first four tabs with Tab-headings as 'DM_XA_1, DM_XA_2, DM_XA_3 and DM_XA_4' consists of eight input buttons each. These eight buttons can send command to different registers in four DM_IF modules and four XA_CFG modules in the

CZT-RCU firmware. The fifth Tab with Tab-heading 'RCU Master' has two buttons to send commands to the registers in RCU_master module of the CZT-RCU firmware.

In case of the BGO detector mode, the GUI interface is consisting of four tabs with Tab heading as 'PMT_IF_0, PMT_IF_1, PMT_IF_2 and TMON'. The first three tabs have six input buttons each, capable of sending commands to the different registers in three PMT_IF modules in the BGO-RCU firmware. The fourth tab with Tab heading 'TMON' has three input buttons to trigger the functions in TMON module (temperature monitor) in the BGO-RCU firmware.

By clicking on different buttons on different tabs according to the requirement, the user selects some command data to send to the RCU with some specific addresses in the RCU firmware. The detailed information related to the buttons on both CSCI interfaces is provided in Appendix-A.

6.1.1. Different ways to send data to the RCU:

The user can send control commands to trigger any function in the RCU firmware in three different following ways. The users can send

- ✓ Direct commands to the RCU after selecting/clicking on the different buttons on different tabs according to the requirements and then by pressing the '*Send direct to RCU'* button. or
- ✓ By writing a 'Configuration Register Data' file by clicking on *'Write Register's File'* button in the right side after selecting/clicking on the different buttons on different tabs and then by pressing the '*Send New File to RCU*' button. or
- ✓ Any old written register file by clicking on '*Select old File'* button and then send to the RCU by pressing '*Send direct to RCU*' button. Also whenever the user selects any old file to send to the RCU, the user has an option to confirm/check the contents of the file by looking/clicking on all tabs, as buttons representing data in file become ON (green) on all tabs automatically. The user can make some changes on different tabs or can send the data as it was, either directly by clicking on '*Send direct to RCU*' button or can write the new 'configuration data register' file by clicking on *'Write register's File'* button.

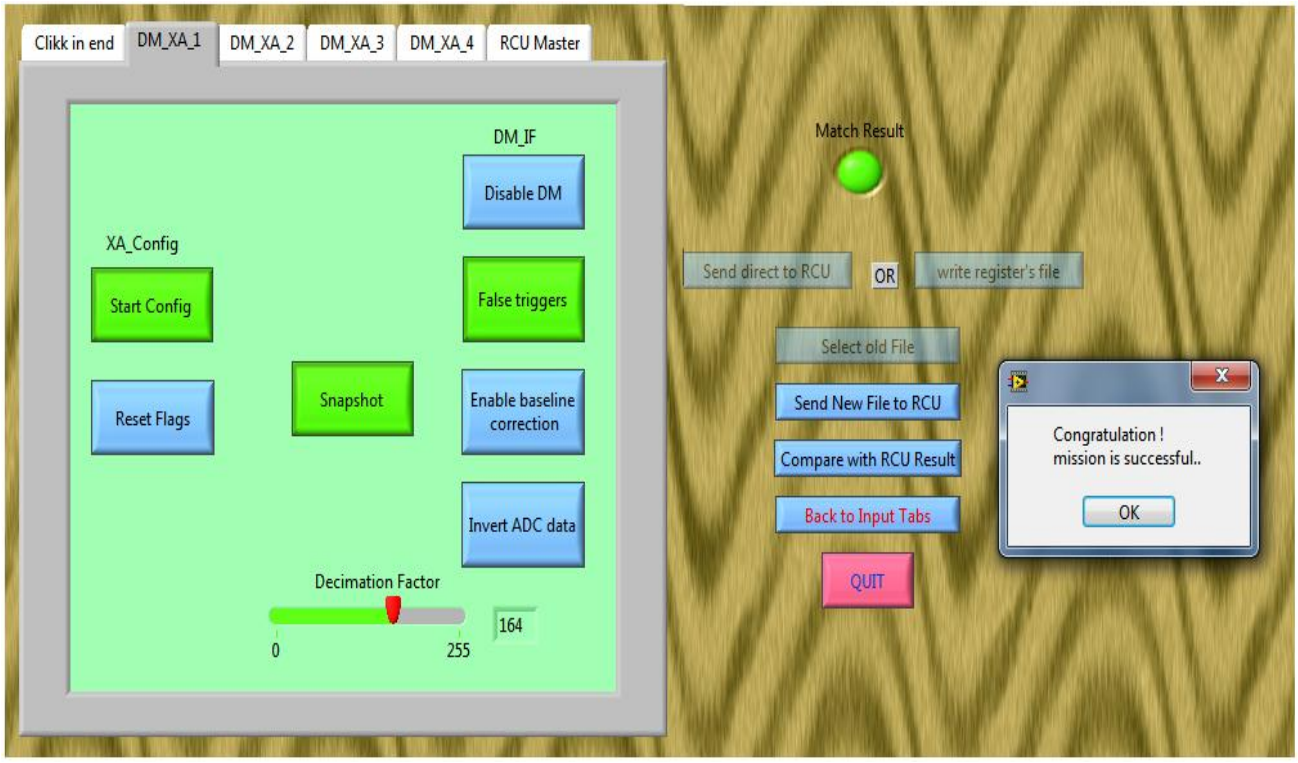


Figure 6.1: The Front panel view of the CZT_RCU_control interface.vi when the match result of the configuration register data and status register data is positive.

6.1.2. Verification of the RCU output with input data

The new CSCI interfaces have a feature to compare the RCU input with the RCU output and verify the result. Whenever the user sends any command data to the RCU, the system sends a response data back to the user (RCU to DPU) as defined in the firmware.

The program is developed in such a way that in these new CSCI interfaces, whenever the users send commands to the RCU, a file prompt will appear and asking the user for a file location to write the RCU response in a file. This file will contain the RCU response data under the heading 'Status Register Data'.

The next job is to compare the RCU input with its output. The user can compare the 'Configuration Register Data' (user sent data to the RCU) with 'Status Register Data' (the RCU response) by clicking on the '*Compare with RCU Result*' button. The program will ask the user to select one RCU's input file and one RCU's output file and then compares the contents of two selected files. The result will be shown on the CSCI interface by a colored

indicator. If the users sent data and the RCU response are same, verification will be shown by green indicator on the same window with a prompted 'congratulation message' (as shown in figure 6.1), otherwise a red indicator and prompted 'sorry message' will be appeared. Furthermore, in case of the match result failed, the '*differences of file A from the B*' and '*differences of file B from A*' will also be appeared on the front panel as shown in figure 6.2.

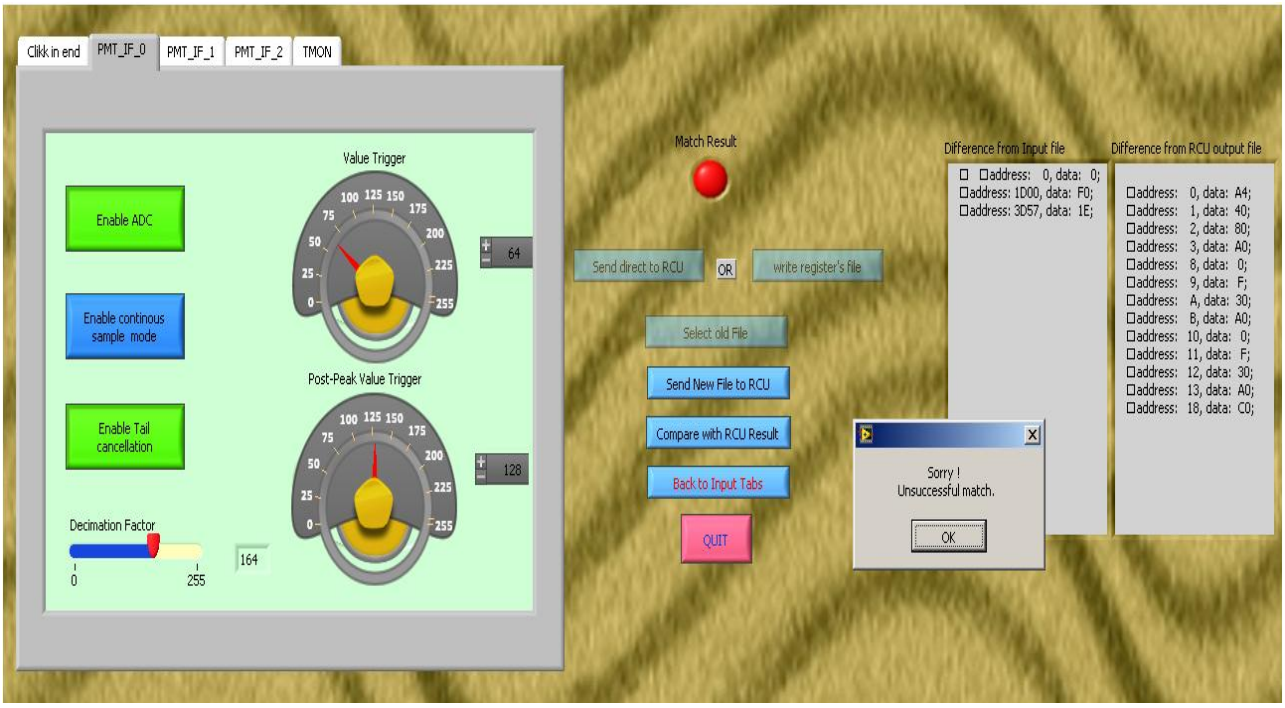


Figure 6.2: The Front panel view of the BGO_RCU_control interface.vi when the match result of
the configuration register data and the status register data is negative.

6.2. Compatible with both CZT and BGO detector

Another improvement in the observation and configuration LabVIEW program is made by making it compatible with both the CZT and the BGO detectors. Now the program is also capable of sending commands to the BGO-RCU and analyzes the collected measurement data from the BGO detector. The separate CSCI interface for the BGO detector is developed as shown in figure 6.2 where the user can access the BGO-RCU. A separate histogram is made for the BGO collected data and that measurement data is displayed on the front panel of the main program on different plots compare to the CZT's plots. Five plots are created to display the BGO-histogram data as shown in the figure 6.3.

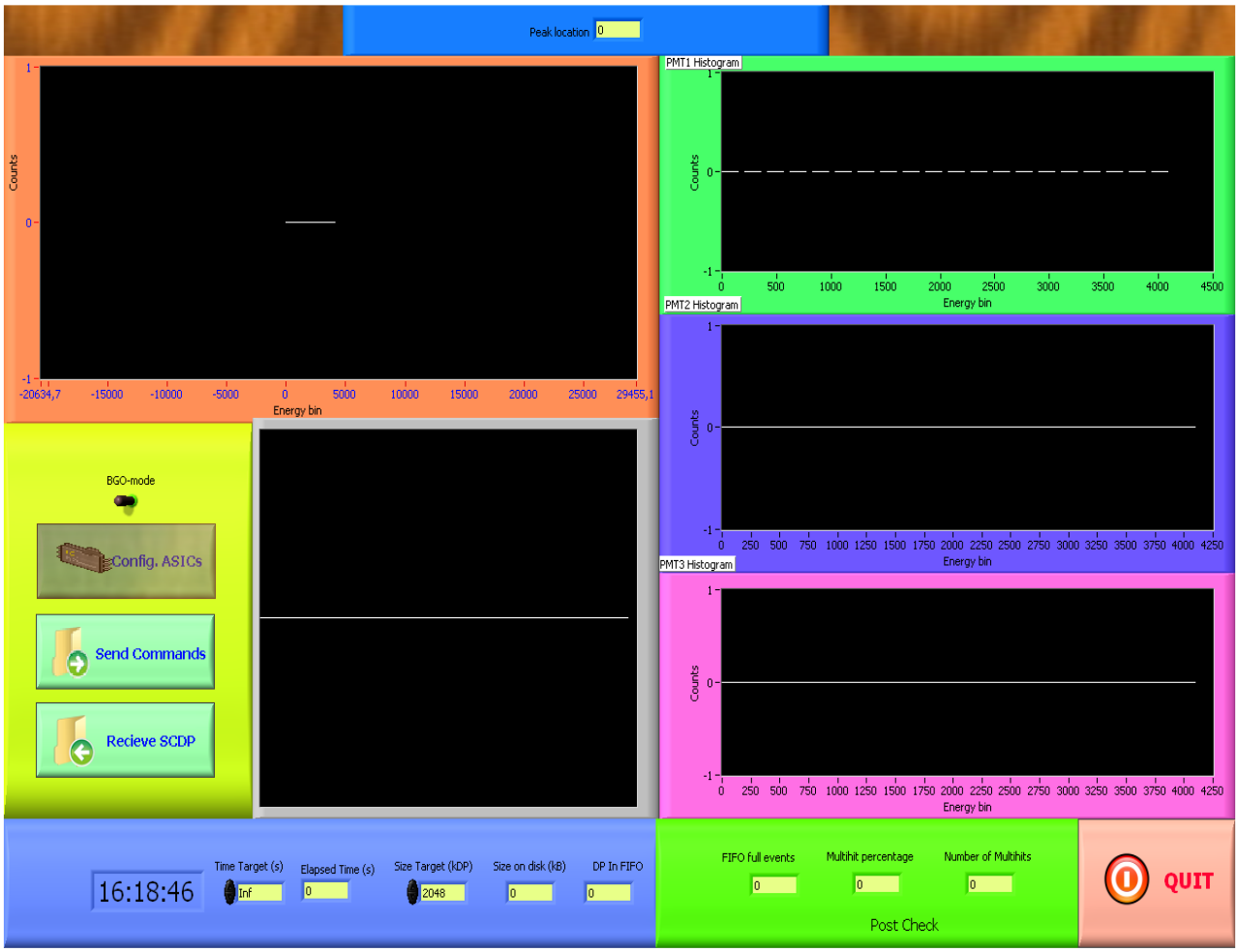


Figure 6.3: The Front Panel view of the main program in case of the BGO detector mode.

The plots are:

1. PMT1 Histogram: display the counts and energy of all the hits detected by first Photo Multiplier Tube Interface in the BGO detector module.
2. PMT2 Histogram: display the counts and energy of all the hits detected by second Photo Multiplier Tube Interface in the BGO detector module.
3. PMT3 Histogram: display the counts and energy of all the hits detected by third Photo Multiplier Tube Interface in the BGO detector module.
4. Overall Histogram: display the total counts and energy values observed by all three Photo Multiplier Tube interfaces. This plot also provides the peak location for the highest number of counts.
5. Fast Time-Tag: display the fast time tag data.

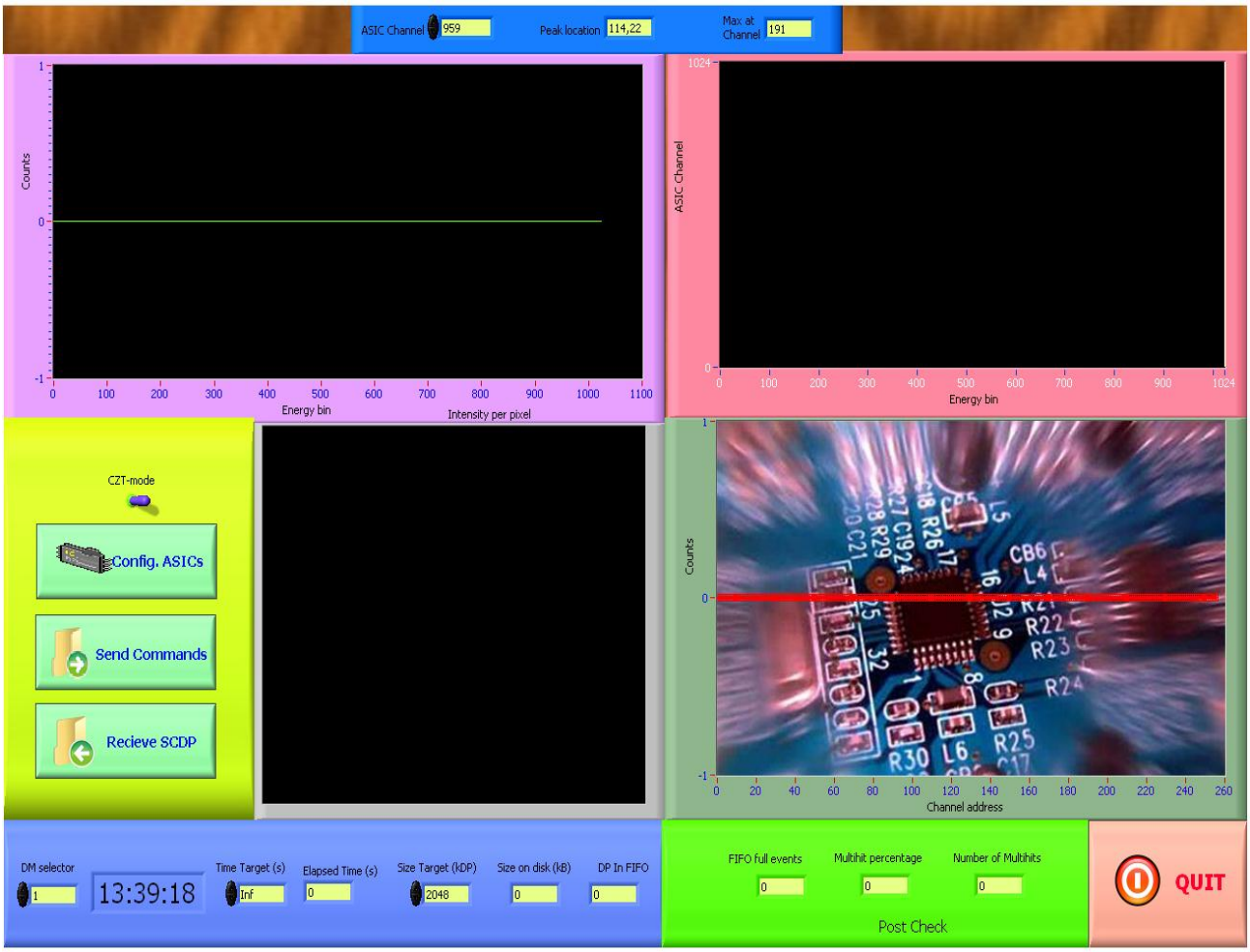


Figure 6.4: The Front Panel view of the main program in case of the CZT detector mode.

In case of the CZT detector mode, the following four plots are displayed on the front panel of the main program as shown in figure 6.4. The more detail about the CZT-plots is provided in Chapter 5, section 5.3.3.

1. Number of counts for single channel plot: It shows the number of counts for a user selected channel, which also gives the peak location.
2. Intensity per channel plot: It is created between numbers of count per energy versus the channel address. The color indicates the intensity. The lighter the color of spot is, the more events have been counted in the energy bin.
3. Intensity per pixel plot: It shows the intensity of the all the pixels at their physical position in a user selected DM. Again the color indicates the quantity of counts on the pixel.
4. Fourth plot shows the total number of counts with respect to channel address in a user selected DM.

The program is made in such a way that it displays only plots related to the selected Detector Mode and hides other plots on the front panel of the main program.

6.3. Full description of the architectural implementations of the improvements

As discussed above, the improvements have been made in the observation & configuration LabVIEW program. The description of the architectural and functional implementation of these improvements in the main program is discussed in detail in this section.

As it is discussed in section 5.3 in Chapter 5, the main program consists of three parallel loops, i.e. the producer loop, the display loop and the consumer loop, performing various functions and has different performance priorities. In this improved program, the main changes have been made in the producer loop and the display loop. Only a few changes have been made in the consumer loop. The modifications in these loops are described in detail in following sections.

6.3.1. Changes in the Producer Loop

As mentioned in chapter 5, the main objective of the Producer Loop is the production and acquisition of data from the Opal Kelly XEM module. The loop has the highest priority which controls the sends the data to and from the XEM module.

Comparing to the previous program, the producer loop of this improved program has fewer numbers of states as it is made as a state machine with following states: Idle (Default), Command to RCU, Initialize, Configure XA Registers, Initialize Get Science Data, Get Science Data, Exit Get Science Data and ShutDown.

The changes and improvements in the different states of FSM in the Producer Loop are described below.

1. Idle

This is the default state for the state machine where the program waits for next instruction. If the user presses any one of these, “Send Command”, “Config. ASIC”, “Receive “or “Quit” button, the program goes to next state according to pressed button.

The Idle state of the FSM consists of "Send Command", "Config. ASIC" and "Quit" buttons available on the front panel of the main program. The functioning of these buttons is possible because of 'Select' function in LabVIEW, which is discussed in Chapter 5. The "Receive" button is actually placed outside the Idle state because it functions on many states.

2. Command to RCU

The main function of this state is to send commands to different modules in the RCU firmware by sending address and data bits. The main changes in the program are made in this state. The user friendly CSCI (control and status command interface) graphical interface and the switching detector modes (CZT \ BGO) are developed in this state. The *CZT_RCU_control interface.vi* and the *BGO_RCU_control interface.vi* are two sub-VIs placed in this state in a case structure controlled by detector mode switch. The Detector Mode switch is visible as a toggle switch on the front panel of the main program as CZT-mode/BGO-mode.

- **The *CZT_RCU_control interface.vi*:** This VI is created to send commands and acquire data to and from the RCU firmware in CZT detector. The RCU firmware includes the DM-interface, XA config and the RCU Master modules. Four DM_XA_1 to DM_XA_4 tabs are made by using *tab control* function in LabVIEW as there are four DM interfaces and four XA_CFG modules defined in the RCU firmware. The fifth tab 'RCU Master' is made separate because of the only RCU-Master module defined in the RCU firmware.

  Each of the DM_XA_1 to DM_XA_4 tab consists of some click buttons. These buttons represent some control command data to specific addresses in the four DM_IF and four XA_CFG modules of the RCU firmware.

The detailed architectural description of the *CZT_RCU_control interface.vi* is discussed in section 6.4.

- **The *BGO_RCU_control interface.vi*:** This VI is created to send commands to the PMT_IF (Photo Multiplier Tube Interface) module and TMON (Temperature Monitor) module in the RCU firmware of the BGO detector. The PMT_IF firmware modules collect and process the data coming from the ADCs in BGO detector. There are three PMT_IF modules defined in the BGO-RCU firmware, one for each readout chain. On the other hand, TMON module

gathers and analyses temperature values from the PM tubes used for temperature compensation in the PMT_IF modules. Because of three PMT_IF and one TMON interfaces in firmware, four tabs i.e. PMT_IF_0, PMT_IF_1, PMT_IF_2 and TMON are created by using *tab control* function in the LabVIEW program.

The three PMT_IF tabs contain five buttons each and fourth tab contains only three buttons, which can trigger some functions by sending commands to control registers defined in the BGO-RCU firmware.

The detailed architectural description of the *BGO_RCU_control interface.vi* is discussed in section 6.5.

The next state is Idle.

3. Initialize

This state initializes the XA_1.82 registers's data. No new modification is made in this state. More detail about this state is discussed in Chapter 5, section 5.3.1.

The next state is Idle.

4. Configuration of XA Registers

The ASICs configuration activates the ASICs and makes them ready for processing the event signals. More detail about this state is discussed in Chapter 5, section 5.3.1.

The next state is Idle.

5. Initialize Get Science Data

This state initializes the values and variables that are necessary for the acquisition of data from the RCU. More detail about this state is discussed in Chapter 5, section 5.3.1.

6. Get Science Data

This state updates the RCU outputs and acquires data from the RCU. If there is enough data in the FIFO, it reads the data otherwise shows FIFO data packet count value on the front panel of the main program. More detail about this state is discussed in Chapter 5, section 5.3.1.

7. Exit Get Science Data

This state stops the loading of science data packets into the FIFO and closes the queue.

The next state is Idle.

8. ShutDown

This state shut down the connection with the RCU, shutdown the file loop queue and ends the state machine loop.

6.3.2. Changes in the Consumer Loop

The consumer loop handles the data packages received from the producer loop and sort the science data into a histogram array for further use in the display loop. As discussed already about the consumer loop in chapter 5, the only change is made in the 'write' state of this loop. Two histogram arrays according to detector mode, i.e. CZT histogram and BGO histogram are made in this state. If there are less than 25 arrays or elements in the queue, the science data is sorted in any one histogram depending on the detector mode which is read by the display loop.

6.3.3. Changes in the Display Loop

Some changes have been made in the display loop. Now the display loop is capable of displaying all the measurement data detected by the CZT as well as the BGO detector on the front panel of the main program. In the CZT detector mode, four different plots display the collected data on the front panel, whereas five different plots are created in the BGO detector mode to represent the collected data. The display loop displays plots depending on the selected Detector Mode.

The 'Case Structure' function with two cases controlled by the 'Detector Mode' switch is used to make the display loop. In the first case, the science data packets from the "Histogram (ADC chnl, ASIC chnl)" global variables are sorted and the four plots are created and displayed on the front panel of the main program as discussed thoroughly in Chapter 5.

In second case of 'Case Structure' function, the science data packets from the "BGO" global variable are sorted, and the five plots are created and displayed. The "BGO" global variable consists of two types of data, i.e. energy data array and fast time-tag data array. The first three plots are created with the Energy data array whereas the fourth plot is plotted with the Fast Time-Tag data array. The plots are discussed in section 6.2.

The 'Property Node' function is used repeatedly on different controls and indicator to make the front panel simpler and effective. For example, all the plots of the BGO detector mode are made invisible when the CZT detector mode is ON and vice-versa. The '*Config. ASIC*' button

on the front panel of the main program in the BGO detector mode is disabled and grayed whereas enabled in the CZT detector mode with property node function in LabVIEW.

The global variable 'Quit' is also present in the display loop. Click on the Quit button on the front panel of the main program quits the display loop as well as the main program.

6.4. Architectural description of the CZT_RCU_control interface.vi.

The *CZT_RCU_control interface.vi* is used to send control commands to the CZT-RCU firmware, to get the RCU response as status register's data and to compare the input and output of the RCU in the CZT detector. The front panel of this VI is shown in figure 6.1.

The *CZT_RCU_control interface.vi* is a self-executable VI made as FSM with following states: Initialize, Idle, Direct Data to RCU, Write Registers in File, File Data to RCU, Read from RCU, Compare Input with RCU output, Select file for input buttons, Match File to input buttons and Shutdown. The block diagram-snapshot of this VI is shown in figure 6.5.

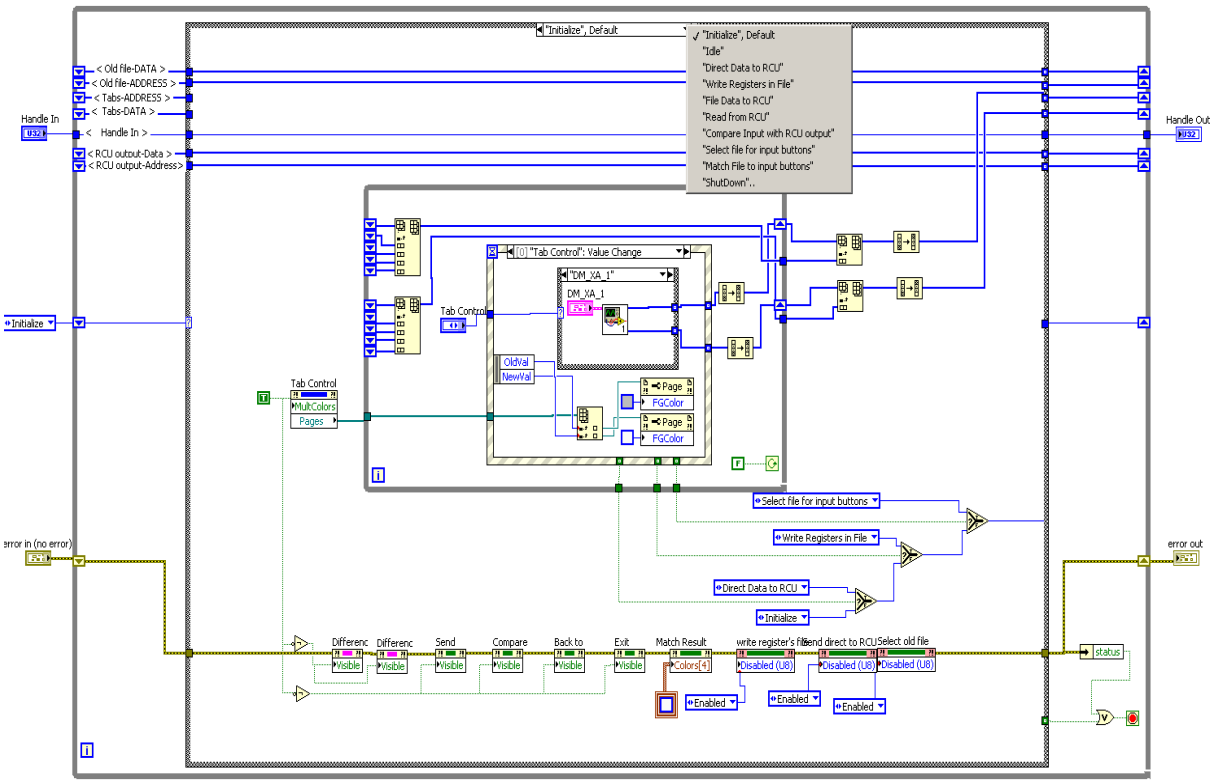


Figure 6.5: A snapshot of Initialize state of the CZT_RCU_control interface.vi.

The various states are described below.

1. Initialize

Initialize state is the default state of the FSM which means it runs as the first state when this VI runs. An 'Event Structure' with five different events inside the 'While Loop' is placed in this state. A 'Tab Control' function with five different tabs is placed in the first sub-diagram of an event structure. The input clusters DM_XA_1, DM_XA_2, DM_XA_3, DM_XA_4 and RCU Master (inputs of the DM_XA_RCU master_1.vi, DM_XA_RCU master_2.vi, DM_XA_RCU master_3.vi, DM_XA_RCU master_4.vi and RCU_buttons.vi VIs) are placed on the different tabs of tab control as shown in figure 6.1.

The input clusters DM_XA_1, DM_XA_2, DM_XA_3, DM_XA_4 and RCU Master are actually switches/buttons, which are available on the front panel of the *CZT_RCU_control interface.vi.* The shift register function is used to collect the selected information on different tabs (when the user press some buttons on different tabs) and then two arrays are created and stored in another shift register for further use.

The *'Write Register's file', 'Send direct to RCU'* and *'Select old file'* are three control buttons on the front panel which are placed in three consecutive sub-diagrams of the event structure. The fifth sub-diagram is related with *'EXIT'* ('*QUIT'* on the front panel) button. A display message to the user is also placed in the fifth sub-diagram which appears when the user press *'QUIT'* button on the front panel without selecting any of Tab Control's button. The '*EXIT'* is placed outside the states of the FSM because it functions from more than one state. When the user presses any of these three buttons, the 'Select' function set the FSM to the related state.

The property node function is used repeatedly to enable or disable and visible or invisible some switches/buttons to make the front panel simpler and user friendly. A 'Property Node' function on Tab Control with 'colors' as property is used to change the color of selected tab on the front panel.

2. Idle

The program waits for next instruction in Idle state if nothing happens. The *'Send New File to RCU', 'Compare with RCU result'* and *'Back to Input Tabs'*, are three control buttons which also available on the front panel of this VI, exists in this state. Clicking

on any of them put the FSM into that state. The '*EXIT'* ('QUIT' button on the front panel) button can also put the FSM into 'Shutdown' state in this state.

3. Direct Data to RCU

The user selected data on different tabs in Initialize state is stored in shift registers. The data from saved arrays are then sent and write one by one in the RCU with memory write command through *Send data to RCU.vi*. This is implemented by using 'For Loop'. The data sent to the RCU consists of two types of data. It consists of command data and address data. The command data is converted into Boolean array before send to the RCU. The RCU provides a response through *read from RCU.vi* and is saved in shift register for further use.

The 'Property Node' functions are used to initialize the values of all the buttons on different tabs as well as to visible and disable some other control button to make easier for the user.

The 'Read from RCU' is the next state.

4. Write Registers in File

Instead of sending selected commands and address data directly to the RCU in the previous state, the user can write that data into a text file in this state. The 'Build Text' function is used to convert the selected data array into a hexadecimal string. A 'File Dialog' function is used so a file dialog will be prompted to ask the user to select a file/file location to write. The 'Write to Text File' function writes the hexadecimal strings into the selected file one by one under the heading "Configuration Register Data".

Again, the 'Property Node' functions are used to initialize the values of all the buttons on different tabs and to make some side buttons visible and disable to create a user friendly interface. The error with error code 43 which arise due to cancellation of the file dialog is eliminated by using 'General Error Handler.vi' and some trick.

If the prompted file dialog is canceled, the Initialize state will be the next state otherwise Idle state will be the next state.

5. File Data to RCU

This state is created to send data to the RCU, in case the user wants to send the configuration register data file to the RCU.

The File dialog function is used so that a file prompt will appear when the FSM runs this state. The user can select a file from any file location to send to the RCU. The file is first read with 'Read from Text File' function. As the file contains register's address information and command data bit information, so both are separated with 'Pick Line' function and then converted from hexadecimal string to number. The command data bit information is converted into a Boolean array and then input to '*Send data to RCU_1.vi*' with address data. The command data bit information on the corresponding addresses will write with Memory Write Command in '*Send data to RCU_1.vi*'. The '*new read from RCU_1.vi*' provides the RCU response which is stored in shift register for further use in the next state.

A file error with error code 43 which generally arise on cancellation of file prompt is eliminated with 'General Error handler.vi' function and some tricks.

If the user cancels the file prompt, the Idle will be the next state otherwise 'Read from RCU'.

6. Read from RCU

In this state, the RCU response is written into a text file for further verification.

A file dialog function is used so the user gets an option to choose a file/file location to save the RCU response data. The RCU provides two data arrays. The RCU response data array is first converted into a hexadecimal string and then written into the selected file by 'Write to Text file' function under 'Status Register Data' heading.

A file error with error code 43 is eliminated in this state as well.

The next state is Idle.

7. Compare Input with RCU output

In this state, the configuration register data is compared with the status register data which verify of functionality of the RCU. Two 'File Dialog' functions are used to ask the user to choose one configuration register data file and one status register data file. Files are first opened. File position is set to start and data is read from the files with 'Read from Text File' function. The data strings of files are then compared by using 'Variant' function. If the data strings of both files are same, a message dialog with 'Congratulation' message will be appeared and match-result will be verified with green color indicator on the front panel of this VI. In second case, a 'sorry' message and red indicator will verify the failed match result. The 'difference of file A from B'

and ‘difference of file B from A’ will be appeared on the front panel of this VI in case of the match result failed. The Property Node functions are used to make the indicator red and green as well as to appear and disappear the differences of two files.

In case of any file dialog cancellation, the error 43 is eliminated in this state as well.

The Idle is the next state.

8. Select file for input buttons

In this state, the user can choose any previously written configuration registers’s data file to send to the RCU. A file dialog function provides an option to the user to select any desirable, old configuration register data file. The file is first open and then data is read with ‘Read from Text File’ function. The ‘address data’ and ‘command data’ are separated, and two data arrays are created. Each array data is converted into ‘Number’ from hexadecimal strings. Both number arrays are saved in shift register for further use.

The *‘Send New file to RCU’*, *‘Compare with RCU Result’*, *‘Back to Input Tabs’* and *‘Quit’* buttons are enabled with ‘Property Node’ function in this state. The error 43 is also eliminated in this state in case of file dialog cancellation.

If the user cancels the prompted file dialog, the Initialize state will be the next state otherwise ‘Match File to input buttons’ state.

9. Match File to input buttons

In this state, the previously selected file data (configuration register data) is set and represent automatically onto the buttons of different tabs in Initialize state. The ‘Case Structure’ function with some conditions is used to represent data back onto each button. If the ‘Address data’ in the file is equal to given-specific address (according to the firmware), then the button belongs to that address is made ON/OFF according to ‘Command Data’ in the file with ‘Property Node’ function. Another ‘Case Structure’ function is used to make it possible for different combination of ‘Command Data’.

This is possible because each button on the Tab gives two types of information, i.e. ‘Address data’ and ‘Command data’.

Eighteen such cases are made for different data combinations in the selected file to represent file data back onto the buttons on five tabs.

The Initialize is the next state.

10. ShutDown

This state stops the functioning of the loop and the FSM shutdown.

6.5 Architectural description of the BGO_RCU_control interface.vi.

The *BGO_RCU_control interface.vi* is used to send commands or data bits to the BGO-RCU firmware, to get the RCU response as status registers data and to compare the input and output of the RCU in the BGO detector. The front panel of this VI is shown in figure 6.2.

The *BGO_RCU_control interface.vi* is also a self-executable VI made as FSM with same states as in the *CZT_RCU_control interface.vi.* The construction and functioning of this VI is almost same as the *CZT_RCU_control interface.vi* except the Tab buttons in 'Initialize' state.

The Initialize state is the default state of the FSM. A 'Tab Control' function with four different tabs is placed in the first sub-diagram of an event structure. The input clusters PMT_IF_0, PMT_IF_1, PMT_IF_2 and TMON (input clusters of PMT_IF_0.vi, PMT_IF_1.vi, PMT_IF_2.vi and BGO_TMON.vi Vis) are placed on the different tabs of 'Tab Control' function.

As shown in figure 6.2, the input clusters PMT_IF_0, PMT_IF_1, PMT_IF_2 and TMON are made of some switches/buttons which are visible on the tabs on the front panel of the *BGO_RCU_control interface.vi.* The shift register function is used to collect the selected information from all tabs when the user press some buttons. All the buttons on different tabs are set to their default values according to the BGO-RCU firmware.

The '*Write Register's file', 'Send direct to RCU'* and '*Select old file*' are three buttons on the front panel which are placed in the sub-diagrams of the event structure. Whenever the user presses any of these three buttons, *'Select'* function set the FSM into the related state.

The property node function is used on different controls and indicators repeatedly almost in every state to make them enable or disable and visible or invisible. This makes the front panel of this VI simpler and more user-friendly.

Rest of the FSM states and their functioning in the *BGO_RCU_control interface.vi* is almost same as states of the *CZT_RCU_control interface.vi*, which are described thoroughly in section 6.4.

Chapter 7

Testing and Verification

This chapter provides the description of testing the LabVIEW program developed for observation and configuration of the MXGS instrument inside the Lab. The block diagram in figure 7.1 gives an idea of the testing setup in case of the CZT detector by using the LabVIEW program in the Lab.

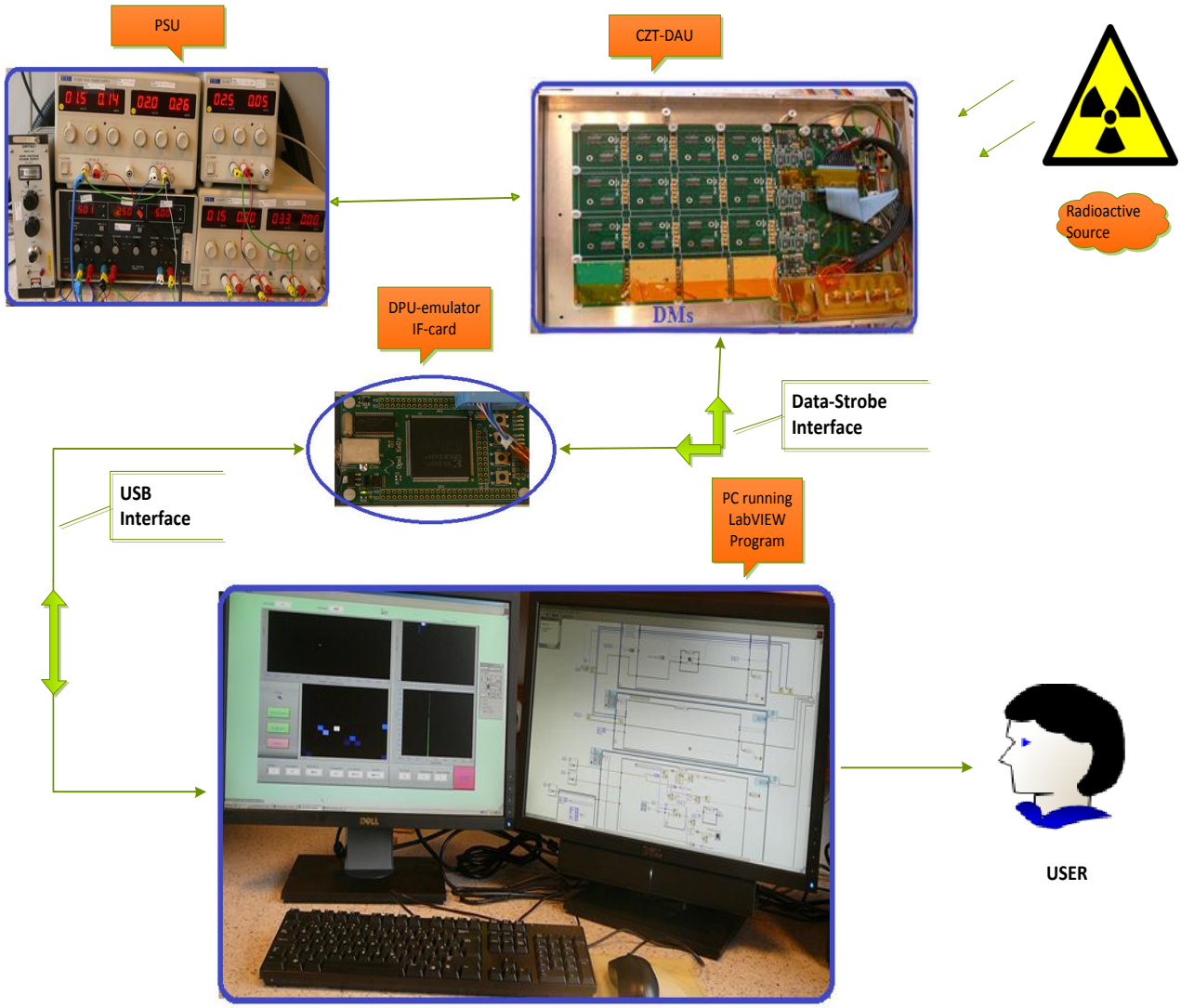


Figure 7.1: The block diagram view for testing setup of the CZT detector inside the Lab.

The testing of the LabVIEW program is started by connecting the CZT detector assembly unit with the PSU and the DPU, and then the DPU with PC having observation and configuration LabVIEW program as shown in figure 7.1. An artificial radioactive isotope **Americum** (241_{Am}) is used as a radiation source on the detector. Americium has a half life of 472.5 years and decays with the emission of Gamma rays. Americium is made in nuclear reactors and is a decay product of plutonium-241. Americium-241 is used in most of the smoke detectors.

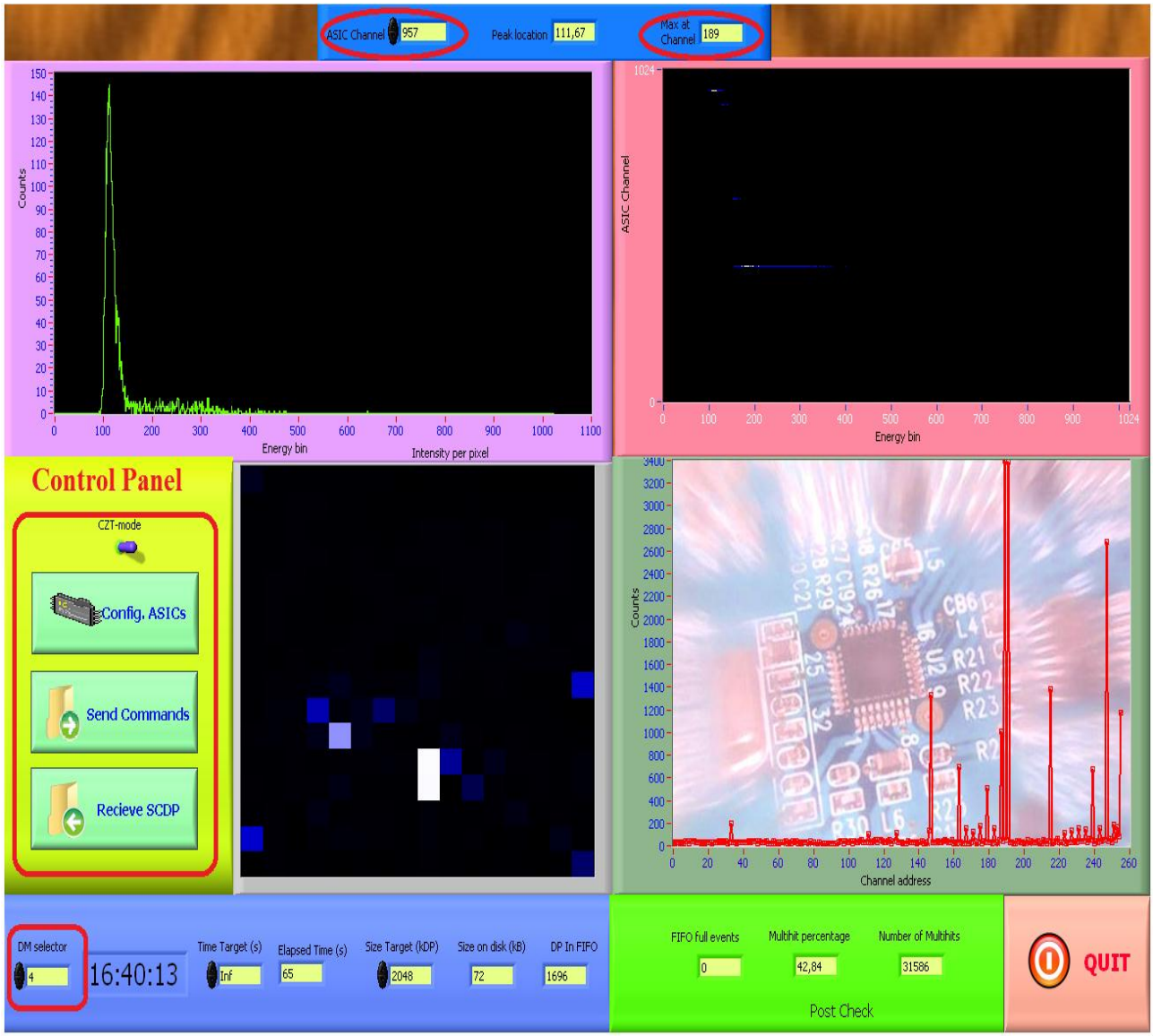


Figure 7.2: The Front panel interface of the observation and configuration LabVIEW program.

The LabVIEW program on the PC is made run by Ctrl+R command. The Control Panel on the front panel interface of this program is made to configure the XA-ASICs on the detector

assembly unit, to send commands to the detecter/RCU and to receive the science data from the detector /RCU as shown in figure 7.2.

7.1 ASICs configuration

The first important task of the developed program is to configure the ASICs which activate the ASIC modules and make them ready for processing the event signals.

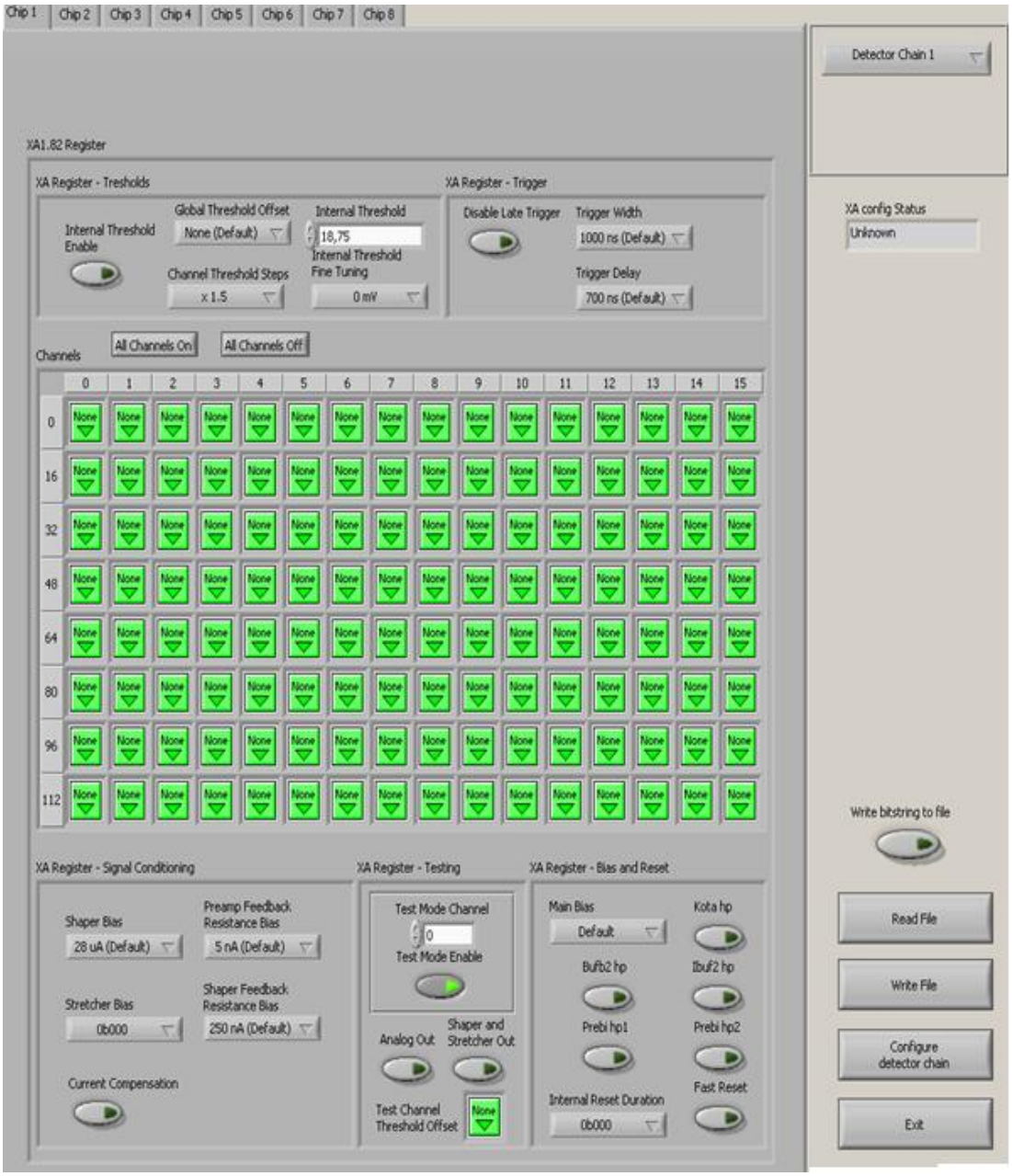


Figure 7.3: The XA ASICs configuration GUI interface.

The configuration of the ASICs is necessary after each restart as ASICs have no memory to store the configuration data bits. By pressing the *'Config. ASIC'* button on the control panel of

the main program, a new GUI interface pop-ups as shown in figure 7.3. The configuration setting can be made by clicking different buttons on this GUI and saved in a text file (.xar format) by pressing the 'Write File' button. By pressing the *'Configure detector chain'* button, the current values selected on the interface will be converted into a bit stream and sent to the ASICs. After configuration, the ASICs are ready to process the charge pulses created by photon events.

The *'Exit'* button hides the XA-ASICs configuration GUI interface.

7.2 Send Commands

The second main task of this developed LabVIEW program is to send commands to the RCU firmware. The *"Send Command"* button on the control panel is used to send commands. By clicking on this button a new GUI interface appears as shown in figure 7.4.

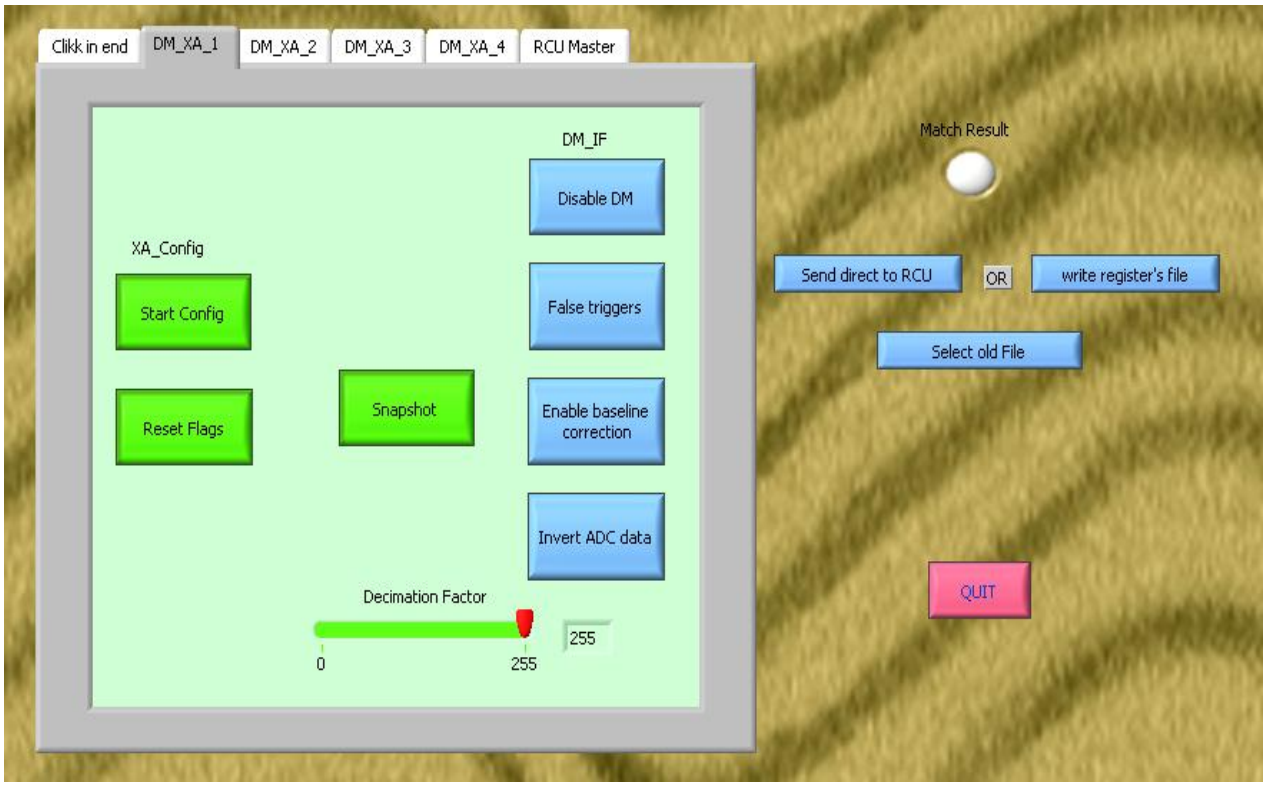


Figure 7.4: The GUI interface to select and send the control commands to the RCU.

By selecting any tab button, the various control command buttons on that tab will appear on the tab. By clicking on *'Start Config'*, *'Decimation Factor'* and *'Snapshot'* buttons, three control commands are selected to send to the RCU (see Appendix-B). The next three tabs also

contain the same command buttons but for different register addresses in the RCU. We selected commands only on the DM_XA_1 tab because we have the only DM readout chain available on the DAU. Rest of the tabs are clicked without selecting any command button. By pressing the *'Write Register's File'* button, the selected command data is stored in a text file named *'CZT-Input registers data.txt'*.

After writing the control command data in file, some new buttons appear on this GUI interface. By selecting the *'Send New File to RCU'* button, file data is written into the RCU control registers with memory write command.

7.3 Verifications

According to the firmware, the system always responses back when control command data is send to the RCU. In response to the '*CZT-Input registers data.txt'* file data, the RCU generates and sends same data back, which is stored in another text file named *'RCU output data (CZT).txt'* as status register data.

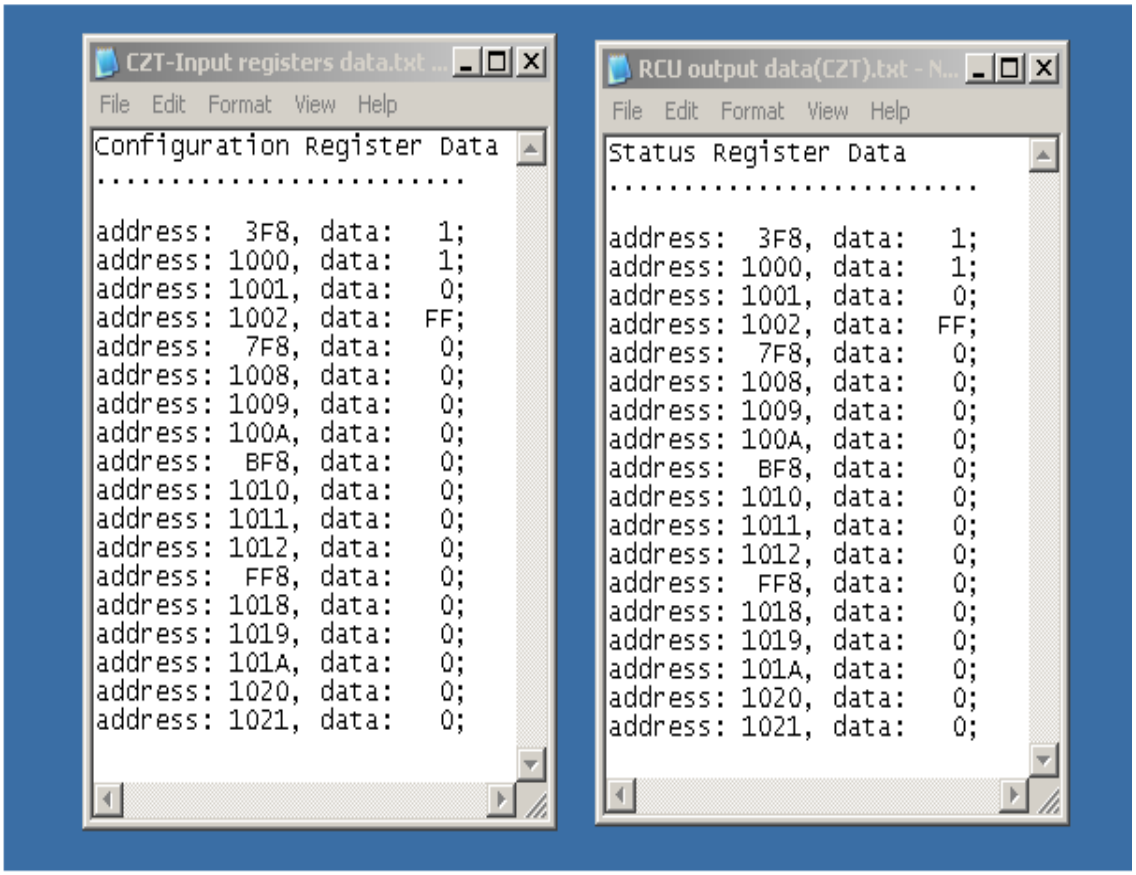


Figure 7.5: The snapshot of the RCU input and the RCU output files.

Figure 7.5 shows the contents of the *CZT-Input registers data.txt* and the *RCU output data (CZT).txt* files. By pressing the *'Compare with RCU Result'* button, the data from the '*CZT-Input registers data.txt'* and *'RCU output data (CZT).txt'* files is compared and match result is confirmed on the GUI with colored indicator. If match result is positive, verification is done with the green indicator and a congratulation message as shown in figure 7.6. In case of negative match result a red indicator with a sorry message will appear on the same GUI.

Users can return back on to the tabs again, to select other control commands by pressing '*Select to Input Tabs'* button.

The CSCI interface is made disappeared by clicking on 'QUIT' button.

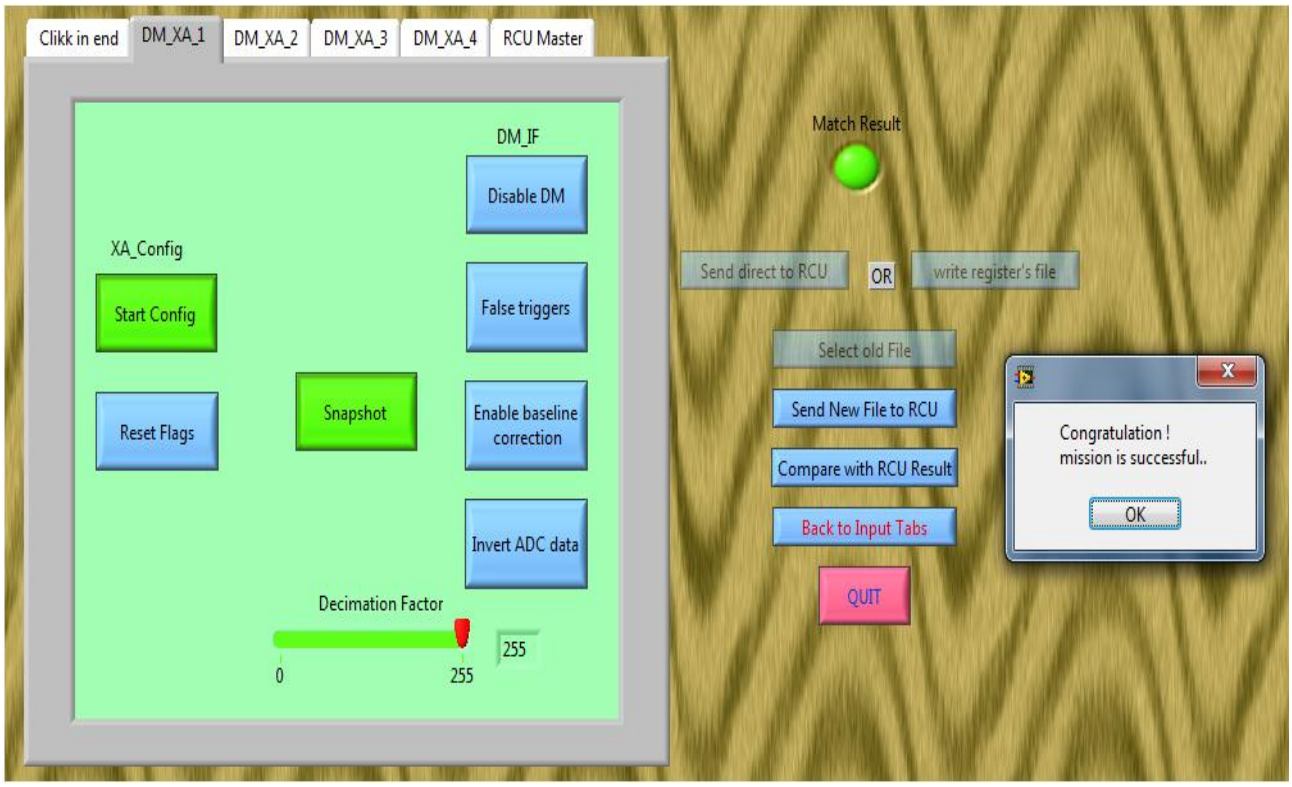


Figure 7.6: Snapshot of GUI in case of the match result positive.

7.4 Data Acquisition and analysis

The third important task of the developed LabVIEW program is to acquire the measurement data and display it in real-time plots, on the PC monitor.

The process of acquisition is started by pressing the *'Receive'* button on the control panel. A storage path is asked to define by the user. After choosing the file name, the data acquisition

is started. The value of currently saved SCDP and *DP in the FIFO* are available on the front panel. The measurement data from the FIFO is displayed on four graphical panels as shown in figure 7.2. The top left graph display the energy distribution of detected events for single selected channel. In figure 7.2, the top left plot displaying the energy and counts distribution for selected ASIC channel number 957. The channel can be selected by *'ASIC Channel'* control placed above this plot. The *Max at channel* indicator shows the channel address value in the selected DM, having the highest number of counts. In figure 7.2, channel address 189 in fourth DM has the maximum number of hits. The top right plot displays the energy distribution for all available channels. The color indicates the intensity of the pixels.

The bottom two graphical plots show energy data for a single selected DM. The DM can be select from the control named *'DM Selector'*. In figure 7.2, these two plots are displaying measurement data for fourth DM in the DM readout chain as fourth DM is selected in '*DM Selector*' control. The bottom right plot shows total number of counts with respect to channel address for fourth DM. The bottom left plot is an intensity plot with respect to pixel address for fourth DM. Each square represents the physical position of the pixel in the DM. The color of the square represents the intensity of the pixel. Lighter the color of the square (Pixel), more the events have been detected at that pixel position. The Post Check data displays the *Number of Multihits* and *Multihit-percentage* and the number of *FIFO full events* during data acquisition, after the completion of SCDPs acquisition.

One click on the *QUIT* button stops the running program.

7.5 Result

The program is tested with one readout ASIC chain having four DMs on the CZT detector assembly unit. The fully populated DAU with 16 DMs is not available yet. Artificial radioactive source *Americium* is used as a source of gamma radiations. The functioning of the program with BGO detector is not tested yet because the fully functional BGO-RCU firmware are not available so far. The program functioned as it was expected and is expected to function properly with the BGO detector board. The satisfactory results are obtained.

Chapter 8

Conclusion and Outlook

8.1 Conclusion

More than 75 research groups from 29 countries have shown interest in the ASIM project which is running under the guideline and financing of the ESA and under the technical leadership of the Danish Aerospace company, Terma. The ASIM instrument will monitor the Earth's atmosphere for X-rays and Gamma-rays in the energy range of 3 keV to 20 MeV after its installation on ISS. The Microelectronics and Space Physics groups at the University of Bergen have responsibility to develop the ASIM-MXGS detector plane and its readout electronics. The ASIM project is in its final stage, phase-C/D. In this project-phase, the flight version module of the ASIM payload is going to develop. For testing the MXGS instrument in the lab at UiB, a readout and configuration program has been developed in LabVIEW. In this thesis, the design architecture and improvements made in the readout and configuration program for the ASIM-MXGS instrument are described.

Each detector assembly unit has detector modules as well as a readout control unit (RCU). A LabVIEW program is designed to make the RCU communication and initialization procedures along with the test procedure, which include features to stream control data, configuration of ASICs and collection and analysis of all received science data from the RCU. The LabVIEW program is capable of creating a communication link between the user and the detector. The user can utilize this program on his PC to send different control commands from the developed CSCI (Control and Status Command Interface) just by pressing some buttons to trigger the various functions in the RCU firmware. A separate GUI has been developed to configure the ASICs in the CZT detector as the ASICs are always in uninitialized state each time the power is switched on. The configuration of the ASICs makes them activate and ready for their functioning on incoming pulses. The science data packets collected by the RCU are made available on the user's PC through this program and further visualized by creating different plots on the monitor. The user can select and check the energy level of any particular ASIC channel in any particular given detector module. The Noise detection is also possible as the channels with Noise show a higher number of counts compared to its surrounding. Also

corner pixels on the DMs are expected to have slightly different behavior than the central pixels.

8.1.1. What does User-Friendly program mean?

As mentioned in Chapter 6, a new and more user-friendly CSCI interface has to be developed in the LabVIEW environment. But what is actually a user-friendly interface or program?

People have tried to come up with a set of laws to define what is user-friendly and what is not. Different people have different opinions regarding the user-friendliness. According to my supervisor, a program must be intuitive to be user-friendly, which means it is easy to use, learn and understand and error-free having understandable error messages and information messages. According to Steve Oualline, the best law is the "Law of Least Astonishment": the program should act in a way that least astonishes the user [38].

According to me, all programs are made to make things easier for users but some programs make the user's job much easier and faster without confusion. This we consider 'better'. That better factor can be termed as 'user friendliness'. According to Galitz & O.Wilbert [39] and web-writer [2]Jack Wallen, a program must have the following characteristics to be user-friendly:

a) Efficient

The program must be efficient which means it must complete the user expected jobs. The developed readout and configuration LabVIEW program for the MXGS instrument is capable to control, readout and analysis, and configure both the CZT and the BGO detectors as per requirements, and therefore defined to be efficient to its jobs.

b) Simple and nice GUI

The program must have a simple and attractive GUI. The main purpose of the GUI is to make the user's job easier and this idea has been followed in the development of the LabVIEW program's GUIs presented here. The new CSCI interface (GUI) is created with buttons and tabs used to send commands to the detectors (RCU) or to do other actions. Different colors are used to differentiate the various components of the GUI and make the GUI attractive. All the buttons show what function they will trigger in the detector (RCU) firmware. Some

[2] Jack Wallen is an award-winning writer for Linux.com and Techrepublic.com.

control buttons are hidden or visualized according to the situation to avoid confusion. The ASIC Configuration GUI is also created with simple buttons, so the user can easily change the configuration parameters and send them to activate the ASICs. The front panel of the readout and configuration program is also made simple by using some buttons to trigger functions, and the real-time plots visualize the science data.

c) Effective Error handling

The error handling is the key factor in this LabVIEW program. Every user makes mistakes. This can be because of programming mistakes, invalid runtime's inputs, unavailability of required resources, etc. The programs without error handling will often crash or invisibly perform incorrectly. Writing good error and information messages is also an art which makes the program easier to use.

It is tried to handle errors properly in the developed program by using error handling VIs from the LabVIEW programming environment and by writing understandable error and information messages. Every single loop in the program is checked for errors. The *Error In* and *Error Out* clusters are used in all the VIs and sub-VIs to flow the error information from beginning to end and handle the errors with *Error Handler* VI in the end.

d) Comprehensive and Intuitive

The system should be easily learnable and understandable. It is tried to make this LabVIEW program comprehensive and intuitive by creating simple, easy, attractive GUI interfaces. The user can recognize easily what to do, when to do it, where to do it and how to do it.

8.1.2. Is this program Robust?

Robustness refers to stability of a program in extreme situations, i.e. a program that performs well under usual conditions as well as unusual conditions is a robust program. All software and programs contain errors, which reduce the robustness. To make a program robust it is very important to deals with errors. The errors can be for example, user errors (invalid input) or internal errors (bugs). The reliable program must ideally detect or prevent all these conditions in safe way without crashing. As mentioned before, a LabVIEW program needs proper error handling for their proper functioning and in this developed readout and configuration LabVIEW program, errors are checked properly and eliminated.

Is this program really robust? This is still a question. The program is tested many times and sometime gives unexpected errors. These errors can be due to malfunctioning of hardware or software. The user-tests are not performed so far on this program but are expected to give feedback on its robustness.

8.2 Outlook - What could be done?

The ASIM project is in its final phase, and the flight module containing space qualified components and boards are under development. The developed LabVIEW program has the capability to run full-fledged test on both the CZT and the BGO detectors. The program is tested on the CZT detector with four DMs, or one readout detector chain but we didn't have all the detector modules available to run fully populated test in case of CZT detector. On the other hand, the final, fully functional firmware is not available for testing the BGO detector during the time frame of this project work.

The Bin Control Module (BCM) is defined in the RCU firmware of both the detectors. The BCM processes the SCDPs by binning them into a two-dimensional table of temporal and spectral bins stored in the memory. But the developed LabVIEW program is not able to access the BCM registers because it required its own readout interface. Also the most important control commands defined in the RCU firmware are inserted onto the control buttons on both CSCI interfaces so far. All the control commands defined in the RCU firmware can be added onto the buttons on CSCI in the future, if needed.

On both CSCI interfaces, there is an additional tab named '*Click in End'*, which must be pressed each time before sending selected data on different tabs to the RCU. The '*Click in End'* tab button includes the data from last selected tab in the program. It is desirable to avoid this extra compulsory click.

Appendix-A

1 Different control buttons on the CZT-CSCI interface Tabs

1. The **'Start Config'** button configures the XA-ASICs in one DM-readout chain by sending a serial bit stream and clock from an embedded RAM. When users click on this button, it will send a predefined number of configuration bits to the XA-ASICs starting from address 0.
2. The **'Reset Flags'** button on four tabs send command to reset the status flag bits in the SR0 (status register 0) register in the RCU firmware.
3. The **'Disable DM'** button asserts command to disable the read-out chain associated with the modules and stops the ADC clock.
4. The **'False Triggers'** button introduces the command which make a module to ignore the XA-ASIC trigger output but sampling data continuously.
5. The **'Enable baseline correction'** button inserts command to enable the baseline correction. The module then evaluates the ADC offset and subtracts it in real time.
6. The **'Invert ADC data'** button inverts the ADC value.
7. The **'Snapshot'** button asserts command to copy the current ADC, address, and multihit inputs to the status registers.
8. The **'Decimation Factor'** button introduces a data reduction factor for detected photon hits (specially the case of large magnitude of observed data).
9. The **'Master Reset'** button initializes all state machines and registers to their default state.
10. The **'Test Mode'** button enables the test mode which displays the energy and address data with onboard LEDs.

The above mentioned control buttons 1-2, 3-8 and 9-10 trigger functions in the XA_CFG modules, the DM_IF modules and the RCU_master module respectively in the CZT-RCU firmware.

2. Different control buttons on the BGO-CSCI interface Tabs

1. The **'Enable ADC'** button sends command to disable the ADC clock which disables the concerned read-out chain when set to 0.
2. The **'Enable continuous sample mode'** button disables the trigger. When this button is pressed on, the module will fill the FIFO with raw sample data from the ADC.
3. The **'Enable Tail Cancellation'** button enables the tail cancellation which increases the energy resolution at higher event rates.
4. The **'Value Trigger'** button inserts a command to check whether a pulse should generate a trigger or not. It is a lower-level discriminator.
5. The **'Post-Peak Value Trigger'** button inserts command to check the values immediately a pulse is observed. It is also a lower-level discriminator but should be higher than 'Value Trigger' to avoid excessive false-trigger generation.
6. The **'Decimation Factor'** button introduces a data reduction factor for detected photon hits (specially the case of large magnitude of observed data).
7. The **'Enable TMON'** button sends a command to enable the entire module with its temperature compensation functionality.
8. The **'Enable Auto Cycle'** button as the name implies, enables the TMON module to continuously cycle through the possible bit combinations for channel selection ("00", "01", and "10").
9. The **'Temperature Channel Select'** button asserts the bits to set the select temperature if auto cycle mode is disabled.

The above-mentioned control buttons 1-6 and 7-9 trigger functions in the PMT_IF modules and TMON module respectively in the BGO-RCU firmware.

All the buttons used on the both CSCI interfaces and discussed above insert commands to a specific address defined in the RCU firmware according to Appendix-B or references [16] and [17].

Appendix-B

1. The CZT-RCU Firmware

The data about the CZT-RCU firmware is taken form reference [16].

Modules base addresses

Base Address	Name	Description
0x0000	XA CFG 1	XA configuration module 1
0x0400	XA CFG 2	XA configuration module 2
0x0800	XA CFG 3	XA configuration module 3
0x0C00	XA CFG 4	XA configuration module 4
0x1000	DM IF 1	Detector module interface 1
0x1008	DM IF 2	Detector module interface 2
0x1010	DM IF 3	Detector module interface 3
0x1018	DM IF 4	Detector module interface 4
0x1020	RCU MASTER	RCU master
0x1028	BIN CTRL	Bin Controller Module
0x343C	TTGEN	Time Tag Generator

XA_CFG module Registers

Address Offset	General Name	Specific Name	Bitfield	Description	Default Value
0x0000	CR0	STC	Bit 0	Start configuration	0
		SRST	Bit 1	Reset configuration status flags	0
0x0004	SR0	RPASS	Bit 0	Register verification passed	0
		RFAIL	Bit 1	Register verification failed	0

DM_IF module Registers

Address Offset	General Name	Specific Name	Bitfield	Description	Default Value
0x0000	CR0	SNAP	0	Take snapshot of address, energy, and multihit inputs	0
0x0001	CR1	DIS	0	Disable module, stops ADC clock	0
		HIT	1	Generate false triggers	0
		OFFS	2	Enable baseline correction	0
		INV	3	Invert ADC data	1
0x0002	CR2	DECF	7:0	Decimation factor	0x00
0x0004	SR0	ADCD1	7:0	ADC snapshot data (11:4)	0x00
0x0005	SR1	ADCD2	7:4	ADC snapshot data (3:0)	0x00
			3:0	Multihit snapshot data	0x00
0x0006	SR2	ADR1	7:0	Address snapshot data (11:4)	0x00
0x0007	SR3	ADR2	7:4	Address snapshot data (3:0)	0x00

RCU MASTER Module Registers

Address Offset	General Name	Specific Name	Bitfield	Description	Default Value
0x0000	CR0	LSHIFT	0	Start LED shift test sequence	0
		MRST	1	Master reset	0
0x0001	CR1	TEST	0	Test mode enable	0
		SRSEL	1	Source select	0
		CHSEL	3:2	Chain select	“00”
		TTMODE	4	Time tag mode	0
		LDARK	5	Disable all onboard LEDs	1
		ASIC	6	Set ASIC version	0
		SCRUBS	7	Stop EDAC RAM scrubbing	0
0x0002	CR2	TRIG	3:0	Trigger output mask	“0000”

		ETEST	4	Perform energy test sequence	1
		ATEST	5	Perform address test sequence	1
		TTEST	6	Perform time-tag test sequence	0
		MTEST	7	Perform multihit test sequence	0
0x0003	CR3				
0x0004	SR0	ERR	7:0	Number of un-correctable double errors	X”00”
0x0005	SR1	CORR	7:0	Number of correctable single errors	X”00”

BIN CTRL module Registers

Address	Name	Description	Default Value
0x0000 – 0x270F	Bin Memory	Holds bin values. Little endian.	0x00
0x2710	Spectral boundary low	Events with energies lower than this value (exclusive) are discarded. Sets 8 most significant bits of 10 bit value (zero padded 0b00BBBBBBBB).	0x00
0x2711	Spectral boundary 1	Spectral boundary between spectral bin 0 and 1 in units of 4. Sets 8 most significant bits of 10 bit value (zero padded 0bBBBBBBBB00).	0x1A (104)
0x2712	Spectral boundary 2	Equivalent to the above.	0x33 (204)
0x2713	Spectral boundary 3	Equivalent to the above.	0x4D (308)
0x2714	Spectral boundary 4	Equivalent to the above.	0x66 (408)
0x2715	Spectral boundary 5	Equivalent to the above.	0x80 (512)
0x2716	Spectral boundary 6	Equivalent to the above.	0x99 (612)
0x2717	Spectral boundary 7	Equivalent to the above.	0xB3 (716)

0x2718	Spectral boundary 8	Equivalent to the above.	0xCC (816)
0x2719	Spectral boundary 9	Equivalent to the above.	0xE6 (920)
0x271A	Spectral boundary High	Events with energies higher than this value (exclusive) are discarded. Sets 8 most significant bits of 10 bit value (one padded 0bBBBBBBBB11).	0xFF (1023)
0x271B	Temporal Bin Size	The value of the temporal bin size in units of 80 µs. Sets 8 most significant bits of 11 bit value (zero padded 0bBBBBBBBB000).	0x19 (2 ms)

2. The BGO-RCU Firmware

The data about the BGO-RCU firmware is taken form reference [17].

Modules base addresses

Base Address	Name	Description
0x0000	PMT IF 0	Photo Multiplier Tube Interface 0
0x0008	PMT IF 1	Photo Multiplier Tube Interface 1
0x0010	PMT IF 2	Photo Multiplier Tube Interface 2
0x0018	TMON	Temperature Monitor
0x0020	BCM	Bin Controller Module
0x1434	RCUMaster	RCU master

PMT_IF Registers

Address Offset	General Name	Specific Name	Bit field	Description	Default Value
0x0001	CR1	VALTRG	7:0	Value trigger	0x0F
0x0002	CR2	PPTRG	7:0	Post-peak value trigger	0x30
0x0003	CR3		Bit 7	ADE: Enable ADC	1
			Bit 6	SMP: Enable continuous sample mode	0

			Bit 5	TCE: Enable Tail cancellation	1

TMON Registers

Address Offset	General Name	Specific Name	Bit field	Description	Default Value
0x0000	CR0	TME	bit 7	TMON Enable	1
		MUXE	bit 6	MUX08 Enable	1
		MUXS	bit 1:0	MUX Select	11
0x0001	CR1	TALM0	7:0	Temperature Alarm value for TVAL0	0x80
0x0002	CR2	TALM1	7:0	Temperature Alarm value for TVAL1	0x80
0x0003	CR3	TALM2	7:0	Temperature Alarm value for TVAL2	0x80
0x0004	SR0	TAL0	bit 3	Temperature 2 above limit	
		TAL1	bit 2	Temperature 1 above limit	
		TAL2	bit 1	Temperature 0 above limit	
0x0005	SR1	TVAL0	7:0	Sampled temperature value 0	NA
0x0006	SR2	TVAL1	7:0	Sampled temperature value 1	NA
0x0007	SR3	TVAL2	7:0	Sampled temperature value 2	NA

BCM Registers

Address	Name	Description	Default Value
0x142C	CRO		
0x142D	CR1		
0x142E	CR2		
0x142F	CR3		

Address	Name	Description	Default Value
0x1430	SR0	HITCNT(7:0)	X”00”
0x1431	SR1	HITCNT(15:8)	X”00”
0x1432	SR2	HITCNT(23:16)	X”00”
0x1433	SR3		
0x0020 – 0x141F	Bin Memory	Holds bin values. Little endian.	0x00
0x1420	Spectral boundary low	Events with energies lower than this value (exclusive) are discarded. Sets 8 most significant bits of 12 bit value (zero padded 0b0000BBBBBBBB).	0x00
0x1421	Spectral boundary 1	Spectral boundary between spectral bin 0 and 1 in steps of 16. Sets 8 most significant bits of 12 bit value (zero padded 0bBBBBBBBB0000).	0x1A (104)
0x1422	Spectral boundary 2	Equivalent to the above.	0x33 (204)
0x1423	Spectral boundary 3	Equivalent to the above.	0x4D (308)
0x1424	Spectral boundary 4	Equivalent to the above.	0x66 (408)
0x1425	Spectral boundary 5	Equivalent to the above.	0x80 (512)
0x1426	Spectral boundary 6	Equivalent to the above.	0x99 (612)
0x1427	Spectral boundary 7	Equivalent to the above.	0xB3 (716)
0x1428	Spectral boundary 8	Equivalent to the above.	0xCC (816)
0x1429	Spectral boundary 9	Equivalent to the above.	0xE6 (920)
0x142A	Spectral boundary High	Events with energies higher than this value (exclusive) are discarded. Sets 8 most significant bits of 12 bit value (one padded 0bBBBBBBBB1111).	0xFF (1023)
0x142B	Temporal Bin Size	The value of the temporal bin size in units of 80 µs. Sets 8 most significant bits of 11 bit value (zero padded 0bBBBBBBBB000).	0x32 (4 ms)

RCU-Master Registers

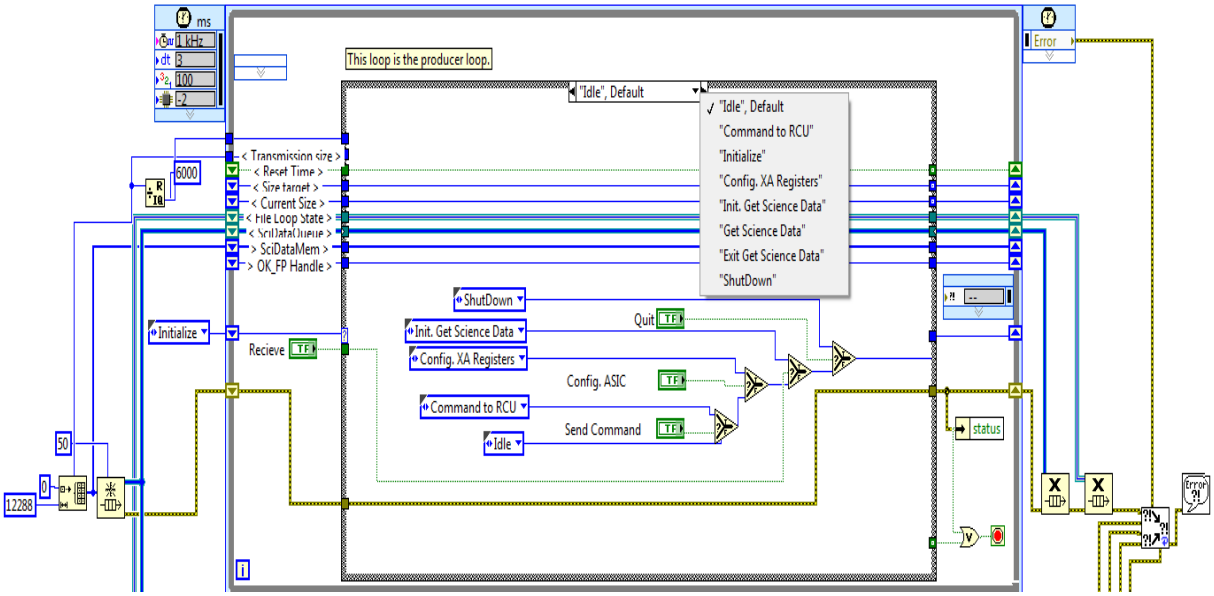
Address Offset	General Name	Specific Name	Bitfield	Description	Default Value
0x0000	CR0	MRST	1	Master reset	0
		LSHIFT	2	Start LED shift test sequence	
0x0001	CR1	TTMODE	4	Time Tag mode	0
0x0002	CR2	ETEST	4	Perform energy test sequence	1
		ATEST	5	Perform address test sequence	1
		TTEST	6	Perform time-tag test sequence	0
		FTEST	7	Perform flagbit test sequence	0
0x0003	SR1	TTSYNC1	3:0	Timetag bits 19:16 at time of SYNC	0
0x0004	SR2	TTSYNC2	7:0	Timetag bits 15:8 at time of SYNC	0
0x0005	SR3	TTSYNC3	7:0	Timetag bits 7:0 at time of SYNC	0

Appendix-C

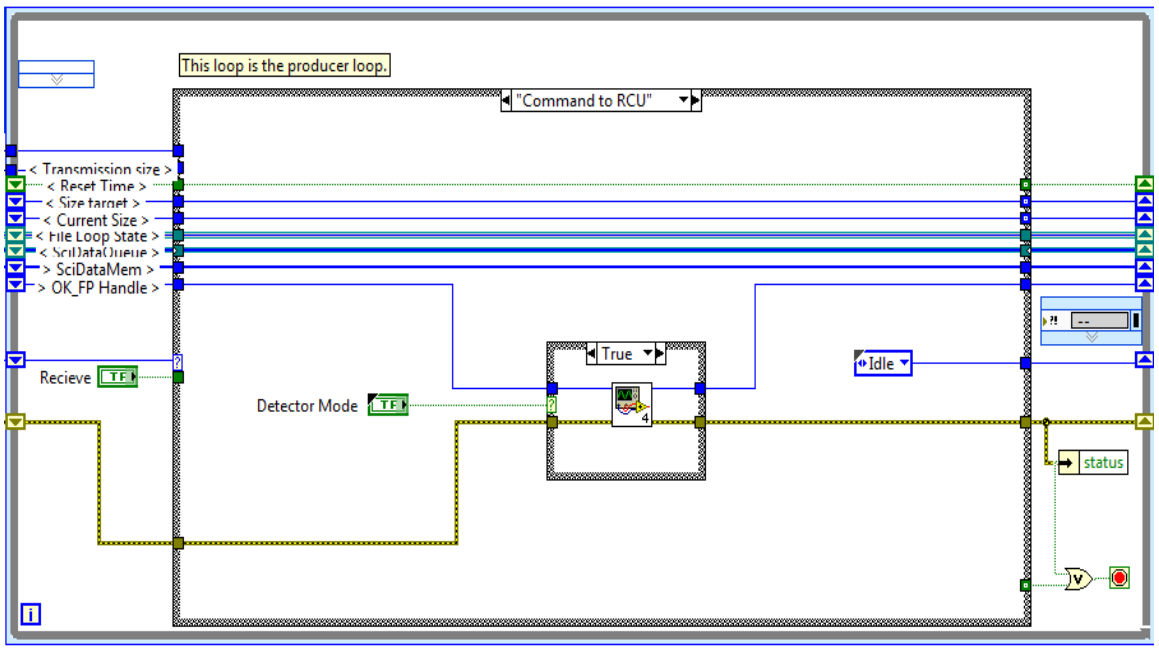
LabVIEW program code for the observation and configuration of MXGS instrument.

1. LabVIEW program code for the producer loop.

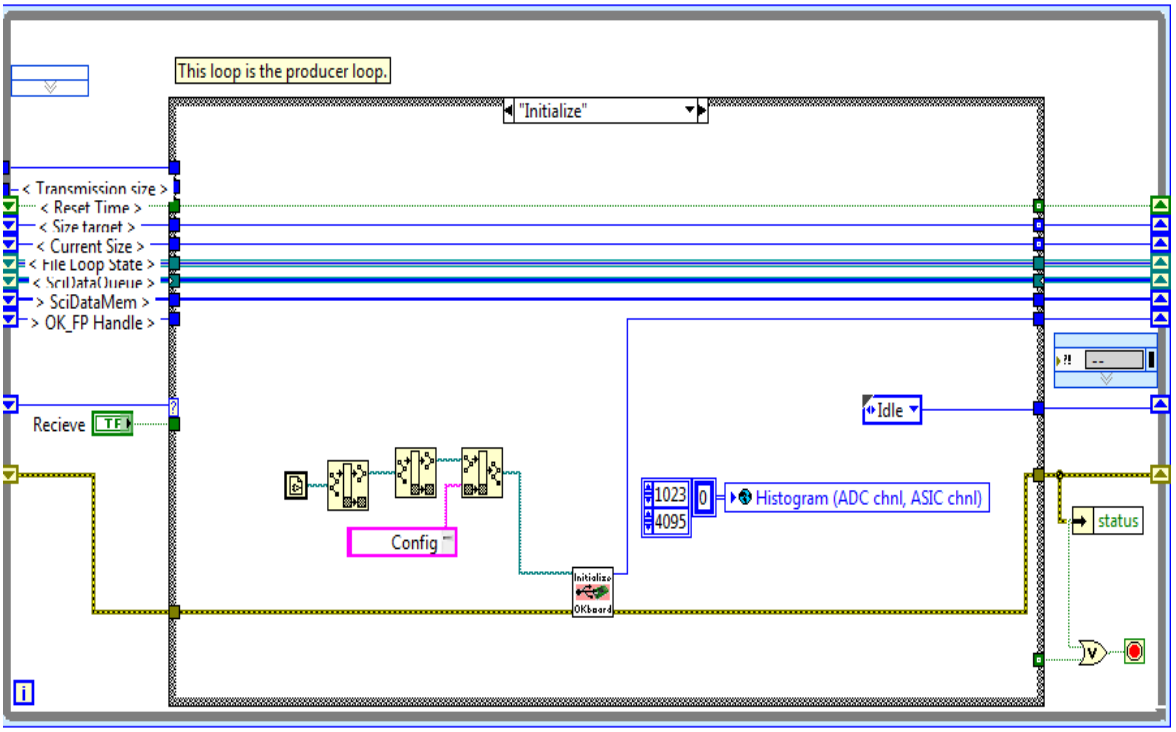
1. ‘Idle (Default)’ state in the producer loop.



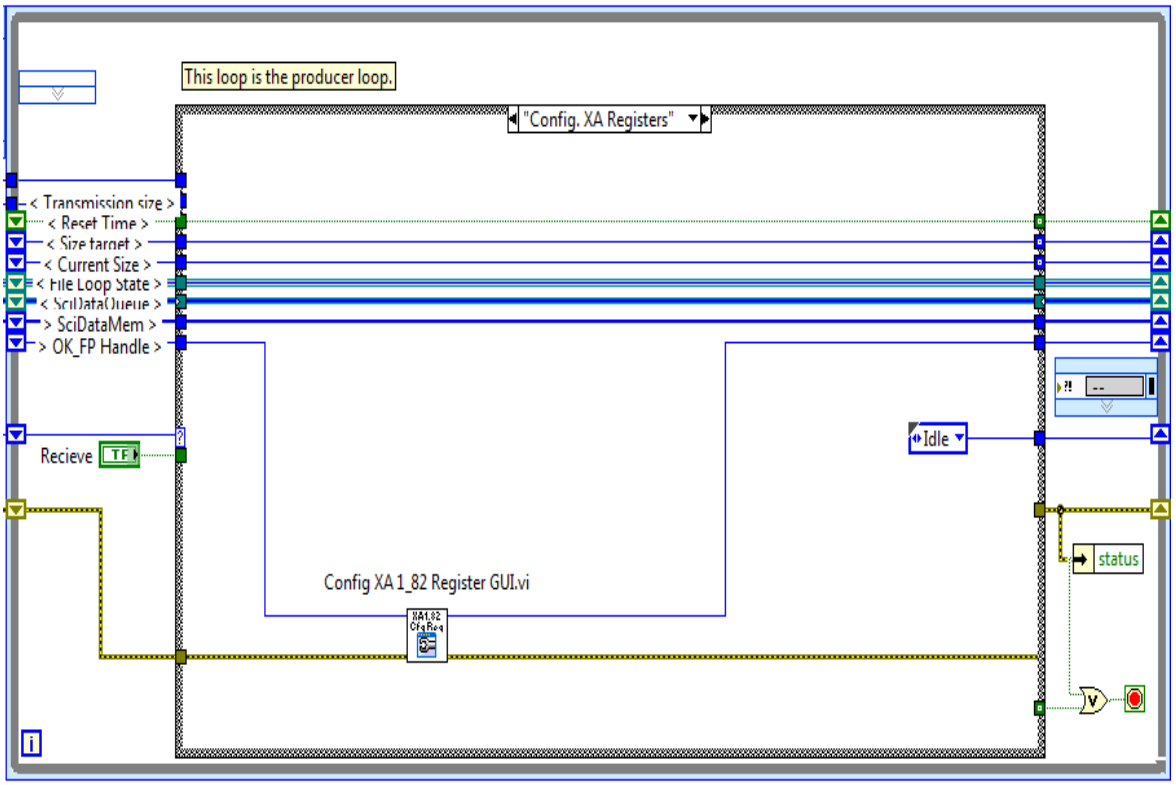
2. ‘Command to RCU’ state in the producer loop.



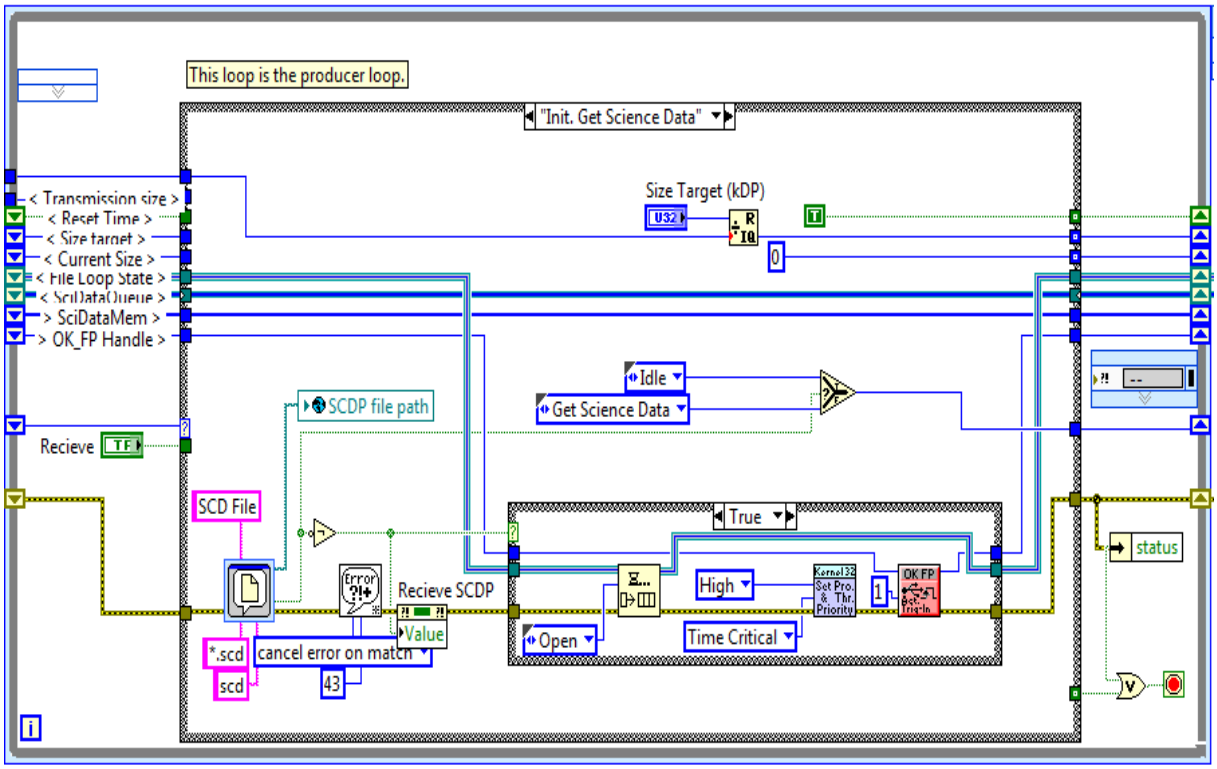
3. ‘Initialize’ state in the producer loop.



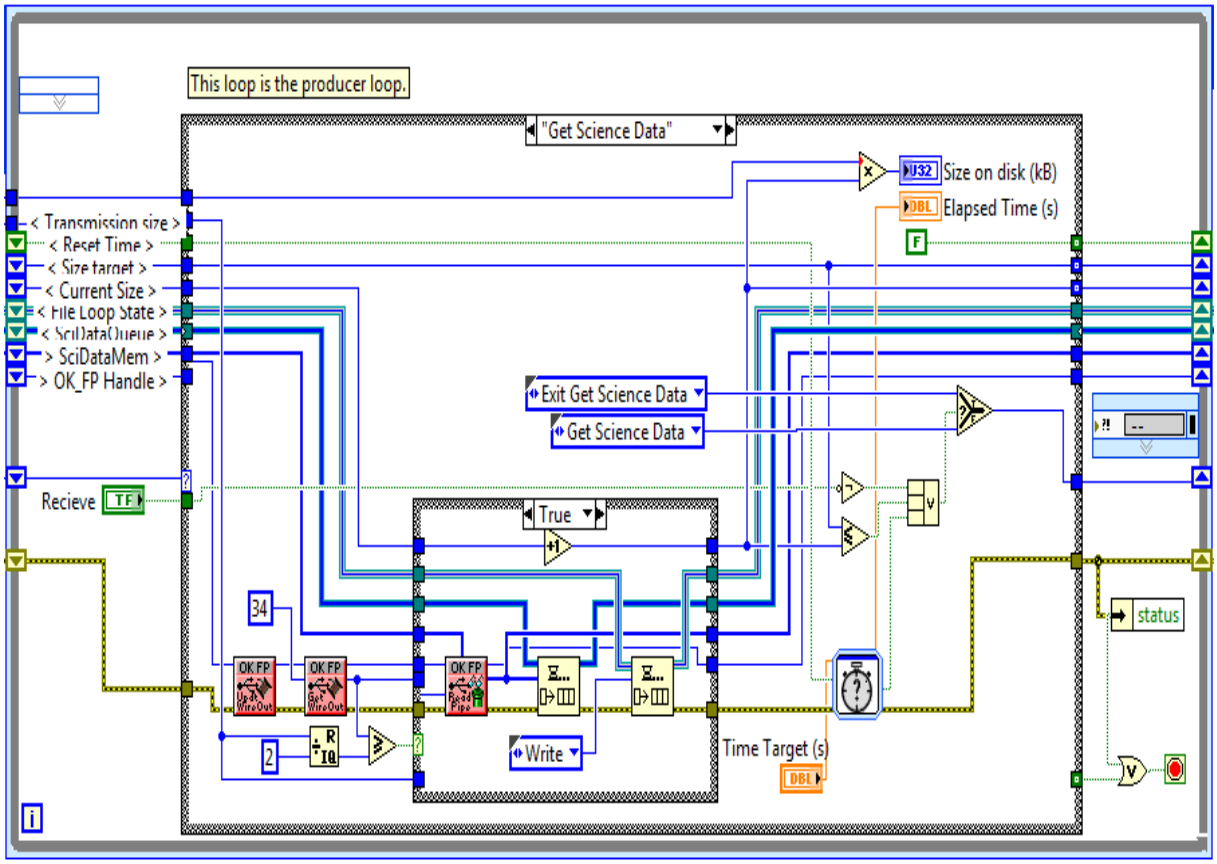
4. ‘Config. XA Registers’ state in the producer loop.



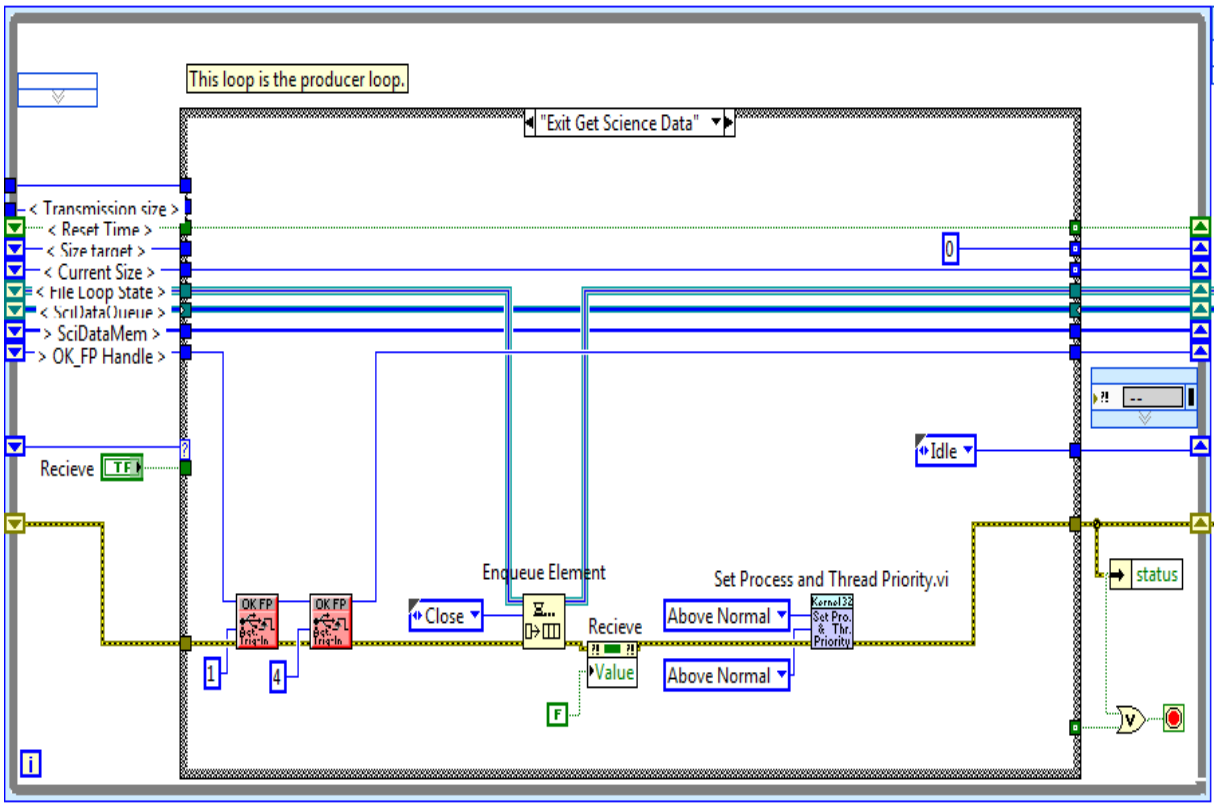
5. ‘Int. Get Science Data’ state in the producer loop.



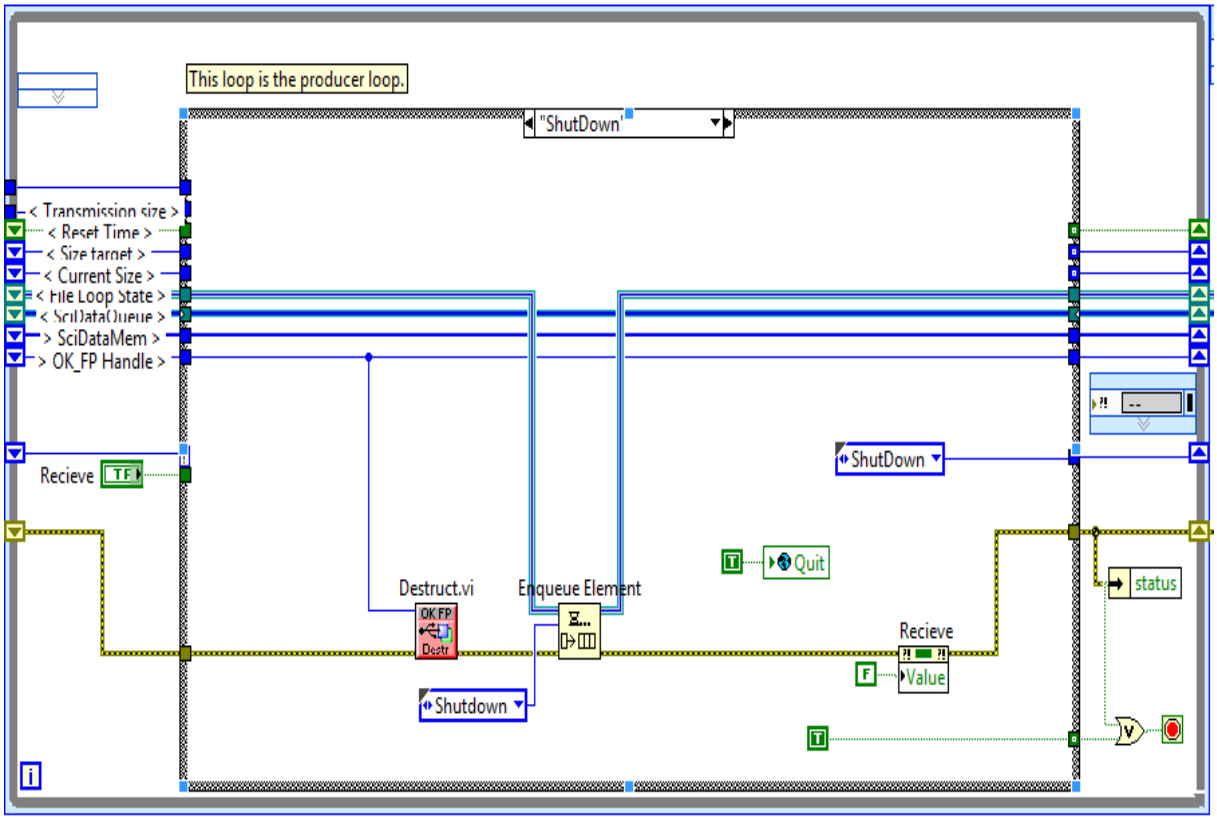
6. ‘Get Science Data’ state in the producer loop.



7. ‘Exit Get Science Data’ state in the producer loop.

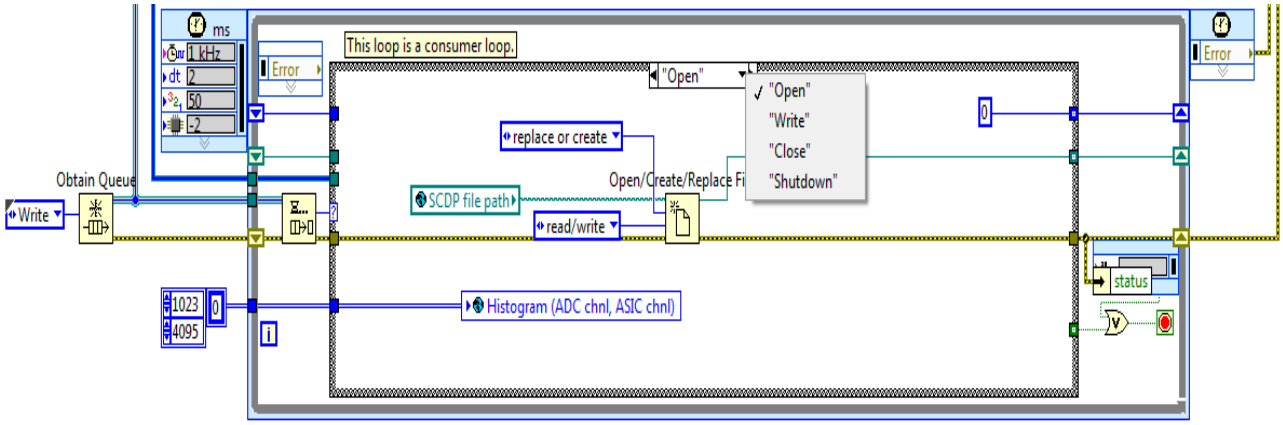


8. ‘Shutdown’ state in the producer loop.

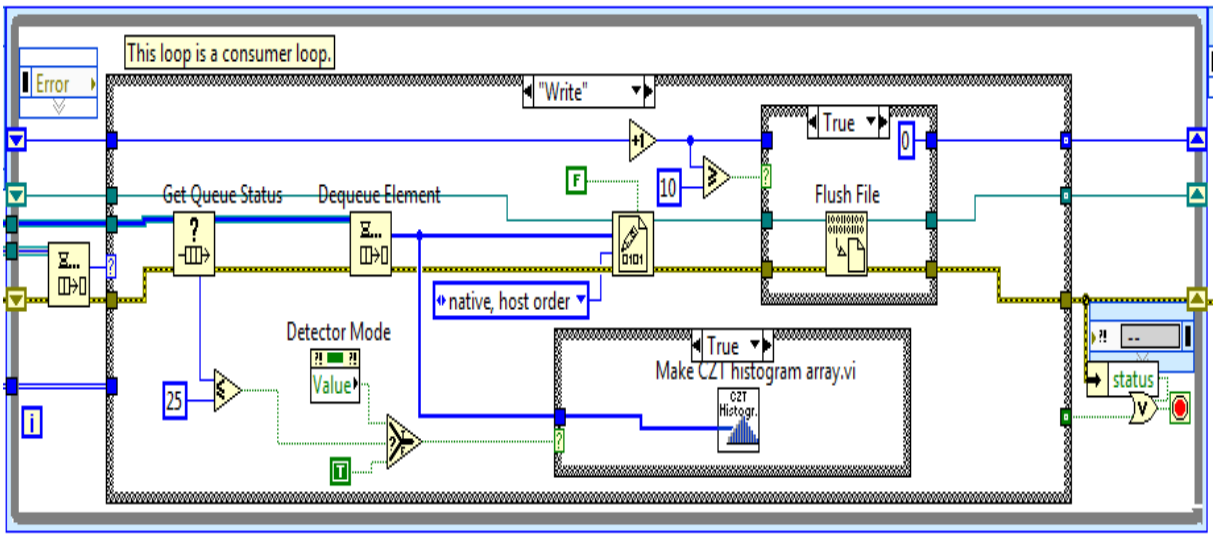


2. LabVIEW program code for the consumer loop.

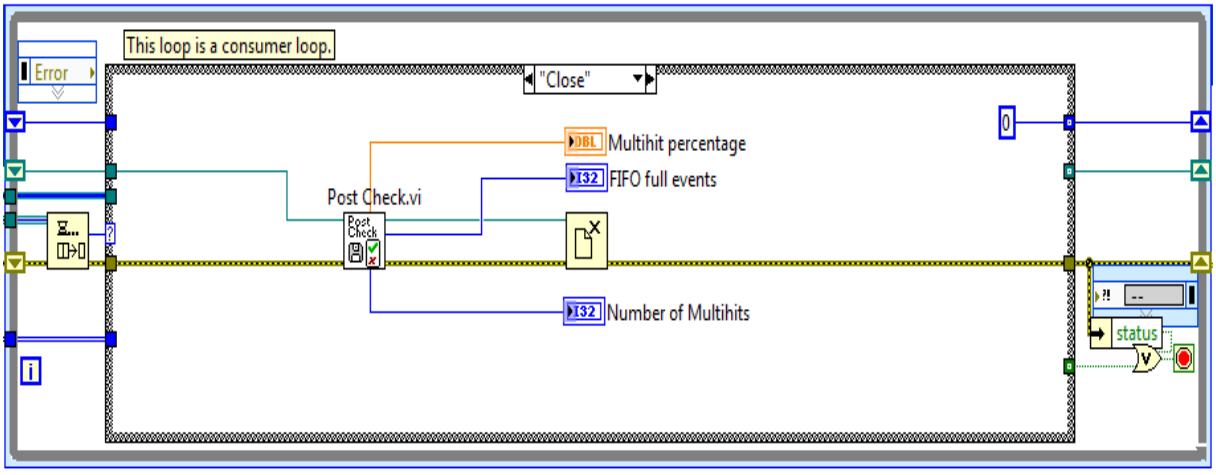
1. ‘Open’ state in the consumer loop.



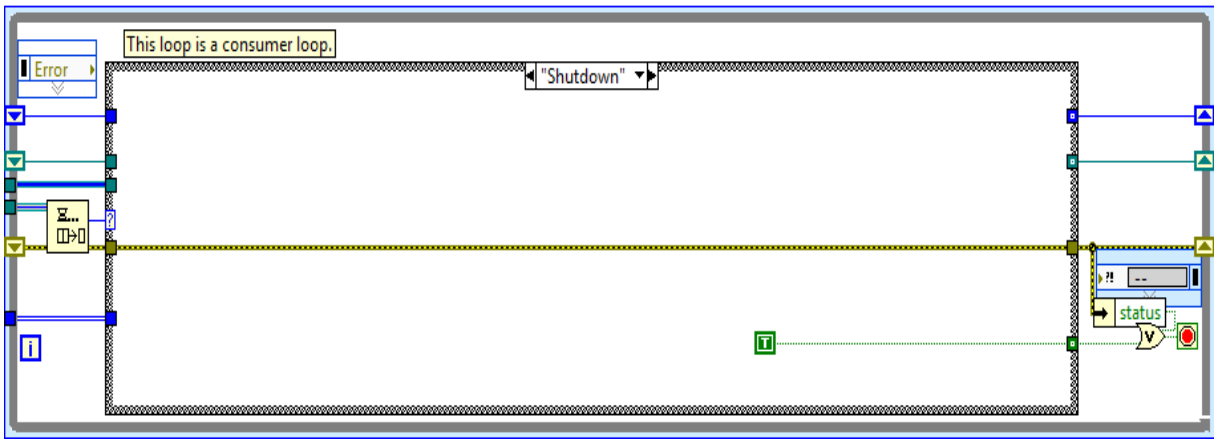
2. ‘Write’ state in the consumer loop



3. ‘Close’ state in the consumer loop

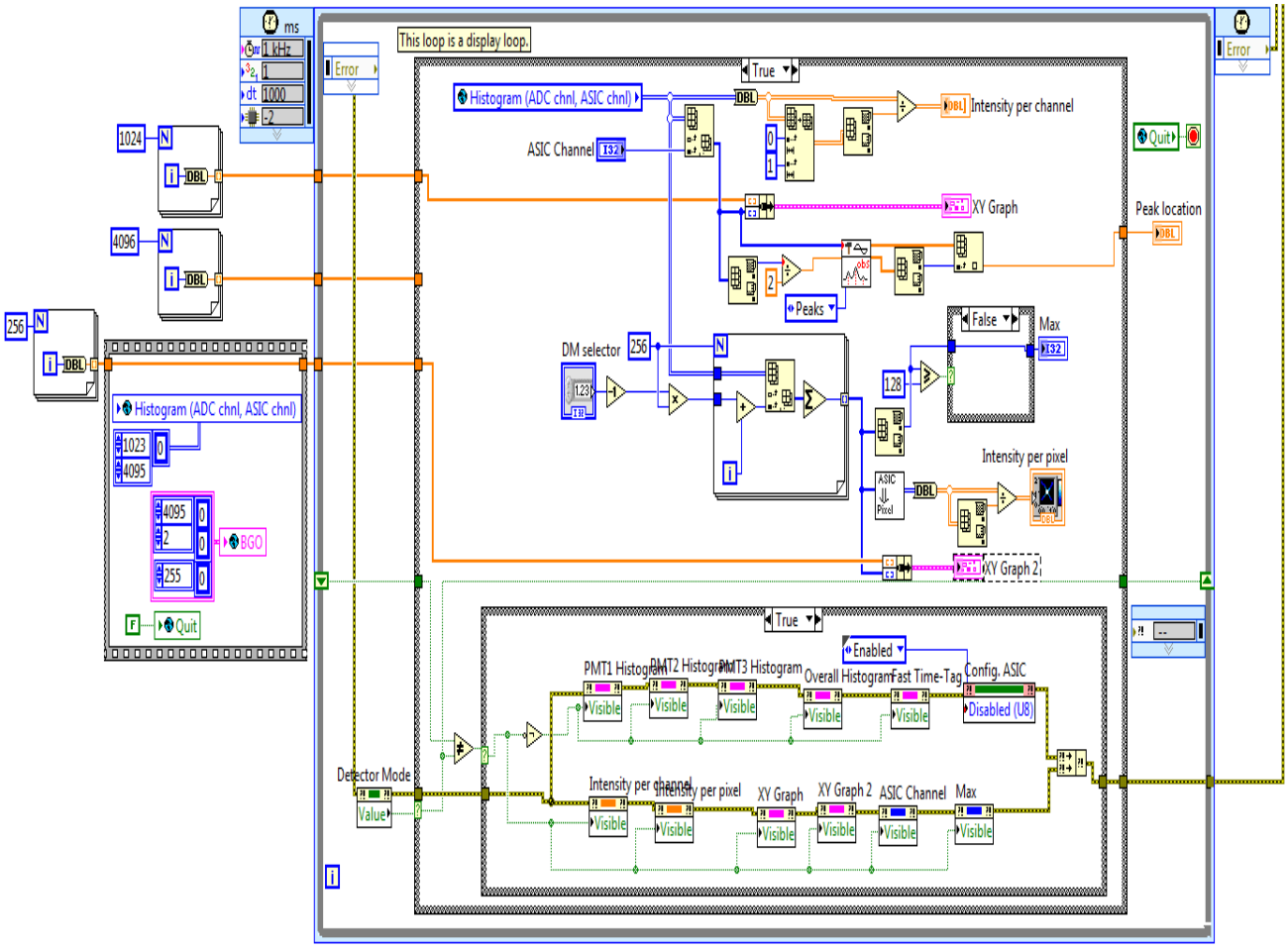


4. ‘Shutdown’ state in the consumer loop.

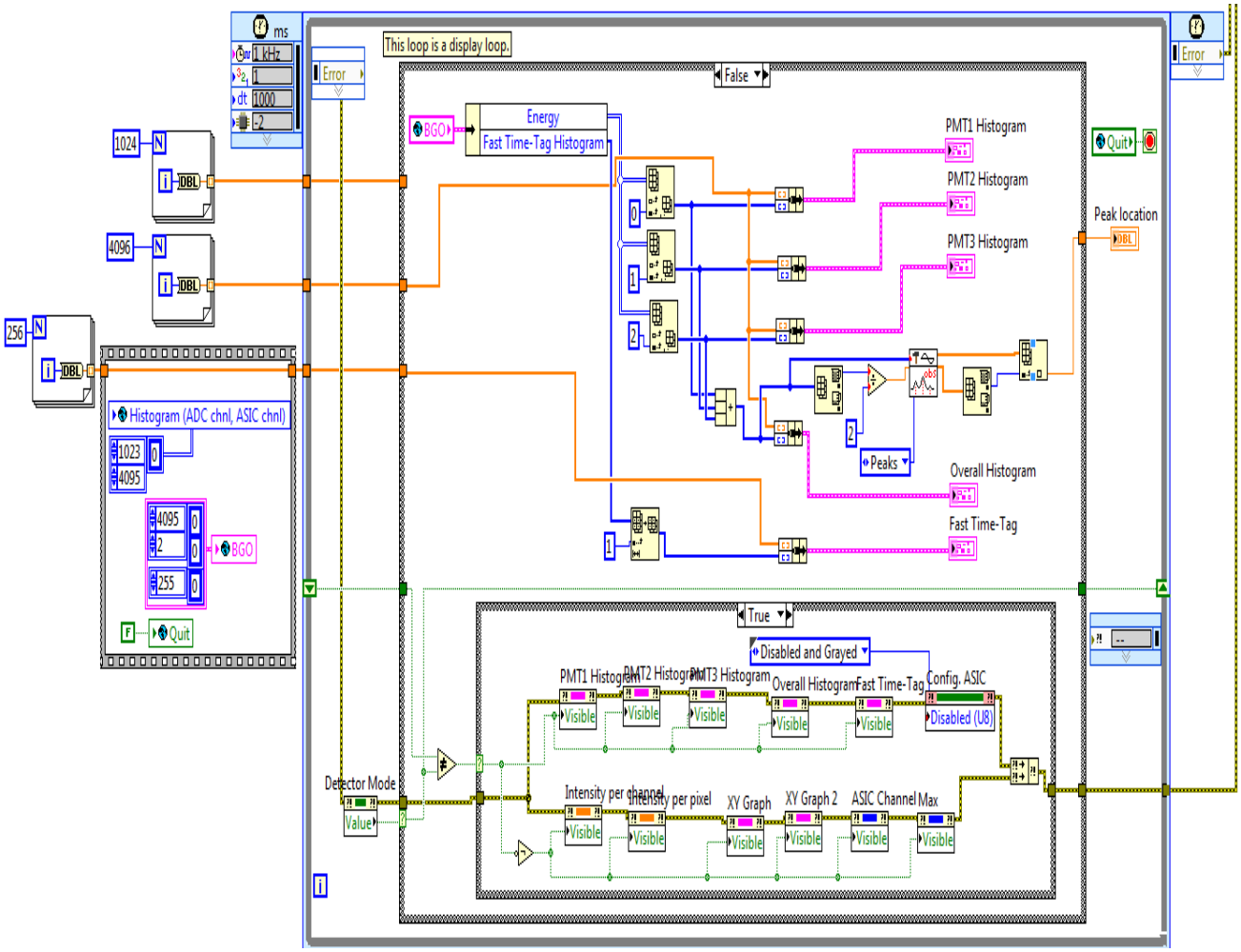


3. LabVIEW program code for the display loop.

1. In case of the CZT detector mode.



2. In case of the BGO detector mode.



Appendix-D

Manual for Observation and Configuration LabVIEW program

1. Different components on the main front panel interface

The different components on the front panel interface as marked in figure C-1 are described below as:

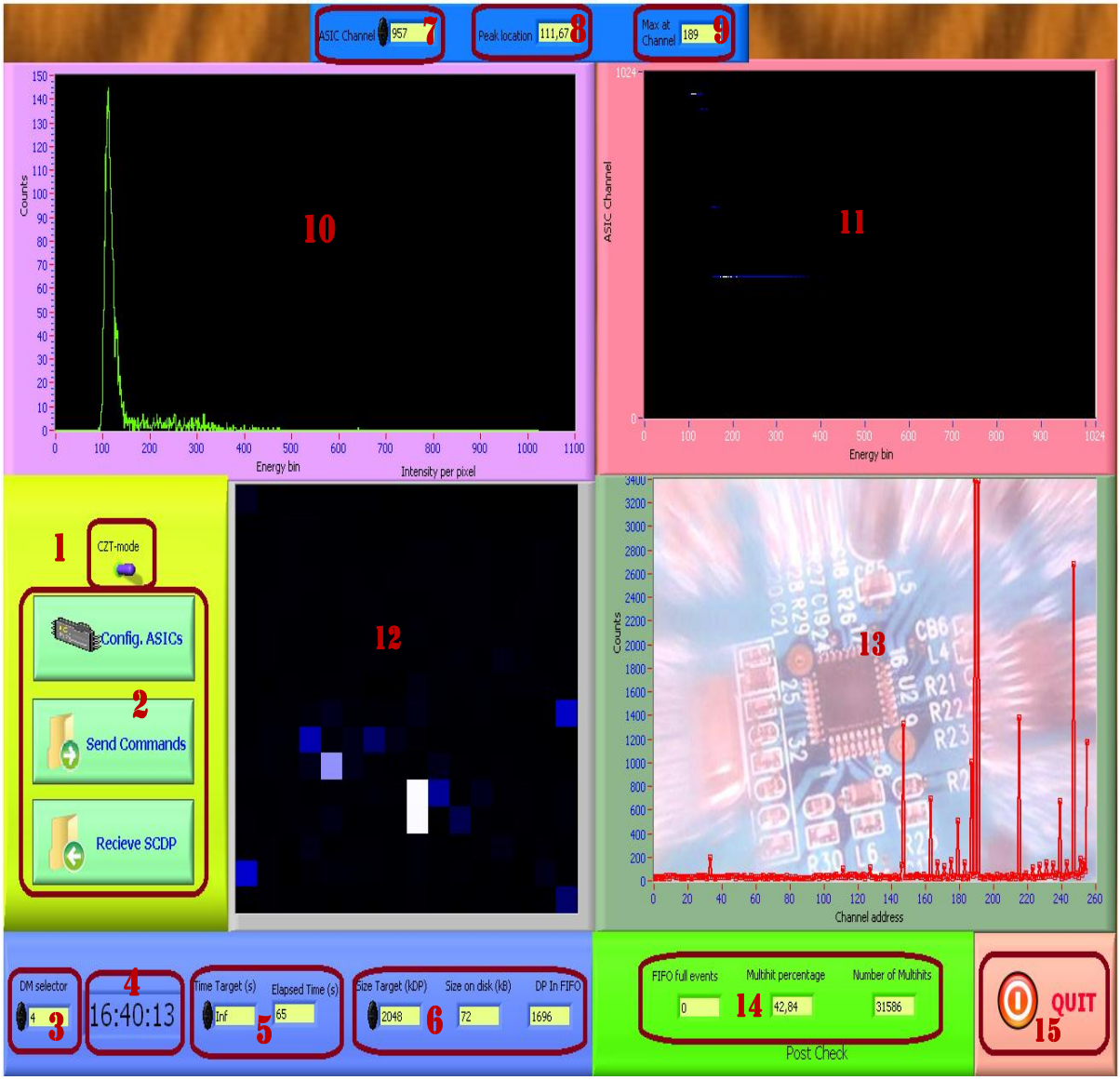


Figure D-1: The different components on the front panel interface of the main program.

1. *Detector mode switch* is used to change the detector mode to the CZT or the BGO depends upon which detector is connected for data acquisition.

2. *Control Panel* consists of three buttons for ASICs configuration, send commands to the RCU and to start acquisition of science data packets respectively.

3. *DM Selector control* is used to select any desirable detector module to display the detected events in two bottom plots.

4. *Time indicator* shows the current time when the VI is in running mode.

5. *Data acquisition control (Time Target)* is used to stop the data acquisition if the user set any time target. The default value is infinity.

6. *Data acquisition control (Size Target)* is used to stop the data acquisition if the user set any size target. The default value is 2048 SCDPs. Also users can see the current data count on *DP in FIFO* indicator.

7. *ASIC Channel selector control* is used to select any desirable ASIC channel/pixel to display the event data for that particular channel in the top-left plot.

8. *Peak Location indicator* displays the highest energy value for the selected ASIC channel.

9. *Max at channel* indicator shows a channel address value in the selected DM having the maximum number of counts.

10. *Single Channel plot* is the plot between energy and counts for the selected ASIC channel.

11. *All over plot* is the plot between energy, address and counts for all channels.

12. *DM Pixel distribution plot* is plotted between pixel position and counts for any selected DM. Each square in the plot represent a pixel in its physical position in the ASIC. Also lighter the color of the pixel means more photon-hits detected on that pixel. Plotted data depends upon *DM Selector* control.

13. *DM Count Distribution plot* is plot between total counts and channel address for a selected DM. Plot can control with *DM Selector* control.

14. *Data acquisition meta information* shows the information about the total number of multihits detected by detector, multihit percentage and FIFO full events during the data acquisition process.

15. *QUIT* button aborts the running mode of the main program.

2. Installation

1. The program made for the observation and configuration of the MXGS instrument is developed in LabVIEW programming environment in LabVIEW 8.6 and LabVIEW 2010 version. Make sure LabVIEW software is installed on your computer.
2. In order to operate the XEM3001v2 card which is used as DPU emulator for testing in the lab, the *okFrontPanel*-drivers must be installed on the computer. This can be done by extracting the content of the *okFrontPanel.zip* into the *user.lib* folder in the LabVIEW installation software folder. Also the file named *okFrontPanel-errors.txt* must be placed in the *resources\errors* folder in the LabVIEW installation software folder. These files can be found in the repository under *D:\ASIM\ASIM-arun\LabVIEW 2010\FrontPanel for LabVIEW.*

3. Usage

Preparation

1. Switch on the computer and make sure that LabVIEW and the *okFrontPanel*-drivers are installed properly.
2. Connect the Opel Kelly XEM3001v2 card with the computer through USB port.
3. Connect the XEM with the Detector Assembly Unit.
4. Check if the jumper J1 is connected so that the XEM draws power from the USB port (indicate with green indicator near the USB microcontroller on XEM card).
5. Connect the detector with the power supply units and provide required power-supply.
6. Install a radiation source (e.g.241_{Am}) above the detector.
7. Check that all DMs are illuminated equally with the radiation source.

Run the program

1. Open the LabVIEW project under *D:\ASIM\ASIM-arun\LabVIEW 2010\CSCI-control environment\MXGS DPU Emulator* and then the VI called *MXGS_DPU Emulator 1_82 Top Level VI (arun).VI* in the project explorer inside folder LV Source.

2. Run the main program with Ctrl+R command.
3. Select the desired detector mode by changing toggle switch on the front panel.
4. In case of CZT detector mode, press "*Config. ASIC*" button on the front panel of the main program. A new GUI interface for configuration of ASICs will appear as shown in figure 7.3.
5. Select any stored ASICs configuration setting by pressing "*Read File*" button and then click on *"Configure detector Chain"* button or manipulate the present setting by clicking on the different click buttons and then click on *"Configure detector Chain"* button or just click on *"Configure detector Chain"* button to select the last used configuration setting.
6. Press *"Exit"* button in the end which makes the ASIC configuration GUI disappeared.
7. Now click "*Send Command*" button on the control panel. A new CSCI-GUI interface will pop-ups as shown in figure 7.4. Click on DM_XA_1 tab and then select some buttons according to the requirement as each button shows which command he will insert in the RCU firmware.
8. In case of fully populated DAU, click on other tabs and select command buttons as each tab represent for one DM readout chain.
9. Click on '*Click in end*' tab in the end.
10. Now you have two options to send the selected data to the RCU. Either send directly by click on '*Send direct to RCU*' button or write the selected data in a text file as Configuration Register Data by pressing '*write register's file*' button and then send by click on '*Send New File to RCU*' button.
11. A file prompt will appear to ask to choose a file and file location to write RCU response in a text file as Status Register Data.
12. Now you can compare the RCU response with RCU input data (your selected configuration data) by click on the '*Compare with RCU result'* button.
13. Two file dialogues will pop-ups one after one to ask you to select one configuration register data file and one status register data file. The files data will be compared and result will be shown on the CSCI-GUI.
14. If the RCU input and RCU output are same, verification will be done by green indicator and a congratulation message. In second case a red indicator with a sorry

message and the differences in the file contents with respect to each other will be shown on the CSCI-GUI.

15. Press *'QUIT'* button to hide the CSCI-GUI.
16. Set the '*Time Target'* or '*Size Target'* in case you want to stop receiving science data packets into FIFO during data acquisition or let it to its default values.
17. Press *'Receive'* button on the main front panel. A file dialogue will pop-ups to select the file name to store the SCDPs. The data acquisition will start right after and can be seen as changing count values in '*DP in FIFO'*. Also data will display on the different plots.
18. You can select the detector module by '*DM selector'* to observe the photon events on a particular DM.
19. You can select any pixel/channel in any ASIC with '*ASIC Channel*' control to observe the photon events on a particular pixel/ASIC channel.
20. The program will stop the acquisition of SCDPs if the condition of '*Time Target'* or '*Size Target'* matched with user defined values or can stop by clicking on *'Stop Rec'* button on the front panel.
21. To take other measurement repeat step 7-17.
22. Exit the program by clicking on the '*QUIT*' button at the bottom-right corner.

In case of the BGO detector mode, ignore the steps 4-6 and 18-20.

4. Troubleshooting during usage

1. Failed to initialize error

- ✓ Check if the XEM card is connected with computer.
- ✓ Check if the XEM card's drivers are available in the LabVIEW installation folder under user.lib folder.
- ✓ Restart the LabVIEW program.
- ✓ Maybe restart your computer.

2. Data acquisition is running, but no counts are detected?

- ✓ Check if the XEM card is connected with the detector board.
- ✓ Check if all the power supplies are on.

- ✓ Check if the high-voltage source is on.
- ✓ Send again a non-zero value of the *decimation factor* register in the DM_IF_1 module if you are using only one DM readout chain.
- ✓ Re-configure the ASICs.
- ✓ Switch all the power supplies off and on again.

3. Many thousand counts in few seconds detected!!

- ✓ Check if the cover plate is screwed properly onto the case containing the DAU and check that no external light reaching the detector.
- ✓ Check the ground wire of the XEM card is connected with detector's ground.
- ✓ Re-configure the ASICs.
- ✓ Switch all power supplies off and on again.
- ✓ Re-run the LabVIEW program, configure the ASICs and send non-zero value for decimation factor.

4. Still not working and don't know why?

- ✓ Don't feel bad and disappointed. Check and re-start everything. You probably missed something.

Acronyms

ADC	Analog to Digital Converter
API	Application Programming Interface
ASIC	Application Specific Integrated Circuit
ASIM	Atmosphere-Space Interaction Monitor
BATSE	Burst and Transient Source Experiment
BB	Breadboard
BGO	Bismuth Germanate - $Bi_4Ge_3O_{12}$
CEPA	Columbus External Payload Assembly
CSCI	Command and Status Control Interface
CZT	Cadmium Zinc Telluride – *CdZnTe*
DAB	Detector Assembly Board
DAF	Detector Assembly Frame
DAU	Detector Assembly Unit
DAC	Digital to Analog Converter
DFEE	Detector Front End Electronics
DM	Detector Module
DPU	Data Processing Unit
DTU	Technical university of Denmark
EBB	Elegant Breadboard
EEPROM	Electrical Erasable Programmable Read Only Memory
ESA	European Space Agency
FIFO	First In-First Out
FM	Flight Module
FP	Front-Panel
FPGA	Field Programmable Gate Array
FSM	Finite State Machine
GPIB	General Purpose Interface Bus
GUI	Graphical User Interface
HDL	Hardware Description Language
HTV	Heavy Transfer Vehicle

HVPS	High Voltage Power Source
I/O	Input/output
IP	Internet Protocol
ISS	International Space Station
LabVIEW	Laboratory Virtual Instrument Engineering Workbench
LED	Light Emitting Diode
LVPS	Low Voltage Power Source
MDDP	Memory Dump Data Packet
MMIA	Modular Multi-spectral Imaging Array
MRC	Memory Read Command
MWC	Memory Write Command
MXGS	Modular X-ray and Gamma-ray Sensor
NASA	National Aeronautics and Space Administration
NI	National Instruments
NOx	Nitro Oxides
PC	Personal Computer
PCB	Printed Circuit Board
PLL	Phase Locked Loop
RAM	Random Access Memory
RCU	Readout Control Unit
RHESSI	Reuven Ramty High Energy Solar Spectroscopic Imager
SCDP	Science Data Packet
SRAM	Static Random Access Memory
TCP	Transmission Control Protocol
TGF	Terrestrial Gamma Flashes
TLE	Transient Luminous Events
UiB	Universitetet i Bergen
UV	University of Valencia
USB	Universal Serial Bus
VI	Virtual Instrument
VHDL	Very high speed integrated circuit Hardware Description Language
XEM	Opal Kelly XEM3001v2

Bibliography

[01] Walter A. Lyons. ***THE METEOROLOGY OF TRANSIENT LUMINOUS EVENTS-AN INTRODUCTION AND OVERVIEW*** (Sprites, Elves and Intense Lightning Discharges), NATO Science Series, 2006, Volume 225, pages 19-56.

[02] C.P. Barrington-Leigh, ***Terrestrial Gamma-ray Flashes after CGRO: prospects for HESSI,*** Retrieved April 2, 2005, University of California Berkeley. Space Physics Research group.

[03] ***TLE (Transient Luminous Events) – Mysterious Phenomena in the upper Atmosphere***, http://ulisse.medes.fr/en/content/tle-transient-luminous-events-mysterious-phenomena-upper-atmosphere. Last checked on 25.08.2011.

[04] H.T Su, R. R. Hsu, A. B. Chen and Team. ***Gigantic jets between a thundercloud and the ionosphere,*** http://www.nature.com/nature/journal/v423/n6943/full/nature01759.html. Last checked on 18.08.2011.

[05] CHOU. J, Tsai. L, Kuo. C and team. ***Blue And Gigantic Jets From Taiwan 2007 TLE Campaign***, American Geophysical Union, Fall Meeting 2007, abstract #AE42A-02, December 2007.

[06] ***Upper-atmospheric lightning***, http://en.wikipedia.org/wiki/Upper-atmospheric_lightning. Last checked on 18.08.2011.

[07] ***Terrestrial gamma-ray flashes***, http://en.wikipedia.org/wiki/Terrestrial_gamma-ray_flash. Last checked on 18.08.2011.

[08] ***The Reuven Ramaty High Energy Solar Spectroscopic Imager (RHESSI),*** Source: NASA, http://hesperia.gsfc.nasa.gov/hessi/index.html. Last checked on 19.08.2011.

[09] TERMA space. ***ASIM Instruments Development***, http://www.terma.com/multimedia/ASIM_flyer_v3_Sep_2010.pdf. Last checked on 18.08.2011.

[10] ***International Space Station,*** http://en.wikipedia.org/wiki/International_Space_Station. Last checked on 18.08.2011.

[11] ***Nadir***, http://en.wikipedia.org/wiki/Nadir. Last checked on 24.08.2011.

[12] The DNSC ASIM team. ***MMIA Mechanical Analysis & Design Report,*** Danish national space center, Doc. no. ASIM-DNSC-MMIA-RP-002. Dated 04.29.2009.

[13] ESA. ***ISS Utilization: ASIM (Atmosphere-Space Interactions Monitor)***, http://events.eoportal.org/get_announce.php?an_id=15103. Last checked on 18.08.2011.

[14] UiB ASIM Team. ***MXGS Design Description Report***, University of Bergen, Department of Physics and Technology, April 2009.

[15] Technical University of Denmark. ***ASIM: Climate and giant lighting discharges to be studied from the international space station***. http://www.space.dtu.dk/English/Research/Projects/ASIM.aspx. Last checked on 18.08.2011.

[16] UiB ASIM Team. ***MXGS CZT DAU Design Report***. University of Bergen, Department of Physics and Technology, July 2011.

[17] UiB ASIM Team. ***MXGS BGO DAU Design Report***, University of Bergen, Department of Physics and Technology, 2011.

[18] UiB ASIM Team. ***CZT-EGSR User Manual***, Technical Report. University of Bergen, Department of Physics and Technology, December 2010.

[19] UiB ASIM Team, ***Detector Array and Front End Electronics Design,*** Technical Report. University of Bergen, Department of Physics and Technology, May 2009.

[20] UiB ASIM Team. ***MXGS DFEE Requirements specifications,*** Technical Report. University of Bergen, Department of Physics and Technology, August 2009.

[21] UiB ASIM Team. ***MXGS DPU Hardware Requirements Specifications,*** Technical Report. University of Bergen, Department of Physics and Technology, Oct 2007.

[22] SPG Research. ***The Modular X-ray and Gamma-ray sensor (MXGS) and University of Bergen***, http://web.ift.uib.no/Romfysikk/RESEARCH/PROJECTS/ASIM/. Last checked on 18.08.2011.

[23] European Space Agency (ESA). ***ASIM-ESA Portal***, http://www.esa.int/SPECIALS/HSF_Research/SEMTTK0YDUF_0.html. Last checked on 18.08.2011.

[24] ***Atmosphere-Space Interaction Monitor,*** http://en.wikipedia.org/wiki/Atmosphere-Space_Interaction_Monitor. Last checked on 18.08.2011.

[25] ElectricStorms. ***ASIM Mission***, http://www.electricstorms.net/wiki/ASIM_Mission. Last checked on 18.08.2011.

[26] Gamma Medica-Ideas. ***XA 1.82 Documentations***, V1R0, 2007.

[27] Gamma Medica-Ideas. ***XA 1.82 Datasheet,*** V1R0, 2007.

[28] Opal Kelly. ***Front Panel***, http://www.opalkelly.com/library/FrontPanel-UM.pdf. Last checked on 18.08.2011.

[29] Opal Kelly. ***XEM3001v2 User's Manual,*** http://www.opalkelly.com/library/XEM3001v2-UM.pdf. Last checked on 18.08.2011.

[30] C. Budtz-Jørgensen. I Kuvvetli, Y. Skogseide, K. Ullaland, N. Ostgaard, ***Characterization of CZT Detectors for the ASIM Mission,*** Source: IEEE TRANSATIONS ON NUCLEAR SCIENCE, Volume: 56 (4), Pages: 1842-1847, Aug 2009.

[31] Anja Kohfeldt. ***Characterization and Verification of the MXGS DFEE Detector Array,*** Master's thesis, University of Bergen, Department of Physics and Technology, 2010.

[32] Jone Tveiten. ***Development of an ASIM MXGS DPU Interface Emulator***, Master's thesis, University of Bergen, Department of Physics and Technology, 2008.

[33] National Instruments. ***Introduction to NI LabView,*** http://www.ni.com/gettingstarted/labviewbasics/. Last checked on 18.08.2011.

[34] Robert H. Bishop. ***Learning with LabView***, Prentice Hall 2008.

[35] John Essick. ***Hands on Introduction to Labview for Scientists and Engineers***, Oxford University press 2009.

[36] Jon Conway and Steve Watts. ***A Software Engineering Approach to LabView***, Prentice Hall. 2003.

[37] DTU Space-The Technical University of Denmark. ***ASIM: Climate and giant lightning discharges to be studied from International Space Station.*** http://www.space.dtu.dk/English/Research/Projects/ASIM.aspx. Last checked on 09.09.2011.

[38] Steve Oualline. ***C Elements of Style***, Prentice Hall.M & T Books. Nov 1992.

[39] Galitz, Wilbert.O. ***The Essential Guide to User Interface Design-An Introduction to GUI Design Principles and Techniques,*** John Wiley & Sons Publications, Oct 2002, 2nd Edition.